Draw keyframes in Inkscape and convert to Synfig Studio animation ! ! !

You may feel comfortable with gvim text editor and Regular Expressions to follow.
Files provided in bounce.tar.gz :

- *bounceDesign.svg (to prepare and convert)*
- *bounce.sif (the converted with XSLT sheet)*
- *one_point_dummy*
- *timecode_dummy*
- *50_point_dummy.sif*
- *bounceFinal.sif*

You may uncompress and open a terminal and change to the directory who has these files to make the steps.

Facts

As I draw and feel really comfortable drawing in Inkscape, I was wondering if all those drawings (keyframes, to be correct) could be transformed into the Synfig Studio animation file format in an automatized way, with the less work possible.

Here is a XSLT transformation sheet (http://www.synfig.com/wiki/Svg2synfig) that do the work in an excellent way, converts the SVG file into a Synfig (.sif) uncompressed file, that is a XML file.

Some things have to be prepared to do the conversion:

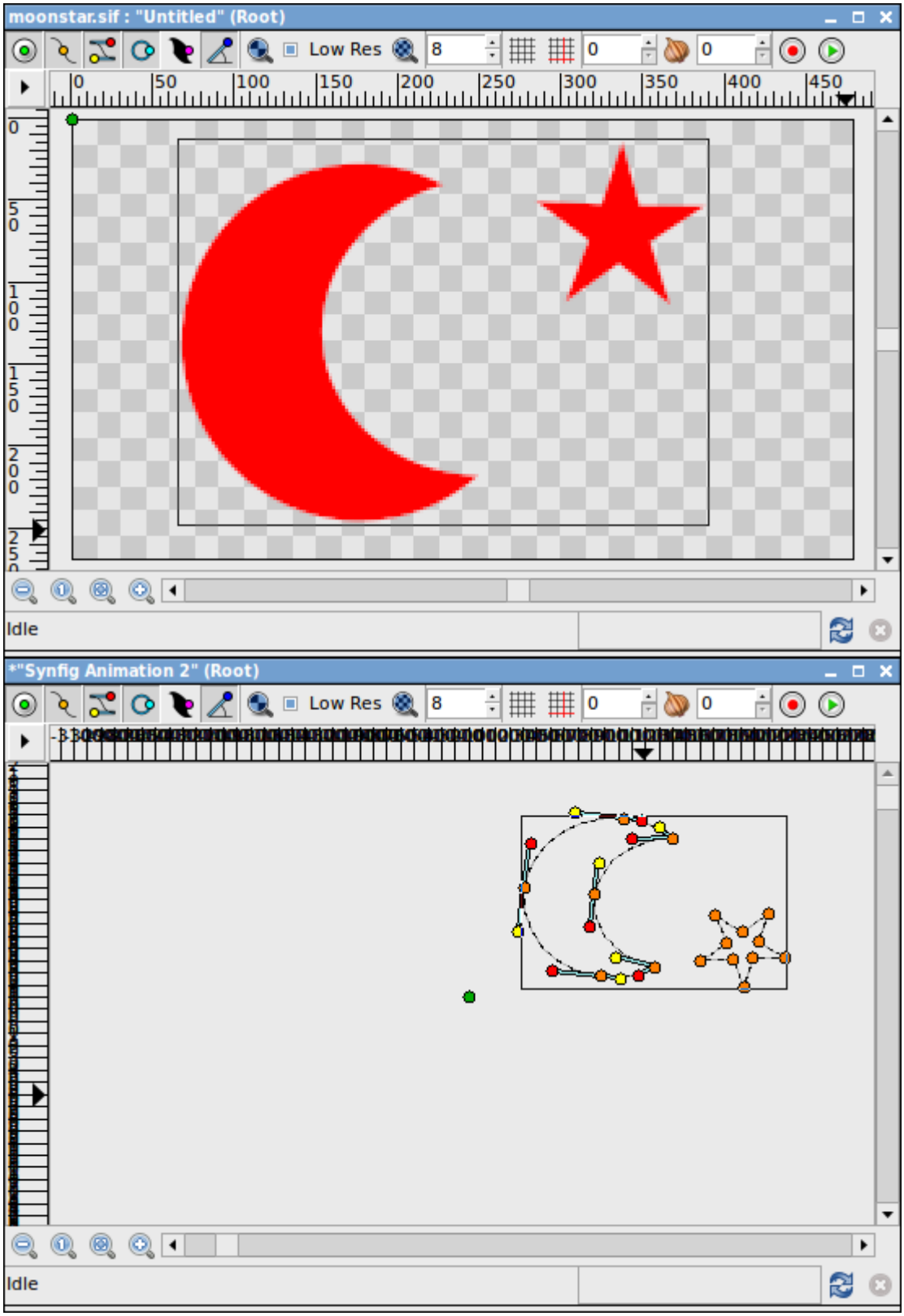
- the Rectangle, Circle and Star will not be converted unless converted to path
- combined shapes will be converted, each one becomes a layer
- if a shape has fill and outline, only the fill will be converted (Region Layer)
- if a shape has only fill, it will be converted as a Region Layer
- if a shape has only outline, it will be converted as Outline Layer
- if open shapes are combined, conversion will left them open (only 2 nodes), the others will be joined in some place
- gradients fills will not be converted, just the transparency with one of its colors
- an Inkscape Layer with all its objects will be converted as Paste Canvas
- it converts all the shapes present in the drawing, not only in the page
- try to keep the same units with Synfig default settings, so work with 480x270pt page size

With this in mind, save the Inkscape an do the conversion as explained in the link. If all went right, theres a new created Synfig file with the same design as your Inkscape drawing.

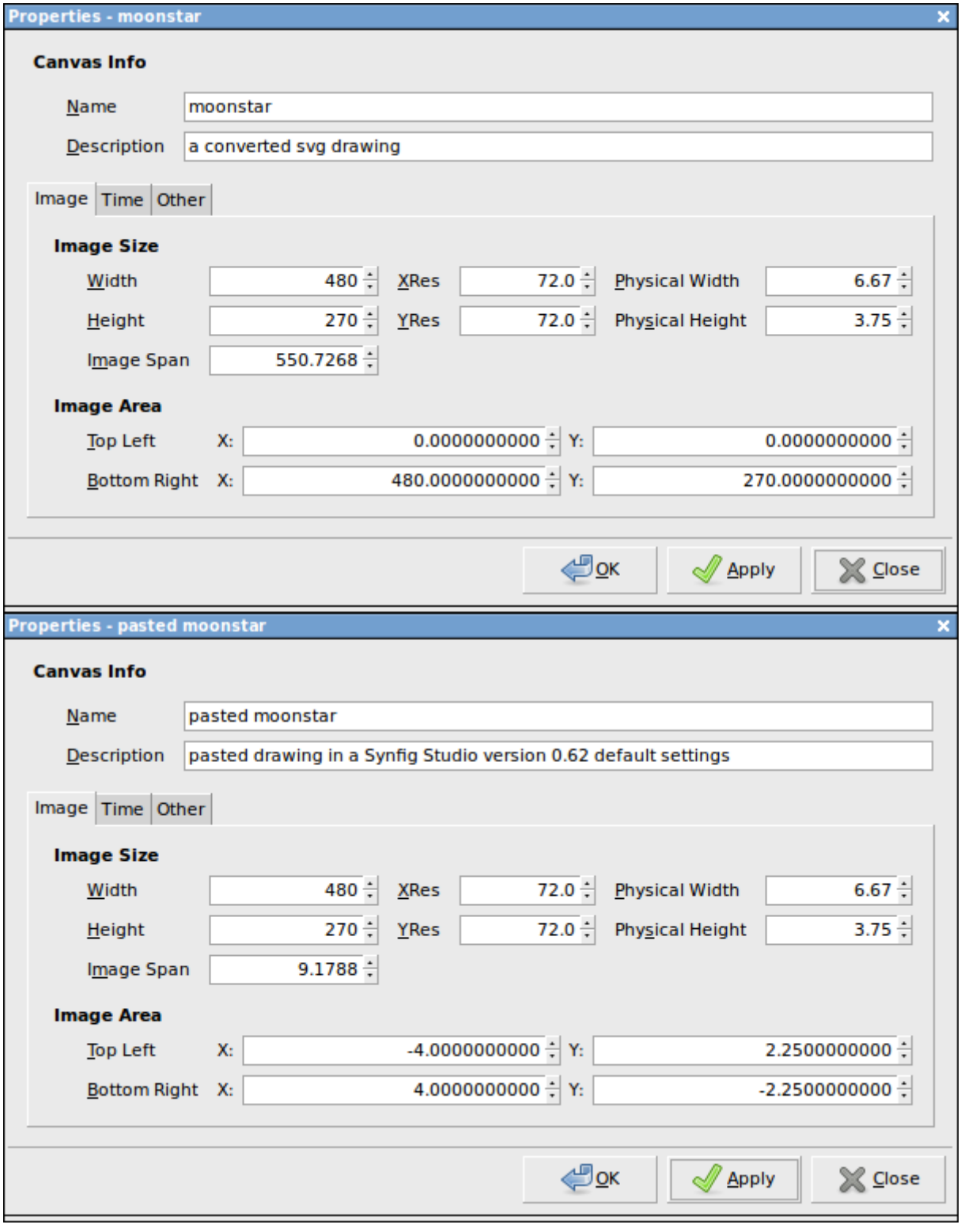
Conversion caveats

The only inconvenient found with the conversion appears when Cut whatever of the converted file and Paste into another Synfig Studio file, with its default settings. When it's pasted, the design appears really big and vertically mirrored. That's because some differences in the image resolution of the files, defined with the menu Edit | Properties Dialog. See images below.

If the converted file means to be inserted into an existent animation or design, who has its canvas with default settings, will behave as shown in the images below, that's not desirable, as I want to convert the file and get a reasonable scale to fit in the defaults image settings of Synfig Studio.



Here you can see what happens when some layers from the converted file are pasted in a new Synfig Studio with default file settings.

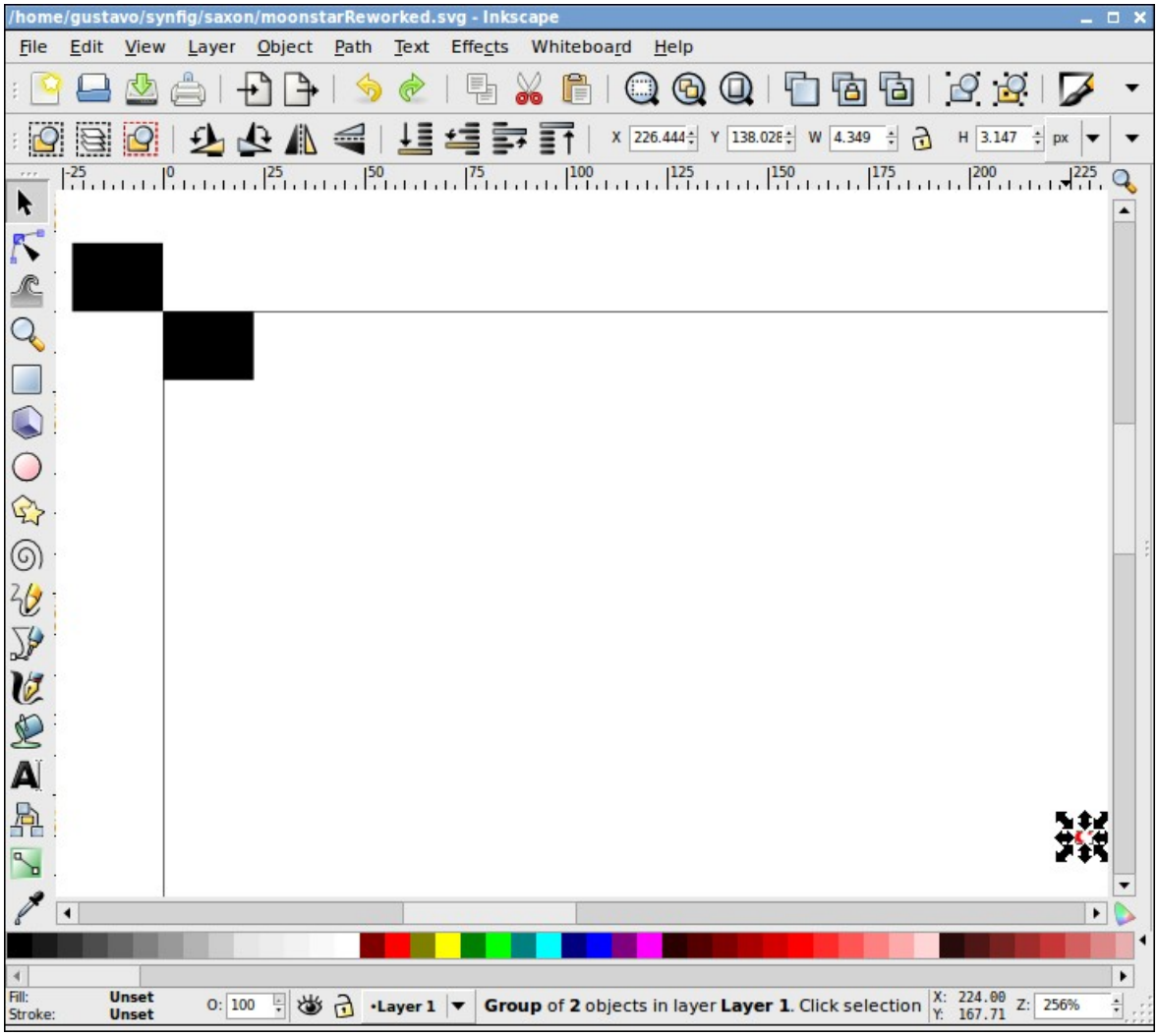


What leads to this behavior are the different setting in the image resolution. The conversion will be done with this in mind, so will have to rework the Inkscape file, for the sake of simplicity.

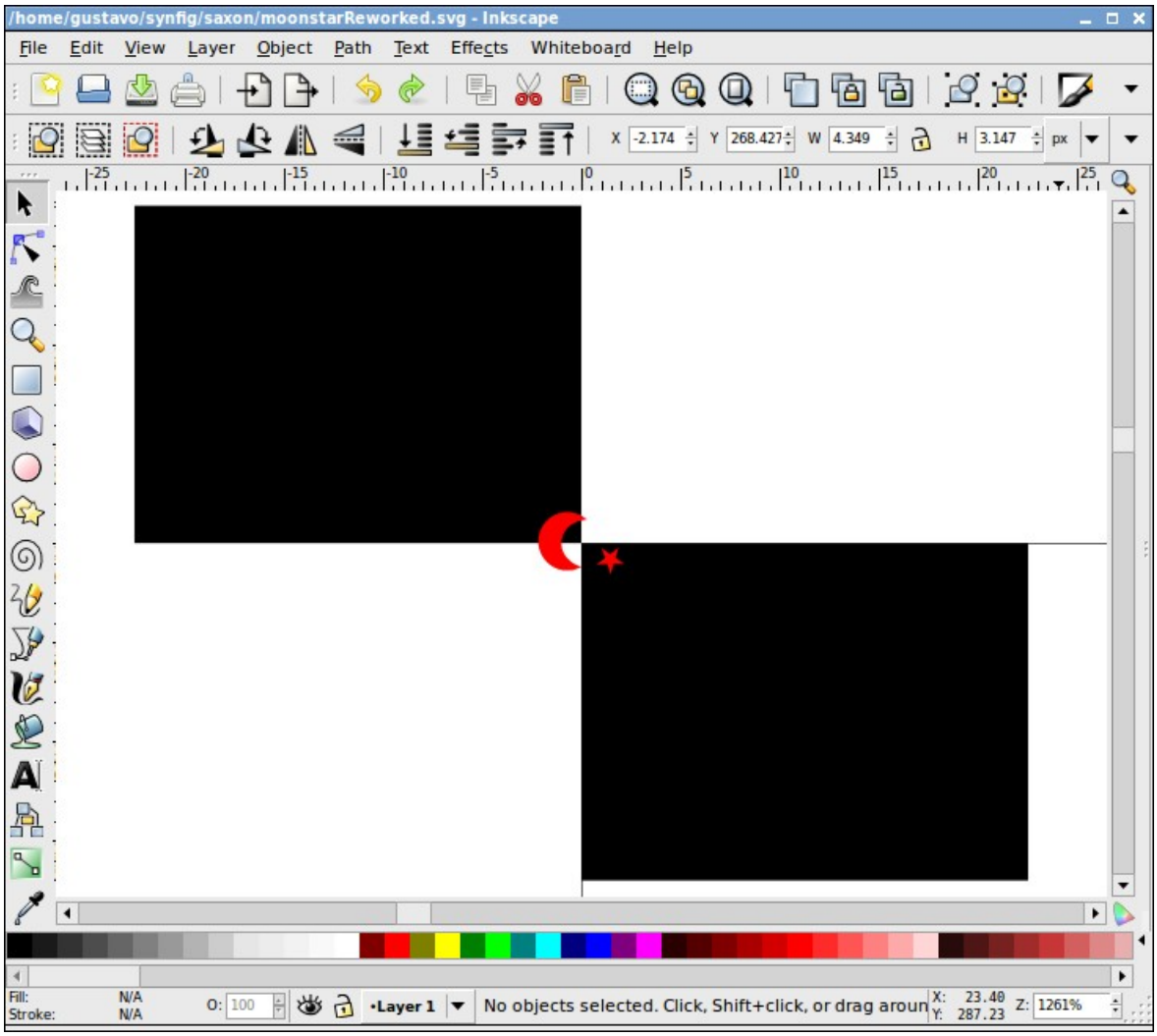
Inkscape rework

There are some simple steps to get rid of that resolution differences.

- Work with Inkscape document size at 480x270pt
- Select all the objects and scale at 1.35%
- Make an 'anchor' with two squares aligned at the top and left edges of the page, one inside and the other outside of the page. Combine or group after that. This is a registration point.
- Group and align the scaled objects at the center vertical and horizontal, of the anchor. Don't let the anchor move. Use the Inkscape Align options to do this.
- Mirror vertically the drawing and delete the anchor. Save the file.



The anchor are the black rectangles, no matter what size are, just intended to be at the origin of the SVG file coordinates, where rulers starts, and to help align the design to its center. See the objects selected, the moon and the star, at the 1.35% of its original size.



The anchor still is at the showed position, while the design is centered on it (positioned at 0 in x and y axes) and mirrored vertically. Now its time to delete the anchor and save the file.

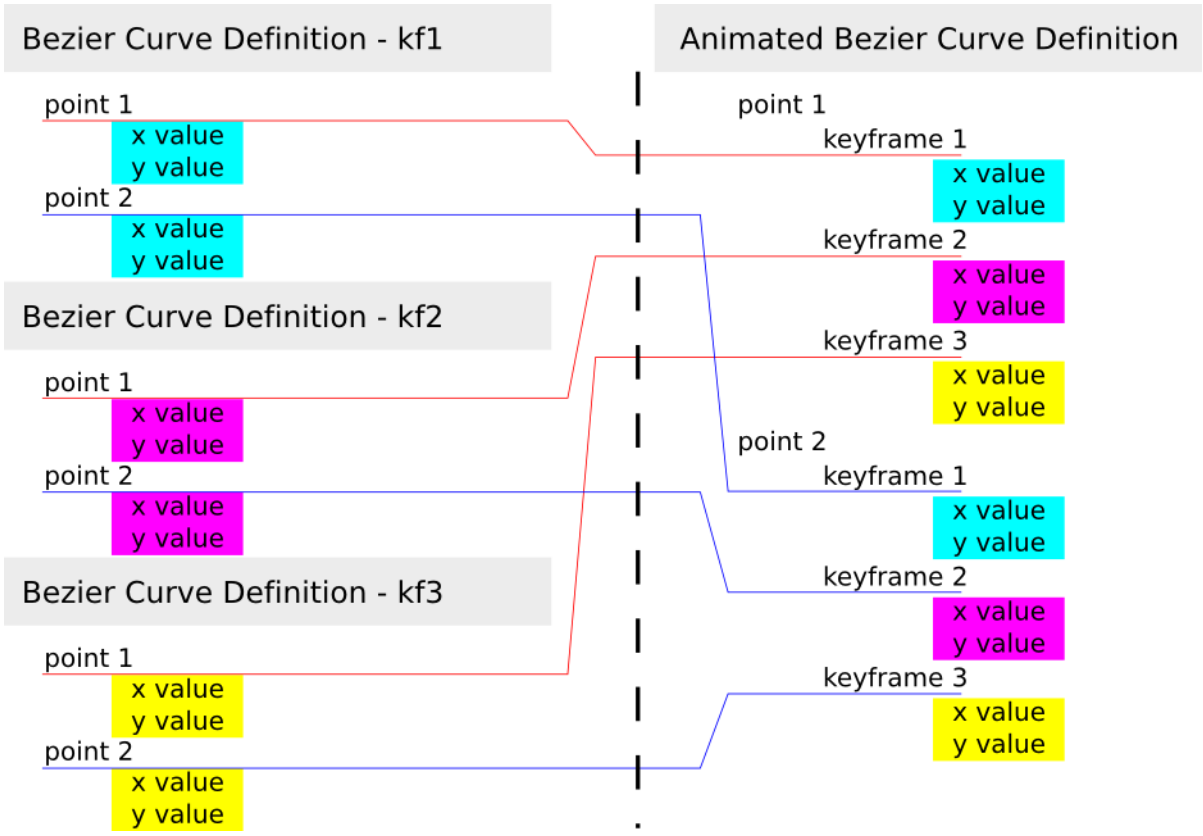
Doing this to the Inkscape file, will take to a conversion without resolutions mislead. You may open the created file in the conversion with Synfig, select all the layers converted an paste into another New Synfig file, with its defaults definitions. Save it in the uncompressed format, with .sif extension. Here ends the conversion.

Synfig XML File Specification

As far as I know, the Synfig file format is just a XML file. It implies a correct syntax, correct matching of start and end tags and some other rules to follow. Theres two flavors of it, the compressed file who extension is .sifz and the uncompressed file with the .sif extension. We need the uncompressed file, just to open with a text editor.

Looking through it, clearly can be seen the pattern to define a Bèzier curve, in a 0 frames file or still, and the pattern to define the same Bèzier curve during its development in time (keyframes).

The graphic bellow shows a simple example, of the general syntax of a 0 frame file with three Bèzier curves that becomes with some regular expressions transformations, into one Bèzier curve with three different coordinates (shapes) at three different points in time (keyframes).



Now, if a *n* number of nodes Bèzier curve is drawn in Inkscape, with some variations in its shape, to get an animation and later the conversion is made with the XSLT transformation sheet, the Synfig file obtained is a 0 frame with all the curve variations as different layers.

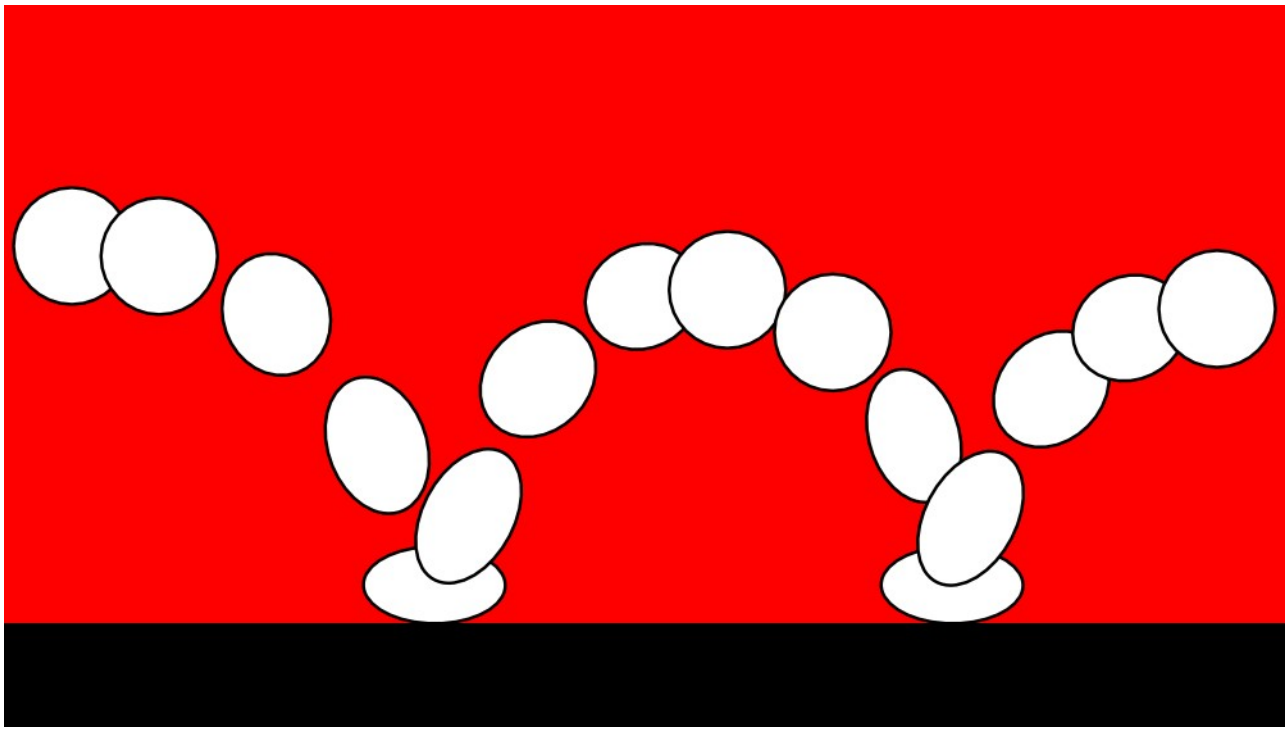
No work is saved in this way, with the conversion, because we need just one of that layers who change its shape in different keyframes.

With some steps we can get all that different Bèzier curves become one Bèzier curve moving through different keyframes, modifying some of the structure of the Synfig XML file, with the help of commands like *egrep* and with the *gvim* text editor.

Inkscape shapes to Synfig keyframes conversion

Lets try some method to make an Inkscape with many Bèzier curves that mandatory has the same number of nodes, with different shapes or dimensional coordinates, converted with the XSLT sheet, to an animation, with the use of some Regular Expressions transformations.

The file is the simple bouncing ball exercise, who layout was made in Inkscape. Prepared with the steps provided before, we convert and open in Synfig, with the undesired result of being a simple still frame.



This shapes, that exist in a page in Inkscape and in a frame in Synfig after the conversion, will become after the process, just one circle shaped Bèzier curve who changes its dimensional properties in the development of the new dimension, time. Note that outline of circles are just to illustrate overlapping.

The process

Here I will give you the idea, somehow a raw algorithm to get the job done. Each curve has a *name*, a *x-y* pair of coordinates, and a keyframe has a *value* gave by its position in time. With this data we will be working. In the numbered list is the pseudo code alike, and next the commands that execute the idea. The gray text explain what the commands do.

First we need to extract the path names and all the pairs of node positions of the recent converted file *(bounce.sif)* and save in a new file, say *bounceArrayVectors*, with the help of the egrep command:

```
~$ egrep '(desc=|<x>|<y>)' bounce.sif > bounceArrayVectors
```

Next, have to edit this created file with the data of the names of the paths and its coordinates in gvim an execute the next command sequence:

1. delete the first line
2. get rid of unwanted text, you may add some space at the beginning or maybe adjust the text to delete
3. deletes unwanted characters at the end of the line where name of path appears
4. deletes the definition of the 0 x-origin of the object
5. deletes the definition of the 0 y-origin of the object
6. deletes the <Spc> at the beginning of lines
7. deletes the empty lines remaining
8. write and quit

```
~$ gvim bounceArrayVectors
dd                                                                    // step 1
:%s/            <layer type="region" version="0.1" desc="//g          // step 2
:%s/">//g                                                             // step 3
%s/<x>0.0000000000<\/x>//g                                            // step 4
%s/<y>0.0000000000<\/y>//g                                            // step 5
:%s/ /g                                                               // step 6
:%s/^\n//g                                                            // step 7
:wq                                                                   // step 8
```

Now extract only the cleaned path names from that file and save under a new name, lets give it the name of *bounceArrayNames*

```
~$ egrep 'path' bounceArrayVectors > bounceArrayNames
```

As this curve has 4 points (nodes), edit the *bounceArrayNames* in gvim and insert the same file itself entirely, to reach the number of nodes, 3 times in this case. Here are the steps:

1. go to the end of file
2. open a new line and back to vi command mode
3. read and insert the same file at where cursor is
4. repeat as necessary
5. deletes the empty lines that maybe has been inserted
6. write and quit

```
~$ gvim bounceArrayNames
<Ctrl>+<End>
o<Esc>
:r bounceArrayNames     // repeat necessary times
:%s/^\n//g              // eliminates empty lines
:wq
```

Then we have to rearrange the *bounceArrayVector* file to meet the animated pattern described in the Synfig XML file specification above. To do that, we will create a stack with the names of the paths saved before, *bounceArrayNames*, in the file that contains the definitions of the coordinates of all the points, *bounceArrayVectors*. With a macro, we will rearrange, with the help of the names of the paths (that's why were saved repeated by the quantity of points of a Bèzier curve). Prepare it:

```
~$ gvim bounceArrayVectors
<Ctrl>+<Home>              // go to the first line
i                          // change to vi edit mode
<Enter>                    // open a blank new line
<Enter>                    // open another blank new line
<Up>                       // one keystroke of Up Arrow Key
unordered                  // insert the text unordered
<Ctrl>+<End>               // go to the last line
<Enter>                    // open a blank new line
ordered                    // insert the text ordered
<Enter>                    // open a blank new line
<Esc>                      // back to vi command mode
<Ctrl>+<Home>              // go to the first (empty) line
:r bounceArrayNames<Enter> // inserts file with the names of the curves
k                          // go up one line
dd                         // and delete it
:w                         // writes the file
```

And the steps to record the macro:

```
<Esc>
qa                  // starts to record the macro under the a key name
<Shift>+<End>       // selects the entire first line
<Crtl>+x            // cuts the line
dd                  // deletes the now empty line
/unordered<Enter>   // searches for the occurrence of string unordered
/<Crtl>+v<Enter>    // searches for a variable string, the one who was cut
j                   // go down one line
v                   // enter in visual mode
<Shift>+<Down>      // selects first line of text
<Shift>+<End>       // selects second line of text (a coordinates pair)
<Crtl>+x            // cuts the selection
dd                  // deletes the empty line
<Crtl>+<End>        // go to the end of file
o                   // open a blank new line
<Crtl>+v            // paste the yanked lines
<Crtl>+<Home>       // go to the first line
<Esc>               // back to vi command mode
q                   // ends the macro recording
```

This could be confusing and a very error prone method, it has to be automatized and enhanced. The first of 64 pairs of numbers is ordered. Now, with a simple command repeat 63 times the a macro... And the job is done for all the points!!!

```
<Esc>              // go to vi command mode if not in it
63@a               // repeat (63) times (at) commands saved in (a)
```

Wait few seconds (depends on complexity) and when the screen finishes the crazy scroll, the job is done. Then, do this:

```
<Ctrl>+<Home>       // go to the first line
v                   // enters vi visual mode
/^ordered<Enter>    // selects until ordered string is reached
<Del>               // deletes selection
2dd                 // delete two lines that remains, line with string ordered
                    // and a blank line
:wq                 // writes the file and quit
```

As you can see, there was left two strings in the file plus the one represented in the 'search for' and 'paste command' that act like a variable, read as the strings 'unordered', 'ordered' and the names of the paths that changes along the development of the algorithm.

At this point, all the x-y declarations of the still file are ordered with the pattern found in the animated file specification.

Now it's time to insert this bunch of meaningless pairs of coordinates into the XML Synfig file structure. To do that, again we need the the help of two another dummy files (provided) and the saved *bounceArrayVectors*.

Edit the timecode_dummy, this file has 120 keyframes, one by line. As we need 16 keyframe declarations, go to the line who contains it and delete till the end of the file. Save as *timecode*, its necessary for future actions.

```
~$ gvim timecode_dummy
16j                    // go down the n (n as keyframes) lines
<Ctrl>+<Shift>+<End> // select till the end of the file
<Del>                  // deletes the selection
dd                     // deletes the blank line that remains
:w timecode            // saves under the timecode name
:q!                    // mandatory quit, because current opened file name is
                       // timecode_dummy
```

Same as before, this curve has 4 points (nodes), edit the *timecode* in gvim and insert the same file itself entirely, till reach the number of nodes, 3 times in this case. Here are the steps to do this:

1. go to the end of file
2. open a new line and back to vi command mode
3. read and insert the same file at where cursor is
4. repeat as necessary
5. deletes the empty lines that maybe has been inserted
6. write and quit

```
~$ gvim timecode
<Ctrl>+<End>
o<Esc>
:r timecode            // repeat as necessary
:%s/^\n//g             // eliminates empty lines
:wq                    // write and quit
```

Now edit the file named *one_point_dummy*. As you see, each keyframe is defined with its value in time, in the line who looks like <*waypoint time="0s" before="auto" after="auto"*>. Delete the unwanted keyframes, till the <*!-- do not delete below this line* --> mark. The values that appears in its x and y declarations are useless, will be replaced with new values later.

Save under a new name, *waypoints*, and insert this last file itself the times as necessary to reach the numbers of points the curve has, don't forget to change the incremental mark as you do the insertions. Save with mandatory command with the same name as before, *waypoints*.

```
~$ gvim one_point_dummy
// delete the unwanted keyframes (from waypoint time="0.625s" in this case)
:w waypoints          // save as waypoints
<Ctrl>+<End>          // go to end of file
o<Esc>                // open a new blank line
:r waypoints          // insert the file itself
i<Left>2<Del>         // changes the incremental mark to 2 (and so on)
<Esc>                 // back to vi command mode, repeat as necessary
:w! waypoints         // write mandatory, waypoints
:q!                   // and mandatory quit
```

Lastly, it's time to edit the three files together and record a macro again to pass values between them, the middle file, *waypoints* is the only who will serve finally as a well formed Synfig Animation definition file.

```
~$ gvim timecode waypoints bounceArrayVectors
qw                      // begins macro record in w key name
<Shift>+<End>           // selects the first line
<Ctrl>+x                // cuts the first line
dd                      // deletes the empty line remaining
:wn                     // write changes and go to the second file
```

```
/<Ctrl>+v<Enter>       // search for paste command an put cursor there
:n                     // go to the third file
<Shift>+<End>+<Down>   // selects the first two lines
<Ctrl>+x               // cuts them
dd                     // deletes the empty line
:wp                    // write changes and go to the second file
j                      // go down one line
j                      // go down another one line
<Home>                 // puts the cursor in the start of that line
<Shift>+<End>+<Down>   // selects the current and below lines
<Ctrl>+v               // replaces them with paste command
:wp                    // write changes and go to the first file
q                      // ends macro recording
```

Now its time to repeat the macro the necessary times to make the job done. 63 times again.

```
63@w                   // executes the w macro 63 times
:q                     // quits gvim session
```

You will end with the file named *waypoints* with the Bèzier curve correctly ordered by each of its point in temporal development. It's time to edit the *waypoints* and *50_point_dummy.sif* together.

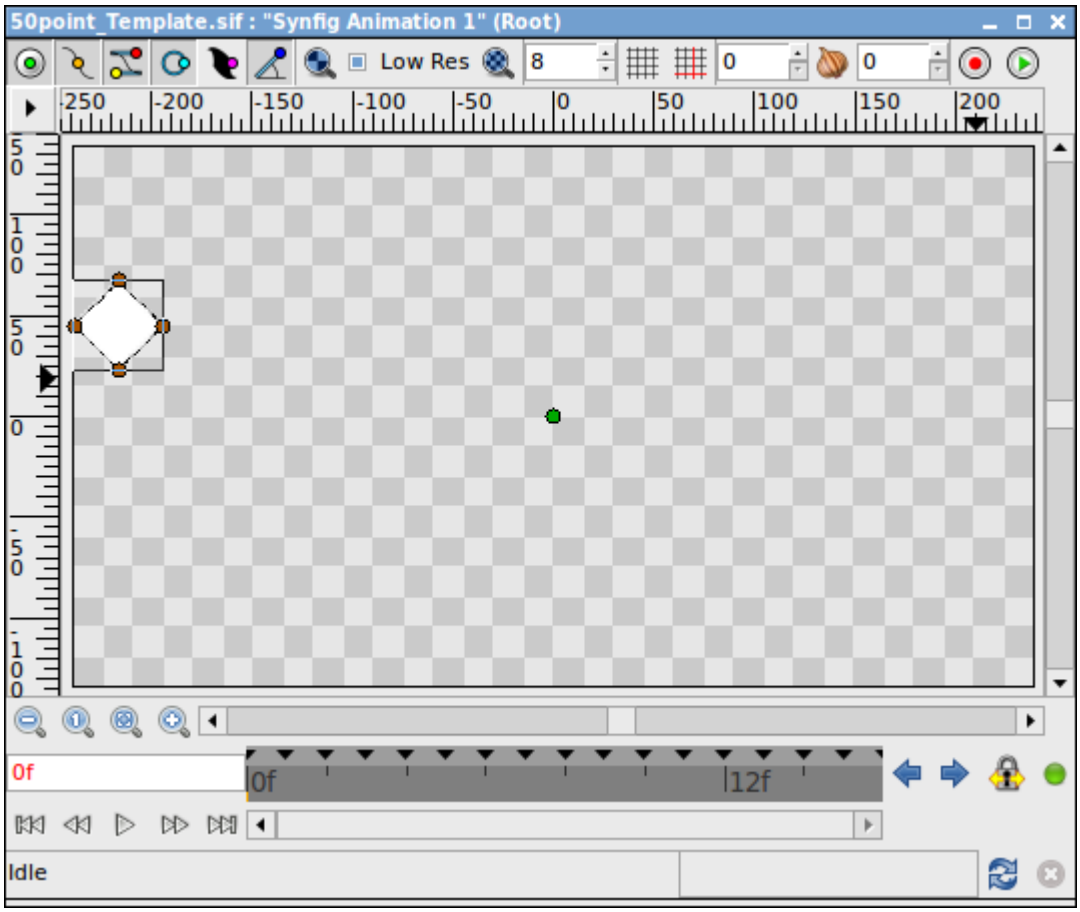
Put the cursor in the line under the *<!-- point 1 -->* mark, change to visual mode and do a search for the string *-- point*, adjust the selection and cut, save the file and go to the next file, look where it indicates to paste the definitions for point 1, select these lines:

```
//        <entry>
//          <!-- Paste here definitions for point 1 -->
//        </entry>
```

and paste the lines that were cut. Repeat with the rest of the points.

```
~$ gvim waypoints 50_point_dummy.sif
// put the cursor under the line with the <!-- point 1 --> mark
v           // enter vi visual mode
/--point    // finds the next point definition
// adjust with the directional arrow keys the selection as necessary
<Ctrl>+x    // cuts the selection
:wn         // save and go to the next file
// look and select the three lines mentioned above and replace with paste
// command
:wp         // save and go to the previous file, then repeat as necessary
```

There's some lines that you may delete from the *50_point_dummy.sif* file, some kind of cosmetics to the file, as in the second line, declare the exact time of the animation, in seconds and frames, or just frames, as in this case, and below of that, delete the unnecessary declared keyframes.



As you can see in the screenshot of *50_point_dummy.sif*, there is a rhomboidal shape and not a circle. There's because all its tangents were not converted in this last process, but if you give it the correct shape it will be converted in all the remaining keyframes, or just have to adjust in few of the 16 keyframes.

Hope to be clear, it's my first tutorial. I complete all this process a few times with some different designs, and all went right. Although I'm not a programmer, I think that is a time saving method, even with a simple example like this one. Error prone as said, but you have to understand what is been doing in each step.

I will try to enhance all the process (more automatized) and implement a fully tangent conversion, with my little programming knowledge. Write for any question. Hope you like it.

Please make your test and give feedback ! ! !