CharToon 2.0 Manual

H. Noot, Zs.M. Ruttkay

Information Systems (INS)

INS-R0004 January 31, 2000

Report INS-R0004
ISSN 1386-3681

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

CharToon 2.0 Manual

Han Noot, Zsófia Ruttkay

CWI

P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

Email: {han.noot, zsofia.ruttkay}@cwi.nl

ABSTRACT

CharToon is a modular system to design and animate $2^1/_2$D faces and other graphical objects. This manual describes the features of each module in version 2.0, with practical tips and sections for sophisticated features marked in the margin. This arrangement allows the novice as well as the advanced user to get introduced to using CharToon. A special extension to CharToon, the EmotionDisc is also described, which allows easy visualization of the entire emotional repertoire of a face made by CharToon. In the Appendices, samples of ascii data files, sample heads and animations and color figures are given.

1991 ACM Computing Classification System: D.2.2, H.5.1, H.5.2, I.3.8, J.5.

Keywords and Phrases: animation, graphical user interface.

Table of Contents

1. Overview

1.1. The components and data files of CharToon

CharToon is an interactive tool to be used to animate $2^1/_2$D cartoon faces and eventually other objects. CharToon consists of three modules (see Figure 1):

- ***Face Editor*** is to be used to design faces with components which can be animated in terms of change of shape, location, size and visibility.
- ***Animation Editor*** is to be used to produce animations, either from scratch or by post-processing existing animations or data produced by facial tracking.
- ***Face Player*** is there to show the animated drawing.

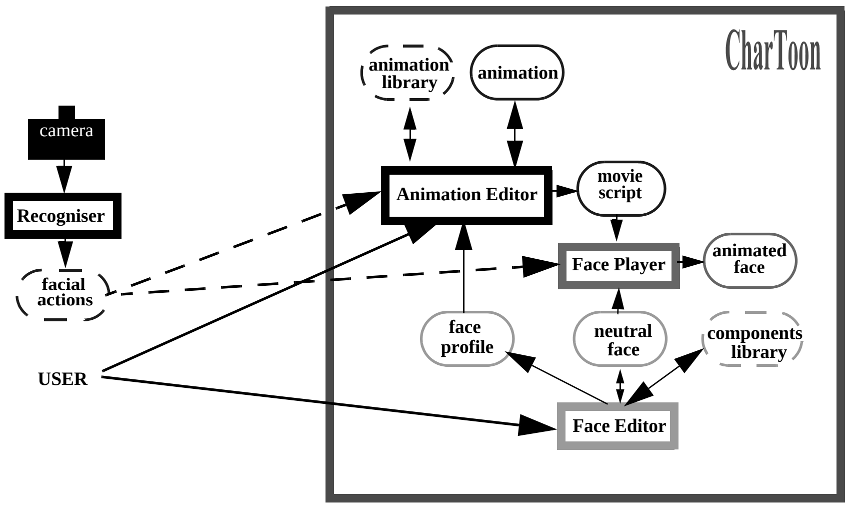


Figure 1. Modules and essential data files of CharToon.

These basic modules provide information for each other by producing ascii files (see Figure 1). The different essential data files reside in specific directories and are identified by specific extensions:

Faces

'.dat' ascii files defining (neutral) faces or other drawings,

output by/input for Face Editor and input for FacePlayer.

Profiles

'.aprof' ascii files, defining 'face profiles', that is the control parameters for a face,

output by Face Editor, input for Animation Editor.

Animations

'.aanim' ascii files defining animation data,

output by/input for Animation Editor, can be also produced outside of CharToon.

CompleteAnimations

'.all' ascii files defining face profiles and animation data,

output by/input for Animation Editor, can be also produced outside of CharToon.

Movies

‘.amov’ ascii files defining animation movies, that is data for each parameter per frame,

output by Animation Editor, input for Face Player.

‘.smov’ ascii files, defining sound arrangements per frame,

output by Animation Editor, input for Face Player.

Sounds

‘.au’ audio files (Sun audio file, mono, 8kb, u-law encoded), containing audio,

input for Face Player and is referred to by the‘.smov’ sound arrangement files.

‘.sprof’ ascii sound profile files, containing information on sound channels and markers,

input for Animation Editor, generated by text editor outside CharToon.

‘.osound’ Java object files, defining the arrangement to play channels,

output by/input for Animation Editor.

AnimDumps

‘.gif’ files, containing image dumps of individual movie frames,

output by Face Player, input for any software to play/make movie from image sequences.

For each of the above mentioned ascii files there is a sample in Appendix 1 with explanation of the syntax of the content. Also Readme files are provided in the directories in which the files reside by default. More about the function of these files, as well as other files needed for the separate components can be found in the dedicated chapters.

!! Face Editor, Face Player and Animation Editor add file-extensions automatically, when the user has to type in a file name. However, if an existing file is to be chosen (from a list by a file dialog), a complete file name is expected as input.

ExternalImages

‘.gif’ image files produced by some sw (e.g. Photoshop) outside of CharToon, to be used as e.g. background, text balloons, etc.

input for Face Editor and Face Player.

In addition to the above files which are meant to store permanent data, there are auxiliary files to be used as default at start-up, as temporary data communication files or as backup files.

Default files:

Profiles/DefaultProfile.aprof, read in automatically if no profile file is specified for Animation Editor,

Sounds/DefaultSoundProf.asprof, read in automatically if present,

ExternalImages/FASELogo.gif, used as default ExternalImage, must be present.

The first two files can be any syntactically correct ones, and should not be necessarily present.

Temporary files:

Profiles/uniqueidCurrent.aprof is used as the profile of the current face being edited in Face Editor (in integrated mode),

Movies/uniqueidSelectionTest.amov is a special temporary file storing a movie of a selected part of the current animation,

Movies/uniqueidSnapshot.amov is a special temporary file storing a single snapshot.

All the temporary file names start with uniqueid, which is an unique identifier of the session generated by the system, in order to allow multi-user usage of the software. In general, the user has nothing to do with the temporary files, which are removed after the user quits CharToon. However, for special advanced purposes the user may want to copy/process some of these files.

!! When using CharToon in integrated mode, the data generated by FaceEditor is written to temporary files. Hence is it very important that you save your work in FaceEditor to permanent files before quitting CharToon.

BackupFile:

Animations/usernameAnimationBackup.aanim animation backup file, re-generated for each run and frequently overwritten by Animation Editor.

The file name starts with the username generated by the system.

1.2. Configuration of the software

The ChartToon software should be installed in a CharToon ***home directory***, without change of the provided directory structure. All Java classes are to be placed in this home directory, while the provided data files are placed in default subdirectories. By default, data files are put into the above mentioned directories which the user may use as working directories. In particular, there are default directories Faces for '.dat' files and Profiles for '.aprof' files. However, especially if CharToon is used in a multi-user environment, it is very much recommended that each user has his own directories with all the data produced by him. Such directories can be created according to one of the principles below:

1) The user uses an identical name (path) to create subdirectories under Profiles and Faces, and loads profiles and face files with explicitly giving the (identical) path to be used under Profiles and Faces. In this case, the user *must* start up CharToon from the home directory.

2) The user creates an own ***user working directory*** anywhere he wishes, with the subdirectories (with write permission) listed in 1.1. The default files (if any to be used) should be put under the appropriate subdirectory (can be copied/linked from the default data files provided with the sw, but new files do as well). The intention is that all essential input/output data files will be loaded from/written to appropriate subdirectories (Profiles, Faces) of the working directory. In this case too it is allowed to create subdirectories under Faces and Profiles. Note: In this case the user *must* start up CharToon from his user working directory.

!! The first case can be seen as the special case when the user working directory is identical to the home directory. In the rest of the manual, we will refer to ***working directory*** as the directory where CharToon should be started up. This is either the CharToon home directory or a user working directory, depending on which principle is used to store data.

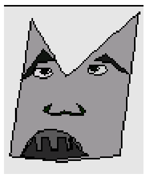
It is allowed and for extensive use it is recommended to make arbitrary new sub-directories for Movies and Animations, e.g. storing animations for each face in separate sub-directories. The user must follow one of the above mentioned protocols, of which 2) is recommended in multi-user environments. However, in both cases it is also possible to load from/save to arbitrary directories containing profile, face and animation files once CharToon was started. However, when CharToon is started in integrated mode, or Face Player is started up from Animation Editor, the face should be found under the same pathname in Faces directory as the profile file under Profiles.

!! In order to keep track of versions and compatible files, one should use systematic naming for files, e.g. animation file names like: face_perf_anim, where the three tags identify the face, the (eventual) performer data the animation is derived from, and the content/version of the animation.

Images and sounds (produced earlier by some other tools) by default are placed in directories ExternalImages and Sounds. The rest of the directories contain auxiliary files for CharToon, these directories should not be touched.

1.3. Hardware/software requirements to run CharToon

In order to run CharToon, the following remarks on hardware and software are relevant:



1) CharToon should be run using jdk 1.1.X. For Sun's and Windows systems, this can be downloaded from `http://java.sun.com/products/jdk/1.1/`. For SGI systems consult `http://www.sgi.com/Products/Evaluation/#jdk`. For Macintosh platforms, download MRJ 2.1.4. from `http://dev-world.apple.com/java/download.html`.

!! The user should *not* attempt to use Java 2, this is not going to work because CharToon as provided has been compiled with the 1.1.X compiler. The reason for that is that at the time of this writing (surprisingly) the drawing speed of 1.1.X. graphics is about 10 times faster than that of Java 2.

2) In order to run all of CharToon the environment variable CLASSPATH should be set to the concatenation (with system-specific separators) of 'current working directory' and 'full pathname of CharToon home directory'. On Unix systems this looks something like '`.:/usr/user-name/CharToon`'.

3) At the time of this writing, CharToon's audio facility only works on Sun systems.

4) In this manual we assume keyboard and mouse with (at least) two buttons as input devices. By 'mouse click' we will mean a click with the *left* mouse button. The mouse may be replaced by some other input device such as tablet and pen.

2. Starting up CharToon

2.1. Integrated mode

All components of CharToon can be started at once and used in a co-ordinated way, in the so-called ***integrated mode***. This usage is recommended if:

- the user is designing a face and testing it by playing animations in an interwoven way,
- the user is a novice.

CharToon is started in integrated mode by the following command:

```
java AnimEdit -i [-f facefile] [-w width] [-h height] [-l userlevel]
```

where **`facefile`** is a file path name (without .dat extension) in the **`Faces`** directory, **`width`** and **`height`** are integers specifying the size (in pixels) of the Animation Editor window, **`userlevel`** indicates the expertise of the user to which CharToon is tailored (see 2.3). Knowing the resolution of your screen, you can specify the proper size of Animation Editor at startup time with the `-w` and `-h` options.

CharToon can be quit by the *`File/Exit`* menu item in the Animation Editor window.

2.2. Arranging windows of components

When started up in integrated mode, three windows pop up: the Face Editor window, the Animation Editor window and the Face Player window. Each window can be iconized in the (somewhat system-dependent) usual way, by selecting a button near the right top corner of the window title bar. Before starting to work with CharToon, iconize all the three windows, and adjust the size and location of each window in turn according to the following principles:

1.a.resize Face Player as small as possible, but still showing the complete face and the buttons below it,

1.b.move Face Player window to the bottom right corner of your screen,

2. move and resize Animation Editor such that it becomes as large as possible, but does not overlap the Face Player window,
3. make sure that the one-line message at the bottom of the Face Editor window is not covered by the icon bar (one may move the icon bar or hide it).

Now you are ready to use CharToon. Typically, you iconize Animation Editor and Face Player when editing a face, and recall Animation Editor and Face Player and iconize Face Editor when working on an animation. Debug messages are sent to the window where CharToon was started from.

!! When the -i option is used without the -f, the user is forced in FaceEditor to open a face or begin a new one.

2.3. Setting the user level

CharToon is a complex program, which can be tailored to the following usage levels:

0: DEMO	limited functionality, nothing can be saved,
1: NOVICE	basic functionalities,
2: ADVANCED	full functionality,
3: EXPERT	as 2 plus some experimental features to use sound and print animation score + debug messages,
4: GURU	as 3 plus additional debug information.

Unless a demo version of CharToon was shipped, the user level can be set in two ways.

- By startup, the user level can be defined by the `[-l` **`level`**`]` option, where **`level`** is the appropriate integer.
- Once CharToon is running and not in demo level, the user level can be redefined by invoking the menu *`Options/Set user level`*.

According to the user level, some functionalities are not available, indicated by menu items which are present but cannot be activated or are even absent. It is recommended to set the user level appropriate. For first sessions, set the user level to 0 or 1. The user level should not be set to 3 or 4, unless sound is to be used or debugging information is needed.

!! Note that if the user level is set to 0, the user level cannot be changed any more in that session.

3. Face Editor

3.1. The principle of creating faces with Face Editor

Face Editor is the component of the CharToon system by which drawings (of a face) are created. The program is intended for the generation of 2D faces with a cartoonish or schematic appearance which later can be animated (see Plate 2.a and 3.a).

Drawings are built from pre-cooked components. The user includes those components in the drawing by selecting them from a menu and dragging an inserted instance of them into place. Generally speaking, components are elementary geometrical shapes like polygons, ellipses and splines or they are combinations of those shapes. Components are defined by the points through which they pass. There are ***control points (CPs)*** and ***fixed points (FPs)***, see further 3.7.1. After a component is included, it can be edited, i.e. its appearance can be changed without changing its type. As an extra there is the possibility to include '.gif' images for a kind of background.

While creating a drawing the user also specifies what its potential for animation is. The possibilities are:

1) Many components have 'internal' animation parameters (control points or CPs) by which their ***shape*** can be changed. These parameters are a permanent property of those components. This is illustrated in Figure 2, top row where the eyelids open because of a shape-change.

2) For every component there is the option to include an instance of it in the drawing in such a way that its ***overall position*** can be changed dynamically. This is illustrated in Figure 2, second row, where the pupil moves without changing shape. (Note: when a component is included with that option, the possibility of position change is a permanent property of this instantiation of it).

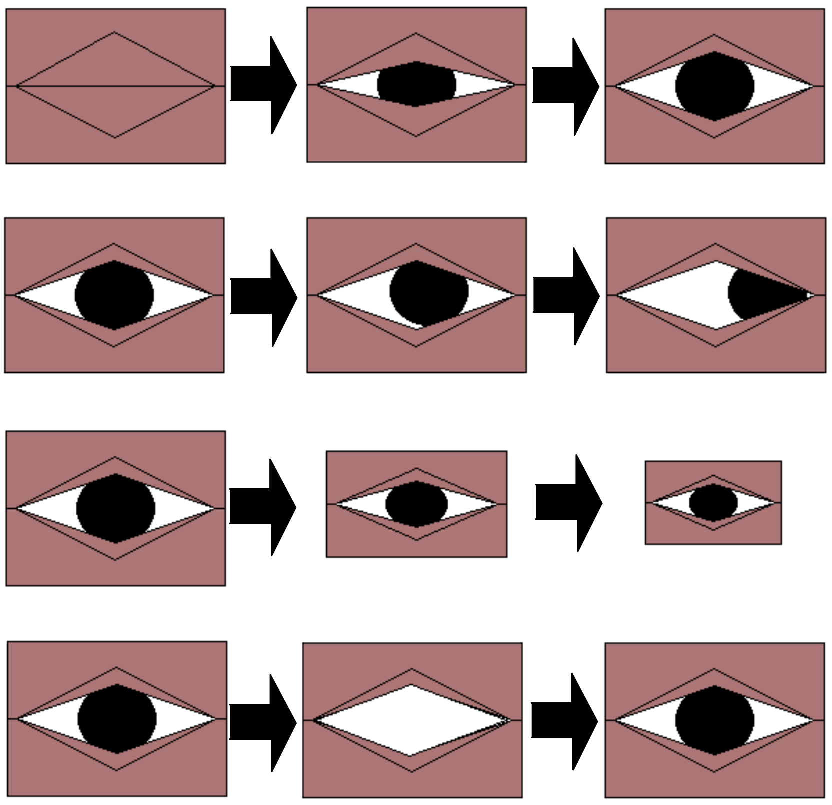
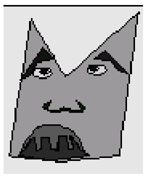


Figure 2. Various ways to animate components. Shape change (top row), changing position (row 2), scaling (row 3) and animated visibility (bottom row). Rows should be interpreted as depicting animations.



3) For every component there is the option to include an instance of if in the drawing in such a way that its ***size*** (scale) can change. This is illustrated in Figure 2, third row, where the eye as a whole is scaled (this implements zooming!). (Note: when a component is included with that option, the possibility of scaling is a permanent property of this instantiation of it).

4) For every component animated ***visibility*** can be enabled (or disabled) while editing the component. In Figure 2 bottom row the pupil has been given the animated visibility property; hence it can disappear and reappear during animation.

Without having to produce (test) animations, it is possible to experiment with the effect of changing the position of CPs by switching to so called ***Test Mode*** (see 3.6.4). In this mode the user can move CPs around by dragging them with the mouse. (As discussed in Chapter 6, the construction of actual animations is done with the help of Animation Editor.)

When a drawing is complete, it has to be saved. In its basic form, this produces ascii files which describe the drawing in some coded form. Saving produces face files which make it possible to reload the drawing in Face Editor in order to modify it and to load it in Face Player in order to be able to play an animation for it. Furthermore, profile files are generated which describe the animation parameters of the face in a form suitable for Animation Editor.

3.2. A simple exercise

If the reader feels like getting the flavor of Face Editor before reading the more technical material in the coming chapters, the following exercise is suggested:

1) Open a window on your system and change the working directory to the home directory of CharToon. Next, type `java FaceEditor`. Now Face Editor's main window appears on the screen.

2) Select *File*, then select *New Face* in the menu which pops up. Next, type a name (e.g. test1) in the dialog-box which just appeared, type <ENTER> while the cursor is still in the text-field of that box and push its *OK* button. The dialog-box and pop-up menu disappear and only Face Editor's main window remains.

3) Select *IncludeComp*, a large menu appears with buttons showing icons of drawing components. Some contain green hairline crosses, depicting CPs, others don't. Select the icon with the label 'Filled Polygon', it will be outlined in red. Then push the *OK* button in the menu; the menu disappears and a triangle is shown in the drawing area of Face Editor's main window. Press the mouse somewhere in this triangle and drag it around.

4) Repeat step 3, but now select the 'Animated Filled Poly' component which looks like the previous one but which has a green hairline cross instead of a red dot as its top corner.

5) Select *Test* in Face Editor's main menu. Check the *All* radio button and select *Apply* followed by *Hide* in the 'Selection- and test specification' menu which pops up. Now try dragging the green CP around by its centre. Observe the effect. Finally, select *EndTest* to quit test mode.

6) Select *SelectionOn* in the main menu. In the 'Selection- and test specification' which appears, check both *All* buttons and select *Apply* and *Hide*. Now click in one of the triangles. Observe that the one clicked in retains its original color and gets a highlighted outline, while the color of the other triangle turns to gray. Next click in the other triangle. In general, the non-gray component is the *selected* one, main menu operations like *Edit*, *Drag* and *Copy* apply to that component. Try *Copy*, next *Delete* and then select one of the remaining two components by clicking.

7) Press the CTRL key and select *Drag* in the main menu. The CTRL key switches on the on-line help function of Face Editor. A window appears describing the *Drag* function. Before going on, the help window must be removed by selecting its *OK* button.

8) Select *Edit* in Face Editor's main menu. An edit menu pops up with operations which are possible for the selected component. Select the *Mirror Vertical* item and see what happens. Next select *DragFixedPoints*, and try to drag one of the red points in the selected component with the mouse. Also try moving

the sliders in the FixedPoint Mover which has meanwhile appeared in the left upper corner of the screen. Observe that the on-line help (CTRL + menu select) also works for edit menus.

9) Select `StopEditing`, select the other component and edit that one. Note the difference in the edit menus for the two components.

10) Select `File`, then in the pop-up menu select `Save`. The drawing is now saved under the name specified in step 2 above.

11) Select `File`, then in the pop-up menu select `Open`. A file-dialog appears which lists already existing faces. Check that the name for the drawing you just created appears in that list.

12) Open one of the faces (for instance 'DemoHead' or 'Chicken') included in the CharToon release, by clicking on its name in the file dialog and thereafter selecting the `Open` button. The selected face appears in Face Editor's draw area. Explore the face using `Test` or `SelectionOn` and `Edit`.

3.3. Ways to start up Face Editor

Face Editor can be started up in two ways: in ***integrated mode*** or in ***stand alone mode***.

Integrated mode

In integrated mode, Face Editor comes up together with Animation Editor and Face Player. Integrated mode is obtained by starting Animation Editor with the '-i' option (see 2.1.). In that case, the three components come up in a setup which is suited for the interwoven development of a face and an animation for it. In particular Face Editor will show the face we are currently developing, Animation Editor is prepared for the construction of an animation for that face and Face Player to show snapshots or play segments of that animation.

Stand alone mode

In stand-alone mode, there is no automatic communication between Face Editor and Face Player or Animation Editor. In order to start Face Editor stand alone, go to the working directory and type:

```
java FaceEditor [-f facefile] [-l level]
```

The `-f` option is used to instruct Face Editor to load the face found under Faces and identified by the path **`facefile`**. If this option is not used Face Editor comes up with a blank drawing area. In that case the user can either begin constructing a new face or explicitly load an existing one.

The -l option deals with the user levels 0 (DEMO), 1 (NOVICE), 2 (ADVANCED), 3 (EXPERT) and 4 (GURU) as used by CharToon (see 2.3.). With DEMO level, save operations are forbidden. Face Editor does not currently distinguish between levels NOVICE, ADVANCED and EXPERT: in those cases full functionality is provided (but the rest of CharToon does differentiate between those levels). In GURU level there is full functionality and a lot of debug information is printed in the start-up window.

3.4. On-line help

Face Editor has two ways of obtaining on-line help:

1) Use the `Help` main-menu item. This function provides a (very sketchy) overview of Face Editor. After selecting `Help`, a scroll list pops up with the titles of some help topics. When double clicking on one of those, a window appears which gives information on this topic. To quit this help facility, select the `Close` button which can be found at the bottom of the window with help titles.

!! Note: there is a bug on Windows systems; the window sometimes comes up grey. Click on its left- or right border to get its content.

2) There is on-line help available on every menu item. To get help press the CTRL key and select the menu item. Push `OK` to get rid of the help window which has popped up.

3.5. Components, drawing order and rank

In Face Editor, drawings are constructed by repeatedly adding components. By default, Face Editor draws components in the order in which they are added. Because components can (partially) overlap, more control over the drawing order is needed then the one imposed by adding order.

This problem is even more serious in Face Player: there the order in which components of a drawing are rendered in every single animation frame can in principle be undetermined because of parallelism inherent in Face Player. So the way in which components overlap is unspecified and can change from frame to frame.

In order to deal with this problem, set two properties of components in Face Editor which are also taken into account in Face Player (see Figure 3):

1) Components can be declared foreground or background (menu item *SetBackGr*).

2) The user can assign a rank (in the range 0 - 9) to a foreground- or background component (menu item *SetRank*).

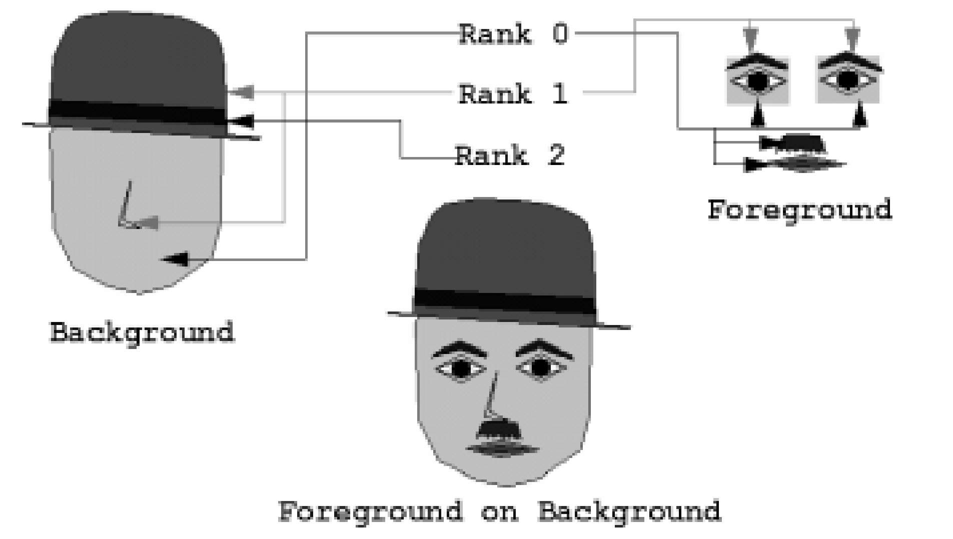


Figure 3. Foreground, Background and Rank.

With foreground/background and rank the drawing order is as follows:

1) First, all background components are drawn, beginning with those of rank 0, then those of rank 1 etc. The drawing order of background components of the same rank is still not specified.

2) Next, the same happens for foreground components.

3) In Face Player, an extra property holds which is important for animation speed:; background components are only rendered once at the start of the animation.

As a consequence, put components into the background which:

1) do not change shape, size and position during animation and

2) do not partially cover a changing component.

By careful layering an effect of depth or 3D perspective can be achieved (see Plate 13).

3.6. Face Editor’s main menu

There are three groups of menu items at the right part of Face Editor’s main window, together referred to here as Face Editor’s ***main menu***. The topmost group is basically about saving and loading drawings, the middle one has operations pertinent to the whole drawing and the bottom one contains operations involving single components or sets of components. In the following paragraphs we discuss these operations one by one, starting at the top and working our way to the bottom. For explanatory reasons, some of the groups are further subdivided in this discussion.

3.6.1 Opening and saving faces

The most important save operations are there to generate the face (‘.dat’) files describing a face and the profile (‘.aprof’) files describing its animation parameters. A face can be opened in Face Editor form its face file. Furthermore, face files and profile files are used for the communication between CharToon components, see 1.1. The `File` item gives immediate access to the basic operations (`NewFace, Open, Save` and `SaveAs`) which is all a non-expert user needs. Furthermore there are some advanced save (and corresponding open) operations which are to be found under `File/More`.

In the remainder of this paragraph we discuss the open- and save operations one by one in the order in which they appear in Face Editor’s main window.

SetCurrent: This menu item only appears in integrated mode. When selected, it transmits changes to the face being edited to Animation Editor.

`SetCurrent` generates a face file for Face Player and a profile file for Animation Editor (both with a unique user-dependent name).

!! Note that these generated files are temporary. Hence it is a must to save the face to a user-defined file before quitting CharToon.

File: The `File` menu item gives access to Face Editor’s operations on face files. It brings up a submenu with items `NewFace, Open, Save, SaveAs, More` and `ParentMenu` all of which are explained below.

File/NewFace: `NewFace` is there to begin a new drawing. It clears the drawing area after which a dialog appears, through which the system asks for a name for this drawing.

The user should:

1) Enter the name.

2) Hit the <ENTER>-key while the cursor is positioned at the end of that name.

3) Push `OK`.

This name is used as a default for save operations, unless the user saves to explicitly given filenames.

See also: `Open, More/OpenObject`.

File/Open: `Open` is used to open a drawing which was saved (as a ‘.dat’ file) by `Save` or `SaveAs`. When `Open` is selected, a file selection dialog will appear from which the user selects the name of the face to open by either selecting a file in the Files list or by explicitly entering a filename.

See also: `Save, SaveAs, More/OpenObject, More/SaveObject, More/OpenObjectAs`.

File/Save: `Save` is used to save the current drawing. Initially, the drawing is saved using the name with which it has been created by `NewFace`. When meanwhile another name has been specified during a `SaveAs` operation, that name will be used by `Save` instead. While saving, two ascii files are generated: a description of the face’s components (a ‘.dat’ file, put in directory Faces) and a separate description of its animation parameters (an ‘.aprof’ file, placed in directory Profiles).

See also: `Open, SaveAs, More/SaveFaceScriptAs, More/SaveProfileAs.`

File/SaveAs: `SaveAs` is used to save the current drawing.

A file selection dialog will appear by which the user specifies a file name by either selecting a file in the Files list (the file then will be overwritten), or by explicitly entering a name for the file.

!! Although face files should have a '.dat' extension, the user should not type one: It is automatically generated by CharToon. If a file name has been selected from a list, the .dat extension should be removed.

While saving, two ascii files are generated: a description of the face's components and a separate description of its animation parameters. By default, the face file is placed in subdirectory 'Faces' and the profile in subdirectory 'Profiles'. .

See also: *Open, SaveAs, More/SaveFaceScriptAs, More/SaveProfileAs.*

File/More: Selecting this menu item produces a submenu with open- and save operations of a more complex kind then the ones in the *File* menu. Those operations are explained below.

File/More/OpenJavaObject: *OpenJavaObject* opens a face which has previously been saved as a Java object by *SaveObject* or *SaveObjectAs*. A file selection dialog will appear by which the user specifies a face by either selecting a file in the Files list or by explicitly entering the name of the face to be opened.

See also: *NewFace, SaveObject, SaveObjectAs.*

File/More/SaveJavaObject: *SaveObject* is used to save the current drawing as a Java object. By default, the object is saved using the name with which the drawing is created (by *NewFace)* or loaded (by *Open* or *OpenJavaObject*). When meanwhile another name has been specified during a *SaveAs* operation, that name will be used by *Save* instead. The file in which the object is saved is placed in directory 'FaceObjects'.

The use of *SaveJavaObject* is to save a drawing in order to continue working on it at a later time using *OpenJavaObject*. *SaveJavaObject* can *not* be used to save the drawing in a form suitable for Animation Editor or Face Player. To that end, use *Save, SaveAs, SaveFaceScriptAs* or *SaveProfileAs*.

Note: Within the framework of CharToon, there is not much point in saving faces as Java objects. Saving as face files is as good or better. The possibilities for saving as object are provided in order to enable a user to generate faces which can be potentially dealt with by other Java program which uses the drawing component class hierarchy of CharToon but does not know of face files.

See also: *OpenObject, SaveObjectAs, Save, SaveAs, SaveFaceScriptAs, SaveProfileAs.*.

File/More/SaveJavaObjectAs: *SaveJavaObjectAs* is used to save the current drawing as a Java object.

A file selection dialog will appear from which the user selects a name for the saved object by either selecting a file in the Files list or by explicitly entering the name of the saved face object. The object is stored as a file which is placed in directory 'FaceObjects' unless the user explicitly specifies another directory in the file dialog. Saved object files do not have a file extension, neither generated automatically nor provided by the user.

The reasons for using *SaveJavaObjectAs* are identical to those for using *SaveJavaObject.*

See also: *SaveObject, OpenObject, Save, SaveAs, SaveFaceScriptAs, SaveProfileAs.*

File/More/SaveFaceScriptAs: *SaveFaceScriptAs* is used to save the current drawing as an ascii '.dat' file. stored in directory 'Faces'. The difference with *File/Save* and *File/SaveAs* is, that *only* a '.dat' file is generated and *no* '.aprof' file. In other aspects, *SaveFaceScriptAs* behaves as *File/SaveAs*.

Suppose the user wants to change a face a little, e.g. by removing a CP and still be able to edit and play old animations for it. In that case, the changed face should be saved without changing its profile. *SaveFaceScriptAs* does just that. Such possibilities are further explained in 6.1.1.

See also: *Save, SaveAs, IncludeScript, More/SaveProfileAs.*

File/More/SaveProfileAs: *SaveProfileAs* creates a profile file with '.aprof' extension in directory 'Profiles'. This file describes the CPs of the current drawing in the format expected by Animation Editor. The difference with *File/Save* and *File/SaveAs* is, that *only* an '.aprof' file is created and *no* '.dat' file. The name of the saved file is entered through a file-dialog which pops up when *SaveProfileAs* is selected.

This facility is to be used if the user wants to experiment with an existing face but with different ranges for its animation parameters and wishes to reuse animations made earlier for the original face. A solution is to edit the face's control ranges in Face Editor and then to produce a profile file reflecting those changed ranges (without changing the face file) using *SaveProfileAs.* For using such a profile see 6.1.1.

See also: *Save, SaveAs, More/SaveFaceScriptAs.*

File/More/PackIds: Animation parameters are assigned numerical identifiers when they are created by Face Editor. These ids play a crucial part in the interface between Face Editor and Animation Editor. When components (or just CPs) are deleted during editing of a face, gaps are produced in the sequence of identifiers. Normally this does not matter, but when the gaps and consequently the large ids are getting annoying (Animation Editor will show these gaps in the parameter numbers). *PackIds* will generate a new set of consecutive ids which will be used by *Save, SaveAs, SaveFaceScriptAs* and *SaveProfileAs* for the generation of profile files.

!! After the execution of *PackIds* and one of the *Save* operations, *animations made earlier* by Animation Editor *will no longer work as intended*! So either use *PackIds* after editing a face but before making animations, or make sure that the parameter names are not changed so that Animation Editor's Reanimate facility can be used for the loading of animations which were made prior to the *PackIds* operation.

File/More/ParentMenu: See **File/ParentMenu**.

File/ParentMenu: Selecting *ParentMenu* brings one back to the menu one level higher in the menu hierarchy.

3.6.2 Global drawing options

The operations discussed here are there to assist the user in producing drawings by providing grids and hairlines or by controlling the way in which components can be selected. They are all accessed from the main menu item *DrawOptions*.

Because Java draws in pixels, Face Editor does the same. This has the consequence that if a drawing is designed on a large, high-resolution screen and later it is displayed on a screen with less pixels, it will only be displayed partially. All of it can still be viewed though, using Face Editor's main menu *Drag* function. That way, the drawing can be viewed piece by piece.

When the user resizes the drawing window, the following happens, depending on which one of the menu items *DrawOptions/AdaptDrawingSize* or *DrawOptions/FreezeDrawingSize* has been activated the last.

In the case of *AdaptDrawingSize***,** enlarging the window results in a larger drawing area without scaling the drawing, reducing the window results in hiding part of the drawing, again without scaling. In any case a resize can be undone by the inverse resize. This effect can be made permanent by a *Save*: the drawing is then saved with the size as shown on the screen at the time of the *Save*.

In the case of *FreezeDrawingSize*, the drawing does not respond to window-resizes: only when the window is made smaller then the drawing, part of it is obscured.

!! It is recommended to fix the drawing size to avoid unintended size changes.

Including a face file of a drawing (*IncludeScript*) into another drawing will influence the size of the other drawing if the included one is larger. In that case the drawing area will be enlarged to the size of the included drawing without resizing or displacing the first one.

DrawOptions: When this item is selected, a submenu appears having entries: *Grid, Hairline, ShowSelection_TestControl, HideSelection_TestControl* and *ParentMenu*. These items are explained directly below.

DrawOptions/Grid: *Grid* displays a dialog box which can be used to specify a drawing grid. A drawing grid is a temporary collection of horizontal- and vertical lines which can be used to precisely place drawing components into position. The dialog box contains:

1) A set of radio buttons (top row) used to select the grid-spacing. There are buttons with labels like '*50*50*' which will produce a grid dividing the drawing in 50 squares horizontally and 50 squares vertically irrespective of drawing size and there are buttons like '*10px*' which will produce a 10 pixels wide grid.

2) A set of radio buttons (middle row) used to select the grid-color.

When the *OK* button of the dialog is pressed, the grid is actually applied. The *Cancel* button disposes of the dialog without any effect on the grid.

See also *HairLine.*

DrawOptions/Hairline: *Hairline* displays a dialog box which can be used to specify a hairline cross. A hairline cross consists of a horizontal and a vertical line which can be used to precisely place drawing components into position. The hairline can be positioned by dragging its arrow-heads.

The dialog box contains:

1) A button (top row) used to indicate whether the hairline should be shown or not.

2) A set of radio buttons (middle row) used to select the hairline-color.

When the *OK* button on the dialog is pressed, the hairline is actually applied. The *Cancel* button disposes the dialog without any effect on the hairline.

See also *Grid.*

DrawOptions/ShowSelection_TestControl: *ShowSlection_TestControl* limits the set of components of a drawing from which it is possible to select.

The user has to select a component if it must be dragged, deleted or modified by Edit operations. Select by either clicking in the interior of a closed component or by clicking on one if its CPs or FPs. The cursor must also be placed at CPs when in test mode.

Because a complex drawing is crowded with (sometimes hierarchical) components, selecting by clicking in the unrestricted set of components can be rather difficult because sensitive elements are far too close to each other. *ShowSelection_TestControl* displays a dialog box which can be used to limit the set of sensitive components. The components which can be selected are highlighted (as well as their FPs- and CPs). The dialog contains:

1) A button (*Select or test foreground components*) which, when pushed, makes foreground objects selectable.

2) A button (*Select or test background components*) which, when pushed, makes background objects selectable.

3) A slider which when used specifies *one* foreground layer to select from. Only the layer indicated by the slider is made sensitive, other layers which might have been made sensitive by the check-buttons (see below) are deactivated. So use the slider to find out to which layer a component belongs and to make that layer the only one which is sensitive for selection (ore testing).

4) A set of check-buttons, one for every foreground layer. When the button labelled ‘i’ is pushed, foreground objects with rank ‘i’ are selectable. The user can also press the *all* button. In that case all foreground objects, irrespective of their rank, can be selected.

5) The same as 3) and 4) above, but then for background components.

6) A button labelled *Show not selected in gray*. When checked, the whole drawing is shown in various shades of gray, apart from the selected (sub)component, which is rendered in its full color. When this button is not checked, all of the drawing is in full color and the selected (sub)component is only outlined by a thick contour of contrasting color.

For testing (see 3.6.4), a subset of this dialog appears. In that case it is only possible to indicate which foreground objects can be tested.

The user actually changes the set of selectable components by pressing *Apply*. The dialog disappears without any effect when *Hide* is pushed.

See also: *DrawOptions/HideSelection_TestControl*, *Selection On, Selection Off, Test, EndTest, SetRank, SetBackGr.*

DrawOptions/HideSelection_TestControl: This menu-choice removes the Selection- and Test control dialog from the screen.

See also *DrawOptions/ShowSelection_TestControl*.

DrawOtions/FreezeDrawingSize: Normally, the face drawing adapts its size (without scaling) when the Face Editor window is resized interactively. When *FreezeDrawingSize* is selected, this feature is disabled; the face's size is frozen to its current size. It the window is made bigger, a white empty border is produced, if the window is made smaller, the face is partially obscured but does not get smaller**.**

See also: *AdaptDrawingSize.*

DrawOptions/AdaptDrawingSize

Normally, the facial drawing adapts its size when the Face Editor window is resized interactively. When this feature has been disabled by *FreezeDrawingSize, AdaptDrawingSize* enables it again.

See also: *FreezeDrawingSize.*

DrawOptions/ParentMenu: See **File/ParentMenu.**

3.6.3 Global drawing operations

The operations discussed in this paragraph act on the whole drawing by changing its global color, position or size.

SetBackgrColor: *SetBackGrColor* displays a color selection tool through which the color of the background of the drawing is specified.

In the color selector a color is selected either by:

1) clicking in the upper left hand square to select a color.

2) clicking in the face drawing to select a color which has already been used (this may take a long time).

3) fine-tuning the selected color by clicking in one of the two color strips on the right hand side of the selector.

4) using the sliders on the lower left hand side of the color selector.

5) clicking in the horizontal grey strip to select a shade of grey.

In all cases the selected color appears in the lower right hand square of the color selector. The background is actually set to the selected color after pressing the *Apply* button. Terminate color selection by pressing the *Stop* button.

Drag: *Drag* is used to indicate that the user wants to drag the drawing to a different position. In order to drag select *Drag* and hold down the mouse button and move the mouse in order to move the drawing.

!! Note: After *Drag* the face must be saved because the position of CPs has changed. Note also that scaling does not change the size of the drawing area if it was frozen before.

Scale: *Scale* is used to indicate that the user wants to scale the drawing to a different size. In order to scale: select *Scale* and move the mouse holding the button down.

During scaling a cross appears in the middle of the drawing. This cross represents the origin of the scaling transformation. Moving the mouse to the right or left (holding the button down) enlarges resp. reduces the drawing in the horizontal direction. Moving the mouse up or down changes the vertical dimensions of the drawing.

!! Note: After *Scale* the face must be saved because the position and range of CPs has changed.

3.6.4 Test mode

It is possible to get an idea of the potential for animating a face by trying out the effect of dragging the positions of CPs. (Visibility- or scale parameters cannot be tested.)

Test: The *Test* button puts Face Editor in test mode. In this mode, the user can drag a CP to observe its effect. Changes made that way are not permanent. When *EndTest* is selected, the drawing is restored to its neutral state.

See also: *EndTest.*

EndTest: When *EndTest* is selected, Faced Editor is put back from test mode to normal mode. Changes to the drawing made during testing are undone.

See also: *Test.*

3.6.5 Intermezzo: component selection

Before the user can perform operations (e.g. Copy, Drag or Edit) on a single component he first has to indicate the component which is the target of the operation. This process is called ***selection***. Having a rough understanding of selection is necessary before reading the next paragraphs, so here is how it works:

- First enable selection using the *Selection On* operation (for details, see below). While doing so, one can also restrict the set of selectable components.
- Next select a component by clicking in its inside or on one of its fixed or control points (which are shown with emphasis for selectable components).

When it is necessary to edit a subcomponent (or subcomponent of a subcomponent etc.) of a composite component select that one by descending the (sub)component's hierarchy. This is done as follows:

- First select the component, then *Edit* it (for details see below). In the edit menu which pops up, select *SelectSubComponent* after which it is possible to select one of the component's subcomponents in the usual way by clicking.
- Next deal further with this subcomponent via the *EditSelectedSubcomponent* menu item.

Repeat this process as long as one has to descend to deeper subcomponent levels. At every step, an extra edit menu appears which is pertinent for the subcomponent selected latest (i.e. the deepest in the hierarchy). As an aid in keeping track of those menus they show the nesting level of their subject (sub)component in their title-bar. In order to go up in the hierarchy use *StopEditing* from one of the edit menus. Suppose uses *StopEditing* is used from a menu which is at level i. Then all menu's with level >= i disappear and one ends up with the menus for those components with levels < i which were visited while descending the hierarchy. In short it is possible to move up in the hierarchy without retracing all the steps taken on the way down.

3.6.6 Operations on components

Face Editor's main menu operations discussed in this paragraph only operate on one single drawing component or are used to add components to the drawing. For operations which act on a component, that component has to be selected (see 3.6.5) first.

IncludeComp: *IncludeComp* shows a dialog box from which the user can select a drawing component to be added to the current drawing (see Figure 4). A component is selected by clicking on its icon. The selected component is highlighted by a red border around its icon. The user can get interactive help on the component by clicking on it while holding the CTRL key down.

At the bottom of the dialog box there are 4 buttons:

1) *Animate Origin*. When this button is checked, the selected component will be included in the drawing with an animated origin, which makes it possible later to dynamically move the component without changing its shape by modifying an x- and y- position animation parameter (using Animation Editor). In Face Editor, an animated origin is shown as a green hairline cross with a light blue circle, an non-animated origin (the default) is displayed as a light blue dot.

2) *Animated Scale*. When this button is checked, the selected component will be included in the drawing with an animated scale, i.e. it is possible later to dynamically change the x or y scale of the component by modifying just an x- and y- scale animation parameter (using Animation Editor). Scaling amounts to multiplying the drawing's coordinates with respect to some point. This point's position is identical to the position of the component's origin and (in the current version of Face Editor) it cannot be set to a different value.

3) *OK* The selected component is included in the drawing.

4) `Discard` The dialog box disappears. No new component will be included.

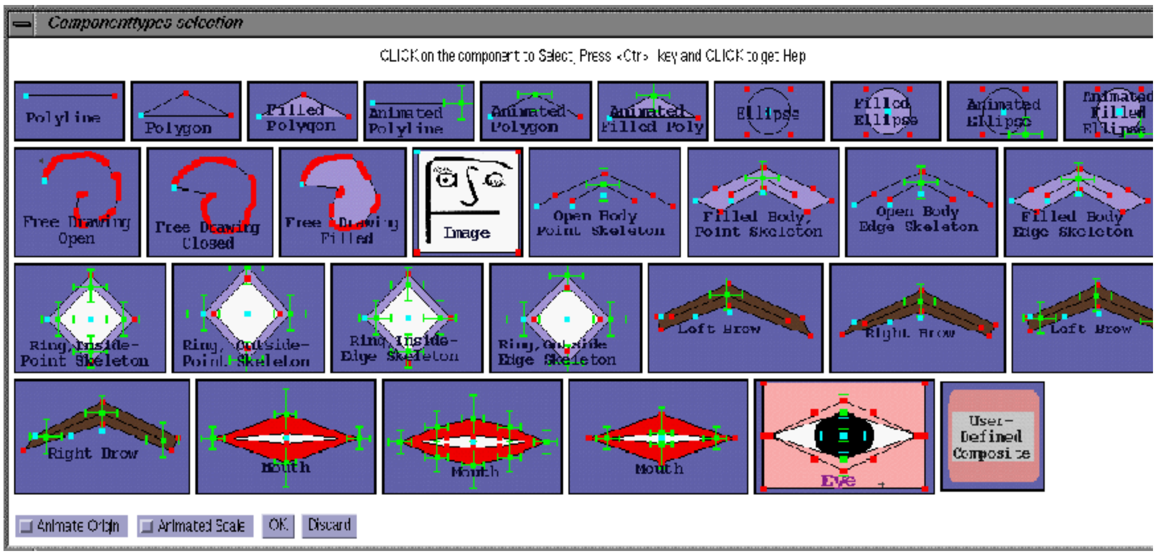

CLICK on the component to Select, Press <Ctrl> key and CLICK to get Help

Figure 4. Face Editor's component selection window. For every type of component, there is an icon which must be selected in order to select that component for inclusion. At the lower left corner there are the 'Animated Origin' and 'Animated Scale' buttons which can be checked for any selected component, before its is included by pressing the 'OK' button.

IncludeScript: `IncludeScript` is used to include in the current drawing another drawing which was saved as a '.dat' file by `Save`, `SaveAs` or `SaveFaceAs`. When `IncludeScript` is selected, a file selection dialog will appear from which the user selects the name of the drawing to include by either selecting a file in the Files list or by explicitly entering the filename. The difference with `File/Open` is that `IncludeScript` adds the drawing to the existing one, while `File/Open` replaces the current drawing.

See also: `Save`, `SaveAs`, `SaveFaceAs`.

Selection On: `Selection On` turns on the possibility to select a drawing component by a mouse click. After `Selection On` is selected the 'Selection- and test specification' dialogue pops up which must be used to indicate which components of the drawing can be selected. For information on this dialog, see the discussion of `DrawOptions/ShowSelection_TestControl`. The components which are selectable according to the 'Selection- and test specification' dialog are shown with highlighted FPs and CPs (see Plate 2.b and 3.b). A component is selected by either clicking on one of those points or (for a closed component) by clicking in its interior. Selecting a component is necessary before performing component-specific operations like `Edit`, `Delete`, `Copy` etc.

See also: `Selection Off`, `DrawOptions/Selecion_TestControl`.

Selection Off: `Selection Off` has a twofold effect:

1) there is no longer any drawing component selected,

2) the FPs and CPs of components are no longer shown explicitly in the facial drawing, i.e. the drawing looks the way it will look like in an animation (see Plate 2a and 3a).

See also: `Selection On`, `DrawOptions/Selection_TestControl`.

SaveSelAsComp: When `SaveSelAsComp` is selected, the *currently selected component* of the drawing is saved as a so called User Defined Composite Component (see 3.7.6).

A file selection dialog will appear in which the user specifies a name for the composite component to save by either selecting a file in the Files list or by explicitly entering a filename.

The User Defined Composite Component which is constructed this way can be incorporated in (other) drawings by using the *IncludeComp* main menu item followed by the 'UserDefinedComposite' button (appearing in the menu which pops up after *IncludeComp* is selected).

The motivation for creating a User Defined Composite Component can be:

1) to reuse parts of a drawing.

2) to create a component from a set of other components which behaves as a unit under scaling, dragging etc.

See also: *SaveAllAsComp, IncludeComp.*

SaveAllAsComp: When *SaveAllAsComp* is selected, the *collection of all components* of the current drawing is saved as a so-called User Defined Composite Component. In all other aspects, *SaveAllAsComp* is identical to *SaveSelAsComp*

See also: *SaveSelAsComp, IncludeComp.*

Delete: Selecting *Delete* deletes the currently selected component from the drawing. When no component is selected, nothing happens.

Copy: *Copy* makes a copy of the selected component which is included straight away in the current drawing.

Edit: When *Edit* is selected, the currently selected component (if any) will be editable. A menu pops up showing the edit operations which are possible for the component (see 3.8.).

Drag: *Drag* is used to indicate that the user wants to drag the selected component to a different position. In order to drag select *Drag* and drag with the mouse.

SetRank: *SetRank* gives the user the possibility to set the rank of a component by entering the rank number in the appropriate field of the dialog which pops up after *SetRank* is selected. Rank-numbers range from 0 to 9 inclusive. The rank-numbers of foreground and background components are taken into account independently.

During animation, first the background components are drawn in order of increasing rank, next the foreground objects are drawn, also in order of increasing rank. The drawing order of objects of the same rank which both belong to the background or both belong to the foreground is unspecified.

See also: *SetBackGr, SelectionControl.*

SetBackGr: *SetBackGr* places a component in the background or brings it back to the foreground. The components of a drawing are either foreground- or background components. During animation, background components are only drawn once, whereas the foreground components are drawn on top of the background for every new animation frame.

For efficiency reasons it is recommended to place in the background every component which:

1) does not change shape, size and position during animation and

2) does not partially cover a changing component.

See also: *SetRank, SelectionControl.*

3.7. Components of drawings

3.7.1 Elements of components

Every component has an ***origin*** which defines its overall position and possibly the centre around which it is scaled. The origin either is a FP (see below) in which case the component stays in place during animation or it is a CP (see below), enabling the animator to move the component as a whole using only two (x- and y) parameters. Origins are shown in Face Editor as cyan (light blue) colored dots, or green crosses with a cyan circle around them.

Internally, an origin serves as the centre of the coordinate system in which the component is defined. This explains why moving the origin amounts to moving the component and why the position of the origin with respect to other elements of the component cannot be changed.

Components are built up from polylines, polygons, splines, ellipses or (.gif) images. All these elements of components are defined, among other things, by their origin, and by points through which they pass or which specify a bounding box which encloses them. In Face Editor these points are rendered as colored objects (see below) and they are mouse-sensitive in order to allow them to be edited. During animation (i.e. when using Face Player) they are not shown.

There are two types of points: Fixed Points (FPs) and Control Points (CPs).

Fixed Points: In Face Editor, fixed points are normally rendered as red dots (unless they are 'frozen' in which case they are dark blue see 3.7.2). There are edit operations to insert, delete or drag them. During animation they remain stationary or they change position as a result of moving or scaling the component to which they belong to or of deforming its skeleton (see 3.7.2).

Control Points: In Face Editor, CPs are normally rendered as crosses with tiny lines at the end of the arms of the cross. A CP corresponds to two animation parameters, one to specify the time-dependent horizontal position of the CP, another one to specify vertical position. The length of the arms of the cross indicates the interval in which they are allowed to move. Normally, these intervals are graphically represented in Animation Editor. Furthermore, CPs have parameter names which can be set by the user. Meaningful names help to identify the corresponding animation parameters in Animation Editor for the benefit of the user. (Note that the software-wise communication between Face Editor, Animation Editor and Face Player does not use those names, but depends on automatically generated unique identifiers.)

There are three kinds of CPs: ***independent*** ones (shown in green), ***leaders*** (shown in yellow) and ***followers*** (shown in grey). Each follower has a leader. When a follower must become a lead itself, *its* leader is used instead. When the leader changes position, the follower moves by a proportional amount (i.e. according to the ratio between the intervals in which the leader and the follower may move). Followers are not shown in Animation Editor and do not have an independent control range.

Naturally, there are edit operations to insert, delete and drag CPs, to change their ranges and parameter names and to specify leader/follower properties.

We can now discuss the elements out of which components are built. As said, these elements themselves depend on FPs and CPs.

Polyline: Components can contain series of FPs and CPs connected by lines. When the first and last point of such a series are not connected we call the connected series a polyline.

Polygon: A polygon is just a closed polyline i.e. its first and the last point are always connected.

Spline: When a component contains a polyline or polygon there is the possibility to draw the same component not with straight lines connecting its points, but with smooth curves (cubic splines). The choice between these possibilities is made through an edit operation in Face Editor, it cannot be changed while playing an animation.

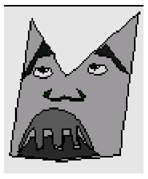
Ellipse: Components can also contain ellipses defined by their width and height.

Image: Finally there are components which correspond to a'.gif' image.

3.7.2 Types of components

We distinguish components in a number of different ways. This results in a (somewhat adhoc) classification which we will often use in this manual.

The main distinction between components is that of ***fixed*** and ***animated***. Fixed components contain only FPs, they do *not change shape* during animation (although they can move around or be scaled when included with an 'animated origin' or 'dynamic scaling', see 3.6.6). Animated ones contain CPs so they *can always change shape* during animation (on top of that they can of course be included in the drawing with animated origin etc.)



Another important distinction between components is that of ***basic*** and ***composite***.

Basic components consist of one and only one polyline, polygon, ellipse or image. We also refer to polyline basic components as ***open***, to polygon and ellipse basic components as ***closed***. In addition, there is the distinction between ***transparent*** and ***filled*** (or opaque) for closed components. The transparent ones are just polygons rendered as a collection of lines, the filled ones consist of a polygon whose interior area is rendered in a specific color.

Composite components are constructions using basic components and other composite components as building blocks. So they consist of a hierarchically ordered set of polylines, polygons etc. Important subclasses of the composite components are the *point skeleton* and *edge skeleton* classes which we will discuss now (The discussion of the UserDefinedComposite subclass will be deferred to 3.7.6).

Skeleton components are a subset of composite components. They consist of two subcomponents: a skeleton and a body both of which are a polygon or polyline. The skeleton is always animated (i.e. contains CPs), while the body contains only FPs which can be considered as coupled to the skeleton. The idea is that when the skeleton moves (i.e. its CPs change position), the FPs of the body move in synchrony with the skeleton.

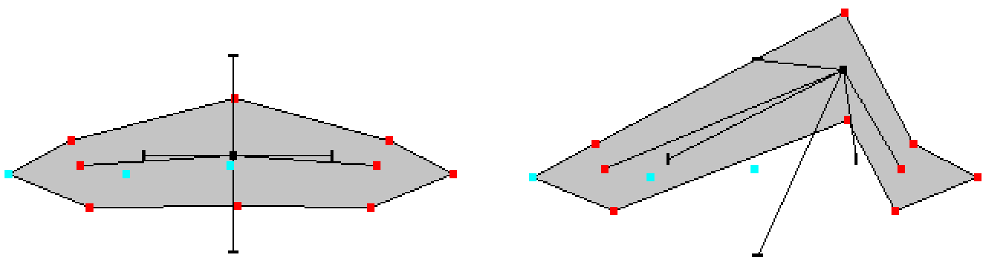


Figure 5. A point skeleton component, neutral (left) and deformed (right).

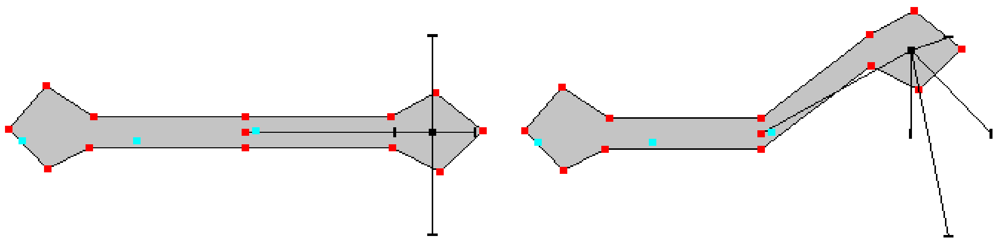


Figure 6. A point skeleton component, neutral (left) and deformed (right). Note the 'joint' in the middle, the short skeleton and the way the 'diamond' shape at the right is preserved.

The way in which body points are coupled to the skeleton leads to the distinction between *point skeleton* and *edge skeleton* components. The two types of skeleton component have in common that FPs of their body's can be immobilized by the *FreezeFixedPoints* edit operation (see 3.8).

Point skeleton components work as follows: When the component is created, hence when the skeleton is in its neutral position and all CPs on the skeleton have neutral value, for every *point* B on the body the *point* S is determined on the skeleton which is closest to B, and where S can be either a CP or a FP. Next the *vector* (i.e. distance and direction) form S to B is determined. During animation these vectors are kept constant, so when the skeleton moves, a body points B remains at a constant distance and in a constant direction relative to its associated skeleton point S. So, when B is initially closest to a CP S it moves when S moves (see Figure 5). When it is closest to a FP S, it remains in place. This last feature gives an easy way to roughly mimic the effect of skeletons with joints (see Figure 6). Unfortunately, the fact that body points *closest* to fixed skeleton points do not move, can have undesirable effects. Consider the mouth-shape shown in Figure 7 below which consists of a ring-shaped point skeleton (modeling the inside of the lips) and a ring-shaped body (modeling the outside of the lips). The four body points close to the mouth-corners are coupled to the fixed skeleton points representing the inside of the mouth-corners. Hence, these four body points do not move when the mouth is opened, which leads at certain places to zero lip thickness. This phenomena can be remedied using edge skeletons (see Figure 10).

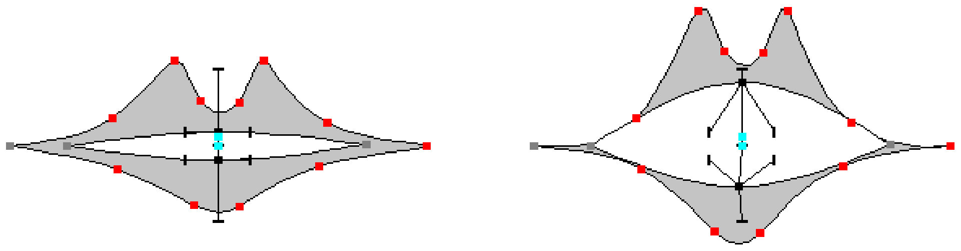


Figure 7. Mouth shape with point skeleton, neutral (left) and deformed (right). The four body-points marked with ‘X’ do not move which results in zero lip-thickness (indicated by arrows).

As said, body points of both point skeleton and edge skeleton components can be immobilized using the `FreezeFixedPoints` edit operation. In this way, body points can be anchored to their environment. The use of this is illustrated in Figure 8 which shows a ‘lower jaw’ constructed from a point skeleton with only *one* control point and a body of eight points. The leftmost and rightmost body points are frozen. That way, the jaw’s endpoints stay in place.

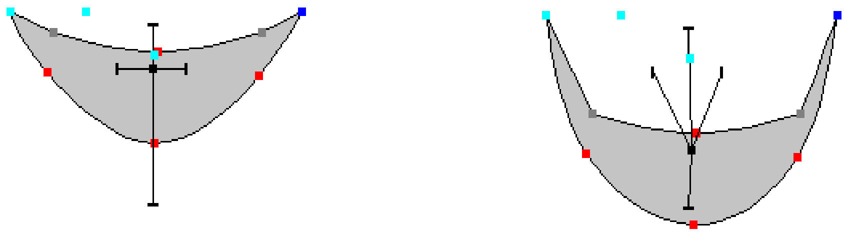
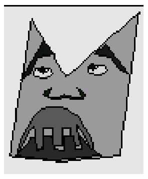


Figure 8. A point skeleton component with two frozen body points (indicated by arrows). Neutral shape (left) and deformed shape (right).

Edge skeleton components differ from point skeletons in the way the body points are coupled to the skeleton. Roughly speaking, body *points* are projected on skeleton *edges* and certain features of this projection are kept constant during animation. The most striking result is, that in principle *all* body points of an edge skeleton component can move. How this works is explained in the following example:



In Figure 9 we see body point B1 projected on a skeleton edge whereby the projected point P1 lies between the endpoints of the edge. Body point B2 has a projection P2 which lies on an extension of the edge. When the skeleton is in its neutral position, the projection distances d1 and d2 are calculated as well as the distances of the projections P1 and P2 from the endpoints of the edges on which they lie (i.e. the distances l1, r1, l2 and r2). Now what happens is, that during animation d1 and d2 and the ratios l1/r1 and l2/r2 are kept constant.

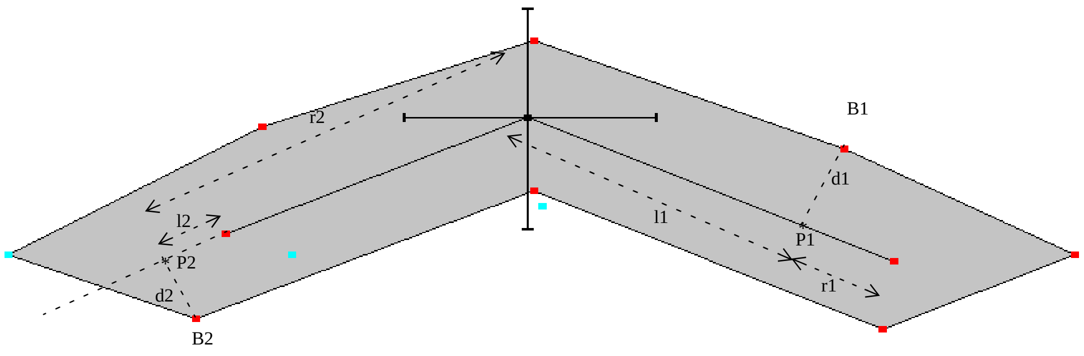


Figure 9. Projection of body points in a skeleton component.

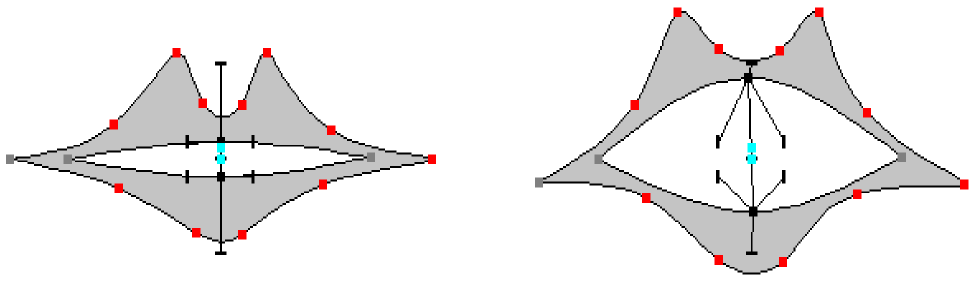


Figure 10. Mouth shape using an edge skeleton, neutral position (left) and deformed (right). Note that the problem occurring with the point skeleton mouth (see Figure 7) is absent here.

How CharToon chooses the skeleton edge on which body point B is projected is somewhat complex. The algorithm is as follows:

1) Do not consider skeleton edges E where the line connecting B with its projection P on E does intersect any other edge.

2) From the remaining edges select those who have an endpoint which lies closest to B. Sometimes there is only one such edge which is then the one to which B will be coupled. Sometimes there are more then one of such edges, either by coincidence or (more probable) because B lies closest to an endpoint which is common to two connecting edges.

3) If among the remaining edges there are edges E such that the projection P of B on E lies between the endpoints of E, retain only those. In other words, remove those edges where the projection P lies on an extension of the edge if there are other edges at all.

4) From the remaining edges choose the one for which the distance between B and P is minimal.

5) If the set of remaining edges still contains more then one edge, just choose the first one encountered in the process.

3.7.3 Basic components

In this section, the basic components are discussed one by one in the order in which their icons occur (from left to right and from top to bottom) in Face Editor's component selection menu (see Figure 4) which appears after main menu item `IncludeComp` is activated.

Polyline: A prototype for a fixed (i.e. containing only FPs) open polyline, having an arbitrary number of FPs as vertices. FPs can be added, deleted and dragged by edit operations.

Polygon: A prototype for a closed transparent polygon, having an arbitrary number of FPs as vertices. FPs can be added, deleted and dragged by edit operations. The polygon remains closed while editing.

Filled Polygon: A prototype for a fixed closed filled (i.e. opaque) polygon, having an arbitrary number of FPs as vertices. FPs can be added, deleted and dragged by edit operations. The polygon remains closed while editing. The user can also edit the (opaque) interior color of the polygon.

Animated polyline: A prototype for an animated (i.e. containing CPs) polyline, having an arbitrary number of FPs and at least one CP as vertices. FPs and CPs can be added, deleted and dragged by edit operations. During animation the shape of the polyline can be changed by moving its CPs.

Animated Polygon: A prototype for an animated closed transparent polygon, having an arbitrary number of FPs and at least one CP as vertices. FPs and CPs can be added, deleted and dragged by edit operations. The polygon remains closed while editing. During animation the polygon can be changed by moving its CPs.

Animated Filled Polygon: A prototype for an animated closed opaque polygon, having an arbitrary number of FPs and at least one CP as vertices. FPs and CPs can be added, deleted and dragged by edit operations. The polygon remains closed while editing. The user can also edit the interior color of the polygon. During animation the shape of the polygon can be changed by moving its CPs.

Ellipse: A fixed transparent ellipse which is specified by a bounding box. The bounding box contains 4 FPs and the ellipse cannot have an interior color, it is transparent. The user can move the vertices of the bounding box by editing.

Filled Ellipse: A fixed opaque (i.e. filled) ellipse which is specified by a bounding box. The bounding box contains 4 FPs. The ellipse has an interior color which can be edited. The user can move the vertices of the bounding box by editing.

Animated Ellipse: An animated transparent ellipse which is specified by a bounding box. The bounding box has three FPs and one CP. The ellipse is transparent. In Face Editor the user can move the vertices of the bounding box by editing. The shape of the ellipse can be changed during animation by changing the position of the CP.

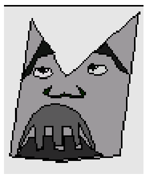
Animated Filled ellipse: An animated opaque ellipse which is specified by a bounding box. The bounding box has three FPs and one CP. The ellipse has an interior color (which can be edited). The user can move the vertices of the bounding box by editing. During animation the shape of the ellipse can be changed by moving its CP.

Free Drawing Open: An arbitrary one-piece fixed open polyline which is created by pressing the mouse somewhere in the drawing and dragging thereafter. When the mouse is released the drawing cannot be changed anymore. After creation, this object can be edited as if it were a Fixed Open Polyline.

Free Drawing Closed: An arbitrary closed fixed transparent polygon which is created by drawing with the mouse just like creating a Free Open Drawing. When the mouse is released, the polygon is completed with one final edge to close it. After creation, this object can be edited as if it were a Fixed Closed Transparent Polygon.

Free Drawing Filled: An arbitrary closed fixed filled polygon which is created by drawing with the mouse just like creating an Free Closed Transparent Drawing. After creation, this object can edited as if it were a Fixed Closed Filled Polygon.

Image: A component which displays an image which must be available as a '.gif' file. Upon creation, this component displays the FASE logo as Image. There is an edit option to change this default image into any



other ‘.gif’ image. The image can be scaled in Face Editor too but is of fixed size and content in an animation. It’s intended use is as background, logo etc.

3.7.4 Skeleton components

Skeleton components give the possibility to roughly mimic the effect of a body coupled to a skeleton. The skeleton is animated (by only a few) animation parameters; while moving it drags along the body (see further 3.7.2.).

Open Body Point Skeleton: An animated ribbon-like object consisting of an open body polyline and a polyline skeleton. The skeleton has CPs. The body is point-animated so when the skeleton moves the *vertices* of the exterior body polygon follow the *vertices* of the skeleton. (For details, see 3.7.2.)

Filled Body Point Skeleton: An animated ribbon-like object consisting of a closed, opaque body polygon with a polyline skeleton inside. The skeleton has CPs. When the skeleton is animated the *vertices* of the outside polygon follow the *vertices* of the skeleton. (For details, see 3.7.2.)

Open Body Edge Skeleton: An animated ribbon-like object consisting of an open body polyline and a polyline skeleton. The skeleton has CPs. When the skeleton is animated the *vertices* of the outside polygon follow the *edges* of the skeleton. (For details, see 3.7.2.)

Filled Body Edge Skeleton: An animated ribbon-like object consisting of a closed, opaque body polygon with a polyline skeleton inside. The skeleton has CPs. When the skeleton is animated the *vertices* of the outside polygon follow the *edges* of the skeleton. (For details, see 3.7.2.)

Ring Inside Point Skeleton: An animated ring-like object consisting of a closed, opaque exterior body polygon with a closed, opaque polygon skeleton inside. The skeleton has CPs. When the skeleton is animated the *vertices* of the outside polygon follow the *vertices* of the skeleton. (For details, see 3.7.2.). Note: This component looks like a band with an opaque interior area. It can be used for the construction of simple mouth-shapes, pairs of eyelids etc.

Ring Outside Point Skeleton: An animated ring-like object consisting of a closed, opaque interior body polygon with a closed, opaque polygon skeleton outside. The skeleton has CPs. When the skeleton is animated the *vertices* of the inside polygon follow the *vertices* of the skeleton. (For details, see 3.7.2.) Note: This component looks like a band with an opaque interior area. It can be used for the construction of simple mouth-shapes, pairs of eyelids etc.

Ring Inside Edge Skeleton: An animated ring-like object consisting of a closed, opaque exterior body polygon with a closed, opaque polygon skeleton inside. The skeleton has CPs. When the skeleton is animated the *vertices* of the outside polygon follow the *edges* of the skeleton. (For details, see 3.7.2.) Note: This component looks like a band with an opaque interior area. It can be used for the construction of simple mouth-shapes, pairs of eyelids etc.

Ring Outside Edge Skeleton: An animated ring-like object consisting of a closed, opaque interior body polygon with a closed, opaque polygon skeleton outside. The skeleton has CPs. When the skeleton is animated the *vertices* of the inside polygon follow the *edges* of the skeleton. (For details, see 3.7.2.) Note: This component looks like a band with an opaque interior area. It can be used for the construction of simple mouth-shapes, pairs of eyelids etc.

3.7.5 Predefined facial elements

There are some (simple) components which can directly be used for the modeling of eyebrows, mouth and eye. More refined mouth-shapes etc. have to be build by the user from elementary components and skeleton components discussed above. In particular, the eyebrows and mouth shapes provided are point animated skeleton components. They are derived from Filled Body Point Skeleton resp. Ring Point Skeleton components. The user can derive analogous components from the EdgesAnimated components. Those allow for more subtle shapes when the user adds more FPs to the body polygon of the component. If so desired the user can make his own library of such components using the ***User Defined Composite*** mechanism discussed below.

!! Note: In the following discussion, ‘left’ means the left side of the face, which is at the right for the user.

Left Brow-1: A left eyebrow consisting of an interior skeleton with one CP and an enveloping polygon. While animating, the *vertices* of the polygon follow the *vertices* of the skeleton.

Left Brow-2: A left eyebrow consisting of an interior skeleton with three CPs and an enveloping polygon. While animating, the *vertices* of the polygon follow the *vertices* of the skeleton.

Right Brow-1: A right eyebrow consisting of an interior skeleton with one CP and an enveloping polygon. While animating, the *vertices* of the polygon follow the *vertices* of the skeleton.

Right Brow-2: A right eyebrow consisting of an interior skeleton with three CPs and an enveloping polygon. While animating, the *vertices* of the polygon follow the *vertices* of the skeleton.

Mouth-1: A mouth consisting of an outside skeleton with four CPs and another interior polygon. While animating, the *vertices* of the interior polygon follow the *vertices* of the skeleton.

Mouth-2: A mouth consisting of an outside skeleton with eight CPs and another interior polygon. While animating, the *vertices* of the interior polygon follow the *vertices* of the skeleton.

Mouth-3: A mouth consisting of an inside skeleton with four CPs and another exterior polygon. While animating, the *vertices* of the exterior polygon follow the *vertices* of the skeleton.

Eye: An eye with eyelids which can move up or down and a pupil which can move up, down, left and right.

In all those components animation parameters have meaningful names, like r_m_brow for the control point in the *m*iddle of the *r*ight eye*brow*.

3.7.6 User Defined Composite components

A user can define a library of components in addition to the ones provided by Face Editor. Such components are called ***UserDefinedComposite*** components.

A UserDefinedComposite component is produced by making a drawing of it in Face Editor and then saving it. This drawing consists of a collection of components generated in the usual ways (via *IncludeComp* or *IncludeScript*). These components will be the subcomponents of the UserDefinedComposite. When used in a drawing, UserDefinedComposites behave as any other component, they can be selected, edited, deleted etc. as *one single* unit. This distinguishes them from included drawings (see *IncludeScript*) which add a collection of separate components to the drawing.

There are two possibilities to create a UserDefinedComposite:

- Change the shape, color etc. of one already existing component, select the changed component and turn it into a UserDefinedComposite by menu item *SaveSelAsComp.*
- Make a drawing consisting of more than one component and turn the whole into *one single* UserDefinedComposite using menu item *SaveAllAsComp.*

When saving, the user is asked for a name for the component via a standard file-dialog.

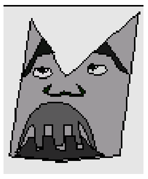
Including the UserDefinedComposite in a drawing is done by:

1) Select *IncludeComp* in Face Editor’s main menu.

2) Select the ‘UserDefinedComposite’ icon from the component selection menu.

3) The file-dialog which now appears shows the contents of subdirectory ‘UserDefinedComposite’ which contains the UserDefinedComposites constructed so far. Select the desired component.

The drawing order of the subcomponents of the included UserDefinedComposite is the usual one determined by the order of their inclusion and their rank as given during their construction (see 3.5 for drawing order and rank). This implies that UserDefinedComposites have an internal ranking of their subcomponents which takes its effect relative to its overall ranking.



3.8. Component editing

When components have been included in a drawing they still have to be tailored to their desired shape, color etc. This is done by editing them, using the main menu item `Edit`, which brings up a menu with edit operations applicable to the selected component. For every component type there is a set of allowed edit operations. This set is such that the restrictions on a component due to its type cannot be violated (e.g. a closed component remains closed, an animated one animated etc.). While editing a component, the user has access to an edit menu which contains only those edit operations which are permissible for that component.

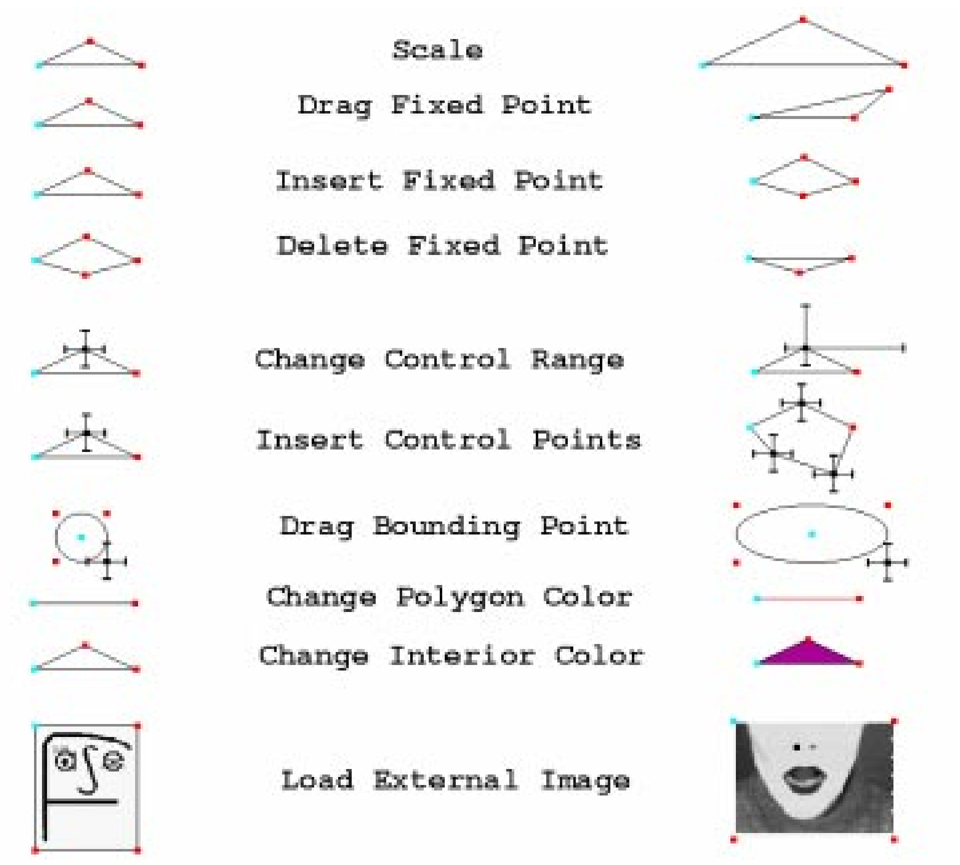


Figure 11. Example of the effect of some edit operations.

!! The edit operations which work on points (e.g. `DeleteControlPoints`) can be efficiently repeated: After their menu item has been chosen, the user can go on performing the operation on (other) points until another edit operation is chosen.

!! Some edit operations involve clicking on edges (e.g. `InsertFixedPoint`). This does not work when drawing with curves because then it is not seen where the edge is. In that case first switch to drawing components as polygons (see `Curve/Polygon Control`), perform the edit operation and finally switch back to drawing curves. Another trick is to insert points close to vertices and then drag them to the desired position. The user could also guess where the (straight) edge would go and click there to insert a point.

Below we discuss all possible edit operations in alphabetic order

AddControlPointsToEnd: `AddControlPointsToEnd` is used to add a CP to the end of the polyline of an open component. When this menu item has been chosen, a CP is created whenever the mouse button is clicked with the cursor on one of the end-points of the polyline of the component. When the button is held down and the mouse moved, the new point can be dragged to its desired position immediately. Thereafter the user can set the control range of the new point by dragging the endpoints of the arms of its hairline cross.

See also: *DeleteControlPoints*, *InsertFixedPoints*, *DeleteFixedPoints*, *AddFixedPointsToEnd*, *Insert-ControlPoints.*

AddFixedPointsToEnd: *AddFixedPointsToEnd* is used to add a FP to the end of the polyline of an open component. When this menu item has been chosen, a FP is created whenever the mouse button is clicked with the cursor on one of the end-points of the polyline of the component. When the button is held down and the mouse moved, the new point can be dragged to its desired position immediately.

See also: *DeleteControlPoints*, *InsertFixedPoints*, *DeleteFixedPoints*, *AddControlPointsToEnd*, *InsertControlPoints.*

CoupleControlPoints: *CoupleControlPoints* is used to assign leader/follower role to CPs. CPs can be coupled to each other by mouse clicks. The user can choose a CP in the selected component and couple it to another CP anywhere in the drawing. *CoupleControlPoints* provides the interface for this operation. It also gives possibilities to visually inspect how CPs are coupled. Independent CPs (shown in green) and CPs who are already leaders (shown in yellow) can be made followers of a (new) leader CP. If a leader is made follower of another leader the effect is that the first leader and its followers become followers of the new leader. Followers (shown in gray) can only be made independent again. All kinds of CPs can be chosen as a (new) leader: an independent CP, a CP which is a follower and a CP which already is a leader. When a CP which already is a leader is chosen, it just gets additional followers, when an independent CP is chosen, it is turned into a leader. When a follower is appointed as a leader, the effect is that the leader of that follower will become the leader of the CP which should follow the selected CP (which itself is a follower).

To make a CP a follower, first click on it (it will light up in dark blue). In the dialog which appears (with blue buttons), push *Make*. A new dialog with red buttons appears. Now select the candidate leader by clicking on a CP (it will light up in red). Another dialog will appear, this time with red buttons. When satisfied with the selection , push the red *Make* button.

To make a follower (gray) independent, click on it. (It will light up in dark blue.) In the dialog which pops up, push the *Make* button.

The dialogs discussed above also contain show buttons which give the possibility to find out what the followers of a leader are and which CP is the leader of a selected follower. By checking the appropriate show button, these leader or follower CPs will temporarily light up in a different color. So, when only inspecting how CPs are coupled, use these show buttons and thereafter the *Cancel* buttons of the dialogs instead of the *Make*.

!! Is is not possible to set the parameter name etc. of a follower CP; it inherits these properties from its leader.

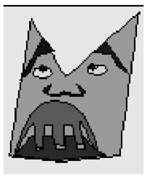
!! Is is not possible to delete a component containing leaders because this could result in followers outside the component ending up without a leader. If one wants to delete such a component, one first should make all the followers independent of its leaders.

!! Is is not possible to save a component containing followers or leaders coupled to CPs outside it as a Used-DefinedComposite component, because that would lead to followers without a leader or leaders without followers in this UserDefinedComposite.

Curve_Polygon Control: The *Curve_Polygon Control* item produces a dialog box by which the user specifies whether a (sub)component is drawn with a polygon or a (set of) spline(s) as outline.

First, there is a button which, when checked, leads to spline curves being drawn and when not checked, signals that the component is to be drawn as a polygon.

Next, there is a field in which the user can enter the distance (in pixels) between calculated points on the splines. This is useful because when the object is drawn with splines, CharToon actually draws straight lines connecting calculated points on those splines. So the distance field determines how smooth the drawn curves will look.



Finally, there is the *Set* button. Through this button CharToon enters an edit mode in which the user can, by clicking on polygon vertices, specify that connecting curves should brake on that point. (By clicking again, a smooth curve through the vertex is obtained again). When this option is not used, the result will be one single smooth curve through the objects vertices.

DeleteControlPoints: *DeleteControlPoints* is used to delete a CP somewhere on the contour of a polygon. After menu item *DeleteControlPoints* has been chosen, a CP of the polygon (polyline) being edited is deleted whenever the mouse button is clicked with the cursor on that point.

See also: *InsertControlPoints, InsertFixedPoints, DeleteFixedPoints, AddFixedPointsToEnd, AddControlPointsToEnd.*

DeleteFixedPoints: *DeleteFixedPoints* is used to delete a FP somewhere on the contour of a polygon. When this menu item has been chosen, a FP of the polygon (polyline) being edited is deleted whenever the mouse button is clicked with the cursor on that point.

See also: *InsertFixedPoints, InsertControlPoints, DeleteControlPoints, AddFixedPointsToEnd, AddControlPointsToEnd.*

DragControlPoints: *DragControlPoints* is used to drag a CP somewhere on the contour of a polygon to a different position. When this menu item has been chosen, a CP of the polygon (polyline) being edited is dragged whenever the mouse button is pressed with the cursor on that point after which the mouse is moved holding the button down. As an alternative, the user can move the control point by using the sliders which are found in the dialog window which pops up after *DragControlPoints* has been selected. Finally, it is possible to continue dragging (other) control points in this way, as long as no other item form the edit menu is selected.

See also: *DragFixedPoints, DragBoundingPoints.*

DragFixedPoints: *DragFixedPoints* is used to drag a FP somewhere on the contour of a polygon to a different position. When this menu item has been chosen, a FP of the polygon (polyline) being edited is dragged whenever the mouse button is pressed with the cursor on that point after which the mouse is moved holding the button down. As an alternative, it is possible to move the FP by using the sliders which are found in the dialog window which pops up after *DragFixedPoints* has been selected. Finally, one can continue dragging (other) FPs in this way, as long as no other item form the edit menu is selected.

See also: *DragControlPoints, DragBoundingPoints.*

DragBoundingPoints: *DragBoundingPoints* is used to drag a FP or CP of the defining bounding-box of an ellipse to a different position thereby changing the shape and size of the ellipse. When this menu item has been chosen, a corner point of the bounding box can be grabbed and dragged by the mouse.

See also: *DragFixedPoints, DragControlPoints.*

DynamicScaleSpecs: When components are included through *IncludeComp* with the *Animated Scale* button checked, they can be later dynamically scaled during animation. That means among other things, that when the '.aprof' file for a drawing which contains such a component is loaded in Animation Editor, there will be 'staves' to specify their time-varying x- and/or y- scale.

After *DynamicScaleSpecs* is selected a window pops up with:

1) Four sliders through which the x-scale upper limit, x-scale lower limit, y-scale upper limit and y-scale lower limit can be set.

2) A text field in which the user can assign a parameter name to the scale. This name will be displayed in Animation Editor adjacent to the particular scaling (in x and y direction) staves.

3) A button which, when pushed, produces a dialog box by which the user can set the Mpeg group attribute (see later) of the scale parameters.

EditSelectedSubComponent: When a subcomponent of a composite component has been selected, the user can start to edit this subcomponent by selecting *EditSelectedSubComponent*. (For an explanation, see 3.6.5.)

See also: *SelectSubComponent, StopEditing.*

FreezeFixedPoints: Through the *FreezeFixedPoint* item CharToon enters a state in which the user can, by clicking on the vertices of the body polygon of a skeleton component, specify that they must be frozen, i.e. that they cannot be dragged along by the skeleton. Through this device an object can for instance be created which only moves one of its sides. (By clicking again, the point is made mobile again.)

InsertControlPoints: *InsertControlPoints* is used to insert a CP somewhere on the contour of a polygon. When this menu item has been chosen, a CP is created whenever the mouse button is pressed with the cursor somewhere on the contour of the polygon (polyline) currently being edited. When the button is held down and the mouse moved, the new point can be dragged to its desired position immediately. Thereafter the user can set the control range of the new point by dragging the endpoints of the arms of its hairline cross.

See also: *DeleteControlPoints, InsertFixedPoints, DeleteFixedPoints, AddFixedPointsToEnd, AddControlPointsToEnd.*

InsertFixedPoints: *InsertFixedPoints* is used to insert a FP somewhere on the contour of a polygon. When this menu item has been chosen, a FP is created whenever the mouse button is pressed with the cursor somewhere on the contour of the polygon (polyline) currently being edited. When the button is held down and the mouse moved, the new point can be dragged to its desired position immediately.

See also: *DeleteFixedPoints, InsertControlPoints, DeleteControlPoints, AddFixedPointsToEnd, AddControlPointsToEnd.*

LabelControlPoints: A control point can be identified by a parameter name which is, among other things, gets transmitted to Animation Editor where it shows up adjacent to the ‘staff’ by which the control point’s time-varying value can be specified.

Initially, the parameter names have software generated rather obscure but unique values. Using *LabelControlPoints* the user can give them more meaningful values. After selecting *LabelControlPoints* he can click on control points in the drawing. Then a window pops up in which he finds a field in which he can enter a value for the name.

!! Although Face Editor accepts arbitrary long parameter names, the user is strongly advised to restrict herself to 12 characters because Animation Editor uses only the first 12 characters of such a name.

LoadImage: When an ExternalImage component is created, it displays a default image. The user can change this image by using *LoadImage*. When this menu item is selected, a file selection dialog will appear from which the user selects the name of the image to load by either selecting a file in the Files list or by explicitly entering a file name. The file should contain a ‘.gif’ image.

MpegGrpPERControlPoint: It is handy to be able to concentrate on groups of CPs when making animations in Animation Editor (see 6.4.1). The elements of a group can be shown/hidden together there, so even when there are dozens of CPs the user can (when groups are used properly) easily see all points pertinent to the nose, the mouth etc. In Face Editor the groups are to be chosen from the ones specified for faces in the Mpeg4 standard. For drawings with many CPs, it is advised to group CPs according to these Mpeg categories even if the drawing is not of a face.

When *MpegGrpPERControlPoints* is activated, the user can thereafter select a CP by clicking on it. When that is done, a window pops up, featuring a list of possible groups. The user assigns a group to the CP by double clicking on an item of this list and thereafter pressing the *OK* button of this window.

See also: *MpegGrpALLControlPoints, DynamicScaleSpecs.*

MpegGrpALLControlPoints: Grouping CPs can be useful as discussed above.

When *MpegALLControlPoints* is selected, a window pops up, featuring a list of possible groups. The user assigns a group to *all* CPs of this group, and recursively to those of its subcomponents, by double clicking on an item of this list and thereafter pressing the *OK* button of this window. The group attribute of dynamic scale and visibility parameters are also changed in the process.

See also: *MpegGrpPERControlPoints, DynamicScaleSpecs.*

Mirror vertical: When *Mirror vertical* is selected, the component is mirrored in the vertical direction about a horizontal line through its origin.

See also: *Mirror horizontal.*

Mirror horizontal: When *Mirror horizontal* is selected, the component is mirrored in the horizontal direction about a vertical line through its origin.

See also: *Mirror vertical.*

SetControlRange: Control points can move independently in the x- and y direction. The range in which their movement can be controlled by Animation Editor is indicated in a drawing by a hairline cross. (Ranges are not taken into account in Face Editor's test mode.) *SetControlRange* is used to change that range.

Changing the range involves the following steps:

1) Select *SetControlRange.*

2) Click on the CP whose range you want to change. A dialog box with four sliders, one for each arm of the cross, will pop up.

3) In the drawing, drag the endpoints of the cross to their desired length or use the sliders in the dialog box.

4) Click anywhere in the drawing to signal that the range is ok.

5) If desired, repeat steps 2 - 4 for other control points.

Extras:

1) It is possible to restrict the control point's movement to horizontal- or vertical motion only. To that end use the *Hor range allowed* resp. *Vert range allowed* buttons. When such a button is checked, the corresponding movement is allowed, otherwise it is not.

2) It is possible to force the CP to move in steps > 1. To that end the user should:

- Have the *Adapt movement to step* size button depressed.
- Enter values > 1 in the 'Hor step size' and/or 'Vert step size' fields.

It is also possible to have step sizes > 1 without having the *Adapt movement to step size* button depressed. This feature is only useful for the generation of Animation Editor profiles. Those profiles will have granularity > 1 but will allow parameter values in between those following from the granularity.

SetFillColor: *SetFillColor* displays a color selection tool through which the user can specify the color of the interior of the component which is currently being edited.

In the color selector a color is selected either by:

1) clicking in the upper left hand square to select a color.

2) clicking in the face drawing to select a color which has already been used (this may take long).

3) fine-tuning the selected color by clicking in one of the two color strips on the right hand side.

4) using the sliders on the lower left hand side of the color selector.

5) clicking in the horizontal gray strip to select a shade of gray.

The selected color appears in the lower right hand square of the color selector. The background is actually set to the selected color after pressing the *Apply* button. Terminate color selection by pressing the *Stop* button.

See also: *SetPolyColor.*

SetPolyColor: *SetPolyColor* displays a color selection tool through which the user can specify the color of the enveloping polygon or curve of the component which is currently being edited. This color selection tool is explained directly above.

See also: *SetFillColor.*

Scale: *Scale* is used to indicate that the selected component must be scaled to a different size. In order to scale: select *Scale* and press and move the mouse.

During scaling a cross appears at the origin of the component. This cross also represents the origin of the scaling transformation (Note: in this version of Face Editor it cannot be moved to another position. The effect of scaling with respect to another origin can be mimicked by a combination of scaling and drawing). Moving the mouse to the right or left enlarges resp. reduces the component in the horizontal direction. Moving the mouse up or down changes the vertical dimensions of the component.

SelectSubComponent: When the user wants to edit a subcomponent of a composite component, first he has to select that subcomponent by clicking on one of its FPs or CPs or by clicking inside it. *SelectSubComponent* switches Face Editor to a state where such a selection can take place. (For details on selection, see 3.6.5)

See also: *EditSelectedSubComponent.*

Show_HidePolyEdge: When the *Show_HidePolyEdge* edit operation is chosen, it is possible to enforce for every polygon edge or spline segment that it will or will not be displayed in Face Editor and in an animation. Note: When a component is selected in Face Editor, edges (spline segments) that should not be displayed are only highlighted a little; in FacePlayer they do not show up indeed. Switching between displaying or not displaying an edge is done by mouse-clicking on it.

StopEditing: The *StopEditing* button ends the editing of this (sub)component. When the user stops while editing a subcomponent, he ends up in the edit menu of the composite component containing this subcomponent, otherwise editing stops totally. (For explanation see 3.6.5.)

Visibility Control: A drawing component can be either:

1) permanently visible (the default case).

2) permanently invisible (probably desired for skeletons of skeleton animated components).

3) visibility controlled (i.e. being switched on or off) during animation.

In case 3, when the '.aprof' file for a drawing containing such a component is loaded into Animation Editor, a 'staff' appears on which it can be specified at which times the component is invisible. This staff, just as the others, is preceded by a parameter name which identifies it.

When *Visibility Control* is selected, a window pops up in which it is possible to specify the visibility desired by checking a button and which has a text field where the user can specify the parameter name shown in Animation Editor.

4. Face Player

4.1. Using Face Player

Face Player is used to show the effect of an animation on a drawing (of a face). It can be started in four different ways: integrated mode, stand alone mode, from Animation Editor and from Emotion Disc. Face Player can be used for two different purposes: playing movies and showing snapshots.

- While playing movies it displays animation frames at a give frame-rate. It does so by obtaining a set of animation parameters for every frame. This is what Face Player is used for in stand alone mode, but playing (pieces of) the animation can be done too when Face Player was started in integrated mode or from Animation Editor.
- Face Player has also special methods for accepting the animation parameters for one single frame and displaying that frame. This so called ***snapshot mode*** is only accessible through Java code and can be used when Face Player is started in integrated mode or from Animation Editor, or used from Emotion Disc.

While playing a movie, Face Player can also be instructed to dump every animation frame as a '.gif' file. In doing so, the user can thereafter use movie editors like SGI's Moviemaker or Adobe's Premiere to construct proper movies (e.g. QuickTime movies or movies in other formats) on the basis of these '.gif' images.

For details on these possibilities, see the discussion of Face Player in stand-alone mode (see 4.2.4).

4.2. Ways to start up Face Player

Face Player can be started in four different ways: from integrated mode, form Animation Editor, from Emotion Disc and stand-alone.

4.2.1 Integrated mode

In integrated mode, Face Player comes up together with Animation Editor and Face Editor. Integrated mode is obtained by starting Animation Editor with the '-i' option. In that case, the three components come up in a way in which they work together in a setup which is suited for the interwoven development of a face and an animation for it. In particular Face Player will show the face the user is currently developing in Face Editor and will play the animation which is being constructed in Animation Editor. For details see 2.1.

In integrated mode, Face Player comes up with four buttons: *Play* to start- or continue playing, *Step* to play the next frame when Face Player has not yet been started or is in pause-mode, *Pau* to stop Face Player while retaining its position in the movie being played and *Rew* to rewind the movie to its beginning.

4.2.2 Starting Face Player from Animation Editor

When Animation Editor operates in stand-alone mode and a profile has been loaded to which a valid (face) drawing corresponds, Face Player can be started in order to show animations for that drawing (for details see 6.2.2).

4.2.3 Starting Face Player from Emotion Disc

When Emotion Disc (see chapter 5) is started, a Face Player is automatically started showing the face which was specified on the command-line for the startup of Emotion Disc.

4.2.4 Stand alone mode

In stand alone mode, Face Player can show movies of an animated face. In order to start Face Player stand alone, go to the working directory and type

```
java FacePlayer -f facefile -s moviefile [-a soundfile] [-e enlargement]
[-d dumpfile] [-n] [-t]
```

Arguments:

`-f` **`facefile:`** This argument specifies the name of the face description, where **`facefile`** is a '.dat' file (without the '.dat' extension) to be specified relative to directory Faces. This argument must be present.

`-s` **`moviefile`**: This argument specifies the name of a movie, where **`moviefile`** is an '.amov' file (without the '.amov' extension) to be specified relative to directory Movies. This argument must be present.

`-a` **`soundfile:`** This option specifies the name of a file containing instructions for audio to be played in sync with the movie, where **`soundfile`** is a '.smov' file (without the '.smov' extension) to be specified relative to directory Movies.

Note: The '.smov' file itself contains the name(s) of actual audio files. For the time being, only Sun's '.au' files are accepted and hence the audio facility only works on Sun workstations.

`-e` **`enlargement:`** The `-e` option specifies that the frame has to be enlarged by factor **`enlarge-ment`** before displaying it. **`enlargement`** is a decimal fraction >= 0.

`-d` **`dumpfile`**: The `-d` option instructs Face Player to dump every animation frame separately as a'.gif' file. The prefix of the filenames is **`dumpfile.`** The images are dumped as **`dumpfile0001.gif`**, **`dumpfile0002.gif`** etc. They are placed in CharToon's subdirectory AnimDump. Note: **`dumpfile`** may also be a pathname relative to AnimDump.

`-n`: By default, Face Player tries to play animations in accordance with the prescribed frame-rate. Some-times it does not succeed e.g. when the drawing is too complex or the system load is too high. In that case, Face Player starts skipping frames in order to stay in sync. The `-n` option (No skip) inhibits frame skip-ping; all frames are shown but possibly at a slower rate.

`-t`: By default, all components of a drawing execute in the same thread. (This does not hold for Face Player's timer and audio-player: they always run in threads of their own.). When the -t option is given, every *foreground* component of a drawing executes in its own thread. This may be faster on parallel machines and in any case it is a prerequisite when an animation must obtain its animation parameters from different sources (e.g. in case of video-conferencing). In the last case, the code of Face Player should be extended.

Note: There is no option for the animation frame-rate. This is so, because a movie file itself tells Face Player at which rate it has to be played.

In stand-alone mode, Face Player comes with a *Start* button to begin playing, *Pause* to suspend playing, *Continu* to resume playing after a pause, *SingleStep* to show the next frame during pause or when not yet started and *Exit.*

When Face Player has played an animation to the end a message dialog will appear telling so. This message has to be acknowledged by pushing its *OK* button. While playing, Face Player will show the frame number of the frame currently being displayed.

5. Emotion Disc

5.1. Principle and usage of the Emotion Disc

Emotion Disc gives a very direct way to animate a face mde by Face Editor by selecting expressions from a pre-defined set of basic expressions or by automatically interpolating between those expressions. The user directly selects a complete facial expression (i.e. a coherent set of values for all animation parameters) in contrast to animating with Animation Editor where he must provide explicit values for the face’s animation parameters separately. On the other hand, this set of basic expressions has to be constructed once for every face and that is where Animation Editor has to be used.

Emotion Disc is based on the observation[1] that (as a first approximation) emotional expressions can be ordered in two dimensions: an intensity dimension and a dimension which orders the expressions according to which emotion they express. Furthermore, the ordering on the second dimension is ‘circular’, and the circular order of emotions is: **happiness-surprise-fear-sadness-anger-disgust-happiness**. The claim is that neighboring emotions correspond to expressions between which direct transitions (i.e. not going through a neutral intermediate state) are possible.

Because of this, emotions can be represented in a circle; the radial dimension represents intensity (with zero at the origin), the angular dimension represents the kind of emotion.

Emotion Disc is meant to display such a circle together with the corresponding face. When dragging the mouse through the emotion circle, the expression on the animated face follows the position of the cursor (see Plate 4.)

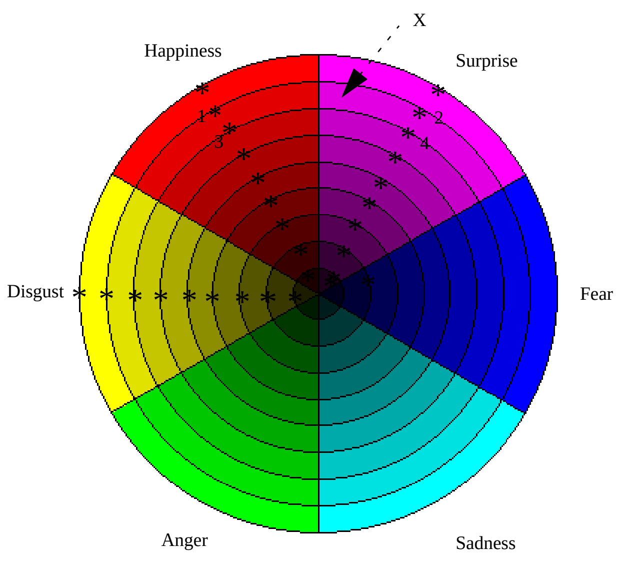


Figure 12. An annotated Emotion Disc example, showing position of expression samples and possible assignment of emotions. Expression sample positions are indicated by asterisks in the Surprise, Happiness and Disgust segments. There are analogous sample positions in the other sectors which are not shown.

1. H. Schlosberg: The description of facial expressions in terms of two dimensions, Journal of Experimental Psychology, Vol. 44. No. 4. 1952, pp 229-237.

As seen in Figure 12, the disc is divided in sectors and tracks. Sectors correspond to different emotions, tracks to intensity of emotion. The total number of regions equals the number of sectors multiplied by the number of tracks. In every region there is a special point which corresponds to a predefined expression (see Figure 12). Furthermore the neutral expression is associated with the centre of the disc. The expressions corresponding to all other points of the disc are found by locating the four special points surrounding the given point and doing a bi-linear interpolation (in polar coordinates). Example: the expression corresponding to position X in Figure 12. is calculated by interpolating between the parameters of sample points 1, 2, 3 and 4.

Emotion Disc configures itself for different faces, an adjustable nr. of tracks and an adjustable nr. of sectors. What the disc and face will look like is determined upon startup by the content of the disc description (‘.dsc’) file, which is produced by using Animation Editor and a conventional text editor. (For a detailed description of ‘.dsc’ files, see Appendix 1.)

5.2. Running Emotion Disc

Emotion Disc either just shows the disc and the face or it can be started in a version intended for special testing of the expressiveness of synthetic faces. In all cases, a usual profile and face file must be present under EmotionDisc/Profiles and EmotionDisc/Faces. These files (as in case of CharToon in general) must have identical names. Moreover, a disc file (with .dsc extension) must be present in the Discs subdirectory.

In order to start Emotion Disc, go to subdirectory Emotion Disc of the CharToon home directory and type:

`java EmotionDisc -b` **`disc-description-file`** `or`

`java EmotionDisc -s` **`disc-description-file stored-selection-file`** or

`java EmotionDisc -p` **`disc-description-file stored-selection-file`**.

With the `-b` (bare) option, the disc and the face come up without any control buttons. The disc just serves to select expressions of the face if one drags or clicks in it.

With the `-s` (selection) flag the disc and face come up. Underneath the disc there is a button *`OK, volgende foto`* (which means: ‘OK next picture’). When this button is pushed, the selected expression (actually just the current mouse coordinates) is stored in **`stored-selection-file`**. (Showing the photos is out of the range of Emotion Disc, can be done manually.)

With the `-p` (play) flag, the disc is not shown, but a window comes up containing button *`OK, volgende expressie`* (which means: ‘OK, next expression’). By repeatedly pushing this button, the expressions stored in **`stored-selection`** are shown one after the other.

In both cases, **`disc-description-file`** is the ‘.dsc’ file (without the ‘.dsc’ extension being typed) discussed above.

6. Animation Editor

6.1. Basics

6.1.1 The principle of defining animations

An animation is defined by a number of time-varying ***animation parameters (APs***). Defining an animation means to define the values of some (possibly all) animation parameters through time. The AP values are given by specifying the value at certain times - these are the ***knot points (KPs)*** of the AP. KPs are linearly interpolated to compute the value of an AP in-between. If an AP has no value specified at a given time, then a specific, so-called neutral value is assumed.

The animation profile, or shortly, ***profile*** defines the set of animation parameters which can be used to animate a face. The range and granularity of the allowed values as well as the name and eventual explanation for each animation parameter are specified in the profile file (see Appendix 1.1 for details). Usually a profile file is generated automatically by Face Editor. However, a profile file according to the required syntax can be produced by a text editor too, and not only for CharToon faces.

In order to use the animation editor, it is sufficient to know that:

- an AP has a numerical ***identifier*** (***ID)*** for identification and a name for ease of use,
- an AP can take either float or integer values between a ***minimum*** and a ***maximum*** value,
- there is a ***neutral value*** which usually corresponds to the parameter value in the neutral face,
- the set of allowed values can be either all values within the range or the interval is divided into equal steps, and the value must be one of the resulting discrete values,
- an AP belongs to a ***group*** (such as eye, mouth etc., see Appendix 1.1 for the complete list).

An animation may have max. 100 APs and 1000 KPs per AP, and be max. 3 minutes long (the latter can be redefined by setting options, see 6.8.2).

Normally, there is a single face with a given profile. However, it is sometimes useful to have many versions of a face with identical profile, or to make different profiles for a single face.

Several profiles for a single face

This is useful while experimenting with different ways to animate a single face. Namely, driving a given face with subsets of all the parameters defined in the face, or with the same parameters but (some of them) with different ranges. In such cases, the user should open the single face in Face Editor, modify it by deleting KPs and/or changing ranges, and save only the profile in Face Editor (see 3.6.1), but not the modified face. Then, by using a text editor, he should add to the first line of this new profile the face name of the old face. Face Player will be able to show an animation with the modified profiles on the unchanged face.

Several faces for a single profile

This is useful while experimenting with various versions of a face which are derived from the original one without having touched the KPs and other control parameters (scale, visibility). Particularly, identical faces with different rendering/background are such variants. In such cases, the user should open the single face in Face Editor, modify it and save it under a new name. Then with a text editor a new version of the original profile file is to be made, which differs only in the face name from the original. Then this profile file can be used to animate the modified face with animations made for the original face.

6.1.2 Starting up Animation Editor in stand alone mode

Animation Editor can be started in a stand-alone way by the following command:

```
java AnimEdit [-f profilefile] [-w width] [-h height] [-l userlevel]
```

where **`profilefile`** is the path name of the profile file (without '.aprof' extension) to be loaded, **`width`** and **`height`** are integers specifying the size of the Animation Editor window in pixels, and **`userlevel`** is the user level (see 2.3). No Face Player is started up automatically.

Animation Editor can be quit by the *File/Quit* menu item.

Animation Editor can be started up together with Face Editor in the so-called integrated mode too (see 2.1).

6.1.3 The Animation Editor window

The main window of Animation Editor is shown in Figure 13.

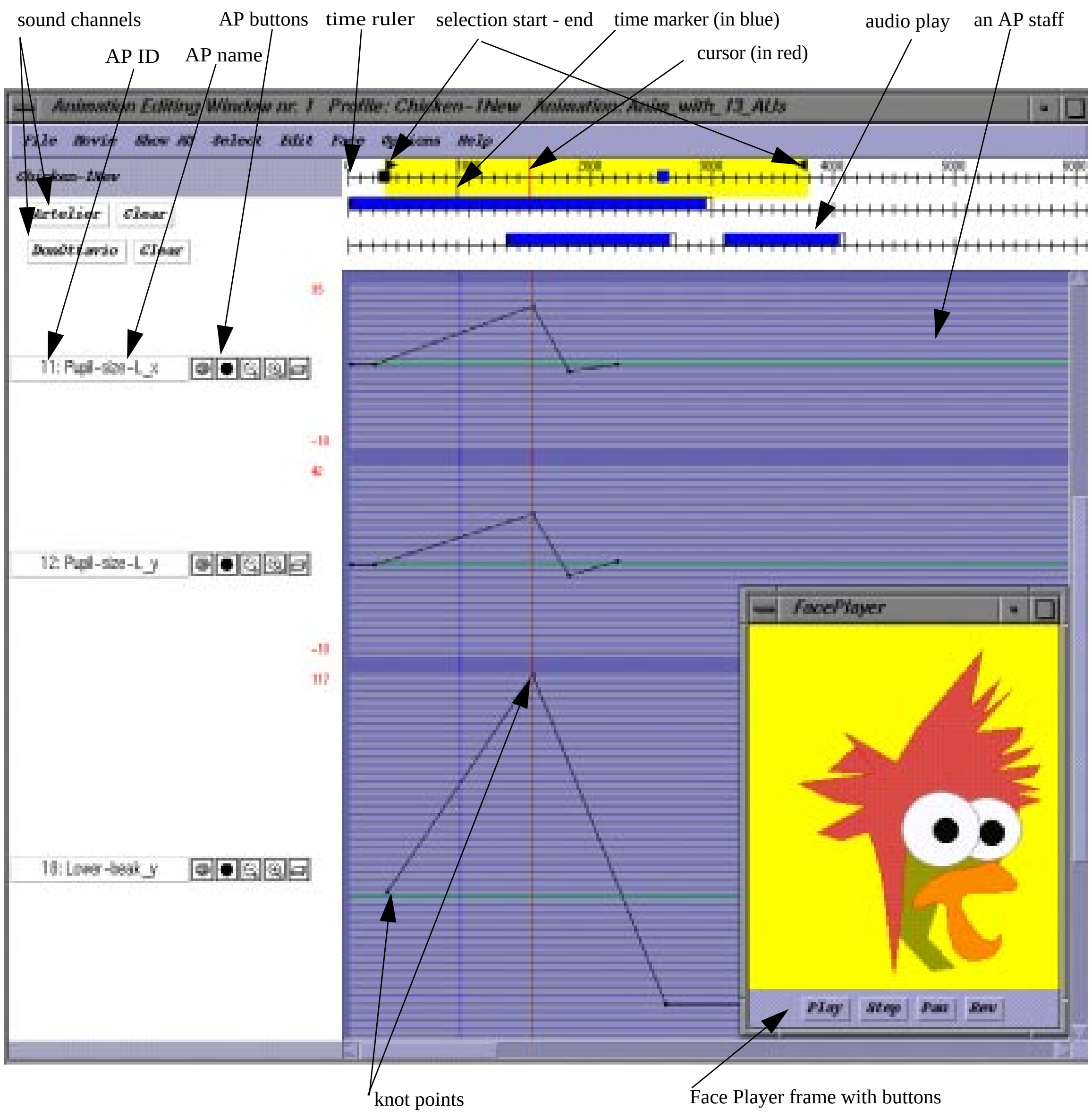


Figure 13. Snapshot of and Animation Editor window.

The Animation Editor window contains the following regions:

1. time ruler;
2. sound buttons and editing panel for each sound channel, if a sound profile with some channels has been loaded (either at start up if a Sounds/DefaultSoundProf.asprof was present, or later, by using the *`Sounds/ Load sound profile`* menu item);
3. AP buttons and staff for each parameter channel.

On the top, a menu bar can be seen with different pull-down menus. The window contains a title bar with the name of the profile, the face and the animation being used.

The *`Help`* menu offers the *`Show help topics`* menu item, by which the user can get on-line help on a list of topics. The on-line help for each topic is an abbreviated version of the corresponding manual pages.

6.2. Getting started to make an animation

6.2.1 Loading a profile

The user can load a profile by the *`File/Load profile`* menu item. The ascii file to be selected should contain a syntactically correct animation profile definition (see Appendix 1.1 for details).

When Animation Editor is started with the `-f` **`profilefile`** option, the given profile file is loaded, otherwise the profile in the ‘Profiles/DefaultProfile.aprof’ file is loaded. If at startup the loading of a profile file fails or Animation Editor has been started up in integrated mode without the `-f` option, an empty profile is generated. A special profile file uniqueidCurrent.aprof is used in integrated mode, containing the latest profile generated from Face Editor by the *`SetCurrent`* button (see 3.6.1).

When a profile is loaded, a new (empty) animation is generated automatically, which can be 'filled' by interactive editing, or by loading a previously saved animation.

6.2.2 Seeing the animated face

In principle it is possible to compose an animation just working with Animation Editor, and play the animation later via movie script files. However, usually the user would like to see snapshots of the animated face and try-out pieces of the animation being edited. To do so, he should start up Face Player by the *`Face/Show Face Player`* menu item. Face Player will look for a face file under Faces at the relative path of the current profile (under the directory Profiles), and the name specified by the profile (see Appendix 1.1 for more).

For completeness we mention that a ‘realistic’ 3D face model used at CWI (called Persona) can also be animated with Animation Editor, by the *`Face/Persona`* menu item. Persona should be started up first, maybe on another machine, as long as the machine is specified through the *`Options/Set parameters`* menu item. For socket communication, Animation Editor is using port 12345. Note that Persona is slow, so wait for the effects to be shown.

When inserting or dragging a KP, in both cases the face will show the expression according to the current time, taking into account all the APs (also the non-visible ones, see 6.4.1). The user can test an expression at a given time by moving the cursor to the time ruler and making a snapshot with Mouse Right Press.

A piece of the animation can be played by pushing the control buttons of Face Player. The animation can be played, paused, rewound and single frame steps can be made. Ways to select a part of the animation to be played is discussed in 6.6.1.

!! Note that the playing speed may be slower than the intended one, especially if the face to be animated is big and complex, and the machine is overloaded. Improve playing speed by taking a small size for the Face Player window before it is started up. (This is possible only in non-integrated mode, see 6.9.2.)

6.2.3 Using the animation parameter staves

In the Animation Editor window there are parallel horizontal regions, one for each AP. The user can find and look at specific AP staves by using the scroll bar on the right side.

!! If there are many AP staves, scrolling and refreshing the window may take long. The best way of scrolling is to find the desired location roughly, by clicking below/above the vertical bar indicating the currently visible portion, and then to locate the visible part precisely by dragging the ‘visible portion’ bar or clicking on the up/down arrows in the scroll bar.

The order of displaying of APs is fixed, reflecting the order of APs in the profile file, and cannot be influenced. But it is possible to restrict the shown APs to certain subsets by the *Show AP* menu, see 6.4.1.

On the very left, in the AP text button, the ID and the first characters of its name (altogether max. 15 characters) are shown. By pushing this button details on the AP are shown in an information window.

!! Note that APs are identified by the IDs, which are automatically generated by Face Editor. The ‘one_word’ AP names (given by the user in Face Editor) are useful to indicate the function of the AP. If the AP name has not been specified, the default name generated by Face Editor will be shown. It is advised to set the AP names in Face Editor for a face before starting to make animations. It is highly recommended to use short (max 12 character) labels which follow some principle, e.g. using _l_ tag for “left”, “_t_” for top, “_o_” for outer, etc.

Next to the AP text button there are 5 view buttons (from left to right) with the following functionality:

focus:	add the AP to /remove the AP from the ones in focus (collective editing operations effect only the APs in focus, see 6.4.2),
hide:	hide the AP (hidden ones can be seen again via the *Show AP* menu item),
zoom out:	shrink the staff height for the AP,
zoom in:	magnify the staff height for the AP,
erase:	erase everything from the AP staff.

The hide function is useful to concentrate on certain APs, and even to be able to save some channels only (namely the shown ones) with a special menu item (see 6.3.1). Focus plays a role in selecting and editing multiple APs.

The ***range*** of each AP is indicated in red if ***discrete values*** are allowed only, in blue if ***any value*** within the range is allowed. Note that the ***minimum value*** is at the bottom of the staff and the ***maximum value*** is at the top for each AP.

!! This is opposite to the direction of the co-ordinate system in Face Editor, so e.g. you have to increase the upper eyelid position value to get the upper lid closed!

For each AP, to the right of the name of the AP and the buttons, a 'composition area', the animation parameter ***staff*** is presented. This area consists of parallel lines, one of them in green. These lines aid the user in specifying the value of the animation parameter at a given time, by setting/editing knot points. The green line corresponds to the ***neutral value*** for the animation parameter. In the smallest (most zoomed out) size there are two parallel lines, all KPs are placed between those, and thus only the time of KPs is indicated in this view.

6.2.4 Creating a new animation

The user specifies values for an animation parameter by inserting KPs. To make a new animation from scratch, click at different time locations in an AP staff to insert KPs. The value at in-between times is indicated by the line connecting the KPs. You may test the animation you have just made by using Face Player, see 6.2.2. You can alter your current animation by the local or global editing operations described in 6.5.

You may edit at any level of zooming. Mind that editing a ’zoomed out’ presentation may offer you rougher granularity than the one defined for the AP. So for refined editing, use the maximum size (zoom in). The rough presentations (zoom out) can be used for global editing operations (cut&paste, shift, scale) or as indications for the AP being edited.

6.3. Saving and loading animations

6.3.1 Saving an animation

The basic way to save the current animation is to save it as an ascii file:

`File/Save animation` saves the entire animation into an ascii file.

For more sophisticated usage, there are further possibilities:

`File/Save.../Save shown`	saves the *shown* APs into an ascii file,
`File/Save.../Save as object`	saves the *entire* animation as *Java object*, (Not recommended unless for good reason by a GURU user.)
`File/Save.../Save complete`	saves the profile *and* the entire animation into an ascii file.

`Save shown` is to be used to store animations for specific regions for the face (e.g. a blink), by showing first only the appropriate APs. `File/Save.../Save complete` is to be used to archive an animation with the corresponding profile. This option is recommended if the face and thus the profile is still being changed, but the user wishes to re-use the animation later for the modified face (or a different face) by ’reanimate’ (see 6.3.3). Complete animations are recommended to be stored in the CompleteAnimations directory, and they get .all extension.

In order to maintain profile and animation correspondence, it is recommended to use some file name convention in the Animations directory (e.g. Profname_Aname_Version) and to create specific subdirectories.

We mention here that one can get a ‘printed staves’ of the animation, which may be useful for analysis/presentation purposes. This is to be done by the menu item:

`File/Print`	prints the shown APs on paper, according to format specification given in the Print Dialog by the user,

6.3.2 Loading an animation

An animation *of the same profile as the current* profile can be loaded by:

`File/Load animation`	load an animation from an ascii file,
`File/Load.../Load from object`	load an animation from a Java object file.

Both a (new) profile and an animation (saved earlier with `Save complete`) can be loaded by:

`File/Load.../Load complete`	load a profile and an animation from an ascii file.

In this case from an ‘.all’ file both the profile and the animation are loaded, thus the problem of not knowing (or even, having stored) the profile of a particular animation can be avoided.

When loading an animation, APs are identified by IDs. KP values eventually out of the allowed range (as defined for the AP in the profile) are replaced by allowed min-max value for the AP in question.

6.3.3 Re-using earlier made animations by reanimate

An animation with a different profile than the current profile can be ‘reanimated’. The user is asked for the ‘.all’ file containing the animation to be reanimated, that is, to be re-used with the current profile. The user is then asked to decide if reanimation should look for identical AP names or for identical IDs. If a matching pair is found in the old and current profile, then the AP is bilinearly rescaled, by assigning the neutral, min and max value of the old profile to the ones in the current profile. APs of the current profile for which no matching old AP was found, remain empty. This service can be well used if AP names are generated in a consequent way (e.g. having the 'Mouth_R' name for the mouth right corner parameters) and in case of incremental extension/modification of faces (e.g. changing ranges, adding new APs).

Reanimation can be activated by:

File/Reanimate

Basically such a reanimation takes place when Animation Editor is used in integrated mode, and the face has been changed and then from Face Editor the *SetCurrent* menu is activated. Then the current animation is reanimated with respect to the changed profile, see 3.6.1. Again, the user may choose the way to find matching APs.

6.3.4 Saving movie data

In order to make a movie to be played later by Face Player, an '.amov' ascii movie file containing AP values for each frame (see Appendix 1.4) has to be produced. To generate such a movie file, invoke one of the menu items:

Movie/Full animation	generates the specified movie file of the entire animation.
Movie/Selection	writes the selection as movie into a movie file.

The user has to specify the sampling rate and number of digits used for parameter values, and give the name of the output '.amov' file.

!! Use 2-3 digits (the default is 3), especially if the drawing to be animated is small, to avoid jerky animation due to rounding errors of the parameters.

The user can get a snapshot (one frame) of all AP values at a given time by positioning the mouse on the required time in the time ruler, and clicking:

Mouse right press	which updates the face if shown and writes the snapshot to the temporary 'uniqueidSnapshotTest.amov' movie file.

The produced '.amov' movie files are ascii files, containing parameter values for the APs for each frame. (see Appendix 1.4). Face Player and Persona can play such a movie file. Note that if the user wants to produce a video movie file (to be played by a media sw or video), the frames produced by Face Player (or Persona) have to be dumped and a movie has to be made from the still images by using some sw.

There is no possibility to make a movie file of shown APs only. However, this can be achieved by making a movie of part of an animation which was saved with *File/Save.../Save shown.* Another solution is to select the APs of interest, and make a movie of the selection.

6.4. Changing the view, visibility and focus of APs

The full size of all AP staves may be large, resulting in the unpleasant situation that only a few APs can be seen at a time in the window, and it is difficult to find the APs one is really interested in at a given moment. There are different features to be used together to remedy such a situation:

- Change the ***view*** (size and granularity) of AP staves, by zooming in/out for each AP staff from an overview via small and medium view to the full size view by the zoom buttons aligned with the AP name.
- ***Hide*** APs which are not used at all in the given animation, or are not relevant in a current editing situation (e.g. for lip-sync the APs for the upper face should be hidden). Animation Editor provides means to ***show*** earlier hidden APs again.
- Bring APs into ***focus***, in order to restrict the effect of collective editing operations or testing the animation to these APs only.

6.4.1 Views and zooming

The user can zoom in/out for each AP staff from an ***overview*** via ***small*** and ***medium*** view to the ***full*** size view and the other way around by the zoom buttons (third and fourth to the right of the AP name), see Figure 14. In the full view, a number of equidistant parallel lines are shown in the staff. The parameter interval between the minimum and maximum value is divided into equal intervals, as many as the ‘granularity’ number given for the AP in the profile file (see Appendix 1.1), or 10 as default if no granularity is specified. Depending on the type of the AP, also given in the profile file (see Appendix 1.1), only values corresponding to the parallel lines are allowed, or other values between them too. In the latter case, the parallel lines provide only visual aid to define values. In the first case, if the user clicks between lines, the nearest line will give the intended value.

!! If the difference is big between the minimum and maximum values, define a larger granularity value, otherwise the staff for the AU will be too big. This can be given in Face Editor when setting a control point’s range, see 3.8.

In medium and small views, the number of lines is the same, but the distance between them is smaller.

In overview view, there are only two lines shown in the AP staff, and the KPs are placed aligned between the two lines. Hence in this view only the time of the KPs can be seen.

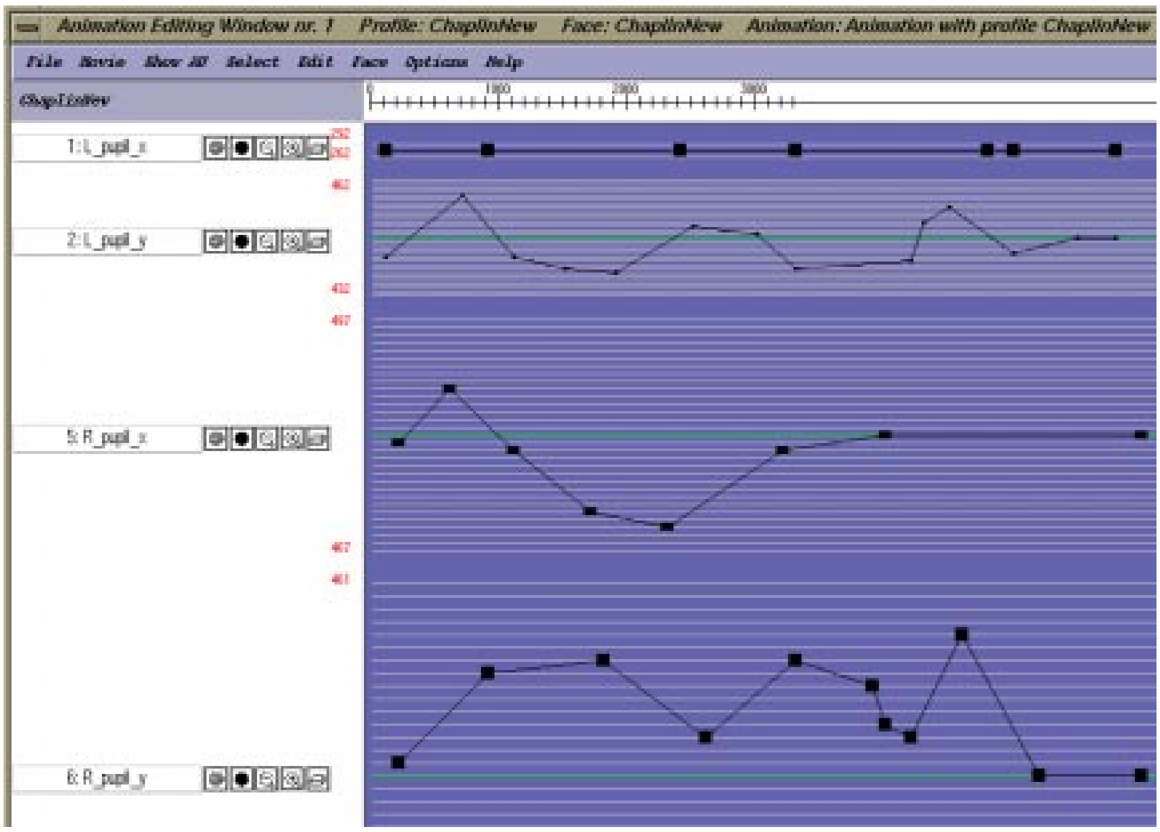


Figure 14. Action parameter staves with overview, small, medium and full view.

For ‘detailed editing’, full view should be used, while for rough editing the medium view is sufficient. The overview view can be handy for editing operations effecting a selection. It is also useful to show APs which are taken into account (e.g. for synchronization) when editing a particular AP in full view. If all the APs are shown in overview, the user can get an idea about the entire animation at a glance, without knowing the details in each AP staff.

6.4.2 Hide/show APs

By the hide button (second button to the right of the name of the AP) the entire row for the AP gets hidden. It is useful to have only the current few APs shown one is working on. Hidden staves are not shown for editing, but are taken into account when a snapshot is made. When playing the animation, all the APs are taken into account, as long as no AP is in focus. (If there are APs in focus, testing the animation is restricted to those.) A hidden AP remains in focus, if it was in focus when the hide button was pressed.

All but one of the *Show AP* menu items are to specify which APs are to be displayed. Besides one by one selection and selecting all APs, it is possible to restrict the APs to be shown by groups. Namely, each AP belongs to a group, as given in the profile file (see Appendix 1.1). If the group assignment was done properly, APs corresponding to a facial feature can be shown/hidden by selecting/deselecting their group. The corresponding menu items are:

Show AP/Hide all	all APs are hidden (handy if the user wants to see only a few of them next),
Show AP/Show all	all APs are shown,
Show AP/Show used	all used APs - that is non-empty ones - are shown,
Show AP/Show some	the user can select one by one hidden APs to be shown,
Show AP/Show groups	a list is presented with the groups of APs to be selected from.

One menu item of the *Show AP* menu serves to turn all the shown APs into overview view:

Show AP/Overview	all shown APs are to be displayed in overview view.

6.4.3 Focusing and selecting

The user can select a time slice of certain APs, and test or edit only those. The APs to be considered are identified by ‘bringing them into focus’. The user can put in/remove from focus APs one by one by the focus button for the AP. The focus button (first one to the right of the AP name) is shown in red if ‘checked’, that is if the AP is in focus. The following menu items with shortcuts are useful to speed up focusing and selection:

Select/Focus none	CTRL+F	removes all animation parameters from focus;
Select/Focus all used	CTRL+SHIFT+D	brings all used (non-empty) APs in focus,
Select/Focus all shown	CTRL+SHIFT+N	brings all shown APs in focus,
Select/Select all in focus	CTRL+SHIFT+S	selects the entire lengths of the APs in focus,
Select/Select range in focus	CTRL+SHIFT+M	selects the time interval of the APs in focus-between the start and end marker (shown in the ruler),
Select/Clear selection range	CTRL+SHIFT+C	clear selection range (nothing is selected).

For more information on selecting, see 6.6.2.

6.5. Editing an animation

In this chapter editing operations will be discussed, first those affecting a single KP, then those referring to KPs in a single AP, finally the ones affecting focused APs. It is important to remember all the time that editing operations effect KPs. Hence there is no way to change a part of a time curve which does not contain at least one KP. On the other hand, the change of a single KP does change the time curve between the preceding and the following KPs.

In all cases, the latest editing operation can be undone by:

Edit/Undo	CTRL+Z	undoes the latest editing operation, including the 'clear all' action.

When editing an animation with several APs, it is very much recommended to use the zoom, hide/show and focus facilities. It is possible to perform all the editing operations in all zoom views. Dragging/inserting KPs takes place on the level of granularity allowed by the view. If a piece is copied and pasted, even if done in a rough view (e.g. overview), the original values are preserved. Refined editing of a single AP should be done in full view, medium and small view could be used to sketch the animation. Overview should be used, first of all, as reference when editing another AP. Overview is also handy when several APs are copied/pasted/transformed.

6.5.1 Adding/deleting/moving KPs

A single animation parameter can be edited while the mouse is over the corresponding animation parameter staff, by the following mouse actions:

Mouse Press	insert a KP, if there was none; otherwise grab the one already at the location,
Mouse Press + SHIFT	delete the KP at the given location (if there is one there),
Mouse Press + CTRL	start selecting,
Mouse Press + ALT	paste earlier copied single AP part at current time,
Mouse Press + CTRL + SHIFT	select/deselect the entire animation parameter,
Mouse Drag	drag grabbed knot point (if one was grabbed),
Mouse Drag + CTRL	select part of the animation parameter (selected part is shown in red).

The AP staff is automatically scrolled when dragging a point or indicating a selection area with Mouse Drag + CTRL. Without mouse actions, the animation parameter staves can be scrolled by the scroll bars at the bottom and on the right.

To clear an AP, use the button to the left of the animation parameter staff.

To clear the entire animation, use the *`Select/Focus all used`*, *`Select/Select all in focus`* and *`Select/Clear selection range`* menu items.

6.5.2 Cut and paste within and among APs

The user can cut and copy a time slice of a single AP or of the APs in focus. The single paste operation takes place if the cursor is in one of the AP staves, the multiple paste, referring to APs in focus, takes place if the cursor is in the time ruler. For cut and copy, the latest selection is the target. There is a separate buffer to store cut/copied single and multiple AP pieces, to be used when pasting in the single AP or the multiple AP case. A copied or cut selection can be pasted. In the multiple APs case, the number of animation parameters currently in focus should be the same as the number of copied animation parameters. If so, the animation parameters are inserted ’parallel’, the first of the copied ones into the first of the ones in focus, etc. The time of the insertion will be the time corresponding to the position of the cursor.

If there is a KP at the time of pasting, the value of the KP must be the same as that of the (identical) value of the first and last KP to be pasted. This condition is checked for all of the APs to be pasted.

The copy/cut/paste actions can be invoked by the following menu items and shortcuts:

`Edit/Cut`	CTRL+X	cuts the selection;
`Edit/Copy`	CTRL+C	copies the selection;
`Edit/Paste`	CTRL+P	paste the latest copied AP(s) at the current time.

When inserting one animation parameter into another one with a different animation parameter profile, a two-piece linear scaling is performed, making sure that the neutral, minimum and maximum values correspond to each other. If a ’bi-directional’ animation parameter is inserted into a uni-directional one, one half will be chopped (set to the neutral value).

A selection can be flipped, that is ’mirrored’ on the neutral value (green line). The *`Flip`* editing operation performs a mirroring and a bi-linear re-scale, making sure that the flipped animation remains within range. The user can paste the flipped version of the latest copied animation by the *`Paste with flip`* editing operation.

The flip operations can be invoked by the following menu items and shortcuts:

Edit/Flip	CTRL+W	‘mirrors’ the selection on the line of the neutral value, with re-scaling upper and lower parts,
Edit/Paste with flip	CTRL+Q	copies the flipped version of the latest copied AP(s).

6.5.3 Shifting and scaling selections

There are transformations which are defined on a selection. A selection is either a continuous interval of one animation parameter (to be chosen as explained above), or a continuous time interval of the animation parameters in focus (see 6.6.2).

The operations scale time, scale value, shift time, shift value are to be performed after a selection has been made. Note that the current selection is either one produced by selecting a time interval of the animation parameters in focus, or the last single animation parameter selection, whichever happened the latest.

The scaling and shifting transformations can be performed only if the 'natural' constraints will not be violated by performing the transformation, namely:

- knot points do not collide (along time),
- knot point values remain within allowed range,
- the length of the animation remains within maximum (for time scale up).

The corresponding menu items, with key shortcuts, are elements of the *Edit* menu. The scaling operations use default or the latest user given scaling values (to be changed by *Options/Set parameters*).

Edit/Scale time down	CTRL+T	scales the selection‘s duration by a factor 0.8 (default), the first selected KP remains fixed,
Edit/Scale time up	CTRL+T+SHIFT	scales the selection’s duration by a factor 2.0 (default), the first selected KP remains fixed,
Edit/Scale value down	CTRL+V	scales the selection’s value by a factor 0.5 (default), relative to neutral value,
Edit/Scale value up	CTRL+V+SHIFT	scales the selection’s value by a factor 1.25 (default), relative to neutral value,
Edit/Move value up	CTRL+U	moves the KPs in the selection up with one step,
Edit/Move value down	CTRL+D	moves the KPs in the selection down with one step,
Edit/Move selection left	CTRL+L	moves the selection left with one time unit,
Edit/Move selection right	CTRL+R	moves the selection right with one time unit.

The above transformations are not carried out if they would violate the above described constraints.

6.6. Using the time ruler

The ruler is at the top of an Animation Editor window, showing time in msecs, in 100 mseconds units. These discrete units are 'enforced' by the editor, with the result that the control points all have n*100 mseconds as time. The maximum time (the maximum length of an animation) is 180000 mseconds (can be changed by setting parameters of Animation Editor, see 6.8.2). The time corresponding to the location of the cursor in one of the animation parameter staves or in the time ruler is highlighted in red.

6.6.1 Ruler markers for playing and synchronization

In the ruler there can be markers in black and in blue (see Figure 15, Plate 5). The ***begin*** and ***end markers*** in black indicate the begin and end of the selection or time slice of the animation being tested. That is, the portion of the animation between the begin and end markers will be played by Face Player. The begin and end markers are updated automatically to the begin-end of the animation whenever a KP is inserted before/after the beginning or end of the current animation. The user may specify his own ‘interval of interest’ in one of the following ways:

- drags the start or the end marker to a new position,
- specifies a selection with CTRL+Mouse Drag in the ruler region.

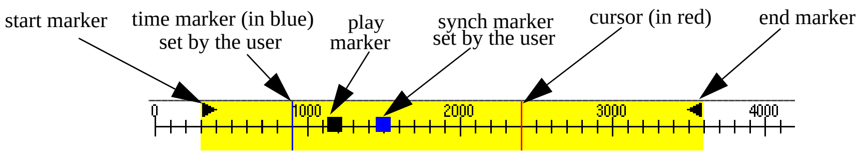

start marker
time marker (in blue) set by the user
play marker
synch marker set by the user
cursor (in red)
end marker
0
1000
2000
3000
4000

Figure 15. Markers in the time ruler.

As soon as the user has specified a selection range explicitly (see 6.6.2), it will not be updated automatically. The latest user selected region of the time ruler is always highlighted in yellow. This can be cleared by:

`Select/Clear selection range`

When an animation is played by Face Player, the cursor indicates the time of the frame being shown, and in the ruler the ***play marker*** - a black square - is moving accordingly. If the playing is stopped by the Face Player `Pau` button or because of reaching the end marker, then the play marker remains, to show where the playing was suspended.

Synch markers - blue squares - can be added to the time ruler to remember special moments in time, e.g. for synchronization. The following mouse operations are to be used:

Mouse Press	add sync marker,
Mouse Press + SHIFT	delete sync marker.

!! Note that by pushing the mouse at the same location in the ruler more than once, several ruler makers are placed ‘on top of each other’, and all of them can be removed by an equal number of Mouse Press + SHIFT-s at the location. However, do not insert multiple markers at the same location.

6.6.2 Selecting APs in focus

To be able to select parts of more than one animation parameter, the user can identify the ones of 'current interest' by bringing those into focus (see 6.4.2). Then he can select a time interval of the APs in focus by the following mouse operations in the time ruler:

Mouse Press + CTRL	start selecting animation parameters in focus,
Mouse Drag + CTRL	mark selection for all animation parameters in focus.

The selection is highlighted in yellow in each affected AP, the selected KPs and connecting curves are turned red.

It is possible to mark a special time by setting a single ***time marker***, and use it as reference for editing or for selection. The time marker is a line in blue drawn through all the AP staves. The relevant mouse operations are:

Mouse Press + ALT	set time marker,
Mouse Press + ALT +SHIFT	remove time marker, if there was one at the time of mouse press,
Mouse Press + CTRL+ALT	select time slice between set time marker and the current time, and re-set vertical time marker to current time.

The time marker is handy when one wants to select long time intervals.

6.7. Working with multiple animations

Once Animation Editor has been started, it is possible to open further Animation Editor windows. This is useful if the user wishes to copy into the current animation pieces from earlier made animations. Another usage is when he wants to compare pieces of animations (usually, made for the same face).

Open a new frame and close one by the following menu items:

File/Open new window	opens a new window, reading in the DefaultProfile.aprof profile,
Edit/Close this window	closes the current window.

If by closing windows the number of windows is decreased to 0, then Animation Editor is quit automatically.

The different windows behave as ‘independent’ Animation Editor windows. When a new window is opened, it corresponds to starting up Animation Editor in stand alone mode, but with the same other options as Animation Editor was started up initially. The windows can be identified by the window number shown in the window title. The user may start up Face Player for each window.

6.8. Using sound

6.8.1 Adding sound

It is possible to prescribe multi-channel sound to accompany the animation and give play instructions. The number of channels, audio files containing the sound for each channel and further information on each channel should be given in a ‘.asprof’ sound profile file (see Appendix 1.2). For each sound channel, ***sound labels*** can be included in the ‘.asprof’ file, indicating e.g. vowels pronounced at certain time moments, or a sudden noise. The sound labels are shown at the proper times in the sound play channels, and are meant to be used for synchronization purposes while editing an animation. For defining the labels, see Appendix 1.2. A sound profile file has to be prepared by hand, with a text editor, on the basis of e.g. the ascii file with markers produced by the SGI soundtrack sw.

Once sound channels are present, the user can indicate by interactive editing (explained 6.8.2) when to ‘play’, ‘suspend’ and ‘restart’ the corresponding audio (see Figure 16). Channels with such play instructions define the accompanying audio for the animation. All the channels with play instructions are stored/loaded as Java objects.

The user can load/save a sound profile (containing the reference to a sound file and play instructions) by the following menu items:

Sound/Load profile	loads a sound profile (generate sound channels without play instructions),
Sound/Load from object	loads a sound object (the sound profile is adjusted automatically),
Sound/Save as object	saves the sound as object, for further re-use.

Face Player can play (in a hardware-dependent quality, at the moment of writing on Suns only) a movie with sound play instructions recorded in a sound movie file (see Appendix 1.5).

Movie/Sound	writes playing instructions frame by frame into ‘.asmov’ sound movie file which can serve as an input for Face Player.

6.8.2 Sound play instructions

When the cursor is in the staff of a sound channel, play intervals per sound channel can be specified, starting at the time of the cursor, by the following mouse events:

Mouse Press	continue playing for 1 second (if not playing at the given time),
Mouse Press + ALT	play from the beginning for 1 second (if not playing),
Mouse Press + SHIFT	delete playing interval (the cursor must be over the interval),

Mouse Press + CTRL	grab start/end of playing interval (if there is one at mouse location),
Mouse Drag + CTRL	move start/end of playing interval.

In order to specify longer play times, add a 1 sec long play interval, and then adjust the endpoint. Some consistency is maintained: you cannot add ‘continue’ if the last play interval terminated with ‘stop’. If a ‘start play’ instruction is given at certain time, the termination of the previous interval will turn to stop automatically.

!! The user can see if a play interval starts from the beginning or continues the previous one by looking at the sign at the end of the previous interval: if it is filled, then the current interval plays from the beginning, otherwise it continues.

!! It is not assured that the time intervals do not overlap. Overlapping play might result in suspending the sound at the end of play intervals, so such situations should be avoided. When an interval is added or a playing length is changed, the sound labels (if any) will be placed properly and shown in red. They are for information, and cannot be edited in Animation Editor.

It is possible to clear all playing instructions by the clear button next to the name of the sound channel.

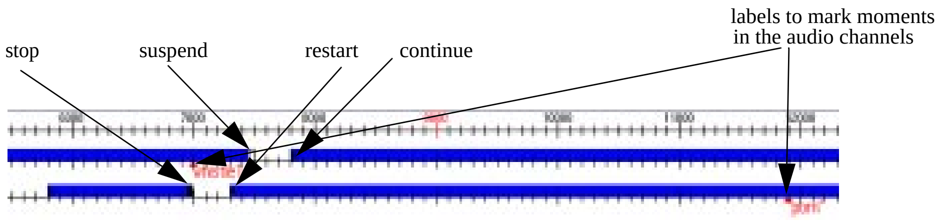


Figure 16. Two sound channels with play intervals given.

6.9. Tuning Animation Editor

6.9.1 Automatic backup generation

When Animation Editor is started up, a timer is started too and it is assured that periodically, after a certain time interval, the so-called Autosave frequency, the current animation is saved into the `Animations/usernameAnimationBackup.aanim` file. If the system crashes, or unintended changes are made, or Animation Editor is quit without having saved the animation before, this file can be read in to restore the last saved status of the animation.

6.9.2 Setting options

Via the `Options/Set parameters` menu the user can modify the following parameters (default values shown in brackets):

Sampling rate in ms (40)	the sampling rate offered when a movie is made,
Maximum animation length in ms (180000):	the intended max length for the animation. If a loaded animation is longer, then its length will be the max length,
Autosave frequency in ms (600000):	the current animation is saved with this time frequency,
Time upscaling rate default (1.25):	time scale up operation will be done by this rate, which must be at least 1,
Time downscaling rate default (0.8):	time scale down operation will be done by this rate, which must be between 0 and 1,

Value upscaling rate default (1.25):	value scale up operation will be done by this rate, which must be at least 1,
Value downscaling rate default (0.8):	value scale down operation will be done by this rate, which must be between 0 and 1.
Face Player window scale (0.4):	The Face Player windows will be this fraction of the full size. It is recommended to use a small fraction, otherwise the window may take up the whole screen and the drawing speed would be insufficient.
Machine on which Persona is running (localhost)	To be used at CWI only, to open a socket between Persona and Animation Editor.

6.10. Overview of keyboard shortcuts in Animation Editor

Focus and selection

Ctrl+F	Focus none
Ctrl+Shift+D	Focus all used
Ctrl+Shift+N	Focus all shown
Ctrl+Shift+S	Select all in focus
CTRL+SHIFT+M	Selects the time interval between the start and end markers of the APs in focus.
CTRL+SHIFT+C	Clear selection range (nothing is selected).

Editing while the mouse is in an AP staff

Ctrl+Z	Undo
Ctrl+X	Cut
Ctrl+C	Copy
Ctrl+P	Paste
Ctrl+W	Flip
Ctrl+Q	Paste with flip
Ctrl+Shift +T	Scale time up
Ctrl+T	Scale time down
Ctrl+Shift +V	Scale value up
Ctrl+T	Scale value down
Ctrl+U	Move value up
Ctrl+D	Move value up
Ctrl+L	Move selection left
Ctrl+R	Move selection right

Selecting and placing markers while the mouse is in the time ruler

Mouse Press + CTRL	Start selecting animation parameters in focus
Mouse Drag + CTRL	Mark selection for all animation parameters in focus
Mouse Press	Insert synch marker
Mouse Press + SHIFT	Delete synch marker
Mouse Press + ALT	Set vertical time marker
Mouse Press + ALT +SHIFT	Remove time marker, if there was one at the time of mouse press
Mouse Press + CTRL+ALT	Select time slice between set vertical time marker and the current time

Editing sound

Mouse Press	Play
Mouse Press + ALT	Play from beginning
Mouse Press + SHIFT	Delete playing interval
Mouse Press + CTRL	Grab start/end of playing interval
Mouse Drag + CTRL	Move start/end of playing interval

7. Implementational issues

7.1. General considerations

As already said, CharToon as a whole is implemented in Java. Our main reasons for choosing Java where:

1) The promise of problem-free portability between Unix-, Windows- and Macintosh platforms.

2) The possibility of producing animations for Web-applications: A Face Player in Java can be embedded in Applets. In such a setup only animation parameters have to be transmitted, which for reasonable faces animated at 25 frames per second amounts to about 15kB of data per second. So real-time animation via the web is easily achievable.

Java (in its 1.1 version) did qualify for the task because:

1) Its interface and graphics tools (the AWK toolkit) where (just) sufficient.

2) There is support for multi-threading which opens the possibility of driving animations from multiple sources which in its turn gives support for net based rendering of interactions between avatars where different avatars are controlled from different places but shown at the same place (think of video-conferencing!).

3) Experiments were done which demonstrated that a Face Player in Java was fast enough to animate a drawing with some 40 CPs on a Sun Sparc Ultra at 25 frames/second. (Note: on a 400Mhz Wintel platform which we use now, speed is still much higher)

7.2. Communication using ascii scripts

Face, movie and animation descriptions ('.dat' files, '.amov' files etc.) could have been implemented as serialized Java objects. We choose ascii descriptions because:

1) Working with ascii files keeps the way open for using CharToon with data from other non-Java sources (e.g. driving Face Player directly with performer data instead of with movies produced by Animation Editor).

2) In the case of saved faces they turn out to result in faster save- and open operations then their Java-object alternative. (For the other files we have no comparison, but io-speed is certainly not a problem.)

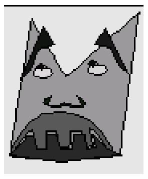
3) They can be understood by someone who knows how to read them (which is not so difficult. maybe with the exception of '.dat' files). This helps tremendously during debugging of a system under construction.

4) More often then not, serialized objects cannot be read in by a newer version of the system. This also holds true (but in our case less frequently) for our ascii descriptions but those can be salvaged by simple text editing. That way precious face-descriptions and animations survive system-modifications.

7.3. Face Editor and Face Player

The drawing components used by Face Editor are implemented as a carefully chosen hierarchy of Java classes. They have the property that when they must be drawn, they have methods to draw themselves. This class hierarchy serves a dual purpose: Face Player displays its drawings using the same set of classes. In addition, the two programs share code for the instantiation of objects of those classes and for reading face description ('.dat') files.

Face Player begins by instructing all components of a drawing to draw themselves in their neutral form whereby if so desired, a separate thread is created for every *foreground* component. Next a timer is started which executes a special method of Face Player at the desired frame-rate. This method instructs all foreground components to redraw themselves. Before doing so, every component independently calls a method of a specially provided 'parameter obtainer' class which returns the values of the animation parameters for the animation frame at hand. Because of this independence and the 'parameter obtainer' class it is possible to deal with different sources of animation parameters by using different 'parameter obtainer' classes (which are not provided by the system as it is; they have to be tailor-made by coding them in Java). Cur-



rently, Face Player can play movies from a file, but is can also obtain parameters from a socket which is supposed to provide (real time) performer data.

7.4. Emotion Disc

Emotion Disc is a Face Player surrounded by a layer which implements the disc. When Emotion Disc is used for the selection of expressions, at every mouse event caused by dragging the mouse through the disc the animation parameters corresponding to the current mouse position are transmitted to Face Player by a call of its Snapshot method. The mouse position is stored in a file when the ‘OK’ button is pushed. When Emotion Disc is used for playing a sequence of selected expressions, those recorded mouse positions are read back, mapped to the corresponding animation parameters and transmitted to Face Player.

7.5. Animation Editor

Animation Editor is basically an interactive graphical editor. The staff for each AP is implemented as a separate canvas. These canvases are repainted one by one, in order to minimize the time spent on refreshing the window.

7.6. Known problems

We have experienced that if Animation Editor or Face Editor has been running for a long time (>10 hours), then sometimes the program crashed with the Java error of ‘Out of memory’. If a complex face and/or long animation is to be dealt with, it is recommended to extend the memory assigned to the java Virtual Machine before starting up CharToon. This is to be done by: `java -mx64m AnimEdit` ....

Repaint of windows may turn suddenly slow every now and then, probably as a side effect of Java’s garbage collection.

Sometimes Animation Editor windows do not get repainted or updated when they should. An update/repaint can be then forced by resizing/moving the window slightly.

Sometimes the Face Player window gets hidden behind the Animation Editor window. It should be clicked at somewhere in its bounding frame or title to be shown again.

While Face Player (started up from Animation Editor) is playing an animation, the user should not move the mouse over AP staves, as this results in many red cursor lines.

There is a bug on Windows systems: the on-line *Help* windows sometimes come up grey. Click on its right side below the scroll bar (to perform a minor scroll) in order to get its content.

7.7. Reporting bugs

This manual documents the first release of CharToon. The user thus may encounter bugs, especially when running CharToon under systems like MAC OS or Windows 98 with which we have not had experience.

We ask you to report any bug by sending an e-mail to: Han.Noot@cwi.nl.

In the e-mail you should include the description of the environment and the scenario (integrated or stand alone mode, what has been edited,...), the anomaly, and all the error messages shown in the window where CharToon was started.

Further releases with new features (e.g. handling of higher-level building blocks of animations and constraints) are to be expected.

Acknowledgments

We thank Zs. Paál, B. Kiers, A. Lelièvre, J. Hendrix, K. Thórisson for making some of the demos shown as examples. We are indebted to Chris Thórisson for his ToonFace system, which inspired us to develop our system, and for his many useful comments on earlier versions of CharToon. Finally, we thank Paul ten Hagen for his useful comments on this manual and throughout the project. The work has been carried out as part of the FASE project, sponsored by STW under nr. CWI 66.4088.

1. Appendix: CharToon's file-formats and directories

1.1. .aprof profile file

Below, the expected syntax of the profile files (ascii files with .aprof extension) and the intended usage of the profile parameters is explained.

When reading in a profile file, lines starting with // are not dealt with, and they may occur anywhere in the file. The rest of a line after // will be ignored.

Parameters in the same line should be separated by at least one space (more allowed).

By n-th line we refer to the n-th non-comment line in the file. Below you find the DemoHead.aprof with sample data, with the explanation as comment lines. Load this file into Animation Editor and check the different AP definitions. Realize the advantage of having granularity > 1 for large intervals of possible values. Moreover, note the difference (in zoomed-in view) between APs with forced discrete values (e.g R_Head_x) and with arbitrary integer values (e.g. T_Head_x). Note that the neutral value of dynamic scales is not necessarily 1, but the middle of the min-max interval. Check the group ids.

```
// 1st line:
// PROFILE NAME    Characters   Max 20 chars
// To identify different profiles, roughly speaking one per model.
// FACE NAME    Characters   Max 20 chars   OPTIONAL!
// To specify which face model file is to be used. If not given in the
// file, it will be assumed to be the same as PROFILE NAME. It can be
// useful with the Persona model.
DemoHead DemoHead
// 2nd line:
// NUMBER OF APs Integer      Positive
13
// In the rest of the file, this number of AP description lines will
// follow.
1 -1 5 R_Head_x 0 -130 50 -40 18
2 -1 5 R_Head_y 0 -59 41 21 10
3 -1 10 T_Head_x 1 52 132 90 40
4 -1 10 T_Head_y 1 -343 -255 -259 22
5 -1 5 L_Head_x 0 85 315 225 23
6 -1 5 L_Head_y 0 -63 47 27 11
7 -1 3 R_Eye_x 1 -12 52 29 64
8 -1 3 R_Eye_y 1 -61 21 -20 82
9 -1 3 L_Eye_x 1 -12 52 29 64
10 -1 3 L_Eye_y 1 -61 21 -20 82
11 -1 6 Mouth_Loc_x 1 320 440 390 12
12 -1 6 Mouth_Scale_x 3 0.5 3.0 1.75 0
13 -1 6 Mouth_Scale_y 3 0.0 4.0 2.0 0
// 3rd, 4th,....15th line (total: NUMBER OF APs+2 lines):
// each such line has values for the following parameters:
// ID FACSID GRID AP NAME TYPE MIN_VAL MAX_VAL NEU_VAL GRAN SCALE COMM.
// ID              Integer      0=<ID=<1000
// To identify the animation parameter within the profile.
// FACSID          Integer      0=<ID=<1000
// The ID used for reference to MPEG or other AU sets, -1 as default
// (not used in CharToon).
// GRID          Integer      0=<ID=<19   OPTIONAL!
// The ID of the group the AU belongs to, 0 as default.
// The following group ids are to be used (FaceEditor offers these):
//nr      name    comment
//0       misc    default if no grouping is to be used
//1       eyebrows & forehead
//2       eyelids
//3       eyeballs, pupils
//4       nose
//5       cheek
//6       mouth
//7       jaw
//8       tongue
//9       ears
//10      head
//11      neck
//12      hair
//13      more facial features  features for which 1-12 do not apply
//14      moustaches
//15      tears
//16      glasses
//17      face objects1 (e.g. cap, earrings)
//18      face objects2 (e.g. tie,...)
//19      non-face objects        (e.g. backgrounds)
// AP NAME         Characters   Max 12 non-space chars
// To show to the user (usually NOT to identify the AP, except with
// “reanimate”)
// TYPE            Integer      0,1,2,3
// 0: the value should be an integer according to the given granularity
// (discrete values)
//  1: the value should be an integer within the extreme values
//  2: the value should be a float according to the given granularity
//  3: the value should be a float within the extreme values
//
// MIN_VAL Float        ####.&&&
// Minimum value, integers without decimal point and only decimals
// are OK too
// MAX_VAL Float        ####.&&&
// maximum value, integers without decimal point and only decimals are
// OK too
// NEU_VAL Float        ####.&&&
// neutral value, integers without decimal point and only decimals are
// OK too
// GRAN            Integer      0=<GRAN
// The [MIN_VALUE, MAX_VALUE] interval is divided into GRAN equal
// parts. The value of the AU should be one of the GRAN+1 division
// values. The NEU VALUE should be one of these values.
// Depending on TYPE, the GRAN is compulsory (TYPE 0,2) or
// will be used initially, but can be refined later by the user.
// GRAN=0 will result in no discretisation at all.
// SCALE           Float or Integer 1
// Optional, for later use, now skip or give 1.
// COMMENT         String
// Short explanation of the effect of the AU (optional).
```

1.2. .asprof sound profile file

The sound ascii file is used to store .au file reference and annotations (made by AnimEdit or a text editor).

Below is the Demo.asprof file, with data for a single sound channel and comment lines starting with //.

```
// Name of the sound profile
Default_Sound_1
// Sound profile with 1 sound channel
// First row: the number of sound channels
1
// Then for each sound channel a row, each followed by row(s) for the
markers (if any).
// ID name_of_Chan sound_file   dur. (ms) nofMarkers comment(optional)
1      DonGiovanni       "DonGiovanni.au"      6000  2 "something as com-
ment"
// if nofMarkers>0, then for each marker in a separate row
// time (in mseconds) and label (string)
500  "blink"
1500  "look_up_end"
```

1.3. .dat face file

The '.dat' files (face files) describe faces as constructed by Face Editor. By default. they are found in CharToon-home-directory/Faces. UserDefinedComposite components are also stored in files with the '.dat' extension. Those files are basically subsets of face description files; they will not be discussed separately here. By default, they are to be found in CharToon-home-directory/UserDefinedComposite.

The format of a '.dat' file is:

```
background-color width-of-drawing height-of-drawing
comp-spec
          .
          .
comp-spec
```

Below we discuss the contents of ‘.dat’ files in a form which loosely resembles a grammatical description. Note: For the grammatical alternative we use the symbol ‘!’ instead of the usual ‘|’. Syntactical catagories are -as usual- enclosed between ‘<‘ and ‘>’ All other symbols are literals including the ‘|’ symbol which does NOT have the usual meaning of grammatical alternative; it is a character which actually occurs in ‘.dat’ files.

There is a comp_spec line for every drawing-component. These lines have different format for different component types; <comp_spec> can be <b_comp_spec>, <c_comp_spec> or <usd_comp_spec> for basic components, composite components or UserDefinedComposite components respectively. They all begin with a <global_properties> part but they end different for different kinds of components.

<b_comp_spec>=<global_properties>/<nr_of_points>|‘nr-of-points’<pointspec>s

<c_comp_spec = <global_properties> | 0 | a-component-dependent-nr-of <b_comp_spec>s

<usd_comp_spec>=<global_properties>|<nr_of_comps>|’nr-of-comps’ <comp_spec>s

<global_properties>=<name>|<visibility>|<backgr_and_layer>|<spline_or_poly>|<origin>|<colors>|<dynscale>

<name>= A name by which this component is known to the system e.g. ‘FixedClosedFilledPoly’

<visibility>=0!1!2|<parameter_id>|<parameter_name>|<mpeg_group>

Here, 0 indicates that the component is permanently invisible (used for skeletons), 1 that it is permanently visible and 2 that its visibility can be dynamically switched on or off. <parameter_id> is a unique integer identifying this animation parameter, <parameter_name> is a string (the same one which shows up in Animation Editor next to this animation parameter and <mpeg_group> is again an integer.

<backgr_and_layer>=0!1|<integer between 0 and 9 >

Here 0 signals a foreground- and 1 a background component while the <integer between 0 and 9> denotes the component’s rank.

<spline_or_poly>=0!1|<distance>

Here 0 indicates that the component is drawn as a polygon or polyline, 1 that it is drawn with a curved outline. <distance> is an integer specifying the distance in pixels between calculated points on the curve. Iin the case of a polygon it has value 0.

<origin>=<controlpoint_spec>

The data describing the <origin> of the component have the form of a <controlpoint_spec> (see below). When we have an animated origin, all 17 fields of this specification are relevant, otherwise only the x- and y- fields are significant although the other fields are present too.

<colors>=<poly_or_spline_color>|<fill_color>

In the colors specification there are two integers each specifying a color. The integers have values representing three bytes for the RGB components of the color.

```
<dynscale>=0!1|<x_parameter_id>|<y_parameter_id>|<parameter_name>|<min_x
_fact>|<max_x_fact>|<min_y_fact>|<max_y_fact>|<mpeg_group>|<neut_x_fact
>|<neut_y_fact>
```

Here `0` indicates that the object is not dynamically scalable, `1` that it is. Only in the later case all the other fields are relevant, but they are always there. First, we have to unique id-s (integers) identifying the animation parameters for scaling in the x- and y-direction. Then comes a parameter name (a string) which is shown in Animation Editor to identify the parameters. Finally we have 7 integers defining minimum, maximum and neutral values for the scale parameters and the mpeg-group associated with this parameter.

The last element to discuss is the <`point_spec`>s part of <`b_comp_spec`>:

```
point_spec= <fixedpoint_spec>!<controlpoint_spec>
fixedpoint_spec = <f!z><+!-!&!*>|<x>|<y>|
```

Here, `f` indicates a FP, `z` a frozen FP of a body polygon. A ‘`+`’ indicates a spline boundary where the following spline segment is really drawn, a & indicates a spline boundary followed by a spline which is not shown, while & and `-` indicate continuation of the spline in a shown - resp. not shown outline. <x> and <y> are integers specifying the point’s position.

```
<controlpoint_spec>=c<+!-&!*>|<x>|<y>|<tx>|<ty>|<bx>|<by>|<lx>|<ly>|<rx>
|<ry>|<parameter_name>|0!1|0!1|<x_parameter_id>|<y_parameter_id>|<x_ste
p>|<y_step>|<mpeg_group>
```

The meaning of most of the elements of `<controlpoint_spec>` can be deduced from their names and the discussion of other elements above. The new ones are:

1) The integers `<tx>...<ly>` specifying the coordinates of the controlpoint’s hairline top-, bottom-, left- and right endpoints.

2) The two `0!1` fields which tell whether the point cannot- or can move in the x-, resp y- direction.

3) The integer `<xstep>`- and `<ystep>` fields which specify the step (in pixels) by which the controlpoint may move.

Example of a part of the ‘.dat’ file for the Chicken shown in Plate 1.

```
16776960 1000 970

FixedFilledEllipse|1|-1|Visibility|0|0|7|0|10|490|423|0|0|0|0|0|0|0|0|
FixedFilledEllipse1:Ctrl0|0|0|0|0|1|1|0|0|8684676|16579836|0|null|1|2|
0.0|2.0|0.0|2.0|0|0|0|5|f-|-70|-66|f-|70|-66|f-|70|66|f-|-70|66|f-|-70
|-66

FixedFilledEllipse|1|-1|Visibility|0|0|8|0|10|397|423|0|0|0|0|0|0|0|0|
FixedFilledEllipse1:Ctrl0|0|0|0|0|1|1|0|0|8684676|16514043|0|null|1|2|
0.0|2.0|0.0|2.0|0|0|0|5|f-|-82|-78|f-|82|-78|f-|82|78|f-|-82|78|f-|-82
|-78

AnimatedFilledEllipse|1|-1|Visibility|0|0|9|0|10|421|427|421|412|421|4
42|406|427|436|427|Pupil-pos-R|1|1|1|2|1|1|0|1|263172|460551|1|Animate
dFilledEllipse1:Scale1|3|4|0.0|2.0|0.0|2.0|0|0|0|5|f-|-16|-18|f-|16|-1
8|c-|16|18|16|3|16|44|-10|18|37|18|Pupil-size-R|1|1|5|6|1|1|0|f-|-16|1
8|f-|-16|-18

AnimatedFilledEllipse|1|-1|Visibility|0|0|9|0|10|516|432|516|417|516|4
47|501|432|531|432|Pupil-pos-L|1|1|7|8|1|1|0|1|263172|460551|1|Animate
dFilledEllipse1:Scale1|9|10|0.0|2.0|0.0|2.0|0|0|0|5|f-|-13|-15|f-|13|-
15|c-|13|15|13|-10|13|42|-10|15|35|15|Pupil-size-L|1|1|11|12|1|1|0|f-|
-13|15|f-|-13|-15

SkeletonPointsAnimatedClosed|1|1|Visibility|0|0|5|0|10|394|370|0|0|0|0
|0|0|0|0|SkeletonPointsAnimatedClosed1:Ctrl0|0|0|0|0|1|1|0|0|0|1671193
5|0|null|11|12|0.0|2.0|0.0|2.0|0|0|0|0|ClosedFilledBodyPoly|1|1|Visibi
```

```
lity|0|0|0|0|10|364|224|0|0|0|0|0|0|0|0|SkeletonPointsAnimatedClosed1B
odyPoly:Ctrl0|0|0|0|0|1|1|0|0|14038047|14371908|0|null|0|0|0.0|2.0|0.0
|2.0|0|0|0|43|z-|-65|501|f-|-75|447|f-|-79|414|f-|-84|375|f-|-97|323|f
-|-113|283|f-|-168|299|f-|-217|285|f-
```

etc.

1.4. .amov movie file

Below the beginning of the DemoMovie.amov file is listed, with explanation in comment lines starting with //. The movie was made of the animation in the DemoAnimation.aanim file. Here only the lines for the first 10 frames are given.

```
// name of the animation
DemoAnimation
// name of the profile
DemoHead
// nof APs
13
// ID of all APs for which value is given (in this order) for each frame
1 2 3 4 5 6 7 8 9 10 11 12 13
//sampling frequency in ms
40
// nof frames
51
// for each frame: time (in msecs), followed by a line with the AP values
0
-40 21 90 -260 215 27 29 -20 29 -20 390 1.250 0.933
40
-30 21 90 -277 205 27 29 -29 29 -29 390 1.100 0.960
80
-20 21 90 -293 195 27 29 -37 29 -37 390 0.950 0.987
120
-20 21 90 -310 185 27 29 -45 29 -45 390 0.800 1.013
160
-10 21 90 -326 175 27 29 -53 29 -53 390 0.650 1.040
200
0 21 90 -343 165 27 29 -61 29 -61 390 0.500 1.067
240
-10 21 90 -326 175 27 29 -53 29 -53 390 0.700 1.067
280
-20 21 90 -309 195 27 29 -45 29 -45 390 0.900 1.067
320
-20 21 90 -293 205 27 29 -36 29 -36 390 1.100 1.067
360
-30 21 90 -276 225 27 29 -28 29 -28 390 1.300 1.067
...
```

1.5. .smov sound file

Below the beginning of the DemoSound.smov file is listed, with explanation in comment lines starting with //. The sound movie was made of the sound play instructions in the DemoSound.osound file, for the DemoSound.asprof sound profile (1 sound channel). Below data for the first 6 frames is given.

```
// name of the sound
Sound for DemoAnimation
```

```
// Nof used channels
1
// audio file for each used channel
"DonGiovanni.au"
//sampling frequency in ms
40
// nof frames
51
// for each frame two lines: time (in msecs) and then an integer:
// 0 silent
// 1 keep playing
// 2 play from start (first, or after last stop)
// 3 continue (after pause)
// 4 stop
// 5 pause
0
2
40
1
80
1
120
1
160
1
200
1
...
```

1.6. .aanim animation file

Below you find the DemoAnimation.aanim file, containing comment lines starting with // explaining the syntax.

```
// name of the animation, a complete line of characters
Animation with 10 APs
// name of the profile the animation was made for
DemoHead
// number of all APs with some KP
10
// for each AP with KPs two lines:
// ID of included AP and number of KPs, followed by time-value list for
// each used AP
1     3
0 -40.00 200 0.00 400 -40.00
4     3
0 -260.00 200 -343.00 400 -259.00
5     3
0 215.00 200 165.00 400 235.00
7     3
500 28.00 800 43.00 1000 29.00
```

```
8     6
0 -20.49 200 -61.00 400 -20.00 1600 -19.00 1900 -31.00 2000 -19.00
9     3
500 28.00 800 18.00 1000 29.00
10     6
0 -20.49 200 -61.00 400 -20.00 1600 -20.00 1900 -34.00 2000 -20.00
11     7
500 397.00 800 437.00 1000 407.00 1200 390.00 1300 417.00 1500 417.00 1600
380.00
12     14
0 1.25 200 0.50 400 1.50 900 1.67 1000 1.00 1100 2.00 1200 1.42 1300 2.00
1400 1.67 1500 1.08 1600 2.25 1700 1.50 1800 1.08 2000 0.75
13     14
0 0.93 200 1.07 400 1.07 900 0.80 1000 1.87 1100 0.67 1200 1.33 1300 1.60
1400 0.67 1500 1.73 1600 0.67 1700 1.33 1800 2.27 2000 2.67
```

1.7. .dsc disc file

Emotion Disc's actual configuration is determined by the content of a '.dsc' file, found in CharToon-home-directory/EmotionDisc/Discs.

An example of such a file is given below:

```
//Name of profile
B_maan
//Name of Face
B_maan
//Nr of emotions
6
//Nr of snapshots per emotion (animation)
10
//Nr of used AUs
16
//AUs
53  54  55  56  58  75  76  77  78  80  83  84  81  82  85  86
//Joy
339  220  22  -19  -1  514  220  -22  -19  -1  -36  -38  33  -20  107  -41
339  220  22  -20  -2  514  220  -22  -20  -2  -39  -41  33  -20  110  -44
339  220  22  -20  -2  514  220  -22  -20  -2  -41  -44  33  -20  112  -47
339  220  22  -21  -3  514  220  -22  -21  -3  -44  -47  33  -21  115  -50
339  220  22  -21  -3  514  220  -22  -21  -4  -46  -50  33  -21  118  -53
339  220  22  -22  -4  514  220  -21  -22  -4  -49  -52  33  -21  120  -55
339  220  22  -22  -4  514  220  -21  -22  -5  -51  -55  33  -21  123  -58
339  220  22  -23  -5  514  220  -21  -23  -6  -54  -58  33  -22  126  -61
339  220  22  -23  -5  514  220  -21  -23  -6  -56  -61  33  -22  128  -64
339  220  22  -24  -6  514  220  -21  -24  -7  -59  -64  33  -22  131  -67
//Disgust
```

```
339  220  22  -19  -1  514  220  -22  -19  -1  -36  -38  33  -20  107  -41
....
etc.
```

Explanation:

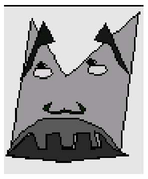
A ‘.dsc’ file is made largely ‘by hand’ using a general purpose text editor together with Animation Editor. It begins (apart from comment lines) with the name of a profile (‘.aprof’) file for the face being animated. This name is given without the ‘.aprof’ extension. The file is to be found in CharToon-home-directory/Emotion-Disc/Profiles.

Next comes the name of the face file (‘.dat’ file) for the face. This name is given without the ‘.dat’ extension. The file is to be found in CharToon-home-directory/EmotionDisc/Faces.

Next come the number of emotions followed by the number of snapshots per emotion. These two numbers correspond to the number of disc-sections resp. the number of disc-tracs + 1.

Expression-samples are passed to Emotion Disc in the form of movie file lines (‘.amov’ files). For every emotion (disc segment) there is a set of animation parameter specification lines, where there is one more line then the number of disc tracks. The first line of the set corresponds to the definition of the neutral expression (centre of disc), the next one to a very weak display of the emotion (track closest to disc centre) and finally the last line to the strongest expression of the emotion (outermost track). In the example ‘.dsc’ file we see the lines for ‘Nr of used AUs’, AUs’ and these expression definition lines (first those for Joy, then for Disgust, the other emotions are not shown). For a detailed understanding of the contents of these lines, the user should read the section on ‘.amov’ files in this Appendix.

It is suggested to produce the expression samples as follows: Start Animation Editor and load the profile for the face involved. Next choose the first emotion on the disc (in this case ‘Joy’) and produce an animation from neutral to maximum intensity for that emotion. When done, save this animation as a movie, thereby using such a sampling rate that the number of frames in the movie equals the number of samples (snapshots) one wants for the disc (in this case: 10). Next, use a text editor to extract the relevant part from the movie file (i.e. remove its header) and add the result to the .dsc file being constructed. Repeat this process for all emotions which have to be represented by its incarnation of the disc.



2. Appendix: List of demo faces and animations

The files are provided with the CharToon release in the Demo subdirs of directories Faces, Animations, etc.

Face	APs	Description	Animations	Movies	Disc
DemoHead	13	Simple head to show basic control parameters	DemoAnimation	DemoMovie	-
Generic	46	A ‘generic’ face, made for MPEG data driven animation especially	-	marcgen, janroelgen, andregen, frontgen	-
Chaplin	22	The very first head made by earlier version of CharToon	-	-	-
ChaplinMpegVert	32	Chaplin with (vertical) Mpeg parameters, used with performer data	-	-	-
Gerard	58	Subtle cartoon face of a real person	-	-	Gerard
Frans	54	Cartoon face of a real person with text panels	Frans	-	-
Mary	32	Face with hidden tears, animation with post-processed performer data	Mary	-	-
Lily	93	Subtle artistic cartoon face which can change shape too	Lily (face is not shown initially!)	-	-
NineFaces	104	Nine human and other small faces in a panel, each with a few control parameters	NineFaces	-	-
Lineheads	28	To experiment with line drawing style and ’turning heads’ effect	Lineheads	-	-
Han	61	A cartoonish head with moving stars	Han	-	-
JanReal	44	‘Realistic’ head of Janroel (real person)	-	JanReal	-
JanCart	58	Cartoonish head of Janroel	-	JanCart	-
Bold	14	Cartoonish bold head, made of predefined components	-	Bold	-
Santa	24	Santa Claus	Santa	-	-
Prof	19	Simple head in profile view	Prof	-	-
ProfPlus	25	Subtle head in profile view	-	-	-
Sandy	74	Simple face used with emotion disc experiments	-	-	Sandy
Maan	32	Moon head	-	-	Maan
Egghead	32	The first head made by artist	Egghead	-	-
Skull	2	Skull with moving jaw	-	-	-
Chicken	20	Crowing chicken-head in half profile	Chicken	-	-
Moose	39	A moose with an animated story	Moose		
Peer	24	Peer head	-	Peer	
Scenery	13	To imitate 3d effect and rotational motion	Scenery	Scenery	-
Lambda	28	A full body figure (known at CWI)	Lambda		
Joke	14	A joke with bicycle & special effects	Joke	-	-
Painting	12	Painting which can be animated	Painting		

3. Appendix: Color plates

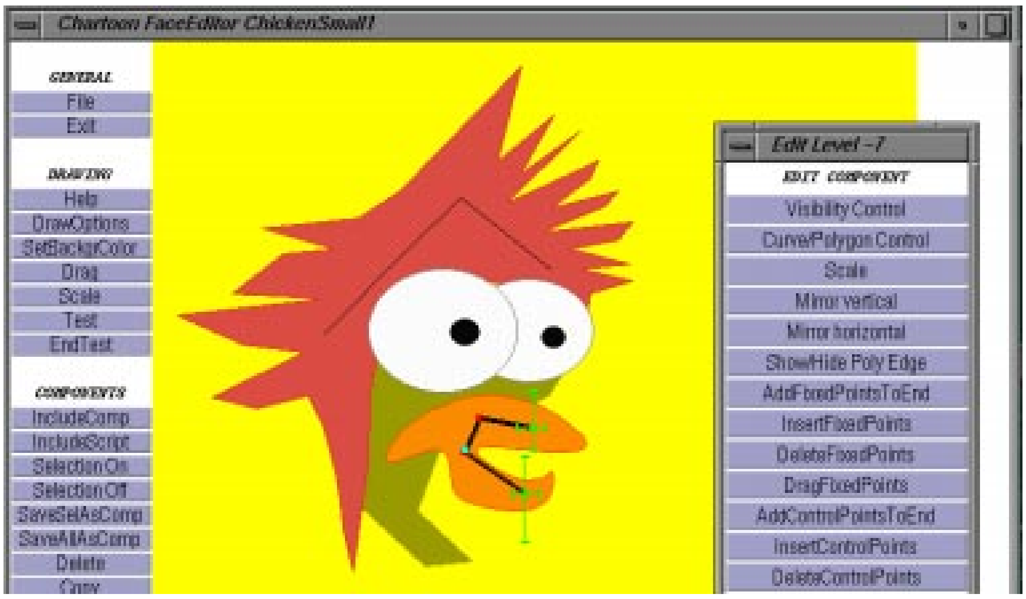


Plate 1. Face Editor. Left: Main menu. Middle: A face. Right: Pop-up edit menu.

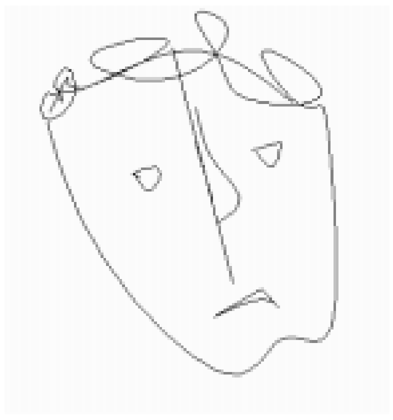


Plate 2.a. A linehead

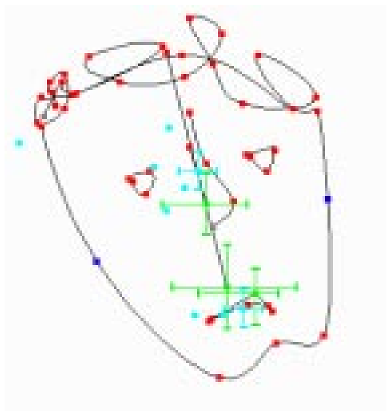


Plate 2.b. The same linehead with its components, fixed points and control points shown.

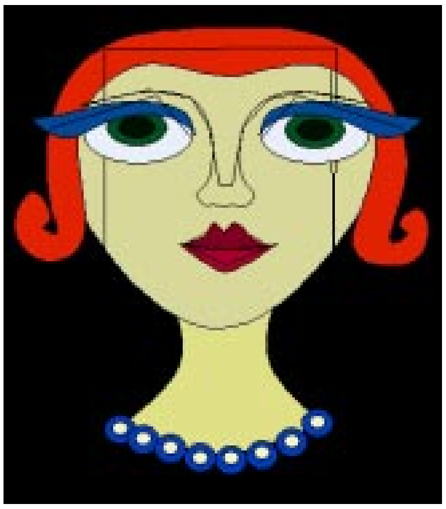


Plate 3.a. Lily with skeletons shown.

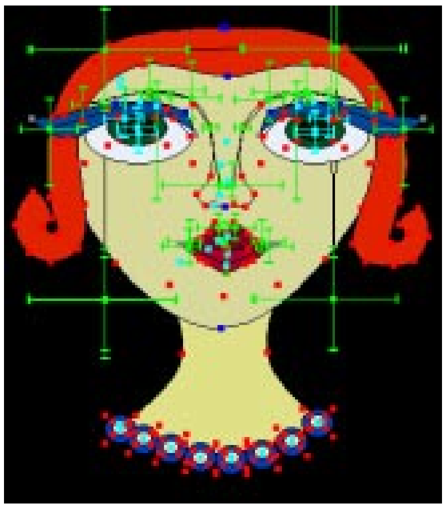


Plate 3.b. Lily with fixed points and control points shown.

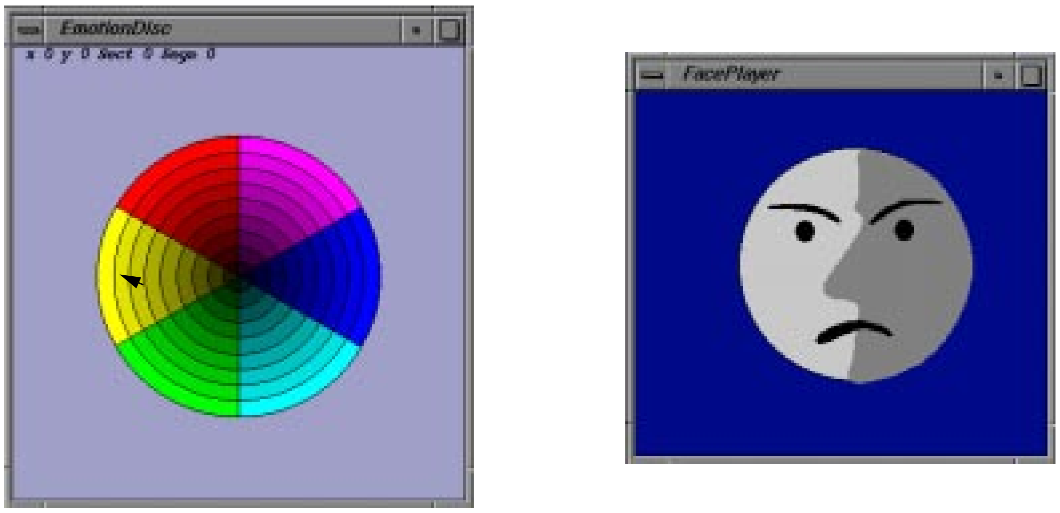


Plate 4. Emotion Disc and the face Maan in Face Player. Face Player show the expression corresponding to the cursor position in the disc.

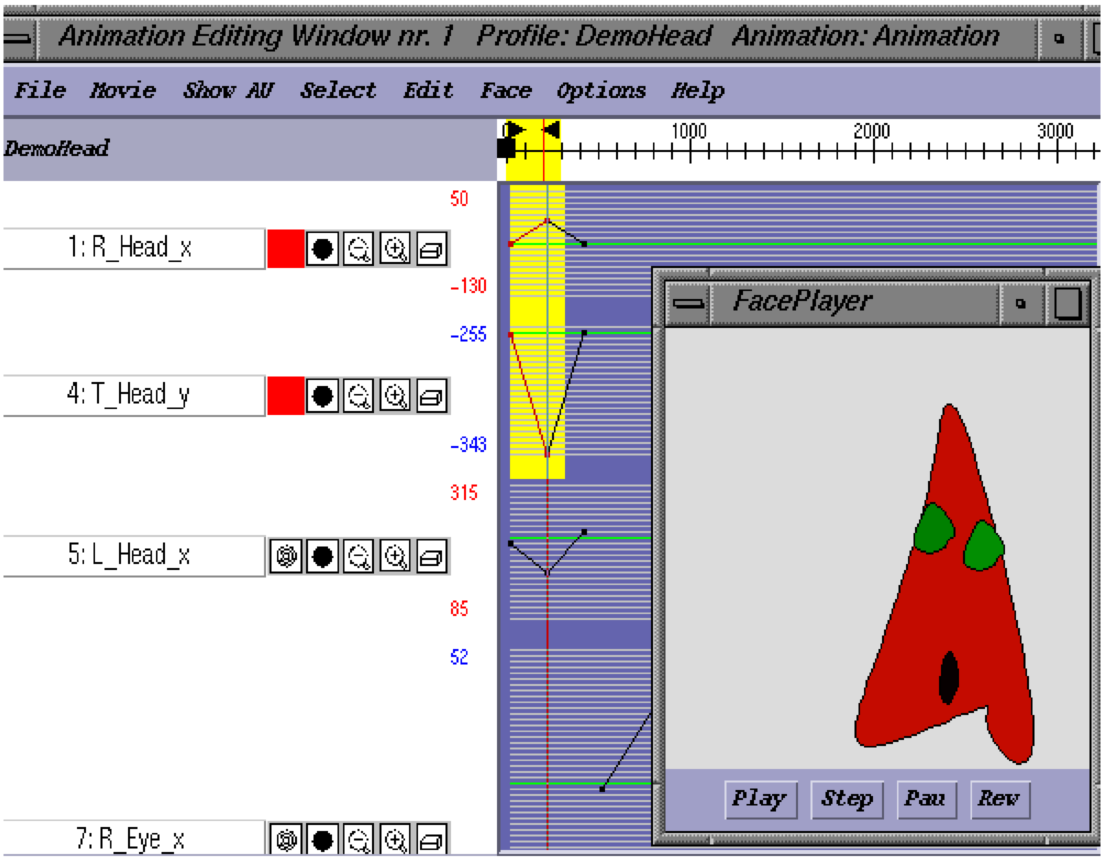


Plate 5. An Animation Editor window, with some parameters in focus, and part of the animation of those selected. Face Player is showing the face to be animated (DemoHead), with an expression corresponding to the cursor position.

Plate 6. Lily.

Plate 7. NineFaces.

Plate 8. JanReal.

Plate 9. JanCart.

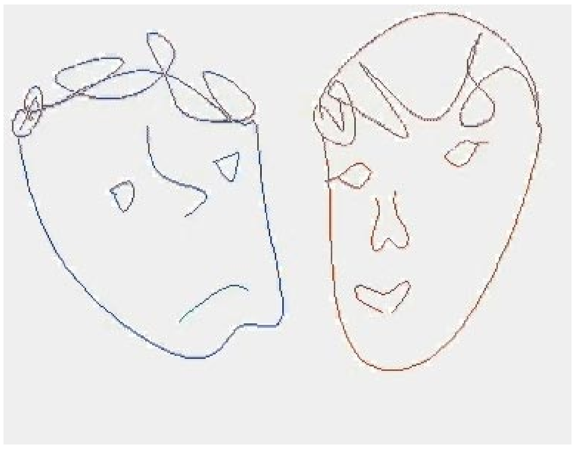


Plate 10. Lineheads.

Plate 11. ProfPlus.

Plate 12. Moose.

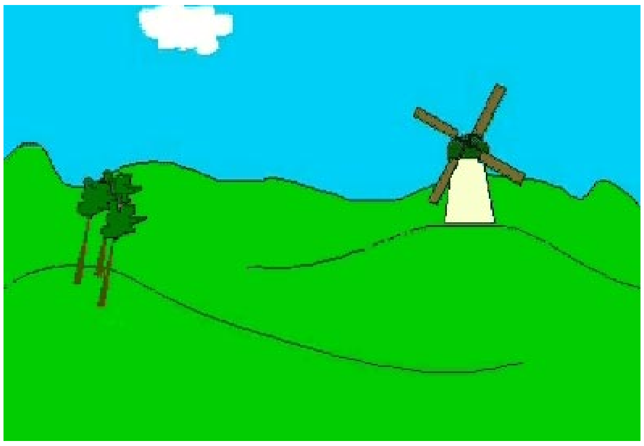


Plate 13. Scenery.