Synfig Documentation

Release 1.4.0

Synfig Community

May 19, 2020

Contents:

CHAPTER 1

Installation

1.1 Install Synfig on Windows

CHAPTER 2

Interface

CHAPTER 3

Main concepts

CHAPTER 4

Importing your artwork

4.1 Importing images

4.2 Importing image sequences

4.3 Vectorization of Bitmaps

Synfig Studio comes with a feature of Vectorization of Bitmaps. In this tutorial, we will learn what is Vectorization feature and how to use it. :)

4.3.1 What does Vectorization mean?

There are two common ways to represent a two-dimensional image:

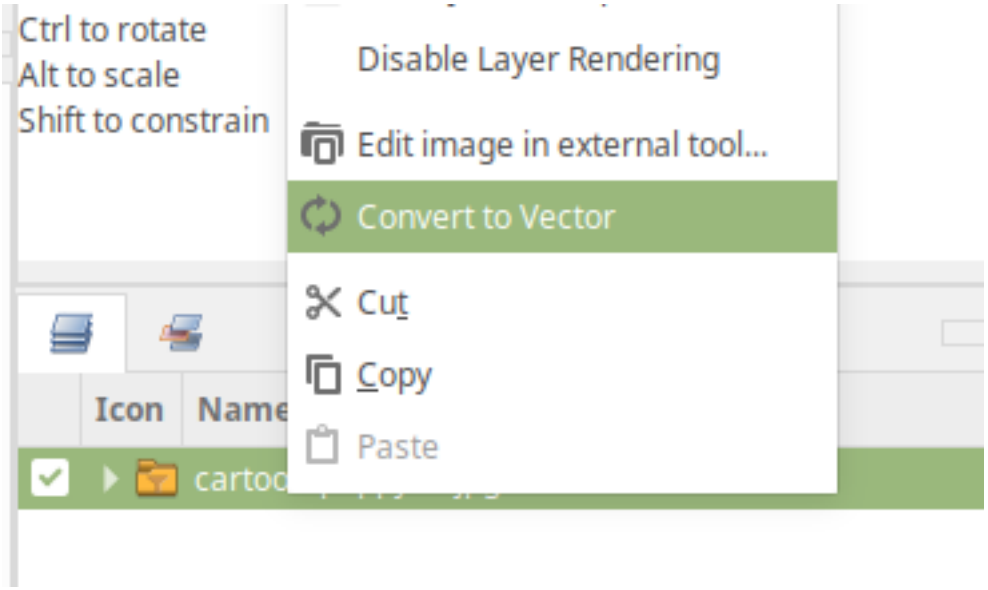
- Raster Graphics or Bitmap Graphics
- Vector Graphics

In Bitmap Graphics images are stored as a two-dimensional grid of pixels. This has some problems for example if a bitmap image is scaled it becomes blurry and pixelated.

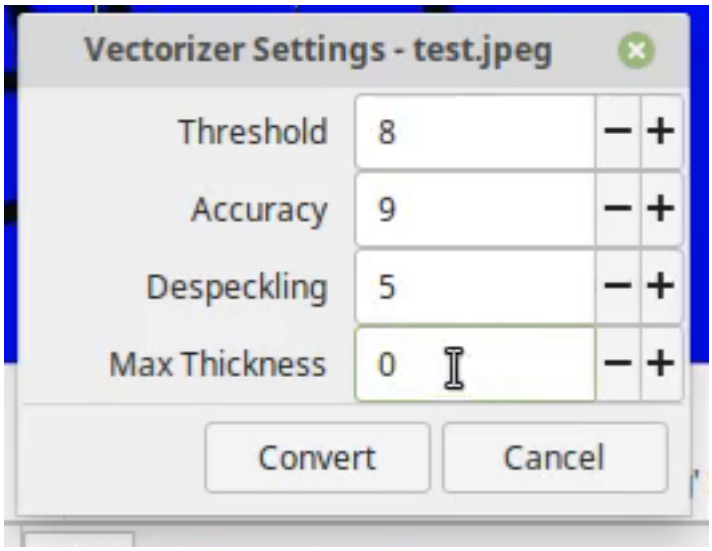
We use Vector Graphics to solve this problem. For converting Bitmap image to Vector Image, Synfig uses vectorization algorithm from OpenToonz

4.3.2 Usage

Upload the image and right-click on the Image Layer and select “Convert to Vector” command.



Clicking on this "Convert to Vector" will open a dialog box with the following settings:



- **Threshold** - sets the value of the darkest pixels to be taken into account to detect lines to be converted to vector
- **Accuracy** - sets how much the vector stroke will follow the shape of the original drawing lines. High values create more precise strokes but makes them more complex.
- **Despeckling** - ignores during the conversion small areas generated by the image noise; the higher the value, the larger the areas ignored.
- **Max thickness** - sets the maximum vector stroke thickness; if this value is low very thick lines will be converted in two centerline strokes defining the line outline; if this value is high, they will be converted in a single centerline stroke.

As soon as the "Convert" button is clicked, the image is converted into a vector which contains all the outlines of the image.

Note: Synfig currently supports only centerline vectorization

4.4 Importing SVG

4.5 Importing videos

CHAPTER 5

Tools

5.1 Spline Tool

CHAPTER 6

Layers

6.1 Blur

CHAPTER 7

Blend Methods

7.1 Composite

CHAPTER 8

Converters

8.1 Composite

CHAPTER 9

Configuration

CHAPTER 10

Export

10.1 Export Animation

10.2 Export for Web (Lottie)

10.3 Exporting images

CHAPTER 11

Plugins

11.1 Summary

The Plugins feature allows to run custom python scripts directly from Synfig Studio menu. Each script takes .sif file as first argument and should modify its contents in some way. After script execution finished, the file is automatically reopened back in Synfig Studio.

All plugins are located in the "Plug-Ins" submenu of the canvas.

11.2 Rationale

People often write some scripts to make useful things on Synfig (sif) files. The most of these scripts are written in python. But for ordinary users running custom scripts from terminal is tricky. With plugins feature user can install scripts as easy as they copy files and transparently run them in the same way as they use standard Synfig Studio commands. Also, runing scripts from menu is much faster than from terminal and it greatly improves the workflow for advanced users. Having this feature allows to easily add simple functions to Synfig Studio by writting scripts in python.

11.3 How to install plugins

Plugin is a directory, containing the python script (*.py file), plugin.xml and maybe some other files if they are required by python script.

To install the plugin user should copy its directory into the following location:

- **Windows Vista/7/8:** C:\Users\USERNAME\Synfig\plugins
- **Windows XP:** C:\Documents and Settings\USERNAME\Synfig\plugins
- **Linux:** ~/.config/synfig/plugins
- **OSX:** /Users/USERNAME/Library/Synfig/plugins

The system-wide location for the plugins is USER_DIRECTORY/SYNFIG_CONFIGURATION_DIR/plugins

11.3.1 Where to find plugins

- Yoyobuae:FreeForm Deform
- Morevna:Plugins feature in Synfig Studio
- Berteh:Import Labels and Timings
- Synfig forum: scripts/plugin thread

11.4 Plugin structure

A plugin keeps all its files inside a directory as described above.

Synfig parses the file called plugin.xml found in each plugin directory, which defines plugin metadata and how to invoke the script.

This section describes the xml elements available and what they do.

11.4.1 <plugin>

Root element. Can contain the following:

- <name> The name of the plugin, must have at least the default version
- <exec> Script to run when the user clicks on the Plugin menu. Can be omitted if there are importers or exporters
- <exporter> Defines an exporter, a plugin can have multiple of these
- <importer> Defines an importer, a plugin can have multiple of these

11.4.2 <name>

Name of the plugin, can be specified multiple times to provide translations

Example:

```
<name>This is the default name</name>
<name xml:lang="it">This is name it will show if you set Italian as your language</
↪name>
```

11.4.3 <exec>

Defines a script to run, the text contents must be a path (relative to the plugin directory) of a script to run.

If present inside <plugin>, the plugin will be shown in the Plugins menu, and the script will be invoked when you click on the corresponding menu item.

It has a few attributes, all optional.

type **python** selects the interpreter (currently only Python is supported)

stdout **ignore** What to do with the script standard output:

- **ignore** output is discarded
- **log** the output is shown in the Synfig log
- **message** an error message is shown to the user

stderr **message** Same as above, but for standard error

Example:

```
<exec>myscript.py</exec>
```

Changing stream behaviour:

```
<exec stdout="log" stderr="ignore">myscript.py</exec>
```

11.4.4 <exporter>

Defines a new exporter, used to convert synfig files into other formats.

A plugin can define multiple exporters.

Exporters will be shown in the Export dialog.

The exporter contains the following sub-elements:

- <exec> must have exactly one of these
- <extension> must have at least one of them
- <description> must have the default version

Example:

```
<exporter>
   <extension>svg</extension>
   <extension>svgz</extension>
   <description>Scalable Vector Graphics (*.svg, *.svgz)</description>
   <exec>svg-exporter.py</exec>
</exporter>
```

11.4.5 <importer>

Works the same as <exporter>, but provides script to convert from other formats into synfig.

A plugin can define multiple importers.

Importers will be shown in the Open file dialog.

Example:

```
<importer>
   <extension>svg</extension>
   <extension>svgz</extension>
   <description>Scalable Vector Graphics (*.svg, *.svgz)</description>
   <exec>svg-exporter.py</exec>
</importer>
```

11.4.6 <extension>

For <importer> and <exporter>, which extensions are supported.

Multiple <extension> elements may be present in an importer or exporter (at least one is required)

Example:

```
<extension>svg</extension>
```

11.4.7 <description>

For <importer> and <exporter>, the text to be shown in the file dialog.

Similarly to <name> this can be translated using xml:lang

Example:

```
<description>Scalable Vector Graphics (*.svg, *.svgz)</description>
```

11.5 Script Invocation

11.5.1 Plugins Menu

For scripts run when the user clicks on the plugin name in the Plugins menu, synfig will save a copy of the open canvas and pass the path to that file as the argument to the script.

The script can then modify that file and synfig will reload the canvas to reflect any changes.

11.5.2 Exporter

For an exporter, synfig will pass two arguments to the script: the first is the path to a synfig file containing the open canvas; the second is the file name.

11.5.3 Importer

For an importer, synfig will pass the file selected in the open dialog as first argument, and the path to a temporary synfig file as second argument.

Once the script is completed, synfig will load that second file, so the plugin script should populate it appropriately.

11.6 Tutorial

11.6.1 Details

Each plugin located in a separate subdirectory with unique name. The part of the name before first "-" symbol is used to set the group plugin belongs to (not implemented yet). The main information about plugin (plugins name and script to execute) is stored in the plugin.xml file. It's self-explanatory :

plugin.xml :

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin>
<name>Unhide All Layers</name>
<name xml:lang="es">Activa todas las capas</name>
<name xml:lang="eu">Erakutsi geruza guztiak</name>
<name xml:lang="eu_ES">Erakutsi geruza guztiak</name>
<name xml:lang="fr">Afficher Tous les Calques</name>
<name xml:lang="lt">Parodyti visus sluoksnius</name>
<name xml:lang="ru">   </name>
<exec>view-unhide-all-layers.py</exec>
</plugin>
```

view-unhide-all-layers.py :

```
#!/usr/bin/env python

#
# Copyright (c) 2012 by Konstantin Dmitriev <k....z...gmail.com>
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.

import os
import sys

def process(filename):

    # Read the input file
    inputfile_f = open(filename, 'r')
    inputfile_contents = inputfile_f.readlines()
    inputfile_f.close()

    # Now write results to the same file
    inputfile_f = open(filename, 'w')

    for line in inputfile_contents:
        if "<layer " in line:
            inputfile_f.write(line.replace(' active="false" ',' active="true" '))
        else:
            inputfile_f.write(line)
    inputfile_f.close()

if len(sys.argv) < 2:
    sys.exit()
else:
    process(sys.argv[1])
```

All scripts are interpreted with python 3.

In Linux and Mac OSX case Synfig Studio looks for “python” or “python3” binary. For windows case Python binary is expected at INSTALL_PREFIX/python/python.exe. New environment variable SYNFIG_PYTHON_BINARY allows to set custom path to the python 3 binary.

CHAPTER 12

Release Notes

12.1 Development Versions

12.1.1 Synfig 1.3.14 Release Notes

Improvements

- Show interpolation type directly in waypoint's context menu (see details).

Bugfixes

- Fixed crash when removing a Spline vertex using "Remove item (Smart)" (issue #1102).
- Fixed incorrect placement of width points on outline when loading old files (see details here and here).
- Fixed crash when Shade Layer has no sublayers (issue #1272).
- Fixed popup menu disappearing right after button release for Widget Gradient/Spline (issue #1274).
- Fixed hang when opening a second .sif file from Explorer on Windows (issue #291).
- Fixed behavior "Local Time" parameter of Time Loop layer (issue #479).
- Fixed wrong percentage displayed when exporting a subset of frames (issue #1304).
- Fixed crash when undoing deletion of Group Layer (issue #1070).
- Fixed TimeTrack not updating when new waypoints added to bone (issue #1342).
- Fixed importing of 16-bit PNG files (issues #1160 and #1371).
- Fixed some memory leaks (PR #1292, #1293, #1319).

12.2 Stable Versions

CHAPTER 13

About this Manual

The Synfig Manual is a community driven effort to which anyone can contribute. Whether you like to fix a tiny spelling mistake or rewrite an entire chapter, your help with the Synfig manual is most welcome!

13.1 License

13.2 Contributors

- Mattia Basaglia (mbasaglia)
- Jerome Blanchi (D.j.a.y)

CHAPTER 14

Indices and tables

- genindex
- modindex
- search