Inkscape: Guide to a Vector Drawing Program

Get the book.

English, Française, Español

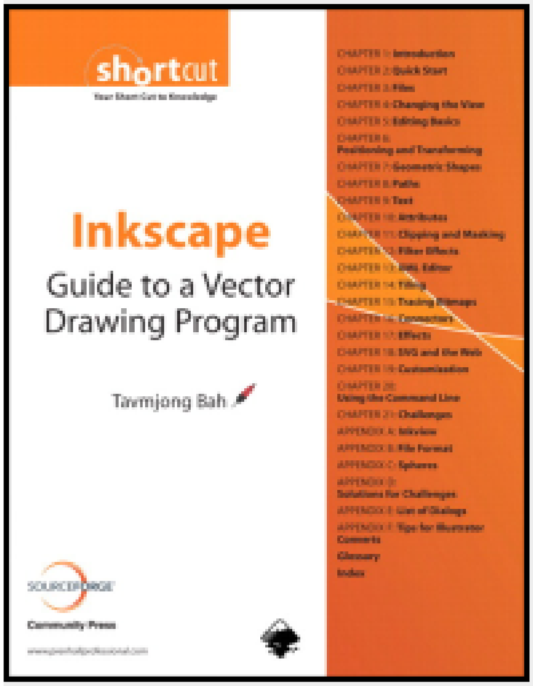
Index

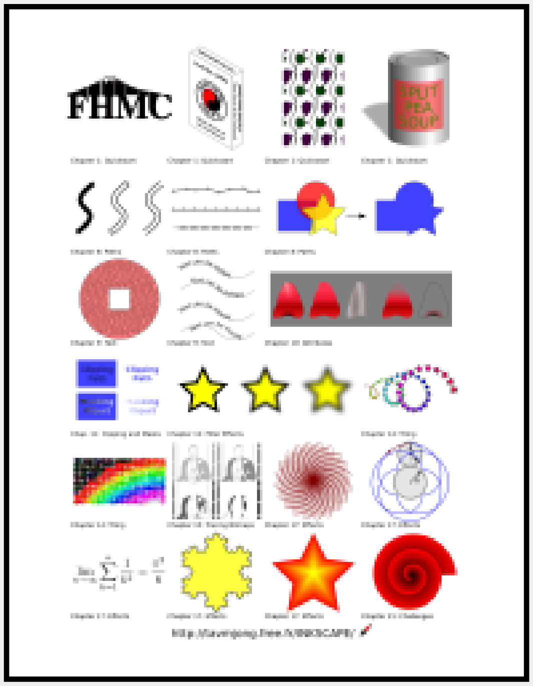
Inkscape

Guide to a Vector Drawing Program

2nd Edition

Tavmjong Bah


shortcut
Your Short Cut to Knowledge
Inkscape
Guide to a Vector Drawing Program
Tavmjong Bah
SOURCEFORGE
Community Press



0.14 (Documenting version 0.46)

Available in PDF form from SourceForge Community Press.

This book has a website with additional material. Go to: http://tavmjong.free.fr/INKSCAPE/.

Suggestions and corrections are welcome and can be sent to tavmjong @ free . fr.

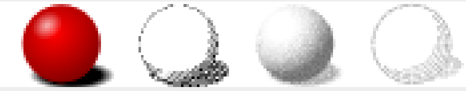
Abstract

Inkscape, Guide to a Vector Drawing Program is **THE GUIDE** to the Inkscape program. The web-based version is linked directly to under the program's Help menu. This book is both an introduction and reference for the Inkscape drawing program. With Inkscape, one can produce a wide variety of art, from photo-realistic drawings to organizational charts. Inkscape uses *SVG*, a powerful vector-based drawing language and W3C web standard, as its native format. *SVG* drawings can be directly viewed by web browsers such as Firefox and Opera. A subset of *SVG* has been adopted by the mobile phone market. Inkscape is available free for Windows, Macintosh, Linux, and Solaris operating systems. The first third of the book is devoted to eight tutorials that progress in difficulty from very basic to very complex (three additional tutorials were written explicitly for the PDF version). The remainder of the book covers each facet of Inkscape in detail. Updated for Inkscape v0.46, the book includes complete coverage of new features including: SVG filters, "Live Path Effects", the 3D box tool, and the Tweak Tool. Advance topics covered include the use of Inkscape's powerful tiling tool, built-in bitmap tracing, and *SVG* use on the web. The book includes plenty of tips (and warnings) about the use of Inkscape and *SVG*.

Table of Contents

Acknowledgments

Get the book.



Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Quick Start

← Index →

Chapter 1. Quick Start

Table of Contents

Let's get started. Inkscape is a very powerful program. However, you need to understand only a small part of it to begin drawing. This section gives you an overview of parts of the Inkscape user interface and then leads you through the creation of a few drawings. We will use a number of examples:

- Swedish Flag: A basic introduction to Inkscape using simple rectangles.
- European Flag: Includes drawing stars and precisely placing objects.
- Hiking Club Logo: Introduces text and is a serious foray into paths.
- Northern Pacific Railway Logo: Shows how to create a drawing from a photograph with

the help of the auto-tracing routine. Layers are also introduced.

- Box of Playing Cards: Shows how to use Inkscape to draw a simple isometric projection of a three-dimensional object. It utilizes precise transformations of objects.
- Can of Soup: Another demonstration of how to simulate a 3D object in a 2D drawing. It introduces gradients.
- Vine Design: Demonstrates how to create a pattern that can be used as a base tile for a repeating pattern. It introduces the powerful *Create Tiled Clones* feature of Inkscape.
- Neon Sign: Introduces using Inkscape for animation.
- Bank Note: Uses a variety of Inkscape features to produce a "secure" bank note. Patterns and scroll work are featured.
- Bottle: Creates a photorealistic drawing of a spritzer bottle. Tracing, gradients, and blurring are used.

Help

Table of Contents

The Anatomy of the Inkscape Window

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Files

Index

Chapter 2. Files

Table of Contents

This section covers the manipulation of the files that are used by Inkscape. This includes the files for storing your drawings in the Inkscape format, and for importing and exporting drawings in other formats. All the commands to manipulate files can be found under the File menu. Several of the commands can be also be found in the *Command Bar* (New-default, Open, Save, Print, Import and Export Bitmap).

A Bottle - Photorealism

Table of Contents

Opening and Saving Files

Get the book.

Inkscape » Changing the View

Index

Chapter 3. Changing the View

Table of Contents

The view presented by the Inkscape window can be changed in many ways. The changes can be divided into two types: those that change the way drawings appear and those that change how the Inkscape interface appears. A few of the latter type of changes are covered in other chapters such as Chapter 5, *Positioning and Transforming*, for *Grids* and *Guide Lines*, and Chapter 20, *Customization*, for changing parts of the *GUI*.

An Inkscape drawing can be viewed in many different ways. The view can be changed by panning and by zooming the canvas. The Inkscape window can be made full screen. Multiple views of the same canvas are possible. An *Outline* or *Wire-frame* mode is available as well as a *Icon Preview* window.

Vacuuming Files

Table of Contents

Panning the Canvas

Get the book.

Chapter 4. Editing Basics

Table of Contents

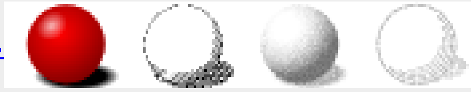
We discuss here some of the basic editing concepts of Inkscape, including: undoing and redoing changes, selecting objects, copying and deleting objects, and grouping objects. It is assumed that you have some familiarity with Inkscape by having worked through some of the examples in Chapter 1, *Quick Start*, or through the tutorials included with Inkscape (Help → Tutorials).

Miscellaneous View Commands

Table of Contents

Undo and Redo

Get the book.



Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Positioning and Transforming

← Index →

Chapter 5. Positioning and Transforming

Table of Contents

Inkscape has a variety of ways to position objects as precisely as required. These include onscreen *Grids* and *Guide Lines* where objects “snap” into alignment, dialogs for moving individual objects or for aligning multiple objects, and the *Create Tiled Clones* dialog for placing multiple clones of an object. Additional features allow you to scale, skew, or rotate an object. This section will begin with a discussion of the coordinate system of Inkscape followed by a discussion of the way Inkscape describes object transformations. Then the commands and dialogs for transforming objects are discussed.

One key thing to know is that transforming a *Regular Shape* object or a *Group* of objects by the methods described in this chapter does not (usually) change the underlying description of the object(s). For example, suppose you have an ellipse that is 100 pixels wide but you need it to have a width of 50 pixels. There are two different ways to achieve the required width. The first is to scale the object by 50% in the horizontal (*x*) direction. The underlying definition of the ellipse width remains 100 pixels but when the ellipse is drawn a scale factor of 50% is applied in the horizontal direction. The second way to change the ellipse is to use the *Ellipse Tool* to resize the ellipse. In this case, the underlying description of the ellipse changes and no scale factor is applied.

An exception to this rule is for *Rectangle* objects. Inkscape will attempt to change the description of the rectangle itself when a simple transformation is applied. This behavior can be changed under the *Transforms* tab in the *Inkscape Preferences* dialog (File → Inkscape Preferences... (**Shift+Ctrl+P**)). Change the *Store transformation* parameter to *Preserved*.

Note that in *Paths*, the individual points are transformed (unless the *Store transformation* parameter has been changed as above).

Layers

Table of Contents

Inkscape Coordinates

Inkscape » Geometric Shapes

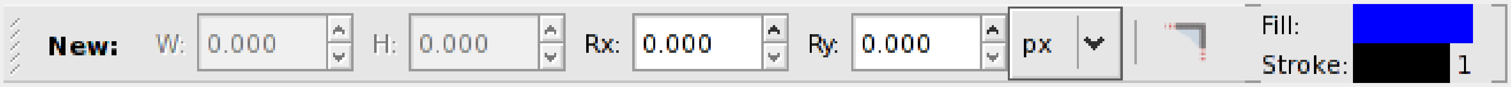
← Index →

Chapter 6. Geometric Shapes

Table of Contents

Inkscape provides a number of tools for drawing geometric shapes. The tools for drawing regular[10] geometric shapes (rectangles, boxes, ellipses, regular polygons, stars, and spirals) are covered here. Path (pencil and pen) tools, discussed in the next section, may be used to draw arbitrary shapes.

Current Style: The style of an object includes attributes that determine how the inside of the shape (*fill*) and how the boundary path (*stroke*) are drawn. It also includes shape-specific attributes such as the number of points in a star. New objects are drawn with the *Current style*. The *Fill* and *Stroke paint* colors as well as the *Stroke* thickness of the *Current style* are shown for the shape and path tools at the right end of the *Tool Controls*.



New: W: 0.000 H: 0.000 Rx: 0.000 Ry: 0.000 px

Fill: 🟦 Stroke: ■ 1

The *Rectangle Tool-Tool Controls* showing the current *Fill* color (blue) and *Stroke* color (black), as well as the *Stroke* width (1 px).

A component of the *Current style* is changed when that component is modified through, for example, an entry in the *Tool Controls* or the *Fill and Stroke* dialog (Object → Fill and Stroke... (**Shift+Ctrl+F**)) discussed in Chapter 9, *Attributes*. Note that if in the *Current style*, a star has five points, just selecting a star with six points (and even modifying its color) is not enough to change the number of points in the *Current style*. The number of points must be explicitly changed.

By default, the *shape tools* (except the *Spiral Tool* and *Box Tool*) as well as the *Calligraphy Tool* are drawn with a global *Current style*. Changing the style for one of these tools, changes the style for all.

Each of the shape tools can be given a fixed style by selecting the *This tool's own style* option under the tool's entry in the *Inkscape Preferences* dialog (File → Inkscape Preferences... (**Shift+Ctrl+P**)). Clicking on the *Current style* color swatches will open the correct section of the *Inkscape Preferences* dialog. Set the style by selecting an object with the desired style and click on the *Take from selection* button. The *Box Tool* style can not be changed. (However the *Box Tool* with the *Last used style* option will remember its own style.)

While drawing some objects (arcs, stars, regular polygons, and spirals), some features (such as the orientation of a polygon) can be constrained to specific angles with respect to the center of the shape and the horizontal axis. These angles are multiples of the *Rotation snap angle*. The default snap angle is 15 degrees. It can be set under the *Steps* entry in the *Inkscape Preferences* dialog.

Shapes can be scaled, rotated, and skewed. (See Chapter 5, *Positioning and Transforming*.) When doing so, a transformation is applied to the shape. The internal parameters defining the shape (such as the width and height of an ellipse) remain *unchanged*.[11] This is important to remember if you later modify a shape, for example, by editing the *XML* file directly.

[10] The *Star Tool* includes a randomization feature so that the resulting shapes are not regular. The underlying description is still based on a regular shape.

[11] This is not always true for rectangles. If the option *Optimized* is selected in the *Store transformation* section of the *Transforms* entry of the *Inkscape Preferences* dialog, the *x*, *y*, *width*, and *height* attributes will change rather than adding a *transformation matrix* for simple translating and scaling operations.

Alignment and Distribution of Objects | Table of Contents | Rectangles and Squares

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Paths

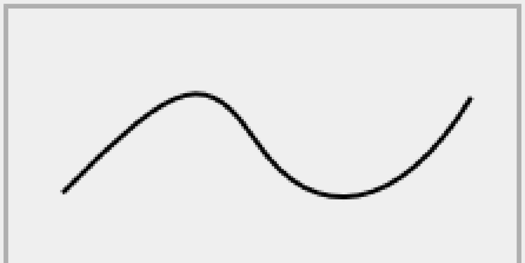
← Index →

Chapter 7. Paths

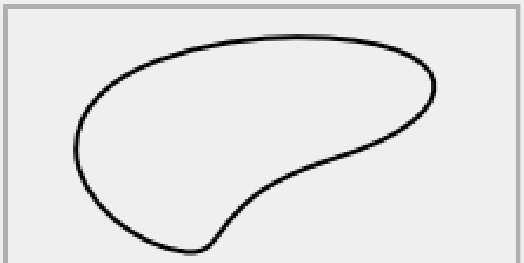
Table of Contents

Paths are arbitrary shaped objects. This chapter first covers some path terminology and how paths are described in Inkscape, then moves onto how paths can be created, and finally how paths are edited.

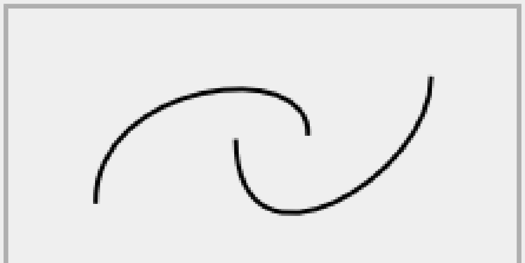
Paths can be *Open* (have two ends) or *Closed* (have no ends). They can also be *Compound* (composed of separate open and/or closed paths).



An Open path.

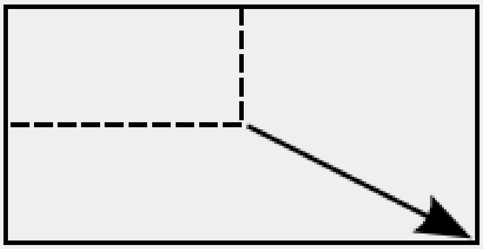


A Closed path.

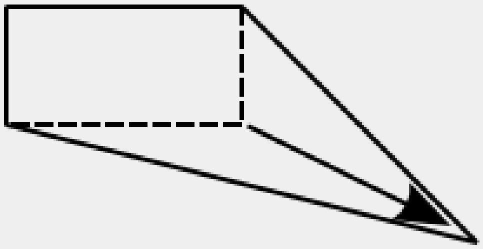


A Compound path.

Paths differ from *Shapes* in that there is no predefined structure. For example, a *Rectangle* shape is defined in terms of a width and height with an *x* and *y* offset. A corner point can not be moved independently of at least one other corner point. A path, in the shape of a rectangle, consists of the coordinates of the four corner points. A single corner point can be moved by itself with the resulting shape no longer rectangular.



A *Rectangle* shape, shown before (dashed line) and after (solid line) a corner has been dragged.



A rectangular path, shown before (dashed line) and after (solid line) a corner has been dragged.

A regular shape can be converted into a path: Path → Object to Path (**Shift**+**Ctrl**+**C**), but the reverse is not possible.

Spirals

Table of Contents

Bezier Curves

Inkscape » Text

Index

Chapter 8. Text

Table of Contents

Inkscape has a sophisticated system for creating and manipulating text. Text strings can include **Bold** or *Italicized* substrings, and changes in font type and size. Text can be justified on the right and/or left. It can be horizontal or vertical. Individual characters can be kerned. Text can be put on a path or flowed into an arbitrarily shaped path.

This is a text string with **Bold** and *Italicized* text.

Some sample text.

There are three types of text objects in Inkscape. The first is *regular* text. The second is *flowed* text; this is a text object that includes a rectangular frame. The third is *linked-flowed* text. This is a text object where the text is flowed into a separate arbitrary shape or path object(s). It is discussed at the end of this chapter. When a text object is selected, its type is shown in the *Notification Region*.

Live Path Effects (LPE)

Table of Contents

Creating Text

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Attributes

← Index →

Chapter 9. Attributes

Table of Contents

An object has attributes such as color and line style. An attribute can apply to the *Fill* or to the *Stroke* of the object. The *Fill* refers to how the area inside an object's boundary path is painted while the *Stroke* refers to the path itself. The *Fill* or *Stroke* (*Stroke paint*) can be a single color, a *Gradient* of colors, a *Pattern*, or nothing at all. With the exception of a few small differences, *Fill* and *Stroke paint* have the same properties and are treated together in the following discussion. A *Stroke* can have additional attributes such as width, dash pattern, and arrow type (*Stroke style*). These are treated in a separate section.

Text can be given the same attributes as other objects with a few small differences: Individual letters can be given different solid colors, but *Gradients* and *Patterns* must be assigned to an entire text object.

There are a number of ways to change attributes:

- *Fill and Stroke* dialog: Three tabs in the dialog allow the setting of the *Fill*, *Stroke paint*, and *Stroke style*. At the very bottom of the dialog is a slider and entry box for the *Master opacity* to set the overall *opacity* (or *transparency*) of an object.
- *Palette*: To change *Fill* and *Stroke paint* colors.
- *Swatches* dialog: To change *Fill* and *Stroke paint* colors (View → Swatches... (**Shift**

+Ctrl+W)).

- *Style Indicator* menu (pops up when a bar in the *Style Indicator* is **Right Mouse Clicked**). An entry box in the *Style Indicator* can be used to set an object's *opacity*.
- *Dropper Tool*: To select *Fill* and *Stroke paint* colors from another object.
- *Gradient Tool*: To create and modify *Gradients*.
- Edit → Paste Style (**Shift+Ctrl+V**) command: To copy attributes from one object to another. Copy the source object (Edit → Copy (**Ctrl+C**)), then select the target object and use the Edit → Paste Style (**Shift+Ctrl+V**) command.
- *XML Editor*: Useful to access attributes defined in the *SVG* standard but are not yet directly accessible through the Inkscape interface.

Text in a Shape

Table of Contents

Fill and Stroke Paint

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Tweak Tool

← Index →

Chapter 10. Tweak Tool

Table of Contents

New in v0.46.

The *Tweak Tool* is used to make small changes to paths and colors. While at first paths and colors may seem to have little to do with each other, the use of *Tweak Tool* to edit them is surprisingly very similar.

To use the *Tweak Tool* select the tool from the *Tool Box* ( ) or use either of the shortcuts: **W** or **Shift**+**F2**.

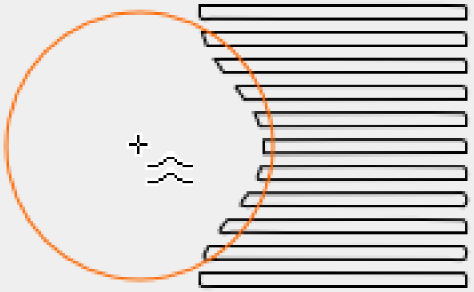
The *Tweak Tool* works like a *brush* that covers a circular part of the screen, indicated by an orange circle. The affect of the brush is strongest in the center and falls off smoothly till the edge. Two parameters, located in the *Tool Controls* affect the “physical” nature of the brush:

- ***Width***: Determines the size of the brush. The range is from 1 to 100, where 20 corresponds to a radius of 100 screen pixels. As the brush width is independent of zoom, zoom can be used to quickly change the size of the brush relative to the size of an object. The **Left Arrow** and **Right Arrow** keys can be used to decrease and increase the width at anytime. **Home** sets the *Width* to 1 while **End** sets it to 100.

- ***Force*:** Determines how strongly a movement of the brush affects the objects on the screen. The range is from 1 to 100. If the “Use Pressure” button (⤓) is toggled on, a pressure-sensitive tablet can be used to control the force; maximum pressure corresponds to the *Force* parameter setting. See the *Calligraphy Tool* section for use of a tablet. The **Up Arrow** and **Down Arrow** keys can be used to decrease and increase the (maximum) *Force*.

The *Tweak Tool* has a number of modes for editing paths and colors. The mode is selected by clicking on the corresponding icon in the *Tool Controls* or using a keyboard shortcut. Each mode has its own cursor. The various modes are discussed in the next two sections.

Objects must be selected to be tweaked. Using the **Space Bar** is a quick way to switch back and forth between the *Select Tool* and the *Tweak Tool*. Note that there is no onscreen indication of what objects are selected when the *Tweak Tool* is in use.



An example of using the *Tweak Tool* on a series of *Rectangles*. The brush size is indicated by the orange circle and the *Mode* by the cursor (in this case *Push*).

Stroke Style

Table of Contents

Tweaking Paths

Chapter 11. Paint Bucket Tool

Table of Contents

Inspired by the need for cartoonists to color their drawings, the *Paint Bucket Tool* flood fills a region with a color. True to an *SVG* drawing program, the new object is defined by vectors and thus is fully scalable. The region to be filled, however, is defined by the pixels on the screen at the time of the fill. This is best explained by the examples in the sections that follow.

Tweaking Colors

Table of Contents

Simple Use

Get the book.

Inkscape: Guide to a Vector Drawing Program

Get the book.

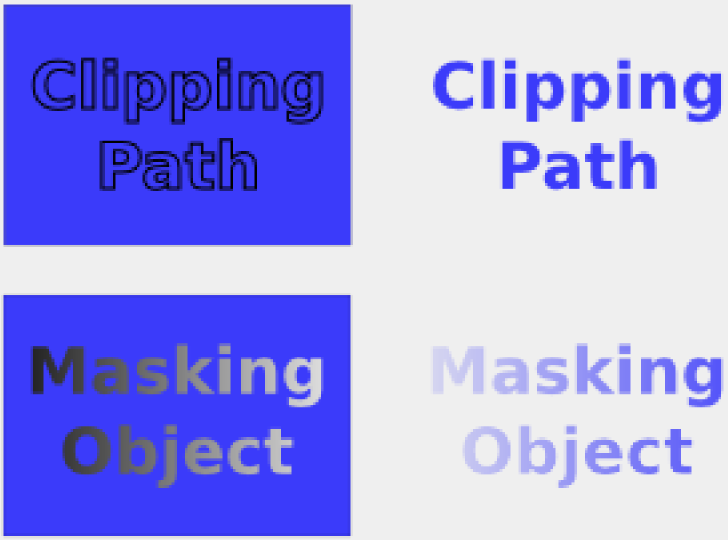
Inkscape » Clipping and Masking

Index

Chapter 12. Clipping and Masking

Table of Contents

Clipping and masking are methods for restricting what part of an object (or *Group* of objects) is visible. For clipping, a *clipping path* defines the visible part of the object while for masking, the *transparency* or *lightness* of one object determines the opacity of a second object. In both cases, the target object is not changed and can be unclipped or unmasked if needed.



Simple examples of clipping (top) and masking (bottom). The left column shows the text that serves as the clipping path and the text for masking, both overlaying blue rectangles that are the targets of the clipping and masking. In the right column are the results of the clipping and masking.

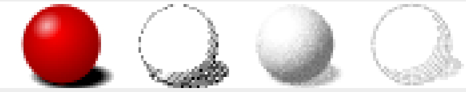
Tip

A clipped or masked object can be edited (transformed, style changed, nodes edited, etc.) while clipped or masked. Objects within a clipped or masked *Group* can also be moved relative to the clipping path or masking object if the *Group* is entered.

Adding to a Fill

Table of Contents

Clipping



Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Filter Effects

Index

Chapter 13. Filter Effects

Table of Contents

Greatly expanded in v0.46.

Filter Effects (*Filters*) are a feature of *SVG* that allow an *SVG* viewer to change the presentation of an object in a well-defined manner such as adding texture to a *Fill*, giving an object a blurred shadow, or modifying the object's color. Version 0.45 of Inkscape introduced support for the *Gaussian Blur Filter* primitive. Version 0.46 expands support to almost all *SVG Filter* primitives.

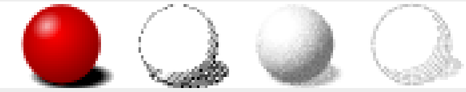
⚠ **Warning**

Support of *Filters* is limited or nonexistent in many web browsers. Firefox 1.5 and 2 do not support filters; Firefox 3 introduces support. Opera 9.5 supports filters. Internet Explorer supports filters through the Adobe plug-in. Squiggle (Batik) and *librsvg* support filters. Filters have not been widely used and thus viewers haven't received a lot of testing. Inkscape, Firefox 3, Opera 9.5, and Batik all have their quirks and will display some filters correctly but not others.

← Masking

Table of Contents

→ Basic Use



Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » XML Editor

Index

Chapter 14. XML Editor

Table of Contents

The *XML Editor* dialog allows one to directly edit the *XML* description of an *SVG* drawing. (Recall that Inkscape is an *SVG*-based drawing program and that *SVG* is an *XML*-based file format.)

The ability to directly edit an *SVG XML* file is very powerful. It allows the user more control over objects in their drawing such as specifying the exact size or position of an object and by giving access to *SVG* parameters that are not directly or easily available through the Inkscape interface.

Complex Examples

Table of Contents

Basic Usage

Get the book.

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Tiling

← Index →

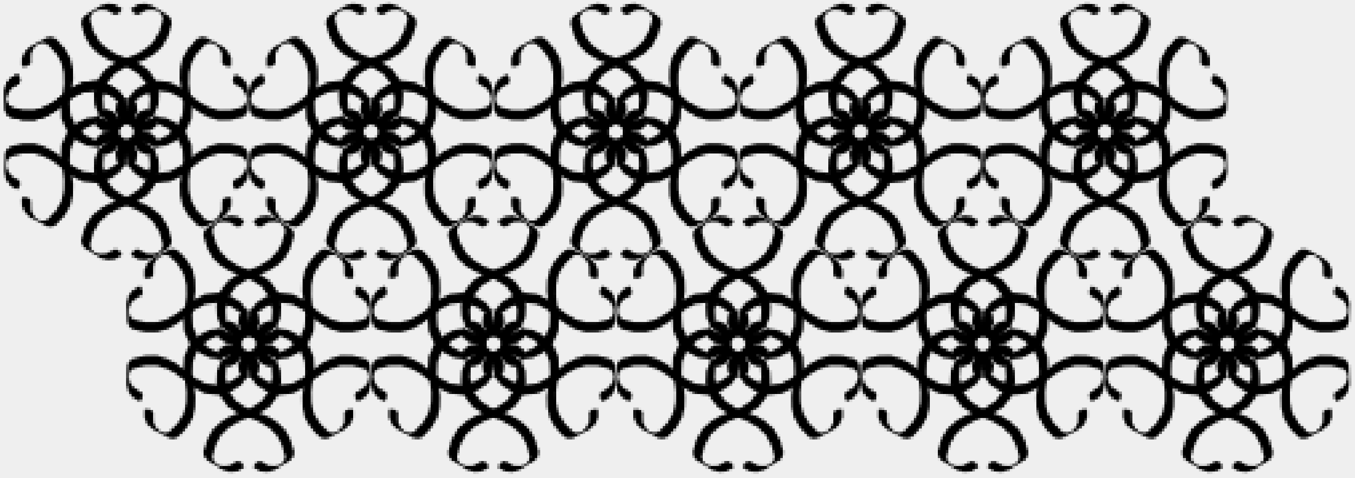
Chapter 15. Tiling

Table of Contents

Tiling or tessellation is the covering of a surface with the repeated use of the same shape tile. A typical example is the tiling in a bathroom. In Inkscape, this concept is expanded to include a multitude of options including progressively changing the tile size, spacing, and orientation.

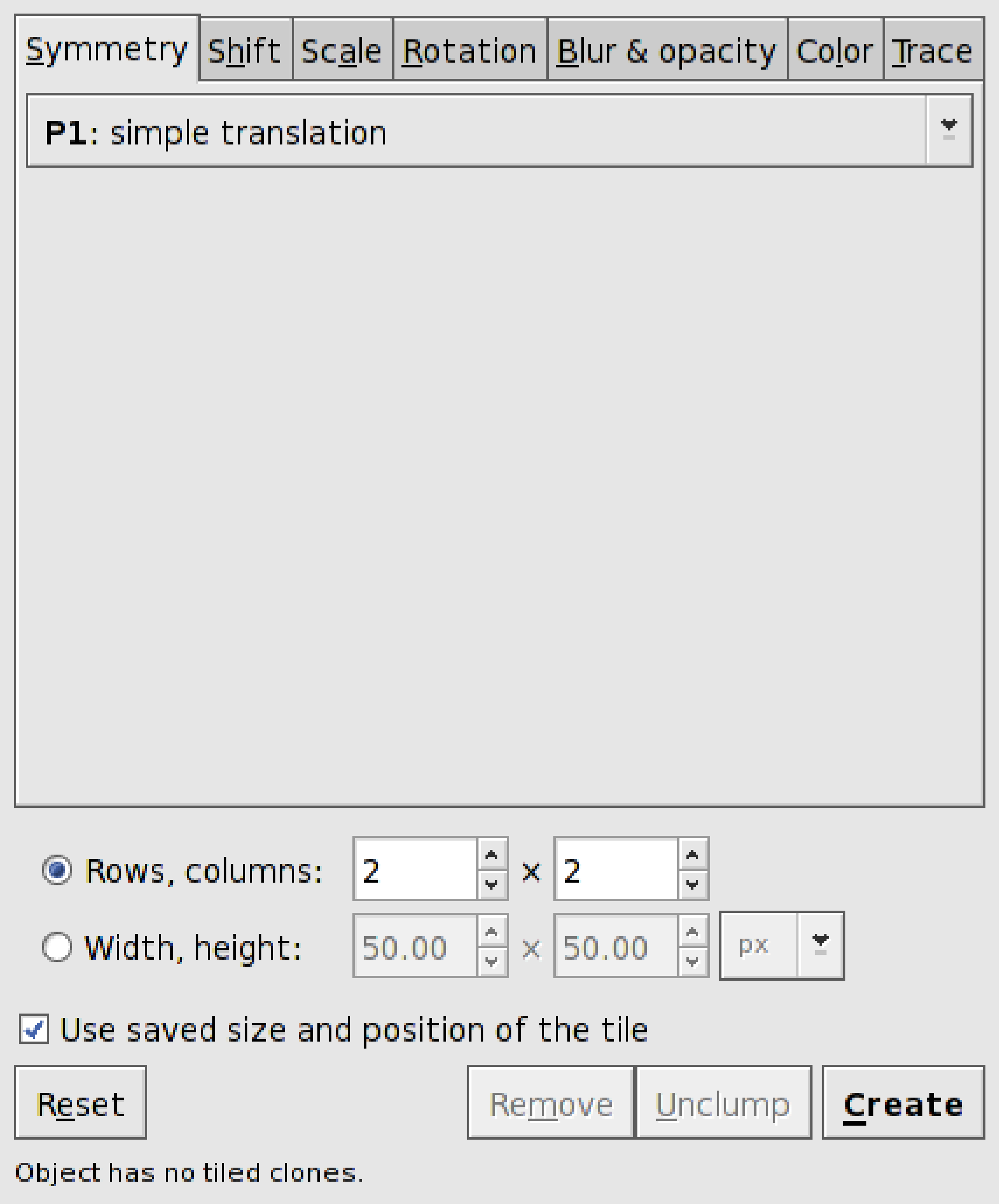
The tiles are in reality just clones of the source tile or object. Thus the same methods that apply to clones apply to tiles. (See the section called “Clones” in Chapter 4, *Editing Basics*.)

While random use of the *Tile Clones* dialog can produce exquisite patterns, it is useful to understand the fundamentals of tessellation in order to have more control over the final design.



An example of using the *Tile Clones* dialog with a simple calligraphic stroke and the *P6M* symmetry group (see text).

To construct a tiling, open up the *Create Tiled Clones* dialog (Edit → Clone → Create Tiled Clones...).



P1: simple translation

● Rows, columns: 2 × 2

○ Width, height: 50.00 × 50.00 px

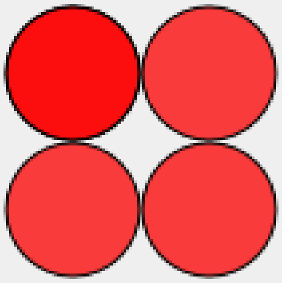
☑ Use saved size and position of the tile

Object has no tiled clones.

The *Tile Clones* dialog with no objects selected.

At the bottom of the dialog is a fixed section where you can choose the size of the tiling either by the number of rows and columns or by the width and height of the area you wish to cover. The terms *Rows* and *Columns* are only really appropriate for tiling of rectangular tiles (see below). Checking the “Used saved size and position of the tile” forces the tiling to use the size and position of the base tile at the last time the tile was used in a tiling. This preserves the spacing between tiles if the bounding box has changed due to editing the base tile. Clicking on the *Reset* button resets most of the entries under the tabs to their default values. The *Remove* button can be used to undo a tiling when the base tile is selected. The *Unclump* button can be used to spread out the clones in a somewhat random fashion (can be repeated). And, finally, the *Create* button creates the tiling.

With a circle and the default values (P1 symmetry, two rows and two columns), you will get the following tiling:



The simple tiling of a circle. The symmetry is "P1" and there are two rows and two columns.

The circle has been replicated four times in two rows and two columns. The original circle is still there, under the top-left cloned circle. The *bounding box* of the circle has been used as the base tile size.

This example is not so interesting, but there are many options under the dialog's tabs that can produce many interesting effects. Each tab will be covered in turn in the following sections.

Examples

Table of Contents

Symmetry Tab

Inkscape » Tracing Bitmaps

Index

Chapter 16. Tracing Bitmaps

Table of Contents

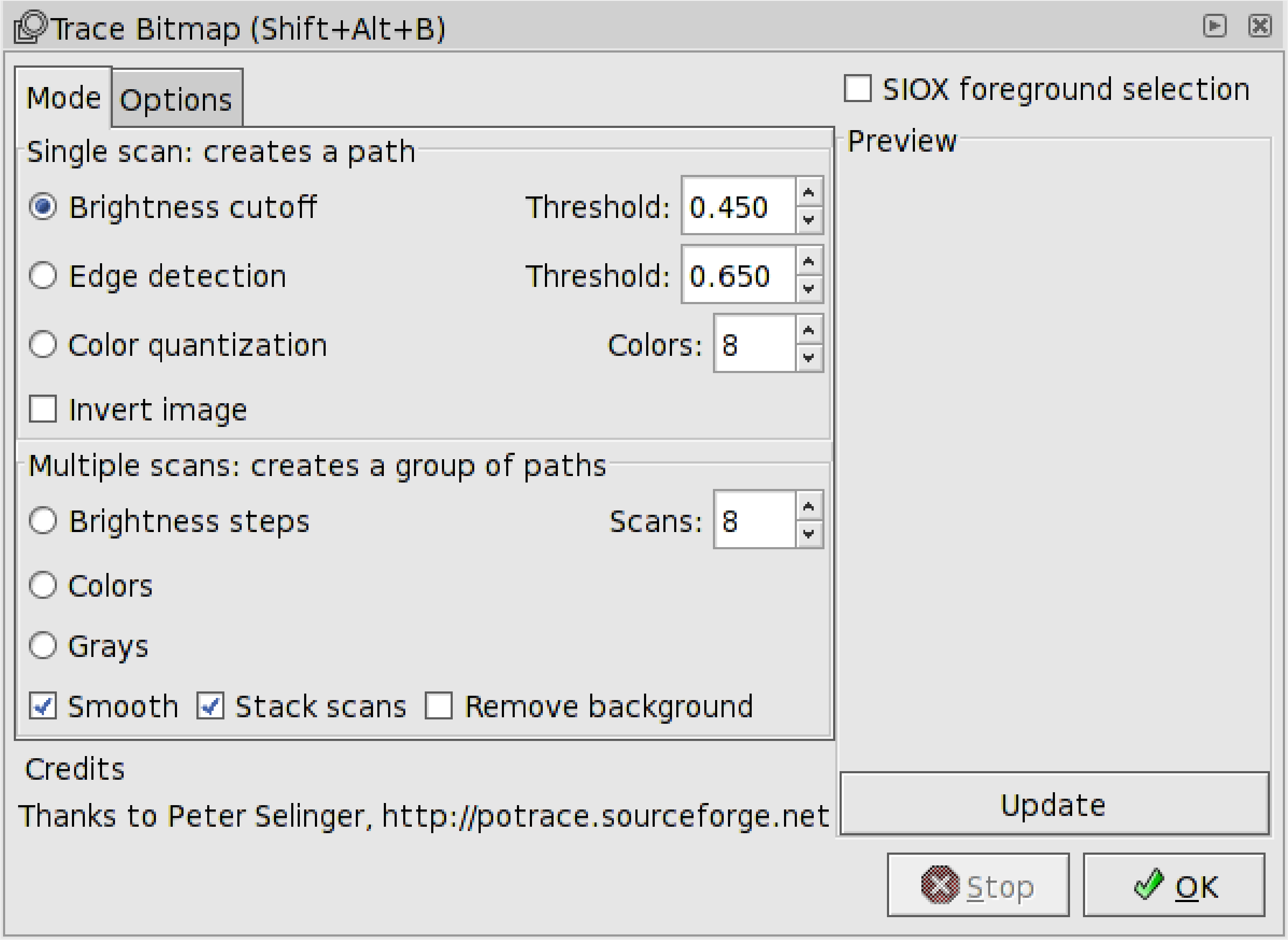
Updated in v0.45.

Inkscape has the ability to convert bitmap images into paths via *tracing*. Inkscape uses routines from Potrace, with the generous permission of the author, Peter Selinger. Optionally, SIOX can be used as a pre-processor to help separate a foreground from a background.

Tracing an image is not an easy thing to do. Potrace works well for some types of artwork (black and white line drawing) and not so well for others (scans from screened color prints). The paths that are created can have thousands of nodes depending on the complexity of the image and may tax the power of your CPU. Using the *Suppress speckles* option can reduce the number of nodes generated by the scan. After the scan, you can use the Path → Simplify (**Ctrl+L**) command to reduce the number of nodes (but at a cost in resolution). In the latter case, careful tuning of the *Simplification threshold* under the *Misc* section of the *Inkscape Preferences* dialog may be necessary to obtain optimal results.

The result of tracing depends heavily on the quality of the input images. Filtering input scans using Gimp (e.g., Gaussian blur) or mkbitmap may improve your results.

To trace a bitmap, call up the *Trace Bitmap* dialog (Path → Trace Bitmap... (**Shift+Alt+B**)). The dialog has two tabs. The first is to select the tracing mode and the second has a list of options.



Trace Bitmap (Shift+Alt+B)

Mode | Options

□ SIOX foreground selection

Single scan: creates a path

● Brightness cutoff Threshold: 0.450

○ Edge detection Threshold: 0.650

○ Color quantization Colors: 8

□ Invert image

Multiple scans: creates a group of paths

○ Brightness steps Scans: 8

○ Colors

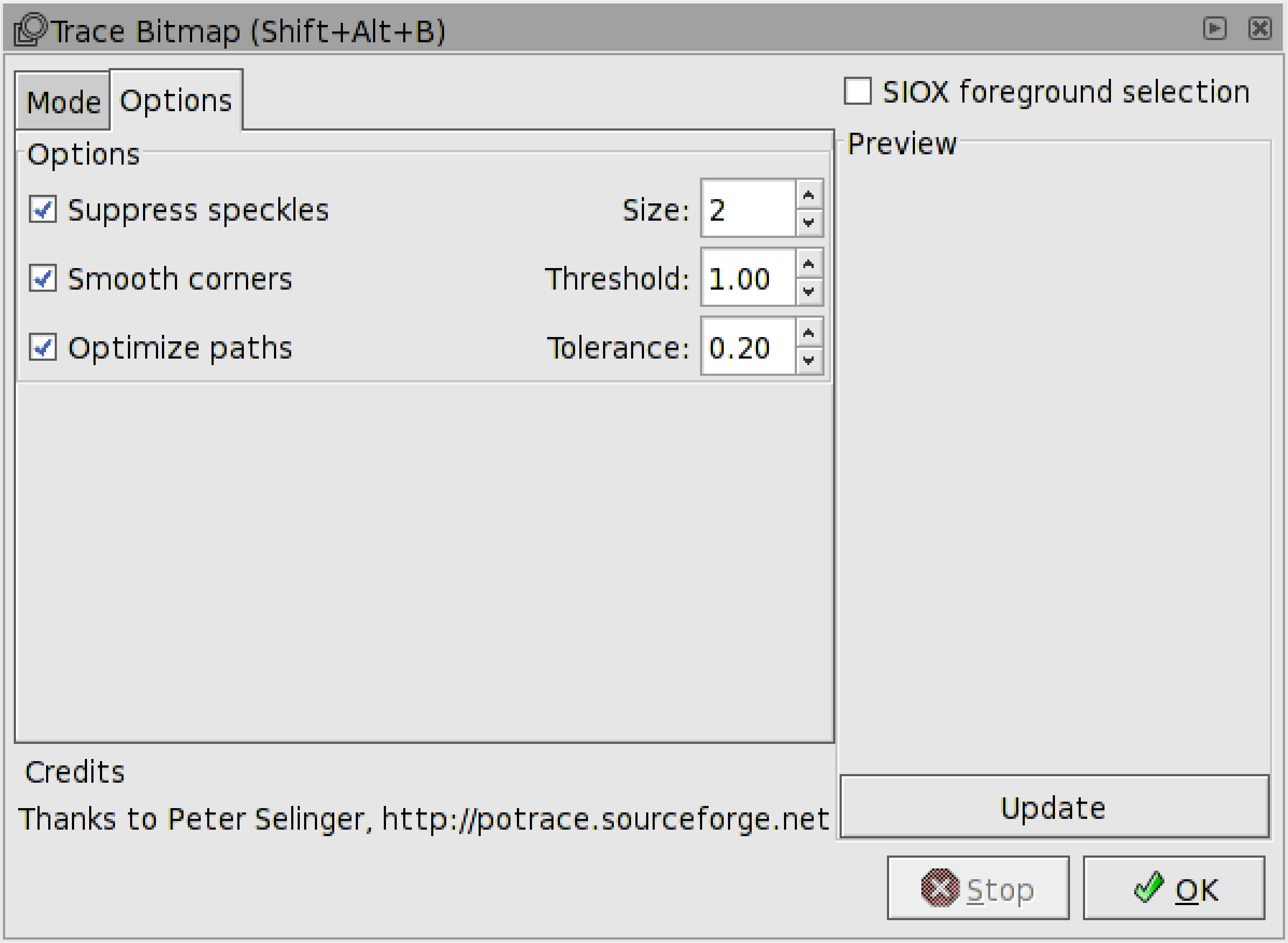
○ Grays

☑ Smooth ☑ Stack scans □ Remove background

Credits

Thanks to Peter Selinger, http://potrace.sourceforge.net

Trace Bitmap dialog, *Mode* tab.



☐ SIOX foreground selection

Options

☑ Suppress speckles Size: 2

☑ Smooth corners Threshold: 1.00

☑ Optimize paths Tolerance: 0.20

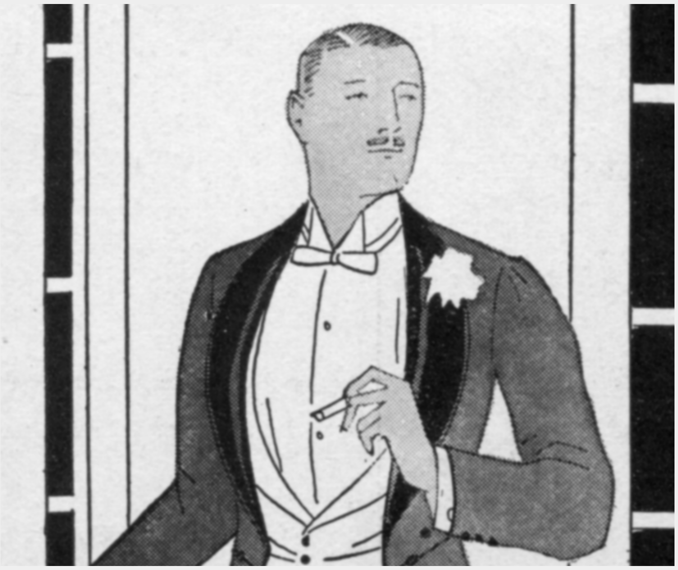
Credits

Thanks to Peter Selinger, http://potrace.sourceforge.net

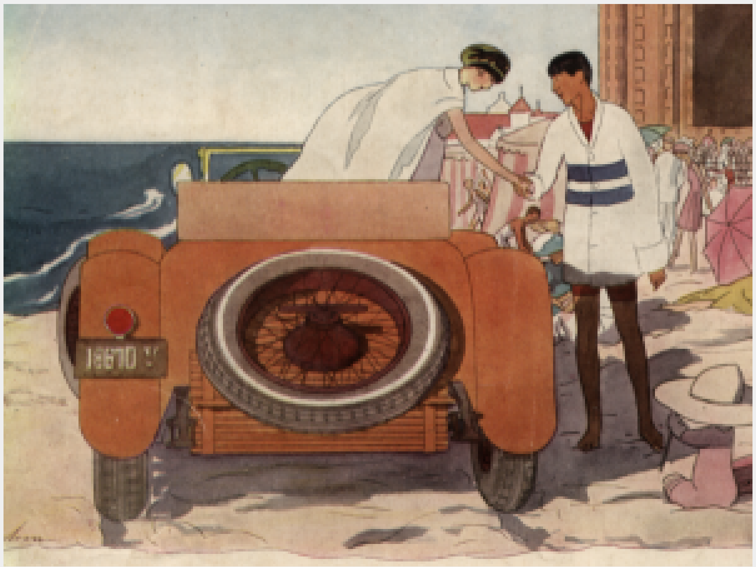
Trace Bitmap dialog, *Options* tab.

The *Mode* tab is divided into a number of parts. On the left are two sections: one for *Single* scans, where one *Path* is created, and one for *Multiple* scans, where several *Paths* are created. On the right is a *Preview* window, which can give you a quick idea of what the final scans will look like. A check box at the top right toggles on and off *SIOX foreground selection* (see below).

A number of scanning strategies are available. Each is discussed in a following section. The sections show the results of tracing a black and white figure and a color figure. The input figures (from the August 1919 edition of Vanity Fair) are shown below. The scans have been passed through the Gimp Gaussian Blur filter to remove the effects of the printing screens.



The source black and white drawing.



The source color drawing.

The following part of the chapter is divided into four parts. The first two cover *Single Scans* and *Multiple Scans*. The last two cover options that can be used both with *Single Scans* and with *Multiple Scans*.

Tricks

Table of Contents

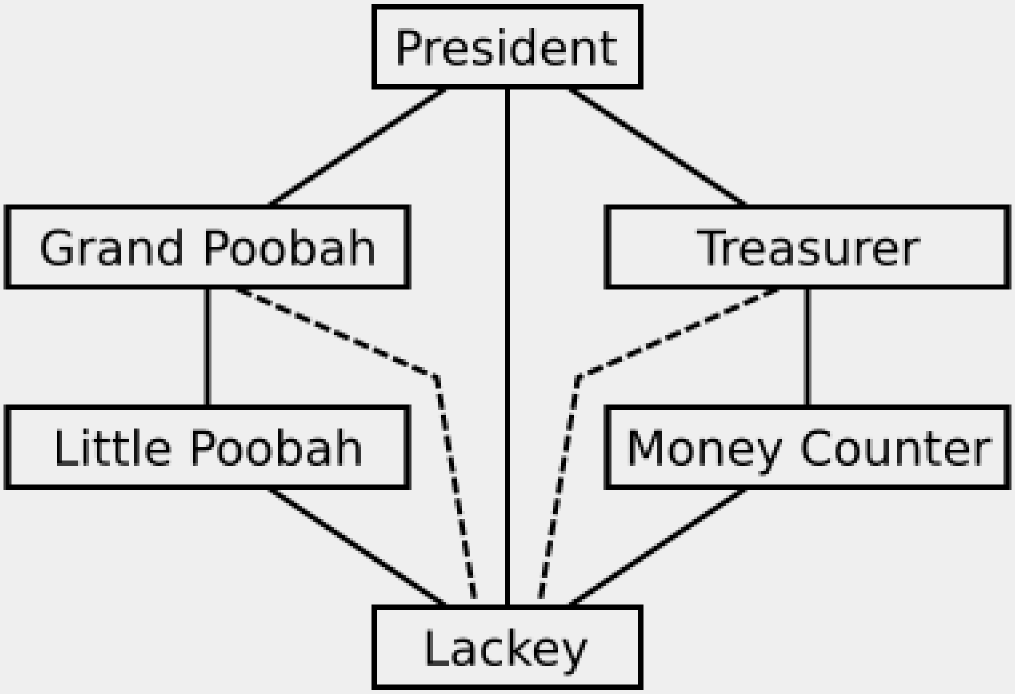
Single Scans

Get the book.

Chapter 17. Connectors

Table of Contents

Connectors are lines that “connect” objects, useful for drawing organization charts or flow diagrams. Connectors remain connected even if the objects they connect are moved. Individual objects can be given an *avoid* property that causes connectors to be routed dynamically around them.



An organizational chart drawn using connectors. The dashed lines have been automatically routed around other objects.

SIOX

Table of Contents

Creating Connectors

Get the book.

Chapter 18. Effects

Table of Contents

Updated for v0.46

Inkscape can be enhanced by *Effects*. These are scripts or programs that can be run from inside Inkscape. Most *Effects* require external programs, usually written in Perl or Python.

Many of the included *Effects* might be of marginal use to the average user. However, much can be learned by examining the code in order to write your own scripts. Look in the `share/inkscape/extensions` directory for the code. Note that on Windows, *Effects* written with Python will pop open an empty console window. This window will disappear when the *Effect* is finished.

A few *Effects* are built into Inkscape; they are located in the source directory `src/extensions/internal`. Often, an *Effect* can be used to quickly prototype a feature that may be included natively in a future version of Inkscape. *Effects* are also a good way to add a feature that may have limited use by the general Inkscape community and thus not warrant the long-term commitment of adding the feature to the main code base.

If an effect doesn't work, it may be that you are missing some external dependency. You can check if this is the case by looking at the log file `extensions-errors.log` in your Inkscape preferences directory (`.inkscape` on Linux, `Documents and Settings\USER\Application Data\Inkscape` on Windows).

As of v0.46, *Effects* can be run live, that is, the script code can be run automatically in the background, responding immediately to changes in parameters. This can both be good (see results of parameter changes immediately) or bad (updating before a parameter is fully modified). Each *Effect* dialog has a button to toggle on and off this *Live Preview*.

It is possible to assign keyboard shortcuts to effects. See the section called “Custom Keyboard Shortcuts” in Chapter 20, *Customization*.

Effects are grouped under several broad categories:

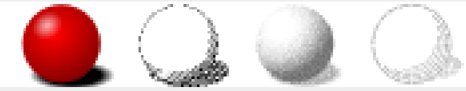
- Color: Modify the colors of an object or a *Group* of objects.
- *Removed in v0.46.* Export: Export groups to *PNG* files. *Use batch export option in Export Bitmap dialog instead.*
- *Removed in v0.46.* Fretboard Designer: Create custom guitar or other stringed instrument fret boards.
- Generate from Path: Utilize a path object to create a new object.
- *New in v0.46.* Generate Template: One effect at the moment to generate a template for a “Perfectly-Bound Cover,” as one would need for “Print On Demand” publishing.
- Images: Extract or embed images in an Inkscape file.
- Modify Path: Modify an existing path.
- *New in v0.46.* Raster: Manipulate the colors in a *bitmap*.
- Render: Create a new object.
- Text: Manipulate text.
- Visualize Path: Extract information about a path.

Two entries under the *Effects* menu allow one quick access to the previously used effect. The first, Effects → Previous Effect, will run the effect with the same parameters. The second, Effects → Previous Effect Settings... , will pop up the parameters dialog.

Routing Connectors

Table of Contents

Color



Inkscape » SVG and the Web

← Index →

Chapter 19. SVG and the Web

Table of Contents

SVG should really come into its own on the Web. Its small compact file structure (when compressed) makes for quick downloads, yet its vector nature results in excellent rendering at any scale. But this is only the beginning. *SVG* plays nicely with other web technologies: *SVG* files can be embedded into web pages. *SVG* drawings can contain hypertext links. And *SVG* drawings can be scripted and animated.

Support for *SVG* is rapidly improving in browsers. Firefox (1.5, 2.0 and 3.0), Opera (8.5, 9.0, 9.5), and Safari (3.0) include different levels of support; Konqueror is in the process of adding support. Users of Internet Explorer and other browsers can join the fun by using the (ancient) Adobe plug-in.

This chapter introduces *SVG* use for the Web and covers the Inkscape features that help with this use. It is not the purpose of this book to cover web design and it provides only a minimal introduction to animation and scripting, which are not (yet) supported by Inkscape. It also does not cover all the tricks that may be needed to get *SVG* files to work on all browsers.

The sections in this chapter build on each other. The first section covers getting an *SVG* to display on a web page. The second covers adding hyperlinks. The third section introduces the use of style sheets. And the final section is a short introduction to animation.

A web page displaying the examples discussed here is available at the book's website. A web page for testing browser support of various *SVG* features can also be found there.

←

Visualize Path

Table of Contents

→

Simple SVG Display

Get the book.

Inkscape » Customization

← Index →

Chapter 20. Customization

Table of Contents

Inkscape is quite customizable. There are two ways to do so: through the *Inkscape Preferences* dialog and through modifying files in the `share/inkscape` directory (or `.inkscape/`).

←

Simple Animation

Table of Contents

→

Inkscape Preferences Dialog

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Using the Command Line

Index

Chapter 21. Using the Command Line

Table of Contents

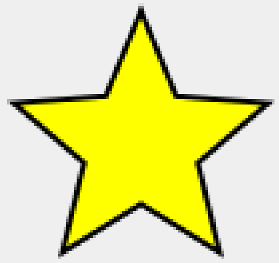
Updated for v0.46.

Inkscape has the ability to batch process *SVG* files without opening up the Graphics User Interface (*GUI*). The available options can be divided into a few categories: general commands, exporting commands (including printing), and query commands.

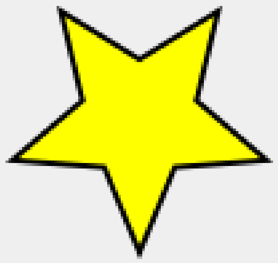
Most Inkscape commands are attached to verbs. Any verb can be called from the command line with the `--verb` argument, allowing complex processing to take place. However, it is not possible to set parameters. A list of all verbs can be obtained using `--verb-list`. It does not appear possible to suppress the *GUI* when using the `--verb`.

Here is a simple example of opening a file, selecting an object, flipping it, and then saving the file. The *Star* has an *id* of “MyStar”.

```
inkscape --select=MyStar --verb ObjectFlipVertically --verb FileSave
--verb FileClose MyStar.svg
```



The file before running the above command.



The file after running the above command.

Most options have two forms: a short form preceded by one dash and a long form proceeded by two dashes. Some options take parameters that can (usually) either be attached to the option with an = sign (e.g., `--export-png=my.png`) or separated by a space (e.g., `--export-png my.png`).

Inkscape Configuration Files

Table of Contents

General Options

Get the book.

Index

Chapter 22. Challenges

Table of Contents

Try your skill at drawing these! Some may not be possible with the current version of Inkscape!

Solutions are given in the Appendix of the printed book (see the book's website).

Query Options

Table of Contents

Red Spiral

Get the book.

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Inkview ← Index →

Appendix A. Inkview

Inkview is a standalone display program for displaying *SVG* files. It can be used to run a slideshow. It is normally shipped with Inkscape.

Inkview can only be used from the command line.

Usage: **inkview** `FILE`

where FILE is one or more filenames of *SVG* files (.svg), compressed (gzipped) *SVG* files (.svgz), or *SVG* archive (.sxw, .jar). The archive option has not been tested by the author.

- **Right-arrow**: Show next slide.
- **Left-arrow**: Show previous slide.
- **Up-arrow**: Go to first slide.
- **Down-arrow**: Go to last slide.
- **Esc** or **q**: Quit.
- **F11**: Toggle between full screen and window modes.
- **Enter**: Pop up window with control buttons (First, Previous, Next, Last).

← Spiral Gyral

Table of Contents

→ Appendix B. File Format

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » File Format

Index

Appendix B. File Format

Table of Contents

It is not the purpose of this section to describe the Inkscape and *SVG* file formats in detail (for that, look at the *W3C SVG* website). Instead, the purpose is to give an overview that could help the user, for example, edit the *SVG* files with the *XML Editor*.

Appendix A. Inkview

Table of Contents

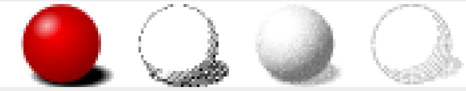
Default Template File

Get the book.

Appendix C. Spheres

Table of Contents

This section gives diagrammatic views of how the spheres on the book's original front cover were created.



Inkscape » List of Dialogs

← Index →

Appendix D. List of Dialogs

- File
  - File → Open... (**Ctrl+O**).
  - File → Save As... (**Shift+Ctrl+S**).
  - File → Import... (**Ctrl+I**).
  - File → Export Bitmap... (**Shift+Ctrl+E**).
  - File → Document Properties... (**Shift+Ctrl+D**).
  - File → Inkscape Preferences... (**Shift+Ctrl+P**).
  - File → Input Devices... .
- Edit
  - Edit → Find... (**Ctrl+F**).
  - Edit → Clone → Create Tiled Clones... .
  - Edit → XML Editor... (**Shift+Ctrl+X**).
- View
  - View → Swatches... (**Shift+Ctrl+W**).
  - View → Messages... .
  - View → Scripts... .
  - View → Icon Preview.
- Layer
  - Layer → Add Layer... .
  - Layer → Rename Layer... .
- Object
  - Object → Fill and Stroke... (**Shift+Ctrl+F**).
  - Object → Object Properties... (**Shift+Ctrl+O**).

    - Object → Transform... (**Shift**+**Ctrl**+**M**).
    - Object → Align and Distribute... (**Shift**+**Ctrl**+**A**).
    - Object → Rows and Columns... .
- Path
    - Path → Trace Bitmap... (**Shift**+**Alt**+**B**).
- Text
    - Text → T Text and Font... (**Shift**+**Ctrl**+**T**).
- Help
    - (None)
- Other
    - Guideline (double click on a *Guide Line*).
    - Gradient Editor (access from *Fill and Stroke* dialog when a gradient is selected or the *Gradient Tool*-*Tool Controls*).
    - Link attributes (right click on object and select from pop-up menu after creating a *link*; see the section called “Adding Links ”).

←

Sphere with Text Shading and Shadow

Table of Contents

→

Appendix E. Tips for Illustrator Converts

Appendix E. Tips for Illustrator Converts

Table of Contents

This section is to help those with experience using Illustrator to adapt to Inkscape.

Appendix D. List of Dialogs

Table of Contents

Key Mappings

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Glossary

← Index →

Glossary

alpha

The transparency of an object (or pixel). If an object with a non-maximal value of alpha is placed over another object, the second object will be visible under the first. In Inkscape, a value of Alpha of 255 means the object is completely opaque while a value of 0 means it is fully transparent (not visible).

Animated Portable Network Graphic (APNG)

An open standard for animated bitmap graphics, the animated parallel of the *PNG* standard and an alternative to *MNG*. There is support for this format in Firefox 3 and Opera 9.5 (alpha) but the *PNG* group has rejected this extension to the standard.

baseline

For text, the line on which most characters (i.e., “x”) rest. Some characters such as “p” extend significantly below the baseline. Other characters such as “O” usually extend a small amount below the baseline so that they optically appear to rest on the baseline. Inkscape can align text to a common baseline. Inkscape also uses the word to describe the point at which vertical text is aligned horizontally. The baseline is indicated by a small square when text is selected.

bitmap graphics

The description of a drawing using pixels (in contrast to vectors). Also refered to as “raster” graphics.

See Also vector graphics.

bounding box

The smallest rectangular box with sides parallel to the *x* and *y* axis that completely encloses an object. Note: In Inkscape, the *bounding box* is calculated assuming a round stroke *Join* and *Cap* style if the stroke is visible and the *Visual bounding box* option is selected in the *Tools* section of the *Inkscape Preferences* dialog. If the *Geometric bounding box* option is selected, only the nodes are considered in the calculating.

bump map

A bitmap graphics used to define the contour of a surface so that a lighting effect can be applied. The *SVG* specification uses the *Alpha* channel for this purpose in several of the *Filter* primitives. See: Wikipedia entry.

Cascading Style Sheets (CSS)

A way of controlling the layout and style of graphics objects (including text) through the use of external files. This allows the separation of content from presentation. It allows documents to be easily adapted for a variety of rendering methods such as printing and web display.

Cyan Magenta Yellow Key (Black) (CMYK)

A method for describing a color by the amount of cyan, magenta, and yellow needed to generate the color. This subtractive color model (where light is absorbed) is most often used in printing. As a good black is difficult to obtain using a mixture of these colors, a fourth ink, the *Key* or black is also used. See: Wikipedia entry.

dots per inch (dpi)

The number of pixels per inch when printing or displaying a digitized image. Inkscape has a default resolution for exporting bitmaps of 90 dpi.

gamma

A correction factor to account for non-linearity in a display device. More technically, the numerical value of the exponent of the power-law correction.

Gaussian distribution

Also called *Normal distribution*, a mathematical function that describes a distribution found often in statistics (e.g., the distribution of scores on a test). The relevance for Inkscape comes from the use of the distribution in the *Gaussian Blur* filter. The key point is that the color of a pixel is determined by the colors of nearby pixels in the source, weighting the nearest pixels more.

ghostscript

An open-source PostScript interpreter. See: Ghostscript home page.

Graphics Interchange Format (GIF)

A patented standard for compressing bitmap graphics supported by most web browsers. The open *PNG* standard is technically superior and should be the format of choice for lossless compressed bitmaps.

Graphics User Interface (GUI)

The interface a computer program presents to the user.

hexadecimal number

A way of representing a number using base 16 rather than the normal base 10. Very commonly used with computers. The base 10 numbers 0-9 are augmented by the letters a through f (which may or may not be capitalized) representing the numbers 10-15. For example, 31 in base 10 is written as 1F in hexadecimal (1 times 16 plus 15 is 31).

hue, saturation, value (HSL)

A method for describing a color using hue, saturation, and lightness. See: Wikipedia entry.

HyperText Markup Language (HTML)

See: Wikipedia entry. The original markup language for web pages. In the process of being superseded by *XHTML*.

Joint Photographic Experts Group (JPEG)

A standard for lossy compression of bitmap graphics supported by most web browsers. More suitable for photographs than line art.

kerning

The process of adjusting the space between letters in text to improve the appearance of the text. A classic example is that the “A” and “v” in “Aviary” should slightly overlap.

LaTeX

A system for producing high-quality documents commonly used in mathematics and physics documents. LaTeX is built on top of TeX. See: LaTeX home page.

Multipurpose Internet Mail Extensions (MIME)

An Internet standard, originally developed for electronic mail, that assigns to each type of document content a unique name so that clients (programs) can interpret the data correctly.

Multiple-Image Network Graphic (MNG)

An open standard for animated bitmap graphics, the animated parallel of the *PNG* standard. Unfortunately, there is little support for this format in web browsers.

name spaces

The use of tags to define a region in a file where certain definitions are applicable. For example, an *XML* file can contain both *SVG* and *XHTML*. Name spaces keep the two from conflicting with each other.

opacity

The property of an object that determines the visibility of an underlying object. Opposite of *transparency*.

See Also alpha, transparency.

PDF

Portable Document Format. A “printing” language created by Adobe that supersedes PostScript. See: Adobe PostScript site.

pixel

Short for picture element. The smallest part of a digitized image that includes all the color information for a region. Computer screen resolutions are typically described as having some number of pixels per inch. The pixel (px) is the default user unit for *SVG*. Inkscape has a default resolution for exporting bitmaps of 90 pixels/inch (ppi). *SVG* viewers typically have a default resolution of either 72 or 90 ppi.

point

A unit derived from the days when printers used letters carved in metal blocks for printing. Various points have been defined. Inkscape uses the computer point, which is 1/72 of an inch or 0.35277 mm.

Portable Network Graphic (PNG)

An open standard for compressing bitmap graphics supported by most web browsers.

PostScript

A printing language created by Adobe. The language can be used to describe a document in a device-independent way. See: Adobe PostScript site.

px

See pixel.

red-green-blue (RGB)

A method for describing a color using the amount of each primary color present. See Wikipedia entry.

red-green-blue-alpha (RGBA)

The addition of *Alpha* (transparency) as a fourth component to a *RGB* specified color.

rubber band

The box drawn when click-dragging the mouse with the *Select Tool* or the *Node Tool* active. The objects or nodes within the box will be selected. The drag must begin over an area without an object or with the **Shift** key held down.

Scalable Vector Graphics (SVG)

An XML standard for describing a drawing using vector graphics. See: W3C SVG page.

Tag Image File Format (TIFF)

A file format for storing high-quality images such as photographs or line drawings.

tool tip

A short dialog shown while the mouse cursor is above some part of window. A typical use is to describe the function of an icon.

transformation matrix

A 3x3 matrix that describes how an object is to be transformed. The upper-left 2x2 sub-matrix controls scaling, rotating, and skewing. While the upper-right 1x2 sub-matrix controls translations, the bottom row is not modifiable.

An advantage of using transformation matrices is that cumulative transformations can be described by simply multiplying the matrices that describe each individual transformation. Inkscape stores an object's transformation internally as a transformation matrix (which can be seen and modified with the *XML Editor*).

In non-matrix form, we have the transformation: $x' = Ax + Cy + E$ and $y' = Bx + Dy + F$ where (x', y') is the new coordinate of a point at (x, y).

With the above set of equations, it is easy to see that E is magnitude of a translation in the *x* direction and F is magnitude of a translation in the *y* direction. For scaling, A and D are the scale factors for the *x* and *y* directions, respectively. For a pure rotation, A = D = sin(theta) and B = -C = cos(theta) where theta is the angle of the desired rotation. For skewing, C and B control skewing parallel to the *x* and *y* axes respectively.

A *transformation matrix* is always defined with respect to some point. The internal representation is with respect to the internal coordinate system origin (upper-left corner of “page”).

transparency

The property of an object that determines the visibility of an underlying object.

See Also alpha, opacity.

Unicode

A standard for encoding characters for all the world's living languages as well as many historic ones. The character mapping for Unicode can be found at the Unicode organization's web pages.

Universal Resource Locator (URL)

The address of a web page, graphic, etc. on the WWW.

vector graphics

The description of a drawing using vectors (in contrast to bitmaps).

See Also bitmap graphics.

eXperimental Computing Facility (XCF)

The native format for Gimp. The acronym comes from the name of the student group at the University of California, Berkeley, that gave birth to Gimp.

eXtensible HyperText Markup Language (XHTML)

An *XML* markup language for web pages. See: Wikipedia entry.

XLink

XML Linking Language. A language for creating sophisticated links between documents. Similar to the simpler *URL*. The *SVG* specification uses “simple” *Xlinks*.

eXtensible Markup Language (XML)

The underlying format for *SVG* files. See: Wikipedia entry.

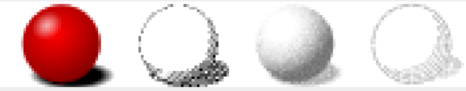
z-order

The order in which objects are drawn when they overlap each other. Objects drawn on top are higher up in z-order. Unless explicitly changed, the most recent object created is on top.

Terminology

Table of Contents

Index



Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Index

Index

A B C D E F G H I J K L M N O P Q R S T U V W X Z

Glossary

Table of Contents

Inkscape: Guide to a Vector Drawing Program

[Get the book](.).

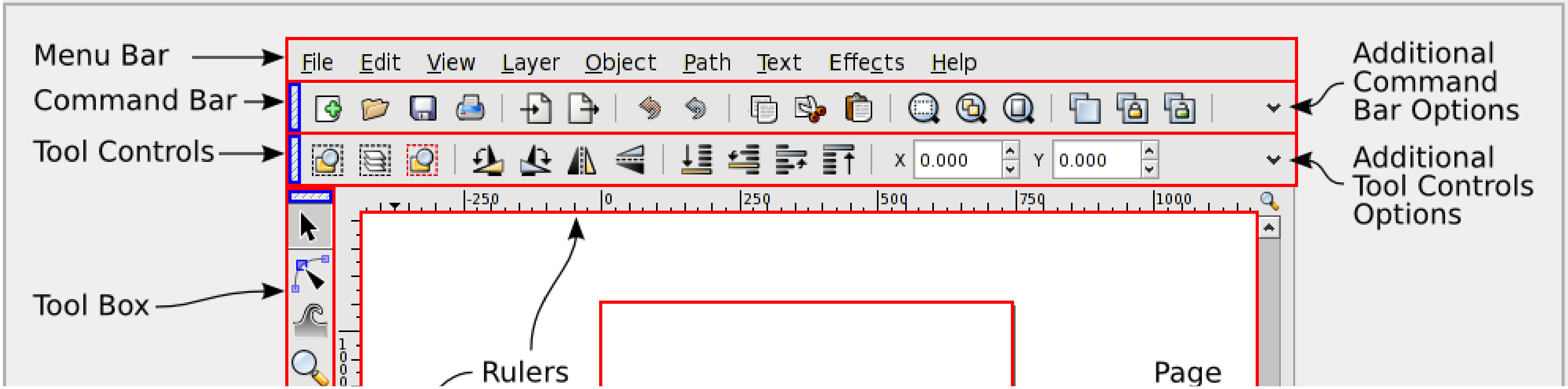
The Anatomy of the Inkscape Window

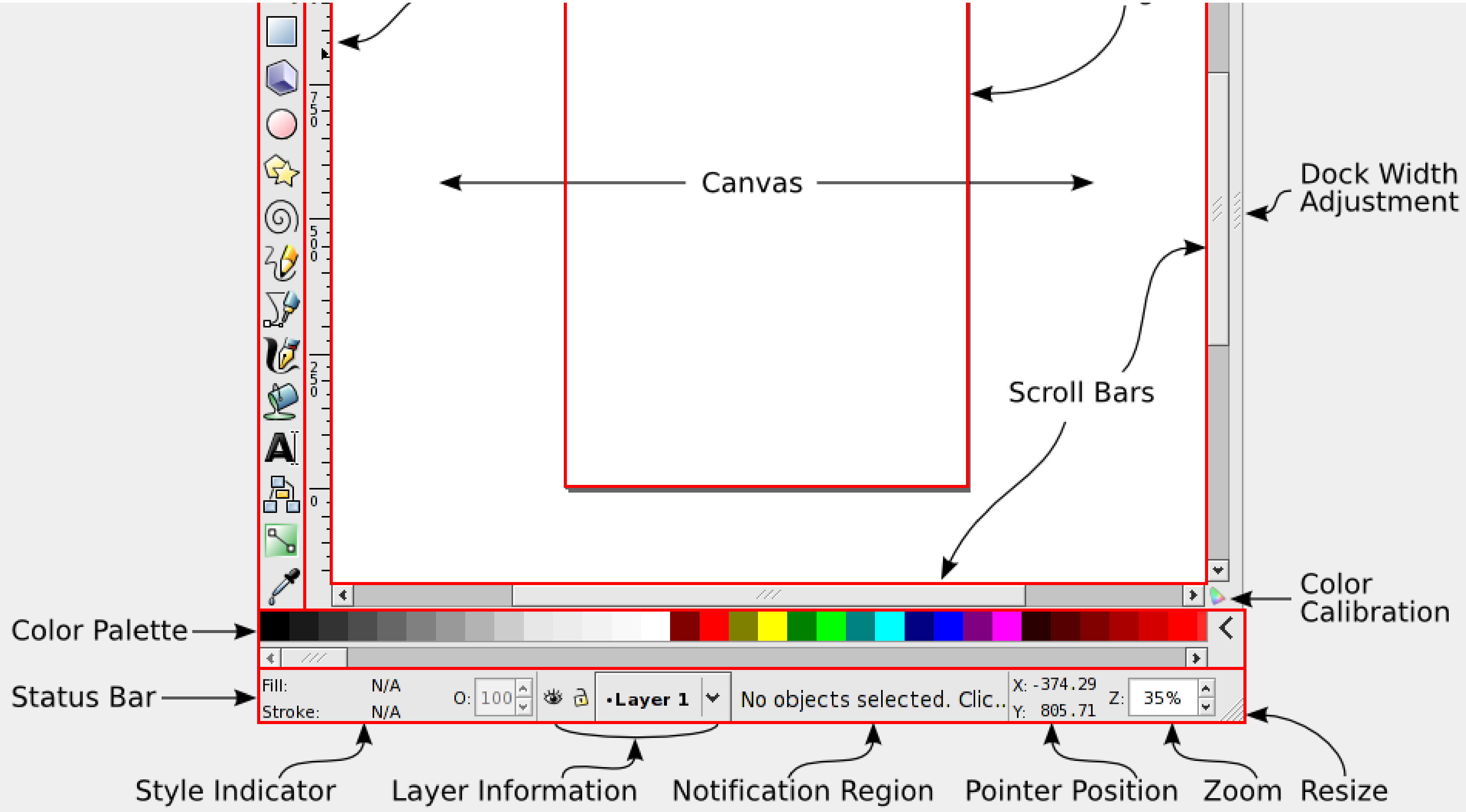
Updated for v0.46.

Start by opening Inkscape.[3] You will see a single window. This window contains several major areas, many containing clickable icons or pull-down menus. The following figure shows this window and labels key parts.

The *Command Bar*, *Tool Controls*, and *Tool Box* are detachable by dragging on the handles (highlighted in blue) at the far left or top. They can be returned to their normal place by dragging them back. They, as well as the *Palette* and *Status Bar*, can be hidden using the View → Show/Hide submenu.

As Inkscape has grown more complex, the area required to include icons and entry boxes for all the various items has also grown leading to problems when Inkscape is used on small screens. As of v0.46 the *Command Bar* and all the *Tool Controls* have variable widths. If there are too many items to be shown in the width the Inkscape window, a small down arrow will appear on the right side of the bars. Clicking on this arrow will open a drop-down menu with access to the missing items. One can also choose to use a smaller set of icons by checking the *Make the commands toolbar icons smaller* and *Make the main toolbar icons smaller* boxes on the *Misc* section of the *Inkscape Preferences* dialog (File → Inkscape Preferences... (**Shift+Ctrl+P**)). With these boxes checked, the smallest Inkscape window becomes 602 pixels wide and 620 pixels high (this may depend on which operating system you are using and the availability of a small icon set). By hiding all the various window components (*Command Bar*, *Palette*, etc.), you can get an Inkscape window just 447 pixels wide and 284 high.





Fill: N/A
Stroke: N/A
O: 100
•Layer 1
No objects selected. Clic..
X: -374.29
Y: 805.71
Z: 35%

The default Inkscape window with the key parts labeled.

Canvas

The drawing area. It may extend outside the viewable area. It can be panned (scrolled left/right and up/down) and zoomed.

Page

The part of the *Canvas* area corresponding to a printed page or other predefined area. Useful for setting an output region in printing or exporting a bitmap image. It may extend outside the viewable area.[4]

Menu Bar

Contains the main pull-down menus.

Command Bar

Contains shortcuts to many of the items located in the menus. Click on the *Down arrow* on the right end to access entries missing due to space.

Tool Box

Contains “Tools” for selecting, drawing, or modifying, objects. Clicking on an icon selects a tool. Double-clicking brings up that tool's preference dialog. The cursor (pointer) changes shape when placed over the canvas depending on which tool is selected.

Available tools:

Tool Controls

Contains entry boxes and clickable icons that are specific to the selected tool. For example, when the *Rectangle Tool* is in use, an entry box to specify a selected rectangle's width appears. Click on the *Down arrow* on the right end to access entries that may be missing due to space. If there is no arrow then all options are being shown.

Color Palette

Contains a color palette. Colors can be dragged from the palette onto objects to change their *Fill*. Using the **Shift** key while dragging will change the *Stroke* color instead. The color used by some tools can be set by clicking on a color swatch. The palette can be changed by clicking on the arrow icon at the right end of the palette. Many predefined palettes are included. If the number of color swatches in a palette exceeds the space allocated, the scroll bar beneath the palette can be used to access the hidden swatches.

Status Bar

Contains several areas including the *Style Indicator*, current drawing layer, pointer position, current drawing layer (and if it is visible or locked), current zoom level, window resize handle, and a *Notification Region* that describes context dependent options.

Style Indicator

Shows the style (*Fill* and *Stroke*) of a selected object, text fragment, or gradient stop. A **Left Mouse Click** on the *Fill* or *Stroke paint* part of the indicator opens the *Fill and Stroke* dialog. A **Right Mouse Click** opens up a pop-up menu. See the section called “Style Indicator” in Chapter 9, *Attributes* for details and more uses.

Notification Region

Contains context dependent information. If the region is too small to view all the text, placing the cursor over the region will display a *tool tip* with the full text.

Tip

The *Notification Region* contains very useful information. Pay close attention to it when using an unfamiliar tool.

Rulers

Show the *x*- and *y*-axis coordinates of the drawing. Click-dragging from a *Ruler* onto the *Canvas* creates a *Guide Line*.

Scroll Bars

Allows scrolling to adjust which part of the *Canvas* is viewable.

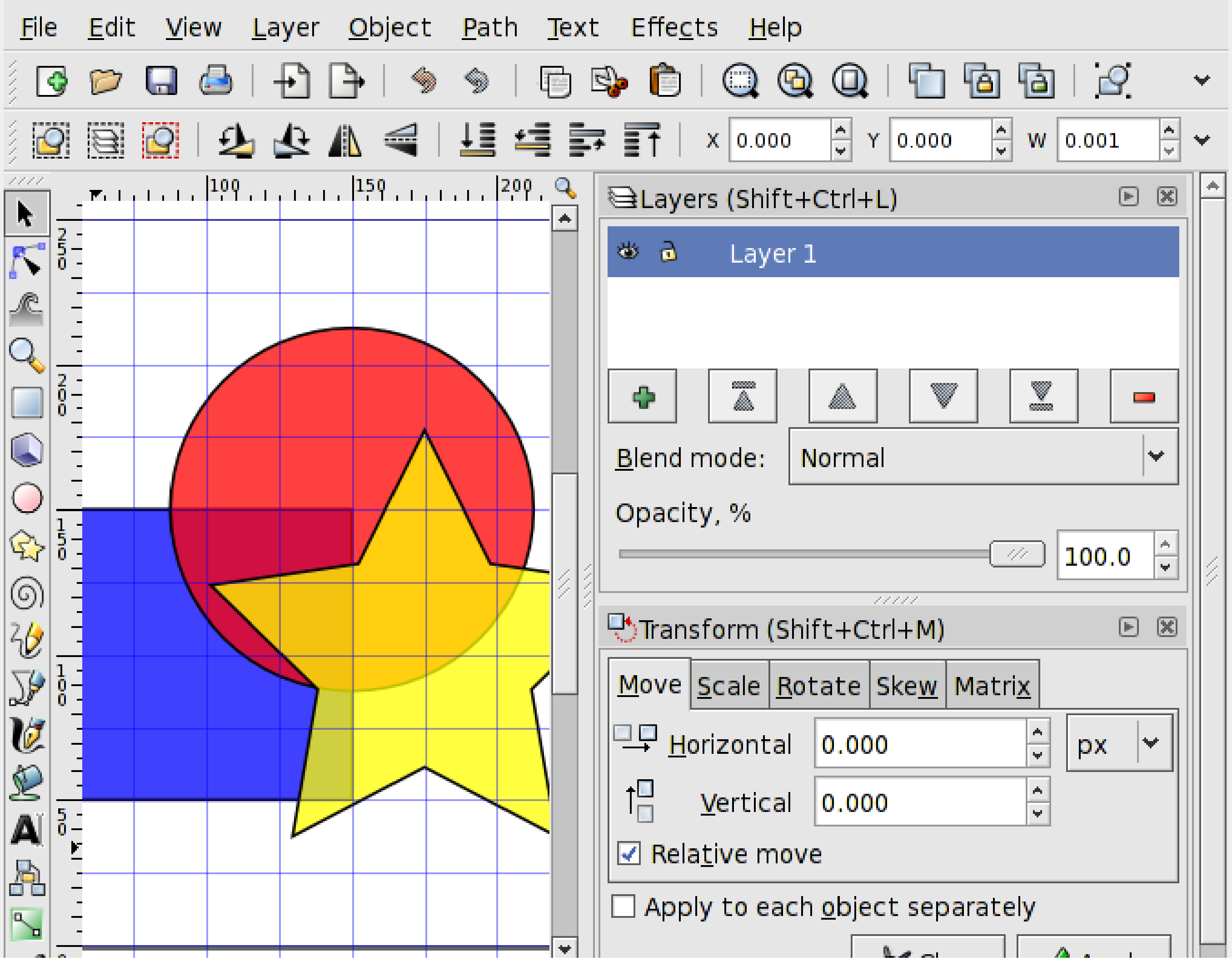
Color Calibration

Button toggles on/off use of a [Color Profile](#) (if set up).

Dockable Dialogs

New in v0.46.

Inkscape v0.46 introduces *Dockable Dialogs*. With this feature, opened dialogs are placed inside the main Inkscape window on the right side as seen in the next figure.



X 0.000 Y 0.000 W 0.001

Layer 1

Blend mode: Normal

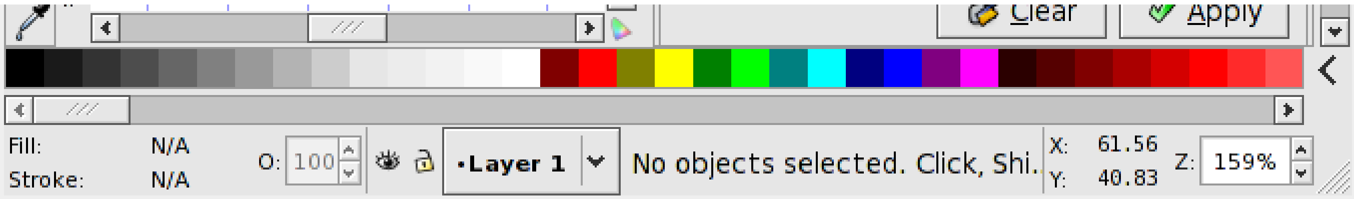
Opacity, % 100.0

Horizontal 0.000 px

Vertical 0.000

☑ Relative move

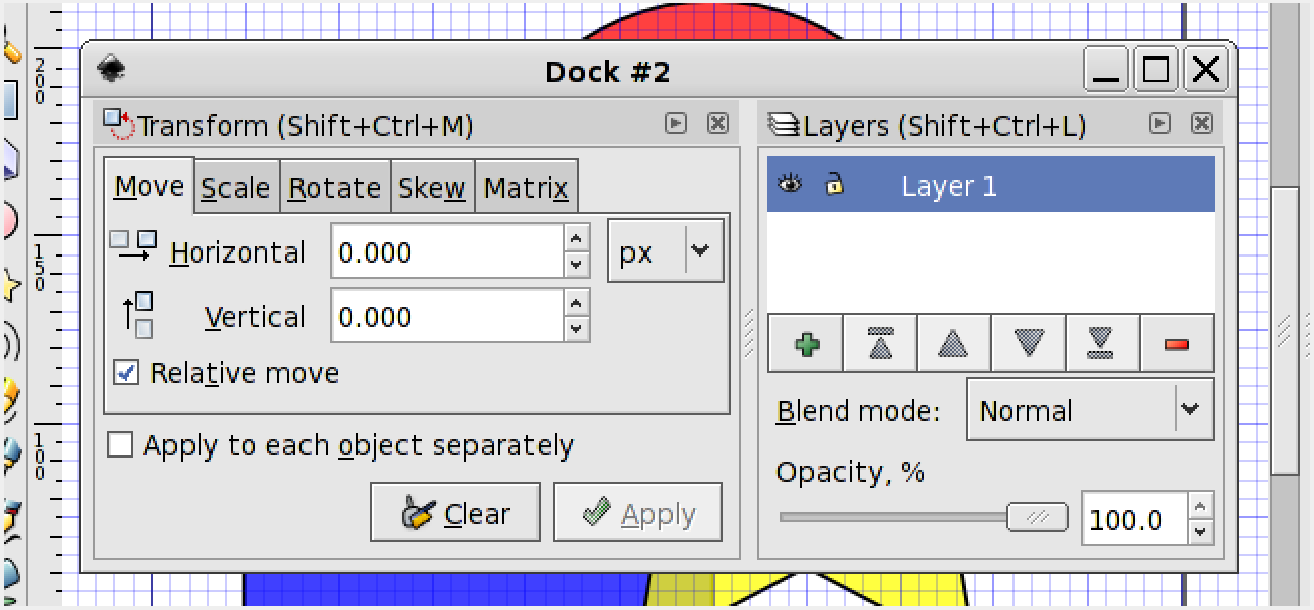
□ Apply to each object separately



Fill: N/A
Stroke: N/A
O: 100
•Layer 1
No objects selected. Click, Shi.
X: 61.56
Y: 40.83
Z: 159%

The Inkscape main window with two docked dialogs.

The docked dialogs can be rearranged, resized (if space permits), stacked, and iconified. To move a dialog, **Left Mouse Drag** in the dialog's title bar. Dialogs can also be dragged off of the main window into their own window. Each dialog can have its own window or they can be grouped in floating docks.



Horizontal 0.000 px
Vertical 0.000
☑ Relative move
□ Apply to each object separately
Layer 1
Blend mode: Normal
Opacity, % 100.0

A floating *Dock* with two dialogs.

Selecting *Floating* under *Dialog behavior* in the *Windows* section of the *Inkscape Preferences* dialog (File → Inkscape Preferences... (**Shift+Ctrl+P**)) disables the use of docks. Instead, all dialogs will be opened in their own window.

There is still some work to be done on the implementation of dockable dialogs. For example, a few dialogs have yet to be converted to be dockable (e.g. *Text*, *Object properties*). Bugs may also be encountered.

[3] On the Mac OS X operating system, the Inkscape interface uses the *X11*-window layer, available on the 10.4 and 10.5 installation DVDs. The non-native

interface lacks the look and feel of “normal” Mac programs. Fear not, it will still work, although starting Inkscape may take a bit longer than other programs, especially the first time. A native version of Inkscape is in the works.

[4] Inkscape uses the terms *Canvas* and *Page* inconsistently. In this manual, *Canvas* will refer to the entire drawing area while *Page* will refer to a specified region of the *Canvas* corresponding to a defined output area.

Chapter 1. Quick Start

Table of Contents

The Swedish Flag - A Short Example

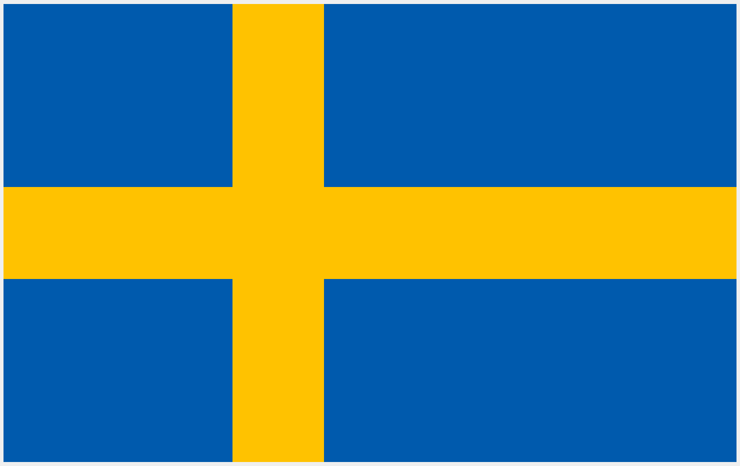
Inkscape » Quick Start » The Swedish Flag - A Short Example

Index

The Swedish Flag - A Short Example

Updated for v0.46 (New Grids)

We will use Inkscape to draw a simple flag, that of Sweden. This example will cover: setting a custom drawing size, setting up a *Grid* to help precisely place objects, the use of the *Rectangle Tool*, changing the color of objects, and finally saving a drawing and exporting the drawing into a form suitable for use on a web page.



Flag of Sweden

The steps we'll take are:

- Start Inkscape.
- Set the drawing size.
- Set up a *Grid* to guide drawing objects.
- Draw the flag background.
- Draw the cross.
- Set the colors of the background and cross.
- Save and export the drawing.

It is assumed that you know how to start Inkscape and to use a mouse, touch pad, or tablet to select menu items and move scroll bars.

Procedure 1.1. Drawing the Swedish Flag

1. **Start Inkscape.**

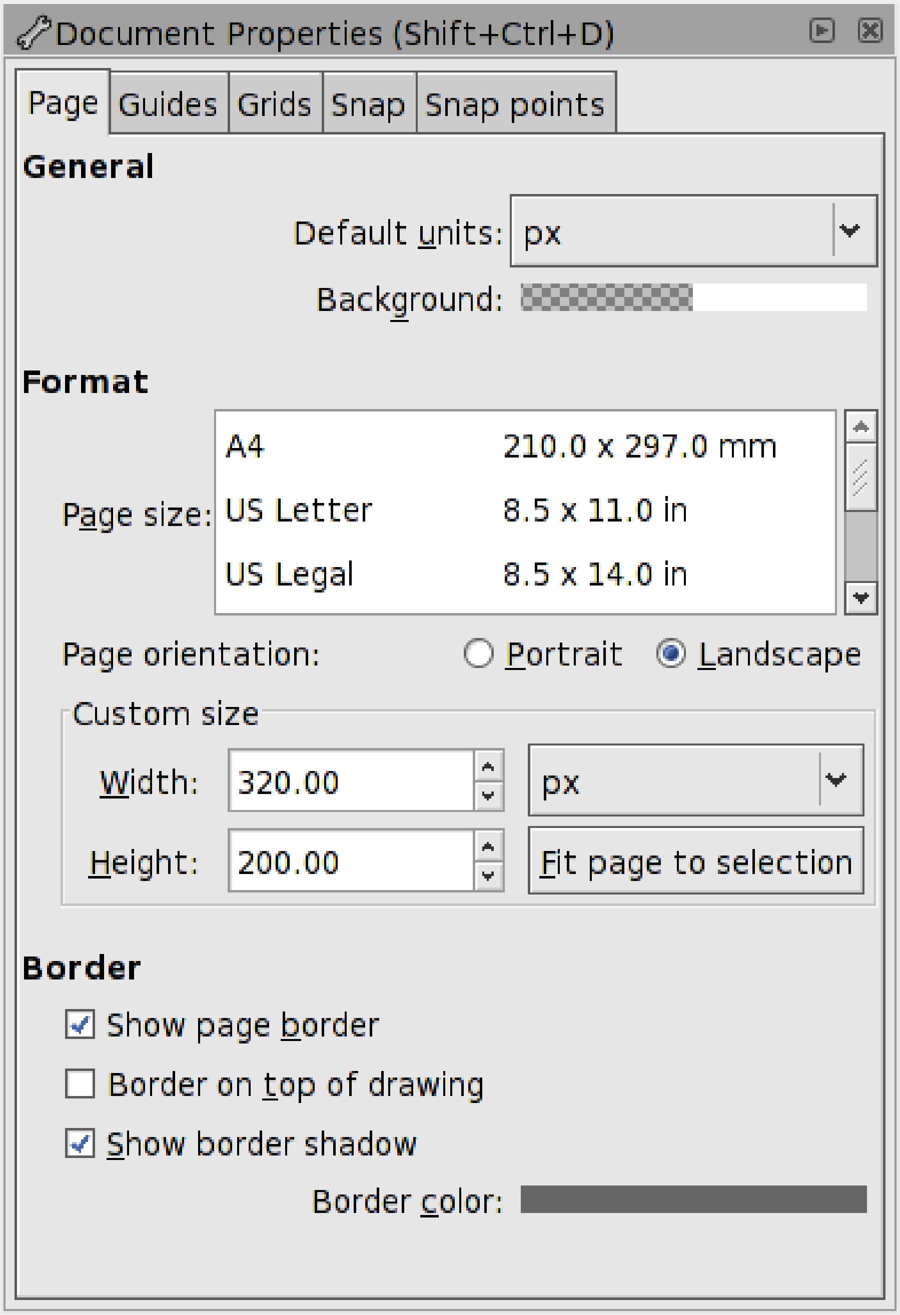
   The program will open a single window with a default page size.
2. **Set the page size to the desired flag size.**

   The correct width to height proportion of the Swedish flag is 16 to 10. We will set the page size to a 320 by 200 pixel area. What is important here is the ratio. The size of drawing when printed or exported to a bitmap can be changed later (by default, a pixel corresponds to a screen pixel when exported).

   a. **Open the *Document Properties* dialog.**

      Open the *Document Properties* dialog by selecting File → Document Properties... (**Shift**+**Ctrl**+**D**).
   b. **Set page size.**

      In the newly opened window, set *Custom size*: *Units* to "px" using the drop-down menus. Then set the flag size by changing *Custom size*: *Width* to 320, and *Height* to 200. This can be done by typing the numbers into the entry boxes next to the labels (one could use the small up and down arrows to change the entered value but this does not work well when an entered number could be non-integer). Note that when you type in new values, changes don't take effect until you hit **Return**, click on a different entry box, or move the cursor from the *Document Properties* dialog to another Inkscape window. *Page orientation* will automatically change to *Landscape*.



Document Properties (Shift+Ctrl+D)

Page | Guides | Grids | Snap | Snap points

General

Default units: px

Background:

Format

Page size:

A4	210.0 x 297.0 mm
US Letter	8.5 x 11.0 in
US Legal	8.5 x 14.0 in

Page orientation: ○ Portrait ● Landscape

Custom size

Width: 320.00 px

Height: 200.00

Fit page to selection

Border

☑ Show page border

☐ Border on top of drawing

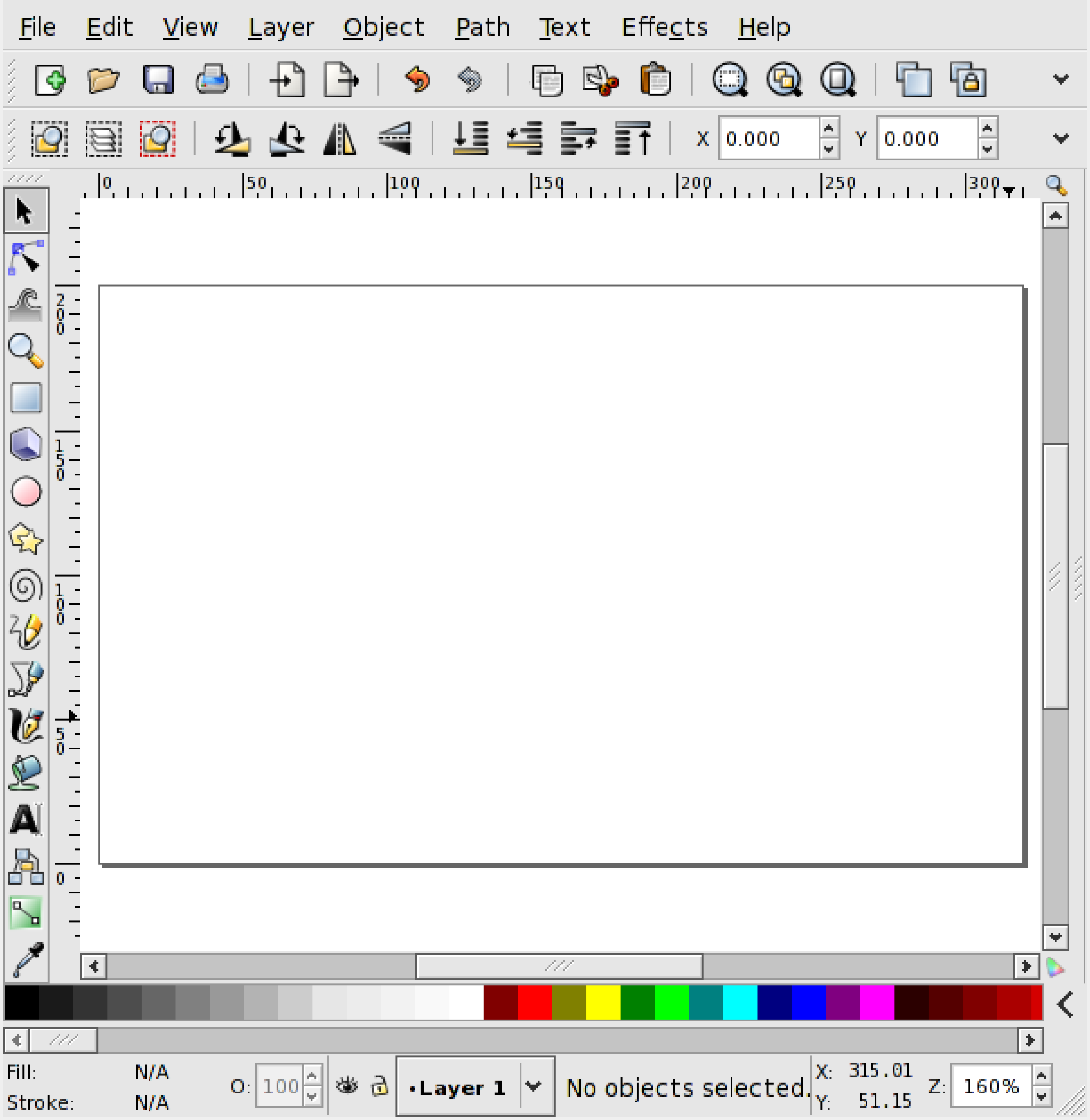
☑ Show border shadow

Border color:

Document Properties dialog.

3. **Fit the page into the drawing area.**

   The page is now a small rectangle at the bottom of the drawing area. To fit the page to the drawing area, click on the *Zoom-Page* icon in the *Command Bar*, or use the keyboard shortcut: **5**. The Inkscape window should then look like this:



The Inkscape window after you have adjusted the page size and zoom level.

- **Set a *Grid*.**

While the *Document Properties* dialog is open, we will make one more change that will make drawing the background and cross easier. A *Grid* is a set of (usually) horizontal and vertical lines that provide a guide for drawing objects. Optionally, objects will “snap” to a *Grid* when being drawn or moved, enabling accurate drawing. A *Grid* will not show when the drawing is printed or exported as a bitmap.

a. **Create a *Grid*.**

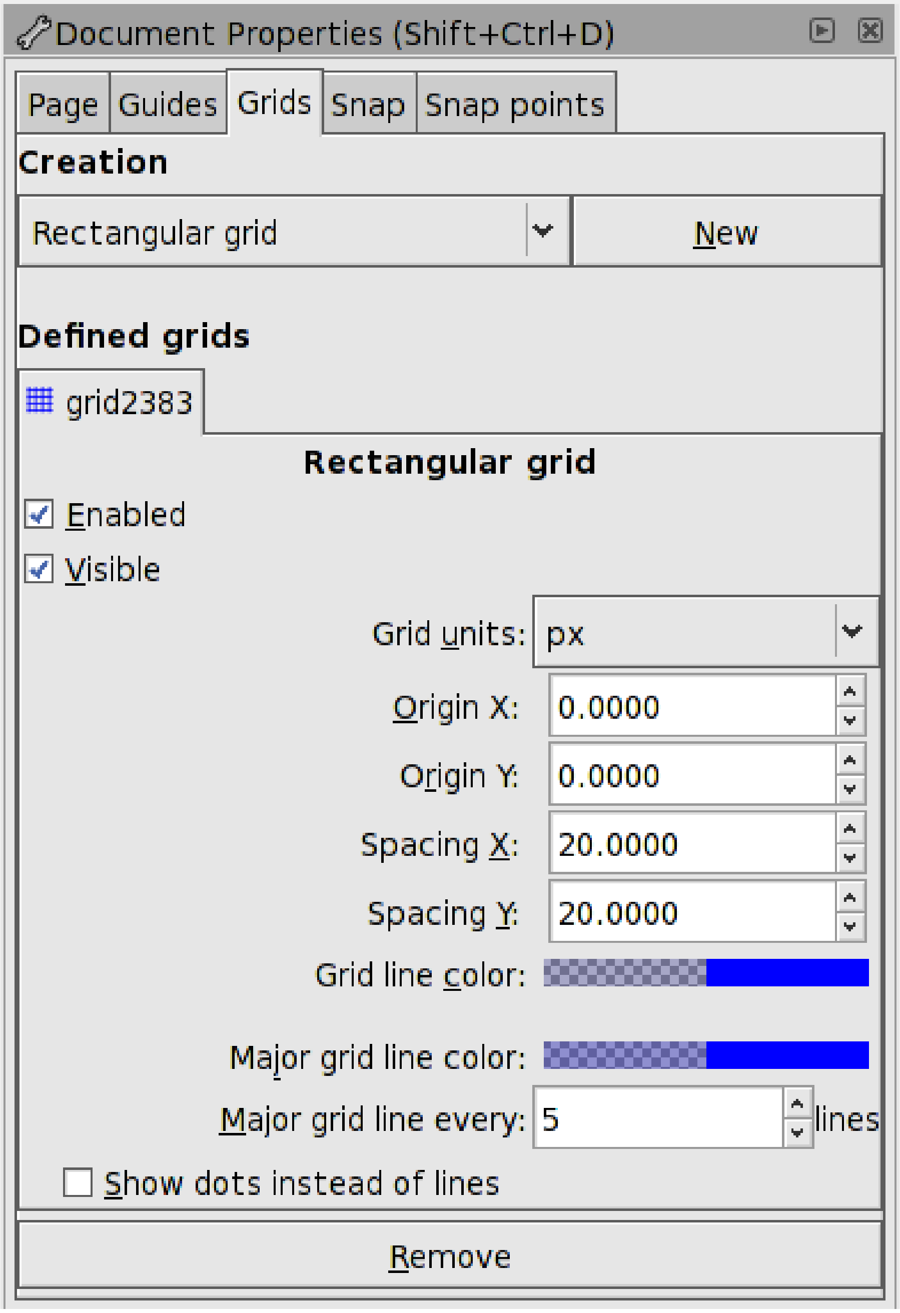
Select the *Grids* tab in the *Document Properties* dialog. Under the *Creation* section, select *Rectangular* *grid* from the drop-down menu (if not already selected) and the click on the *New* button. You should see a grid of blue lines on the canvas. If you don't, make sure both the *Enabled* and the *Visible* boxes are checked in the *Defined grids* section. If they are checked, toggle on the global visibility of *Grids* using the command View → Grid (#).

b. **Adjust the *Grid* spacing.**

You can now adjust the *Grid* to match the cross. The dimension and position of the cross is given by Swedish law. The vertical bar is to be placed between 5/16ths and 7/16ths of the flag width while the horizontal bar is to be placed between 4/10ths and 6/10ths of the flag height.

The default *Grid* has minor divisions every pixel and major divisions every 5 pixels. Depending on the zoom level, not all divisions may be displayed. A more useful *Grid* for drawing the flag would be one with divisions every 20 pixels so that the *Grid* lines divide the flag width into 16 parts and the flag height into 10 parts.

To change the scale of the *Grid* go back to the *Grids* tab of the *Document Properties* window.



Rectangular grid

grid2383

☑ Enabled

☑ Visible

Grid units: px

Origin X: 0.0000

Origin Y: 0.0000

Spacing X: 20.0000

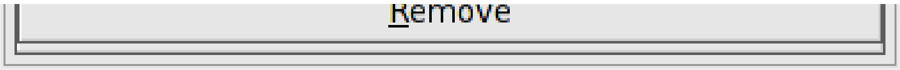
Spacing Y: 20.0000

Grid line color:

Major grid line color:

Major grid line every: 5 lines

☐ Show dots instead of lines

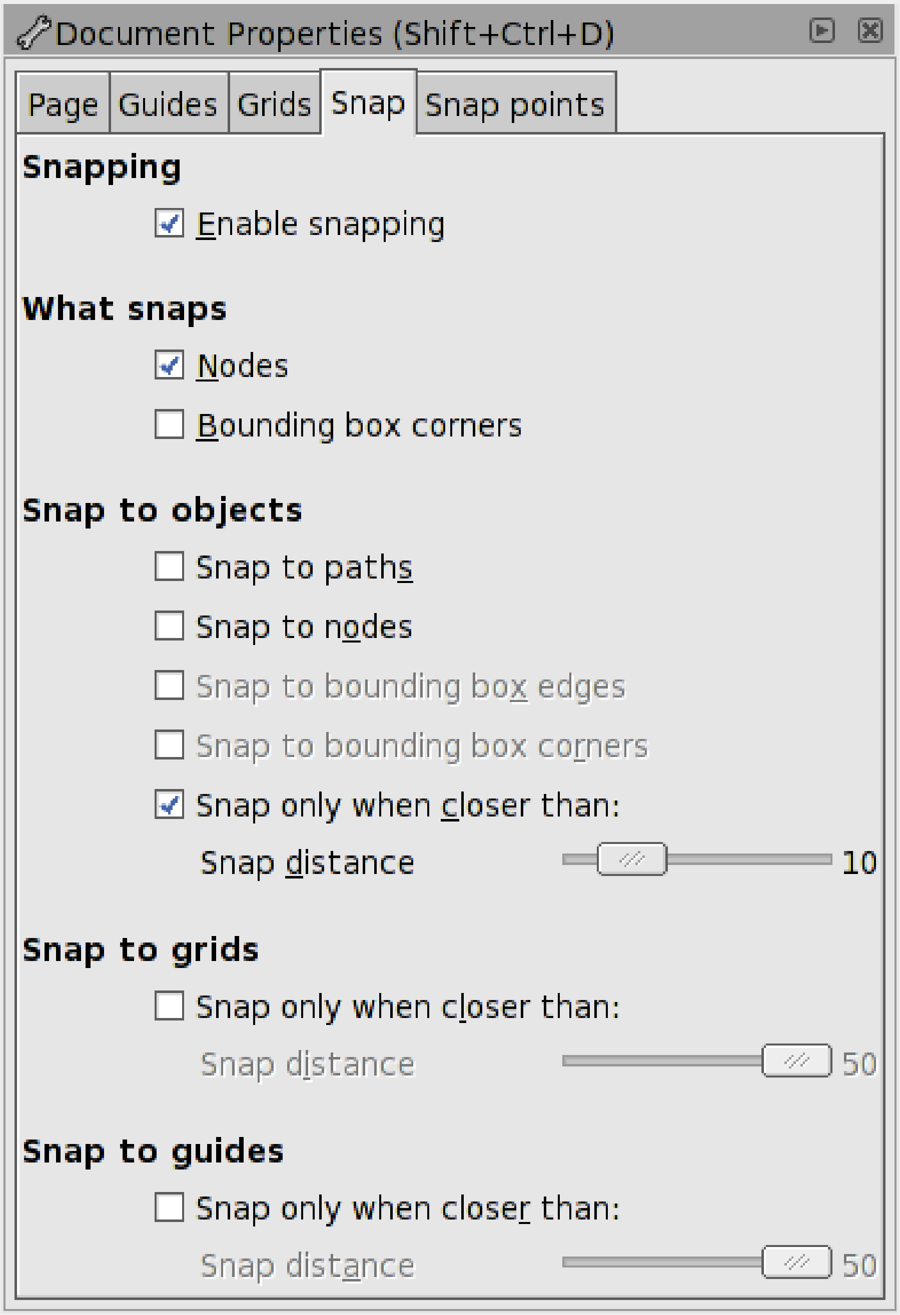


Document Properties dialog, *Grids* tab.

Set both *Spacing X* and *Spacing Y* to 20.

To help draw the flag accurately, we will turn on the snapping of nodes to the grid lines. Select the *Snap* tab. Check the *Enable snapping* box under the *Snapping* section. In the *What snaps* section check the box labeled *Nodes*. This will cause the corners of the rectangles we will draw to align with the grid. Uncheck the box labeled *Bounding box corners*.[5]

Next, in the *Snap to grids* section, uncheck the *Snap only when closer than:* box. This will force snapping to the *Grid* to always occur.



Snapping

☑ Enable snapping

What snaps

☑ Nodes

☐ Bounding box corners

Snap to objects

☐ Snap to paths

☐ Snap to nodes

☐ Snap to bounding box edges

☐ Snap to bounding box corners

☑ Snap only when closer than:

Snap distance 10

Snap to grids

☐ Snap only when closer than:

Snap distance 50

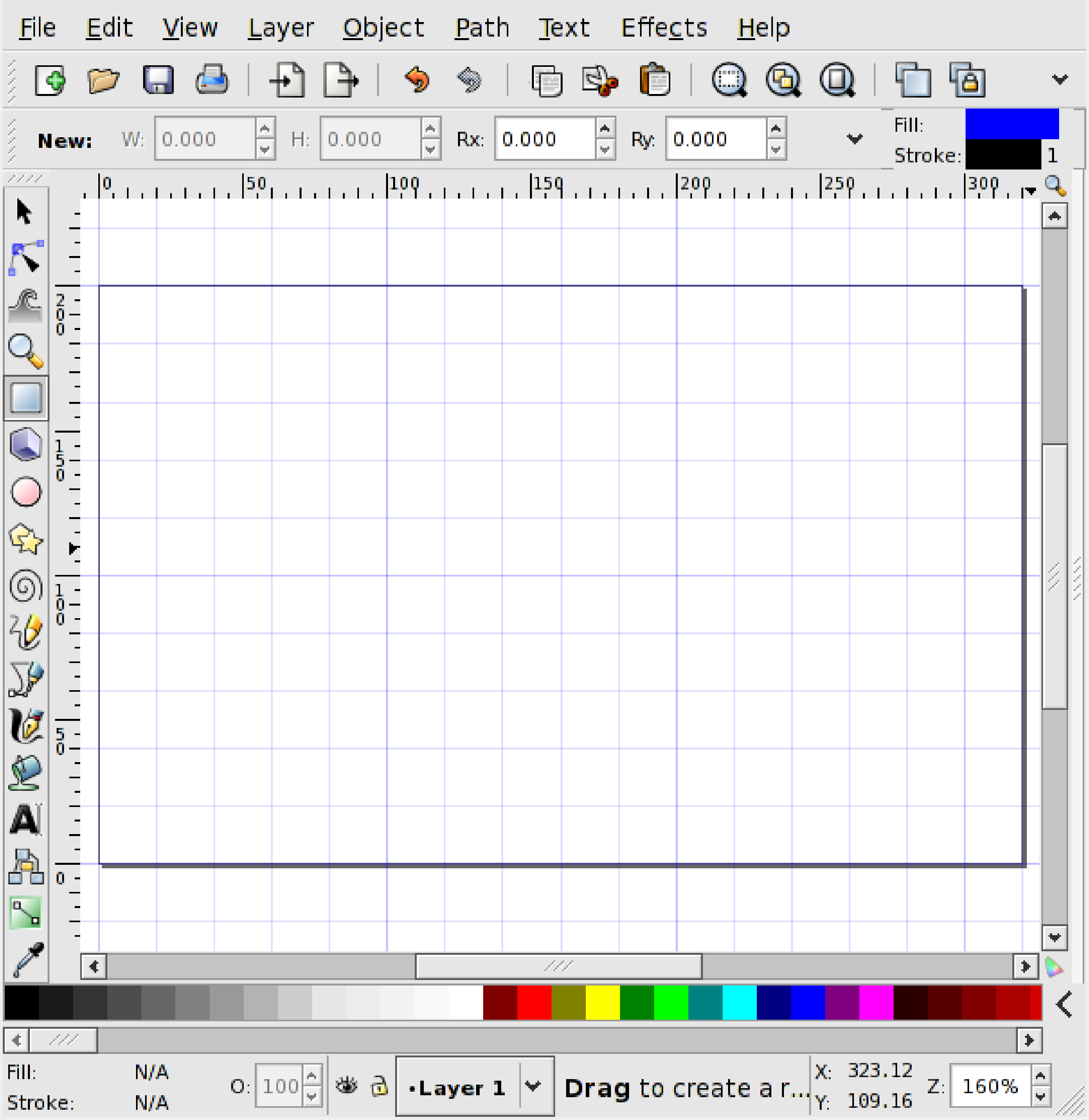
Snap to guides

☐ Snap only when closer than:

Snap distance 50

Document Properties dialog, *Snap* tab.

After making the changes, you may close the *Document Properties* window. The Inkscape window should look like this:



Drawing area with *Grid* turned on and adjusted.

- **Draw the flag background.**

The flag has a light-blue background. There are several ways to accomplish this; we'll use a filled rectangle.

a. **Select the *Rectangle Tool*.**

Click on the *Rectangle Tool* icon in the *Tool Box* on the left of the Inkscape window (or use the keyboard shortcut **F4**) to select the *Rectangle Tool*.

> **Tip**
>
> One nice feature of Inkscape is that there are very good built-in hints. While the mouse pointer is over the *Rectangle Tool* icon, you'll see a *tool tip* describing the use of the *Rectangle Tool*. This features is present for almost all icons and objects in Inkscape. The *tool tip* usually includes the keyboard short cut for an icon, in this case (F4), indicating that pressing **F4** would be another way of selecting the *Rectangle Tool*.

Once you have selected the *Rectangle Tool*, move the pointer over the drawing area. The pointer will become a rectangle. This signifies you are ready to draw a rectangle or square. The small cross at the upper-left corner of the pointer indicates the active point.

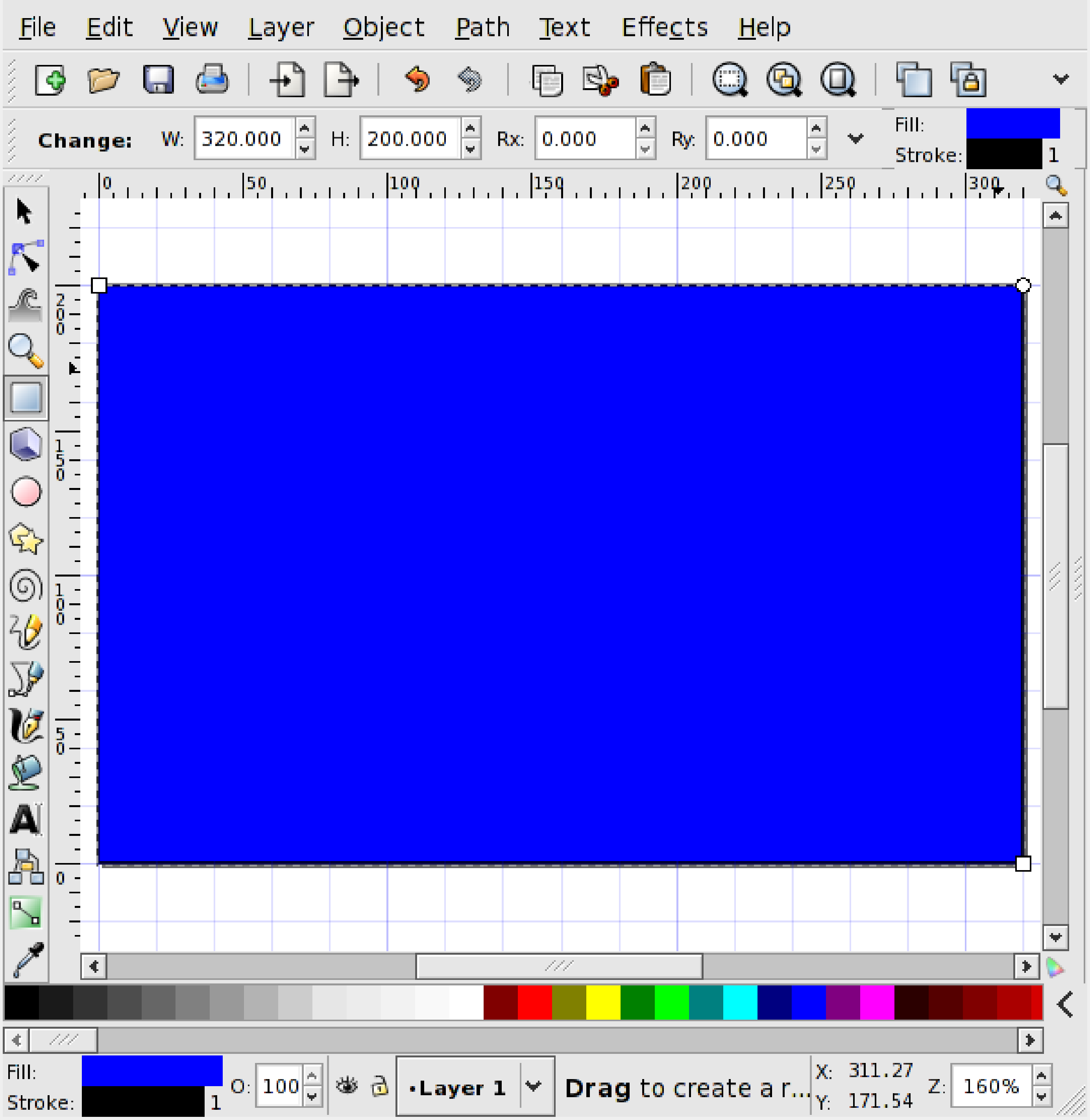
> **Tip**
>
> At the bottom of the window, there is a *Notification Region* (in the *Status Bar*). This tells you what actions you can perform with the selected tool. Note that the region isn't always large enough to show all the options. Moving the mouse over the region will pop up a *tool tip* showing the full content. One could also widen the Inkscape window to see more of the region. When the *Select Tool* is in use, the *Notification Region* will also tell you the number and type of objects that are currently selected.

- **Draw the background rectangle.**

To draw the background rectangle, follow the hint in the *Notification Region*. Click-drag the pointer from one corner to the opposite corner of the page area.

Note how the corners of the rectangle snap to the *Grid*. If your rectangle doesn't match the page size, you can use one of the drag handles (little squares) in the upper left or lower right of the rectangle to adjust the size of the rectangle. (The circle at the upper-right corner has a different function. It is used to round the four corners of a rectangle.) As you drag the squares around, they will snap to the *Grid*.

If you make a mistake, you can click on the *Undo* icon in the *Command Bar*, use Edit → Undo (**Ctrl+Z**) from the *Menu Bar*, or use the keyboard shortcut **Ctrl**+**Z** to undo the change.



Change: W: 320.000 H: 200.000 Rx: 0.000 Ry: 0.000

Fill: Stroke: 1

Fill: Stroke: 1 O: 100 •Layer 1 **Drag** to create a r... X: 311.27 Y: 171.54 Z: 160%

Background rectangle.

The background color will need to be changed. One could do that now, but it is easier to wait and change it at the same time the cross color is set.

- **Draw the cross.**

The cross consists of a horizontal bar and a vertical bar.

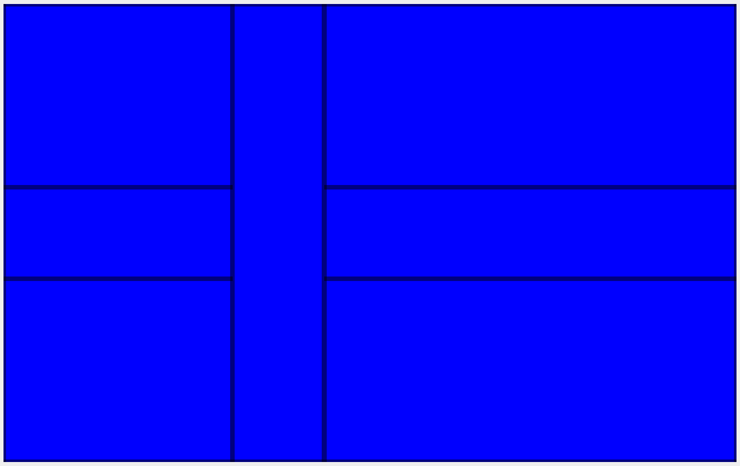
a. **Draw the horizontal bar.**

   With the *Rectangle Tool* still selected, create a bar by starting six grid units up on the left side of the flag and click-dragging the pointer until four units up on the right side of the flag. Notice that the rectangle *[Fill](#)* color is the same as the background color but you should still be able to see the new bar. Make any corrections to the size and position of the rectangle that are necessary.

If you can't see the horizontal bar... don't panic! Inkscape often uses the *Current style* (attributes: color, line style, etc.) to draw new objects. The *Current style* is set to that of the last object where the style was modified (including that of a previous Inkscape session). If the border color matches the *Fill* color of the rectangle or if drawing the border has been turned off, AND if the *transparency* or *Alpha* is set to 100% you will have the same situation as a polar bear in a snow storm. There are many fixes but the easiest one is to change the horizontal bar color to be different from the background color. This can be done by clicking on any of the color samples in the *Palette* while the newly drawn bar is still selected (indicated by the dashed line around the perimeter). If a color sample in the *Palette* is clicked on when no object is selected, the *Current style* will be changed to use that color for the *Fill* of the next *Rectangle* drawn.

b. **Draw the vertical bar.**

The vertical bar should extend between 5 and 7 units from the left.



The display after both the horizontal bar and vertical bar are drawn. The border lines show how one rectangle of the cross overlaps the other.

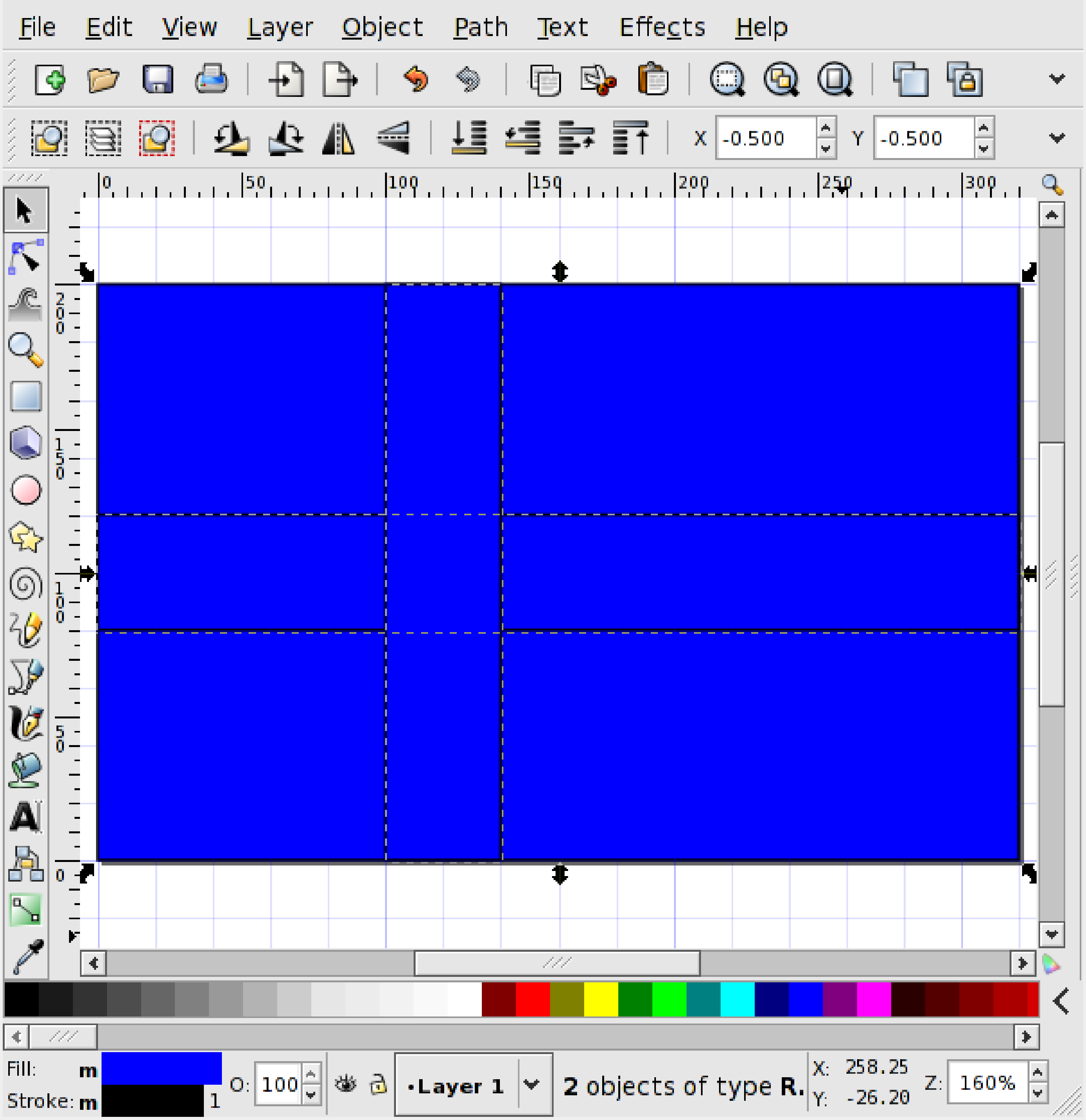
- **Merge the bars into a cross.**

One could stop here. After removing the border for the bars, one would have the desired cross. But it might be better (and at least more pedagogical) to merge the bars into a cross, so that the cross is one *object* rather than two.

i. **Select both the rectangles.**

Both bars needed to be selected at once. This can be done with the *Select Tool* and the **Shift** key. Enable the *Select Tool* by clicking on the ↖ icon in the *Tool Box* or by using one of the keyboard shortcuts **F1** or **s**.

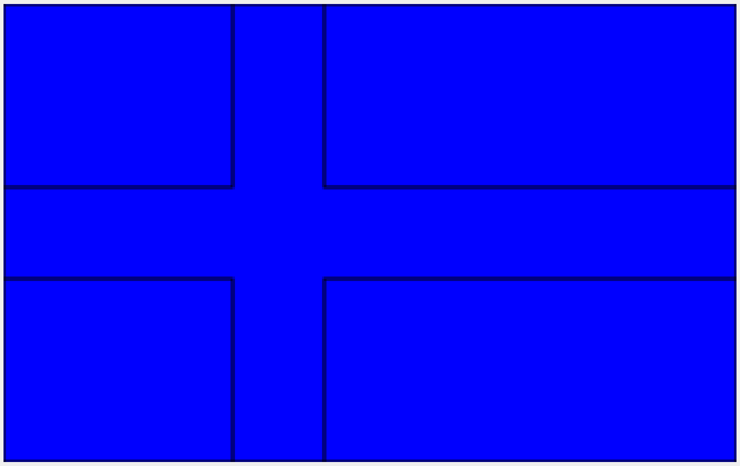
After changing to the *Select Tool* (indicated by the pointer changing to an arrow when over the canvas or crossed arrows when over an object), click on one of the rectangles in the cross. Then, while holding the **Shift** key down, click on the other rectangle in the cross. Both rectangles should be selected as indicated by their dotted borders. Note that the background rectangle is not surrounded by a dotted border and that the *Notification Region* reports that two objects are selected.



The two rectangles that form the cross are selected.

- **Merge paths.**

The two rectangles can be combined by merging their *Paths*. (Here, the *Path* is the border of the rectangle.) To merge the two rectangles, select Path → Union (**Ctrl++**) from the *Menu Bar*. The rectangles are now merged into one object.



The cross after the rectangles have been merged.

The cross is no longer made of *Rectangle* objects but is instead defined as a *Path*.

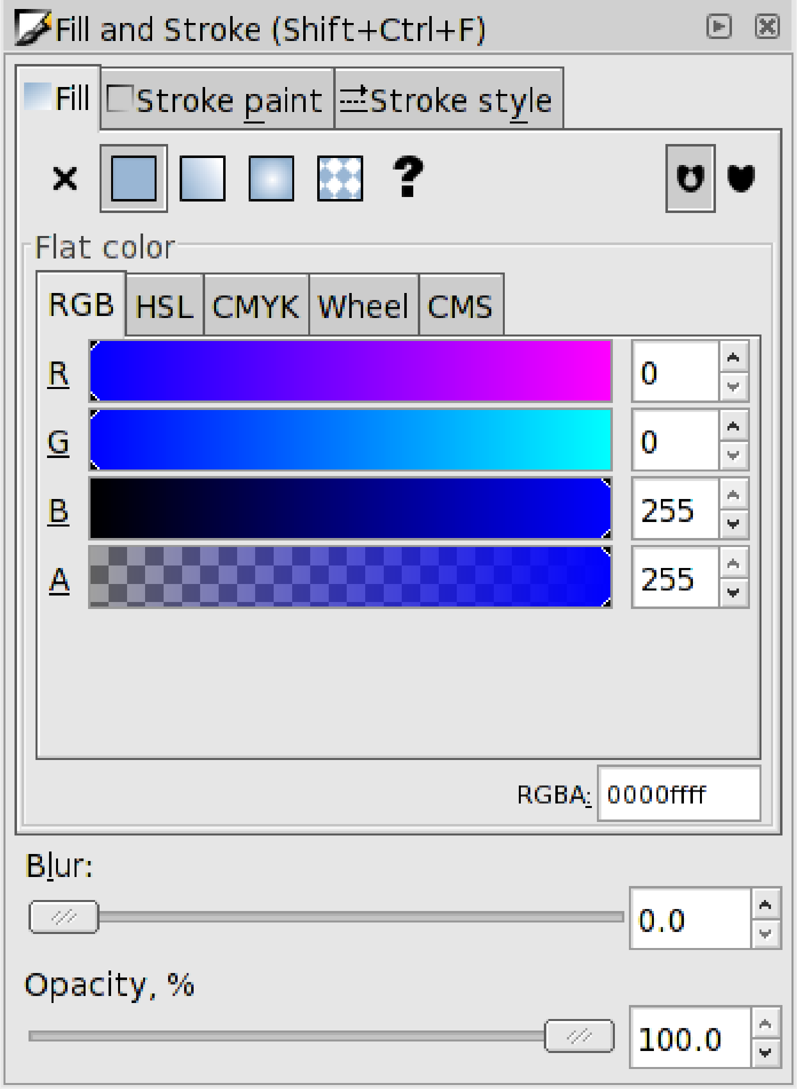
- **Set the colors of the background and cross.**

The next step is to adjust the colors of the background and cross to the colors of the Swedish flag. We'll use the *Fill and Stroke* dialog so that we can precisely set the correct colors.

a. **Bring up the *Fill and Stroke* dialog.**

   Open up the *Fill and Stroke* dialog by clicking on the *Fill and Stroke* icon in the *Command Bar*, clicking on the *Fill* part of the *Style Indicator* in the *Status Bar*, or using the keyboard shortcut **Shift**+**Ctrl**+**F**.

   The dialog will by default be docked on the right inside the Inkscape window. If you have room, undock the dialog by dragging on the top bar of the dialog (the gray bar with *Fill and Stroke* written in it). Drop the dialog outside of the Inkscape window. Hit **5** to recenter the drawing inside the main window (or use the **Middle Mouse** button to drag the drawing back to the center).



R 0
G 0
B 255
A 255
RGBA: 0000ffff
Blur: 0.0
Opacity, % 100.0

Fill and Stroke dialog.

Make sure that the *Fill* tab is highlighted at the top of the dialog; if not, click on the tab. The parameters in the *Fill and Stroke* dialog apply to the currently selected drawing object(s).

- **Set the *Fill* background color.**

Select the background rectangle by clicking on it with the *Select Tool*. The *Notification Region* should report that one rectangle is selected and the *Style Indicator* will show the *Fill* color and the *Stroke* color of the rectangle. The *Flat Color* 🟦 icon should be highlighted. If not, click on it.

There are several ways to specify the desired *Fill* color. We'll use the *Red-Green-Blue (RGB)* mode (select the *RGB* tab if not already highlighted). In this mode, a color is specified by setting the amount of each of the three primary colors. The scale extends from 0 to 255. One can change the amount of each primary color via sliding the little triangles left or right on the bars labeled R, G, B, or by changing the numbers in the boxes to the right of the bars (via typing or using the up/down arrows). For the Swedish flag, the background color is specified by the the NIS standard color 4055-R95B, which is equivalent to the values: Red: 0, Green: 90, Blue: 173. The fourth entry is *Alpha* (A) or transparency, which indicates how opaque the object should be. We want our flag to have a solid, non-see-through background, so *Alpha* should be set to 255 (range is 0 to 255). Likewise, the *Master opacity* slide should be set to 100% (1.0 in v0.44).

One additional step is to turn off any *Stroke* (border) color. In the *Fill and Stroke* dialog, select the *Stroke paint* tab and click on the *No paint* ✕ icon to turn off the stroke.

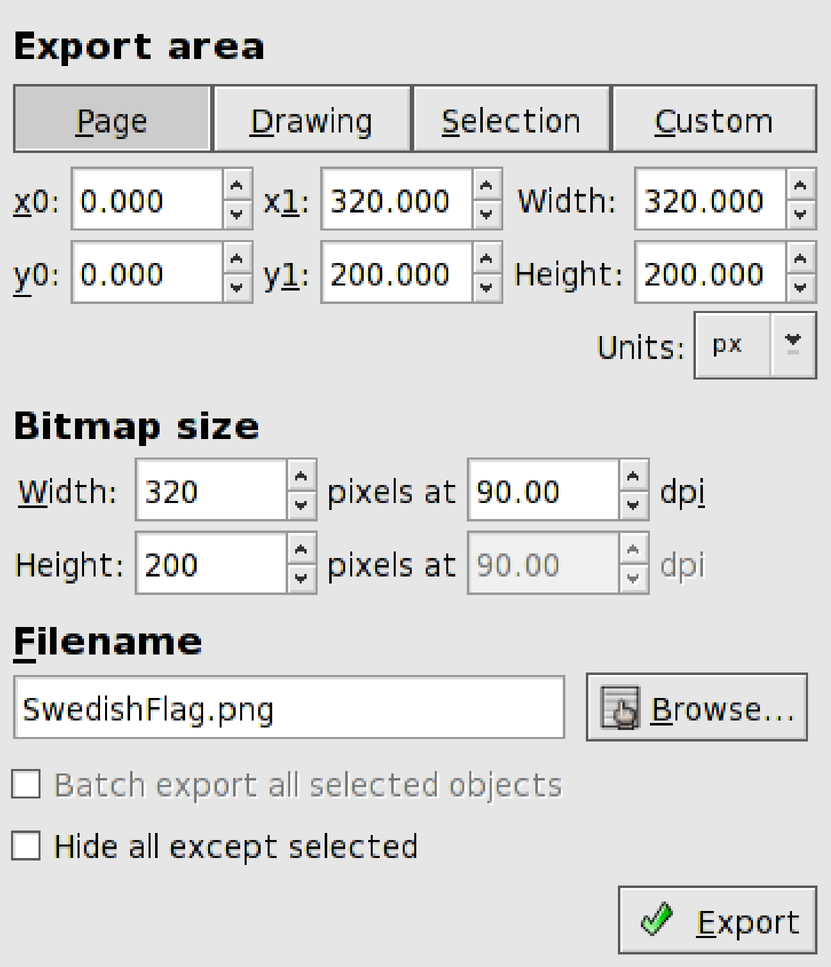
- **Set the cross *Fill* color.**

Select the cross and change the *Fill* color following the previous instructions. When the cross is selected, the *Notification Region* will report that a *Path* with 12 nodes is selected. You'll need to reselect the *Fill* tab. This time, set the colors to R: 255, G: 194, and B: 0. Also turn off the border as done previously.

- **Save and export your work.**

Now is the time to save your work. Select File → Save As... (**Shift+Ctrl+S**). The dialog that appears will depend on your operating system. Select the folder or directory where the drawing should be saved and give the drawing an appropriate name. Finally click on the *Save* button.

One last step is to export your file as a *PNG (Portable Network Graphic)* bitmap that can be used by other graphics programs or on a web page.[6] Bring up the *Export Bitmap* dialog: (File → Export Bitmap... (**Shift+Ctrl+E**)).



x0: 0.000 x1: 320.000 Width: 320.000

y0: 0.000 y1: 200.000 Height: 200.000

Units: px

Width: 320 pixels at 90.00 dpi

Height: 200 pixels at 90.00 dpi

SwedishFlag.png

☐ Batch export all selected objects

☐ Hide all except selected

Export Bitmap dialog.

There are four options for choosing the area to export: *Page*, *Drawing*, *Selection*, and *Custom*. In our case, the *Page* and *Drawing* areas are the same. Select either one, then enter or select a file name at the bottom and click on the *Export* button to save the drawing as a *PNG*. You should now have a *PNG* file with a flag as shown at the beginning of this section. Note, the dialog will remain on the screen even after a successful export.

[5] The *bounding box* includes (normally) the width of any visible border line. The finished flag will not have any visible border lines, but the rectangles used for the drawing of the flag, will have visible borders until they are removed in one of the last steps. If the placement of the rectangles is done using *bounding boxes*, the final placements of the rectangles will be off slightly.

[6] Native support for *SVG* graphics is being incorporated into web browsers. Until it is widely available it may be better to rely on old-fashion bitmap images.

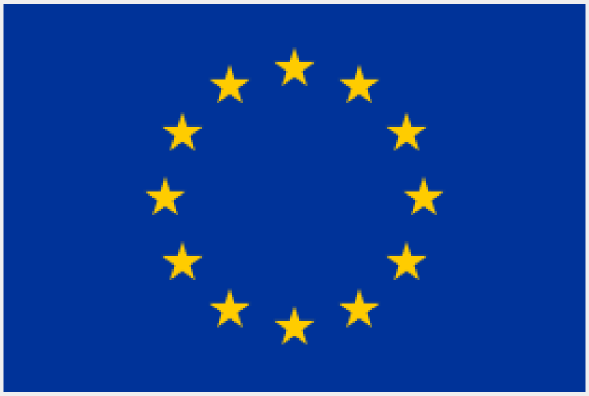
The Anatomy of the Inkscape Window

Table of Contents

The European Flag - A More Elaborate Example

The European Flag - A More Elaborate Example

We will use Inkscape to draw a more complicated flag, that of the European Union (EU). This tutorial will cover using the *Star Tool* to draw a regular star, using *Guide Lines* for positioning, making copies or clones of an object, and precisely moving those copies to their proper places.



Flag of the European Union.

The steps are:

- Set basic drawing parameters (flag size, grid, background color).
- Add *Guide Lines*.
- Draw a single star.
- Duplicate the star and position the duplicates.

Procedure 1.2. Drawing the European Union Flag

1. **Set up the drawing.**

   To begin, start Inkscape. The page size needs to be set to the correct proportions for the EU flag, which has a 3 to 2 width to height ratio. We will use a 270 by 180 pixel area. This will facilitate drawing and placing of the stars to the EU specifications.

   Follow the instructions for setting the page size and creating a grid given in the Swedish flag example but set the flag width to 270 and the flag height to 180 pixels. Also set the grid spacing to 5 pixels rather than 20 (so the center of the flag can easily be located). Enable snapping of nodes in the *Snap* tab.

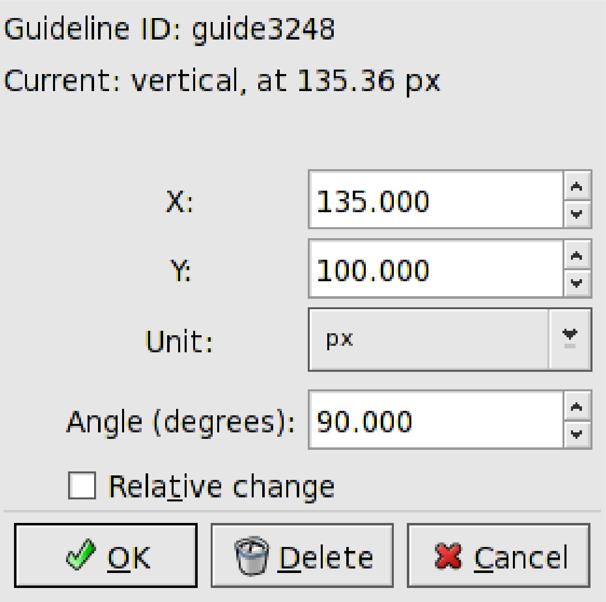
   Draw a rectangle for the flag background that covers the entire page. Next use the *Fill and Stroke* dialog (Object → Fill and Stroke... (**Shift+Ctrl+F**)) to set the color of the background to the officially prescribed *RGB* color:

0, 51, 153. Check that the *Alpha* (A) value is 255 and the *Master opacity* is 100% (1.00 in v0.44)).

2. **Add *Guide Lines*.**

   For placing the stars, it is easiest to draw the first star at the center of the flag (at 135, 90). You can then use simple translations to move the stars to their final positions. You can make it easier to keep track of the center point by adding *Guide Lines*. The *Guide Lines* are lines that, like the *Grid*, are not part of the actual drawing.

   To add a *Guide Line*, click-drag starting on a *Ruler* near the middle and finishing at the desired point on the canvas. (If you start near the edge, an angled *Guide Line* will be created.) The *Guide Line* can be moved by selecting (with the *Select Tool*) and dragging. Be careful not to move the background! If you do, just undo the move (or **Ctrl**+**Z**). To precisely place the *Guide Line*, double click on the *Guide Line* using the *Select Tool*. A dialog will open where you can type in the exact position required.



Guideline ID: guide3248

Current: vertical, at 135.36 px

X: 135.000

Y: 100.000

Unit: px

Angle (degrees): 90.000

□ Relative change

Guide Line dialog.

   The *Guide Lines* can be turned on and off under the *Guides* tab in the *Document Properties* dialog (File → Document Properties... (**Shift+Ctrl+D**)) or via the command View → Guides (**|**). Checking the box *Snap guides while dragging* under the *Guides* tab will allow *Guide Lines* to snap to the *Grid*.

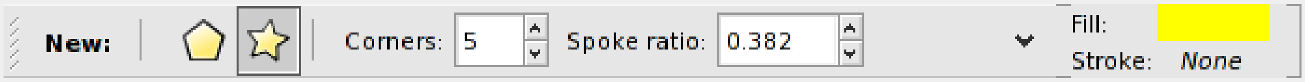
   Add both a horizontal *Guide Line* at *y*=90 px and a vertical *Guide Line* at *x*=135 px.

3. **Draw a star.**

   We need to draw a five-pointed star that will be duplicated to create the 12 stars of the flag. To avoid drawing a blue star on a blue background, click to the side of the drawing to deselect the background rectangle, then click on one of the colors in the *Palette*. When no object is selected, clicking on a color in the *Palette* will set the default *Fill* to that color.

   a. **Select *Star Tool* and set up the star parameters.**

      To draw a star, select the *Star Tool* (keyboard shortcut *****) in the *Tool Box*. According to the EU flag specification, the stars on the flag are five-pointed with one point straight up. The easiest way to get the star the exact shape is to use the *Tool Controls*.



New:

Corners: 5

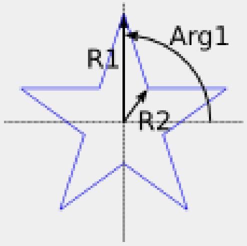
Spoke ratio: 0.382

Fill:

Stroke: None

The *Star Tool*-*Tool Controls*.

In the *Tool Controls*, set the number of *Corners* to 5. Select the *Star* icon (☆) to enable drawing of a star. The *Spoke Ratio* is the ratio of the radius of the innermost point to the radius of the outermost point of a star (R2/R1). For a “regular” five-pointed star this should be 0.382. The other entries in the *Tool Controls*, *Rounded* and *Randomized*, should both be zero (accessible by clicking on the down-arrow at the right of the *Tool Controls*).

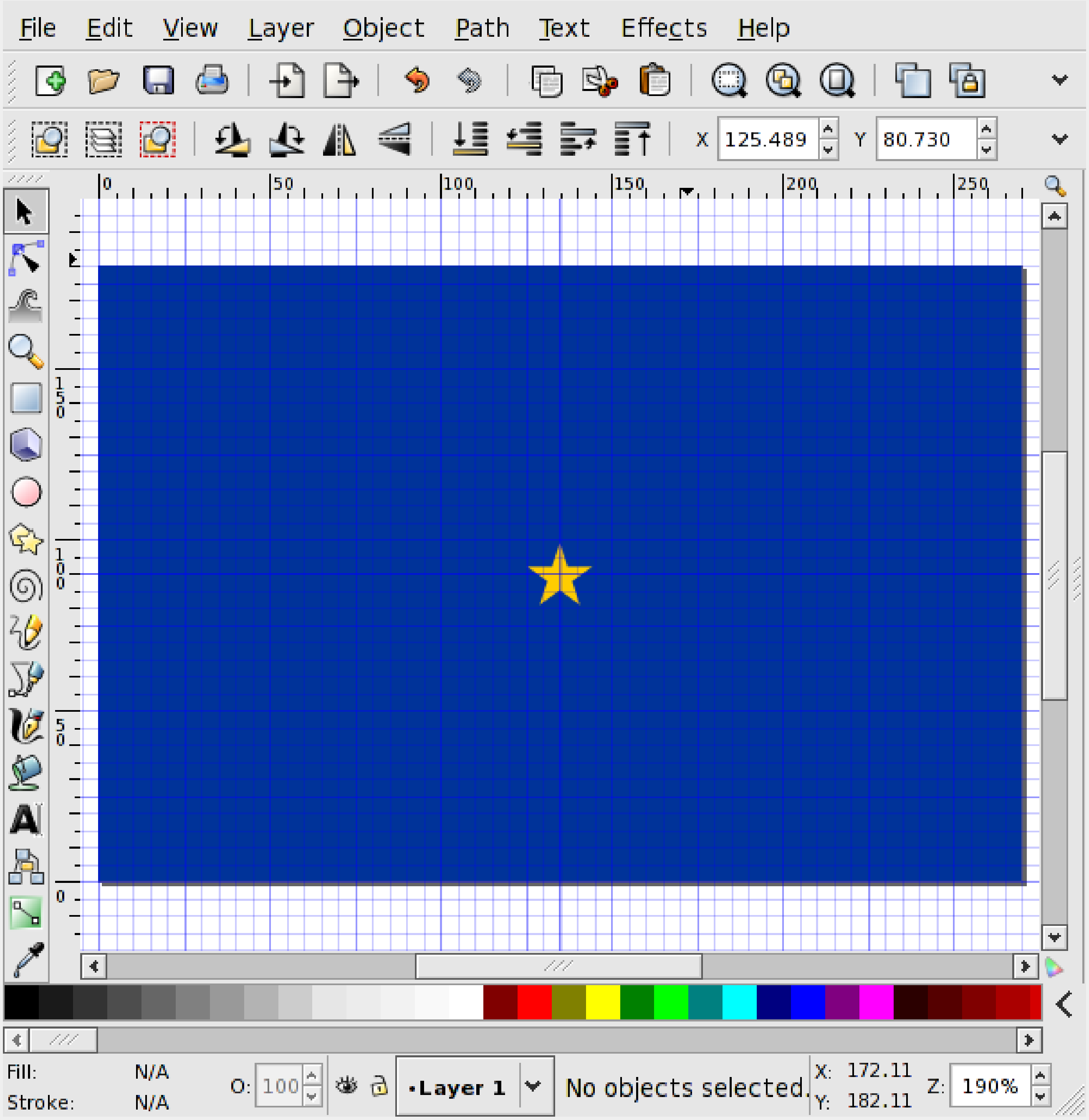


The Star dimensions.

4. **Draw the star.**

   The specified radius (R1) of the star on the EU flag is 1/18 of the flag height, or 10 pixels with our flag size. Starting with the mouse at the intersection of the *Guide Lines*, click drag upward for 10 pixels. You must move the cursor half the distance to the next grid line before you'll see a star. The upper point will snap to the *Grid*.

   Adjust the color of the star to a RGB value of (255, 204, 0). Make sure *Alpha* (A) is 255 and the *Master opacity* is 100%.



X 125.489 Y 80.730

Fill: N/A
Stroke: N/A
O: 100
•Layer 1
No objects selected.
X: 172.11
Y: 182.11
Z: 190%

After the first star has been drawn and the color adjusted.

- **Duplicate and place the 12 stars.**

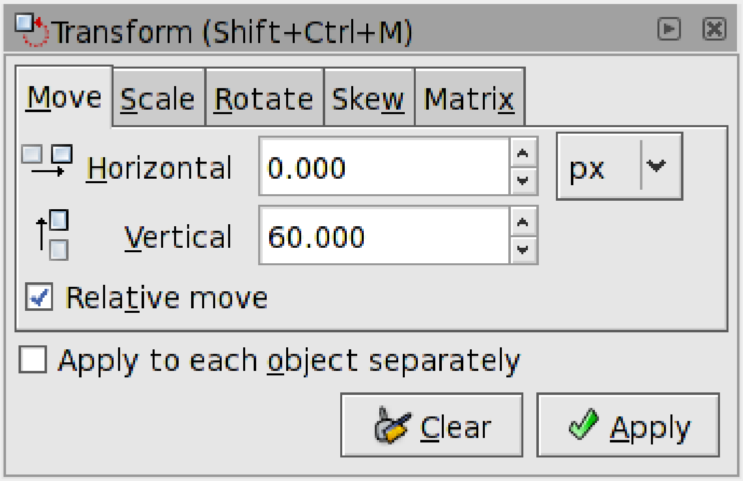
  a. **Clone a star.**

     Either click on the icon in the *Command Bar* or use Edit → Clone → Clone (**Alt+D**) to make a *Clone* of the star. A *Clone* is a copy that is linked to the original so that if you modify the original, the *Clone* will also change. This is handy if you need to make a common adjustment to all the stars (change color, enlarge, etc.).

  b. **Place a cloned star.**

     The EU flag specifies that the stars be evenly distributed on a circle that is 1/3 of the flag height and at positions corresponding to the hours of a clock. The easiest way to place the stars properly is to use the move feature on the *Move* tab of the *Transform* dialog (Object → Transform... (**Shift+Ctrl+M**)). Open the dialog, and if you have room, drag it out of the Inkscape window. To move the cloned star to the 12 o'clock position, set the

Vertical (*y*) direction to 60 pixels. Make sure the *Relative move* box is checked and the units are set to pixels (px), then click the Apply button.



Horizontal 0.000 px

Vertical 60.000

☑ Relative move

□ Apply to each object separately

The *Transform* dialog, set to move the first cloned star into place.

Next, select the original star and make a new *Clone*. This time move the *Clone* 60 pixels down (–60). Repeat for the *Horizontal* (*x*) direction. For the rest of the stars, use the eight permutations of *x*(*y*) = ±30 and *y*(*x*) = ±52 pixels (60 times the sine and cosine of 30° respectively).

After all 12 stars are placed, you can delete the original star. The links between the clones and the original star are automatically broken.

If desired, save your work and export a bitmap as for the Swedish flag.

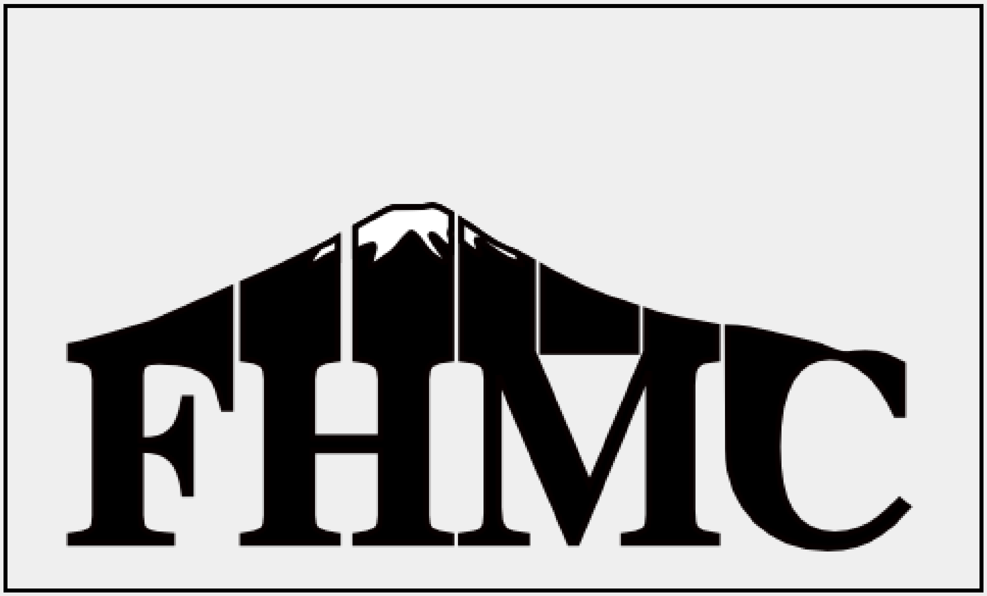
The Swedish Flag - A Short Example

Table of Contents

A Hiking Club Logo - An Exercise in Paths

A Hiking Club Logo - An Exercise in Paths

We will use Inkscape to draw a logo for the Fuji Hiking and Mountaineering Club, as shown below. This tutorial will cover the use of text, importing a bitmap for use as a guide in drawing, and manipulation of paths.



Logo for the Fuji Hiking and Mountaineering Club.

The steps are:

- Start Inkscape and set the drawing size.
- Create the text for the logo.
- Import a bitmap with the shape of Fuji mountain.
- Convert the text to a path and manipulate that path.
- Trace the Fuji mountain picture to obtain a path.
- Trim the text to the mountain shape using Fuji mountain path.
- Add snow to the mountain top.
- Add finishing touches.

Procedure 1.3. Creating the Fuji Hiking and Mountaineering Club Logo

1. **Set up the drawing.**

   To begin, start Inkscape.

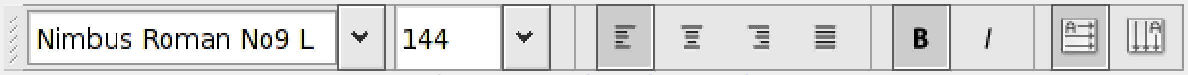
   Follow the instructions for setting the page size and grid spacing given in the Swedish flag example, but set drawing size to a width of 500 and a height of 300 pixels. Do *not* create a *Grid*.
2. **Create the text.**

a. **Enter the text.**

Select the *Text Tool* from the *Tool Box* (keyboard shortcut **F8**). Click on the left side of the page to establish a starting point for the text. You should see a blinking bar. Type the initials for the club “FHMC”; the text should appear in a small size on the page.

b. **Adjust the text.**

The text is too small and may not use the most suitable font. To change the attributes of the text, use the items in the *Text Tool*-*Tool Controls*.

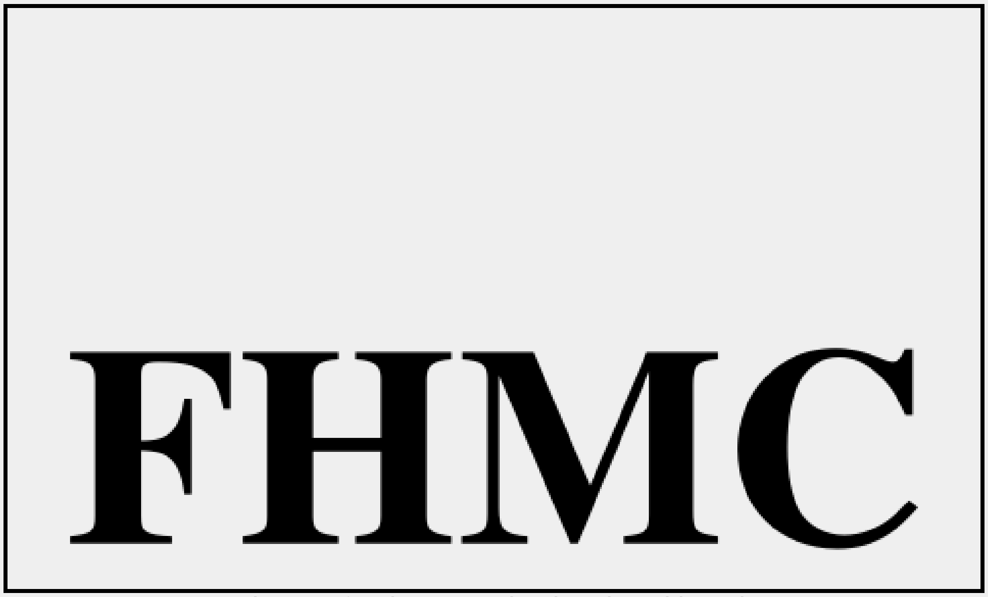

Nimbus Roman No9 L

144

The *Text Tool*-*Tool Controls*.

With the text selected, choose a suitable font from the pull-down menu on the left. Nimbus Roman No9 L is a good freely available font with the wide serifs needed for the logo. Select the **Bold** style by clicking on the “**B**” in the bar, and set the *Font size* to 144. The changes to the text are shown immediately.

Finally, center the text near the bottom of the drawing by using the *Select Tool* and dragging the text down.



The text for the logo, sized and positioned.

- **Import the guide for the mountain shape.**

We'll use as a basis for the shape of Fuji San a bitmap tracing of the mountain. You could use any suitable drawing or picture of the mountain (in *PNG*, *GIF*, or *JPEG* format). You can download the same image used here from the book's website: http://tavmjong.free.fr/INKSCAPE/.

a. **Import the bitmap.**

Import the bitmap using the *Import* dialog (File → Import... (**Ctrl+I**)).

b. **Adjust the bitmap.**

The bitmap's image size doesn't match well with the text. The easiest way to adjust the size is to select the image with the *Select Tool* (keyboard shortcut **F1**). When the image is selected, a set of double-headed arrows appears around the bounding box (dotted line) of the image. Dragging on the handles will scale the image. Dragging while holding down the **Ctrl** key will keep the width to height ratio constant. Dragging on a non-transparent part of the image will move the whole image. Note that if you click on the image twice with the *Select Tool*, the corner arrows change to rotation arrows. Just click on the image one more time to restore the scale arrows. One can also use the **Arrow** keys to move the image. Note: You may want to decrease the zoom a bit (**3** when image selected) or widen the Inkscape window before enlarging the image.

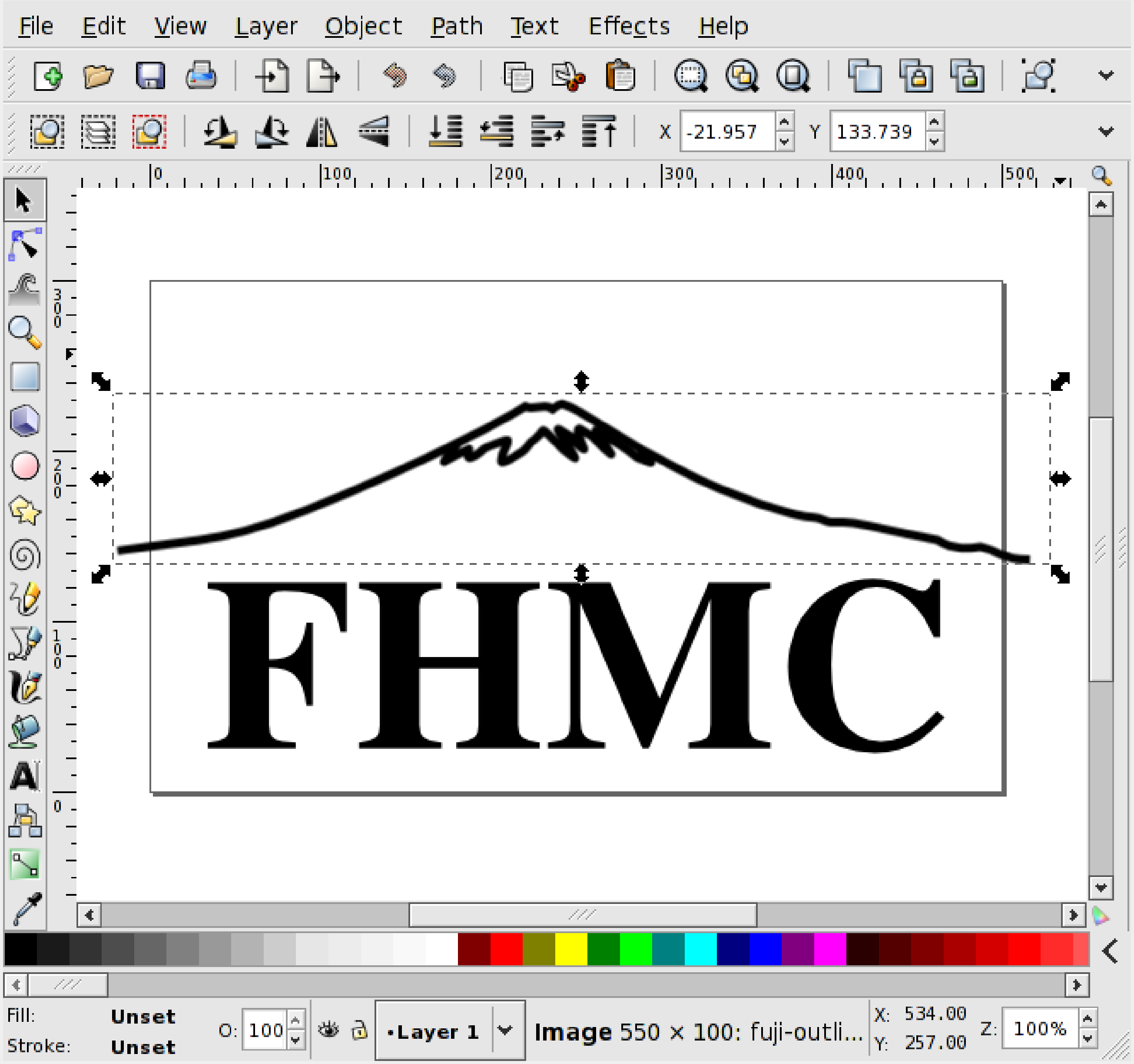
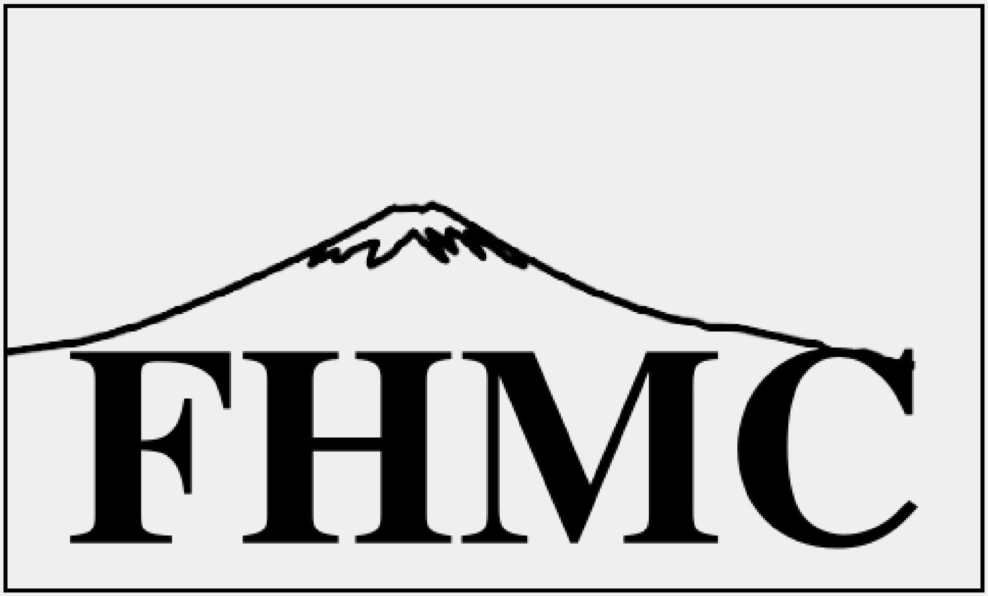


Image 550 × 100: fuji-outli...

Image selected and with arrows, ready for scaling.

Drag on the corner arrows while holding down the shift key and drag the image until you are happy with the scale and placement. I have chosen to center the top of Fuji over the right side serif of the H.



The outline of Fuji San, sized and positioned.

- **Manipulate the text.**

In this section, we will convert the text to a *Path* so we can alter the shape of the letters. The text needs to be extended upward, above the outline of the mountain so that we can *clip* it to match the shape of the mountain. We will also make a few additional cosmetic changes to the letters.

a. **Convert the text to a path.**

   The text, stored as a text object, needs to be converted to a path object for editing. This process is not reversible and the text will lose its memory of being text. To convert the text to a path, select the text with the *Select Tool* and use the Path → Object to Path (**Shift+Ctrl+C**) command.

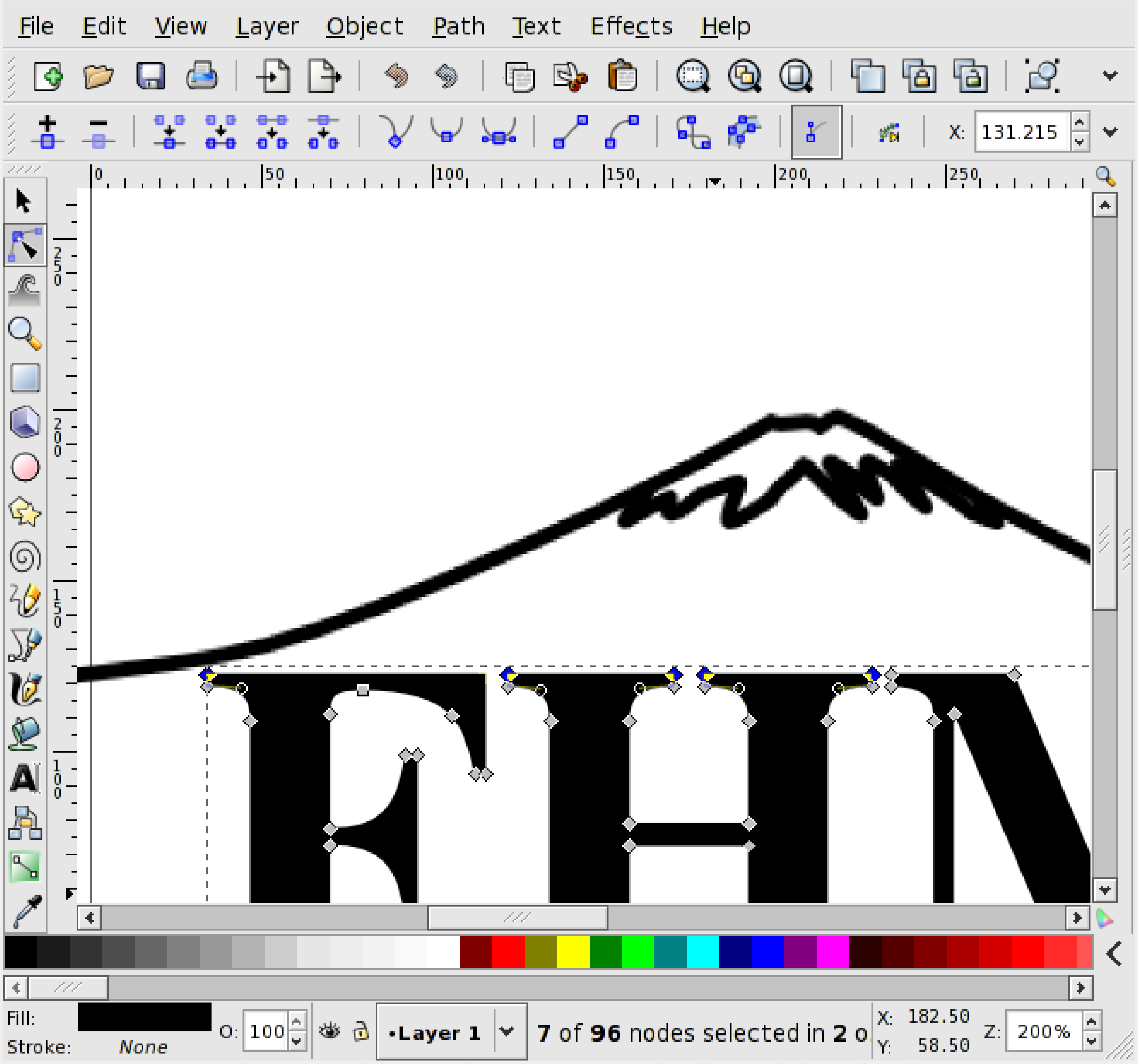
b. **Extend the text upward.**

   i. **Extend the F and H up.**

      We'll start with extending the F and H. Select the *Node Tool* (**F2**). And click on the “text” if not already selected.

      The text is now surrounded by a series of small diamonds. These are the *nodes* of the path. They define places where the path changes direction or curvature. The path is edited by manipulating these nodes.

      At this point, it is easier if you zoom in on the drawing. There are multiple ways to *zoom*. Using the **Ctrl** key with the scroll wheel of a scrolling mouse is one way. The drawing will zoom around the cursor position. Another way is to use the **+** or **=** key. In this case, the zoom is around the center of the viewable canvas. The scroll bars can be used to pan the drawing (i.e., change the viewable region).



X: 131.215

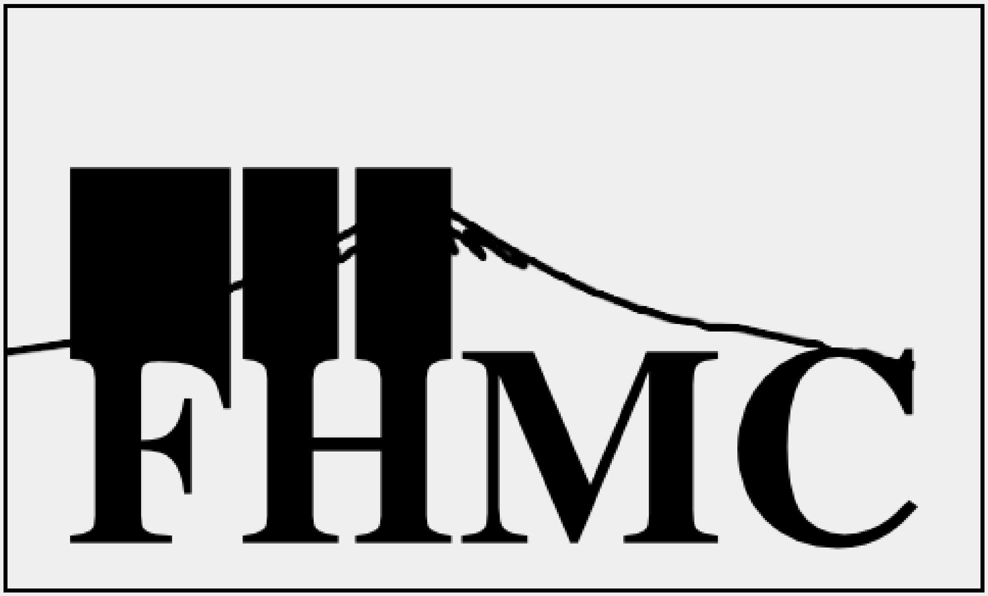
Fill: Stroke: None O: 100 •Layer 1 7 of 96 nodes selected in 2 o X: 182.50 Y: 58.50 Z: 200%

Ready to extend the F and H. You can see the diamond shaped nodes.

To select a node, click on it with the *Node Tool*. Holding the **Shift** key down allows nodes to be added (or removed) from the selection. One can also use a *rubber-band* selection technique to select multiple nodes at one time. To do this, click-drag from one point to another. All nodes within the box defined by the starting and stopping points will be selected. The click-drag must not begin on a node. The selected nodes will change from gray to blue and yellow when selected.

Select the top nodes of the F and of both sides of the H. A total of seven nodes are selected in this example (there are two on top of each other at the upper-right corner of the F). If you have chosen a different font you may have to select a different number.

Now click-drag on any of the selected nodes upward, holding the **Ctrl** key down to constrain the direction of the drag to vertical. Drag until all the selected nodes are above the outline of the mountain.



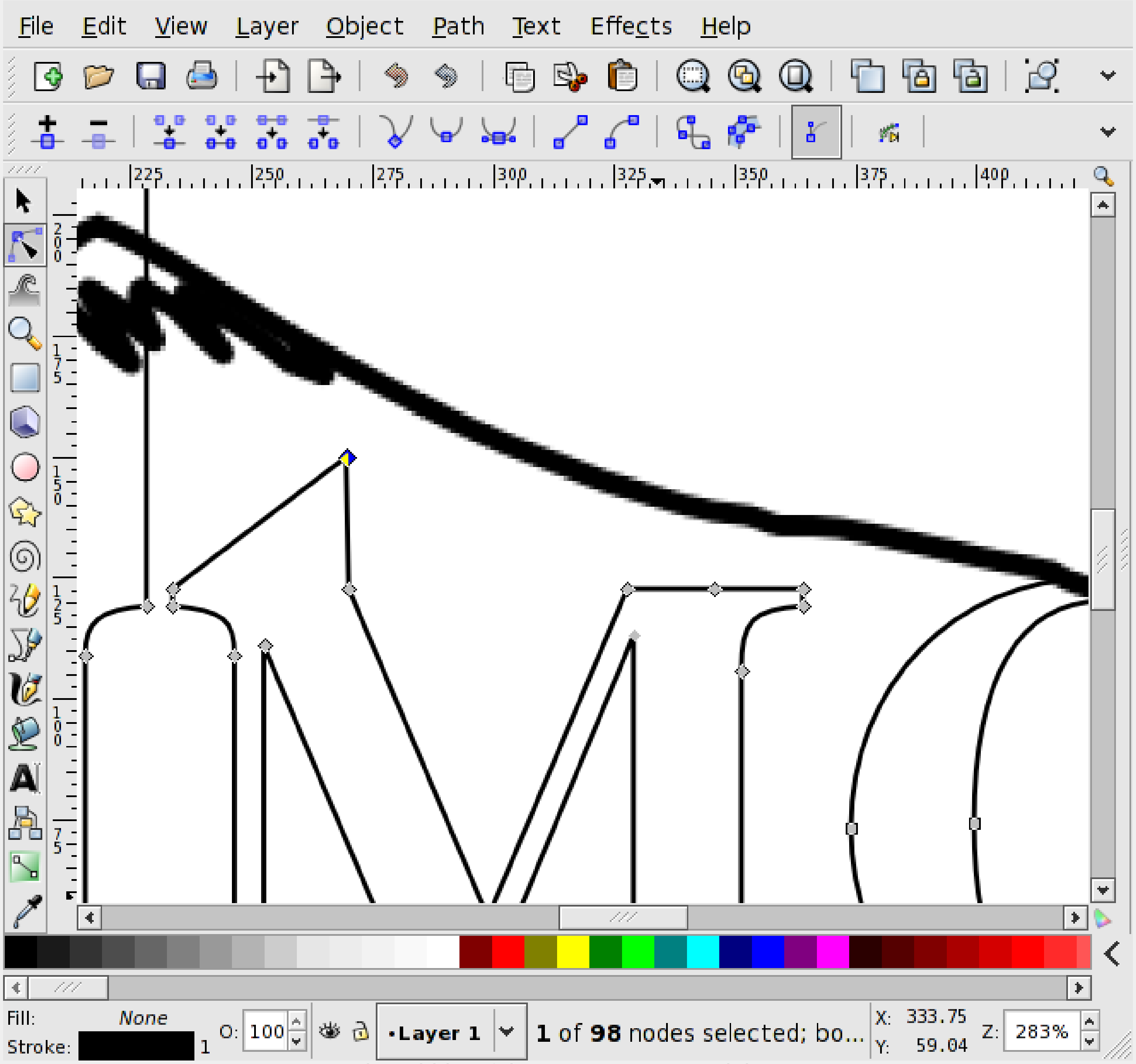
After the F and H have been extended.

One problem that is immediately apparent is that the extended parts hide part of the bitmap image. This problem can be alleviated by temporarily removing the *Fill* of the text objects. To do this, open the *Fill and Stroke* dialog (Object → ✎ Fill and Stroke... (**Shift+Ctrl+F**) or click on ✎ in the Command Bar). Select the *Fill* tab and click on the ✕ icon (*No paint*) while the text is selected. If the text is no longer visible, you will need to make the *stroke* visible. In the *Fill and Stroke* dialog, select the *Stroke paint* tab and click on the ■ icon (*Flat color*) or hold the **Shift** key down and click on one of the colors in the *Palette*.

c. **Extend the M.**

Extending the M may be more difficult than extending the first two letters, it depends on the font selected. For Nimbus Roman No9 L, we need to add two extra nodes in order to preserve the shape of the stems of the M.

Use the *Node Tool* to select the four nodes at the top of the M. Click on the + icon (*Insert New Node*) in the *Tool Controls*. This will add two nodes, each halfway between the pairs of adjacent selected nodes. Click on the background to deselect all the nodes, then click-drag the new node on the left side to be above the rightmost node of the left serif, as shown next.



Fill: None
Stroke: 1
O: 100
•Layer 1
1 of 98 nodes selected; bo...
X: 333.75
Y: 59.04
Z: 283%

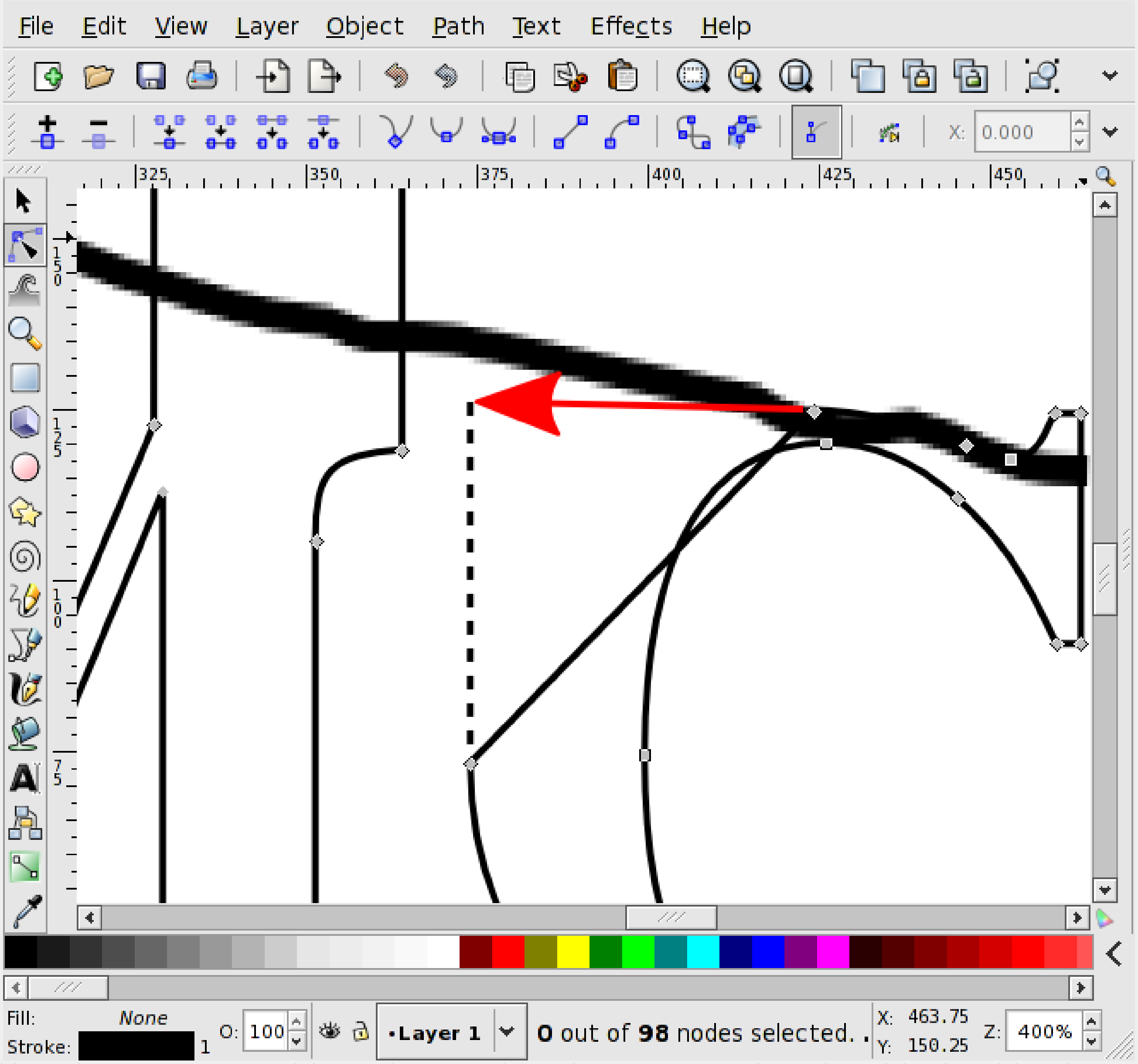
In the middle of the process to extend the M.

Do the equivalent for the new node on the right side.

Now select the four top nodes of the serifs (the two new nodes replacing two of the previous top nodes) and drag them up, holding the **Ctrl** key down to constrain their movement in the vertical direction. Move the nodes above the mountain.

- **Extend the C.**

Extending the C also requires a bit of node manipulation. Select the leftmost node and the top-center node of the C. Convert these to *Corner* nodes by clicking on the icon in the *Tool Controls*. This will allow the path to have an abrupt change in direction at the nodes. Click on the icon in the *Tool Controls* to convert the selected segment to a line. Click on the background to deselect both nodes, then click-drag the topmost node of the line to move it above the lower node of the line, as shown next.



Fill: None

Stroke: 1

O: 100

0 out of 98 nodes selected.

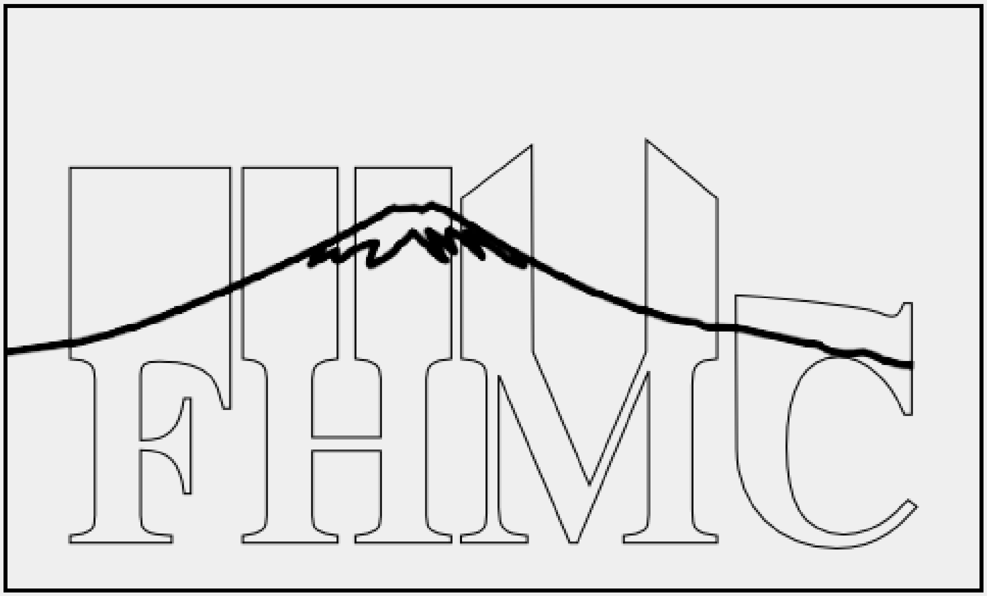
X: 463.75 Y: 150.25

Z: 400%

In the middle of the process to extend the C. The arrow shows the movement of the node and the dotted line, the new position of the line segment.

Select all the nodes along the top of the C and drag them above the mountain, again using the **Ctrl** key to constrain the movement in the vertical direction.

The text path should now look like this:



After all the characters in the text have been extended above the mountain image.

- **Adjust kerning.**

The *kerning* of the text isn't quite right. The C is too far from the M. This could have been corrected while the text was still a *Text* object, but it would have been hard to get correct spacing without seeing the newly extended parts.

To move the C, use the *Node Tool* to select all the nodes in the C, then use the left and right **Arrow** keys to move the C until the gap between the C and M matches the gaps between the F and H, and the H and M. If the movement step is too large, use **Alt**+**Arrow** to make smaller movements.

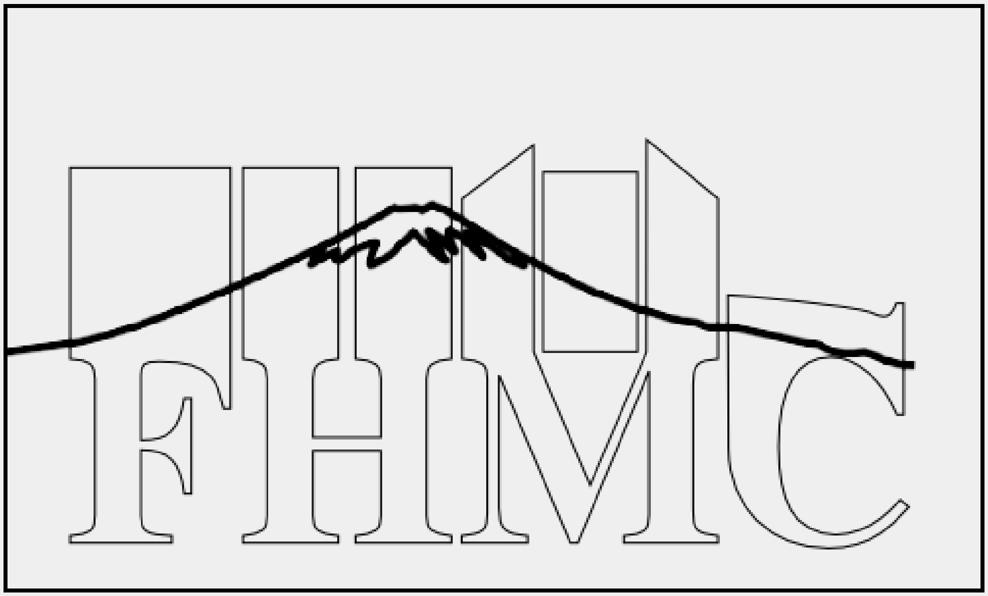
Adjust the spacing between the other letters if needed.

- **Fill in the gap in the M.**

The gap between the two extensions of the M is a bit wide. To reduce the visual effect of the gap we will add a block in between.

i. **Add the rectangle.**

   Select the *Rectangle Tool* (**F4**) and click drag a rectangle between the extensions of the M as shown below. Use the rectangle handles to adjust the position.



The text after adding a rectangle between the extensions of the M.

- **Merge the rectangle with text.**

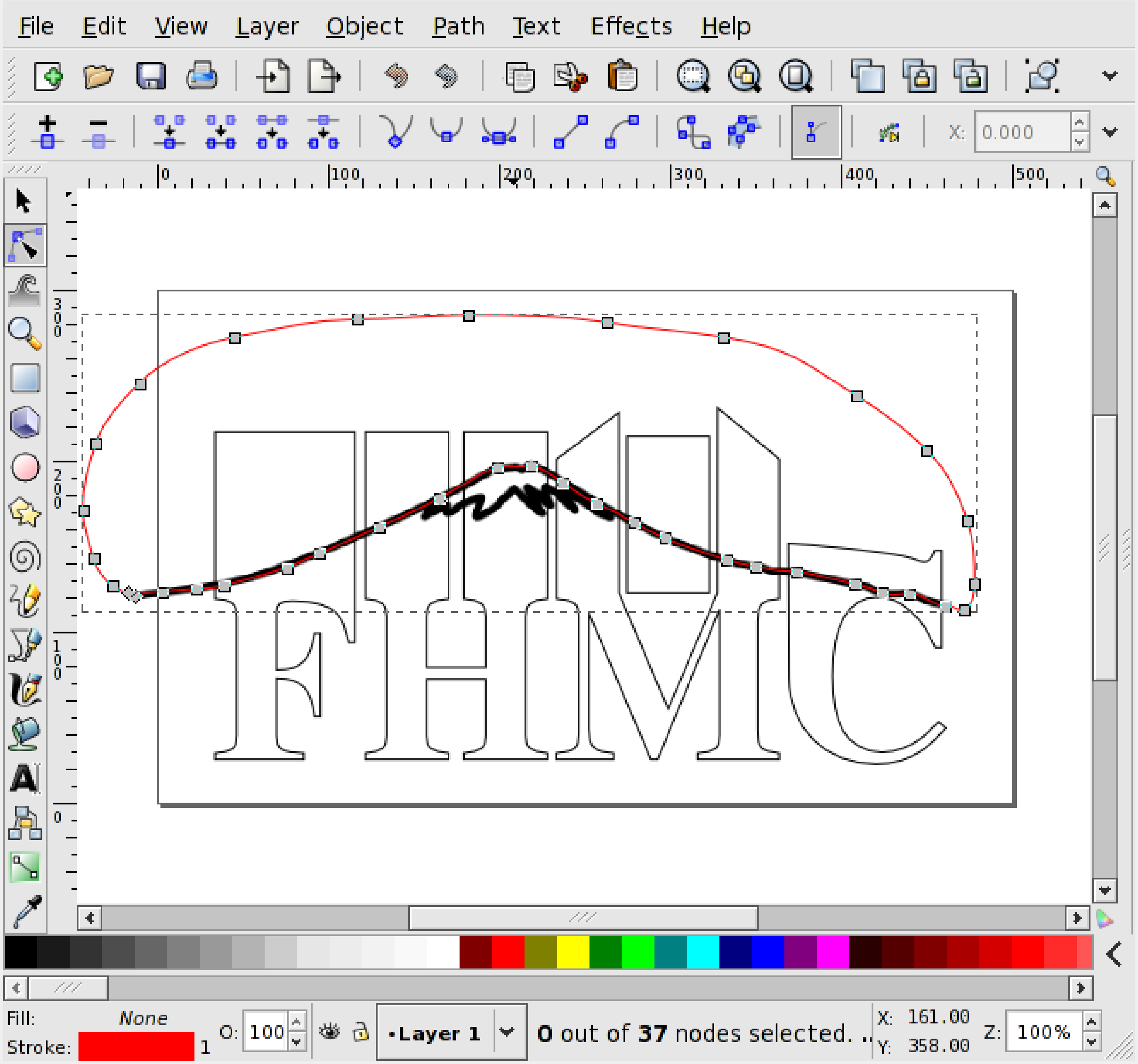
For later steps, the rectangle object must be merged with the text path. Select with the *Select Tool* both the rectangle and the text path. Then use the command: Path → Combine (**Ctrl+K**) to create one path out of the two. The rectangle object is automatically converted to a path object before the merge.

- **Trace the bitmap mountain to form a new path.**

The top of the extended text will be trimmed with the shape of Fuji San. To do this, the bitmap image of the mountain must be converted to a path. But first, make any last-minute adjustments to the position of the mountain using the *Select Tool*.

Inkscape includes a tool to trace the bitmap automatically (Chapter 16, *Tracing Bitmaps*) but the path produced is too complicated for our use.

Instead, we will use the *Freehand Tool* (**F6**). Starting at one end of the mountain, click-drag the pointer along the top edge of the mountain to create a new path. When you reach the far end, loop the path back to the starting point, as shown next.



0 out of 37 nodes selected.

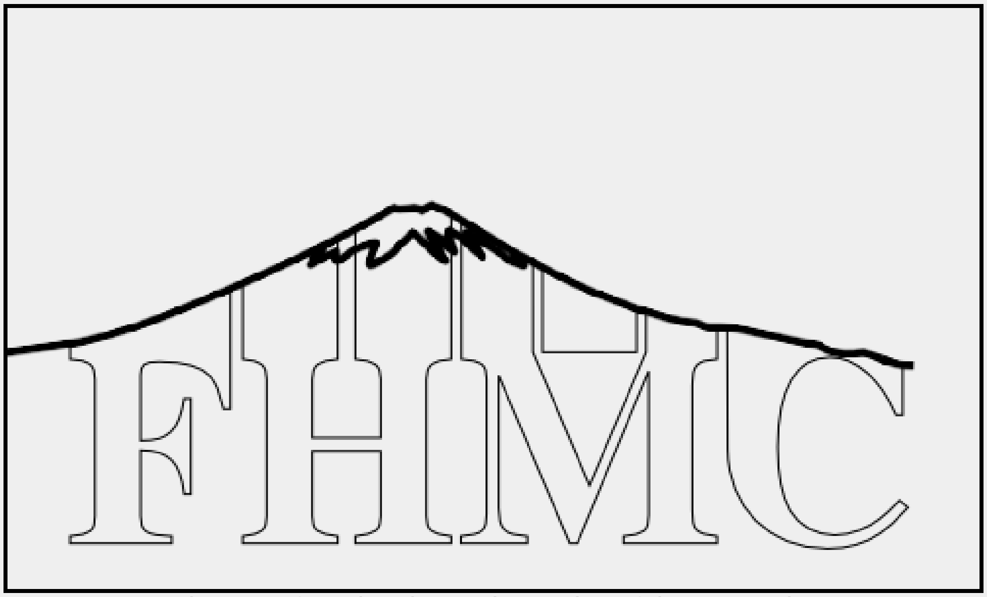
After tracing the mountain to form a path, the stroke color has been changed to red to make it easier to see.

It is not important that the ends of the path meet exactly. It is important that the loop encloses all of the tops of the letters. The path can be tweaked by using the *Node Tool* to reposition any wayward nodes. It helps to make the path a different color using the *Fill and Stroke* dialog.

- **Trim the tops of the letters.**

We'll use the *Path Difference* command to subtract the overlap between the mountain path and the text path from the text path. It is important that the mountain path be on top of the text path because this command subtracts the top path from the bottom path. As the mountain path was created after the text path, it should already be on top.

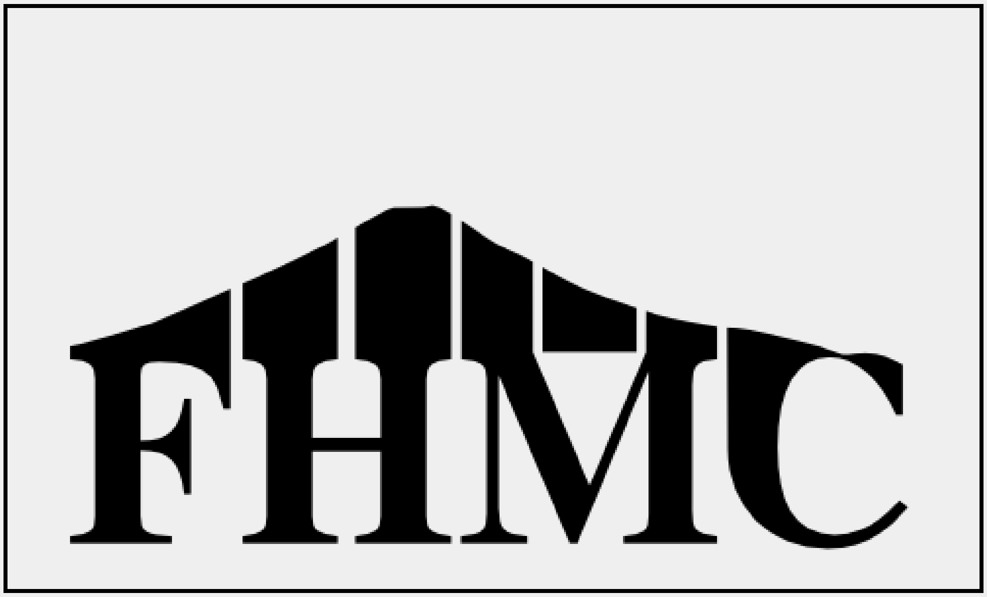
To do the path subtraction, select both the mountain path and the text path using the *Select Tool* (hold the **Shift** key down while selecting the second object) and then use the *Path Difference* command: Path → Difference (**Ctrl+-**). The mountain path will disappear and the text path should look like below.



The text after trimming to the outline of the mountain.

- **Adding the snow to the mountain top.**

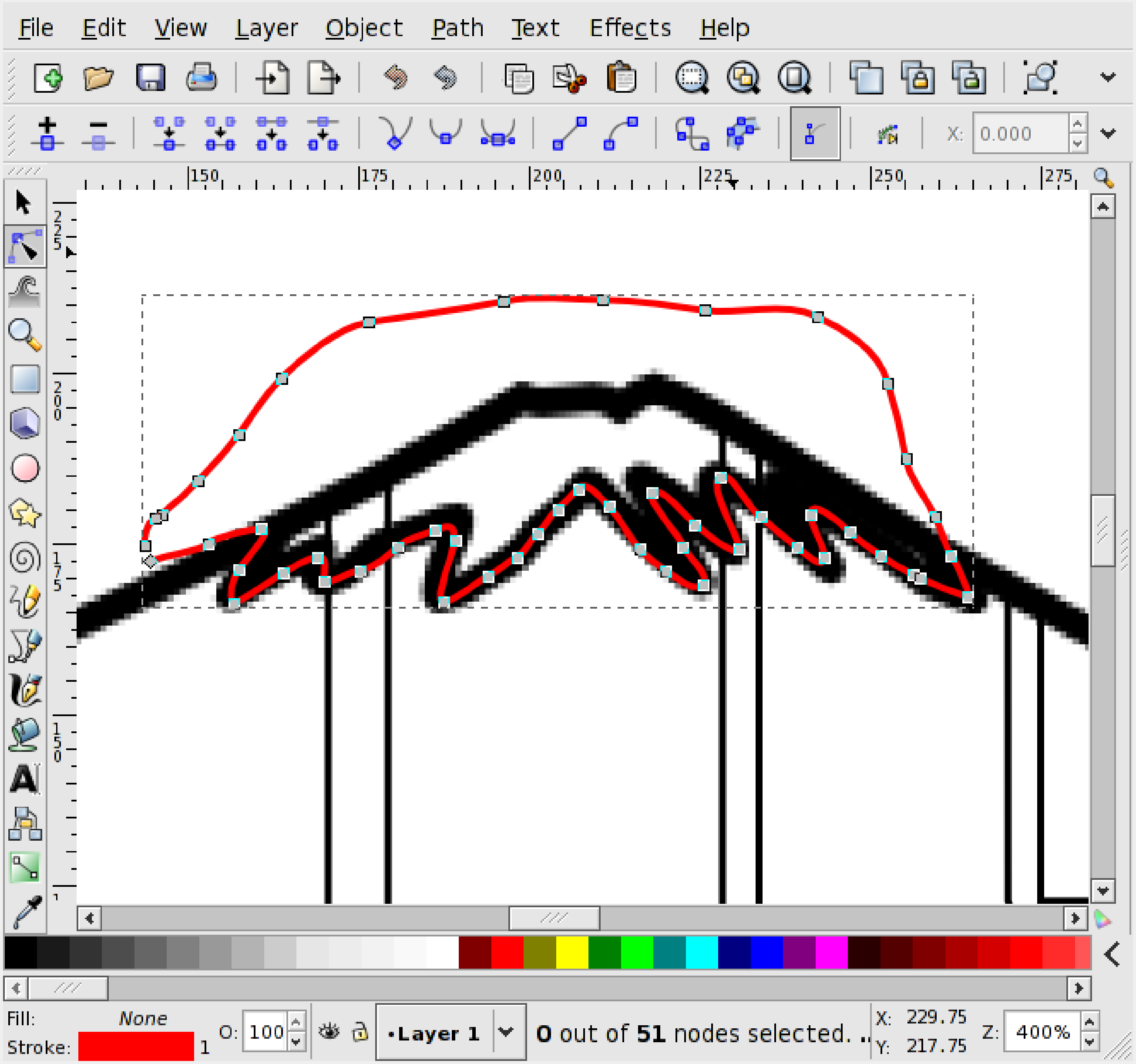
You could now delete the bitmap of Fuji San, change the fill of the text object to solid black, and call it a day, as shown next.



The logo without snow on Fuji San's top.

But it might look better if the logo included a snow cap.

To create a snow cap, zoom in on the bitmap image of the snow and the use the *Freehand Tool* to trace the snow, creating a loop as with the mountain top below.



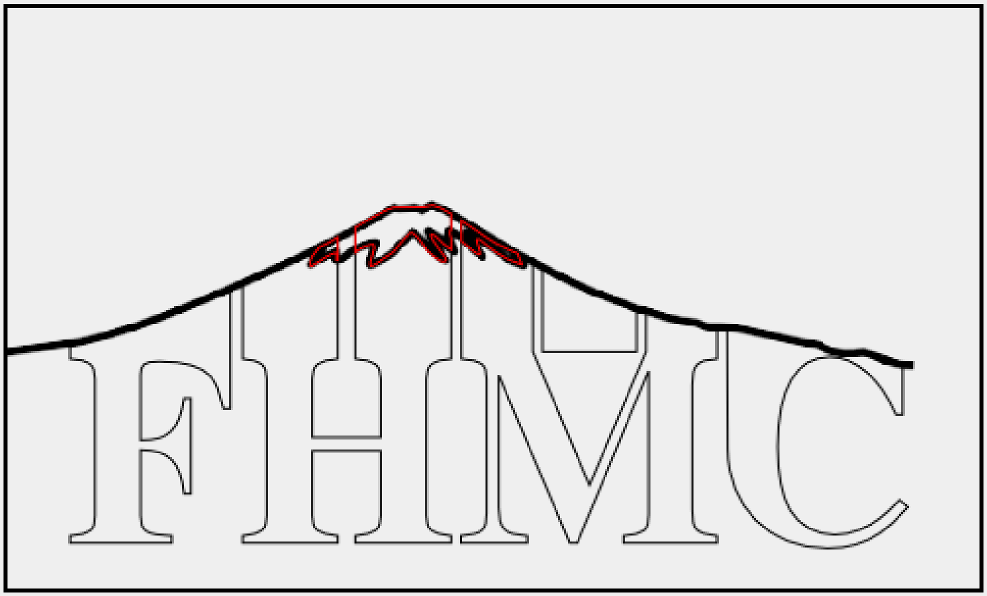
X: 0.000

Fill: None
Stroke: 1
O: 100
•Layer 1
0 out of 51 nodes selected.
X: 229.75
Y: 217.75
Z: 400%

After tracing the snow to form a path.

The next step is to create a copy of the text object, with which to cut the snow path with. To create a copy, select the text and then click on the *Duplicate* icon in the *Command Bar* or use the menu entry Edit → Duplicate (**Ctrl+D**).

With the **Shift** key down, select the snow path. Both the duplicate text path and snow path should be selected. Use the *Path Intersection* command: Path → Intersection (**Ctrl+***) to combine the two paths. The logo should look like:



The logo after adding a path for snow.

- **Finishing touches.**

  a. **Delete the Fuji San bitmap**

     The bitmap is no longer needed. Delete it by selecting it and then using Edit → Delete (**Delete**) or **Ctrl**+**X**.

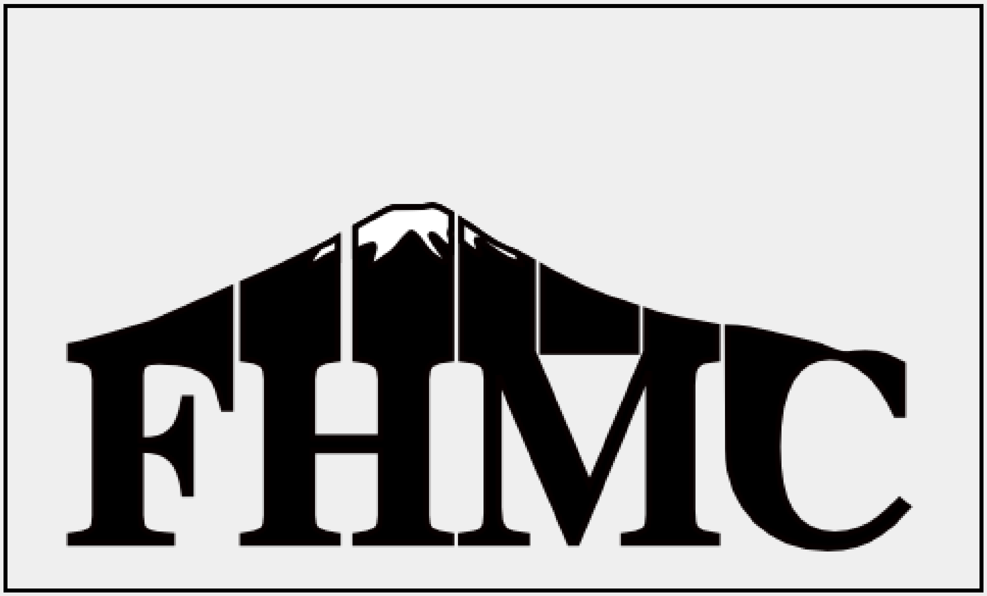
  b. **Correct *Fill* and *Stroke*.**

     Change the fill of the text object to black. Change the *Fill* of the snow object to white and the *Stroke paint* to black.

  c. **Widen the snow outline.**

     The snow outline looks a bit thin. To give it more definition, use the *Stroke style* tab of the *Fill and Stroke* dialog. Change the width to 3 pixels. The width of the text path must also be changed to 3 pixels to match.

The logo is now finished. Save your work as in the previous tutorials.



The finished logo.

The European Flag - A More Elaborate Example

Table of Contents

The Northern Pacific Railway Logo - A Tracing Example

Inkscape: Guide to a Vector Drawing Program

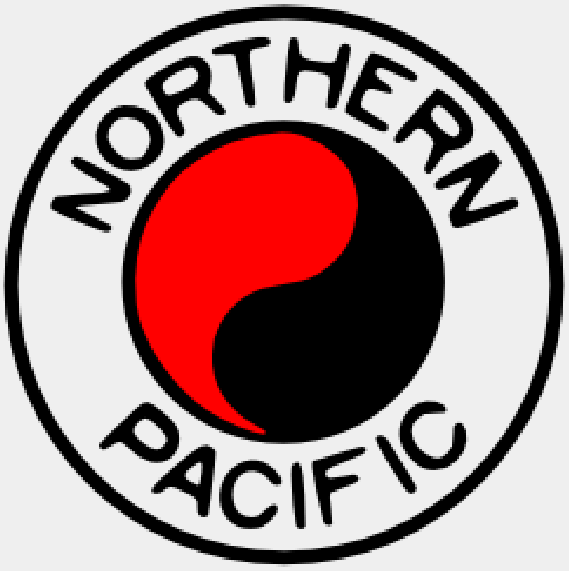
Get the book.

Inkscape » Quick Start » The Northern Pacific Railway Logo - A Tracing Example

Index

The Northern Pacific Railway Logo - A Tracing Example

Inkscape's *auto-tracing* capability is very useful for turning existing artwork into *SVG* drawings. In this example, we will create the artwork for a logo from a photograph. The logo is for the Northern Pacific (or NP) railroad, which features the *Yin and Yang* symbol. This tutorial will cover use of the *Trace Bitmap* dialog as well as manipulation of paths. The use of *Layers* is also introduced.



Logo for the Northern Pacific Railroad featuring the Yin and Yang (Monad) symbol.

The steps are:

- Start Inkscape and setting the drawing size.
- Import the source photo.
- Auto trace the logo.
- Clean up the logo paths.

Procedure 1.4. Creating the Northern Pacific Logo

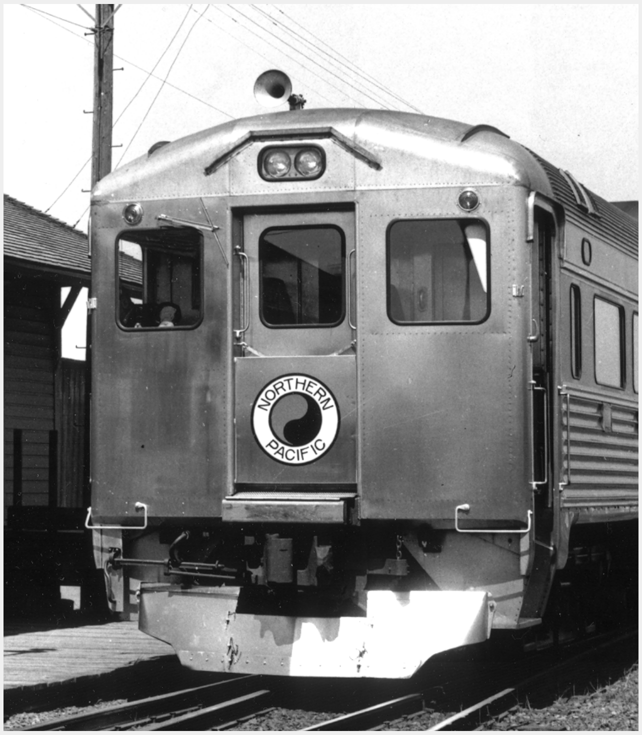
1. **Set up the drawing.**

   To begin, start Inkscape.

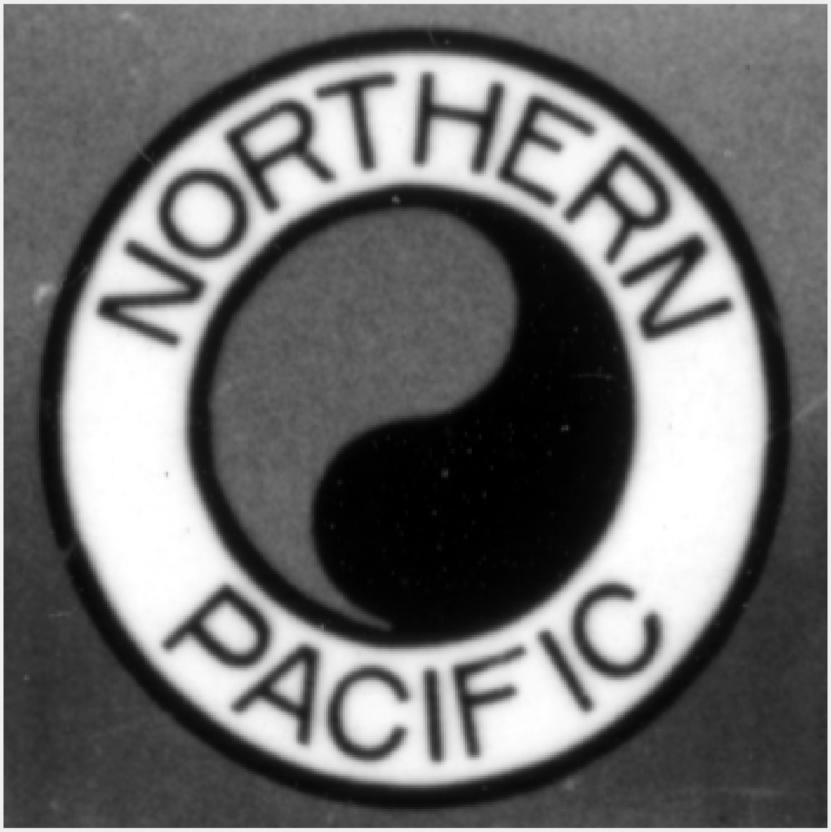
   Follow the instructions for setting the page size and grid spacing given in the Swedish flag example but set drawing size to a width of 500 and a height of 500 pixels. Do *not* turn on the *Grid*.

2. **Import the photograph.**

   You can use any photograph (in *PNG*, *GIF*, or *JPEG* form) with a logo for this exercise. However, sharp, high resolution photographs work best. I will use part of a photograph from the railroad photographer James M. Fredrickson. The photograph has been cropped to show only the logo. If you wish to use the same photograph as used in the exercise, you can download it from the book's website: http://tavmjong.free.fr/INKSCAPE/.



The end of a passenger car showing the Northern Pacific logo.



The NP logo, cropped and zoomed from the previous photograph.

a. **Import the photograph.**

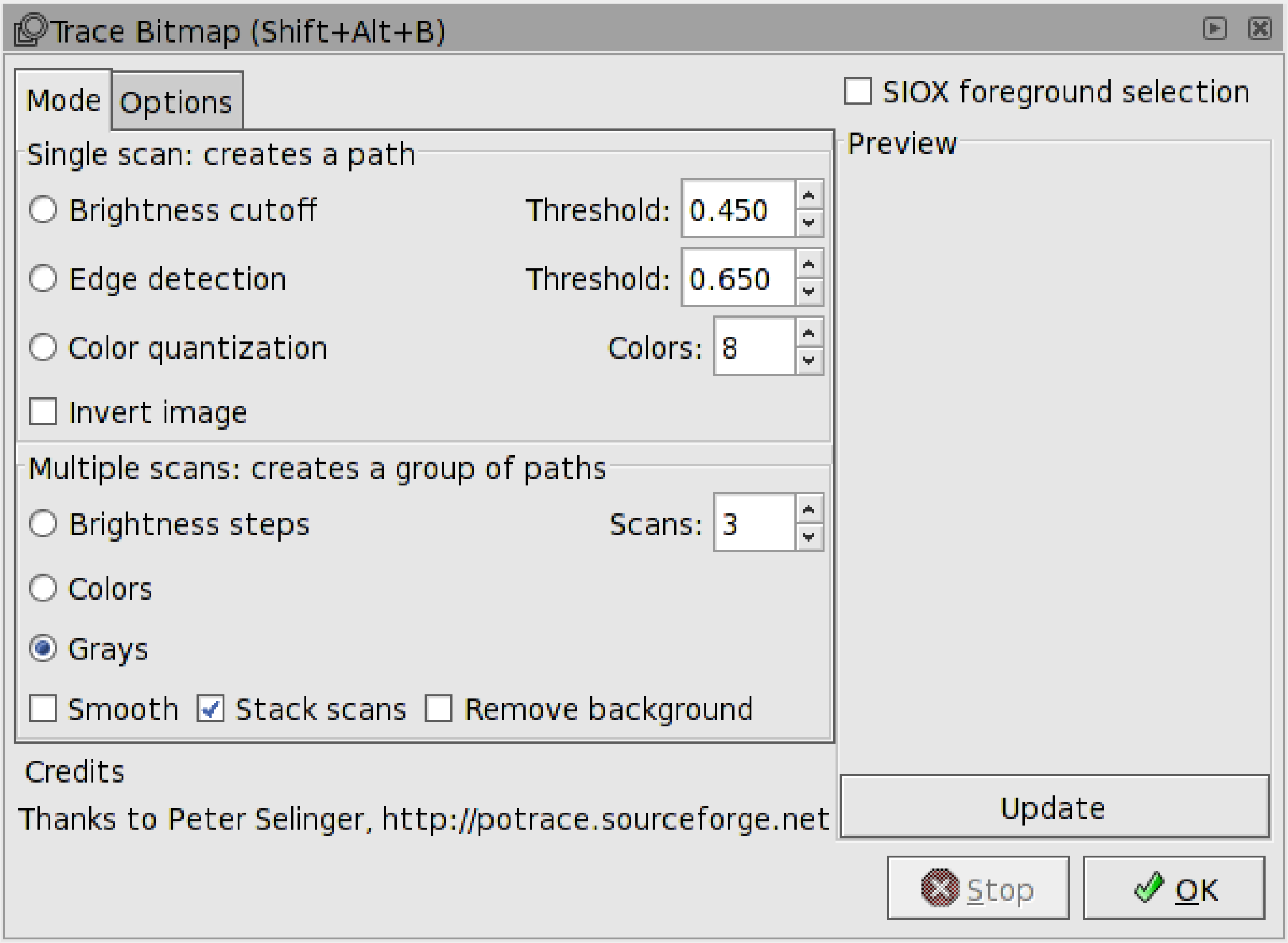
   Import the photograph using the File → Import... (**Ctrl+I**) dialog.

b. **Adjust the size and position of the photograph.**

   Adjust the photograph's size and position to match the Inkscape page with the *Select Tool* (keyboard shortcut **F1**). Drag the corners while holding down the **Ctrl** key to preserve the aspect ratio. Drag the body to translate the photo.

3. **Trace the logo.**

   Inkscape utilizes the *potrace* tracing library to create *SVG* paths from bitmap images. To trace the picture, call up the *Trace Bitmap* dialog (Path → Trace Bitmap... (**Shift+Alt+B**)).



Single scan: creates a path

○ Brightness cutoff Threshold: 0.450

○ Edge detection Threshold: 0.650

○ Color quantization Colors: 8

□ Invert image

Multiple scans: creates a group of paths

○ Brightness steps Scans: 3

○ Colors

● Grays

□ Smooth ☑ Stack scans □ Remove background

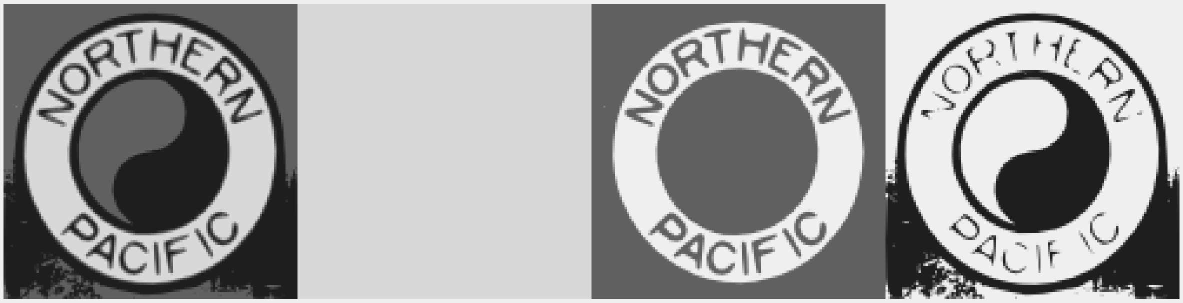
□ SIOX foreground selection

Credits

Thanks to Peter Selinger, http://potrace.sourceforge.net

Trace Bitmap dialog.

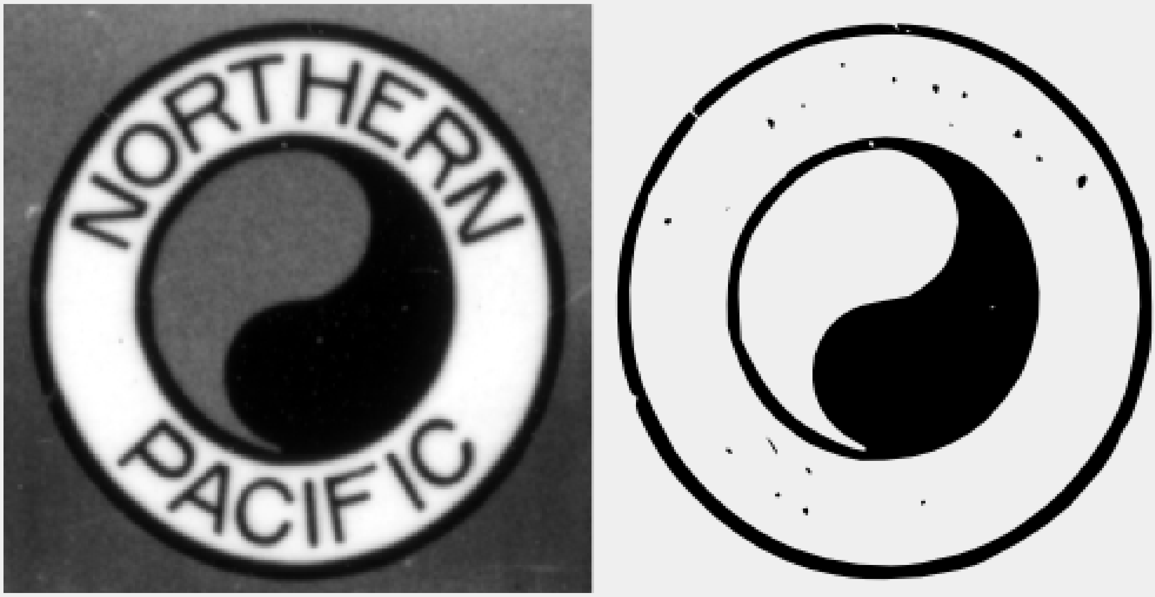
The *Trace Bitmap* dialog presents a number of choices for how the tracing is done. Looking at the image, we see that there are three different grayscale levels. This suggests that we use one of the *Multiple scans* methods. As this is a monochrome photograph, we'll try the *Grays* option with *Scans* set to three. This will give us three regions, each corresponding to one of the grayscale levels. There is the possibility to smooth the bitmap prior to tracing by checking the *Smooth* box. This is not necessary as our starting photograph doesn't have any speckle. Finally, there is an option to generate paths for mutually exclusive areas or that include all darker regions. The latter is more useful for us, so we check the *Stack scans* option box. The resulting traces are shown below.



Results of scanning using the *Multiple Scanning - Grays* option. On the far left is the output of the scan. The three paths that make up the scan are shown separately to the right. As each path includes all darker regions, the path for the lightest region is just a square covering the entire page.

The paths generated with the *Grays* option are usable as a starting point for the NP logo. However, we might be able to do better if we use the *Single scan - Brightness cutoff* method, optimizing the threshold for each scan.

Select the *Image Brightness* option in the *Trace Bitmap* dialog. Adjust the *Threshold* to create a well-defined Yin and Yang path. Pressing the *Update* button will create a preview. A threshold of 0.15 seems to work well. After finding a good value, click on the *OK* button to generate the path.



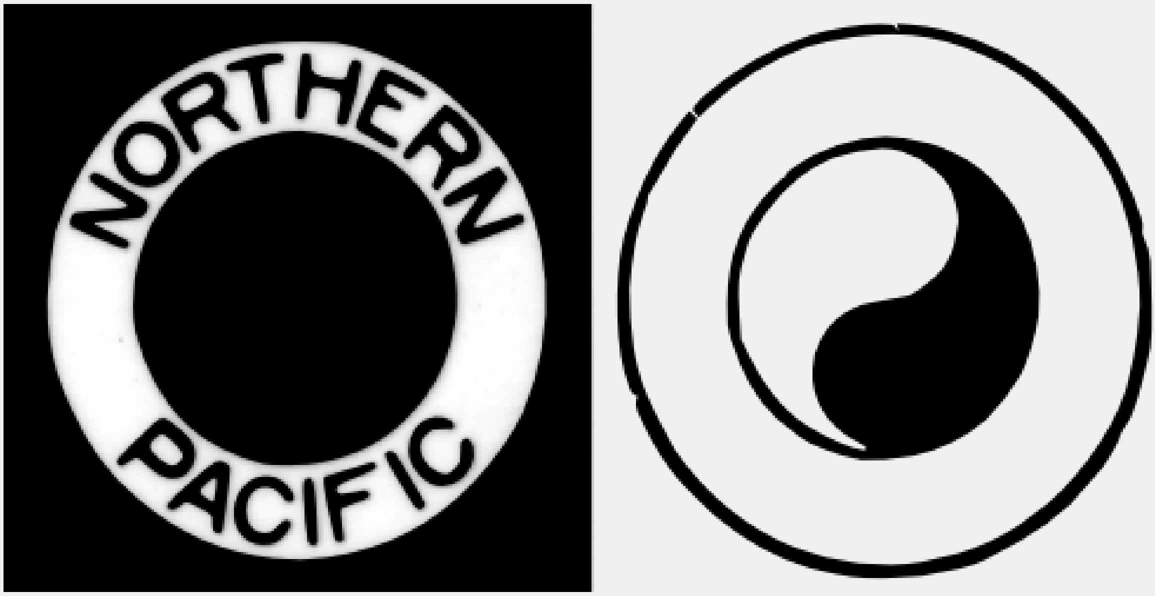
Scanning the photo with a *Threshold* of 0.15. The scan has been shifted for better visibility. Notice the “blobs” in between the two circles.

Updated for v0.45.

The scan has many “blobs” that clutter the picture. We could remove these by hand, but there is an easier way. Select the *Options* tab in the *Trace Bitmap* dialog. The *Suppress speckles* option allows the automatic suppression of paths smaller than the specified size. Make sure the *Suppress speckles* box is checked and set the *Size* to 50. While your at it, make sure the *Smooth corners* and *Optimize paths* boxes are checked. This will result in a smooth scan with fewer nodes. Rescan and the “blobs” should be gone.

It will be easier to edit the traces separately. To do so, we'll temporarily move the path. This requires a precise move which can be done with the *Transform* dialog (Object → Transform... (**Shift+Ctrl+M**)). Bring up the dialog and select the *Move* tab. Move the path a page width to the right by setting the *Horizontal* entry to 500 pixels, setting the *Vertical* entry to 0 pixels, selecting the *Relative move* option, and then clicking on the *Apply* button. (One could also move the path using the **Shift**+**Arrow** keys to move the path 20 pixels at a time. This may be faster than calling up the *Transform* dialog but there is a small risk of making a mistake in the shifts.)

Next, repeat the trace with a new threshold. This time optimize the threshold for the lettering. A threshold of 0.65 looks good. Execute the trace. The following figures shows what you should see.



Results of scanning using the *Image Brightness* twice. A threshold of 0.15 was used for the path in the right while a threshold of 0.65 was used for the path on the left. The original photo is still under the tracing on the left.

4. **Clean up the traces.**

   a. **Move the photograph to a different layer.**

      Objects in Inkscape can be selected by clicking on them. This leads to a small problem when using a *rubber-band* selection technique where one click-drags to select multiple objects as we will want to do a few times. If the click-drag begins over the photograph, the photograph will be moved. One can avoid this by holding the **Shift** key down while starting the click-drag, but this is a bit of a pain to do every time. One could *lock* the picture in place by checking the *Lock* box in the *Object Properties* dialog but this isn't recommended. It is non-trivial to select and unlock a locked object.

      The preferred solution is to put the picture on a separate *Layer*. *Layers* can be thought of as separate drawings on transparent sheets that are stacked on top of each other. The final drawing is what one sees through the stack. In Inkscape, the *Layers* can be easily locked and unlocked against modification. They can also be made invisible. We'll put the photograph on a locked *Layer* by itself. Then there is no chance of it moving while we work on cleaning up the tracings.

      First, create a new *Layer* for the photograph. This is done through the *Add Layer* dialog (Layer → Add Layer...). Enter the name “Photograph” and hit the **Enter** key or click on the *Add* button. The new layer is created above the old layer. It is selected automatically. The layer name is shown in the *Status Bar*.

      Select the photograph by clicking on an opening in the tracing (check the *Notification Region* to see what is selected). Note that the layer is automatically changed back to the original layer

("Layer 1") when the photograph is selected. Move the photograph to the new layer by using the command Layer → Move Selection to Layer Above (**Shift+Page Up**). The tracing is no longer visible. This is because the photograph has been moved to a layer that is above the layer with the tracing.

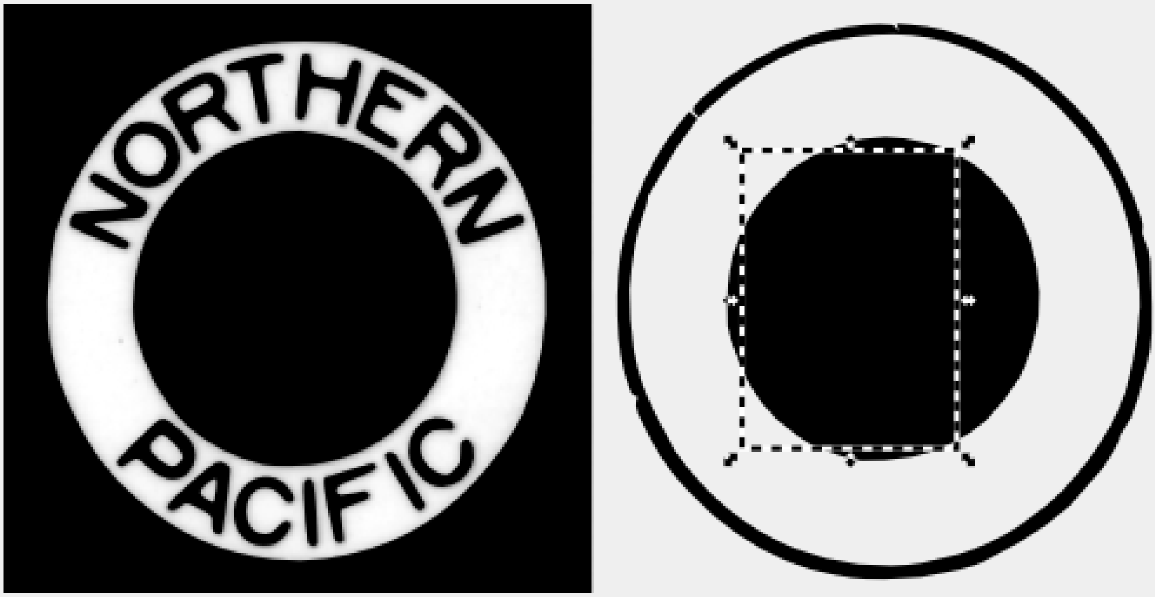
Select the “Photograph” layer through the pull-down menu that pops up when you click on the layer name in the *Status Bar*. Lock this layer by clicking on the icon next to the layer name. The icon should change to the locked state . The photograph can now not be moved.

Finally, move the “Photograph” layer beneath the original layer by using the command Layer → Lower Layer (**Shift+Ctrl+Page Down**).

b. **Clean up the Yin and Yang.**

Select the path for the circles and the Yin and Yang. The original layer will be selected. Zoom in the paths by hitting the **3** key. Each trace creates one path. Each path is made up of many sub-paths. It is generally easier to clean up a tracing by converting the sub-paths into independent paths. Do this by using the Path → Break Apart (**Shift+Ctrl+K**) command.

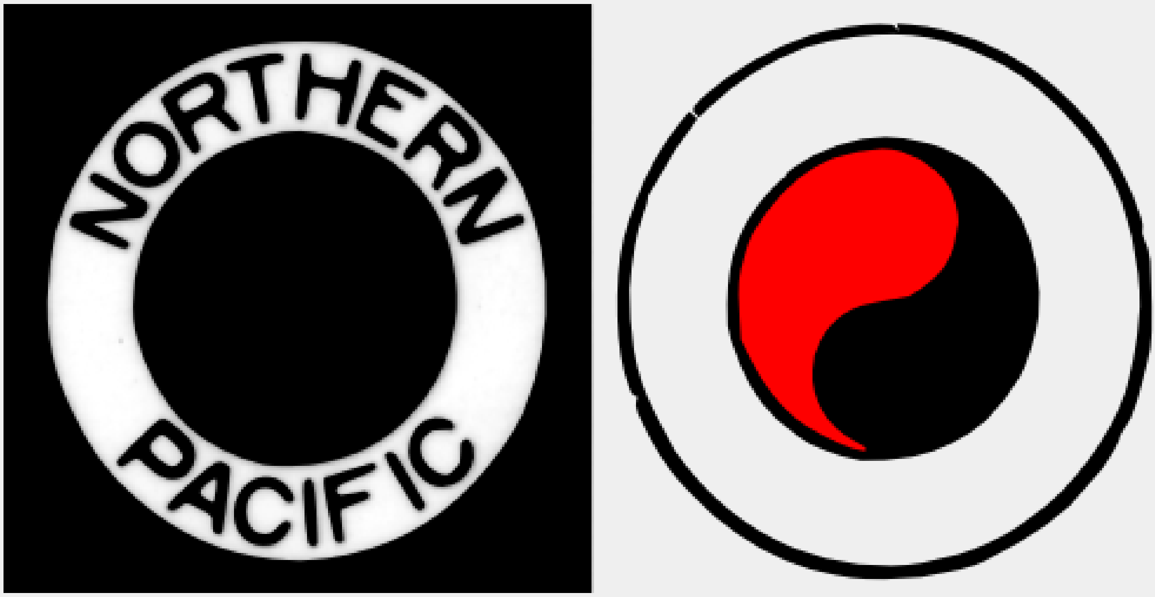
The first thing you will notice is that the Yin and Yang symbol becomes all black. Don't panic! This is because the Yin and Yang has been broken into two pieces, one for the outer circle and one for the Yin shape. Each path takes on the attributes of the prior combined path and thus both have black *Fill*. Select the Yang (left side) path by clicking inside the area of the Yang path (look at the original photo). You can tell you've selected the correct path by the shape of the bounding box that is shown when the path is selected. If you got the wrong path, try again moving the cursor a bit. It may be necessary to hold down the **Alt** key while clicking. This will cycle through selecting each object under the cursor in turn. (If the cycling doesn't work, your window manager may be stealing the **Alt** key. See the section called “Alternative Alt Key” in Chapter 20, *Customization*, for ways to work around this problem.)



The Yang path is selected.

While the photograph is in black and white, the Yang part of the monad in the NP logo should be red. Set the *Fill* color of the path red using the *Fill and Stroke* dialog (Object → Fill and

Stroke... (**Shift**+**Ctrl**+**F**)).



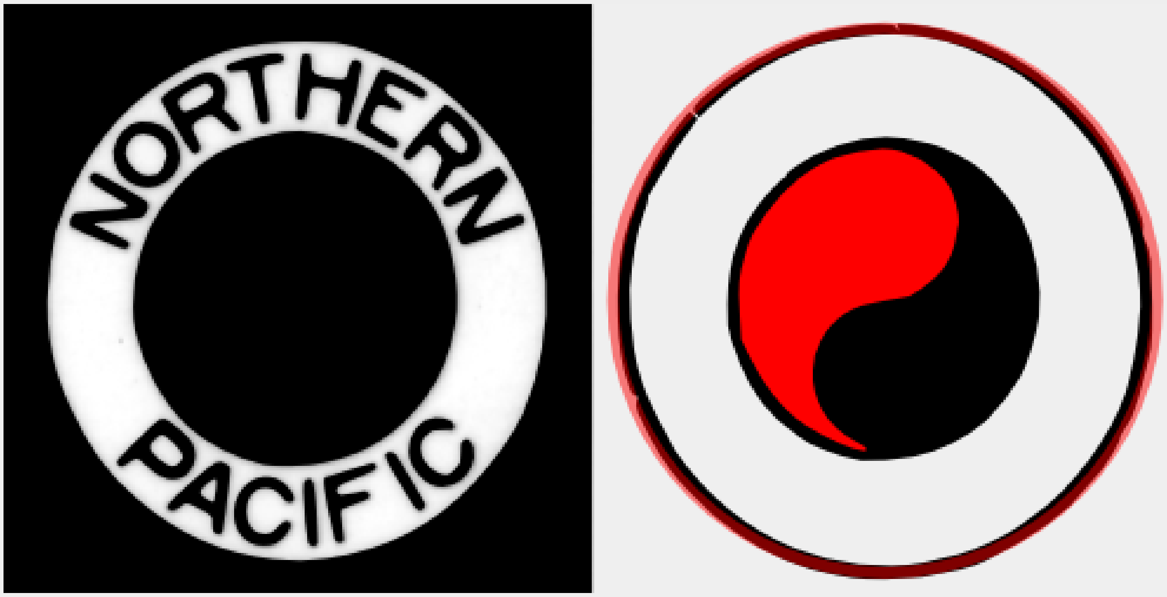
After the Yin and Yang has been cleaned up.

5. **Clean up circles.**

   Next we will replace the complicated path for the outer circle with... you guessed it, a circle. Select the *Ellipse Tool* from the *Tool Controls*. To create a circle, hold down the **Ctrl** key while click-dragging the mouse from one corner of a imaginary box (the *bounding box*) that encloses the outer circle, to the opposite corner of the imaginary box. Don't worry about getting the size and position correct at this point.

   Using the *Fill and Stroke* dialog, turn off the *Fill* and turn on the *Stroke*. It is easier to adjust the circle if the circle is semi-transparent and of a different color than the original circle so set the *Master Opacity* to 50% and the *Stroke color* to red.

   Now we are ready to adjust the circle. Set the stroke width to match the width of the original circle (*Width* under the *Stroke style* tab). Next, adjust the size of the circle by dragging either of the two square handles (on left and top of circle). The handles are visible (by default) when any of the *shape tools* or the *Node Tool* is selected. Hold down the **Ctrl** down while dragging to preserve the circle shape. Tapping the **Space** bar will change to the *Select Tool* for repositioning the circle. Fine adjustments to the position can be made by holding the **Alt** key down while using the arrow keys. Tapping the **Space** bar a second time will reselect the *Ellipse Tool*.

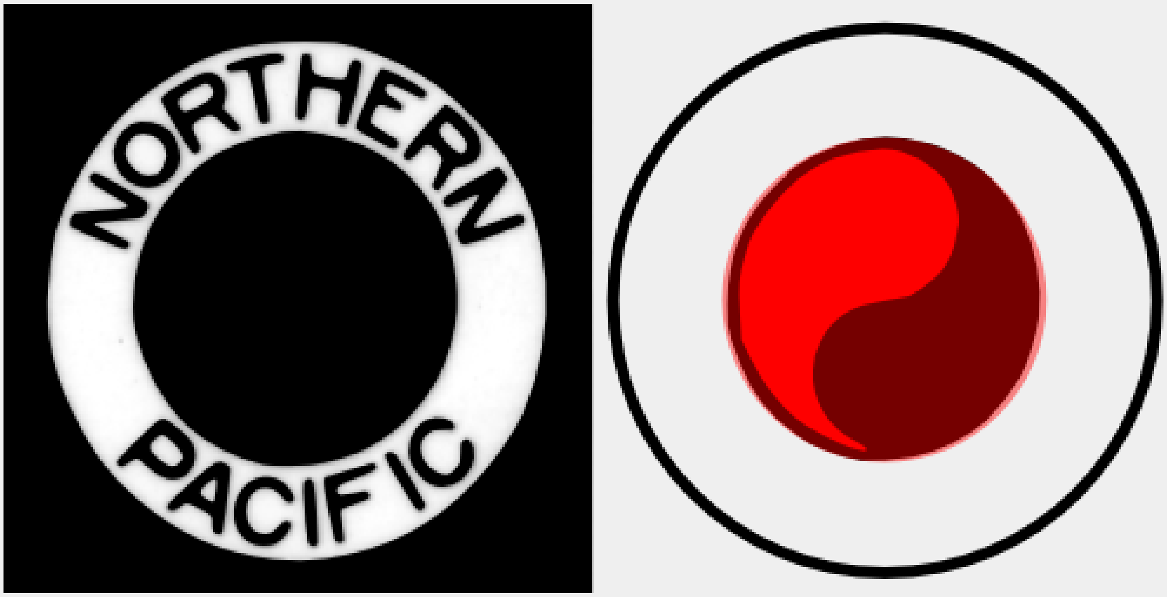


The new outer circle (semi-transparent red), sized and positioned.

By this time, you will have realized that the original outer circle is not a true circle. This is because the logo in the photograph is at an angle with respect to the plane of the photograph so the original circle is distorted. We will keep the true circle for the logo but keep in mind that we may need to make other corrections.

The final steps for the outer circle are to delete the original circle and change the stroke of the new circle to a solid black.

The inner solid black circle can also be replaced following the same steps as for replacing the outer circle with two minor changes: 1. Use a filled circle without a stroke. 2. Place the new circle behind the Yang object with the Object → Lower (**Page Down**) command (also available from the *Select Tool*-*Tool Controls*).

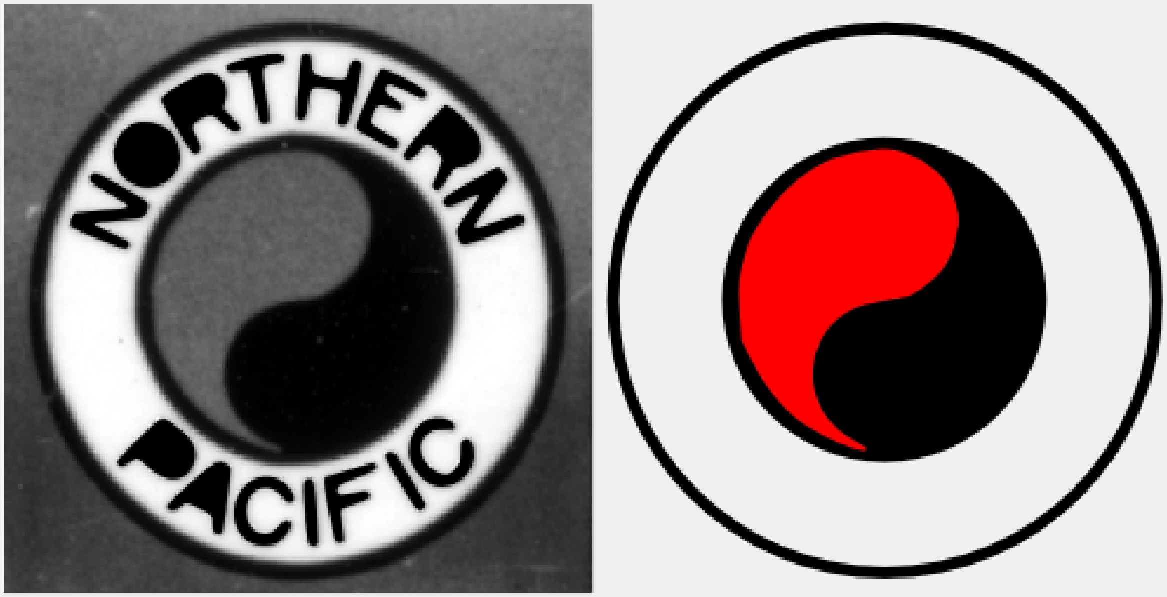


The new inner circle (semi-transparent red), sized and positioned but before moving behind the Yang object.

6. **Clean up the lettering.**

   It is now time to clean up the letters. Again we must break up the path into pieces. After doing so, you'll see one big black square. I don't have to remind you not to panic, do I? Select the square and delete it; you should now see a big black circle. Delete that too (select it by clicking its edge). You should now see letters. While we are at it, delete the center black circle leaving only the letters.

   Notice that the A, O, and R's are missing their holes. (You are not panicking, right?) This is because they are each composed of two paths that need to be put back together. We can do this for each of the letters, but in this case it is more convenient to select all the letter paths and merge them into one path using the Path → Combine (**Ctrl+K**) command.

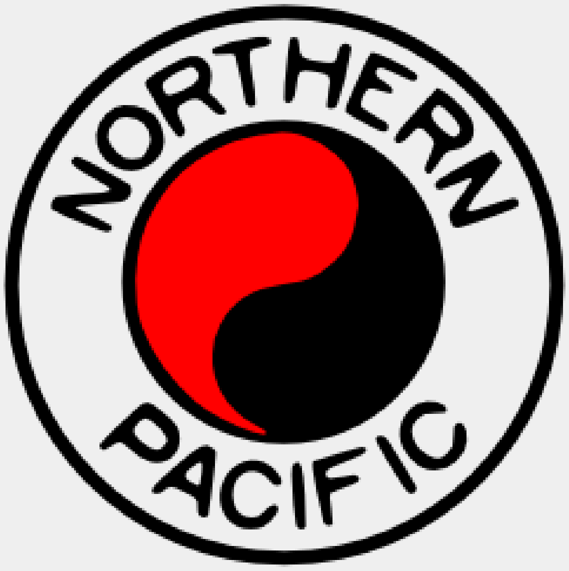
   After extracting only the lettering on the left.

7. **Final steps.**

   It is time to rejoin the Yin and Yang as well as the circles with the text. Select the Yin and Yang, and the circles and use the *Transform* dialog to move them -500 pixels in the horizontal direction.

   One can now see that the lettering isn't quite spaced correctly. This is again due to the orientation of the logo in the original picture. The lettering needs to be widened a little bit. Select the lettering path and scale it by 3% in the horizontal direction using the *Scale* tab of the *Transform* dialog.

The logo is now finished. You can either delete or hide the original photograph. You can also make fine adjustments to the Yin path with the *Node Tool* if desired. Save your work as in the previous tutorials.



The finished Northern Pacific logo.

A Hiking Club Logo - An Exercise in Paths

Table of Contents

A Box for Cards - An Isometric Projection

Get the book.

Inkscape: Guide to a Vector Drawing Program

Get the book.

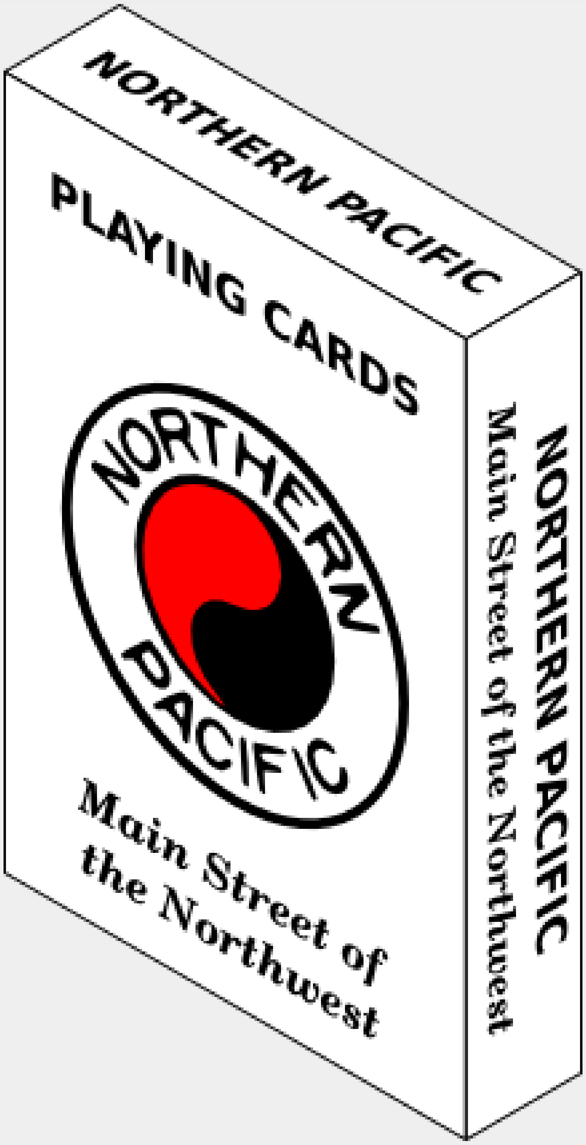
Inkscape » Quick Start » A Box for Cards - An Isometric Projection

Index

A Box for Cards - An Isometric Projection

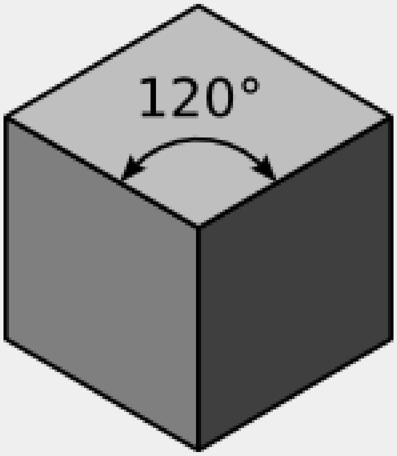
Updated for v0.46.

Inkscape is a two-dimension drawing program. It can, however, be used to create simple three-dimension drawings. We will create in this tutorial a simple *Isometric Projection*.



A box of playing cards.

An isometric projection is a view of an object such that there is an equal opening angle between the three projected orthogonal axes, as shown below.



The isometric projection of a cube. The angle between each pair of projected axes is 120°.

Inkscape as of v0.46 includes axonometric *Grids* that can be used to rapidly draw isometrically projected boxes. However, the method described here works best when drawings are included on the sides of the boxes as distorting the sides requires two precise transformations (scaling and skewing).

The steps are:

- Start Inkscape and set the drawing size.
- Create the three sides of the box, unfolded.
- Add designs to box sides.
- Transform each box side into place.

Procedure 1.5. Creating an Isometric Projection

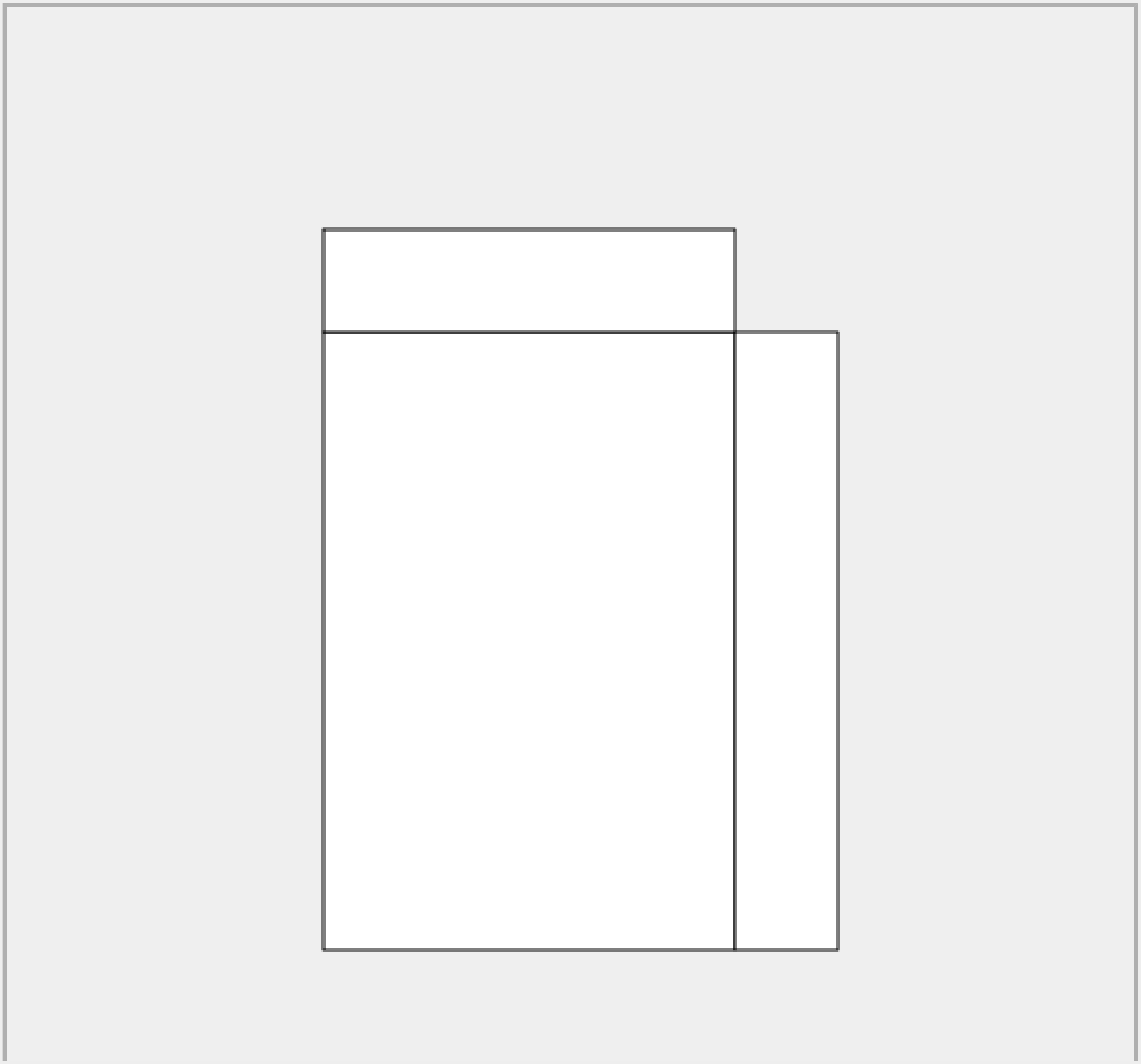
1. **Set up the drawing.**

   To begin, start Inkscape.

   Follow the instructions for setting the page size and defining a *Grid* given in the Swedish flag example but set drawing size to a width of 500 and a height of 500 pixels. Set the *Grid* spacing to 10 pixels. Turn on snapping of nodes to the *Grid* (and turn off snapping of *bounding boxes*). Don't forget to turn the *Grid* on.
2. **Create the three sides of the box.**

We need the front, side, and top of the box in the ratio 6:4:1, similar to the dimensions for a deck of cards. Create one rectangle with a width of 200 pixels and height of 300 pixels with the lower edge 100 pixels from the bottom of the page. Create a rectangle to the left of the first with width of 50 pixels and height of 300. Create another rectangle above the first rectangle with width of 200 pixels and a height of 50 pixels. The paths of the rectangles should overlap.

Turn on the *Fill* of the rectangles with the *Fill and Stroke* dialog (Object → Fill and Stroke... (**Shift**+**Ctrl**+**F**)) and set the *Fill* to white. Turn on the *Stroke* and set the *Stroke* color to black and the width to one pixel. Set the *Stroke Join* to *Round* by clicking on the icon under the *Stroke style* tab. This will make the corners where the rectangles touch neater.



The sides of the box laid out flat.

3. **Add designs to the box sides.**

   You can add any designs you want to the box sides. I will use the NP logo created in the previous tutorial as well as some text.

   Import the logo using the File → Import... (**Ctrl+I**) command. The logo will need to be scaled smaller. You can do this by click-dragging on one of its corners. Hold the **Ctrl** key down to keep the height to width ratio constant. Once you have it sized appropriately, you can center the logo within the large rectangle by using the Object → Align and Distribute... (**Shift+Ctrl+A**) dialog. To add the large rectangle to the logo selection, click on it while holding down the **Shift** key. In the *Align and Distribute* dialog, select from the *Relative to* drop-down menu *Last selected*. Since the rectangle was the last object selected, it will remain fixed and the logo will move. Click on the *Center on vertical axis* icon ( ) and the *Center on horizontal axis* icon ( ).

   Add some text to the sides as you wish. When finished, group each box side with the objects on that side by first selecting the box side with all the objects inside and the using the Object → Group (**Ctrl+G**) command. When objects are placed in a *Group*, they can be manipulated as if they are just one object. This will ensure that when the box sides are transformed, the logo and text will be transformed as well.

NORTHERN PACIFIC
PLAYING CARDS
NORTHERN PACIFIC
Main Street of the Northwest
NORTHERN PACIFIC
Main Street of the Northwest

The sides after adding logo and text.

4. **Transform each box side into place.**

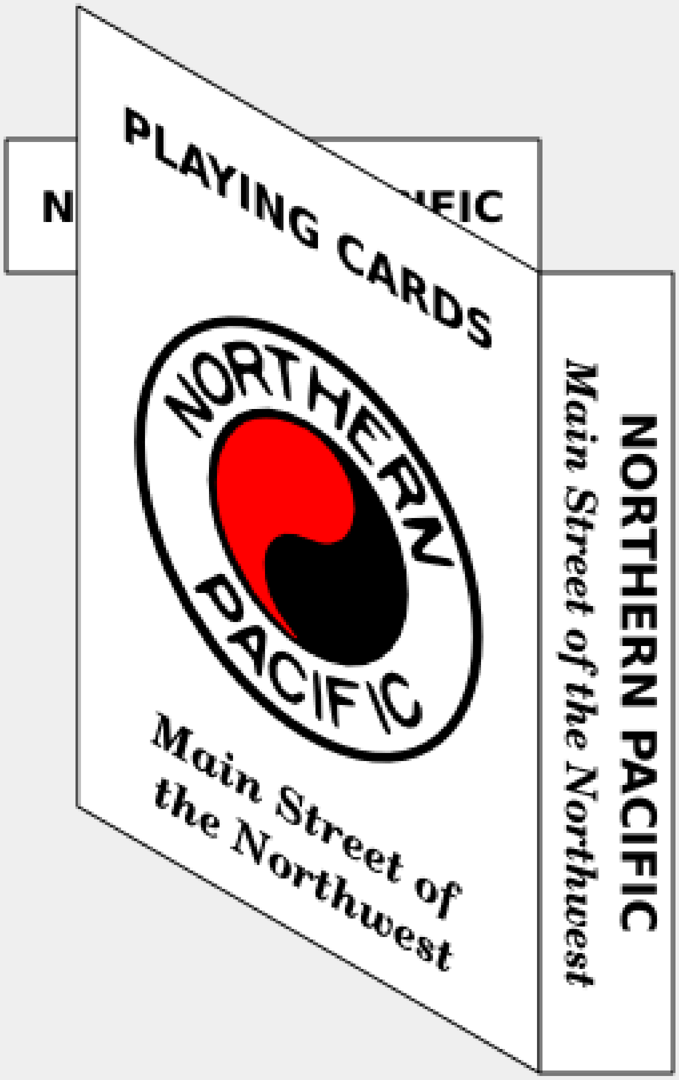
   Now the fun begins. We would like to distort each box side into a rhombus with the edges keeping the same length. This cannot be done directly. The distortions require several steps. The steps are to make the box sides narrower and then to skew them. The box top will also have to be rotated.

   When the sides are narrowed, we will want to keep the stroke width from narrowing at the same time. The option to scale stroke width when an object is scaled can be toggled on and off in the *Tool Controls* when the *Select Tool* is in use. Make sure the *Scale Stroke Width* icon is not highlighted. If it is, click on the icon to toggle off the option.

Select the box front (largest side). We want to be precise when we narrow the sides so we'll use the Object → Transform... (**Shift+Ctrl+M**) dialog. Bring up the dialog and select the *Scale* tab. Set the *Width* box to 86.603%. This is equal to *cos*(30°). (Use –13.397% in v0.45.) Click on the *Apply* button. Use the *Select Tool* to drag the box front back against the other lower box side. It should snap into place.

Next we need to skew the box front by 30°. This is easiest done with the *Select Tool*. Click on the box front a second time to change the *scaling* handles into *rotating* and *skewing* handles. When you do so, a small cross, the *Rotation center*, should appear in the center of the selection. This is the reference point for rotation or skewing. Click-drag this cross to the upper-right corner of the box front. Be careful that it doesn't snap to the corner of the *bounding box* The cross should be centered in the middle of the *Stroke*. If it isn't it will lead to small alignment problems at end. You can zoom in to check.

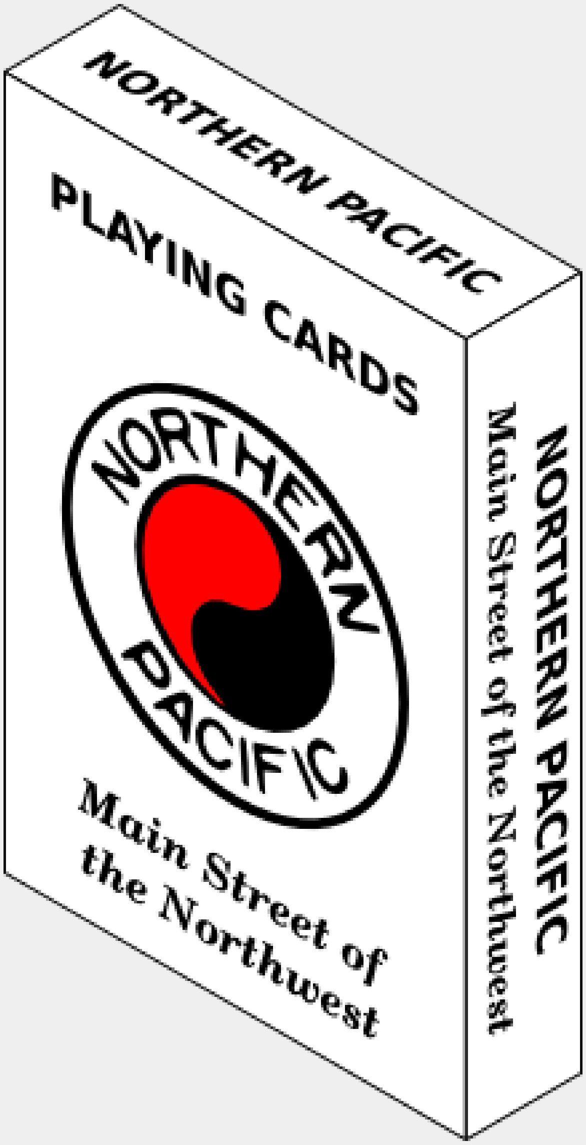
Now click-drag the *scaling* handle on the left side of the box front upward while holding down the **Ctrl** key. Using the **Ctrl** key will snap the skewing to multiples of 15°. Drag until the skew is –30° (watch the *Notification Region*).



After transforming the front of the box into place. The front of the box may be in front (as shown) or behind the top of the box depending on the order in which you grouped the sides.

Repeat the last steps with the box side on the lower right, just interchanging “left” and “right.”

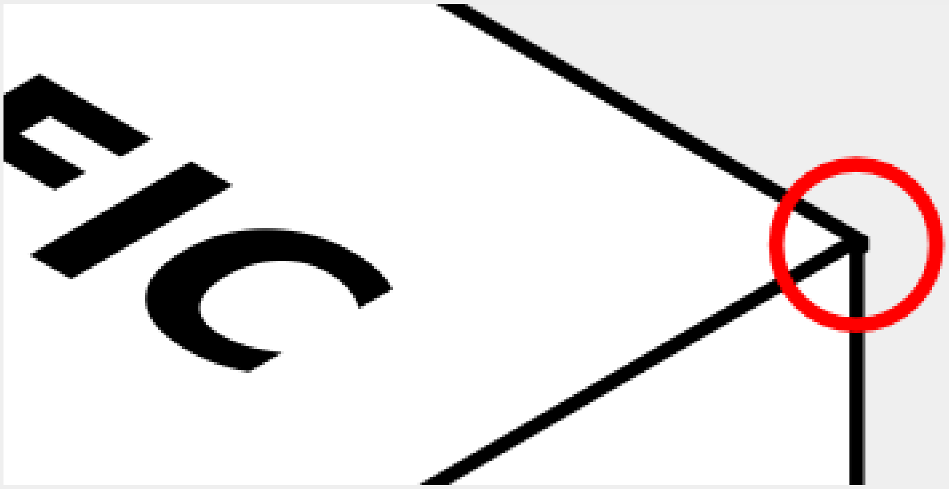
The box top also needs to be narrowed by the same amount as the others, but this time in the vertical direction. Set the *Width* back to its default value and the *Height* to 86.603% before scaling the rectangle. Before skewing, move the *Rotation center* to the lower right corner of the box top. Again make sure the cross is in the middle of the stroke width, aligned with the grid. Skew the rectangle 30° by dragging the top skewing handle right while using the **Ctrl** key. Finally, rotate the box top 30° clockwise by dragging the upper-left rotation handle while holding the **Ctrl** key down.



The finished box of playing cards.

The box of playing cards is now finished. Save your work as in the previous tutorials.

If you did not place the *Rotation center* exactly as specified (e.g., on the *bounding box* rather than the middle of the stroke), you may have a small misalignment in the box sides. One can either redo the steps or simply nudge the offending side into place using the **Arrow** keys while holding the **Alt** key down.



The corner of the top of the box inside the red circle doesn't meet the other corner correctly. This was the result of aligning the *Rotation center* to the *bounding box*.

The Northern Pacific Railway Logo - A Tracing Example

Table of Contents

A Can of Soup - A Three-Dimension Drawing with Gradients

Get the book.

Inkscape: Guide to a Vector Drawing Program

Get the book.

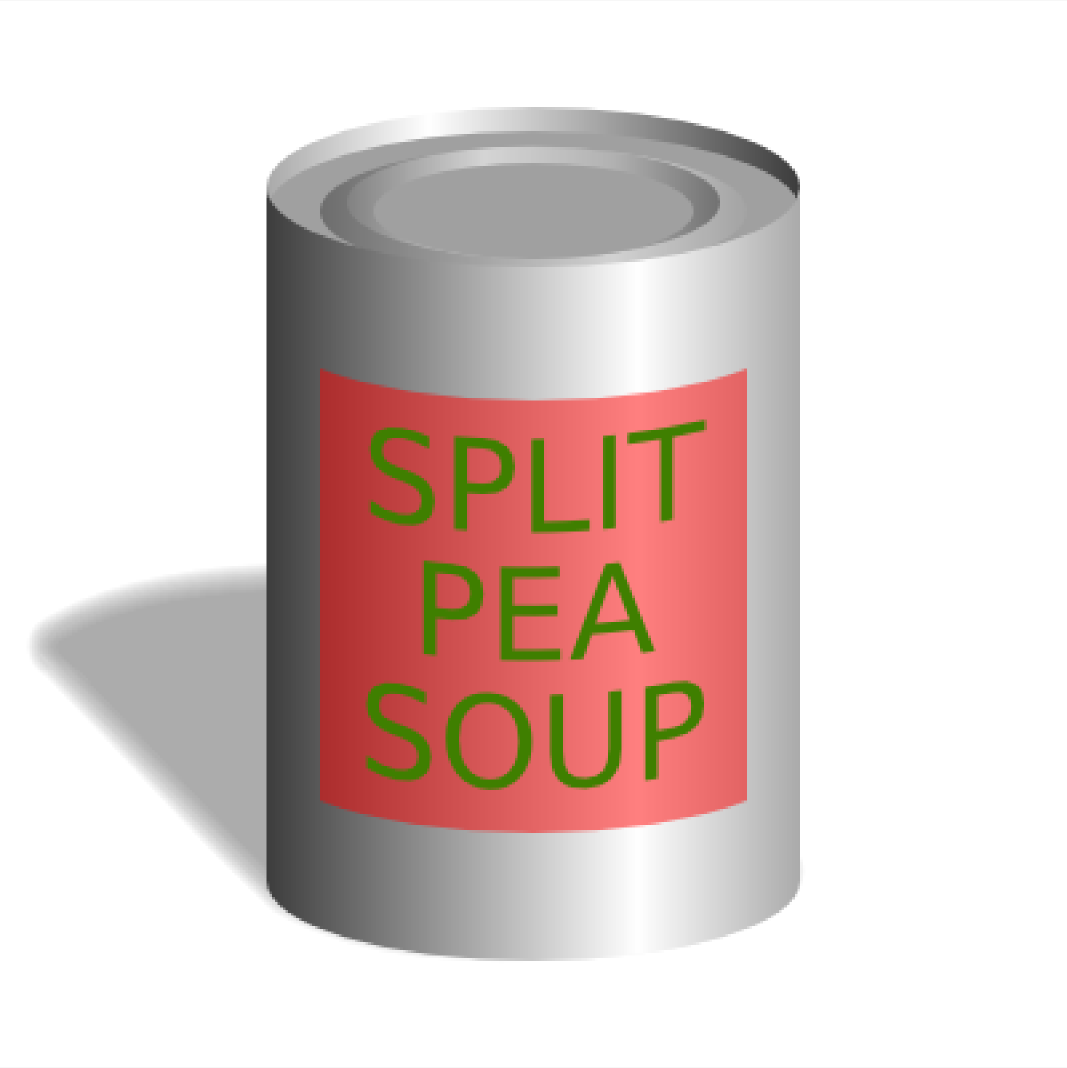
Inkscape » Quick Start » A Can of Soup - A Three-Dimension Drawing with Gradients

← Index →

A Can of Soup - A Three-Dimension Drawing with Gradients

Updated for v0.46 (Gradients).

We will use Inkscape to draw a soup can. This example will cover: combining and dividing paths, using *Gradients*, making shadows, and distorting text.



A souped-up soup can.

The steps we'll take are:

- Start Inkscape, set the drawing size, and specify a grid.
- Draw the can shape.
- Add a gradient.
- Add a shadow.
- Add a label background.
- Add the label text.
- Spiff up the can top.

Procedure 1.6. Drawing a Soup Can

1. **Set the canvas parameters.**

   Start Inkscape. Set the drawing size to 200 by 200 pixels. Set up a *Grid* with spacing of 5 pixels. Enable snapping of nodes to the *Grid*, disable snapping of *bounding boxes*.

2. **Draw the can shape.**

   The can will be composed of two lines connecting parts of two ellipses. (One could use a rectangle to obtain the straight lines but that has the tendency of producing extra nodes.)

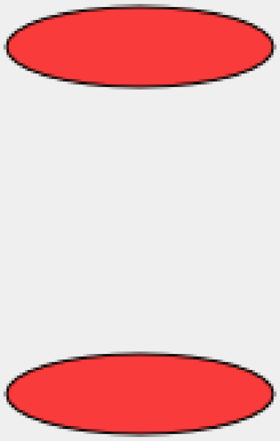
   a. **Draw the top of the can.**

      Click on the *Ellipse Tool* icon in the *Tool Box* on the left of the Inkscape window (or use one of the keyboard shortcuts: **F5** or **e**) to select the *Ellipse Tool*. Draw an ellipse to represent the top of the can by click-dragging between the 50 and 150 pixel marks on the horizontal axis and between the 150 and 180 pixel marks on the vertical axis.

   b. **Draw the bottom of the can.**

      Duplicate the ellipse by selecting the ellipse (if not selected) and clicking on the icon in the *Command Bar* or using the menu entry Edit → Duplicate (**Ctrl+D**). A copy of the ellipse will be placed on top of the original ellipse. The new ellipse will be left selected.

      Move the ellipse down by click-dragging it while holding down the **Ctrl** key to constrain the movement to the vertical direction. Move it down until the top is at the 50 pixel mark on the vertical axis. You now have the top and bottom of the can.



The top and the bottom of the soup can.

3. **Draw the side of the can.**

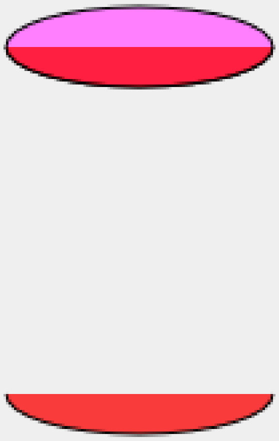
   The side of the can will be formed by joining the bottom half of the top ellipse to the bottom half of the bottom ellipse. In the process, we will sacrifice both ellipses. Because we still need a separate top for the can as it will be colored differently from the body, we'll duplicate the top ellipse and sacrifice the new copy.

Select the top ellipse and duplicate it as above. Then with the duplicate ellipse still selected, convert the ellipse into a path object by using the Path → Object to Path (**Shift**+**Ctrl**+**C**) command. The object will not appear to have changed but the underlying description is now an editable path.

To edit the path, select the *Node Tool* by clicking on the icon (**F2** or **n**) in the *Tool Box*. Select the top node of the path and delete it by clicking on the icon (Delete selected nodes) in the *Tool Controls* or by using one of the keyboard shortcuts: **Backspace** or **Delete**.

Open the path up by selecting the two side nodes and clicking on the (Split path) icon in the *Tool Controls*.

Repeat the above steps for the bottom ellipse (except for duplicating it). You should have a drawing that looks like the following one if you change the color of the still intact top ellipse:

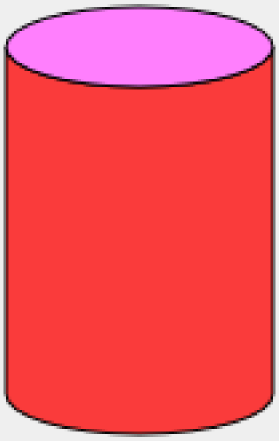


The two half ellipses that will form the can side along with a full ellipse (color changed) for the can top.

Next the top and bottom half ellipses need to be joined together. Select both and combine into one path with the Path → Combine (**Ctrl**+**K**) command.

With the *Node Tool*, select the two leftmost nodes, one from the top and one from the bottom. Join them by clicking on the (Join paths at selected nodes with new segment) icon in the *Tool Controls*. Repeat for the two rightmost nodes.

You should now have a well-constructed can side, as shown below.

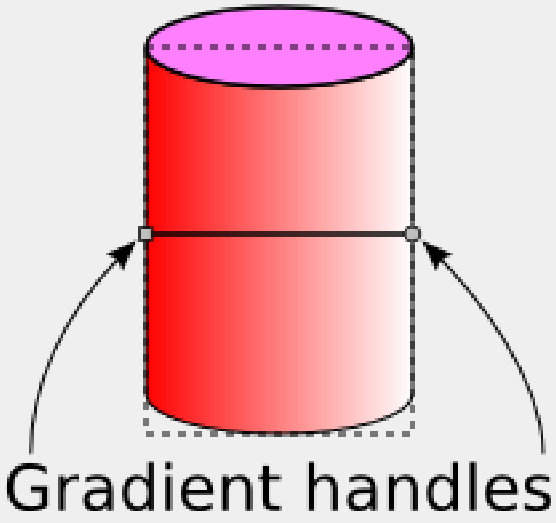


The side of the can after completing the side's path.

- **Add a gradient for a 3D effect.**

A *Gradient* can represent the reflections off the curved part of the can. To add a *Gradient*, open the *Fill and Stroke* dialog (Object → Fill and Stroke... (**Shift**+**Ctrl**+**F**)). Select the *Fill* tab, if not already selected, and with the can side selected, click on the *Linear gradient* ( ) icon. This will create a default *Gradient* across the can side using the path's preexisting color.

The *Gradient* needs a bit of work to make it look proper. Switch to the *Gradient Tool* by clicking on the icon (**Ctrl**+**F1** or **g**) in the *Tool Box*. Two *Gradient* handles will appear when the can side is selected.



After changing the *Fill* of the can side to a *Gradient*. Note the *Gradient* handles. They are displayed when a selected object has a *Gradient* and the *Gradient Tool*, *Node Tool*, *Dropper Tool* or one of the *shape tools* is in use.

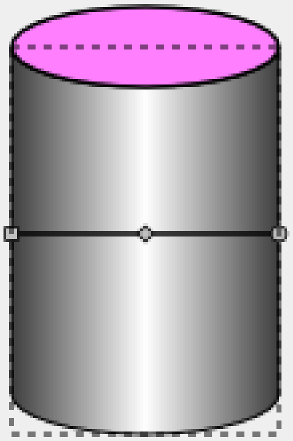
Gradients are defined in terms of *Stops*. A *Stop* has a color and position (offset) in the *Gradient*. The default *Gradient* has two *Stops*, both with the same color but with different transparencies. For the side of the can, we'll use three *Stops*.

Inkscape v0.46 adds on-screen *Gradient* editing. Add a third *Stop* by double clicking on the line connecting the two existing *Stops* with the *Gradient Tool* enabled and the can side is selected. The cursor will have an extra '+' sign when it is possible to add *Stops*. When you add a *Stop* it takes on the color of the *Gradient* at the place where it is added. The look of the *Gradient* will not change.

The *Stop* (handle) can be dragged to move it. Move it to the center.

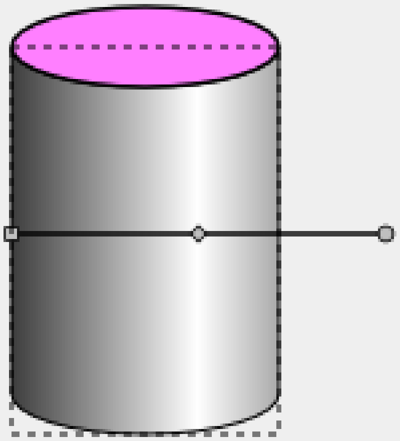
Now let's give our can a shiny metallic look. Select the leftmost *Stop* by clicking once on it. In the *Fill and Stroke* dialog both the *Fill* and *Stroke paint* tabs will show the color of the *Stop*. Change the color of the stop in the *RGB* tab to R: 60, G: 60, B: 60, A: 255.

Select the middle *Stop* and set its color to R: 255, G: 255, B: 255, A: 255. One way to do this is to simply click on a white swatch in the *Palette*. Finally, select the last *Stop* and set its color to R:60, G: 60, B: 60, A: 255. You should now have a metallic can with a highlight that is in the center.



After adding a stop to the *Gradient* and changing the colors of all stops.

Let's move the highlight to the side. One way to do this would be to further edit the *Gradient*, changing the position of the middle *Stop* and maybe lightening or darkening the side *Stops*. However, the easier way is to move the *Gradient* handles.



After dragging the right *Gradient* handle to move the highlight.

Before we continue, let's make a couple of quick cosmetic changes: Turn off the borders of both the can side and top. Change the color of the top to a silver (R: 160, G: 160, B: 160).

A silver can.

- **Adding a shadow.**

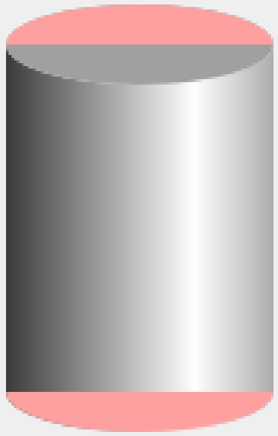
Our can has a light shining on it from the right but no corresponding shadow. We'll fix that now. To do it perfectly is not an easy feat. Inkscape is a 2D drawing program and cannot project shadows for 3D objects (try PovRay for that). But we can do a pretty good approximation.

a. **Create the shadow object.**

   For the shadow, we need to combine copies of the top and of the side of the can into one object. We'll play a similar game to that used to create the side of the can.

   Select the top of the can (an ellipse) and duplicate it. Change its color to make it easier to keep track of. Any color will do. Convert the new ellipse to a path (Path → Object to Path (**Shift**+**Ctrl**+**C**)). Remove the bottom node (change to the *Node Tool*,). Select the two side nodes and remove the line in between them ( ).

   Select the side of the can, duplicate it, and change its color. Remove the top three nodes. Select the remaining two side nodes and remove the line in between them.

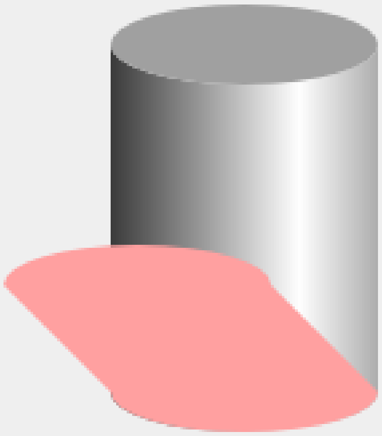


After creating the top and bottom of what will become the shadow.

Select both new objects. Combine them into one path (Path → Combine (**Ctrl+K**)). Select the two nodes on the left side, join them ( ), and convert the path in between to a line ( ). Do the same for the two right nodes.

- **Distort the shadow shape.**

Select the three top nodes and move them down and over, as shown below.



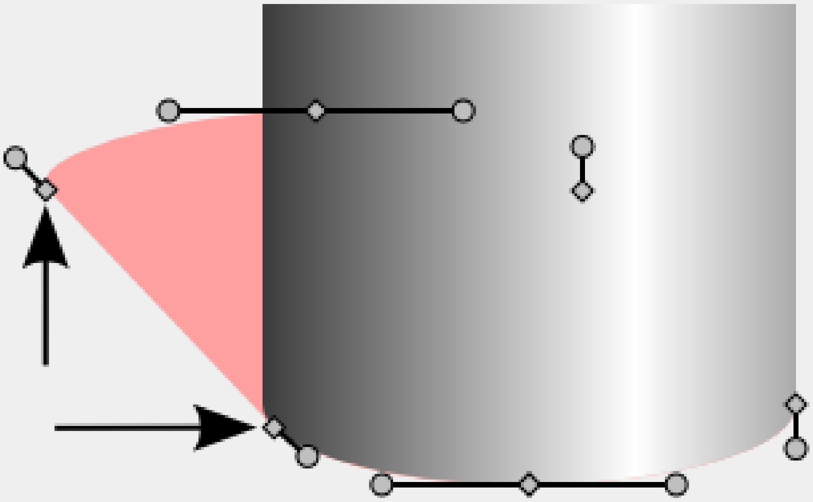
Distorting the shape for the shadow.

It is clear that the side nodes are not properly positioned and the path is distorted. We'll make a few adjustments to set this straight. But first put the shadow object behind the can (Object → Lower to Bottom (**End**)).

Node *handles* are used to control the direction and curvature of a path on either side of a node. The handles are the circles attached by straight lines to a node that are displayed when the node is selected. The lines are always tangent to the path at the point the path intersects the node. The distance between the handle and its node controls the curvature of the path—the farther away, the less curved is the path near the node. If no node handles are visible, make them visible by clicking on the icon in the *Tool Controls*.

The nodes indicated by the arrows in the following figure need to be adjusted. The handle on the leftmost node needs to be rotated a small amount counterclockwise so that the handle line is parallel to the side of the shadow. This produces a smooth transition between the straight side and the curved part of the shadow. To make the rotation, drag the handle while holding down the **Alt** key (which keeps the distance between the handle and node constant).

The other node indicated by the arrow also needs to have its handle adjusted in the same way. In addition, the node should be moved slightly down to the right so that the shadow's straight edge is tangent to the can's bottom.



Adjusting the shadow. The two nodes indicated by the arrows have been adjusted by rotating the handles slightly so the handle lines are parallel to the straight side of the shadow. In addition, the lower node has been shifted slightly.

Now change the color of the fill to gray (R: 127 G: 127 B: 127).

- **Soften the shadow's edge.**

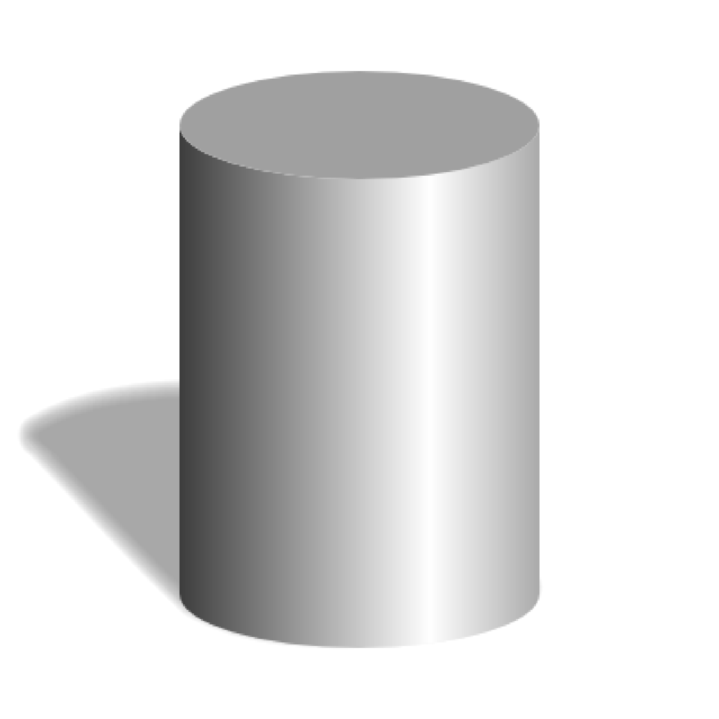
The easiest way to soften the shadow's edge is to use the *Blur* slider on the *Fill* tab of the *Fill and Stroke* dialog. A value of 3% gives a nice shadow.

The *Blur* slider uses a *Gaussian Blur* filter. *Filters* are not well supported yet in many web browsers so it might be better to use an alternative method of softening the shadow. One can use the *Inside/Outside Halo* effect. This effect makes multiples copies of an objects path, each time enlarging or shrinking it a small amount. Each copy has a small opacity so the net effect is a blurred object. The copies are placed in a *Group* that acts as a single object.

To use the effect, select the shadow, then call up the effect from the *Effects* menu. (If the *Effects* menu is missing from the *Menu Bar*, see Chapter 18, *Effects*, for details on how to enable it.) In the dialog that appears, set the *Width* to 5 and the *Number of Steps* to 11. This will produce a blur that is 5 pixels wide with 11 layered objects. The original shadow is still selected. It is no longer needed and can be deleted by using one of the keyboard shortcuts: **Del** or **Ctrl**+**X**.

Select the new blurred shadow. Move the shadow to the back (Object → Lower to Bottom (**End**)).

One problem needs to be taken care of. The shadow leaks out from the bottom of the can in the front. Select the shadow and move it slightly up and to the left. You can do this either by dragging the shadow with the mouse or using the **Arrow** keys. Holding down the **Alt** key while using the **Arrow** keys will allow finer adjustments.



A basic can.

- **Add a label.**

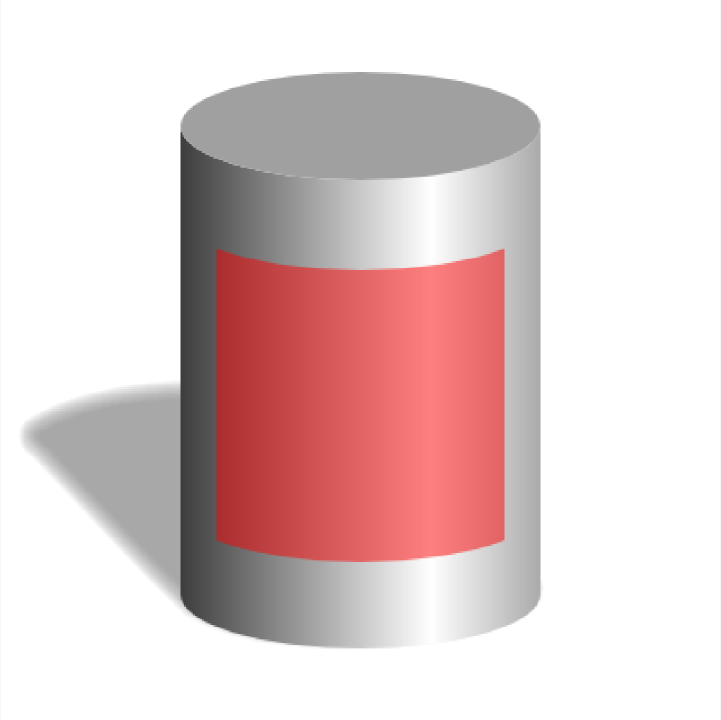
The label has the same curvature as the side of the can. To make the label, we'll take a slice from the can side and then decrease its height.

Select the side of the can and duplicate it. Then select the *Rectangle Tool* and draw a rectangle that extends from above the can to below the can and extends from 60 to 140 pixels in the horizontal direction.

Select both the new rectangle and the duplicate of the can side. Use the Path → Intersection (**Ctrl+***) command to form the label from the intersection of the two selected objects.

Change the fill of the label to a solid color by clicking on the *Flat color* (🟦) icon in the *Fill* tab of the *Fill and Stroke* dialog. Change the color of the label to red (R: 255: G: 0: B: 0). Change the opacity (A) to 127. This will allow a bit of the can highlight to leak through. (If you desire a metallic label, just change the color of the gradient.)

With the *Node Tool*, select the three nodes along the top of the label at the same time and move them down 25 pixels. Move the bottom three nodes of the label up 25 pixels.



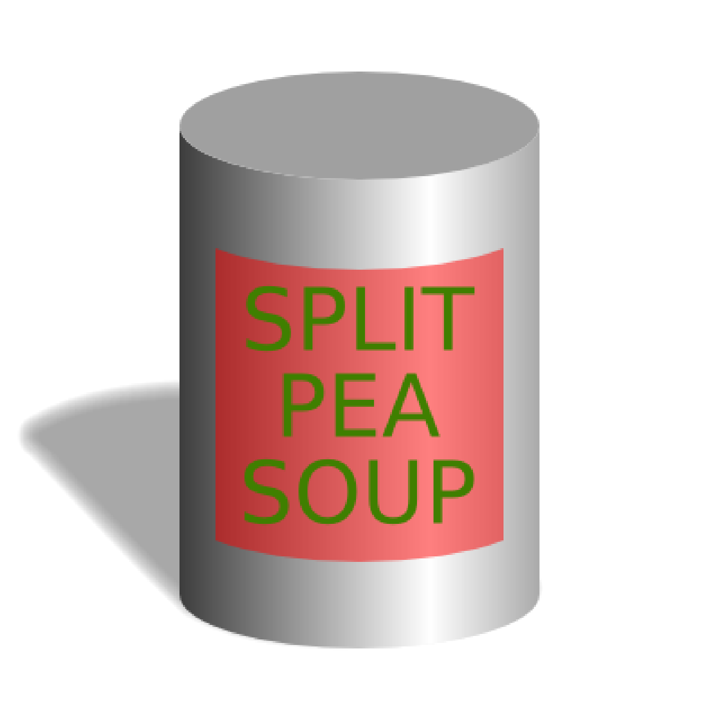
A can with a label.

- **Add text to the label.**

To add text to the label, select the *Text Tool*. Click somewhere on the canvas and type in the text "SPLIT PEA SOUP" with a carriage return after each word. To change the size and style of the text, use the drop-down menus in the *Tool Controls*. Pick an appropriate font (I've used Bitstream Vera Sans) and font size (24). Click on the ≣ icon in the *Tool Controls* to center the text. The line spacing may need to be adjusted. This can be done with the *Text and Font* dialog. Call up the dialog by either clicking on the T icon in the *Command Bar* or through the menu entry (Text → T Text and Font... (**Shift+Ctrl+T**)). Change the line spacing to 100%.

Finally, change the text to a nice Split Pea green with the *Fill and Stroke* dialog (R: 63, G: 63: B: 0).

The text can be centered in the label by using the *Align and Distribute* dialog. Bring up the dialog by clicking on the icon in the *Command Bar* or through the menu entry Object → Align and Distribute... (**Shift+Ctrl+A**). Select both the label and text objects. Set the reference for the alignment by selecting from the *Relative to* drop-down menu *First selected* if you selected the label first or *Last selected* if you selected it last. Then click on both the *Center on vertical axis* ( ) and the *Center on horizontal axis* ( ) icons.



A can with a label and text.

Updated for v0.45.

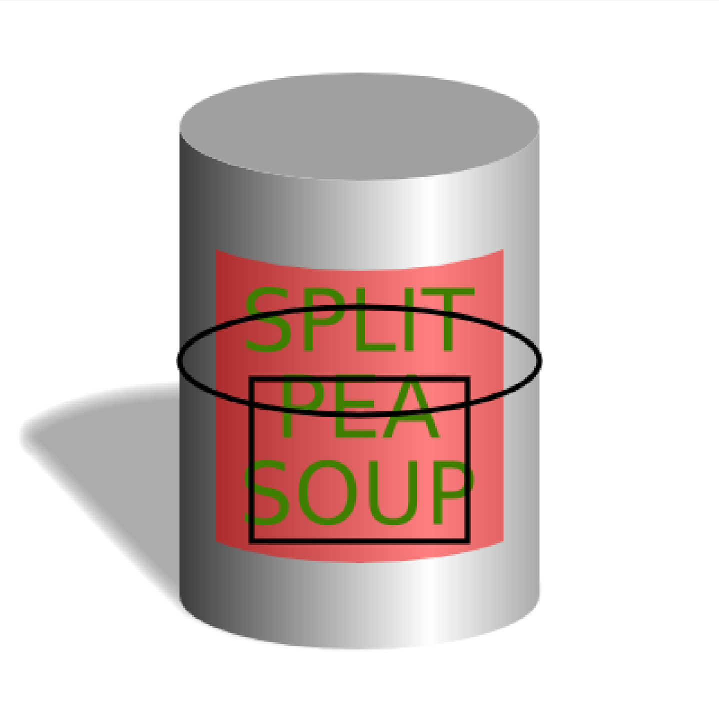
Now we have flat text on a round can! Inkscape can shift and rotate individual letters of text but it cannot skew the text as is needed for the characters toward the edge of the can. In order to skew the letters, we will convert the text into a path and modify the path. To do this, select the text and then use the Path → Object to Path (**Shift+Ctrl+C**) command. The text has become a path object and can no longer be edited as text.

Before v0.45 of Inkscape one would have to break the text object into paths and then tweak each part with the *Transform* dialog. As of v0.45, one can use a couple of very handy *Effects* to do the hard work for us.

Start by using the *Add Nodes* effect to increase the number of nodes in the path. This will improve the look of the final letters. Select the text path, then select Effects → Modify Path → Add Nodes... . A small dialog will pop up, allowing you to set the upper limit to the space between nodes. Set the *Maximum segment length* to 5.0 and click the *OK* button.

Next we'll use the *Pattern along Path* effect to put the text on a path. There is a Text → Put on Path command, but this uses the *SVG textPath* specification, which would result in the letters being rotated to follow the path.

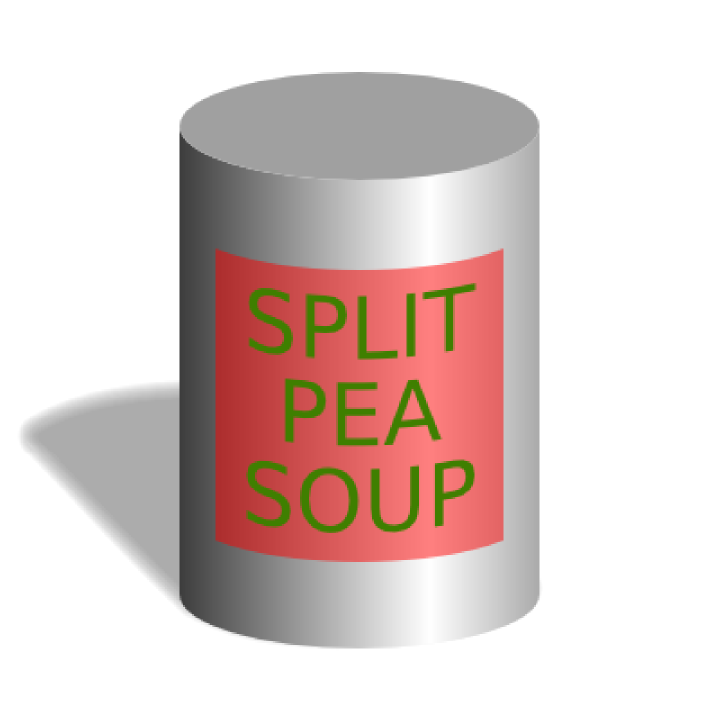
We need a path to put the text on. The path should be an arc with the same shape as the curve of the top ellipse but have the width of the text. Select the top *Ellipse* of the top of the can and duplicate it (Edit → Duplicate (**Ctrl+D**)). Move the *Ellipse* straight down to the center of the can. Remove the *Fill* but add a *Stroke*. Add a *Rectangle* that overlaps the bottom of the *Ellipse*, centered on the can and with the width of the text.



Creating a path for the text from an oval that is a duplicate of the can top and a rectangle with the width of the text.

The *Rectangle* should be on top of the *Ellipse*. Select both and use the Path → Cut Path (**Ctrl+Alt+*/***) command to divide the *Ellipse* into two pieces. The *Rectangle* will disappear. Delete the top piece of the *Ellipse*.

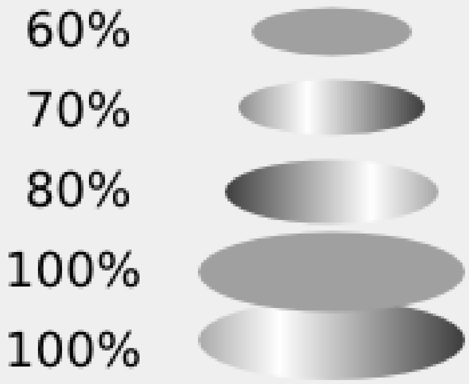
Finally, select first the text and then the new path. Call up the *Pattern Along Path* effect (Effects → Generate from Path → Pattern along Path). This will open a dialog. In the dialog, select *Single* in the *Copies of the pattern* menu and *Ribbon* in the *Deformation type* menu. All the entry boxes should be 0.0 and none of the boxes should be checked. Click the *OK* button. The text will probably be backward. Flip it with the Object → Flip Horizontal (**H**) command. Finally, delete the small arc.



A can with a label and wrapped text.

- **Spiff up the can top.**

There is no stopping at the level of detail you can added. In the final figure, a series of ovals of different sizes, positions, and gradients have been used to give the top of the can more realism. These ovals are shown in an expanded view in the next figure.



The ellipses that make up the can top in an expanded view with the relative sizes indicated.

All the ellipses were created by taking the original can top, duplicating it, and then using the *Scale* tab of the *Transform* dialog. The fill was then changed to either a flat fill or a linear gradient with the same colors as used for the rest of the can.

A Box for Cards - An Isometric Projection

Table of Contents

A Vine Design - A Tiling Example

Get the book.

Inkscape: Guide to a Vector Drawing Program

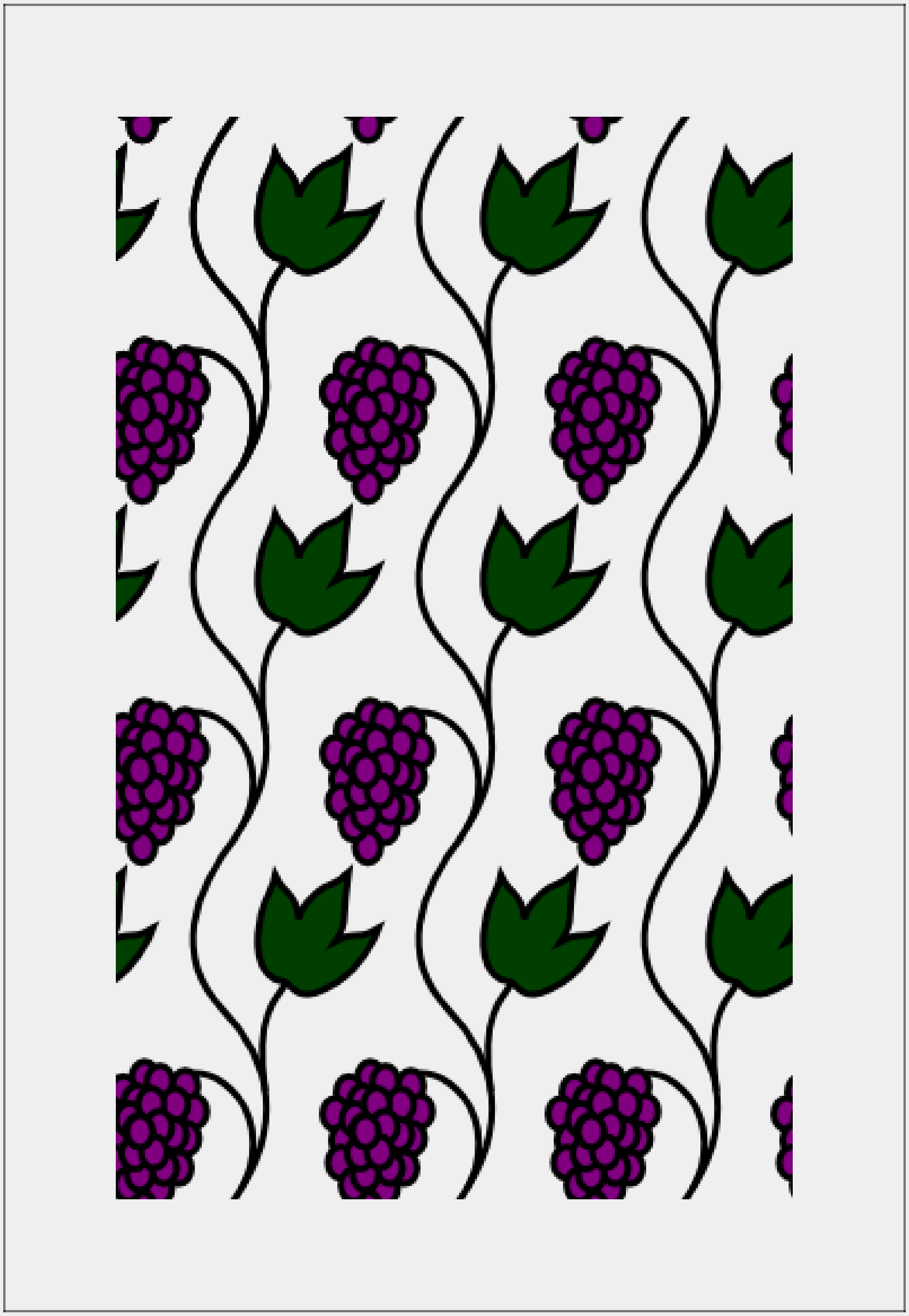
Get the book.

Inkscape » Quick Start » A Vine Design - A Tiling Example

Index

A Vine Design - A Tiling Example

Inkscape has a *Create Tiled Clones* dialog that can be used to produce complex tilings. This tutorial covers preparing a base tile that can be used for such tilings. The objects in the base tile can overlap the tile boundaries to produce continuous designs, as shown below.



A 3x3 tiling of a base tile.

The steps to produce a repeated tiling are:

- Start Inkscape and set the drawing size.
- Create a tile prototype.
- Clone tile.
- Decorate tile.
- Use base tile for tiling.

Procedure 1.7. Creating a Tile Pattern

1. **Set up the canvas.**

   To begin, start Inkscape and set the drawing size to a width of 400 and a height of 460 pixels. Create a *Grid* with spacing of 10 pixels. Turn on snapping of nodes to *Grid* (and off snapping of *bounding boxes* to *Grid*). Don't forget to turn the *Grid* on.

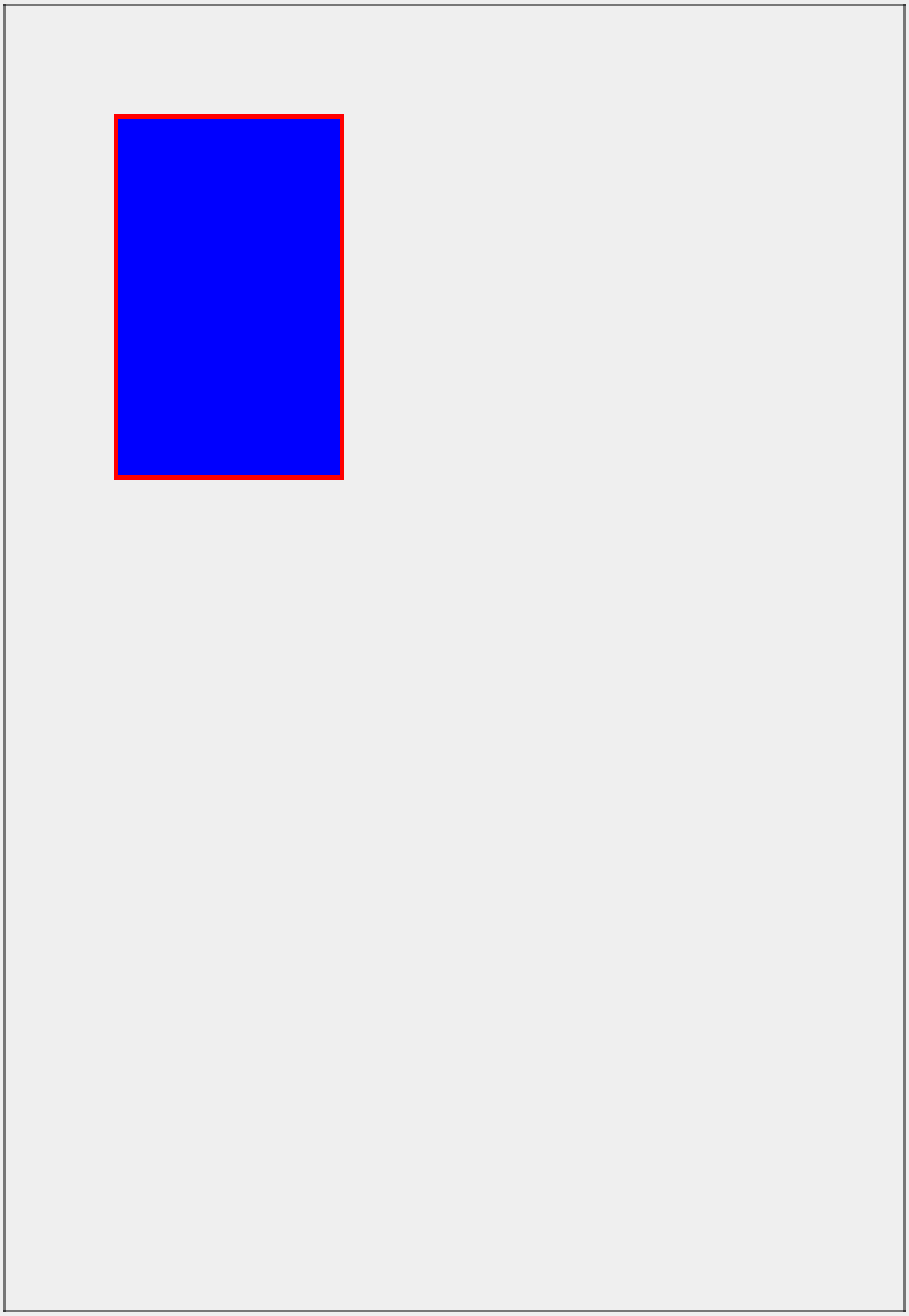
2. **Create a tile prototype.**

   We need to create a tile prototype that consists of a rectangle in a *Group*. Having a *Group* will allow adding new objects to the tile. Prior to v0.46, one could only *Group* two or more objects. This restriction has been removed so that one can put the rectangle in a *Group* by itself.

   Using the *Rectangle Tool*, create a rectangle 100 pixels wide by 120 pixels tall with the upper-left corner 50 pixels from the left edge and top. This will be the size of our base tile. Tile spacing will be determined by the *bounding box* of this rectangle. There are two definitions for the *bounding box*. The first, called the *Visual bounding box*, includes the *Stroke* width in the calculation while the second, called the *Geometric bounding box*, is calculated using just the nodes of an object. It will be easier to make tilings without worrying about the *Stroke* width, so select the *Geometric bounding box* in the *Tools* section of the *Inkscape Preferences* dialog.

   So we can visualize the tiles, give the tiles a *Stroke* by either using the *Fill and Stroke* dialog or by using a **Shift**+**Left Mouse Click** on one of the color swatches in the *Palette*. You can also give the tile a *Fill* by simply clicking on a swatch.

   Now *Group* the tile by selecting the rectangle and using the Object → Group (**Ctrl**+**G**) command.

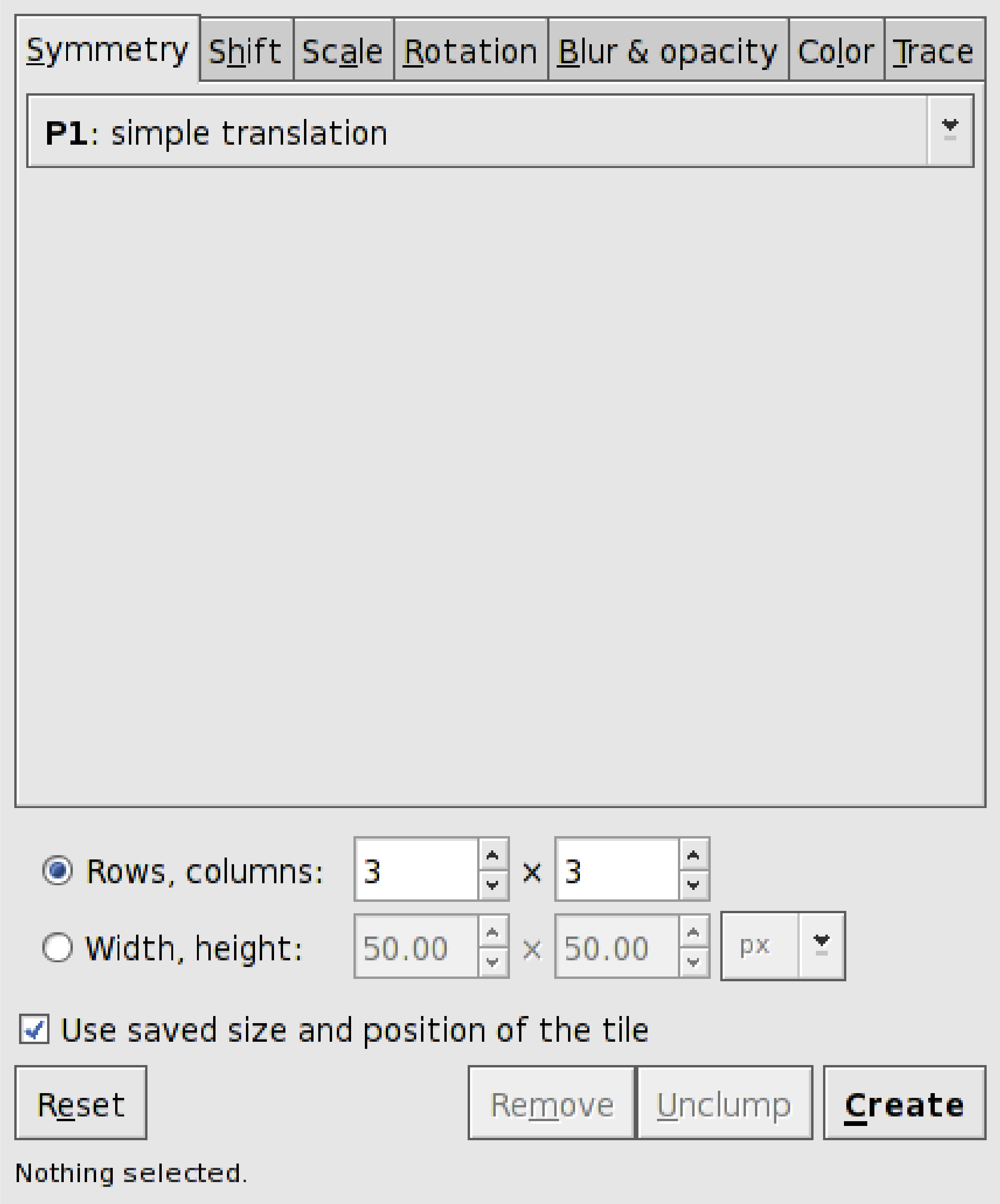


The base tile.

3. **Clone the tile.**

   The next step is to clone the tile. This will give you a way to visualize the tile as it is decorated, making sure that objects that are near or overlap the boundary fit in properly.

   Bring up the *Create Tiled Clones* dialog (Edit → Clone → Create Tiled Clones...).



P1: simple translation

● Rows, columns: 3 × 3

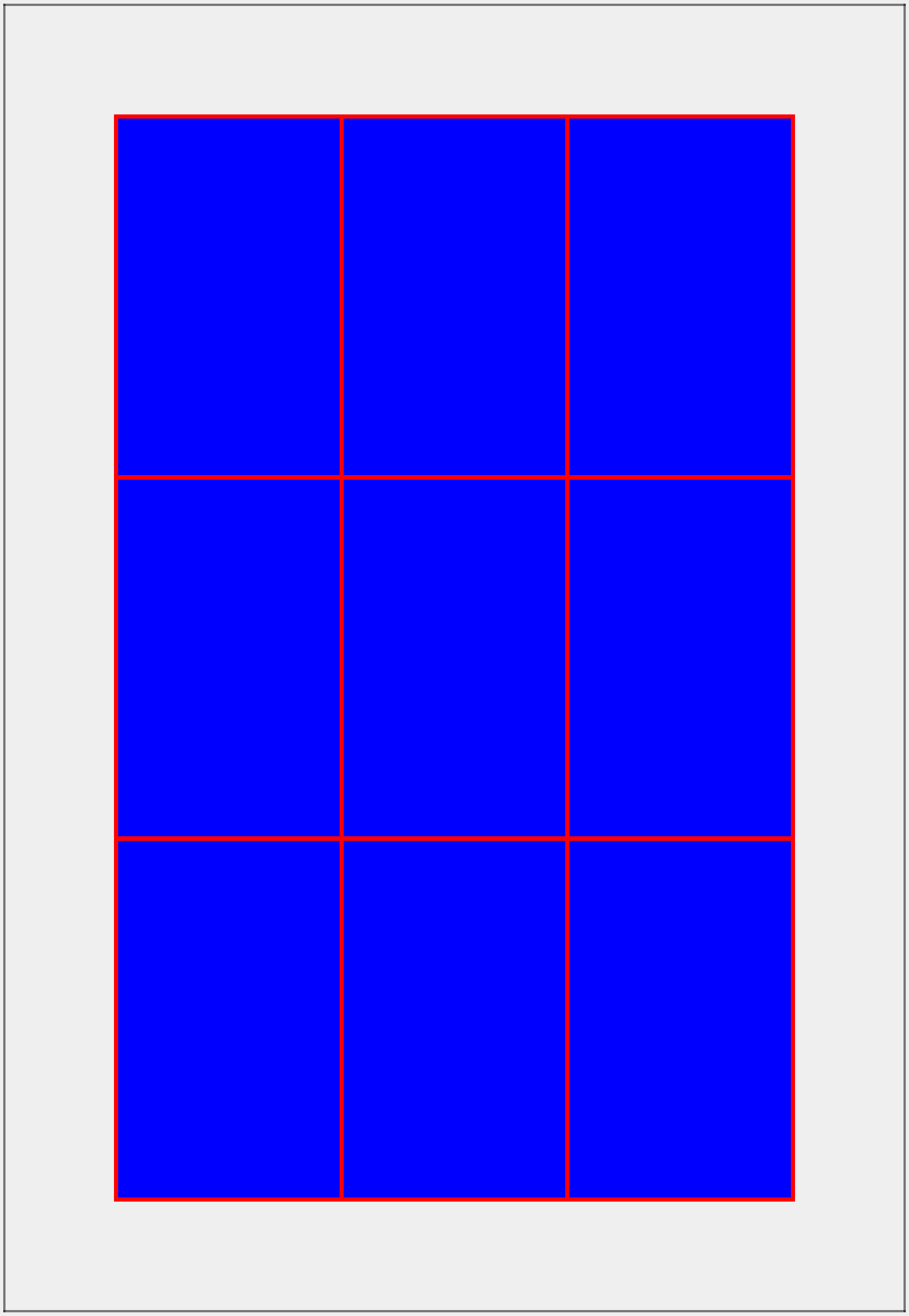
○ Width, height: 50.00 × 50.00 px

☑ Use saved size and position of the tile

Nothing selected.

Tile Clones dialog.

Select *P1: simple translation* from the drop-down menu on the *Symmetry* tab. This will produce a rectangular array of tiles. Choose the *Rows, columns* option and set the number of columns and rows to three by entering 3 in both entry boxes. Make sure the base tile is selected in the drawing (*Notification Region* should state that a "Group of 1 object" is selected) and click on the *Create* button in the dialog.

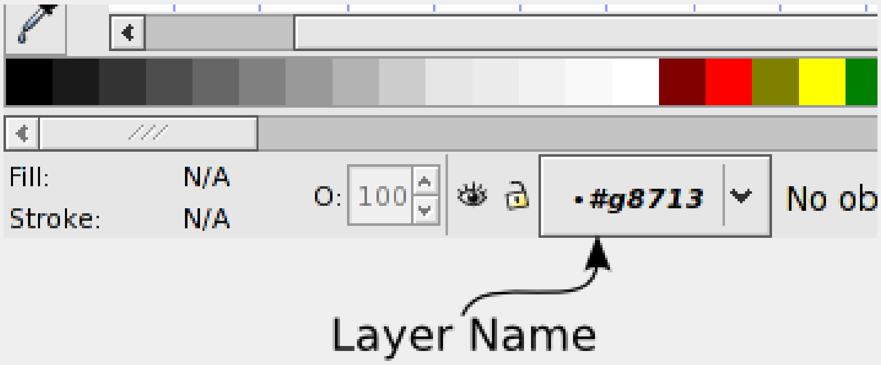


A 3x3 array of the base tile.

4. **Decorate the tile.**

   It is time to draw the design on the base tile. Nine copies (clones) of the base tile have been created, including one on top of the original tile. It is necessary to modify the original tile if the copies are also to be modified. Select the original tile by clicking on any of the clones (*Notification Region* will indicate a clone is selected) and using Edit → Clone → Select Original (**Shift+D**). The *Notification Region* should now say “Group of 1 object.” The original tile is below a clone. Bring it to the top by using Object → Raise to Top (**Home**) (click on the in the *Tool Controls* bar).

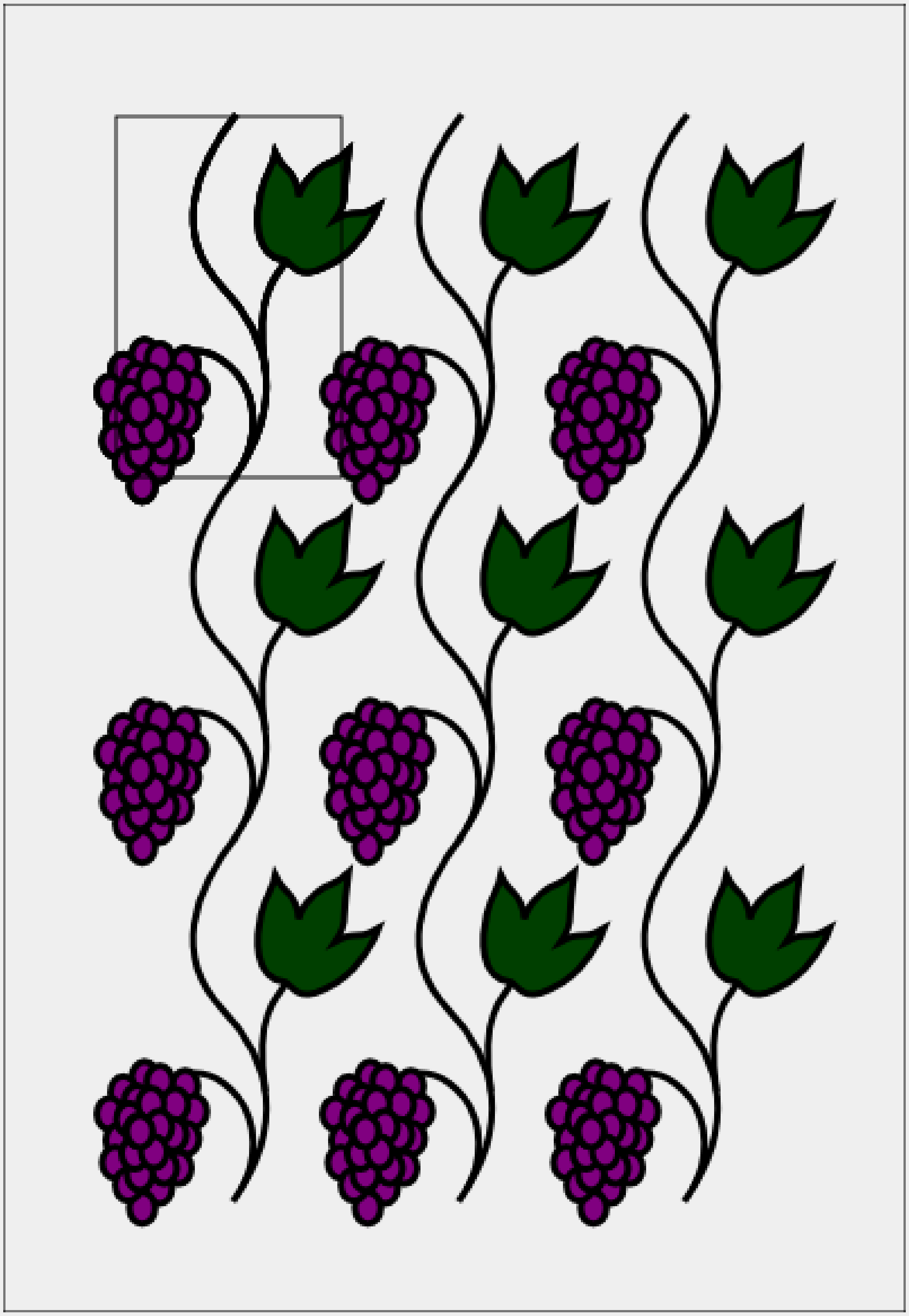
“Enter” the Group by either double clicking on the base tile or selecting *Enter group #xxxx* from the menu that pops up when you **Right Mouse Click** the base tile. The Layer name displayed in the Status Bar will start with '#g' indicating that a *Group* is open for editing (the *Group* has been temporarily been promoted to a *Layer*). While the *Group* is open, any object added to the drawing will be a member of the *Group*.



Fill: N/A
Stroke: N/A
O: 100
#g8713
No ob

The Layer name in the Status Bar showing that a Group is open for editing.

Now that the Group is open, add any details you want. Our example has a continuous grape vine pattern. The main vine is a single line with the place and angle of the ends carefully adjusted for continuity at the base tile boundary. The Fill and Stroke of the base rectangle have been removed (the rectangle, itself, has not been deleted as it will be useful in the next step).



A 3x3 array of the decorated base tile. Note how some of the decoration extends outside the base tile (indicated by the black rectangle).

5. **Use the base tile for tiling.**

   The newly created base tile can be used in several ways to decorate the *Fill* another object. The easiest way is to use the base tile with the *Create Tiled Clones* dialog to create a simple tiling that is then clipped with the path of the object.

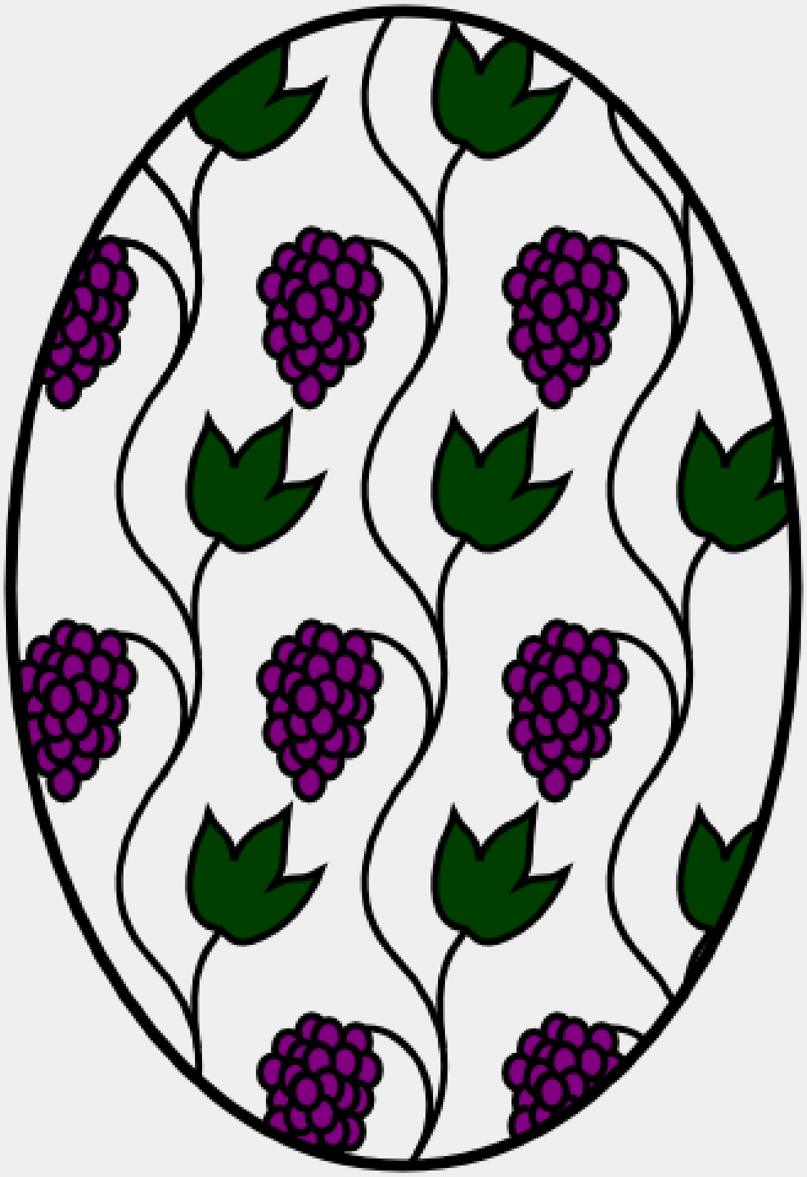
   a. Delete all the clones of the base tile (either by selecting each clone and deleting or by selecting the base tile and clicking on the *Remove* button in the *Create Tiled Clones* dialog).

b. Select base tile (if not already selected).

c. With *Create Tiled Clones* dialog, create a tiling large enough to fill the target object. Make sure *Use saved size and position of the tile* box is checked. The tiling will then use the saved dimensions of the base tile before it was decorated (the dimensions when the first tiling was made) rather than the current *bounding box*, which may include objects that extend outside the border. (Once an object has been tile cloned, Inkscape keeps a private record of the original tile size and position that can be used for later recloning.)

d. If one of the base tile objects has partial transparency it may be necessary to delete (or move) the base tile. If the base tile is deleted, the links to the cloned tiles are broken but the cloned tiles remain.

e. *Group* the tiled clones. Move the object to be filled above the grouped clones in *z-order*. Select both the object and the grouped clones and use the Object → Clip → Set command to make the clipping.



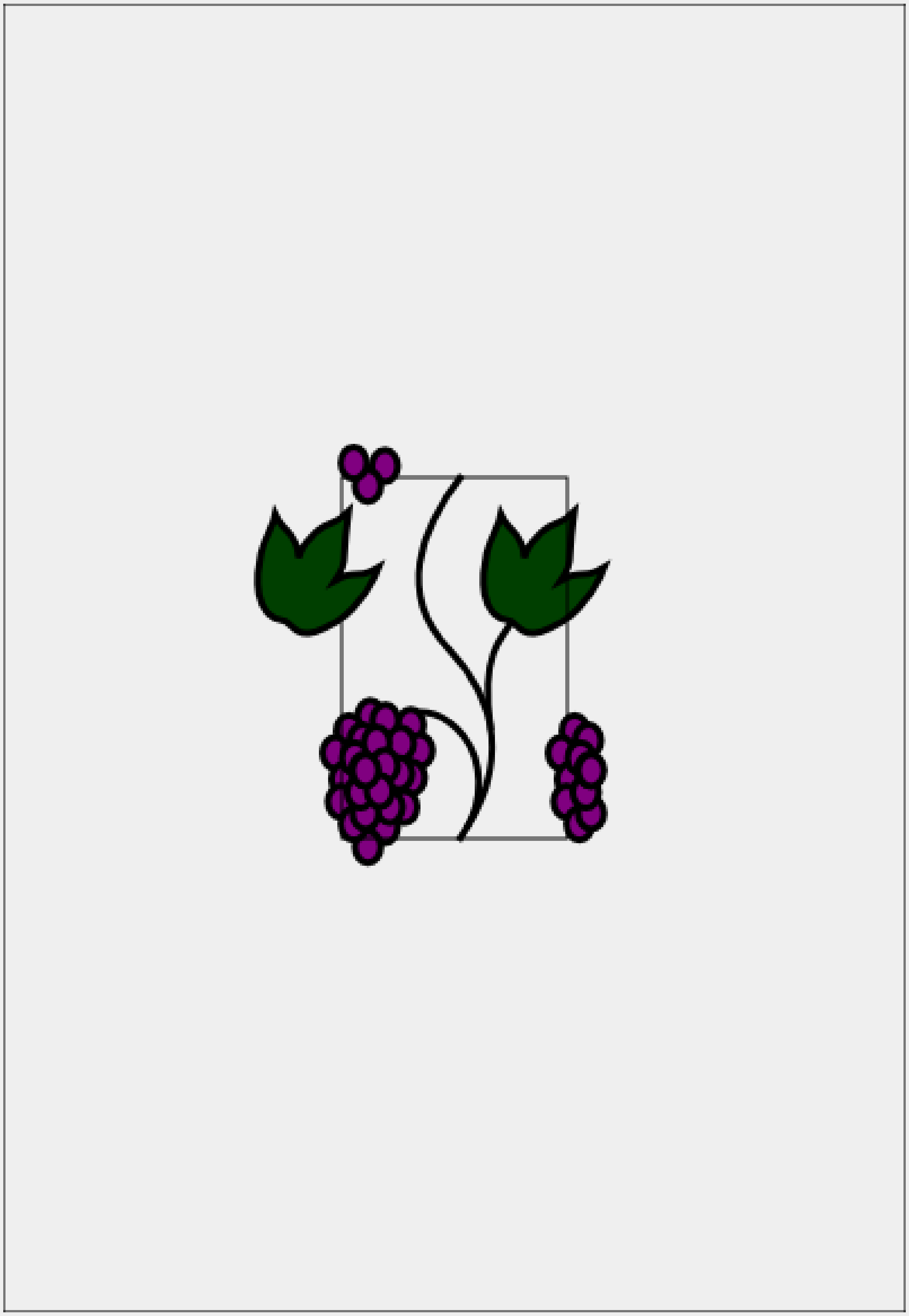
An oval path has been used to clip the tiling. To get the oval border, the oval path was duplicated prior to making the clipping. The duplicated oval was then given a black *Stroke* and no *Fill*.

The base tile can be used for a *Pattern* by selecting it and using the command Object → Pattern → Objects to Pattern (**Alt+I**). When the *Pattern* is first used, you will notice that the spacing is wrong if some of the objects extended outside the borders of the base tile. This can be corrected by using the *XML Editor*. In the *svg:defs* section, select the *svg:pattern* item that corresponds to the object with the pattern fill. This entry may be linked to another pattern. Follow the links until you find the entry with *height* and *width* attributes (and no other pattern links). Change the *height* and *width* attributes to the height and width of the base tile. Note that some *SVG* renderers will not draw areas that are outside the nominal area of the pattern (defined by the *height* and *width* attributes) despite setting the *style* attribute to “overflow:visible”. Batik (which is used for the PDF version of this book) has rendering problems with this type of pattern. Evince, a popular Linux PDF displayer, also has a different bug rendering *Patterns*. Acroread will display *Patterns* correctly.

The base tile has been converted to a *Pattern* that has been used to fill the oval. The pattern's internal height and width attributes have been adjusted to match that of the base tile.

For some purposes, it may be preferred to have the entire design within the edges of the base tile. This is a bit tricky.

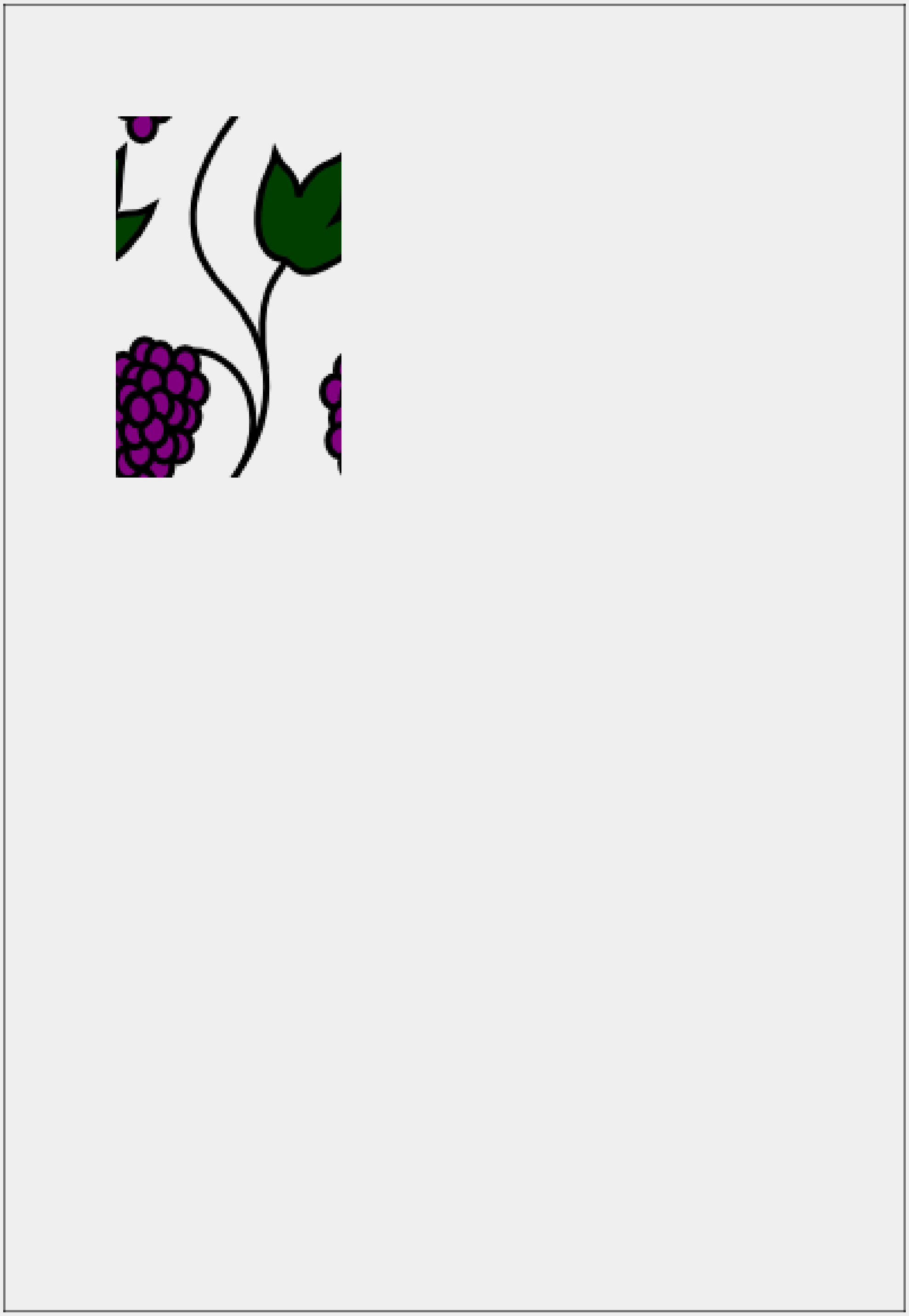
a. Unlink all the clones by selecting the clones and using Edit → Clone → Unlink Clone (**Shift+Alt+D**).
b. Delete the center clone of the 3x3 clone array.
c. Select base tile and move it to the center (it may be under a clone). Use the *Move* tab of the *Transform* dialog to precisely move the base tile (by its width and height). The internal information about the position and size of the base tile that is used for cloning is preserved.
d. For each object that overlaps the base tile but is not part of it, enter the *Group* it belongs to (double click on the *Group*), cut the object (Edit → Cut (**Ctrl+X**)), enter the *Group* of the base tile, and paste the object in place (Edit → Paste In Place (**Ctrl+Alt+V**)). Pay careful attention to the *Notification Region* to keep track of what is selected and the *Layer* information in the *Status Bar* to see which *Group* is open.
e. Delete the clones with the leftover objects.



The base tile with all the necessary decorations, before clipping.

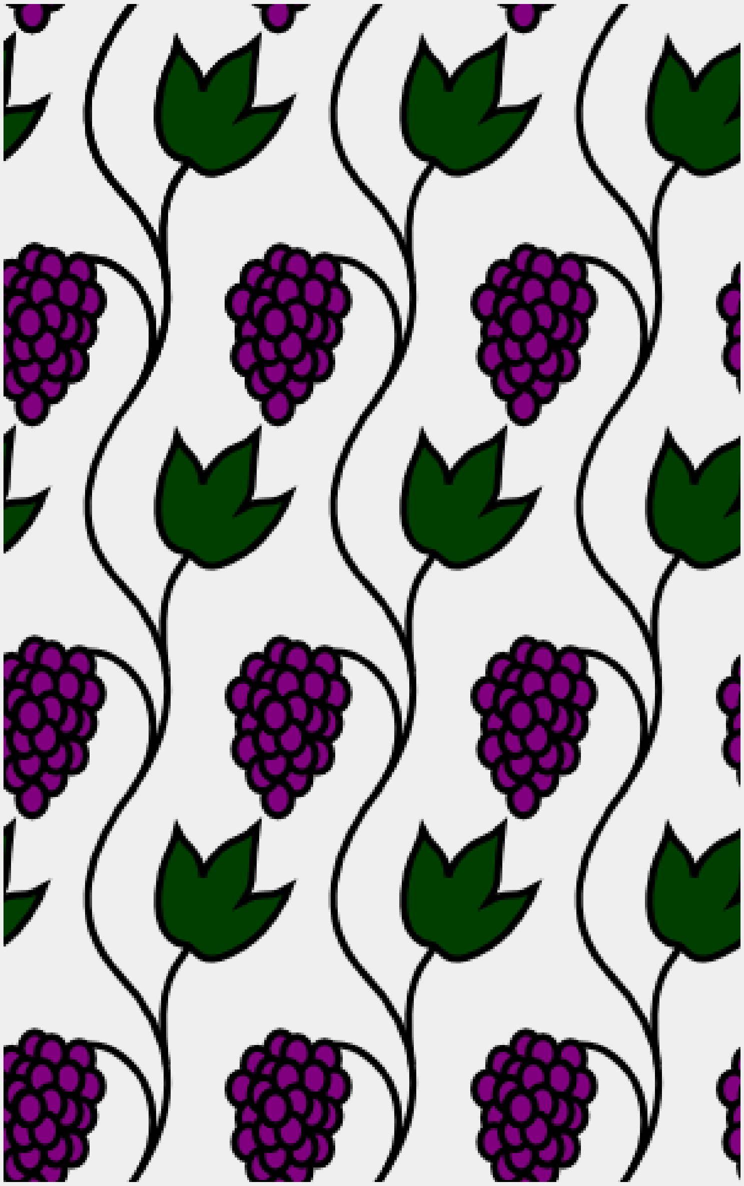
At this point, one could convert the base tile to a *Pattern* (see above). There are still problems with some *SVG* renderers that can be worked around by modifying the *transform* attribute with a negative translation in *x* and *y* for the *Group* in the *Pattern*. Note that Inkscape ignores this *transform* attribute.

f. In the opened base tile group, select the rectangle and cut it. Exit the group and paste it in place. Select both the base tile group and the rectangle and then use the Object → Clip → Set command. The base tile is now finished. You can translate it back to its original position using the *Transform* dialog.



The base tile, clipped and moved back to its original position.

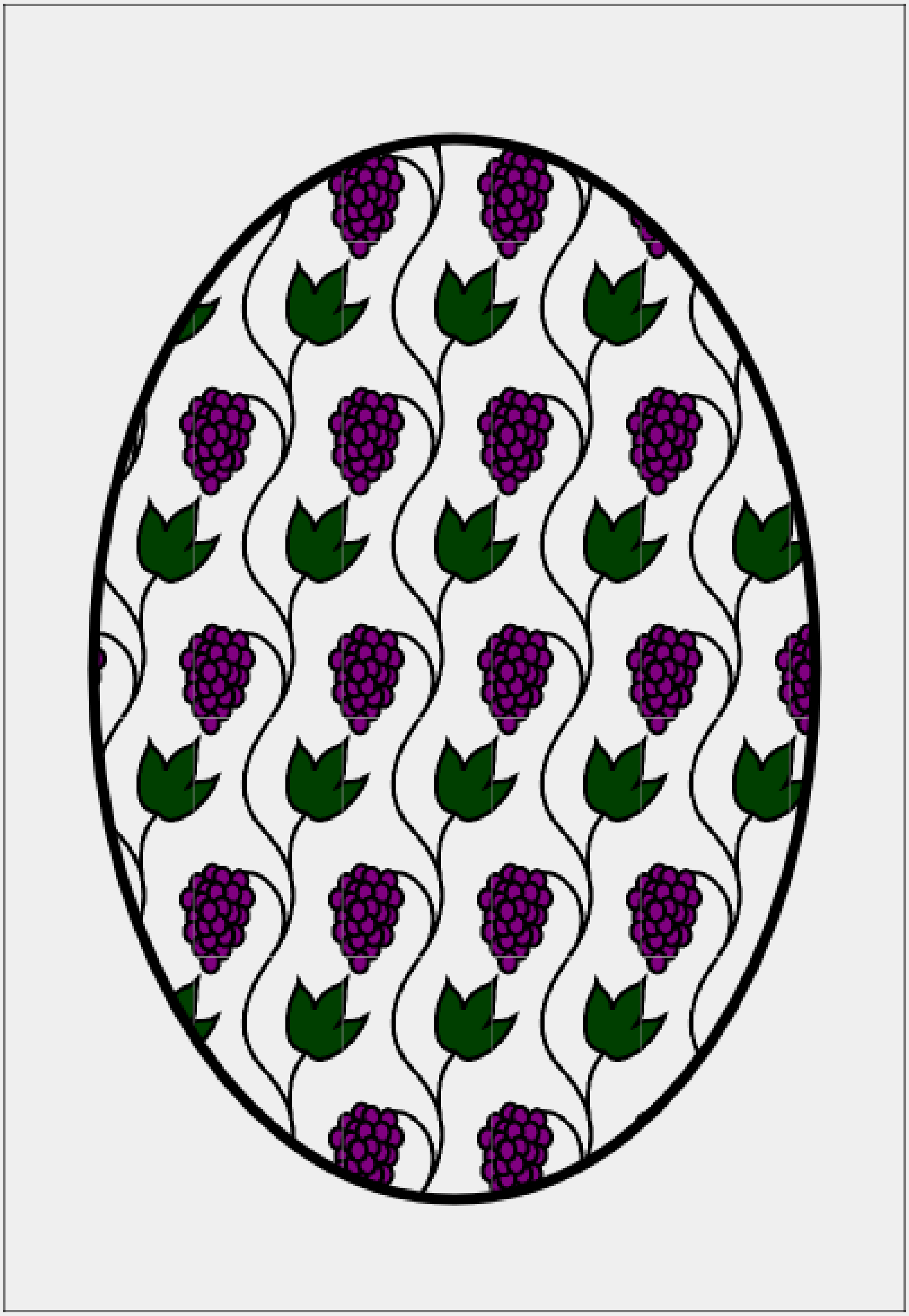
g. The base tile can now be used in a tiling.



A tiling with the clipped base tile.

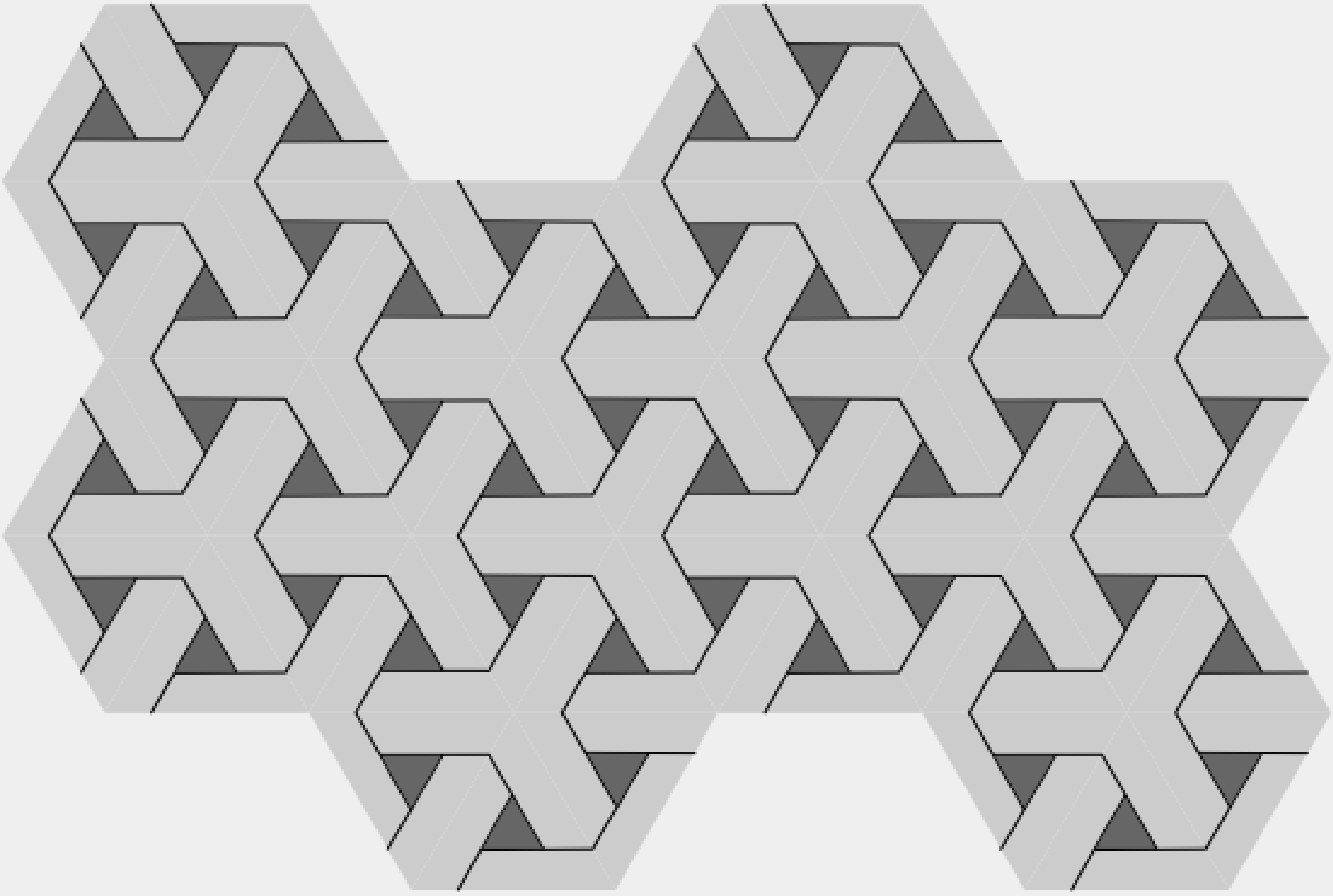
Using the clipped base tile has one main drawback. There may be drawing artifacts where tiles abut. This is due to weaknesses in the drawing algorithms that don't properly handle cases where the tile's edge doesn't align with a screen pixel or other display quanta.

Note that the clipped tile cannot be used directly as a *Pattern*. To create a pattern, first convert the tile into a bitmap (Edit → 📷 Make a Bitmap Copy (**Alt+B**)), then use that bitmap for the pattern. This method may also create drawing artifacts.

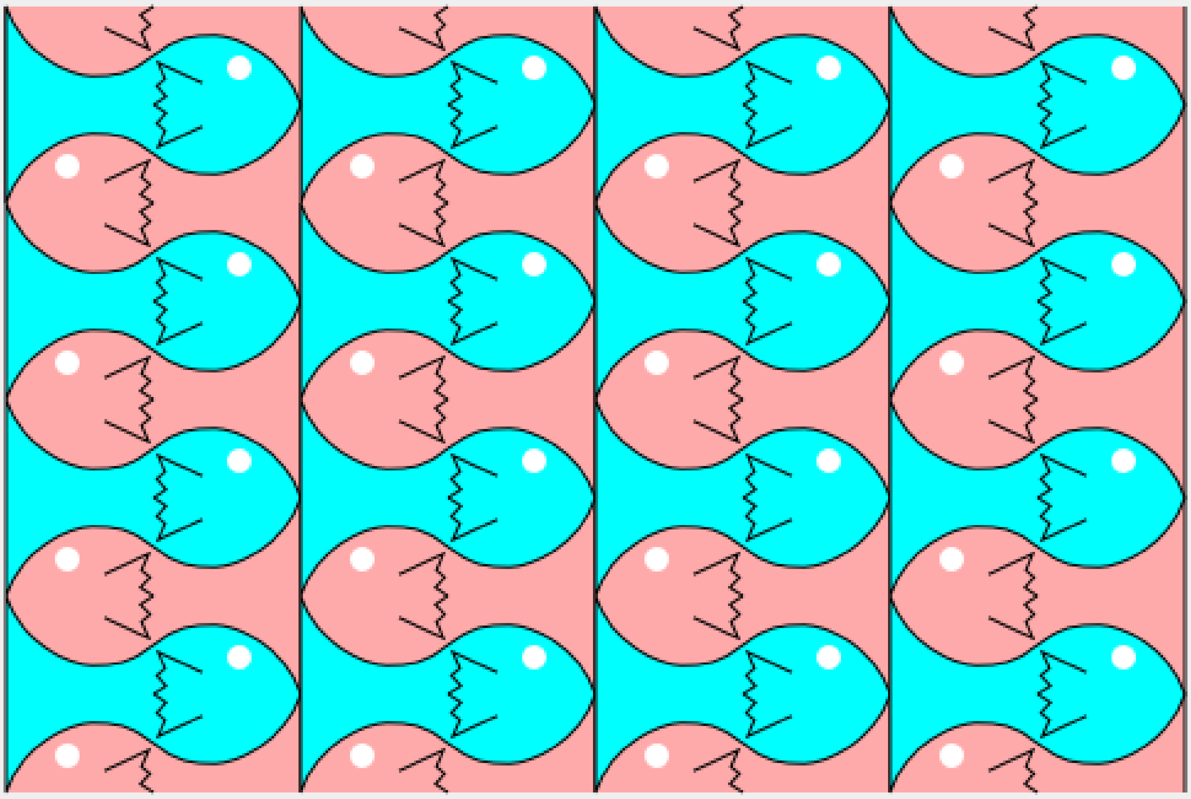


An oval filled with a bitmap pattern. The pattern was created by first turning the base tile into a bitmap. Note the drawing artifacts between the tiles (e.g., *PNG* web version).

This tutorial has covered creating the most simple base tile. More complex base tiles with different symmetries can be just as easily created, as shown next.



A tiling with the "P31M" symmetry.



A "P1" tiling with a base tile created by the "CMM" tiling of a smaller base tile.

A Can of Soup - A Three-Dimension Drawing with Gradients

Table of Contents

A Neon Sign - Animation

Inkscape: Guide to a Vector Drawing Program

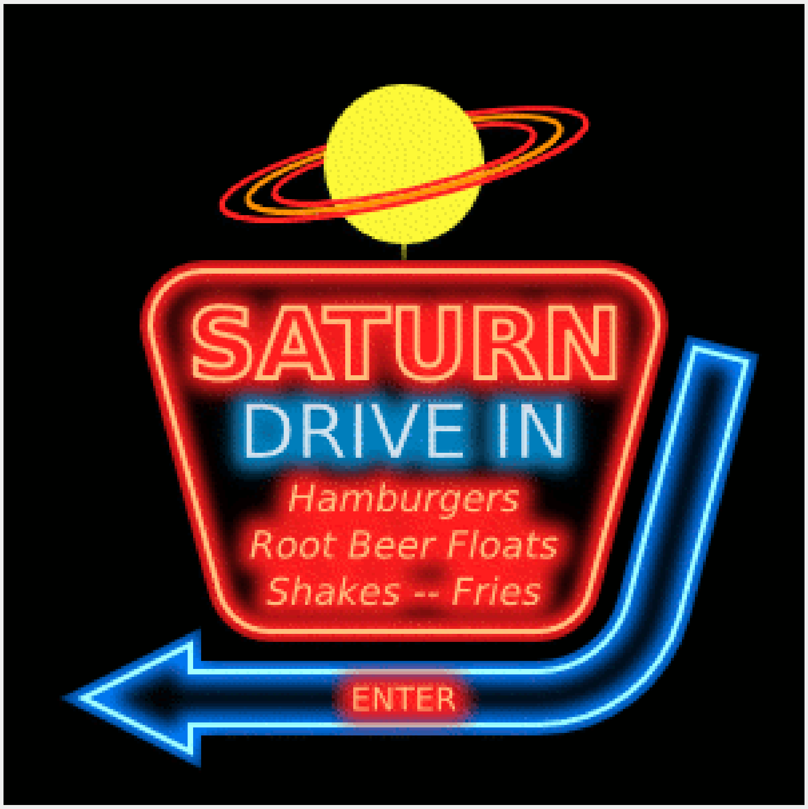
Get the book.

Inkscape » Quick Start » A Neon Sign - Animation

Index

A Neon Sign - Animation

While Inkscape cannot directly handle animation, it is possible to use Inkscape drawings as a starting point for creating animation. This tutorial demonstrates two techniques for creating an animated neon sign. It also discusses a number of issues the artist must consider in creating the animations.



An animated neon sign. (GIF).

An animated neon sign. (SVG).

Several things must be considered in planning an animation. The most critical is how will the animation be displayed? The *SVG* standard includes provisions for natively incorporating animation. Unfortunately, support by *SVG* viewers is very limited. JavaScript or ECMAScript can also be used for animations. This is more widely supported. Web browsers such as Opera and Firefox support JavaScript but the support is not very mature. Firefox 2 and 3 in particular perform poorly with complicated drawings. An alternative method is to use an external program such as Gimp to create an animated *GIF*.[7] In this case, performance is more than adequate but one loses several advantages of using *SVG*, such as scalability.

This tutorial is divided into two parts. The first part covers the creation of an unanimated neon sign and the second part covers animating the sign both via an animated *GIF* and by using JavaScript. Attention is paid to the first part to facilite the animation. Heavy use of the *XML Editor* is made to give the various drawing objects names useful in the animation as well as to adjust some object parameters.

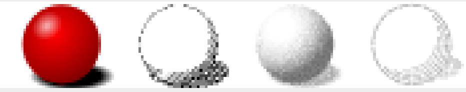
The full tutorial can be found in the PDF version of the book. For more information, check the book's web site.

[7] It is normally recommended to save drawings in the *PNG* format rather than the *GIF* format. Unfortunately, the animated versions of the *PNG* format, *MNG* and *APNG*, have limited support. This leaves the *GIF* format the only real option for animation.

A Vine Design - A Tiling Example

Table of Contents

A Bank Note - Security Features



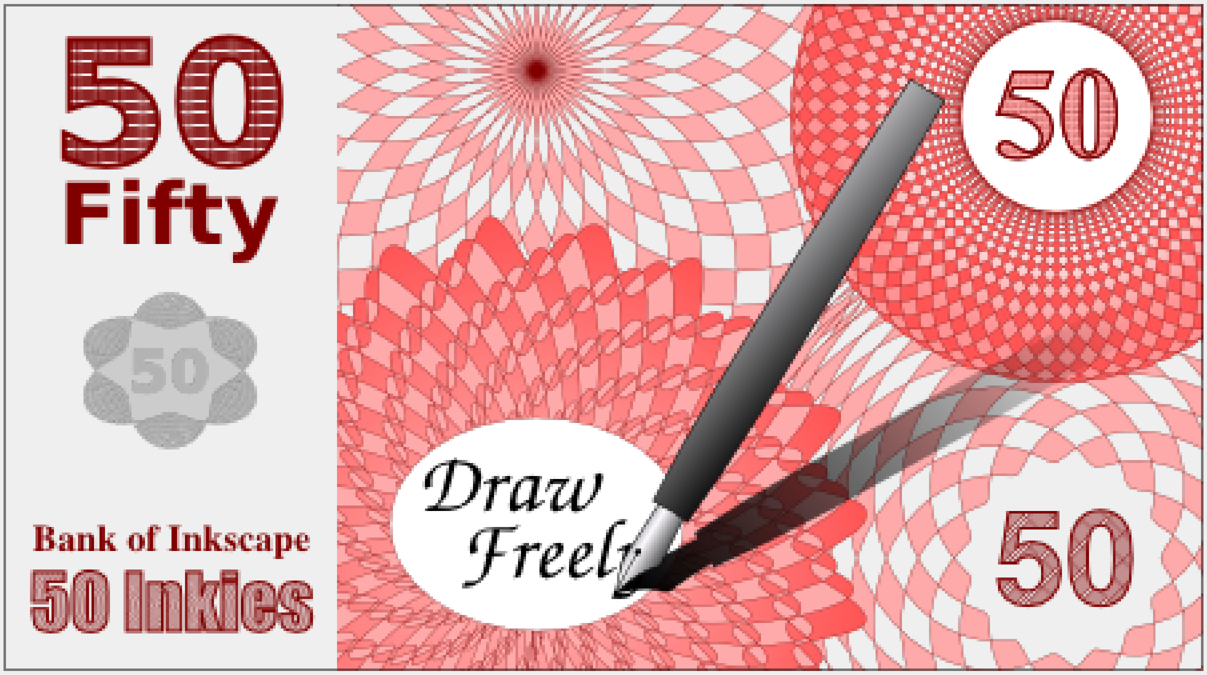
Inkscape » Quick Start » A Bank Note - Security Features

Index

A Bank Note - Security Features

Bank notes are known for their various security features. Most notes include the heavy use of the Intaglio printing where the design is composed of complicated patterns of fine lines. Inkscape has many features that allow one to design attractive bank notes complete with a variety of security features. In this tutorial, we explore some of the tricks while creating a “50 Inkies” bank note.

The bank note is composed of a variety of different types of objects. Most of these objects are independent of each other. Unlike the other tutorials, there is no natural order in creating the drawing. Just pick and choose which features you want and how and where they are placed.



A 50 Inkies bank note.

The full tutorial can be found in the PDF version of the book. For more information, check the book's website.

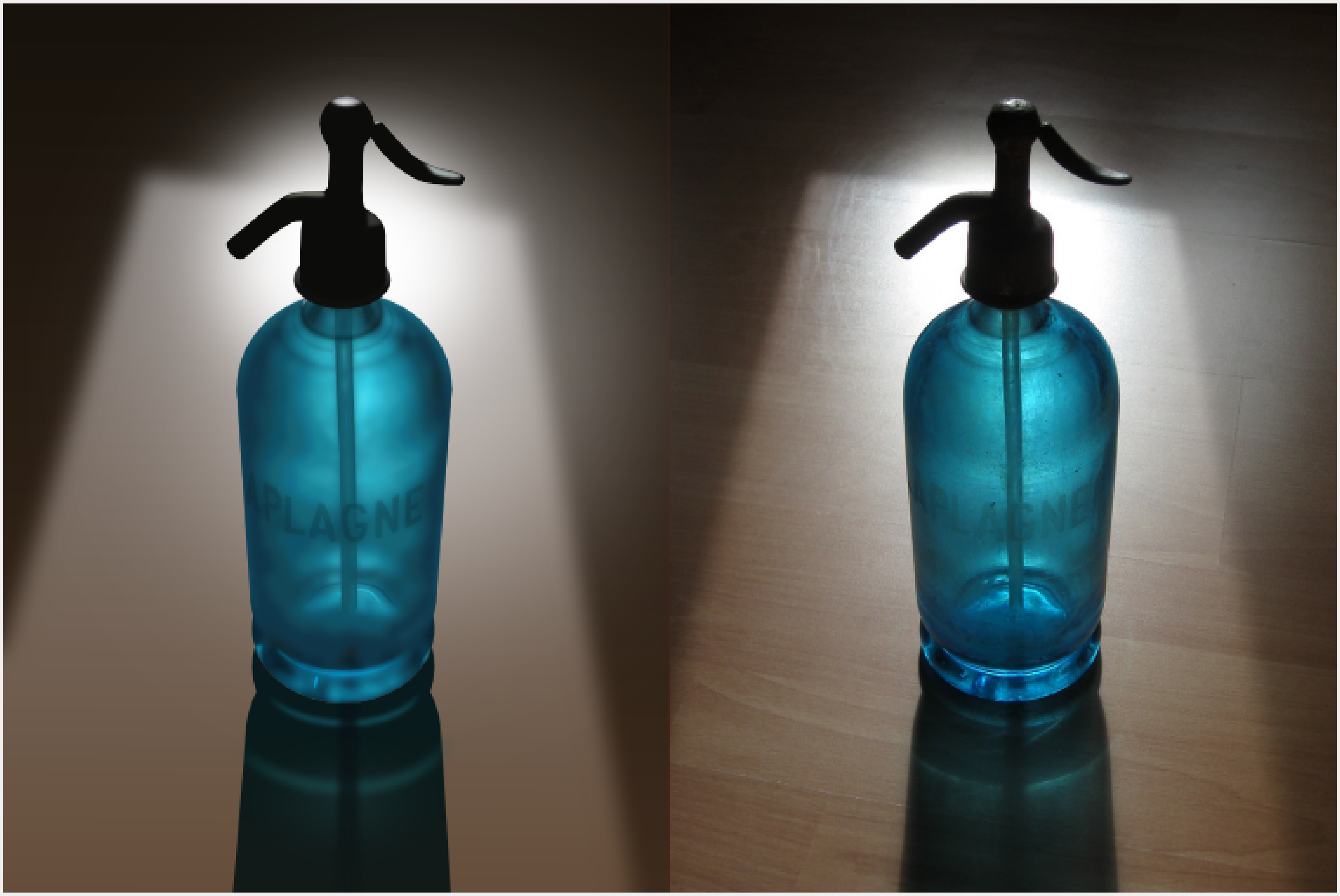
A Neon Sign - Animation
Table of Contents
A Bottle - Photorealism

Get the book.

A Bottle - Photorealism

Inkscape can be used to produce photo-realistic drawings. Inkscape features that are useful for this include: *Gradients*, the *Gaussian Blur* filter, and *Bitmap Tracing*. This tutorial uses all of these to produce a realistic drawing of an old seltzer bottle. The source photograph is available on the book's website.

Our goal is to capture the essence of an antique bottle, not to copy all the fine detail (use a photograph for that). The hand-blown glass is irregular, something that would be hard to reproduce by hand drawing, even with gradients and blurring. To simulate the glass, we'll use the auto-tracing routines built into Inkscape. Tracing has its flaws, though, which we'll need to work around. The handle and background, on the other hand, are prime candidates for *Gradients* and *Blurs*.



A photo-realistic bottle (left), drawn from a photograph (right).

The full tutorial can be found in the PDF version of the book. For more information, check the book's web site.

A Bank Note - Security Features

Table of Contents

Chapter 2. Files

Get the book.

Opening and Saving Files

Inkscape drawings are stored in *SVG* files. Upon starting, Inkscape normally creates a new blank drawing. The parameters used for this drawing can be changed by altering the file `inkscape/templates/default.svg`; see the section called “Custom Templates” in Chapter 20, *Customization*. You can also add your own templates to the template directory.

Depending on how your operating system is set up, a preexisting Inkscape drawing may be opened by clicking on the drawing's icon or by giving the drawing's filename as a command-line argument. Drawings in other formats may be opened the same way if Inkscape supports the format (see next section).

The following commands for working with files are found under the *File* menu:

- File → New: Create a new drawing after Inkscape has been started. The submenu allows you to choose between various sized drawings. Inkscape will automatically give your new drawing a temporary name. The keyboard shortcut **Ctrl**+**N** will open a new file with the default size (template), as will clicking on the icon in the *Command Bar*.
- File → Open... (**Ctrl**+**O**): Open an existing file. Multiple files can be opened at the same time by using **Shift**+**Left Mouse Click** to select more than one file from the file opening dialog. This command can also be used to create a new Inkscape drawing from a file with a non-Inkscape format. The file types that can be opened are the same as those listed in the *Importing Files* section below.

  New in v0.46:

The *Open* dialog includes a folder of Inkscape example drawings under the name *examples*.

- File → Open Recent: Open a recently used file. Choose file from list in submenu.
- File → Revert: Revert a file to remove all changes made in the current Inkscape session.
- File → Save (**Ctrl+S**): Save your drawing to a file.
- File → Save As... (**Shift+Ctrl+S**): Save your drawing to a file with a new name.

  New in v0.46.

  The *Save As* dialog includes a folder for Inkscape templates under the name *templates*. Saving a drawing with the name `default.svg` in this directory will replace the default template (the file opened when **Ctrl**+**N** is used). Saving in this directory under a different name will add the file as a template under the new name in the File → New submenu.
- *New in v0.45.*

  File → Save a Copy... (**Shift+Ctrl+Alt+S**): Save your drawing to a file with a new name but keep the old file name (and type) for future saves.

Chapter 2. Files

Table of Contents

Importing Files

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Files » Importing Files

Index

Importing Files

Inkscape is capable of importing many types of *vector* and *bitmap* graphics files. It can also import text. Some types are handled internally by *GDK* (ani, bmp, gif, ico, jpeg, pcx, png, pnm, ras, tga, tiff, wbmp, xbm, xpm). Other types require additional programs, listed under the relevant entries below. One special feature allows importing images from the Open Clip Art Library, a free source of clip art. This is discussed at the end of this section.

Two methods exist for importing files into an already open document. The first is to use your window manager to drag a file and drop it onto an open Inkscape window. The second is to use the File → Import... (**Ctrl+I**) dialog. These work for both *bitmap* and *SVG* files.

Note

By default *bitmap* images are not stored inside the Inkscape *SVG* file. Only references are stored. If you move your *bitmap* files, the references will be broken. You can embed the images inside the *SVG* file by using the *Embed Images* effect. See the section called “Embed All Images” in Chapter 18, *Effects*.

Warning

> The *SVG* standard only requires renderers to support *PNG*, *JPEG*, and *SVG* formats. The other formats that Inkscape supports internally due to GDK (ani, bmp, ico, pcx, pnm, ras, tga, tiff, wbmp, xbm, xpm) render fine in Inkscape but Inkscape will not embed them in the *SVG* file. Other renderers will most likely not support external files of these types referenced from within an *SVG* file. Your safest bet is to use another program such as Gimp to convert these formats to *JPEG* or *PNG*.

File types supported (a * indicates the author has not tested the file import):

- *Changed for v0.46.*

  .ai (Adobe Illustrator) Opens version 9.0 and later files (based on PDF). See PDF entry below. Older versions (based on PostScript) can be opened by selecting the PostScript or EPS file type from the drop-down menu. On Linux, one can use the **`file`** command to check the version.
- .ai.svg (Adobe Illustrator *SVG*) Strips the input of everything in Adobe Illustrator *Name Spaces*, leaving the file as pure *SVG*.
- .ani (Animation Cursor)
- .bmp (Windows Bitmap)
- .cur (Windows cursor)
- *New in v0.46.*

  .cdr (CoralDraw) Works only on Linux at the moment. Requires UniConverter to be installed. Text objects not converted. File versions 7 to X4 supported by UniConverter 1.1.1.

  Note for Fedora Users: The executable `uniconv` was renamed to `uniconvertor` to avoid a naming conflict. You can add a “symbolic link” from `/usr/bin/uniconv` to `/usr/bin/uniconvertor` (if you don't already have a binary with that name) or you can edit the files `cdr_input.inx` and `cdr2svg.sh` in the `share/inkscape/extensions` directory.
- .dia (Dia) Requires Dia to be installed.
- .dxf (AutoCAD) Requires `dxf2svg` to be installed.
- .eps (Encapsulated PostScript) Requires pstoedit to be installed. Clipping is lost.
- .fig (XFig) Requires xfig (fig2dev) to be installed.
- .ggr (Gimp gradient) Requires Gimp. Imported *Gradients* will appear in the *Gradients* menu. Works only for linear *Gradients*.
- .gif (GIF: Graphic Interchange Format)
- .jpg, .jpe, .jpeg (JPEG: Joint Photographic Experts Group)

- .ico (Windows icon)
- .pcx (PC Paintbrush Bitmap Format)
- .png (PNG: Portable Network Graphic)
- .pbm, .pgm, .pnm, .ppm (PNM, Portable Anymap)
- *New in v0.46.*

  .pdf (Adobe Portable Document Format) Supported natively through the *poppler* library. Also supports .ai (Adobe Illustrator) version 9.0 and later files. A dialog will appear in which one can specify which page of a multipage file should be imported as well as a clip region. *Gradient meshes* are converted to groups of small tiles. Text is imported with manual kerning. To make editing easier, manual kerning can be removed via the Text → Remove Manual Kerns command.
- .ps (PostScript) Requires pstoedit to be installed. Clipping is lost.
- .ras (Sun Raster)*
- .sk (Sketch/Skencil) Requires Skencil to be installed. Special shape information (e.g., rectangles) is not preserved.
- .svg (SVG)
- .svgz (SVG Compressed)
- .tga, .targa (Truevision Advanced Raster Graphics Adapter) Common in video games.
- .tif, tiff (TIFF: Tagged Image Format)
- .txt (Text), Requires Perl SVG.pm module. Text is imported as a group of regular text objects, one for each line.
- .wbmp (Wireless Application Protocol Bitmap Format)
- *Updated in v0.46 (new method that works).*

  .wmf (Windows Meta File) Requires UniConverter to be installed.

  Note for Fedora Users: The executable `uniconv` was renamed to `uniconvertor` to avoid a naming conflict. You can add a “symbolic link” from `/usr/bin/uniconv` to `/usr/bin/uniconvertor` (if you don't already have a binary with that name) or you can edit the files `wmf_input.inx` and `cdr2svg.sh` in the `share/inkscape/extensions` directory.

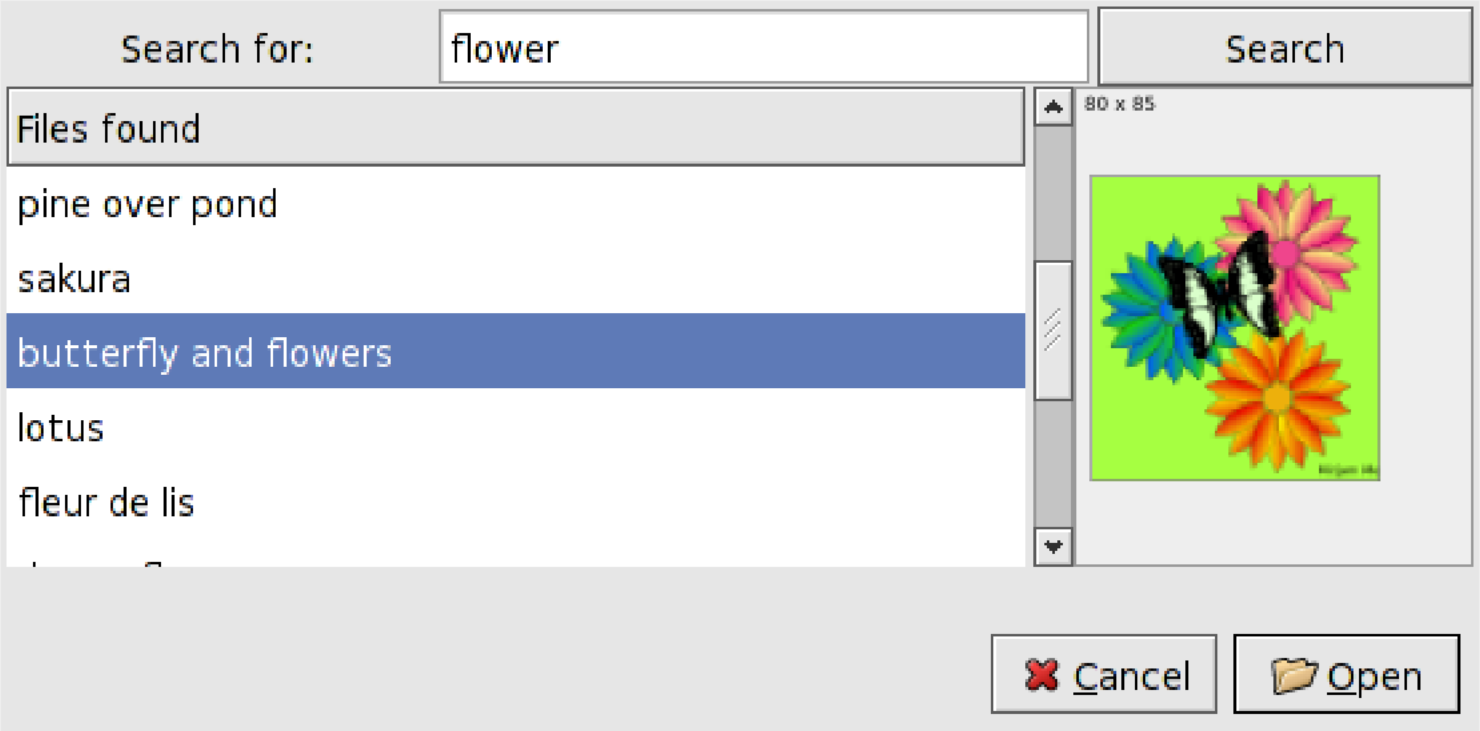
- *New in v0.46.*

  .xaml (Microsoft Application eXtensible Markup Language).*
- .xbm (X-BitMap).
- .xpm (X-Pixmap)

Open Clip Art Library

New in v0.46.

The command File → Import From Open Clip Art Library opens a search dialog that connects to the Open Clip Art Library website, a source of free clip art. The dialog allows searching for drawings that match descriptive words. Previews of files are shown in a window on the right side of the dialog.



Search for: flower

Files found
pine over pond
sakura
butterfly and flowers
lotus
fleur de lis

Import From Open Clip Art Library dialog. An example of searching for a flower is shown.

Unfortunately, at the moment, only *PNG* files can be imported even if a matching *SVG* files exist. You may have to set the server name (`openclipart.org`) in the *Import/Export* section of the *Inkscape Preferences* dialog. In the future, export to the library will also be supported.

Opening and Saving Files

Table of Contents

Exporting Files

Get the book.

Inkscape » Files » Exporting Files

← Index →

Exporting Files

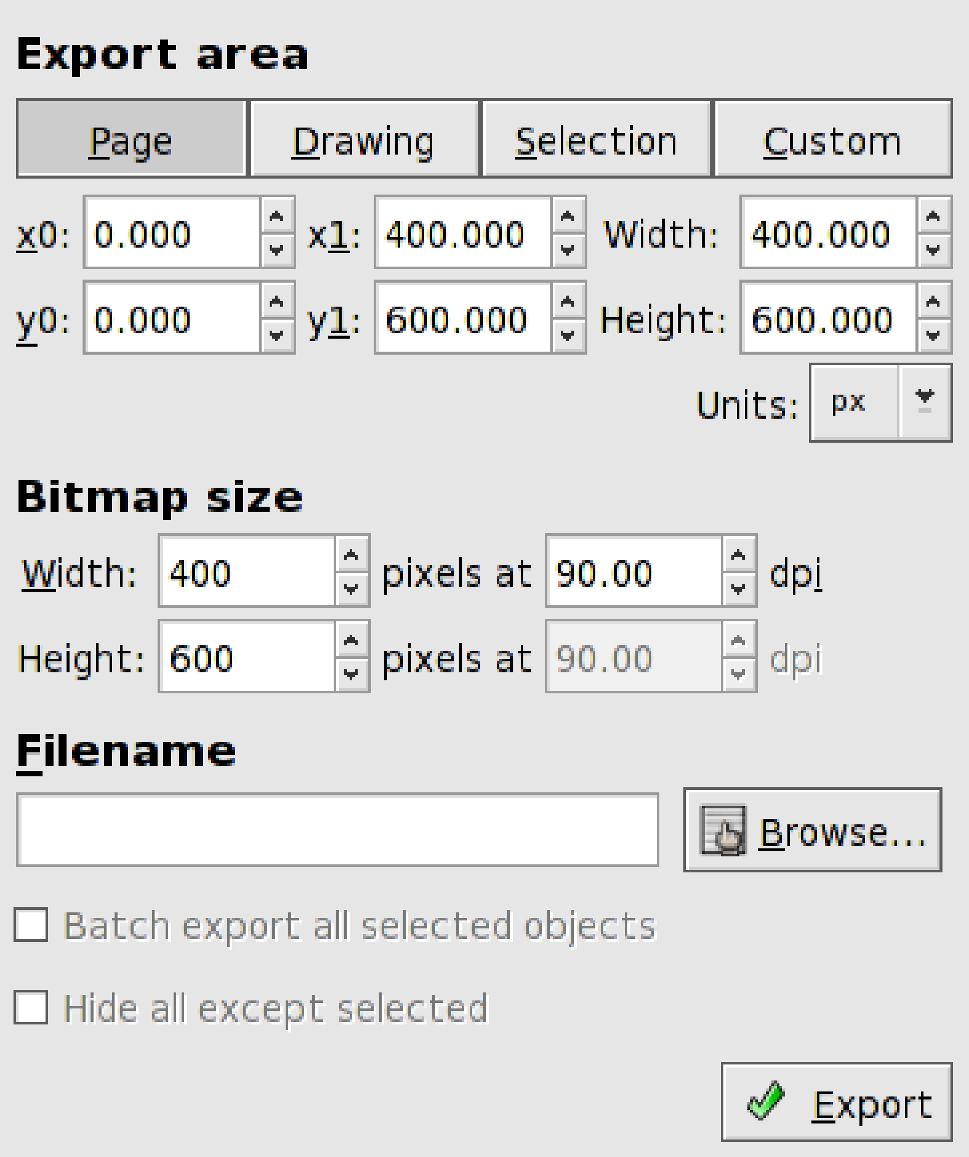
Inkscape is capable of exporting drawings to several types of *vector* and *bitmap* graphics files. Exporting methods are divided between exporting *PNG* (Portable Network Graphics) files and exporting to all other file formats. The *PNG* graphics standard is a patent unencumbered standard that is supported natively by all major web browsers and graphics programs.

> **Tip**
>
> You can set the page size to match the *bounding box* of a selection by clicking on the *Fit page to selection* button in the *Document Properties* dialog. This is useful for setting the page size *after* creating an illustration.

Exporting PNG (Portable Network Graphic) Files

Exporting a *PNG* file is done through the *Export Bitmap* dialog (File → Export Bitmap... (**Shift**+**Ctrl**+**E**)).

> **Note**
>
> This dialog *ONLY* exports *PNG* files, regardless of what file extension you use.



Export area

x0: 0.000 x1: 400.000 Width: 400.000

y0: 0.000 y1: 600.000 Height: 600.000

Units: px

Bitmap size

Width: 400 pixels at 90.00 dpi

Height: 600 pixels at 90.00 dpi

Filename

□ Batch export all selected objects

□ Hide all except selected

Export Bitmap dialog.

At the top of the dialog are four buttons that give an initial setting as to what area of the drawing should be exported (except for the *Custom* button, which does nothing). Once given an initial setting, the area can be adjusted.

- *Page*: Export the area enclosed by the page. The default filename is the *SVG* filename with the “svg” extension replaced by “png”.
- *Drawing*: Export all objects in drawing (including those outside the page). The initial export area is the *bounding box* of all the objects. The default filename is the *SVG* filename with the “svg” extension replaced by “png”.
- *Selection*: Export the region enclosed by the *bounding box* surrounding the selected objects. The default filename is the internal name of the last object selected.
- *Custom*: Export the area defined in the entry boxes.

The area to be exported can be modified by the entry boxes in the *Export area* part of the dialog. The units can be changed in the *Units* pull-down menu.

In the *Bitmap size* section of the dialog, the size of the exported bitmap can be defined. Inkscape uses a default conversion of 90 dpi. Only the *Width* and width *dpi* can be changed; the *Height* will scale to preserve the height to width aspect ratio of the drawing.

In the *Filename* section, a filename can be entered or one can open a dialog to browse for a filename.

New in v0.46:

There are two options at the bottom of the dialog. The first one, *Batch export n selected objects*, is available if more than one object is selected. If checked, each of the selected objects will be exported into its own file. The name of the file will be the *id* name for the object (see XML Editor dialog) unless an object has been previously saved with a specific filename, in which case the previous filename will be used (the filename is stored under the *export-filename* attribute). The resolution will be 90 dpi unless previously saved with a different dpi. Note: Batch export will overwrite files without warning.

The second option, *Hide all except selected*, is available if one or more objects are selected. If checked all unselected objects will be hidden during export.

The following figure shows a test file exported to a PNG. The file includes basic shapes, text, and a linear Gradient.

Export Test

Export test: PNG.

Tip

To slice a drawing into pieces for use on the web, one can create an array of hidden rectangles (no *Stroke* or *Fill*) in a separate “export” *Layer*. Save each rectangle one time to define the export filename. Then when it is time to export the drawing, go to the “export” layer, use the command Edit → Select All (**Ctrl+A**) to select all the rectangles in the *Layer*, then do a batch export.

Tip

The default window size for the dialog may be too narrow to view the output filename including the directory path. Just widen the window to see the entire path.

Warning

Inkscape has rendering problems when two objects touch each other along a common border. This is often seen when a *Pattern* is used for a *Fill*. Firefox, Opera, and Batik do not have this problem. Batik can be used to produce a high-quality *PNG*, *JPEG*, or *TIFF* file.

Exporting Other File Types

Exporting to a type other than a *PNG* is done through the *Save As* dialog (File → Save As... (**Shift+Ctrl+S**)). The file type is chosen from a pull-down menu. Additional programs, listed below, are required for many of the file exports.

Tip

If exporting fails to work, check the `extension-errors.log` (`~/.inkscape/extension-errors.log` with Linux) for error messages. Any missing programs (dependencies) will be listed.

Note

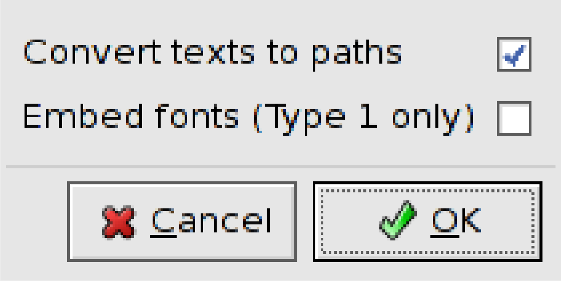
Transparency (the ability to partially see one object through another object) is lost in several of the export types. Text may also have to be converted to paths for satisfactory results in some cases.

File types supported (a * indicates the author has not tested the file export):

- .ai (Adobe Illustrator). Requires *ghostscript* to be installed. The Adobe Illustrator (version 8.0) file format is a modified version of the EPS level 2 format. Files saved in this format can be displayed in many PostScript viewers (you may have to change the “.ai” file tag to “.ps”). Bitmaps are not exported and *transparency* is lost.

  > **Tip**
  >
  > Recent versions of Illustrator can import *SVG* files directly. This will most likely give better results than exporting to an .ai file from Inkscape.

  There is one option that determines if the text is converted to a path or is left as a PostScript text object. There is no guarantee with the latter choice that the printing will be done with the correct font if fonts are not embedded in the file.

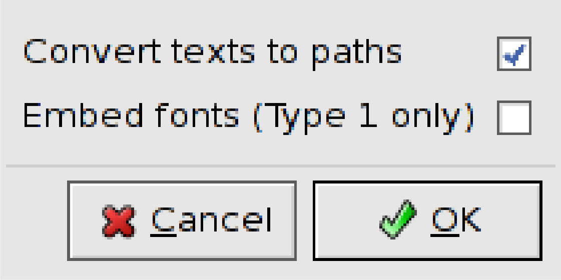
  Convert texts to paths ☑

  Embed fonts (Type 1 only) □

  Adobe Illustrator output dialog.

- .dxf (AutoCAD). All attributes lost. Requires Pstoedit to be installed.

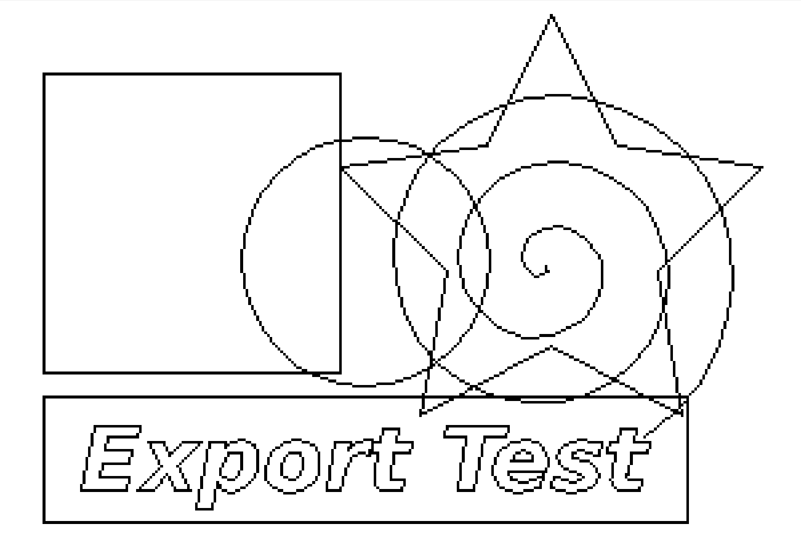
  Two options allow text to be saved as a path or embedded (Type 1 only). The latter option seems to be ignored.

  Convert texts to paths ☑

  Embed fonts (Type 1 only) □

  AutoCAD DXF output dialog.



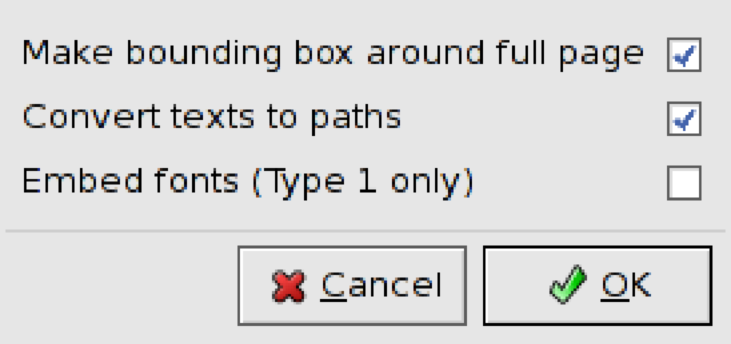
Export test: DXF. As read in by QCad.

A simplified form of DXF export for *desktop cutting plotters* is also supported (labeled “Desktop Cutting Plotter”). The form supports cutters from Wishblade and Craftrobo.

- *New in v0.45.*

  .emf (Enhanced Meta File)*. Only works with Windows. Only exports [*Strokes*] and [*Fills*] with solid colors.
- .eps, .epsi (Encapsulated PostScript, Encapsulated PostScript Interchange). *Transparency* is lost.

  There are three options. The first determines if the *BoundingBox* in the EPS file is set to the *bounding box* of the objects in the drawing or to the dimensions of the page. The second determines if the text is converted to a path or is left as a PostScript text object. The latter allows easy editing of the text in the EPS file but if the font is not embedded in the file, there is no assurance that the printing will be done with the correct font. The third allows embedding fonts in the file. This only works with Type 1 fonts.

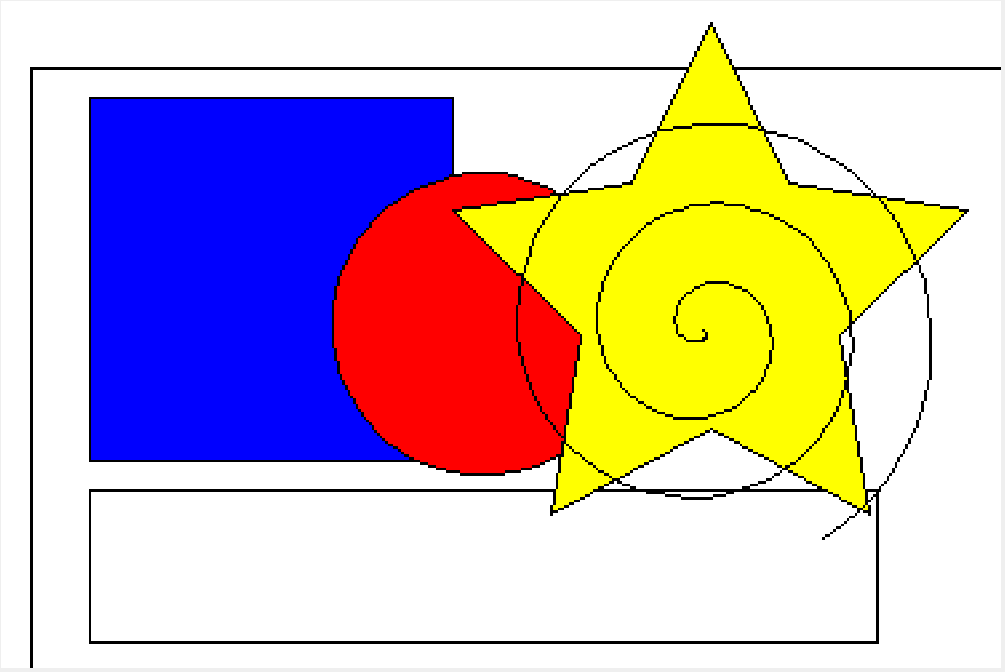

Make bounding box around full page ☑
Convert texts to paths ☑
Embed fonts (Type 1 only) □

Encapsulated PostScript output dialog.

Export Test

Export test: EPS and PostScript export yields the same image.

- .gpl (Gimp Palette). Saves a list of colors used in the drawing to a Gimp Palette file. The file can then be moved to the Inkscape palette directory (`share/inkscape/palettes/`) and the palette can be selected in the *Palette* bar or the *Swatches* dialog. Requires PyXML module.
- .odg (Open Document Graphic). Limited support, better support planned.[8] Currently only export of text, shapes, and solid fill and strokes is supported.



Export test: ODG.

- *Updated for v0.46.*

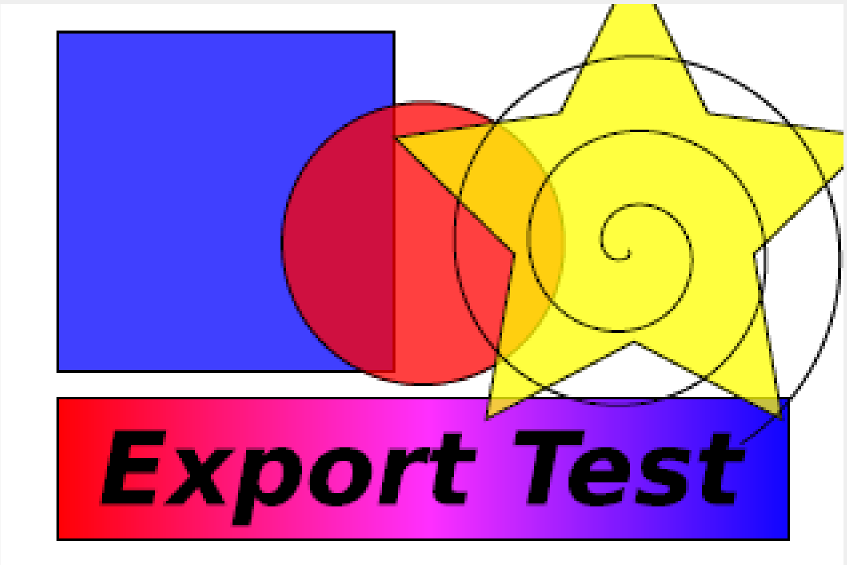
  .pdf (PDF). As of v0.46, a Cairo-based exporter has replaced the old PDF exporter. It is recommended that at least *cairo 1.5.2* be used. Patterns, clipping, and masking are not supported. The methods to produce PDF found under the *Print* dialog may produce better output.

Warning

Some versions of Evince, a popular Linux Postscript and PDF viewer, do not show gradients with *transparency*; use Acroread instead.

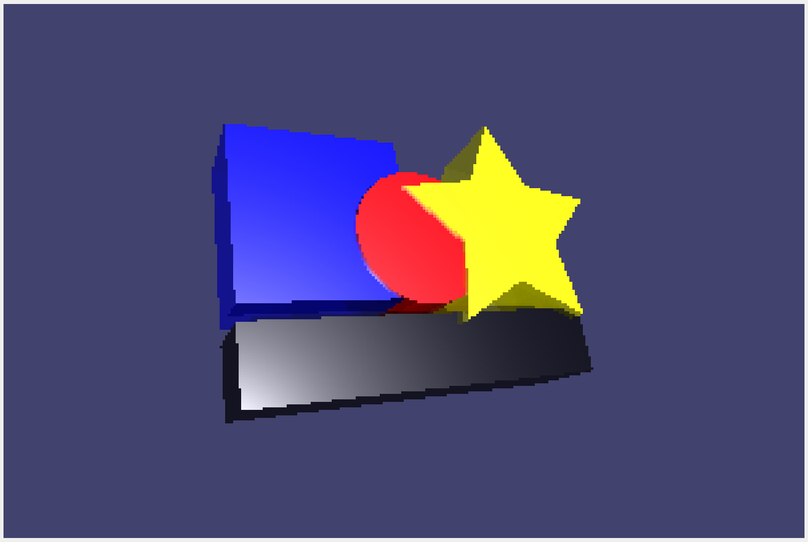
Note

Export to PDF prior to v0.46 produced uncompressed PDF files (which is legal according to the standard). Some operating systems and display software may complain.



Export test: PDF.

- .pov (PovRay). Saves shapes and text, with color and transparency, as *prism* objects. Stroke is not saved. To render a drawing, a camera with lights, etc. must be defined. See `share/inkscape/examples/istest.pov` for an example.



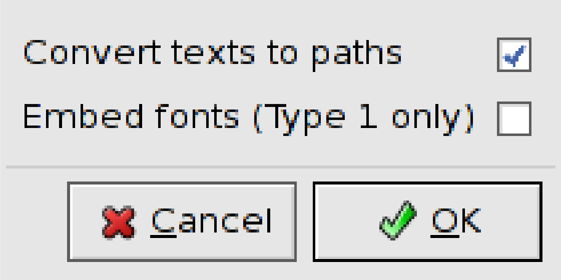
Export test: POV.

- *Updated for v0.46 (Cairo method).*

  .ps (PostScript). Two separate methods are provided: The older built-in method (labeled “PostScript”) and the newer Cairo-based method (labeled “PostScript via Cairo”). The methods to produce PostScript found under the *Print* dialog may produce better output.

  PostScript: Transparency is lost. Clipping, masking, and patterns are not supported.

  There are two options. The first determines if the text is converted to a path or is left as a PostScript text object. With the latter choice, the text is easily editable within the PostScript file. However, there is no guarantee that the printing will be done with the correct fonts if the fonts are not embedded in the PostScript file. The second option allows fonts to be embedded. This only works with Type 1 fonts.



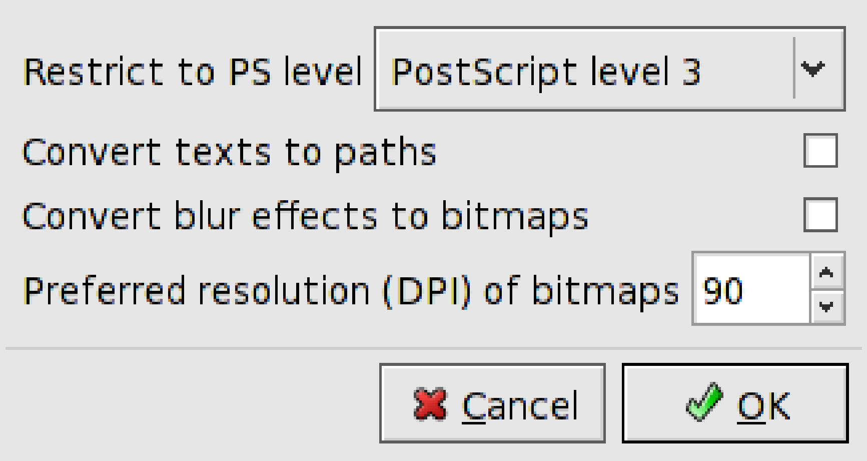
Convert texts to paths ☑

Embed fonts (Type 1 only) □

PostScript output dialog.

  PostScript via Cairo: Patterns and clipping rendered incorrectly. Problems with dashed lines. Masking not supported. Relies on bitmap fallbacks more often than PDF Cairo output.

  Options exist to select the PostScript level (either 2 or 3), if text should be converted to paths, if blurs made with the *Gaussian Blur* filter should be converted to bitmaps, and the preferred resolution of fallback bitmaps. Not converting blurs to bitmaps may yield an empty file.



Restrict to PS level: PostScript level 3

Convert texts to paths □

Convert blur effects to bitmaps □

Preferred resolution (DPI) of bitmaps: 90

PostScript output dialog for Cairo method.

- .tex (LaTeX with PSTricks macros). Requires the PSTricks package to be installed, which is normally included with the LaTeX package.

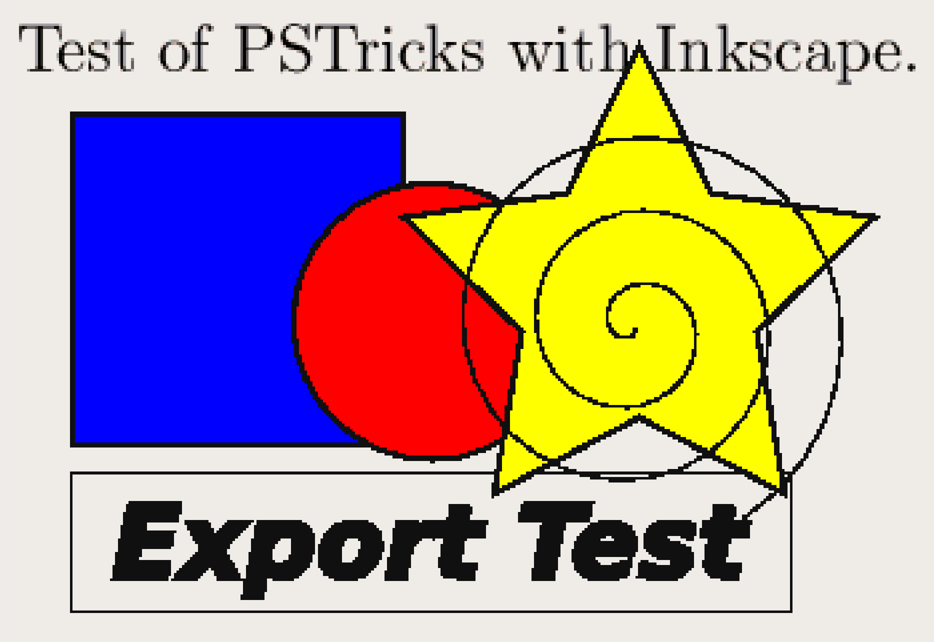
  Here is an example of a LaTeX file to use with Inkscapes tex output saved in the file `my_inkscape_test.tex`.

```
\documentclass[12pt]{article}
\usepackage{pstricks}
\begin{document}

Test of PSTricks with Inkscape.

\input{FileExportTest.tex}

\end{document}
```



Export test: LaTeX.

Tip

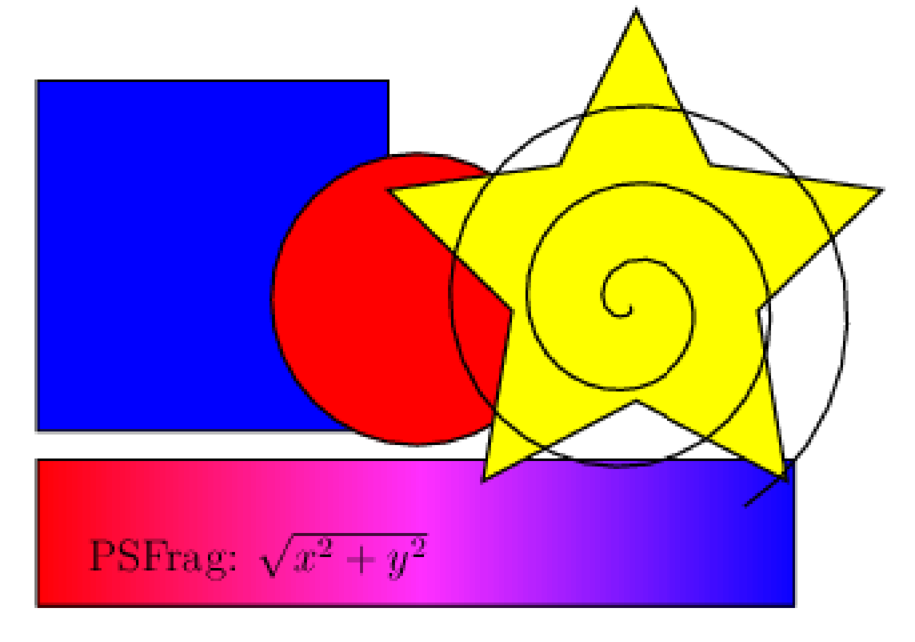
If you need equations in your figures, there are two methods available. The first is to use the *Latex Formula extension*.

Alternatively, you can use the *psfrag* LaTeX package. To use the latter, put a place holders for your equation in a text box. Then export as an EPS file (without converting the text to a path or using manual kerning!). The psfrag package can then be used to replace the placeholder with a latex expression. Putting text on a diagonal is possible.

```
\documentclass[12pt]{article}
\usepackage{graphicx}
\usepackage{psfrag}
\begin{document}

\psfrag{Export Test}{PSFrag: $\sqrt{x^2+y^2}$}
\includegraphics{FileExportTest2.eps}

\end{document}
```



Export test: LaTeX using an EPS file with psfrag. The phrase "Export Test" has been replaced by a LaTeX expression.

Inkscape can also be used to design LaTeX presentation styles. See Latex Presentation Designer.

- .svg, .svgz (*SVG*, *SVG* compressed). Two options are available: Plain *SVG* and *SVG* with Inkscape extensions (for storage of Inkscape meta data). In theory, any program that reads *SVG* files should ignore the Inkscape extensions.
- .xcf (Gimp). All top-level layers and objects are exported as separate layers in the *PNG* format and then inserted as layers into the Gimp native *XCF* format. Requires Python, PyXML, and Gimp 2.2 or higher. Both Gimp and Inkscape (v0.44 or greater) must be in the executable path. This export is not implemented on Windows.

Export Test

Export test: XCF.

- *New in v0.46.*

  .xaml (Microsoft Application eXtensible Markup Language)*.
- .zip (Compressed Inkscape *SVG* with Media). This option will save the drawing as an Inkscape *SVG* file and then package that along with all included (linked) graphics files as a *zip* package. The resulting file cannot be read directly by Inkscape. However, when the file is uncompressed, all the included graphics should be placed where Inkscape can find them when the *SVG* file is opened.

[8] The book discloses forward-looking statements on Inkscape features. These forward-looking statements are based on many assumptions including developer availability and customer demand. While the author believes that these forward-looking statements to be reasonable, the reader should not construe these statements to be a contract of any kind.

Importing Files

Table of Contents

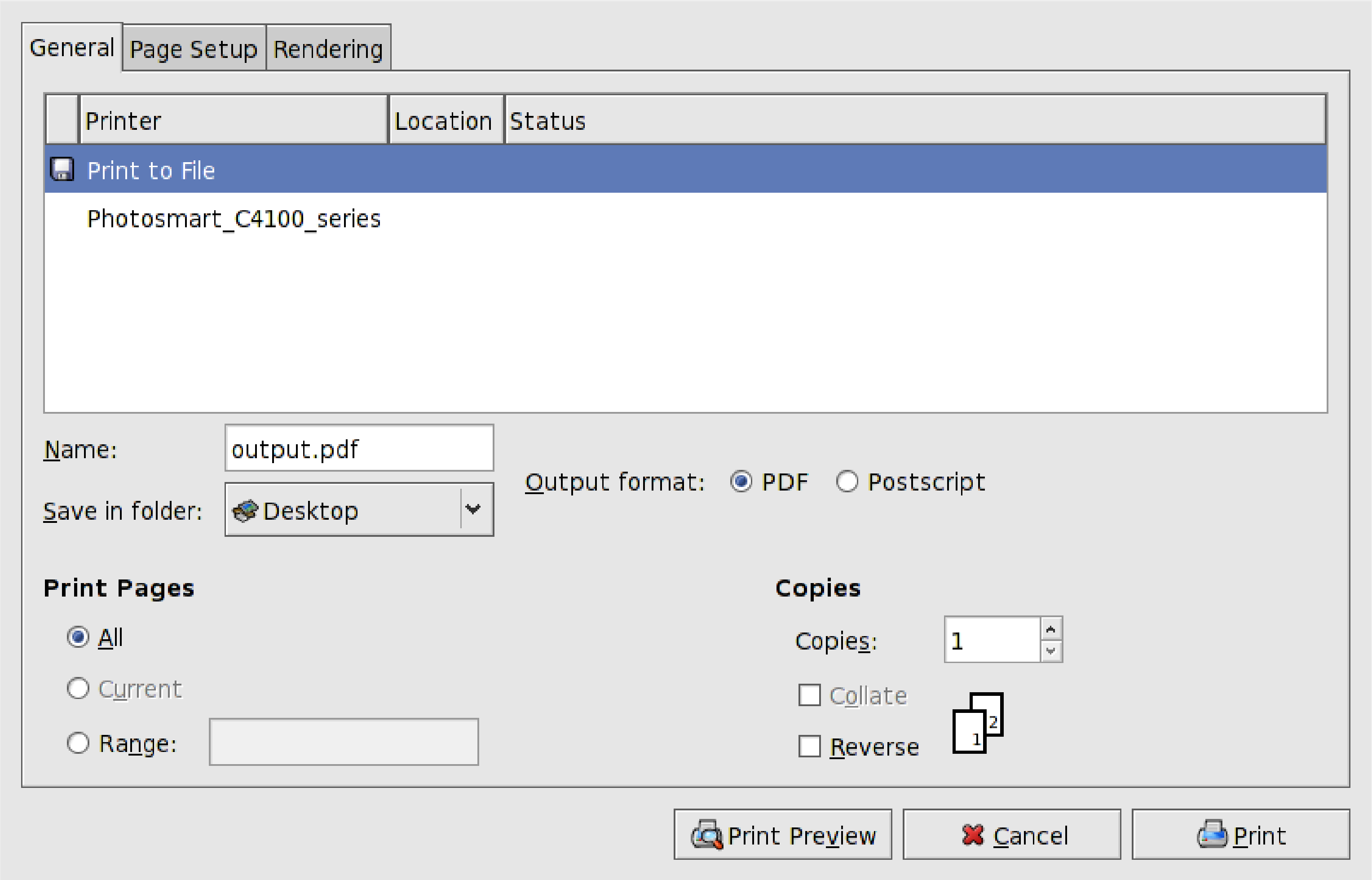
Printing Files

Get the book.

Printing Files

Updated for v0.46

Printing your drawing can be done through the File → Print... (**Ctrl+P**) dialog. As of v0.46, the standard GTK print dialog is used. This allows printing to any PostScript-capable printer as well as to either a PostScript or PDF file. Printing uses the Cairo-based routines (although not the one used to *Save* a PDF file!). The PostScript back-end makes heavy use of rasterizing the image.



Printer	Location	Status
Print to File		
Photosmart_C4100_series		

Name: output.pdf

Save in folder: Desktop

Output format: ● PDF ○ Postscript

Print Pages

● All

○ Current

○ Range:

Copies

Copies: 1

□ Collate

□ Reverse

Print dialog.

There are two options for rendering output found under the *Rendering* tab. The first *Vector* uses vector operators. the drawing is scalable but some properties are lost. Patterns, masks, clips, and dashed lines are rendered incorrectly (Bug).

The second option is *Bitmap* which rasterizes the output. There is an option to set the DPI for the output but this does not appear to have any effect (Bug). All drawing elements are rendered correctly but with poor resolution.

Note

An Inkscape PostScript file can be converted to an Encapsulated PostScript file (EPS) by removing the last line in the file (delete “showpage”). A “BoundingBox” line is already included in the PS file.

Exporting Files

Table of Contents

Vacuuming Files

Inkscape » Files » Vacuuming Files

⬅ Index ➡

Vacuuming Files

The command File → Vacuum Defs removes unused definitions from the *<defs>* section of the *SVG* file. This includes things like unused gradients, patterns, and markers.

> **Note**
>
> Due to the mechanism that Inkscape uses to keep track of which definitions are in use, it may be necessary to close a file and reopen it before this command will remove *all* unused items.

⬅ Printing Files

⬆ Table of Contents

➡ Chapter 3. Changing the View

Inkscape » Changing the View » Panning the Canvas

Index

Panning the Canvas

The region of the canvas that is visible can be changed via

- *Scroll Bars*: Perhaps the easiest method. The visibility of the scroll bars can be toggled using **Ctrl**+**B**.
- **Mouse Wheel**: pan up and down. **Shift**+**Mouse Wheel**: pan sideways.

  New in v0.46:

  By checking the *Mouse wheel zooms by default* button under the *Scrolling* section of the *Inkscape Preferences* dialog, the function of the **Mouse Wheel** changes to zooming while the normal **Ctrl**+**Mouse Wheel** changes to panning up and down. This is the normal behavior in Corel Draw and AutoCAD.
- **Middle Mouse Drag**. With a two-button mouse, use **Shift**+**Right Mouse Drag** or **Ctrl**+**Right Mouse Drag**.
- **Ctrl**+**Arrow** keys.
- *New in v0.46.*

  Space Bar+**Left Mouse Drag** will pan the canvas if the *Left mouse button pans when Space is pressed* option is checked in the *Scrolling* section of the *Inkscape Preferences* dialog. Turning this option on prevents using the **Space Bar** to toggle between tools.

Chapter 3. Changing the View Table of Contents Zooming the Canvas

Get the book.

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Changing the View » Zooming the Canvas

Index

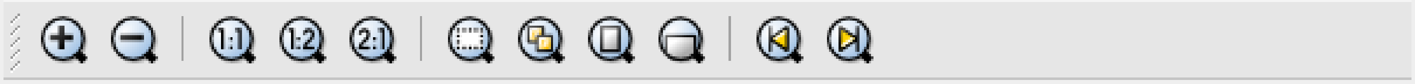
Zooming the Canvas

The canvas can be zoomed in and out in many ways. For many of the options the zoom changes by a default factor of 1.41 (repeat twice to zoom by a factor two). This factor can be changed in the *Steps* section of the *Inkscape Preferences* dialog.

- **+** or **=** keys to zoom in, **-** key to zoom out. Zoom is around center of viewable canvas.
- **Ctrl**+**Middle Mouse Click** or **Ctrl**+**Right Mouse Click** to zoom in, **Shift**+**Middle Mouse Click** or **Shift**+**Right Mouse Click** to zoom out. Zoom in is around cursor position.
- **Shift**+**Middle Mouse Drag** to zoom in on area within *rubber-band*.
- **Ctrl**+**Mouse Wheel** to zoom in and out. Very convenient. The cursor position is the center of the zoom.

  New in v0.46:

  By checking the *Mouse wheel zooms by default* button under the *Scrolling* section of the *Inkscape Preferences* dialog the function of the **Mouse Wheel** changes to zooming while the normal **Ctrl**+**Mouse Wheel** changes to panning up and down. This is the normal behavior in Corel Draw and AutoCAD.
- Zoom section of the *Status Bar*. This is the best way to select a precise zoom level. One can activate the entry box via the keyboard shortcut **Alt**+**Z**.
- *Zoom Tool* (**F3**) from the *Tool Box*. Click on the canvas with the *Zoom Tool* to zoom in around the cursor. Use **Shift** click or click with the **Right Mouse** button to zoom out.
- *Zoom Tool*: *Tool Controls* and under the View → Zoom menu. When the *Zoom Tool* is selected, the *Tool Controls* will show a series of icons representing different zoom options. The View → Zoom menu has the same options.



The *Zoom Tool*-*Tool Controls*.

- (**+**): Zoom in.
- (**-**): Zoom out.
- (**1**): Zoom to 100% of drawing size (one drawing pixel equal one screen pixel).
- (**2**): Zoom to 50% of drawing size.
- Zoom to 200% of drawing size.
- (**3**): Zoom to show the selected object(s).
- (**4**): Zoom to the drawing size (show all the drawn objects which may be smaller or larger than the page).
- (**5**): Zoom to show the entire page.
- (**6** or **Ctrl**+**E**): Zoom to the width of the page.
- (**`**): Zoom to previous zoom level.
- (**Shift**+**`**): Zoom to next setting (after zooming to a previous setting).

Note the two commands (and) that allow one to cycle backward and forward through the previously used zoom levels.

Panning the Canvas

Table of Contents

Miscellaneous View Commands

Inkscape » Changing the View » Miscellaneous View Commands

Index

Miscellaneous View Commands

Hide/Show

The *Hide/Show* submenu can be used to toggle on and off various parts of the graphical user interface including the *Palette*.

Hide/Show Dialogs

Inkscape dialogs can be hidden and unhidden with the View → Show/Hide Dialogs (**F12**) command.

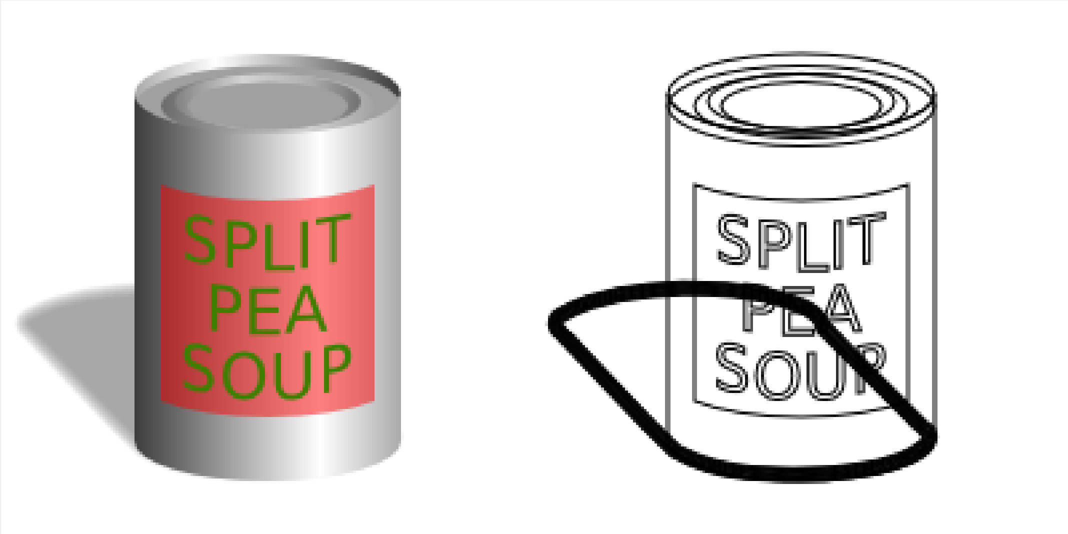
Outline

Updated in v0.45.

Inkscape has an *Outline* or *Wire-frame* mode. In this mode, all paths and shapes are drawn as outlines with a one screen-pixel-wide stroke and no fill, regardless of zoom level. Text is drawn with an inverse fill and no stroke. Images are outlined in red, clip paths in green, and masks in blue.

The *Outline* mode is useful for seeing the overall structure of a drawing, precise node editing, and for finding and selecting those pesky, hidden objects that may have been created by accident. The mode is marginally faster than the normal mode. It can be selected under the View → Display mode menu entry and toggled via View → DisplayMode → Toggle (**Ctrl**

+Keypad 5).



The same drawing shown in the normal view on the left and the wire-frame view on the right (the text has been converted to a path).

The colors used by the *Outline* mode can be changed by editing the *wireframecolors* group in the preferences file (`~/.inkscape/preferences.xml`). Inkscape can be forced to start up in *Outline* mode by adding <group id="startmode" outline="1"/> to the preferences file inside the *options* group.

Full Screen Mode

The Inkscape window can be made to cover the full screen with the View → Full Screen (**F11**) command. A second use of the same command returns the window to its original size and position.

Switch Windows

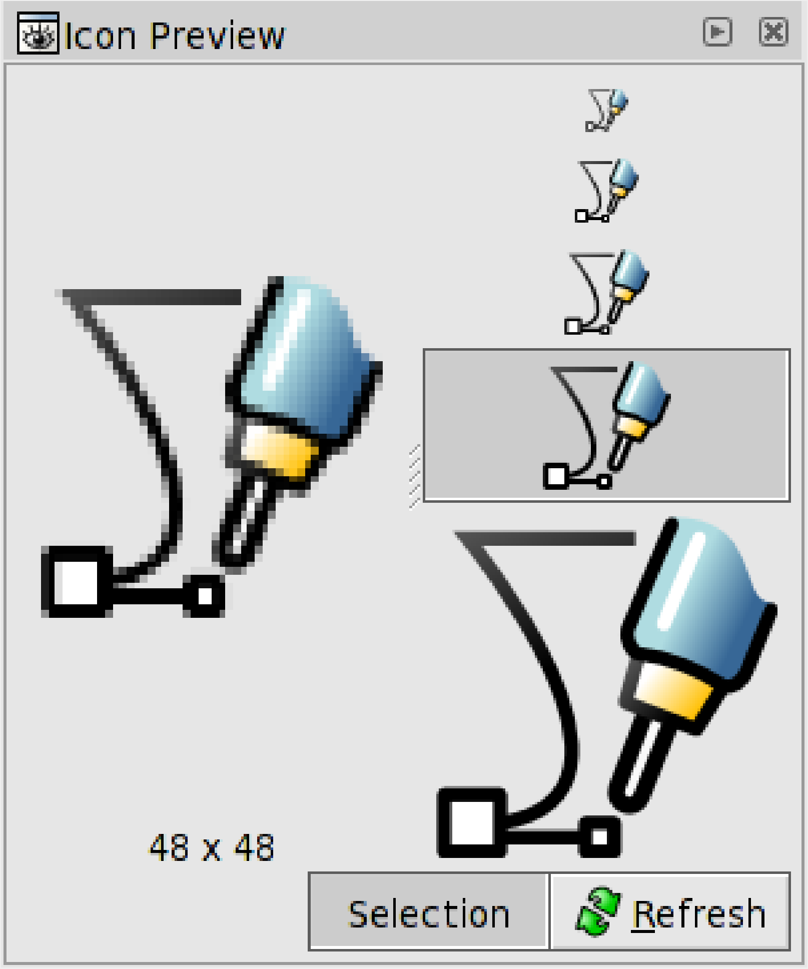
Each new drawing is created in a separate window. To move between these windows, you can use the method provided by your operating system (try **Alt**+**Tab**) or the methods provided by Inkscape View → Next Window (**Ctrl**+**Tab**) and View → Previous Window (**Shift**+**Ctrl**+**Tab**).

Duplicate Window

A duplicate Inkscape window can be created with the View → Duplicate Window command. Both the original and new window refer to the same drawing. Thus, one can use one window for detailed work while keeping watch over how the work affects a larger region of a drawing.

Icon Preview

An *Icon Preview* window can be created with the View → Icon Preview command. This allows one to see what a drawing (or selection) will look like as icons of different sizes. Which sizes are displayed can be specified in the preferences file (Linux: `~/.inkscape/preferences.xml`) under the *iconpreview* group.



A preview of the Inkscape Bezier icon with the *Icon Preview* dialog.

Zooming the Canvas
Table of Contents
Chapter 4. Editing Basics

Get the book.

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Editing Basics » Undo and Redo

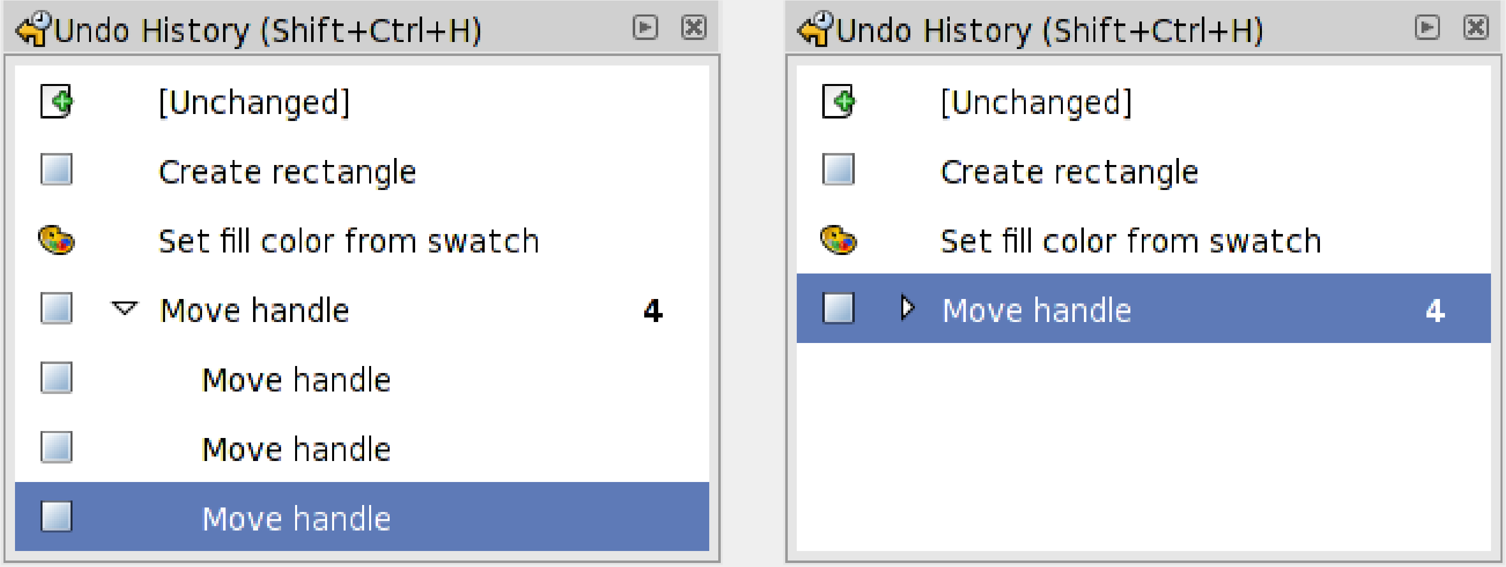
Index

Undo and Redo

One can undo and redo changes made to your drawing in the current Inkscape session. The *Undo* and *Redo* commands can be repeated to undo or redo multiple changes. These commands can be accessed both in the *Command Bar* and under the *Edit* menu in the *Menu Bar*. When accessed through the *Edit* menu, a description of the change that would happen if the command is selected is given.

- Edit → Undo (**Ctrl**+**Z**) (or **Shift**+**Ctrl**+**Y**): Undo change.
- Edit → Redo (**Ctrl**+**Y**) (or **Shift**+**Ctrl**+**Z**): Redo change.

New in v0.45. Multiple changes can also be undone or redone in one step by using the *Undo History* dialog (Edit → Undo History... (**Shift**+**Ctrl**+**H**)).



Two views of the *Undo History* dialog, both showing the drawing of a rectangle followed by changing the color and and four resizings. Clicking on any line returns the drawing to the state when the change described on the line was completed. Consecutive changes of the same type may be collapsed to one line to improve readability; the number of such changes is given on the top line of the group. Clicking on the triangle collapses (left) or uncollapses (right) the list.

All changes to a file can be undone with the File → Revert command.

Chapter 4. Editing Basics

Table of Contents

Selecting Objects

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Editing Basics » Selecting Objects

← Index →

Selecting Objects

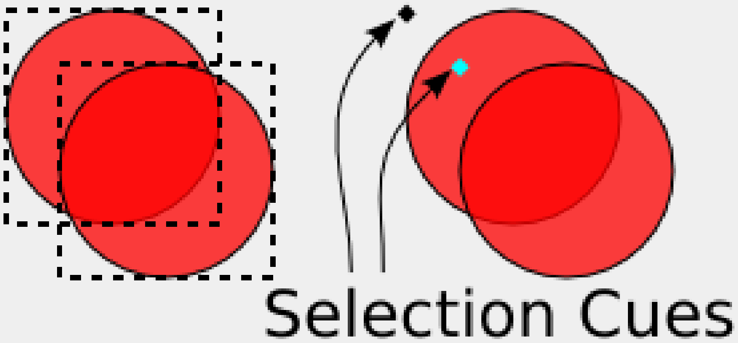
Before objects can be manipulated, they must be selected. This section covers the myriad of ways selection can be done.

Normally, the *Select Tool* is used for selecting objects. To activate, click on the ↖ icon in the *Tool Box* or use the keyboard shortcut (**F1**). One can toggle between another tool (except the *Text Tool* when in *text enter* mode) and the *Select Tool* by using the **Space Bar** if that other tool was selected first.

Some objects can be directly selected by other tools. For example, shapes (e.g., *Rectangles*) can be directly selected with any of the *shape tools*.

Rapid double clicking on an object when the *Select Tool* is active will select an object and change the tool to a tool appropriate for editing that object. For example, double clicking on a rectangle will select the rectangle and change the tool to the *Rectangle Tool*.

Multiple objects can be selected at the same time. The *Notification Region* lists the number and type of objects selected. This is especially useful when objects overlap and it isn't clear which are selected. By default, a box is drawn around each selected object. This can be changed to a small diamond *Selection Cue* under the *Tools-Selector* tab of the *Inkscape Preferences* dialog (File → Inkscape Preferences... (**Shift**+**Ctrl**+**P**)).



Selection indicated by dashed boxes (on left) and by small diamonds (on right). Only one of these two methods can be active at a time.

The *Find* dialog (Edit → Find... (**Ctrl**+**F**)) discussed at the end of this section, is another way of selecting objects.

Tip

If an object cannot be selected, it may be *locked*. Objects can be locked against modification in the *Object Properties* dialog (Object → Object Properties... (**Shift**+**Ctrl**+**O**)). Unlocking individual objects requires use of the *XML Editor* (look for the locked object and then either delete the “sodipidi:insensitive” attribute or uncheck the *Lock* box in the *Object Properties* dialog; see Chapter 14, *XML Editor*). As of v0.46 one can unlock all objects with the command Object → Unlock All. Objects may also be in a locked *Layer* (see the section called “Layers”).

Tip

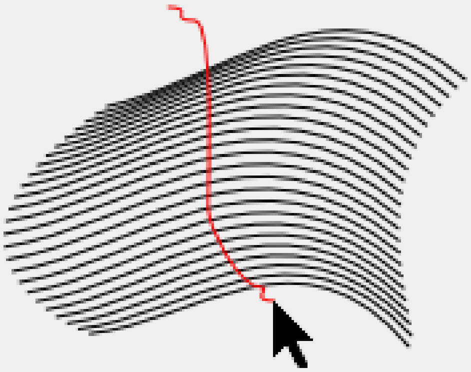
It is possible to “hide” an object by checking the *Hide* box in the *Object Properties* dialog.[9]Which raises the question: how do you select a hidden object? There are several ways. One is to use the *Find* dialog (Edit → Find... (**Ctrl**+**F**)); search for the Style *display:none* with the *Include hidden* box checked. A second way is to look for it with the *XML Editor* (see Chapter 14, *XML Editor*). A third way is to enable the selection of hidden objects using the **Tab** key in the *Inkscape Preferences* dialog's *Selecting* tab. A fourth way (as of v0.46) is to unhide all hidden objects using the Object → Unhide All command. While these are solutions, the ultimate solution is just to not hide objects. Instead put them on a separate *Layer* and hide the *Layer* (see the section called “Layers”).

Selecting with the Mouse

This section covers selecting objects with a mouse. By adding **Ctrl** to many of the commands below, objects within a *Group* can be selected (see the section called “Groups”).

- **Left Mouse Click**: Select Object: Select an object by clicking on it.
- **Shift**+**Left Mouse Click**: Toggle Selection: Add or subtract an object to or from a selection. This allows multiple objects to be selected. If the object clicked on is not already selected, it will be. If it is already selected, it will be deselected.
- **Alt**+**Left Mouse Click**: Select Under: Select next object below the currently selected object under the pointer. This allows one to select objects covered by others. Repeat to move downward in *z-order*. If the bottommost object is already selected, this will select the topmost object.
- **Left Mouse Drag**: Select Multiple Objects: When started from empty space, this will select all objects that are completely within the rectangle that is formed by the start and stop points of the drag. This is often referred to as *rubber-band* selecting. The *rubber band* is the temporary line drawn while dragging.
- **Shift**+**Left Mouse Drag**: Add Objects to Collection: Add objects within the *rubber band* to a preexisting selection. Also inhibits selection of an object under the start of the drag (without the shift, an object under the start of a drag is selected *and* moved). Selected objects within the *rubber band* are *not* deselected.
- *New in v0.46.*

  Alt+**Left Mouse Drag**: Touch Select Multiple Objects: This will select all objects that the mouse cursor touches while being dragged. A red line will indicate the path the cursor took. This is *very* useful when needing to select multiple paths as found in engravings or hair. The drag must begin with no object selected and at a point where no object is under the cursor (otherwise the selected object[s] or object under the cursor will be moved).

  Using **Alt**+**Left Mouse Drag** to “touch” select the individual paths.
- *New in v0.46.*

Shift+**Alt**+**Left Mouse Drag**: Touch-Add Multiple Objects to Collection: This will add all objects touched to a selection. It can also be used when the drag begins over an object.

Selecting with the Keyboard

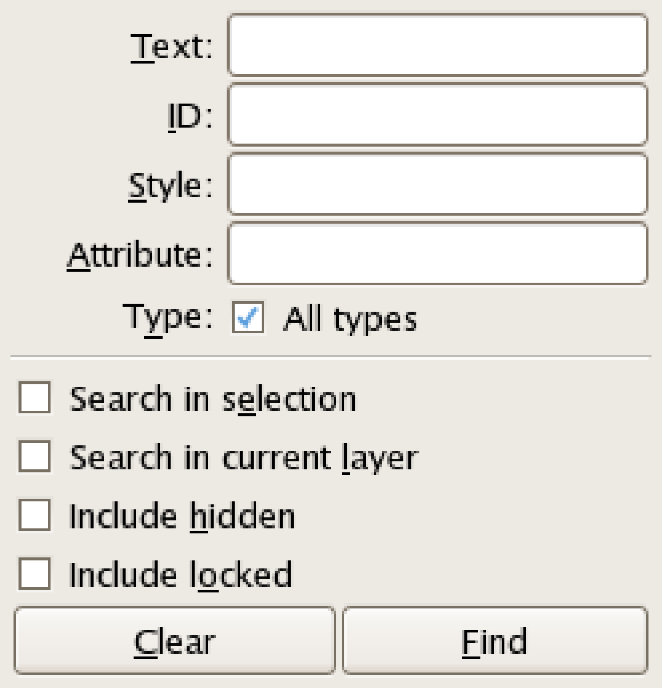
Most selecting of objects is done with the mouse. However, there are a few handy keyboard shortcuts.

- **Tab**: Select next object in *z-order*, that is the next object above the previous selected object. If no object is selected, it will select the lowest object. Works for objects in current *Layer*.
- **Shift**+**Tab**: Select previous object in *z-order*, that is the next object below the previous selected object. If no object is selected, it will select the highest object, which is normally the last object created. Works for objects in current *Layer*.
- **Ctrl**+**A**: Select All: Selects all objects in current *Layer*.
- **Shift**+**Ctrl**+**A**: Select All in All Layers: Selects all objects in all visible and unlocked *Layers*.
- **!**: Invert Selection: Invert selection in current *Layer*. That is, all selected objects are deselected and all unselected objects are selected.
- **Alt**+**!**: Invert selection in all visible and unlocked *Layers*. That is, all selected objects are deselected and all unselected objects are selected.
- **Esc**: Deselect: Deselect all selected objects.

Select All, Select All in All Layers, Invert Selection, and Deselect are also available under the *Edit* menu.

Selecting with the Find Dialog

The *Find* dialog (Edit → Find... (**Ctrl+F**)) is another way to select objects with common features. A search is made through the *SVG* file and the found objects are selected on the Inkscape canvas.



Type: ☑ All types

☐ Search in selection

☐ Search in current layer

☐ Include hidden

☐ Include locked

Find dialog.

The *Find* dialog has two parts. The top part defines what is searched for while the bottom part defines where to search.

You can specify a *Text* string (in a text object), *ID* name (e.g., “rect1314”), *Style* feature (e.g., “fill-opacity”), or an *Attribute* type (e.g., “width”) for the search. The matches can be full or partial (i.e., “rect” would match “rect1314” and “rect1316”).

With the check boxes, you can specify where to search. If the *All types* box is checked, the search will take place in all objects (shapes, paths, text strings...). If the box is not checked, a list of object types is presented that can be individually include or excluded from the search.

The *Search in selection* box determines if the search is limited to the currently selected objects. The *Search in current layer* box determines if the search is limited to the currently selected layer. The *Include hidden* box determines if hidden objects are included in the search while the *Include locked* box determines if locked objects are included in the search.

At the very bottom of the dialog are two buttons: the first, *Clear*, to clear the search entry boxes and the second, *Find*, to perform the actual search. All objects that are found to match the search criteria are selected.

[9] Why would you want to hide and object? One example would be to save a text object before

converting it into a path in case you ever wanted to edit the text again.

← Undo and Redo

↑ Table of Contents

→ Copying, Pasting, and Deleting Objects

Inkscape » Editing Basics » Copying, Pasting, and Deleting Objects

← Index →

Copying, Pasting, and Deleting Objects

Inkscape includes commands to copy, paste, and delete objects. Commands of this type can be found in three places: Under the *Edit* menu (see below), in the *Command Bar* (Cut, Copy, Paste, and Duplicate), and in the pop-up menu when you **Right Mouse Click** over the canvas (Cut, Copy, Paste, Duplicate, and Delete).

Clipboard: Except for text, copying works via an internal *clipboard* (a place where a description of one or more objects are stored temporarily in memory). Thus, it is not possible to paste non-text objects between different instances of Inkscape or Inkscape and an external program. It is possible to paste between different drawings if they were opened via the same Inkscape process. Adding an object or group of objects (a selection) to the *clipboard* removes any previous selection stored in the *clipboard*.

The commands available under the *Edit* menu are:

- Edit → Cut (**Ctrl+X**): Selection is removed from drawing, but stored in *clipboard*.
- Edit → Copy (**Ctrl+C**): Copy of selection is stored in *clipboard*.
- Edit → Paste (**Ctrl+V**): A copy of objects stored in *clipboard* is placed in the drawing at the location of the cursor.
- Edit → Paste In Place (**Ctrl+Alt+V**): Objects are copied from *clipboard* to original location of objects. This is very useful in copying a selection from one drawing to the corresponding spot in another drawing.
- Edit → Paste Style (**Shift+Ctrl+V**): The style (fill and stroke attributes or for text the font, size, and style) of the object in the *clipboard* are given to the selected object(s).
- Edit → Duplicate (**Ctrl+D**): An object is copied and placed on top of the original

object. The copied object is left selected.

- Edit → Make a Bitmap Copy (**Alt+B**): The selection is exported as a bitmap then pasted back into the drawing at the same place. The original objects remain unchanged. The exported bitmap is *not* stored in the SVG file.

  New in v0.45:

  The bitmap will have a resolution specified by one of the following, in order of preference: a resolution specified in the `~/.inkscape/preferences.xml` in the group id="createbitmap" of the form: resolution="xx", a resolution used for exporting the file to a bitmap, or the default bitmap resolution of 90 *dpi*. In the last case, the pixels will be snapped to the pixel grid.
- Edit → Delete (**Delete**) or (**Backspace**): Selection is removed from drawing and *not* stored in *clipboard*.

Hitting the **Space Bar** while dragging or transforming an object with the mouse will “drop” a copy of the object at the current point.

Another method for copying objects, *cloning*, is discussed in the next section.

Selecting Objects

Table of Contents

Clones

Clones

Cloning is a special way to copy an object. The cloned copy retains a link to the original object so if that object is changed, the clone will change in the same way. The copies, however, can be *transformed* (moved, scaled, rotated, and skewed) independently. Technically, the cloning is achieved through the use of the *SVG* "*use*" object type.

The style (color, fill pattern, etc.) of the clones can be changed independently but only if the style of the cloned object is *Unset*. See the section called "Fill and Stroke Paint" in Chapter 9, *Attributes*.

Only a single object can be cloned at a time. If more than one object needs to be cloned, then those objects can be placed into a *Group*.

The original may be cloned any number of times. It is also possible to clone a clone. In this case, changing the original will change both clones while changing the first clone will only change the second clone. It is not possible, however, to set the *Attributes* independently with multiple layers of cloning.

Inkscape includes the *Create Tiled Clones* dialog for creating a set of cloned object that are automatically placed via tiling algorithms. This dialog is very powerful and is sufficiently complex that it deserves its own chapter. See Chapter 15, *Tiling* , for more information.

The following commands are available for working with clones (*Clone* and *Unlink Clone* are also located in the *Command Bar*).

- Edit → Clone → Clone (**Alt+D**): Clone object. Clone is placed exactly over original.
- Edit → Clone → Unlink Clone (**Shift+Alt+D**): Remove link between clone and original object. Clone will no longer update when original object is modified.
- Edit → Clone → Select Original (**Shift+D**): Select original of clone. Use to find the original of a cloned object. Select clone before using. Use multiple times to find original if starting with a clone of a clone.

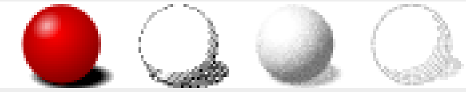
What happens when you move a cloned object? By default, the clones don't move. This is so that if you select a object that was cloned and one of its clones, the two will move together as expected. In the *Inkscape Preferences* dialog under the *Clone* section, you can change this behavior so that if a cloned object is moved, its clones also move. But in this case, if you select a cloned object and one of its clones, then move them, the clone will move “twice” as far... once because the cloned object moved and once because the clone itself was moved. Things can get really strange if you move a cloned object that has been cloned and that clone has been cloned!

← Copying, Pasting, and Deleting Objects

↑ Table of Contents

→ Ordering Objects (Z-order)

Get the book.



Inkscape: Guide to a Vector Drawing Program

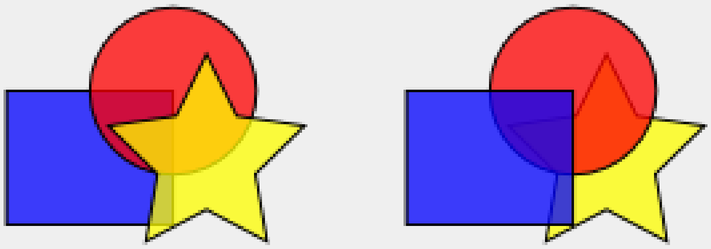
Get the book.

Inkscape » Editing Basics » Ordering Objects (Z-order)

Index

Ordering Objects (Z-order)

The *z-order* determines the order in which objects are drawn on the canvas. Those object with high *z-order* are drawn last and therefor drawn on top of objects with lower *z-order*. The order is determined by the order that the objects are listed in the *SVG* file.



The yellow star has the highest *z-order* on the left but the lowest on the right.

Inkscape has a number of commands to change *z-order*. The commands are available from the Menu and via Keyboard Shortcuts. They are also available in the *Tool Controls* when the *Select Tool* is active.

- Object → Raise (**Page Up**): Move selected object(s) up one step.
- Object → Lower (**Page Down**): Move selected object(s) down one step.
- Object → Raise to Top (**Home**): Move selected object(s) to top.
- Object → Lower to Bottom (**End**): Move selected object(s) to bottom.

The *XML Editor* dialog can also be used to change *z-order*.

← Clones

↑ Table of Contents

→ Groups

Groups

A set of objects can be collected into a *Group*. Once placed in a *Group*, the objects can be manipulated together, the *Group* acting as a single object. *Groups* can be nested; that is, a *Group* can be combined with other *Groups* or objects to make a higher level *Group*.

The following commands, found both in the *Command Bar* and under the *Edit* menu, can be used to make and break up groups:

- Object → Group (**Ctrl**+**G**): Group selected items.
- Object → Ungroup (**Shift**+**Ctrl**+**G**): Ungroup selected *Group*. Ungroups one level.

An alternative method to break up a *Group* is to select the *Ungroup* command from the menu that pops up when you **Right Mouse Click** over a *Group*.

Objects within *Groups* can be edited and manipulated without breaking up the *Group*. The following commands select objects within *Groups*:

- **Ctrl**+**Left Mouse Click**: Select Within Group. Select an object within a *Group*. This command selects objects regardless of how many levels of *Groups* they are buried in.
- **Ctrl**+**Alt**+**Left Mouse Click**: Select Under Within Group. Select an object within a *Group* that is under the cursor and that is next lower in *z-order* than the currently selected object. If the currently selected object is at the bottom in *z-order*, the top object is selected. This command selects objects regardless of how many levels of *Groups* they are buried in.
- **Shift**+**Ctrl**+**Alt**+**Left Mouse Click**: Toggle Under in Groups. Add an unselected object

or remove a selected object within a *Group* to or from a selection.

A *Group* can be *entered* or turned into a temporary *Layer* for editing. Two ways to *Enter a Group* and *Exit a Group* are available:

- **Double**+**Left Mouse Click** on a *Group* to enter a *Group*. Click an object outside the *Group* to exit.
- **Ctrl**+**Enter** on a *Group* to enter a *Group*. **Ctrl**+**Backspace** to leave a *Group*. This will take you up one group level.
- **Right Mouse Click** over *Group* to pop up a menu. Select the *Enter group* command to enter the *Group*. The command line included the name of the *Group* that will be entered. Once in a *Group*, the menu option *Go to parent* will exit the *Group* (although there is no visible feedback).

Tip

To add an existing object to an existing *Group*, cut the object (Edit → Cut (**Ctrl+X**)), enter the *Group* (**Double**+**Left Mouse Click**), and then paste in place (Edit → Paste In Place (**Ctrl+Alt+V**)).

Tip

The *XML Editor* dialog can be very useful in rearranging objects between *Groups*.

Ordering Objects (Z-order)
Table of Contents
Layers

Inkscape » Editing Basics » Layers

Index

Layers

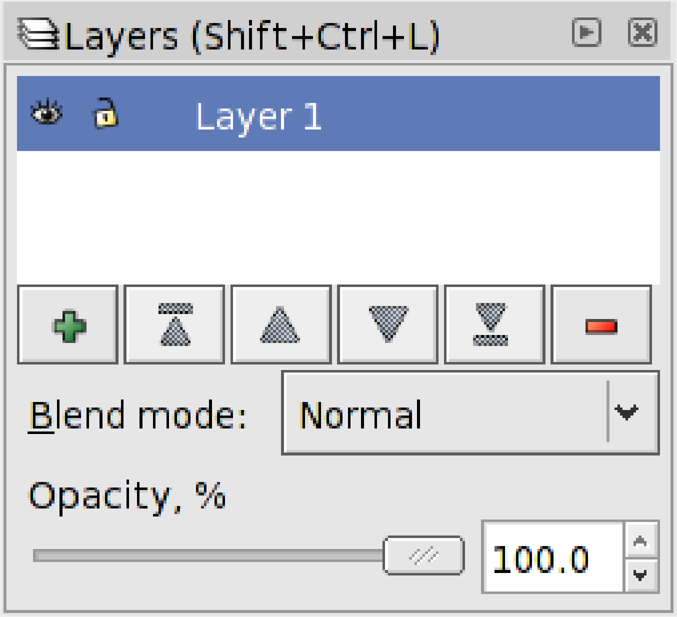
One can think of *Layers* as the acetate sheets that were once used to make animated cartoons. Each *Layer* can contain one or more objects. The final image is made up of all the (visible) *Layers* stacked on top of each other. *Layers* can be made visible or invisible, given an opacity, moved up or down relative to other *Layers*, locked, deleted, and named.

Internally, *Layers* are just *SVG* *Groups* with a few extra Inkscape specific parameters that Inkscape uses to control the *Layer* interface. Like *Groups*, *Layers* can contain sub-*Layers*.

Layers can be manipulated by using the *Layers* dialog, through commands in the *Layers* menu, or through items in the *Status Bar*. New in v0.46, the *Layers* dialog can also be used to quickly blend a *Layer* with the background. See below for details.

Layers Dialog

The *Layers* dialog is the easiest way to manipulate *Layers*. Call up the dialog via the Layer → Layers... (**Shift**+**Ctrl**+**L**) command.



Layers (Shift+Ctrl+L)

👁 🔓 Layer 1

Blend mode: Normal

Opacity, % 100.0

Layers dialog with the default layer.

The dialog will list all the *Layers* in the document. The currently selected *Layer* is highlighted. Selecting an object in a different *Layer* will select the *Layer* that object is in. One can also click on a *Layer* name to select that *Layer*. Only one *Layer* can be selected at a time.

In front of the *Layer* names are two icons that show if a layer is visible (👁) or not (⊘) and if a layer is locked (🔒) or unlocked (🔓). Clicking on one of the icons toggles the state of that icon. A layer need not be selected in order to toggle its states. Objects in locked or invisible *Layers* cannot be selected or changed.

The buttons below the *Layer* list can be used to add or delete a *Layer* and to move the selected *Layer* up or down relative to other *Layers*. A sub-*Layer* cannot be moved outside of the *Layer* it is under; you can use the XML Editor to do this.

- ✚ : Add a new *Layer*. A dialog will pop up allowing input of the *Layer* name. A drop-down menu allows you to add the new *Layer* above, below, or as a sub-*Layer* of the currently selected layer.
- ⤒ : Move selected layer to top.
- ▲ : Move selected layer up one layer.
- ▼ : Move selected layer down one step.
- ⤓ : Move selected layer to bottom.
- ▬ : Delete selected layer.

New in v0.46:

Beneath the row of buttons there is a drop-down menu for the *Blend mode*. This is a short cut to applying the *Blend* filter to the entire *Layer*. The *Normal* entry corresponds to no *Filter*. Selecting one of the other entries creates a *Blend* filter with the selected mode. The first input to the filter is the *Layer* (recall that a *Layer* is just a *Group*) and the second input is the *Background Image*. If the menu is returned to the *Normal* setting, the *Blend* filter is automatically deleted.

Layer 1
Layer 2

A demonstration of using *Layer* blending. The two texts are in different *Layers* with "Layer 1" beneath "Layer 2". The *Blend mode* is set to *Screen* for the upper *Layer*.

At the bottom of the dialog is a slider and an entry box that can be used to change the opacity of the *Layer*.

Layers Menu

The *Layer* menu has commands to create, rename, delete, and change the order of *Layers*. There are also commands to move objects between *Layers*. The commands in the *Layers* menu are

- Layer → Add Layer... : Add a new layer above currently selected layer. A simple dialog opens up where the *Layer* can be named.
- Layer → Rename Layer... : Rename selected layer. A simple dialog opens up where the *Layer* can be renamed.
- Layer → Switch to Layer Above: Select layer above current layer.
- Layer → Switch to Layer Below: Select layer below current layer.
- Layer → Move Selection to Layer Above (**Shift**+**Page Up**): Move selected object(s) to layer above current layer.
- Layer → Move Selection to Layer Below (**Shift**+**Page Down**): Move selected object (s) to layer below current layer.
- Layer → Raise Layer (**Shift**+**Ctrl**+**Page Up**): Move current layer above layer above.
- Layer → Lower Layer (**Shift**+**Ctrl**+**Page Down**): Move current layer under layer below.
- Layer → Layer to Top (**Shift**+**Ctrl**+**Home**): Move current layer to top of layer stack.
- Layer → Layer to Bottom (**Shift**+**Ctrl**+**End**): Move current layer to bottom of layer stack.

- Layer → Delete Current Layer: Delete current layer.

Status Bar

The *Status Bar* includes a pull-down menu to select the current *Layer*. The icons in front of the menu indicate if the layer is visible or hidden and if it is locked or unlocked (see the section called “Layers Dialog”). Clicking on an icon toggles the state.

Groups

Table of Contents

Chapter 5. Positioning and Transforming

Inkscape » Positioning and Transforming » Inkscape Coordinates

← Index →

Inkscape Coordinates

Understanding the way Inkscape handles coordinates is necessary to be able to fully take advantage of the tools Inkscape has for positioning objects. When you open the main Inkscape window, *Rulers* are drawn by default at the top and left edges of the canvas (see the section called “The Anatomy of the Inkscape Window”). The units of the *Rulers* are the same as the default units of the canvas. If you hover the pointer over a *Ruler*, a *tool tip* will show the current unit. The default unit can be changed via the *Document Properties* dialog (File → Document Properties... (**Shift**+**Ctrl**+**D**)). Options for the coordinates system units are:

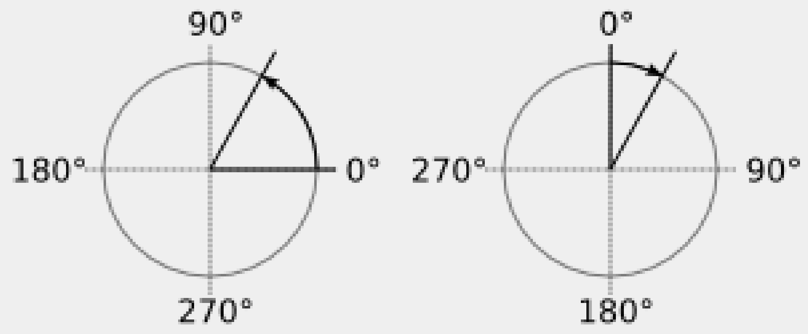
cm	Centimeters
ft	Feet
in	Inches
m	Meters
mm	Millimeters
pc	Picas
pt	Points
px	Pixels

Conversion between the units is fairly straightforward: 1 inch = 1/12 ft = 2.54 cm = 25.4 mm = 0.0254 m = 6 pc = 72 pt. The pixel (px) unit is adjustable in the *Inkscape Preferences* dialog (File → Inkscape Preferences... (**Shift+Ctrl+P**)) under the *Import/Export* tab (Default export resolution). It is also equivalent to the *User Unit* in the *SVG* specification. Inkscape takes care of conversions when changing units. Note: Feet and meters are not *SVG* or *CSS* defined units.

Various Inkscape parameters can be set using independent units. For example, the default *x* scale can be set to millimeters, while the alignment *Grid* can be defined in inches.

One confusing aspect is that Inkscape uses a different scale internally. On the canvas, the *x* and *y* coordinates increase as one moves right or up. Internally the *y* coordinate is flipped as per the *SVG* standard. Thus, (0, 0) is defined from the upper-left corner of the page region internally but at the bottom-left corner in the canvas window. The internal scale is fixed (by default) to 90 *dpi*. The internal scale is important if you wish to edit by hand an object using the *XML Editor*.

There are two competing camps for how angles should be defined. Fortunately for the peace, Inkscape supports both through the *Compass-like display of angles* option under the *Steps* tab in the *Document Properties* dialog (File → Document Properties... (**Shift+Ctrl+D**)).



Definition of angles. Left: The mathematician's view (default). Right: The geographer's view.

Chapter 5. Positioning and Transforming
Table of Contents
Transformations

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Positioning and Transforming » Transformations

← Index →

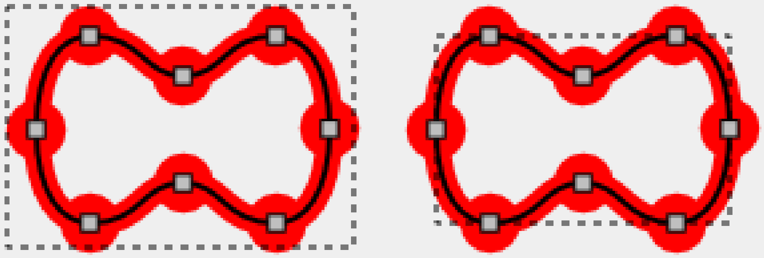
Transformations

Objects can be moved, scaled, rotated, skewed, and flipped. Inkscape provides a variety of ways to make these transformations. These include: using the *mouse*, the *keyboard*, items in the *Tool Controls bar*, the *Transform dialog*, the *Paste Size commands*, and the *XML Editor dialog*. Each method will be discussed in turn. Each method begins by selecting an object or group of objects to be transformed. The *Select Tool* must be active for making transformations with the mouse or keyboard. To activate, click on the icon in the *Tool Box* or use the keyboard shortcut **(F1)**.

There are a couple of things to note:

First, some operations use *SVG* pixels while some use *Screen pixels*. The former is the scale used in the *SVG* file. It is fixed (by default) to 90 *dpi*. The latter refers to a pixel on the screen and changes as the zoom level changes.

Second, positions and dimensions are often with reference to the *bounding box*. As of v0.46, one can choose between two definitions for the *bounding box* in the *Tools* section of the *Inkscape Preferences* dialog (File → Inkscape Preferences... (**Shift+Ctrl+P**)). The *Visual bounding box* includes the stroke width if the stroke is visible. For example, a square 100 pixels to one side (between corner nodes) will have a width of 102 pixels if the stroke width is 2 pixels. The *Visual bounding box* also includes any *Markers* and both the *Stroke Join* and *Cap* styles are assumed to be *Round*. The *Geometric bounding box* mode uses only the nodes to determine the *bounding box*.



The dashed lines are *bounding boxes* for the same path with *Markers*. Left: *Visual bounding box*. Right: *Geometric bounding box*.

Third, transformations do *not* change the underlying definition of regular shape objects or grouped objects, as discussed in the introduction of this chapter (an exception being simple transformations of *Rectangle* objects when the *Store transformation* parameter is set to *Optimized*).

Fourth, there are a number of options that can be toggled on and off in the *Tool Controls* when the *Select Tool* is being used. The following are active when the icon is highlighted:

- Scale stroke width when object is scaled.
- Scale radii of rounded corners for rectangles.
- Transform gradient along with object.
- Transform patterns with object.

Fifth, rotation and skewing take place around a *Rotation center* point. The point is indicated by a draggable “plus”-shaped handle that is viewable when using the *Select Tool* in rotation or skewing mode. See below for details.

Transforms with the Mouse

Translations

The **Shift**, **Ctrl**, and **Alt** keys can be used in combination:

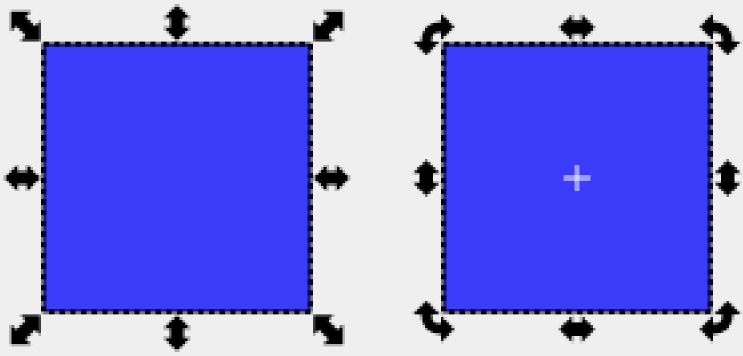
- **Left Mouse Drag**: Select (if not selected) and move object. Move selected object(s) if drag starts on any selected object.
- **Alt**+**Left Mouse Drag**: Move selected object(s) regardless of where drag starts. Don't select object where drag began.
- **Ctrl**+**Left Mouse Drag**: Move selected object(s) constrained to horizontal or vertical

directions.

- **Shift**+**Left Mouse Drag**: Temporarily disable snapping to *Grids* or *Guide Lines*.

Scaling, Rotating, and Skewing

When an object or objects are first selected, eight double-headed arrows will appear in a rectangle around the selection. A **Left Mouse Drag** of any handle will rescale the selection. The corner arrows will scale both in the horizontal (*x*) and vertical (*y*) directions. The side arrows will scale in only one direction.



Scaling arrows (left). Rotation arrows on corners, skewing arrows on sides (right). The “plus” in the center of the square on the right is the *Rotation center* handle.

Updated for v0.46 (keyboard shortcut):

Clicking a second time on a selected object or using the keyboard shortcut **Shift**+**S** will change the direction of the double headed arrows. Now, a **Left Mouse Drag** of a handle will rotate the selection if used on a corner arrow, or skew the selection if used on a side arrow. Click again to revert to the scaling mode.

Rotation Center. Rotation takes place around the *Rotation center* indicated by a “plus”-shaped handle. The handle (and thus the center of rotation) can be dragged. Dragging while holding the **Ctrl** down will restrict movement to the horizontal and vertical directions. The handle will snap to the edge of the *bounding box* or the center axis of the selection's *bounding box*. It will also snap to a *Grid* or *Guide Lines* if snapping of nodes to those items is turned on. The movement of the *Rotation center* handle can be undone (**Ctrl**+**Z**). Holding the **Shift** key down while click on the *Rotation center* handle will restore the handle to the center of the *bounding box*. The *Rotation center* is preserved when an object is moved, scaled, duplicated, cloned, or converted to a path. It is also preserved between editing sessions.

When multiple objects are selected, the *Rotation center* of the first selected object will be used if it has been moved from its default position. Otherwise, the center of the *bounding box* of all selected objects will be used.

The **Shift**, **Ctrl**, and **Alt** keys can be used with the **Left Mouse Drag**. They can be used in combination when scaling, rotating, or skewing.

Scaling

- **Ctrl**: Preserve width to height ratio.
- **Shift**: Rescale symmetrically, around center of selection.
- *Redefined in v0.46.*

  Alt: Restrict scaling up to an integer factor (2, 3, 4, ...) or down to a simple fraction (1/2, 1/3, 1/4, ...). Negative values are also allowed (i.e., mirroring object around *bounding box* edge).

Rotating and Skewing

- **Ctrl**: Constrain a rotation or skew to a multiple of the *Rotation snap angle*. Allows stretching in the orthogonal direction to a skew by a multiple of the width or height of the *bounding box*.
- **Shift**: Keep opposite corner fixed for rotation or opposite side fixed for skew. (This is opposite of what happens for scaling.)

Transforms with the Keyboard

A selection may be moved, scaled, rotated, or flipped (but not skewed) with the keyboard. For some key combinations, the size of the transformation is determined by parameters (e.g., *Nudge factor*) that can be set in the *Inkscape Preferences* dialog (File → Inkscape Preferences... (**Shift**+**Ctrl**+**P**)) under the *Steps* tab.

Translations

- **Arrow**: Move selection by *Nudge factor* (2 SVG pixels by default).
- **Shift**+**Arrow**: Move selection by ten times the *Nudge factor*.
- **Alt**+**Arrow**: Move the selection one *Screen pixel*.
- **Alt**+**Shift**+**Arrow**: Move the selection ten *Screen pixels*.

Scaling

Scaling is around the center point of the *bounding box*.

- **.**, **>**: Scale up by *Scale step* (2 SVG pixels by default).
- **,**, **<**: Scale down by *Scale step* (2 SVG pixels by default).
- **Ctrl**+**.**, **Ctrl**+**>**: Scale up to 200%.
- **Ctrl**+**,**, **Ctrl**+**<**: Scale down to 50%.
- **Alt**+**.**, **Alt**+**>**: Scale up by one *Screen pixel*. Scale factor = 1 screen pixel/distance from center of *bounding box* to farthest edge.
- **Alt**+**,**, **Alt**+**<**: Scale down by one *Screen pixel*.

Rotation

Rotation is around the *Rotation center*.

- **[**: Rotate clockwise by *Rotation snap angle* (15 degrees by default).
- **]**: Rotate counter-clockwise by *Rotation snap angle* (15 degrees by default).
- **Ctrl**+**[**: Rotate clockwise by 90 degrees.
- **Ctrl**+**]**: Rotate counter-clockwise by 90 degrees.
- **Alt**+**[**: Rotate clockwise by one *Screen pixel* (angle = arctan[1 *Screen pixel* divided by the distance from the center to the corner point of the *bounding box*]).
- **Alt**+**]**: Rotate counter-clockwise by one *Screen pixel*.

Flipping

Modified in v0.46.

Flip around center point of *bounding box* if in *scaling* mode or around horizontal/vertical line passing through *Rotation center* if in *rotation*/*skewing* mode. These keys work when any tool is active.

- **H**: Flip horizontally.
- **V**: Flip vertically.

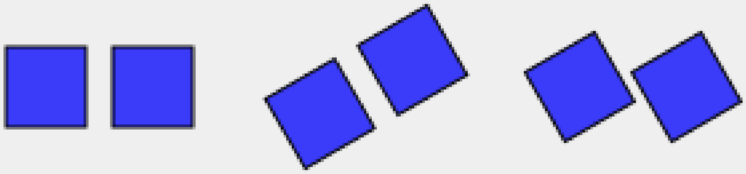
Transforms with the Tool Controls Bar

The *Tool Controls* contains a number of items for transforming an object when the *Select Tool* is in use. An object can be translated using the X and Y entry boxes. An object can be stretched by using the width (W) and height (H) entry boxes. These quantities are specified in a unit of length determined by a selection box just to the right of the entry boxes. The ratio of the height and width can be locked by clicking on the lock icon so that changing one dimension automatically changes the other. There are also icons for rotating and flipping objects.

Transforms with the Transform Dialog

Objects can be moved, scaled, rotated, and skewed using the *Transform* dialog (Object → Transform... (**Shift+Ctrl+M**)). There is a different tab in the dialog for each of these transforms. In addition, there is a *Matrix* tab that allows the application of a *Transformation Matrix* to a selection.

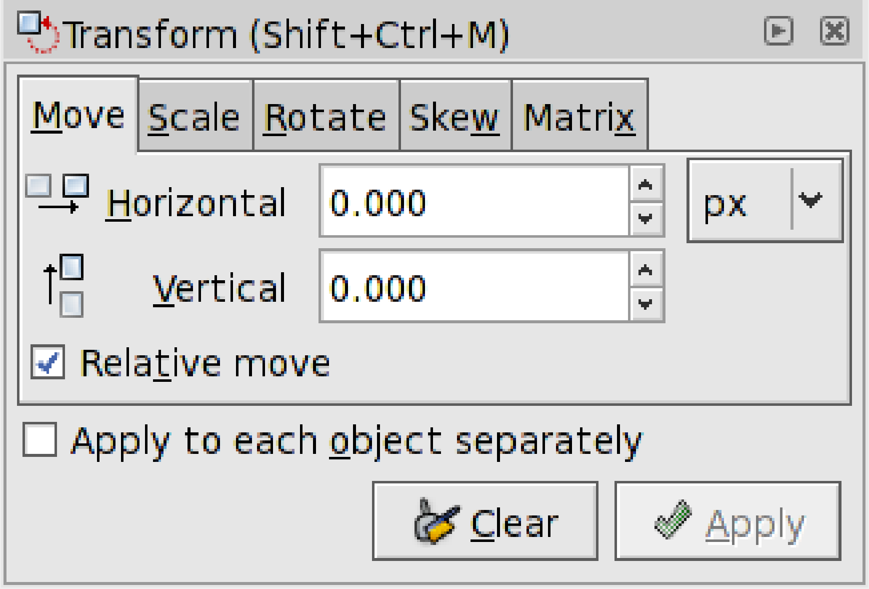
The *Transform* dialog contains an option to apply the chosen transformation to a selection as a group or to the individual objects within the selection. (This option has no effect for the *Move* and *Matrix* tabs.) The dialog also has a *Clear* button to reset the entered values to their default values.



Two squares (Left) are transformed as a group (Center) or separately (Right).

Move Tab

Using the *Move* tab, you can translate an object.



Transform (Shift+Ctrl+M)

Move | Scale | Rotate | Skew | Matrix

Horizontal 0.000 px

Vertical 0.000

☑ Relative move

☐ Apply to each object separately

Clear | Apply

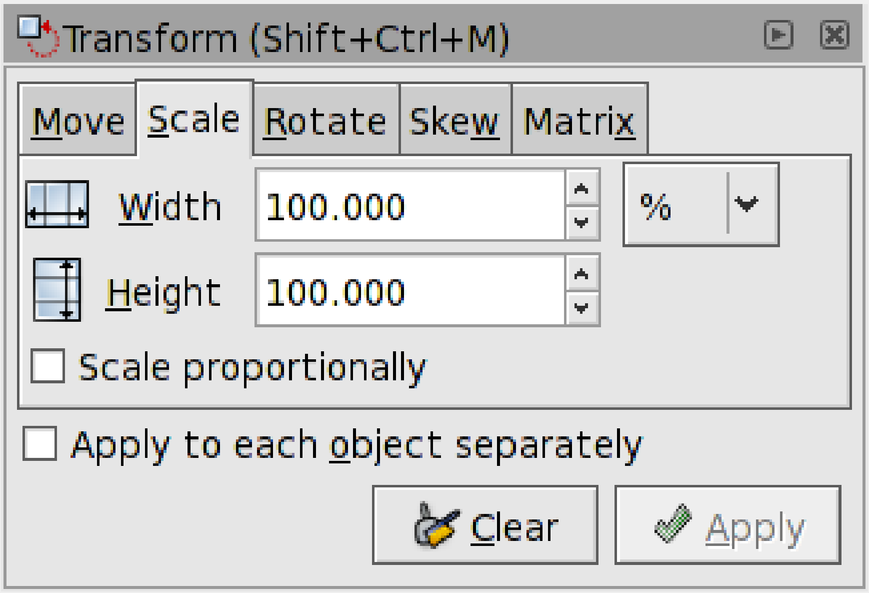
Move tab.

An object will be translated relative to its current position if the *Relative move* box is checked. If the box is not checked, the lower-left corner of the objects bounding box will be moved to the given coordinate.

Scale Tab

Modified (again) for v0.46.

Using the *Scale* tab, you can scale an object.



Width 100.000 %

Height 100.000

□ Scale proportionally

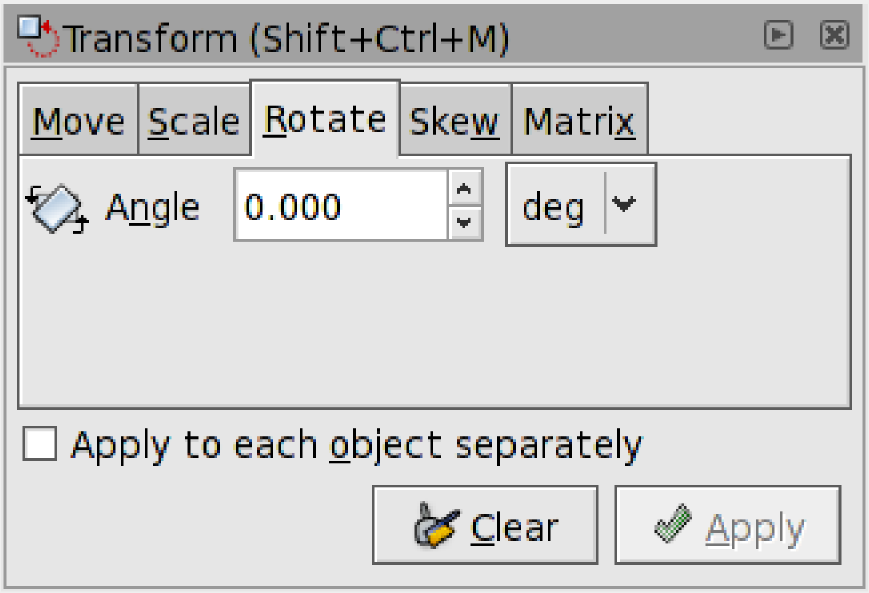
□ Apply to each object separately

Scale tab.

An object will be scaled relative to the center of its bounding box. The *Scale proportionally* option forces the width and height to scale by the same percentage. Note that a scale factor of 100% corresponds to leaving an object unchanged, in contrast to v0.45 where a factor of 0% left an object unchanged.

Rotate Tab

Using the *Rotate* tab, you can rotate an object.



Angle 0.000 deg

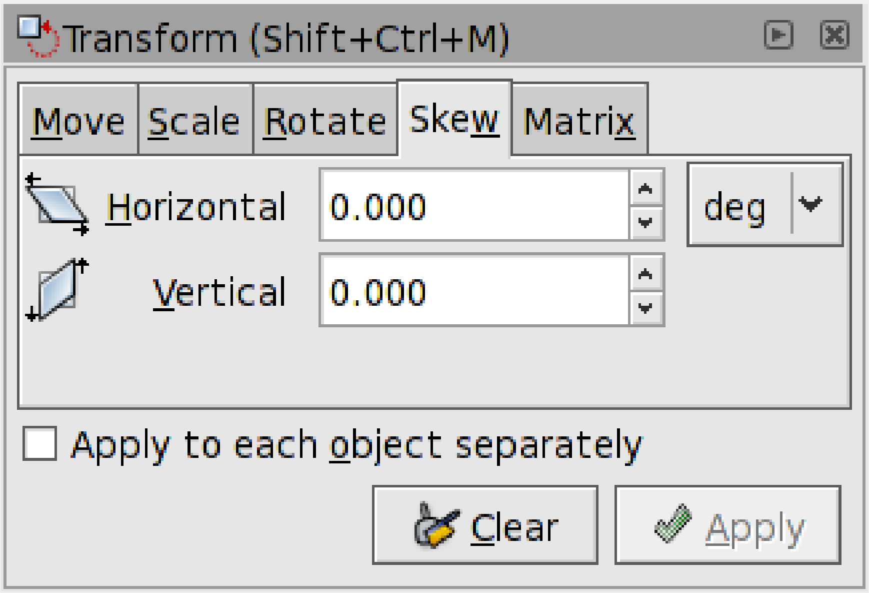
□ Apply to each object separately

Rotate tab.

An object will be rotated relative to [Rotation center](). The direction of the rotation is positive in the counter-clockwise direction.

Skew Tab

Using the *Skew* tab, you can skew an object.



Horizontal 0.000 deg

Vertical 0.000

□ Apply to each object separately

Skew tab.

You can skew in the horizontal and vertical directions separately. The skewing is relative to the center of the [bounding box](). The magnitude of the skew can be specified as a distance, percentage, or angle. In all cases, the skew is relative to the size of the *bounding box*.

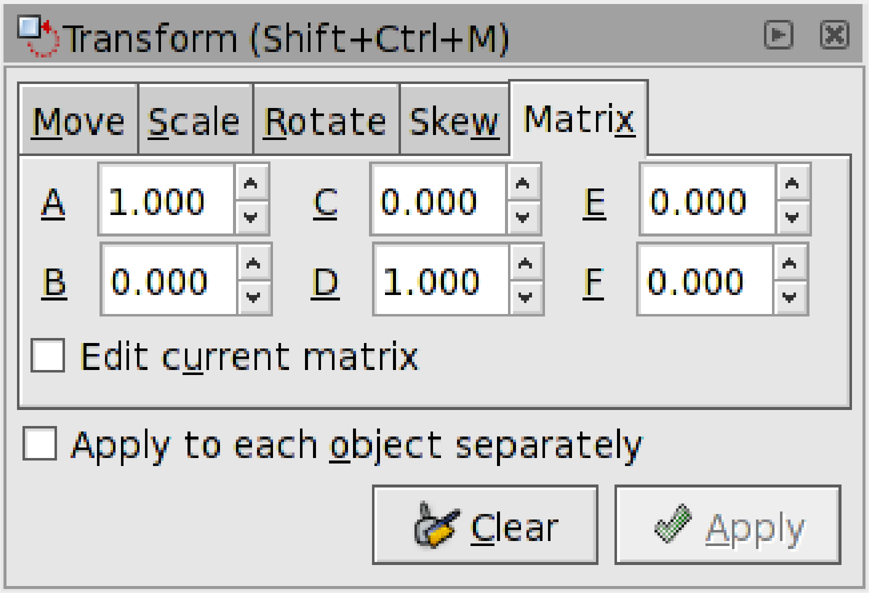
Examples for a horizontal skew of a 100 by 50 px *bounding box* with the *Rotation center* in the middle of the *bounding box*:

- Distance of 20 px: The top edge of the box is moved 10 px (half of 20 px) to the right, the bottom 10 px to the left.
- Percentage of 20%: The top edge is moved 5 px to the right (half of 20% of the height) to the right, the bottom 5 px to the left.
- Angle of 30%: The top edge of the box is moved 14.4 px (tan(30°) × 50px × 0.5) to the *left* (angles are defined to be positive in the counter-clockwise direction), the bottom 14.4 px to the right.

Note: The center of the new *bounding box* is not necessarily at the center of the original *bounding box* if the skewed object was not a rectangle.

Matrix Tab

Using the *Matrix* tab, you can apply a generic transformation to an object.



A 1.000 C 0.000 E 0.000

B 0.000 D 1.000 F 0.000

□ Edit current matrix

□ Apply to each object separately

Matrix tab.

The transformation is described by a 3×3 *Transformation Matrix* of which only the upper two rows are displayed and modifiable. The upper left 2×2 submatrix (A, B, C, and D) controls scaling, rotating, and skewing, while the upper right 1×2 submatrix (E and F) controls translations.

The tab includes the option *Edit current matrix* to select if the entered matrix should post-multiply the existing transformation matrix (option not selected) or if it should replace the current matrix (option selected).

There are two important points to note. First, the transformation matrix is with respect to the point (0, 0) in *screen* coordinates if not editing the current matrix. If editing the current matrix, the transformation is with respect to the *User Coordinate System* which, if an object is not in a *Group*, is equivalent to the *SVG* coordinate system (*Initial View Port*) where the origin is at the top-left corner of the page. See the SVG standard for more details.

Second, Inkscape will modify the matrix and other parameters of an object internally so that the internal E and F terms are zero if the *Store transformation* parameter under the *Transforms* section in the *Inkscape Preferences* dialog is set to *Optimized*. This means, for example, that for a horizontal skew of a rectangle the internal height parameter may change. The displayed object will still look correct.

Transforms with Paste Size Commands

The commands in the Edit → Paste Size submenu can be used to scale a selection or the objects in a selection to match the width and/or height of a selection that is stored in the *clipboard*. To use the commands, first copy (or cut) a selection with the desired dimension(s) to load the selection into the *clipboard*. Then select the target object or objects and use one of the commands below.

The first three commands scale a selection as a whole to match the clipboard while the last three commands scale each object to match the clipboard. Dimensions are determined by *bounding boxes*. The *Scale ratio lock* (🔒 / 🔓) located in the *Select Tool*-*Tool Controls* controls how the width and height transform in some cases.

- Edit → Paste Size → Paste Size: The selection is scaled so its *bounding box* matches that of the selection in the *clipboard*. Scaling is around the center of the selection's *bounding box*.
- Edit → Paste Size → Paste Width: The selection is scaled so the width of its *bounding box* matches that of the width of selection in the *clipboard*. The height is left alone if the *Scale ratio lock* is off; otherwise, the height is scaled in the same proportion as the width.
- Edit → Paste Size → Paste Height: The selection is scaled so the height of its *bounding box* matches that of the height of selection in the *clipboard*. The width is left alone if the *Scale ratio lock* is off; otherwise, the width is scaled in the same proportion as the height.
- Edit → Paste Size → Paste Size Separately: Each object in the selection is scaled so its

bounding box matches that of the selection in the *clipboard*. Scaling is around the center of each object's *bounding box*.

- Edit → Paste Size → Paste Width Separately: Each object in the selection is scaled so the width of its *bounding box* matches the width of the selection in the *clipboard*. The heights are left alone if the *Scale ratio lock* is off; otherwise, the heights are scaled in the same proportion as the widths.
- Edit → Paste Size → Paste Height Separately: Each object in the selection is scaled so the height of its *bounding box* matches the height of the selection in the *clipboard*. The widths are left alone if the *Scale ratio lock* is off; otherwise, the widths are scaled in the same proportion as the heights.

Transforms with the XML Editor

Full control over the transformation of an object is available through the *XML Editor* dialog (Edit → XML Editor... (**Shift+Ctrl+X**)). Selecting an object in the document window will bring up the object's attributes in the *XML Editor* dialog. Any transform an object is subject to is described by the *transform* attribute. A transform can be of type “translate”, “scale”, “rotate”, “skewX”, “skewY”, or “matrix”. In most cases, the transform will be of the “matrix” type. A *matrix* entry contains the *Transformation Matrix* in the order (A, B, C, D, E, F) where (ACE) is the first row of the matrix.

One thing to note is that the matrix describes a transformation with respect to the *User Coordinate System* which, if an object is not in a *Group*, is equivalent to the *SVG* coordinate system (*Initial View Port*) where the origin is at the top-left corner of the page (in contrast to the screen coordinate system where the origin is at the bottom-left of the page). See the SVG standard for more details.

Inkscape Coordinates

Table of Contents

Snapping

Get the book.

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Positioning and Transforming » Snapping

Index

Snapping

Major changes for v0.46.

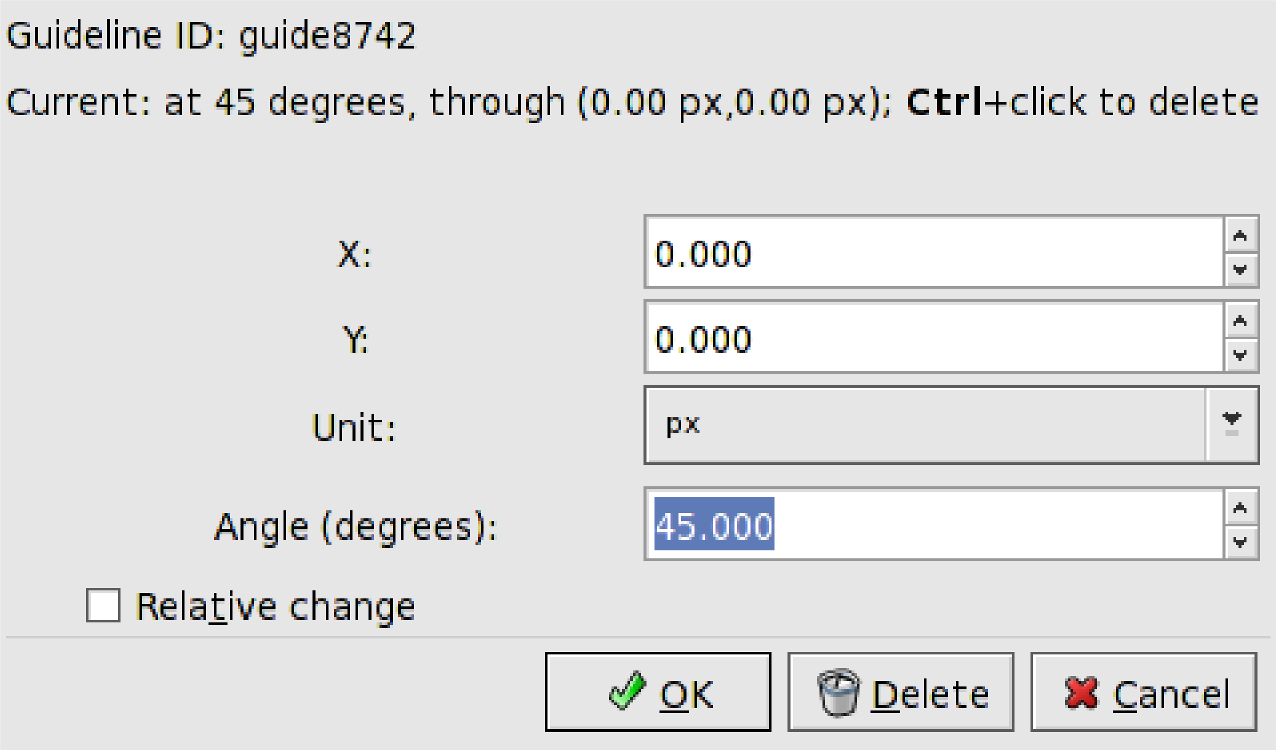
To help precisely place objects on the canvas, an object can be made to *snap* to a target. The target can be a *Grid* line, a *Guide Line*, or another object's path, node, or *bounding box*. Snapping takes place when an object's node or its *bounding box* is near a target. Snapping can be set to always happen or only happen when the object is within a set distance (the *Snap Distance*) from a target. *Snapping* can be divide into two parts: setting the target and determining what can be snapped. All snapping options can be set under the various tabs of the *Document Properties* dialog. Note: Nodes will only snap to other nodes or path, and a *bounding box* will only snap to another *bounding box*. Snapping was introduced in the tutorials at the beginning of the book.

Targets

This section covers creating *Guide Lines* and *Grids*.

Guides

Guide Lines are individual lines that can be arbitrarily placed. To create a *Guide Line*, **Left Mouse Drag** from the left *Ruler* onto the canvas for a vertical *Guide Line* or from the top *Ruler* for a horizontal *Guide Line*. An angled *Guide Line* can be created by dragging from the end of a *Ruler*. By default, the angle is set to 45° if a rectangular *Grid* is displayed or parallel to the angled lines if an axonometric *Grid* is displayed. *Guide Lines* can be moved by dragging them (**Left Mouse Drag**). They can be removed by dragging them to a *Ruler* or clicking on them with the **Ctrl** key held down. Precise placement of a *Guide Line* and changing its angle can be done through the Guideline dialog, opened by double-clicking on the *Guide Line* with the Select Tool, Node Tool, or Tweak Tool. A *Guide Line* is defined by an *x-y* point through which it passes and an angle. The *x* coordinate is ignored for horizontal lines and the *y* coordinate ignored for vertical lines. A *Guide Line* can also be deleted via the dialog. A check box toggles between absolute and relative placement. *Guide Lines* can be snapped. This option is toggled on under the *Guides* tab of the *Document Properties* dialog. *Guide Lines* can also be hidden and their color changed in this dialog. To be active, a *Guide Line* must be visible.



Guideline ID: guide8742

Current: at 45 degrees, through (0.00 px,0.00 px); **Ctrl**+click to delete

X: 0.000

Y: 0.000

Unit: px

Angle (degrees): 45.000

□ Relative change

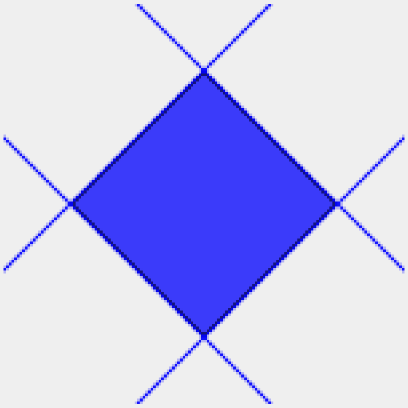
Guide Line dialog showing a line passing through the origin at a 45° angle.

Guide Lines can be created from objects using the Object → Object to Guides (**Shift+G**) command. The keyboard shortcut works with the Select Tool, shape tools, Bezier Tool, and Freehand Tool. Different objects are converted in different ways. In each case, the selected objects are deleted unless the *Keep objects after conversion to guides* entry is checked in the *Tools* section of the Inkscape Preferences dialog.

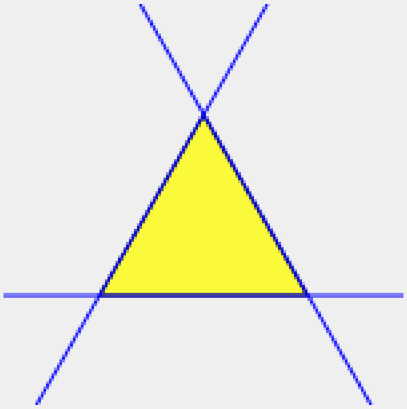
- Rectangles and Boxes: *Guide Lines* are drawn along edges, even when rotated, if the *Conversion to guides uses edges instead of bounding box* box is checked in the Inkscape Preferences dialog. Otherwise the bounding box is used.
- Paths: A *Guide Line* is drawn along each straight line segment.
- Other objects: *Guide Lines* are drawn along bounding box. The *Geometric* or *Visual bounding box* is used depending on which is selected in the *Tools* section of the

Inkscape Preferences dialog.

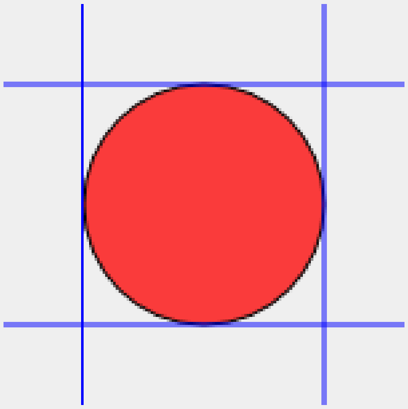
If a *Group* is selected, *Guide Lines* are drawn for each object in the *Group*.



Guide Lines from a rotated *Rectangle*.



Guide Lines from a triangular path, one line for each straight path.

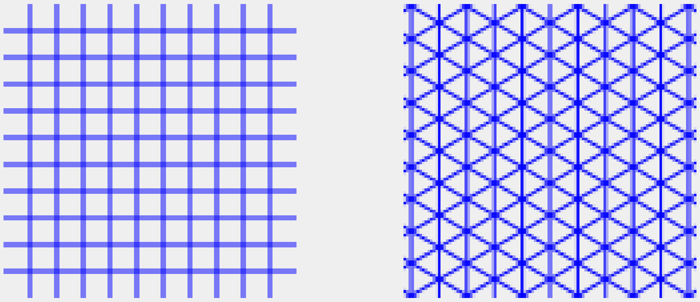


Guide Lines from a circle, lines determined from the *bounding box*.

Grid

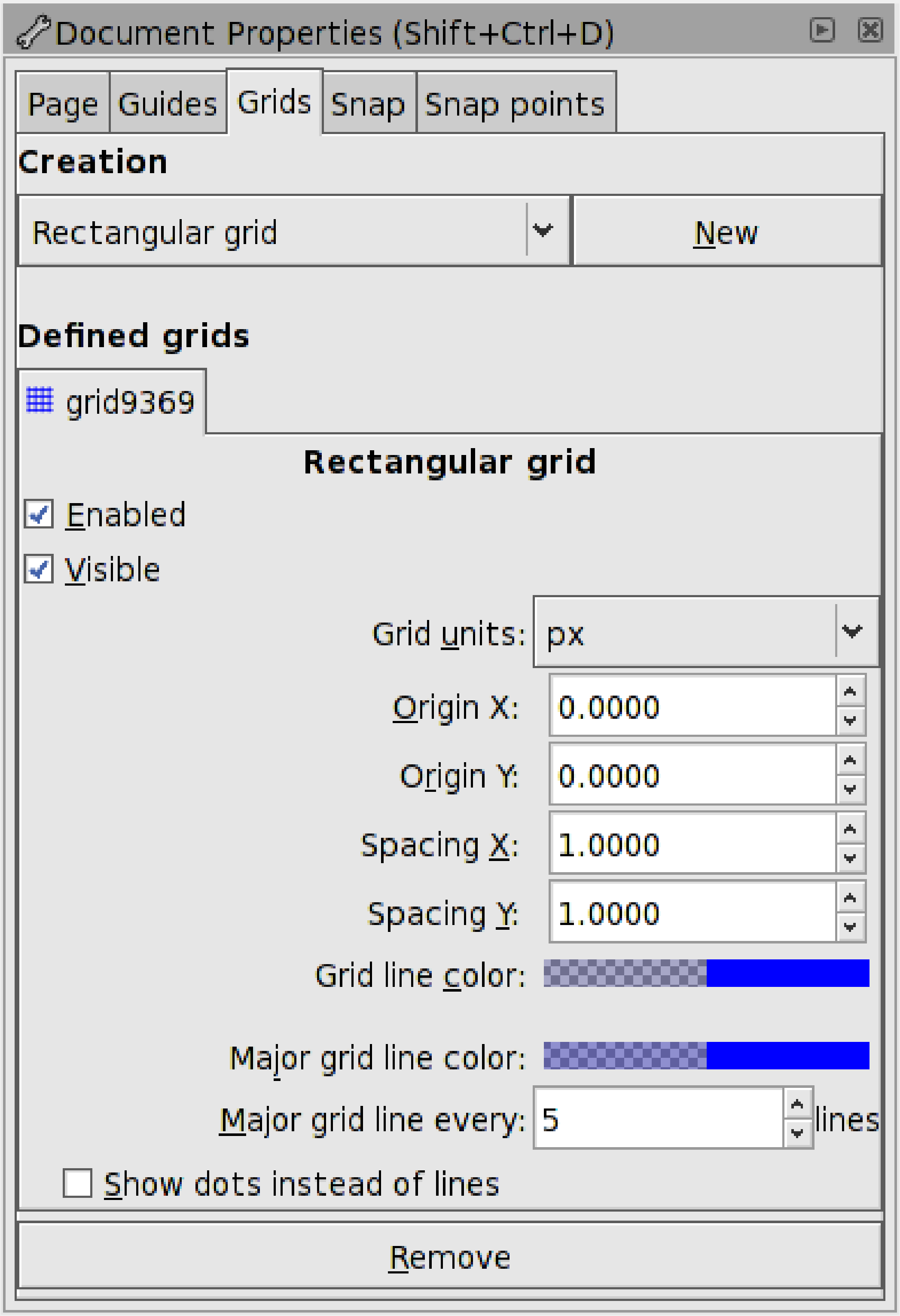
Axonometric grids added in v0.46.

A *Grid* is composed of two or three sets of evenly spaced parallel lines. A *Rectangular Grid* consists of horizontal and vertical lines, much like a sheet of ordinary graph paper. An *Axonometric Grid* consists of three sets of parallel lines, typically one vertical and two at 30° angles from the horizontal. It is often used to draw three-dimensional objects.



Examples of a *Rectangular Grid* (left) and *Axonometric Grid* (right).

Grids can be created and edited on the *Grids* tab of the *Document Properties* dialog. To create a *Grid*, select the type (*Rectangular* or *Axonometric*) from the drop-down menu at the top of the dialog and then click on the *New* button. The parameters for the new *Grid* will then be editable under a tab in the bottom of the dialog. It is possible to have more than one *Grid* defined (and in use). Each *Grid* will have a tab entry.



Rectangular grid

Defined grids

Rectangular grid

☑ Enabled

☑ Visible

Grid units: px

Origin X: 0.0000

Origin Y: 0.0000

Spacing X: 1.0000

Spacing Y: 1.0000

Grid line color: ▨

Major grid line color: ▨

Major grid line every: 5 lines

□ Show dots instead of lines

Grids tab in *Document Properties* dialog showing the parameters for a *Rectangular Grid* with the default parameters.

Each *Grid* can individually be *Enabled* and made *Visible* by the check boxes on the *Grid* tabs. If a *Grid* is not enabled, it will not be visible. However a *Grid* can be enabled and not visible. All *Grids* can be enabled or disabled by using the global command View → Grid (**#**). This setting overrides the settings on the individual *Grid* tabs. Note: There is no visual indication if this overall setting is on or off if the visibility of the individual *Grids* are all set off. Note also that, depending on the zoom scale, not all *Grid* lines are shown. “Missing” lines will not be snapped to.

For both types of *Grids*, the *Grid unit* can be selected from a drop-down menu and the *Grid* origin can also be specified. For *Rectangular Grids*, the *x* and *y* spacing can be set independently. For *Axonometric Grids* the *x* and *z* spacings are derived from the *y* spacing and the angle settings. In both cases the color of the lines can be set by clicking on the color box and making changes in the dialog that pops up.

The default *Grid* parameters can be modified in the *Grids* section of the *Inkscape Preferences* dialog.

Different “views” of the same drawing share the same *Grids* but the *Grids* can be enabled or made visible independently for each view.

Snap Parameters

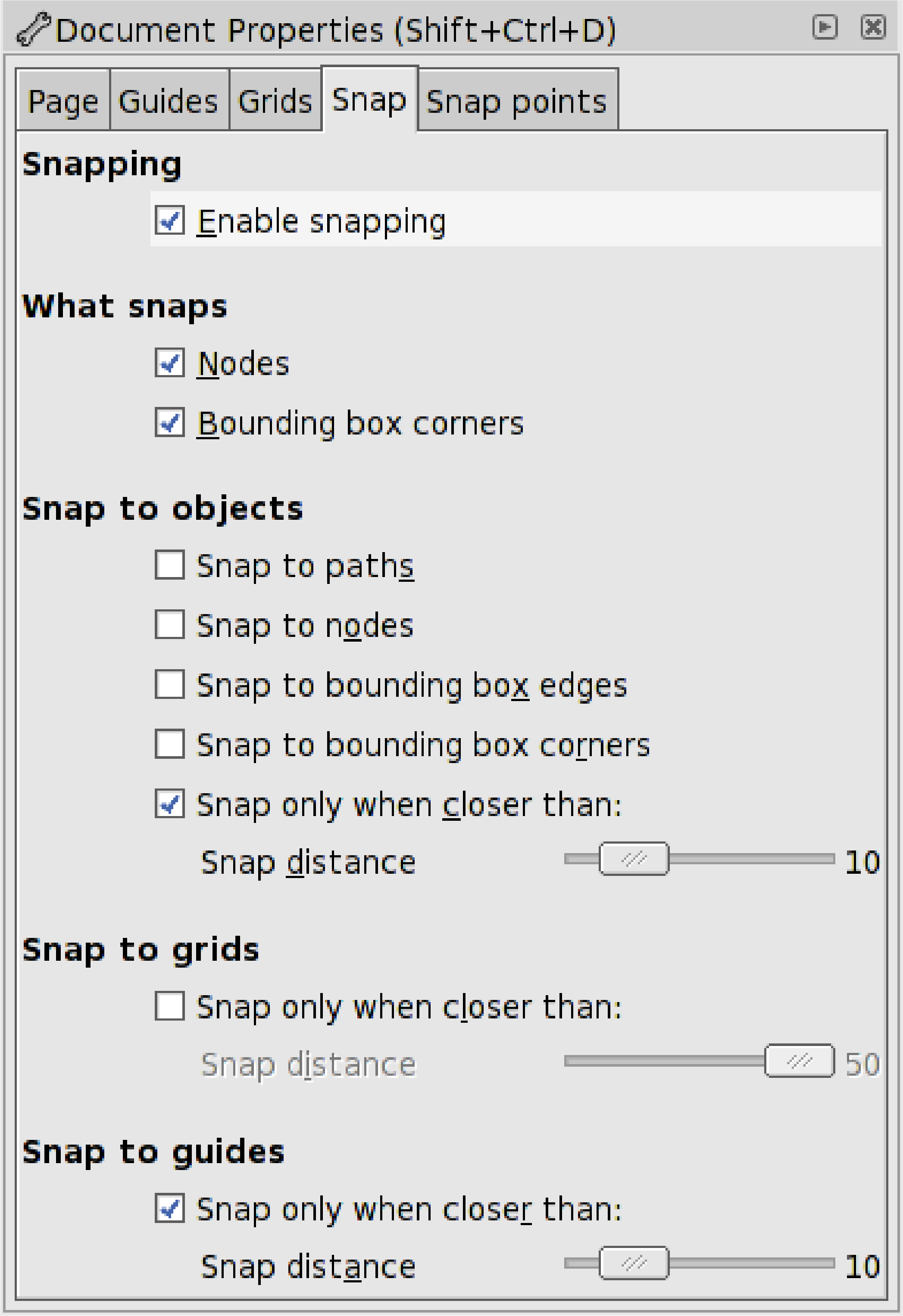
The *Snap* tab of the *Document Properties* dialog is used to control what is snapped to what. It is divided into several sections.

In the first section, *Snapping*, there is a checkbox to control over all snapping. This checkbox supersedes the *Enable* checkboxes on the *Grids* tab. The checkbox can be toggled via the View → Snap (**%**) command.

The next section, *What snaps* controls what points on a moved object or node will snap. The choices are *Nodes* and *Bounding box corners*. Both can be enabled or disabled independently. The *Nodes* mode includes snapping of *Handles*. It does *not* include snapping to smooth nodes in a path. The *bounding box* snapping mode will snap to all four corners of the *bounding box*. The *bounding box* used is either the *Visual* *bounding box* or the *Geometric* (nodes only) *bounding box*, depending on the setting in the *Tools* section of the *Inkscape Preferences* dialog. See the introduction to this chapter for more details.

The third section governs what parts of an object to snap to. Choices include: *paths*, *nodes*, *bounding box* *edges*, and *bounding box* *corners*. Snapping will always occur if one or more of these items are selected unless the *Snap only when closer than* box is checked. If the box is checked, then snapping will only occur if the part to be snapped is closer than the *Snap* *distance* to the thing to be snapped to. The *Snap distance* is in units of screen pixels.

The last two sections govern snapping to *Grids* and *Guide Lines*. If snapping to a *Grid* or *Guide Lines* is enabled, the snapping will always occur unless the *Snap only when closer than* box is checked. If the box is checked, then snapping will only occur if the part to be snapped is closer than the *Snap distance* to a *Grid* line or a *Guide Line* (or another snapping rule is satisfied). The *Snap distance* is in units of screen pixels.



Document Properties (Shift+Ctrl+D)

Snapping

☑ Enable snapping

What snaps

☑ Nodes

☑ Bounding box corners

Snap to objects

□ Snap to paths

□ Snap to nodes

□ Snap to bounding box edges

□ Snap to bounding box corners

☑ Snap only when closer than:

Snap distance 10

Snap to grids

□ Snap only when closer than:

Snap distance 50

Snap to guides

☑ Snap only when closer than:

Snap distance 10

Snap tab in the *Document Properties* dialog.

The last tab, *Snap points*, of the *Document Properties* dialog has options for special cases of snapping. The first section allows toggling on/off snapping to intersections (*Grid* lines with *Guide Lines* or line segments in paths) and the second section covers toggling on/off snapping to the *Rotation center* of an object.

Transformations

Table of Contents

Alignment and Distribution of Objects

Get the book.

Inkscape: Guide to a Vector Drawing Program

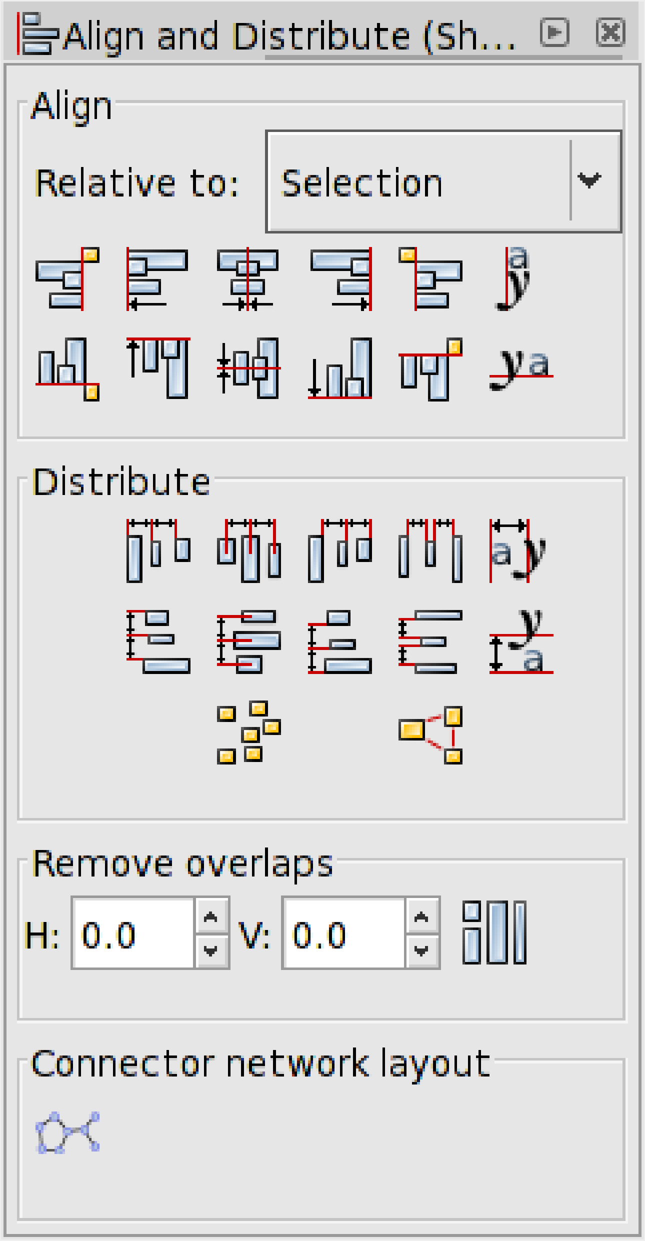
Get the book.

Inkscape » Positioning and Transforming » Alignment and Distribution of Objects

Index

Alignment and Distribution of Objects

This section describes the *Align and Distribute* dialog (Object → Align and Distribute... (**Shift+Ctrl+A**)) which allows the positioning of objects with respect to other objects or to the selection, drawing, or page. Two types of positioning are available: *alignment* where the centers or edges of objects are aligned to one another, and *distributing* where objects are distributed in some direction based on their centers or edges.



Relative to: Selection

H: 0.0 V: 0.0

Align and Distribute dialog.

Align

The alignment of objects is with respect to an *anchor*. The anchor can be an object or be defined by the selection, drawing, or page; the choice of what is used is set in the pull-down menu *Relative to*. The object may be specified to be the first or last object selected. If multiple objects are selected at the same time, this is equivalent to the top or bottom object in *z-order*. Alternatively, the object can be specified to be the biggest or smallest item, where size is determined by the size of the *bounding box*, the width for vertical alignment and the height for horizontal alignment. The *bounding box* sides are used as the reference points for alignment except for text, where the *baseline* is used.

The various alignment options are:

- Horizontal:
  - Align right sides to left side of anchor.
  - Align right sides to right side of anchor.
  - Align horizontal centers to center of anchor.
  - Align left sides to left side of anchor.
  - Align left sides to right side of anchor.
  - Align baselines of text to anchor, horizontally.
- Vertical:
  - Align top sides to bottom side of anchor.
  - Align top sides to top side of anchor.
  - Align vertical centers to center of anchor.
  - Align bottom sides to bottom side of anchor.
  - Align bottom sides to top side of anchor.
  - Align baselines of text to anchor, vertically.

Distribute: Uniform

The distribute part of the *Align and Distribute* dialog allows objects to be evenly spaced in the horizontal or vertical direction based on some criteria. Two options are included in this part of the dialog, which perhaps should be included separately: randomizing the center of objects and unclumping. They will be considered in the next section.

The distribution of objects is between the two objects at the extremes (i.e., the leftmost and rightmost objects for horizontal distribution). The definition of which is the leftmost and rightmost object is made using the objects' *bounding boxes*, and it may depend on the type of distribution selected. For example, if a distribution is based on the rightmost edge of the objects, then the objects rightmost edge will be used to determine which objects are at the extremes.

The various distribution options are:

- Horizontal:
  - Distribute left sides evenly.
  - Distribute centers evenly.
  - Distribute right sides evenly.

  - Distribute with uniform gaps between objects.
  - Distribute baseline anchors evenly.
- Vertical
  - Distribute left sides evenly.
  - Distribute centers evenly.
  - Distribute top sides evenly.
  - Distribute with uniform gaps between objects.
  - Distribute baseline anchors evenly.

Distribute: Non-Uniform

The distribute part of the *Align and Distribute* dialog has two options that modify the distribution of objects in a non-uniform way. Both affect the horizontal and vertical distributions at the same time.

The two options are:

- Randomize the center of objects.
- *Unclump* objects (i.e., move objects to more evenly space the edge-to-edge distances). Repeated application approaches the use of the *Distribute with uniform gaps* commands described above.

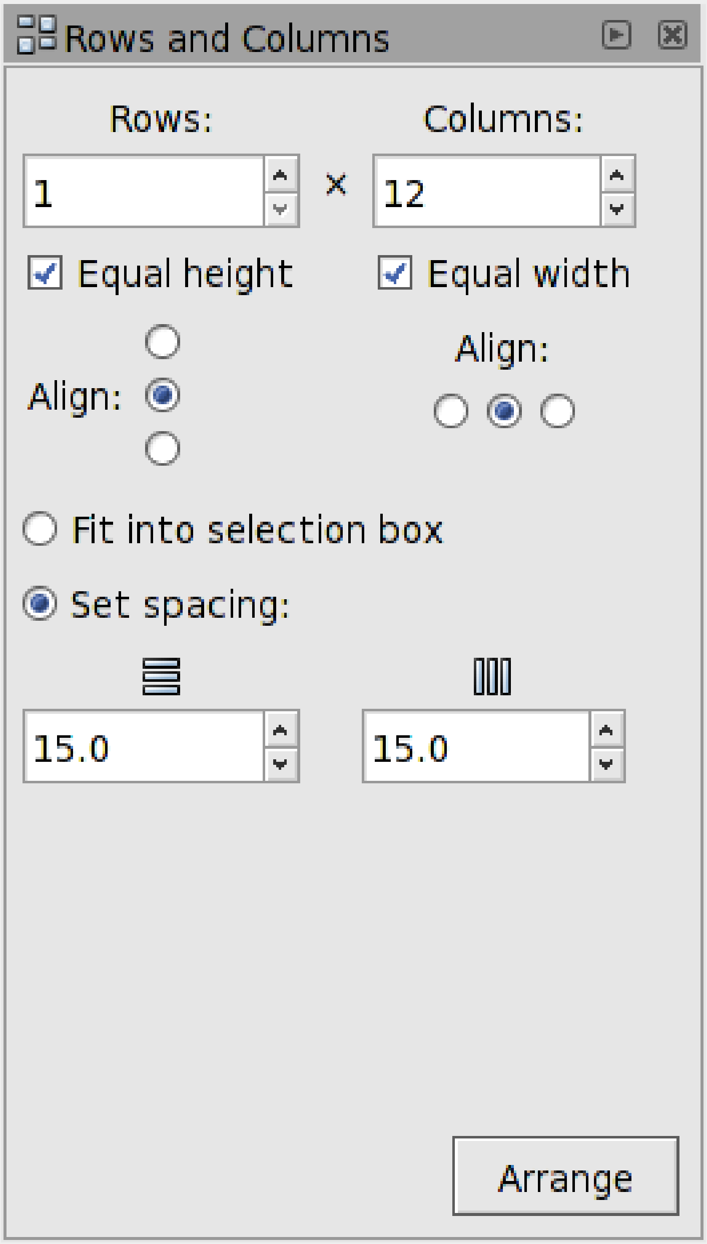
Distribute: Remove Overlaps

Another section of the *Align and Distribute* dialog allows one to move objects just enough that they don't overlap. Two entry boxes, one for the horizontal direction and the other for the vertical direction, allow the addition of a minimum space between adjacent objects.

Rows and Columns

Used to be called Grid Arrange.

The *Rows and Columns* dialog (Object → Rows and Columns...) can be used to arrange an arbitrary number of objects into a two-dimensional grid.



Rows and Columns

Rows: 1 × Columns: 12

☑ Equal height ☑ Equal width

Align: ○ ● ○ Align: ○ ● ○

○ Fit into selection box

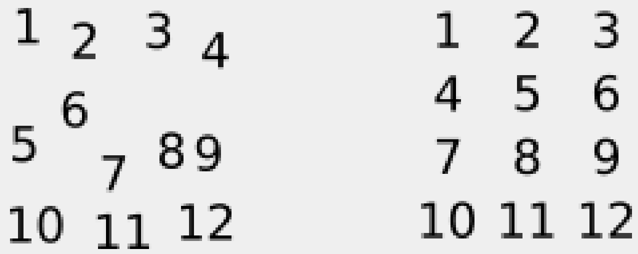
● Set spacing:

15.0 15.0

Rows and Columns dialog.

To use the dialog, first select all the objects you wish to arrange into a grid. The dialog will default to a one-dimensional array. You can use the *Row* and *Column* entry boxes to change the number of rows and columns. If you use the up/down arrows to change the number of rows, the number of columns will change to give you the minimum number required to include all the objects in the selection. A similar change to the number of rows will happen if you change the number of columns. The maximum number of rows or columns is 100.

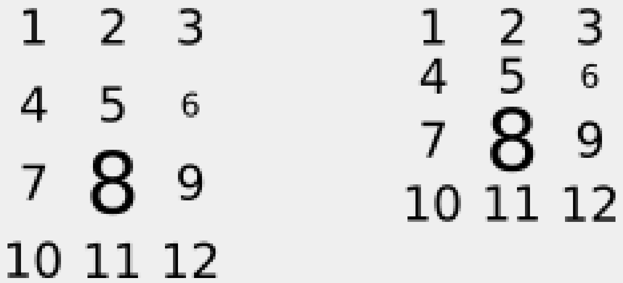
The algorithm for determining the *order* the objects are placed in the array attempts to preserve any existing rows. For this algorithm, the *bounding box* of each object is used. Technically, the objects are first sorted by their vertical positions. Then objects that overlap vertically are sorted by their horizontal positions. Finally, the objects are placed from left to right and from top to bottom in the array. The algorithm works pretty well but cannot handle all situations since determining what objects are in a row is objective. (If in the vertical direction A overlaps B and B overlaps C but A does not overlap C, are A and C in the same row?)



Twelve object before (left) and after (right) aligning with a 3 by 4 grid.

For placing objects, the grid is divided into cells. First, the cell size and placement is determined and then the objects are positioned inside the cells, one object to one cell.

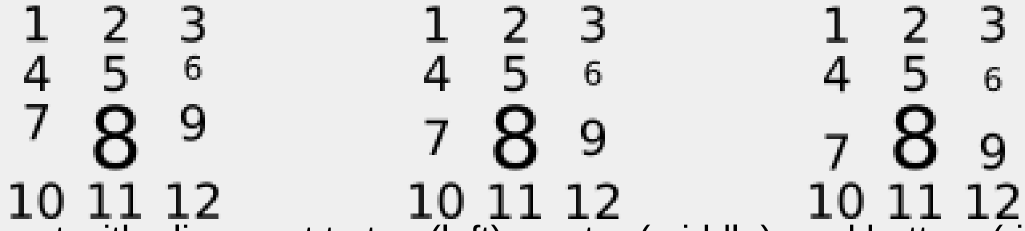
Cells are given the height of the tallest object if the *Equal height* box is checked; otherwise, they are given the height of the tallest object in their row. A similar policy is followed for width.



Alignment with the *Equal height* box checked (left) and unchecked (right). Note how the height of the '8' has been used for all rows on the left and just its own row on the right.

If the *Fit into selection* option is selected, the rows and columns of cells are evenly spaced with the edge rows and columns flush against the bounding box of the selection. If the *Set spacing* option is selected, the rows and columns are separated by the amount entered in the *Row spacing* and *Column spacing* entry boxes. The spacing can be negative.

Once the cell positions have been determined, the objects are placed inside the cell according to the selected *Align* options (top, middle, bottom; left, center, right).



Grid alignment with alignment to top (left), center (middle), and bottom (right) of cell.

Note that the *bounding box* of all the objects after alignment may not be the same as the *bounding box* of the selection prior to alignment even though the *Fit to selection* option has been chosen. This is because the selection *bounding box* has been used to place the *cells*. The objects within the cells may not touch the cell walls. (There also appears sometimes to be a gratuitous shift: Bug?)

← ↑ →

Snapping

Table of Contents

Chapter 6. Geometric Shapes

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Geometric Shapes » Ellipses, Circles, and Arcs

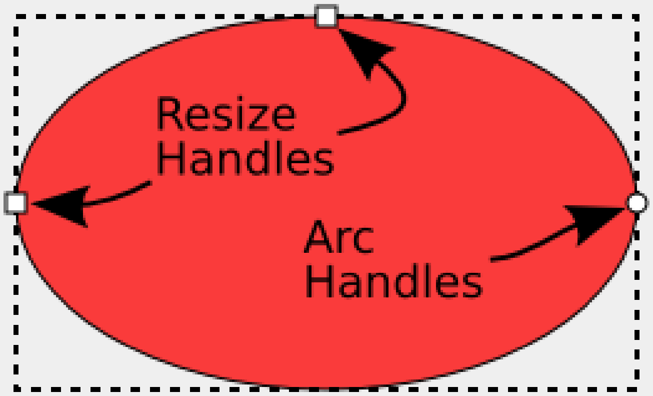
Index

Ellipses, Circles, and Arcs

The *Ellipse Tool* allows one to draw ellipses, circles, and arcs. Select the tool by clicking on the icon (**F5** or **e**) in the *Tool Box*. To draw an ellipse or arc, use a **Left Mouse Drag**. An ellipse will be drawn with the sides touching a rectangular box defined by the starting and stopping points of the drag. To force a circle to be drawn, hold down the **Ctrl** while dragging the mouse. This also allows ellipses with an integer height to width or width to height ratio to be drawn. Holding the **Shift** key down while dragging will create an ellipse centered around the starting point. Holding down the **Alt** key down while dragging will create an ellipse with the circumference passing through the start and end points of the drag (as of v0.46). Using **Alt**+**Ctrl** while dragging will create a circle with a diameter defined by the distance between the start and stop point of the drag.

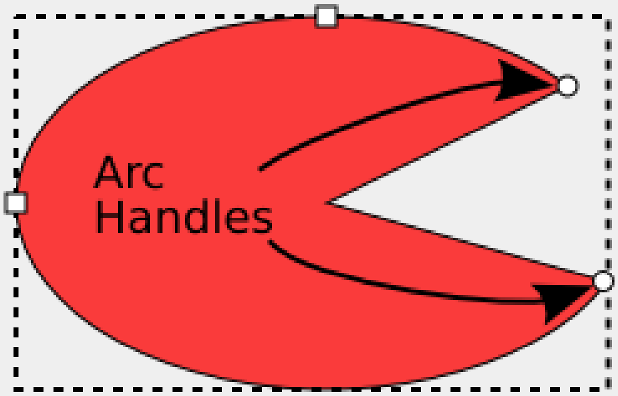
When an ellipse is selected and the *Ellipse Tool* is active, the ellipse will have a set of handles (small squares and circles) that can be used to resize it or convert it to an arc. (The handles are also available if one of other shape *Tools* or the *Node Tool* is active.)

To change the size of the ellipse, drag the handle at the top or left. Again, the **Ctrl** key can be used to force the ellipse to be a circle (or have an integer height to width or width to height ratio).



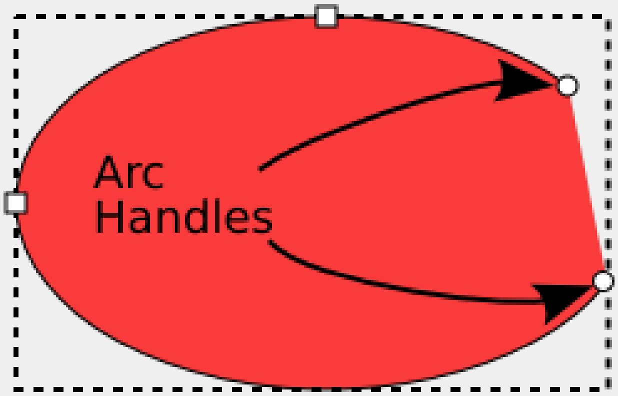
An ellipse showing the *Resize* and *Arc* handles.

To convert an ellipse into an arc, use the two *Arc* handles. Initially both handles are on top of each other. Drag one handle to set one end of the arc, then drag the second handle to set the other end. Holding down the **Ctrl** key while dragging an *Arc* handle will force the angle of the arc to begin or end at a multiple of the *Rotation snap* angle (15 degrees by default).



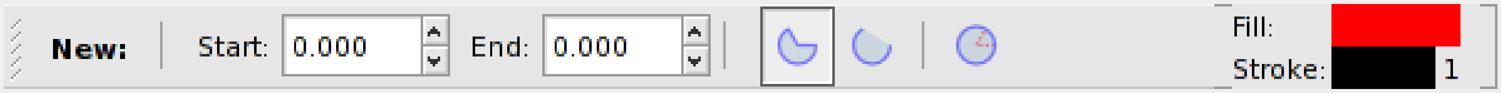
An arc showing the *Resize* and *Arc* handles.The path of the arc is closed by an extension to the point at the center of the arc's curvature.

If an *Arc* handle is dragged with the pointer outside the curve of the virtual ellipse, the arc will be defined with a closed path that has a wedge which extends to the center of curvature (as shown above). If the *Arc* handle is dragged with the pointer inside the curve, the path defining the arc will start and stop at the two *Arc* handles, as shown below.



An open arc.

An arc can also be defined using the settings in the *Tool Controls* when the *Ellipse Tool* is selected. The settings will affect any ellipse or arc that is selected as well as any that are drawn afterward. The *Start* and *End* angles are defined in degrees and are measured in the *clockwise* direction starting at the *x*-axis. There are two buttons that toggle arcs between opened (◔) and closed (◔) (switchable if either angle is not zero). There is also a button to reset an arc to an ellipse (◷).



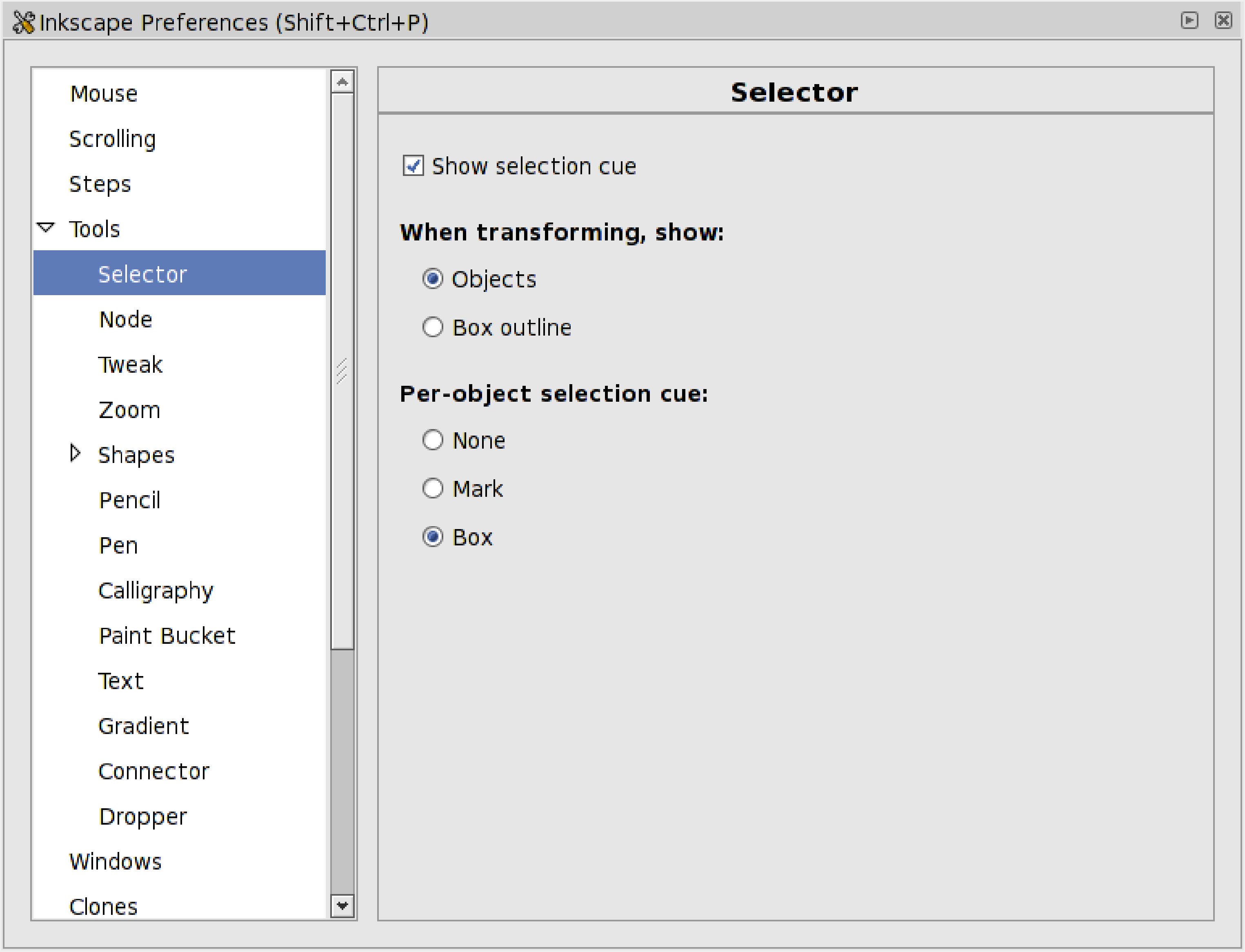
New: Start: 0.000 End: 0.000 Fill: Stroke: 1

The *Ellipse Tool*-*Tool Controls*.

3D Boxes
Table of Contents
Regular Polygons and Stars

Inkscape Preferences Dialog

Inkscape is most easily customizable through the *Inkscape Preferences* dialog (File → Inkscape Preferences... (**Shift+Ctrl +P**)).



Selector

☑ Show selection cue

When transforming, show:

● Objects

○ Box outline

Per-object selection cue:

○ None

○ Mark

● Box

Inkscape Preferences Dialog.

There are too many options in the dialog to cover here. It is worthwhile to scan through the options under each entry. The most important options have already been mentioned in the text. They include: Setting the *Rotation snap* angle and other scaling parameters under the *Step* entry. Determining if transformations should be *Optimized* or *Preserved* under the *Transforms* entry.

Setting the *Default export resolution* under the *Misc* entry.

All the preference are stored in the file `.inkscape/preferences.xml`. There are quite a few preferences, some undocumented, that are accessible *only* by hand editing this file. See the next section for some of these.

Chapter 20. Customization

Table of Contents

Inkscape Configuration Files

Inkscape: Guide to a Vector Drawing Program

Get the book.

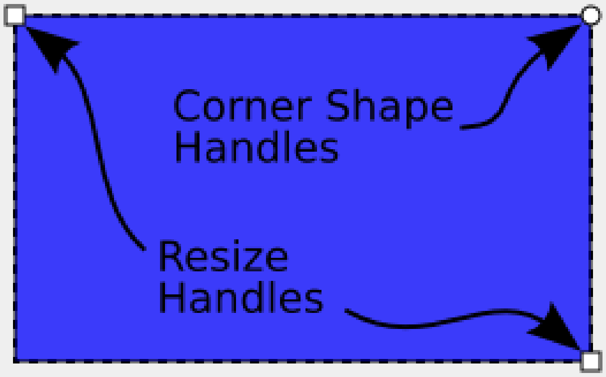
Inkscape » Geometric Shapes » Rectangles and Squares

← Index →

Rectangles and Squares

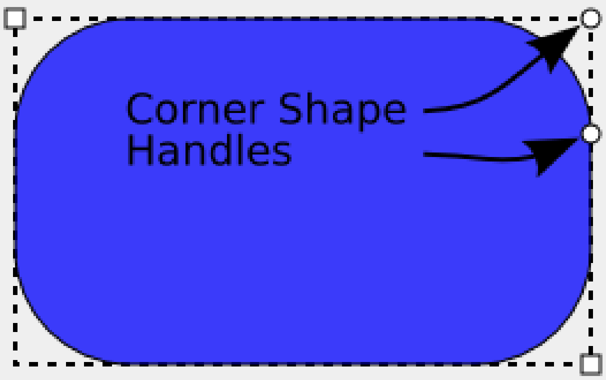
The *Rectangle Tool* allows one to draw rectangles and squares. Basic use of the tool is introduced in Chapter 1, *Quick Start*. To draw a rectangle or square, select the tool by clicking on the □ icon (**F4** or **r**) in the *Tool Box*. To draw a rectangle, use a **Left Mouse Drag** from one corner to the opposite corner. To force a square to be drawn, hold down the **Ctrl** while dragging the mouse. This also allows rectangles with an integer height to width or width to height ratio to be drawn. As a special case, rectangles with sides constrained to the "Golden Ratio" are also allowed with the **Ctrl** key (as of v0.46). Holding the **Shift** key down while dragging will create a rectangle centered around the starting point.

The size of a preexisting rectangle can be changed by selecting the rectangle via a **Left Mouse Click** on the rectangle with the *Rectangle Tool*. Once the rectangle is selected, handles (small squares and circles) will appear at some of the rectangle's corners. **Left Mouse Drag** the handle (square) at the top left or bottom right to change the size of the rectangle.

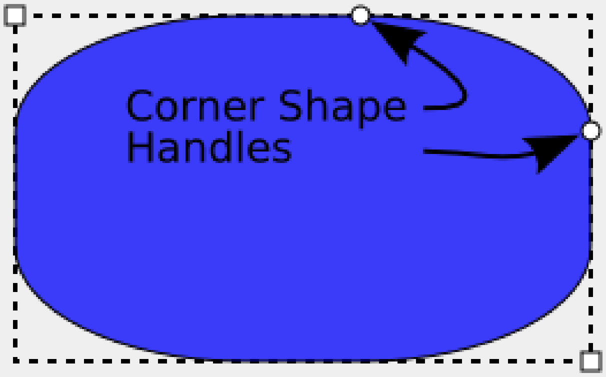


A rectangle showing the *Resize* (square) and *Corner Shape* (circle) handles.

A rectangle can be given rounded corners. There are two ways to do this. The first is to use the handle(s) at the top-right corner of the rectangle. Initially, only one handle is visible. If this handle is dragged down, a rounded corner in the shape of a quarter circle is created. A second handle is now visible. Dragging this second handle to the left will create an elliptical rounded corner. Upon dragging the second handle, the radius of the curvature in the horizontal (*x*) and vertical (*y*) directions are independent.

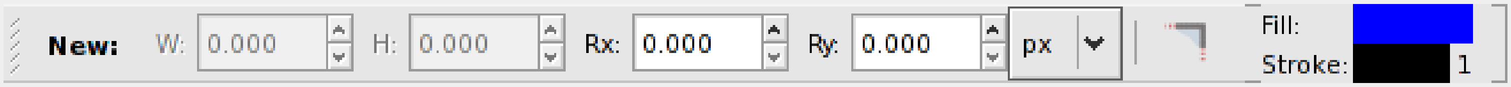


A rectangle with circular rounded corners.



A rectangle with elliptical rounded corners (different x and y corner radii).

A second way to set the radius of curvature of a rectangle is to use the settings *Rx* and *Ry* in the *Tool Controls* when the *Rectangle Tool* is selected. The settings will affect any selected rectangle as well as rectangles that are drawn afterward. Rounded corners can be removed from a rectangle by using the *Square Corners* button (⌝). The width and height of a rectangle can also be set using entry boxes in the *Tool Controls*.


New: W: 0.000 H: 0.000 Rx: 0.000 Ry: 0.000 px Fill: Stroke: 1

The *Rectangle Tool*-*Tool Controls*.

Note

If you want the rounded corners to scale with the rectangle object, you must toggle on this option using the icon that is in the *Tool Controls* when the *Select Tool* is in use.

Chapter 6. Geometric Shapes

Table of Contents

3D Boxes

Get the book.

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Geometric Shapes » 3D Boxes

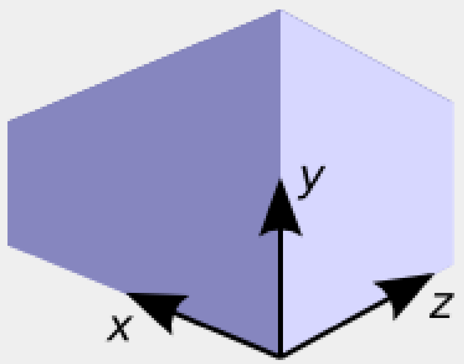
← Index →

3D Boxes

New in v0.46.

The *Box Tool* allows one to draw three-dimensional boxes that remain editable in Inkscape but display normally in other *SVG* renderers. A box is composed of an *SVG Group* of six paths. Information about the vanishing points, etc. are stored in the Inkscape *Name Space*. This extra information is only used by the *Box Tool*. All other tools treat the box as a normal *Group*. The sides of the box can be styled independently (or even deleted).

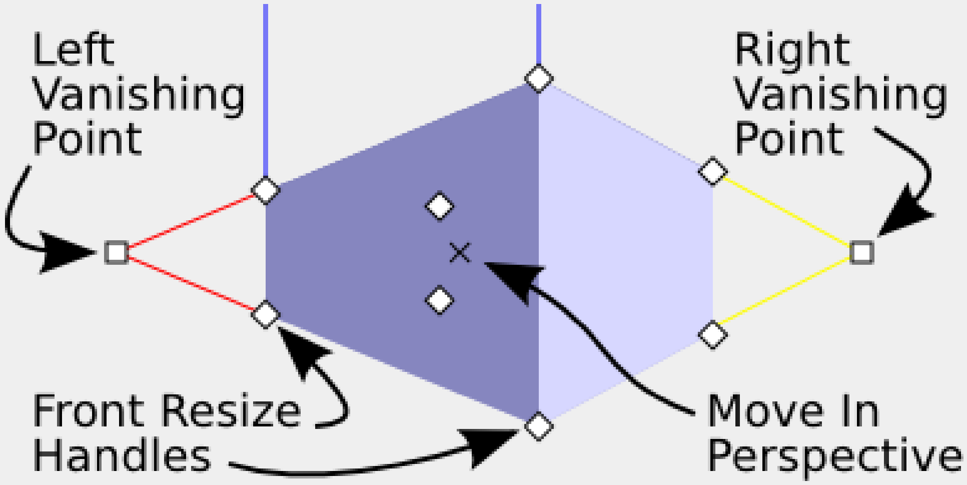
To draw a 3D box select the *Box Tool* by clicking on the icon (**Shift**+**F4**) in the *Tool Box*. Use a **Left Mouse Drag** to draw the left side of the box (in the *x*-*y* plane). The start of the drag sets one corner while the end of the drag sets the opposite corner. The other sides of the box are automatically drawn with the right side of the box set to a default width. Pressing the **Shift** while creating the box changes the function of the cursor to defining the depth (width of the right side or *z* dimension) of the box.



A simple 3D box showing two sides and the box coordinate system. There are an additional four hidden sides.

When a box is selected and the *Box Tool* active, a variety of handles are displayed. The eight handles at the corners of the box are used to adjust the size of the box. The four in front (see figure below) change the size of the left box face in the *x*-*y* plane. The other four change the depth (*z*) of the box. Holding the **Shift** down swaps the functions of the handles. With the **Ctrl** down, the handles are restricted in movement to lines along the box edges or to a box diagonal. This allows adjusting one dimension of a box face while keeping the other fixed in the first case or keeping the aspect ratio fixed in the latter case.

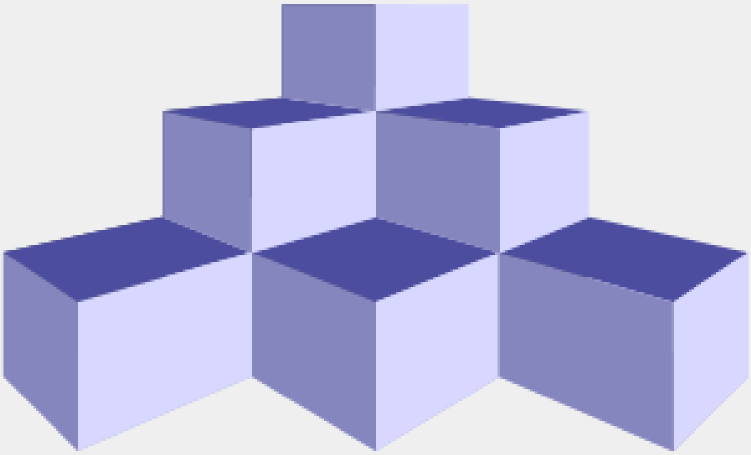
Dragging the *Cross* handle moves the box while keeping the same perspective. Without a modifier key, the box is kept in the *x*-*y* plane. Holding the **Ctrl** down while dragging limits movement to lines along the box edge or along the box diagonal. Holding down the **Shift** while dragging moves the box in the *z* direction.



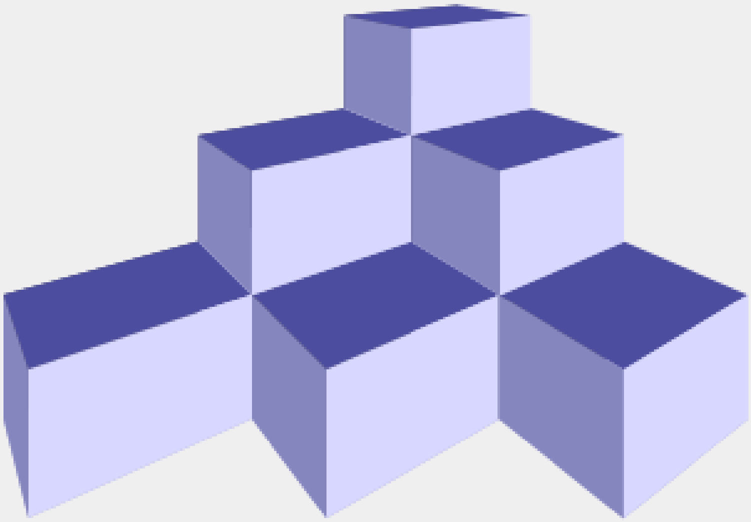
A simple 3D box showing showing its handles. Two of the *Front Resize* handles are labeled. The other two are directly above those. The four remaining handles at the corners of the box are the *Back Resize* handles. The cross in the middle is use to move the box while keeping the same perspective.

By default, a box is drawn with two vanishing points, one each on the left (*x*) and right (*z*) sides. The vanishing points are initially placed at the edge of the page, half way between the top and bottom. These points are determined when the SVG drawing is first created so resizing the page does not move them. The vanishing points can be dragged to new locations. Dragging the points a ways off the page will probably give you a more satisfactory perspective than the default.

All boxes that share the same vanishing points will change together. If you wish to change the vanishing points of just selected boxes, hold down the **Shift** while dragging. If multiple boxes are selected with different vanishing points, dragging a vanishing point for one box near that of another box will “merge” the points together.



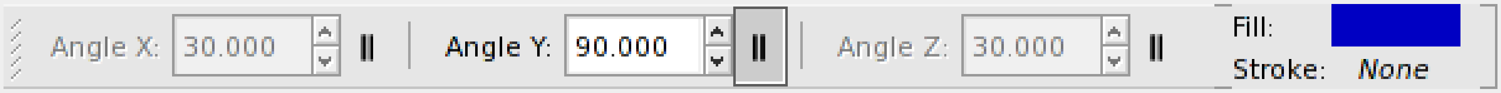
All the boxes share the same vanishing points. The “stack” was created by duplicating the bottom-center box and dragging it using the **Shift** key or **Ctrl** key. Two moves were required for each box.



The same boxes as in the previous figure but with both the *x* and *z* vanishing points moved up and right.

Perspectives

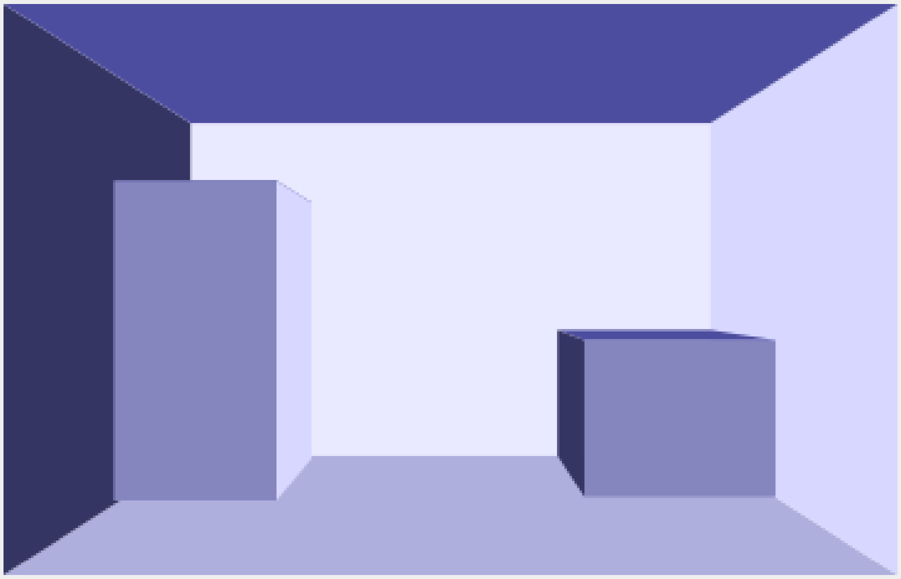
The *Box Tool* can be used to draw boxes with 1, 2, and 3-point perspectives as well as a box using an isometric projection. The above examples use a 2-point perspective with two vanishing points. The type of perspective is changed via the *Box Tool* *Tool Controls*. Each of the three perspective points (*x*, *y*, and *z*) can be set to infinity or to a specific point. To set or unset a perspective point to infinity, toggle the “Parallel Lines” button (‖) in the *Tool Controls* next to the appropriate angle (or use the keyboard shortcuts: **Shift**+**X**, **Shift**+**Y**, and **Shift**+**Z** that are available when using the *Box Tool*). When set to infinity, the direction of a perspective point is set by an *Angle* parameter. The angles can be changed via the entry boxes in the [Tool Controls](#) or by using the keyboard shortcuts: *x*: **[**, **]**; *y*: **(**, **)**; and *z*: **{**, **}**. The angles will be changed by the *Rotation snap angle* (15° by default, settable in the *Steps* section of the *Inkscape Preferences* dialog). With the **Alt**, the angle change will be 0.5°.



Angle X: 30.000 ‖ Angle Y: 90.000 ‖ Angle Z: 30.000 ‖ Fill: Stroke: None

The *Box Tool*-*Tool Controls*, showing the default parameters for a 2-point perspective.

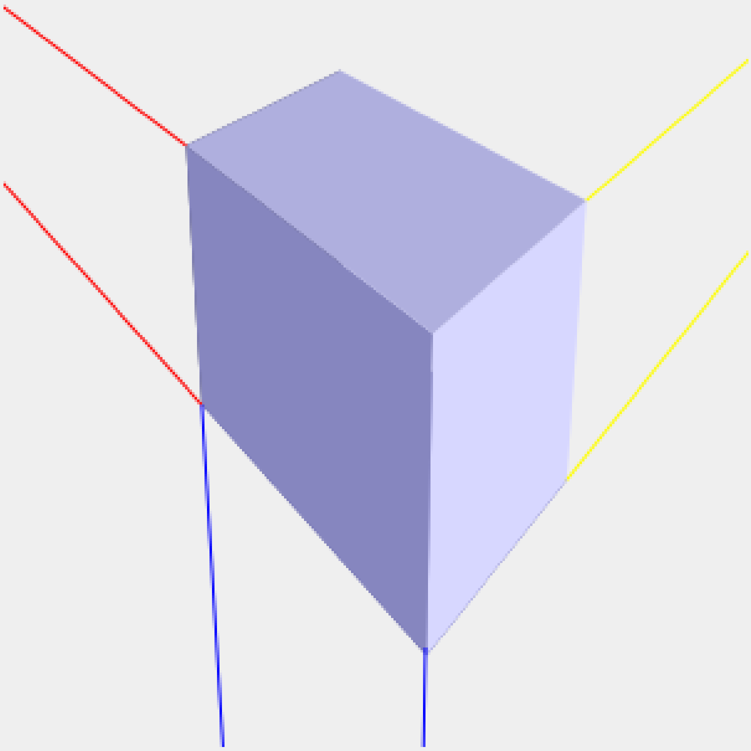
1-point Perspective: Set the *x* and *y* vanishing points to infinity by toggling on the *x* and *y* “Parallel Line” buttons (‖) in the *Tool Controls*. Set *Angle X* to 180° and *Angle Y* to 90°. Toggle off the “Parallel Line” button for *z* and drag the vanishing point to the desired point (typically near the center of the drawing).



The boxes have been drawn using a 1-point perspective. The front side of the outer box has been made transparent.

2-point Perspective: The default perspective. Set the *y* vanishing point to infinity by toggling on the *y* “Parallel Line” button (‖) in the *Tool Controls*. Set *Angle Y* to 90°. Enable the *x* and *z* vanishing points by toggling off their “Parallel Line” buttons. Drag the *x* and *z* vanishing point to the desired points (typically at the same level on opposite sides of the page).

3-point Perspective: Enable the *x*, *y* and *z* vanishing points by toggling off their “Parallel Line” buttons (‖) in the *Tool Controls*. Drag the *x*, *y* and *z* vanishing point to the desired points. Typically the *x* and *z* vanishing points are at the same level on opposite sides of the page. They are either above or below the page depending on if the observer is looking down or up at the scene. The *y* vanishing point is then on the opposite side, either below or above the page.



The box has been drawn with a 3-point perspective. The vanishing points have been dragged off the page as indicated by the red, yellow, and blue lines. Dragging the *y* vanishing point below the others turns the box inside out. The proper perspective can be achieved by changing the *z-order* of the sides or by swapping the *x* and *z* vanishing points.

Isometric Projection: Boxes can be drawn with an *Isometric Projection* by toggling on all “Parallel Line” buttons in the *Tool Controls* and setting the *x*, *y*, and *z* angles to be: 150°, 90°, and 30° respectively.

The perspective information is stored in the *defs* section of the *SVG* file. Look for the “inkscape: perspective” tag. With the *XML Editor* one can precisely place vanishing points. There is a triplet of numbers for each point (e.g. "inkscape:vp_x"). The first two are the *x* and *y* coordinates of the vanishing point and the third is a flag to indicate if the perspective lines converge or are parallel.

Attributes

The attributes of the six sides can be changed independently. To select one of the sides, one must use one of the methods of selecting an object in a *Group*. There is one problem: the common method for entering a *Group*, namely double clicking on an object with the *Select Tool*, won't work as this enables the *Box Tool* instead. The *Group* can be entered by selecting the box and using **Shift**+**Enter** or by selecting the *Enter group* line from the pop-up menu when you **Right Click** on the box. Using **Ctrl**+**Left Mouse Click** will select a side of the box without entering the *Group*. Using **Ctrl**+**Alt**+**Left Mouse Click** to select a hidden side.

By default, a box will always be drawn with sides of the same default colors (shades of blue), even after editing the attributes for one side. This behavior can be changed by selecting the *Last used style* option on the *3D Box* page in the *Inkscape Preferences* dialog. If this option is selected, Inkscape will remember the *Fill* and *Stroke* colors independently for each side. Changing the color of one side will change the *Current style* for all other shapes but the inverse is not true.

⚠ **Warning**

Inkscape does not prevent you from creating a box inside another box *Group*. This can lead to strange behavior.

← Rectangles and Squares

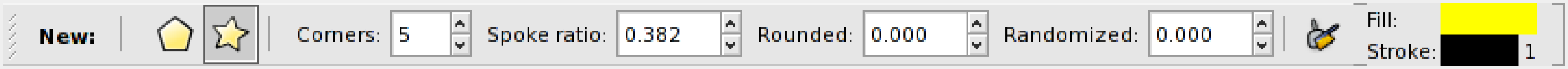
↑ Table of Contents

→ Ellipses, Circles, and Arcs

Regular Polygons and Stars

The *Star Tool* can be used to draw regular polygons and stars. To draw one of these objects, select the tool by clicking on the ☆ icon (**Shift**+**F9** or *) in the *Tool Box*. To draw a polygon or star, use a **Left Mouse Drag**. A star will be drawn with the center at the starting point of the drag and one vertex at the ending point of the drag. The vertex can be forced to be at a multiple of the *Rotation snap* angle (15 degrees by default) by holding down the **Ctrl** key during the drag.

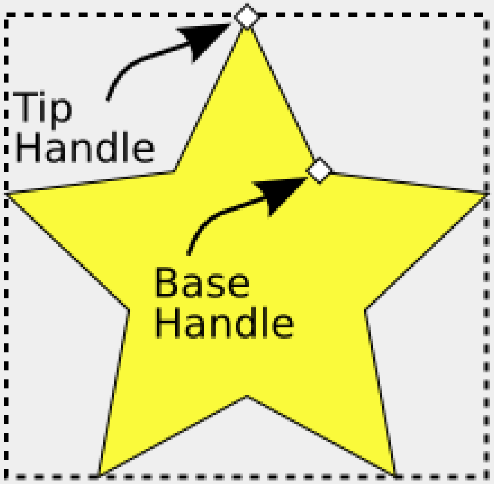
Stars can be reshaped by either dragging handles (small diamonds) on the star or by using the settings in the *Tool Controls* when the *Star Tool* is selected. Two important parameters can only be changed in the *Tool Controls*. The first is an option to specify that the shape drawn be a star or a polygon. This is controlled by two toggled buttons (⬠ and ☆). The second is a parameter that controls the number of points in a star or the number of corners of a polygon.



New: ⬠ ☆ Corners: 5 Spoke ratio: 0.382 Rounded: 0.000 Randomized: 0.000 Fill: 🟨 Stroke: ■ 1

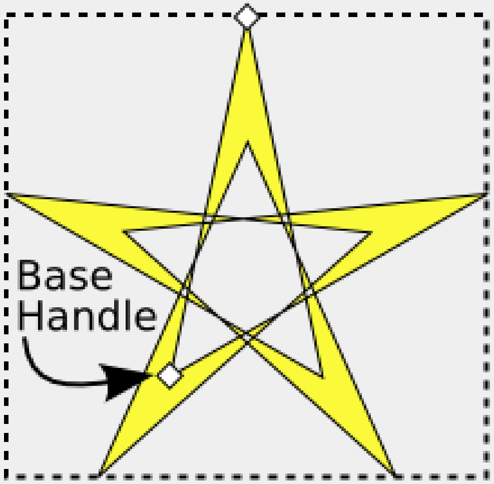
The *Star Tool*-*Tool Controls*.

There are two handles for stars (one for polygons). The *Tip* handle (see the *Notification Region* if in doubt which is which) is used to control the position of the tip of a star or corner vertex of a polygon. This is the handle that was used when first drawing the star or polygon. Using the **Ctrl** key while dragging the handle, restricts it to a radial line.



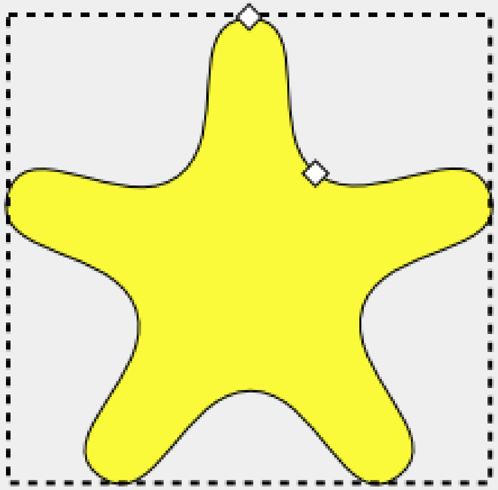
A star showing the *Tip* and *Base* handles.

The *Base* handle controls the position of the “inner” vertex of a star. The *Base* handle can be constrained to have an angle halfway between adjacent tips by holding down the **Ctrl** key while dragging it. Note that it is possible that the radius of the *Base* vertex be larger than the *Tip* vertex or it can be *negative* as shown next.



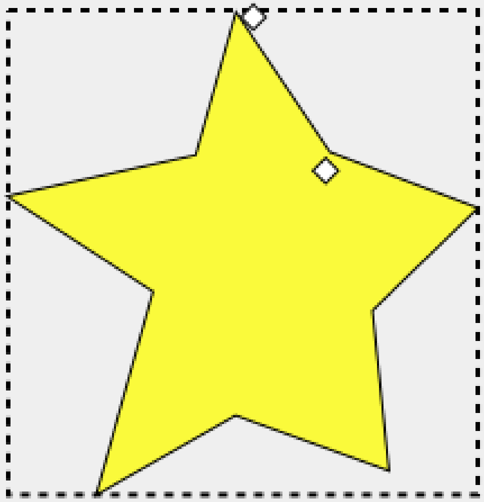
A star with a negative *Base* radius.

Holding the **Shift** key while dragging either handle will round the corners of the star or pentagon.



A star with rounded corners.

Holding the **Alt** key while dragging either handle will move all the star's or polygon's vertices independently in a random fashion.



A star with a random factor added.

As mentioned above, the *Tool Controls* area contains entries to determine if a star or regular polygon is drawn and to set the number of points or vertices. For stars, it also contains a box to set the *Spoke ratio*. This is defined as the ratio of the *Base* radius to the *Tip* radius. Useful values are: for a regular 5-pointed star, 0.382; for a regular 6-pointed star, 0.577; and for a regular 8-pointed star, 0.541. Numerical values for *Rounded* and *Randomized* can also be entered (try -25 for *Rounded*!). And lastly, there is a *Defaults* button ( ) to reset all of the settings to their default values.

← Ellipses, Circles, and Arcs

↑ Table of Contents

→ Spirals

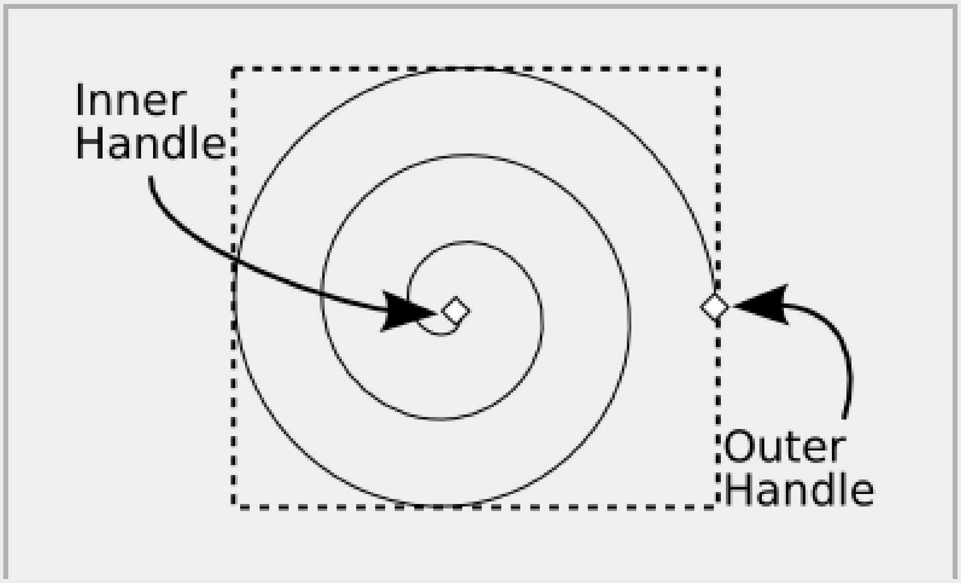
Inkscape: Guide to a Vector Drawing Program

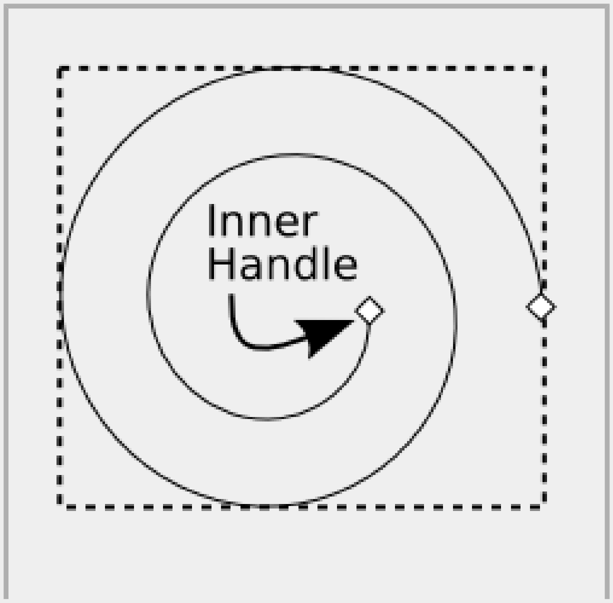
Get the book.

Spirals

The *Spiral Tool* can be used to draw Archimedes spirals. Other types of spirals can be drawn using the *Function Plotter* effect with polar coordinates. Select the tool by clicking on the icon (**F9** or **i**) in the *Tool Box*. To draw a spiral, use a **Left Mouse Drag**. The start of the drag will be the spiral's center. Holding down the **Ctrl** key while dragging will constrain the position of the spiral end point to a multiple of the *Rotation snap angle* (default 15 degrees). To reverse the direction of the spiral, flip it (Object → Flip Horizontal (**H**) or Object → Flip Vertical (**V**)).

Spirals can be reshaped by either dragging handles (small diamonds) on the spiral or by using the settings in the *Tool Controls* when the *Spiral Tool* is selected. There are two handles, the *Inner* and *Outer*. Dragging either handle allows *rolling* and *unrolling* the spiral from its respective end (i.e, making the spiral longer or shorter, or changing the radius of the inner and outer ends). Holding down the **Ctrl** key forces the end to be at a multiple of the *Rotation snap angle* with respect to the center. Holding down the **Shift** key while clicking on the *Inner* handle will set the inner radius to zero.

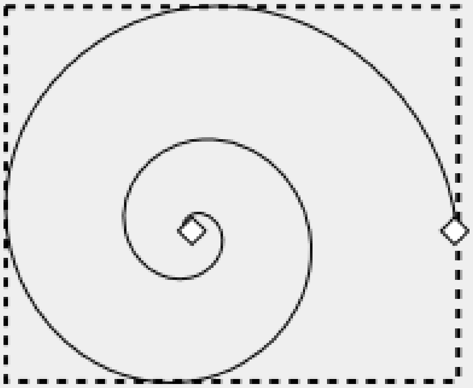




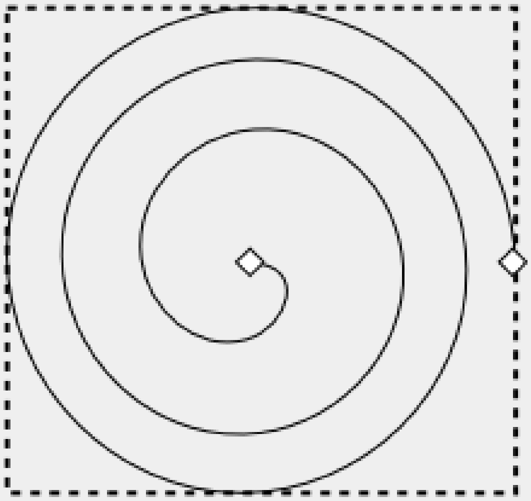
A spiral showing the Inner and Outer handles.

A spiral after unrolling one turn with the Inner handle.

The *Inner* handle can also be used to change the divergence of the spiral by dragging the handle up or down while holding down the **Alt** key. The divergence is a measure of how rapidly the radius changes with respect to the angle as the spiral progresses. A divergence of one gives a spiral where the distance between successive turns remains uniform (an Archimedes' Spiral). Divergences smaller (larger) than one give a spiral where the distance between successive turns decreases (increases) moving outward. Mathematically, the radius of a point is proportional to its angle (measured in radians) raised to a power equal to the divergence. Clicking on the *Inner* handle while holding down the **Alt** key will reset the divergence to one.



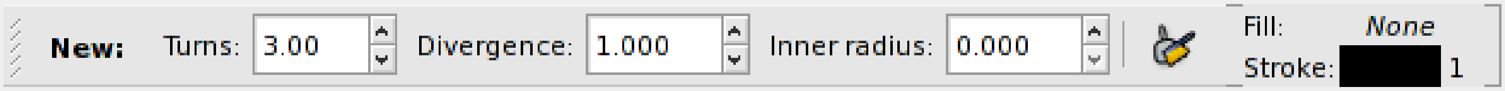
A spiral with a divergence of 2.



A spiral with a divergence of 0.5.

The *Outer* handle can be used to scale and rotate the spiral by dragging it with the **Shift** key pressed. If both the **Shift** and **Alt** keys are held down, then the spiral will only rotate, keeping the radius fixed.

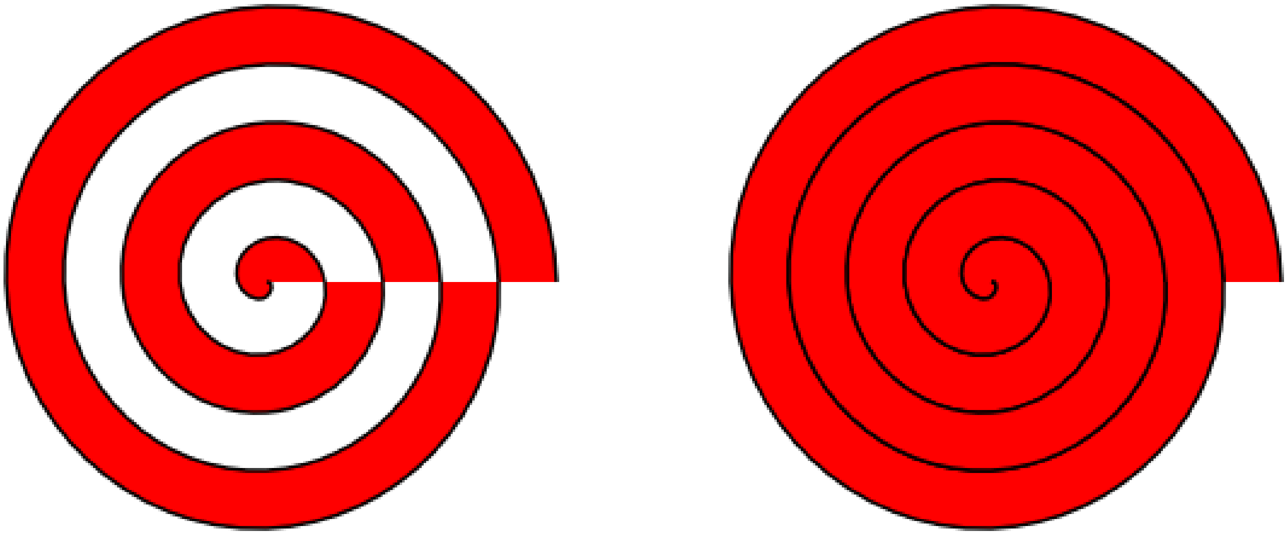
The number of turns, divergence, and inner radius can all be set in the *Tool Controls*. These values can also be reset to their default values by clicking on the *Defaults* button ( ).



New: Turns: 3.00 Divergence: 1.000 Inner radius: 0.000 Fill: *None* Stroke: 1

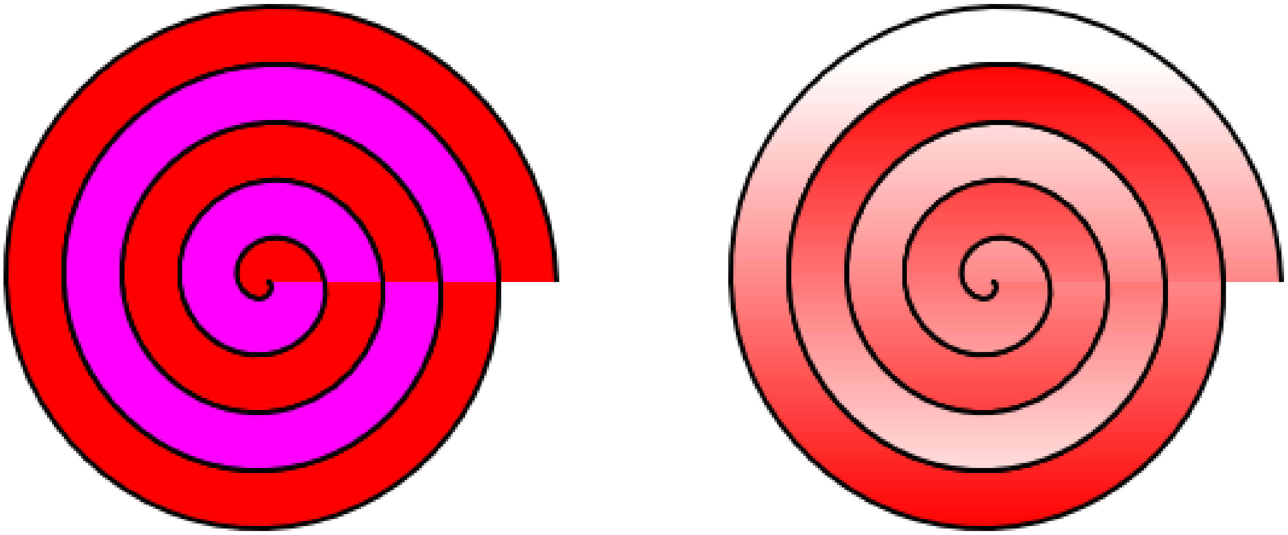
The *Spiral Tool*-*Tool Controls*.

To understand the *Fill* of spiral one must understand how Inkscape calculates *Fill*. This is covered in detail in Chapter 9, *Attributes*. A spiral is basically an open path. The *Fill* is drawn as if the path was closed with a line-segment between the path ends (the *Inner* and *Outer* handles). Then the current *Fill Rule* is applied.



Even-odd fill rule (left). *Non-zero* fill rule (right).

How do you completely fill a spiral with the *Even-odd* fill rule? The trick is to use a second duplicate spiral, with one less turn (drag *Outer* handle to unwind the spiral one turn).



Two overlaid spirals, both with an *Even-odd* fill rule. Solid fill (left), gradient fill (right).

← ↑ →

Regular Polygons and Stars

Table of Contents

Chapter 7. Paths

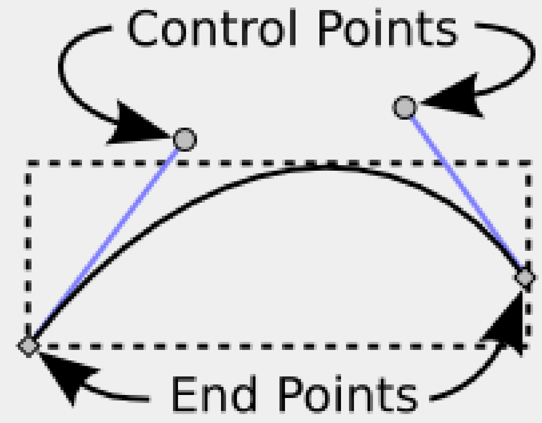
Get the book.

Inkscape » Paths » Bezier Curves

Index

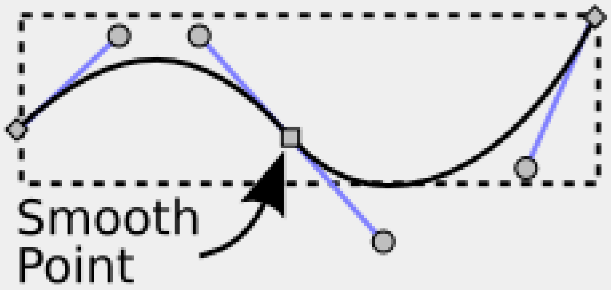
Bezier Curves

All paths are described internally in Inkscape (and in many other drawing programs) as a series of Bezier curves. It is very useful to understand the basic properties of Bezier curves for drawing and manipulating paths. Bezier curves are defined by four points, two of which are the *end* points or *nodes* of the curve. The other two are *control* points or handles, each paired with one of the end points. The control points have the useful property that a line starting at one end of the curve and ending at the corresponding control point is tangent to the curve at the end point. This enables the smooth joining of multiple Bezier curves to form a path.



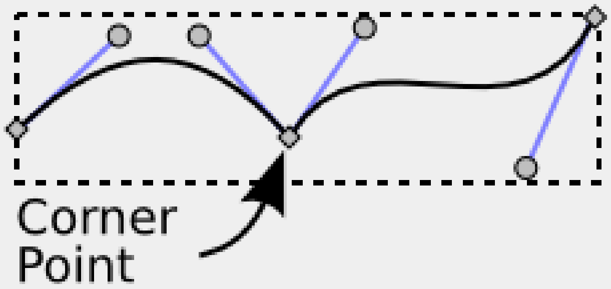
A Bezier curve showing the *end* points (*nodes*) and *control* points (*handles*).

Two or more Bezier curves can be joined to form a more complex path. The node where they are joined may be smooth, indicated by a square.



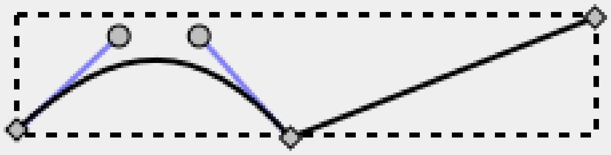
Two Bezier curves joined by a smooth node.

Or the node may be a *corner* node, also referred to as a *cusp* node, indicated by a diamond, where an abrupt change in direction is allowed.



Two Bezier curves joined by a *corner* node.

The segment between two nodes may be a *curve* or a *line*. Note that there are no control points for line segments.



Two segments, one a *curve* and the other a *line*.

Chapter 7. Paths

Table of Contents

Creating Paths

Inkscape: Guide to a Vector Drawing Program

Inkscape » Paths » Creating Paths

Index

Creating Paths

Paths can be created by the *Freehand*, *Bezier*, and *Calligraphy* drawing tools. They can also be created by conversion from a regular shape or text object.

The Freehand Tool

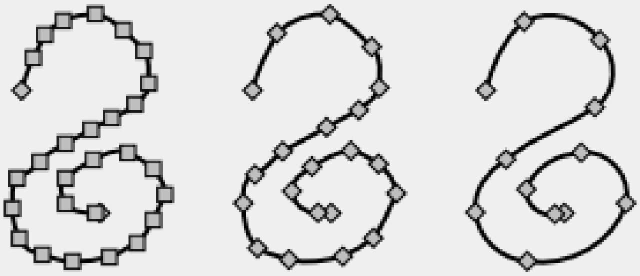
The *Freehand (or pencil) Tool* is perhaps the easiest tool with which to draw a path. Simply click on the icon (**F6** or **p**) in the *Tool Box* and then click-drag the mouse over the canvas to draw a line. Holding down the **Shift** key while drawing temporary disables nodes snapping to the *Grid* or *Guide Lines*.

As long as a path is selected, you can extend the path by click-dragging from one of the path's ends. To prevent adding to a path, deselect the path with the **Esc** key.

New in v0.46.

To delete an unfinished path, use **Esc** or **Ctrl**+**Z**.

Paths drawn with the *Freehand Tool* tend to be composed of many Bezier curves leading to an erratic-looking path. One can smooth and simplify such curves by using the Path → Simplify (**Ctrl+L**) command one or more times.



A path drawn with the *Freehand Tool* after zero, one, and multiple applications of the *Path Simplify* command (with nodes shown).

New in v0.45.

There are two possible modes for this command. The default is to treat all of the selected paths as one object. The second mode is to treat each sub-path separately. To use the second mode, add an entry in the “options” section with “simplifyindividualpaths” set to '1' in the file `.inkscape/preferences.xml`.

New in v0.46.

Single dots can be created by using **Ctrl**+**Left Mouse Click**. The size of the dot can be set in the *Pencil* section of the *Inkscape Preferences* dialog as a multiple of the current *Stroke* width. The dot is represented in *SVG* as a filled path. Adding the **Shift** key doubles the dot size while adding the **Alt** creates a random size dot. The *Bezier Tool* has the same options.

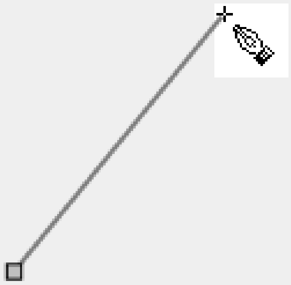
The Bezier Tool

As mentioned previously, all paths are represented in Inkscape as a series of Bezier curves. The *Bezier (or pen) Tool* allows you more directly control the Bezier parameters as you draw a path. To select the tool, click on the icon (**Shift**+**F6** or **b**) in the *Tool Box*.

Tip

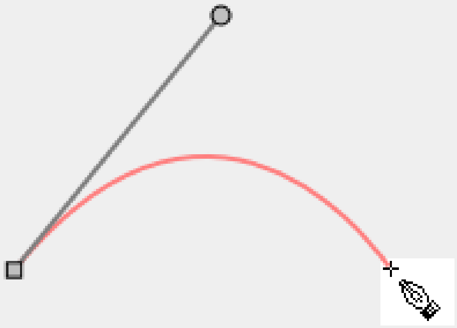
This is one place where paying attention to the *Notification Region* is especially useful. The region not only lists your options at each step but also gives the distance and angle the cursor is from the last node when placing a new node or dragging a handle.

To begin to draw a curve, click-drag on the canvas. The point where you click becomes the end point or node of the curve. As you drag the cursor, you'll see a gray line between the end point and the cursor. This line is a tangent to your curve at the end point. Release the mouse button to establish the first control point.



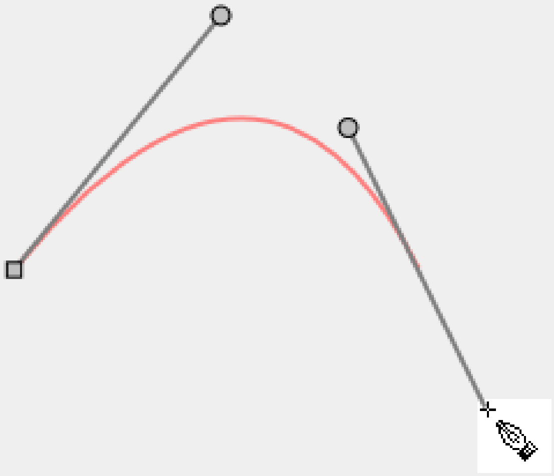
Start of drawing Bezier curve.

Move the cursor to the position of the next Bezier curve end point or node. A red line will show you the shape of the curve.



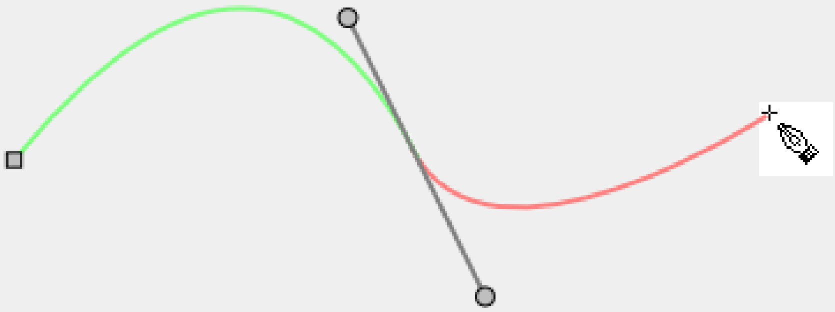
Positioning of next Bezier curve end point (node).

Click-drag from the end point to draw out a handle that allows you to set the second control point. The pointer is actually pointing to the sister of the second control point, which is the initial control point of a second Bezier curve to be attached to the first. The two points are collinear with and the same distance from the Bezier curve end point or node.



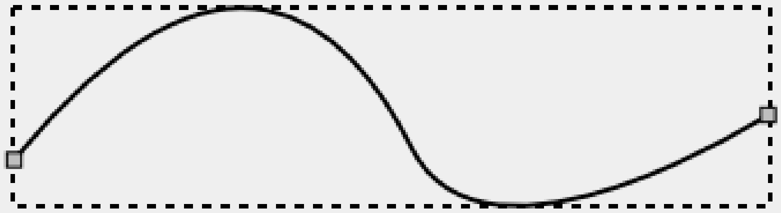
Setting of second Bezier curve control point.

Next, move the cursor to the end point of the second Bezier curve.



Positioning of end point (node) of second Bezier curve.

One can repeat the above steps to add as many Bezier curves to the path as required. To end the path, press **Enter** or do a **Right Mouse Click** after placing the last Bezier curve end point.



The finished path, composed of two Bezier curves.

Other useful things to know while using the Bezier Tool:

- To create a path of straight lines, click rather than click-drag at each node.
- You can use the **Arrow** keys to move the last node created while drawing a path. **Shift**+**Arrow** moves the node by ten times the normal step, **Alt**+**Arrow** will move the node by a screen pixel.
- To set the two control points of a node separately (and force the node to be a *corner* point), first set the control point for the end point of the previous Bezier curve, then hold down the **Shift** key while setting the control point for the next Bezier curve.
- To constrain a node to be at a multiple of the *Rotation snap angle* with respect to the previous node, hold down the **Ctrl** key while setting it.
- To constrain a control point to be at a multiple of the of the *Rotation snap angle* with respect to a node, hold down the **Ctrl** key while setting it. (Can be used in conjunction with the **Shift** key.)
- To delete the last node drawn, use the **Backspace** key or the **Del** key.
- To delete an unfinished path, use **Esc** or **Ctrl**+**Z**.
- To change an unfinished segment (the red line) from a curve to a line, use **Shift**+**L**. To change an unfinished segment from a line to a curve, use **Shift**+**U**.
- To extend a previously drawn path, select the path, then click or click-drag on an end point.
- To close a path, click on the first endpoint when placing the last endpoint.

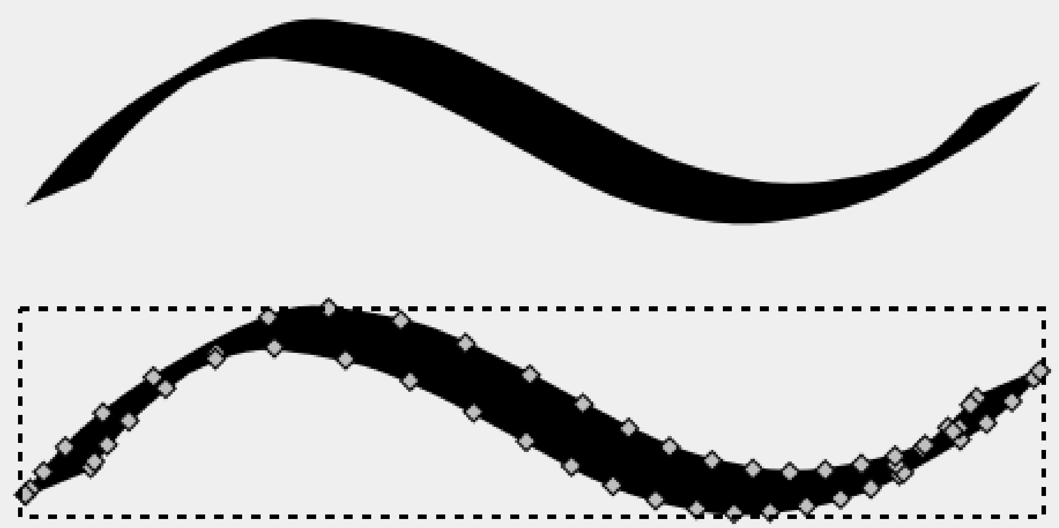
New in v0.46.

Single dots can be created by using **Ctrl**+**Left Mouse Click**. The size of the dot can be set in the *Pen* section of the *Inkscape Preferences* dialog as a multiple of the current *Stroke* width. The dot is represented in *SVG* as a filled path. Adding the **Shift** key doubles the dot size while adding the **Alt** creates a random size dot. The *Freehand Tool* has the same options.

The Calligraphy Tool

As the name suggests, the *Calligraphy Tool* can be used to draw calligraphic lines. The resulting paths are a bit different than those drawn with the *Freehand* and *Bezier* tools in that they are composed of two parallel (or almost parallel) sub-paths, allowing the resulting line to have a variable width. The path is not stroked but the

Fill is solid (see Chapter 9, *Attributes*).



A path drawn with the *Calligraphy Tool*. The bottom path shows how the nodes are placed.

To begin drawing a Calligraphy path, select the tool by clicking on the icon (**Ctrl**+**F6** or **c**) in the *Tool Box* and then click-drag the mouse over the canvas to draw a line. The line will have by default the *Current style*. You can choose to draw all lines with a fixed style by selecting the *This tool's own style* option under the Calligraphy section in the *Inkscape Preferences* dialog. If you turn off the *Keep selected* option in the dialog, a newly drawn object will not remain selected; you can then choose a color from the *Palette* for the next calligraphic stroke without changing the color of the previously drawn stroke.

The Calligraphy Tutorial (Help → Tutorials → Inkscape: Calligraphy) has many ideas on how to use the *Calligraphy Tool*.

The *Calligraphy Tool* has many options, two accessible from the keyboard and the rest from the *Tool Controls*. It is best just to try changing the various option settings to get the feel for them.

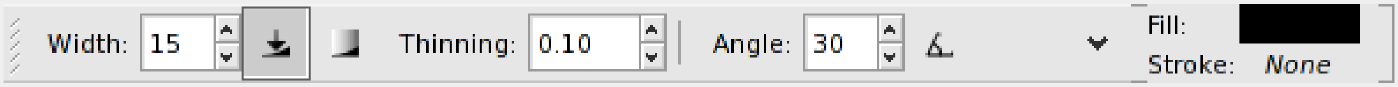
Options with keyboard:

- To change the pen width while drawing: **Left Arrow** and **Right Arrow**. *New in v0.46:* The **Home** key sets the width to the minimum while the **End** key sets the width to the maximum. Typing **Alt**+**X** will enable the *Width* entry box in the *Tool Controls*; enter a number and then hit **Enter** to set an exact width while drawing.
- To change the pen angle while drawing: **Up Arrow** and **Down Arrow**.
- *New in v0.46.*

  To add to an existing path (form a union), hold the **Shift** down while drawing.
- *New in v0.46.*

To delete an unfinished path, use **Esc** or **Ctrl**+**Z**.

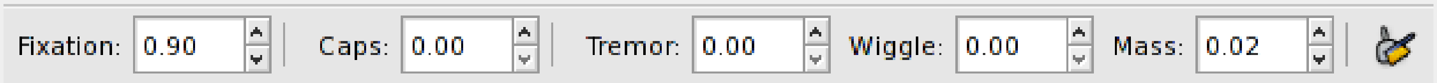
The *Tool Controls* contains too many options to all be shown at the default Inkscape window width. To access all the options, widen the Inkscape window or click on the triangle at the right of the bar, which will show a drop-down menu with the missing options.



Width: 15 Thinning: 0.10 Angle: 30

Fill: Stroke: *None*

The *Calligraphy Tool*-*Tool Controls*.



Fixation: 0.90 Caps: 0.00 Tremor: 0.00 Wiggle: 0.00 Mass: 0.02

The Calligraphy Tool Controls — expansion. To see these options, widen the Inkscape window.

- Pen width (1 to 100): number is tenths of percent of canvas width (i.e., 15 is 1.5% of canvas width). Note that if you change the zoom level, the effective pen width will change. To keep the pen width constant, check the *Width is in absolute units* box under the *Tools-Calligraphy* page in the *Inkscape Preferences* dialog. The pen width will then be in units of *px*.
- ⤓ : On/Off button: Use tablet pressure for width (requires tablet input device).
- *New in v0.46.*

  ◪ : Trace background: Vary the width of the pen as a function of the background. A dark background yields a wider pen.
- Thinning (-1.0 to 1.0): How the width change depends on the speed of the drawing stroke:
  - > 0.0: Line width decreases with speed.
  - = 0.0: Line width independent of speed.
  - < 0.0: Line width increases with speed.
- Angle (-90 to 90 degrees): Angle of pen relative to horizontal axis.
- ∠ : On/Off button: Use tablet tilt for pen orientation (requires tablet input device).
- Fixation (0.0 to 1.0): How the angle of the pen follows the direction of the pen.
  - 0.0: Angle follows pen direction (always perpendicular to motion).
  - 1.0: Angle fixed to angle defined in *Angle* entry.
- *New in v0.45.*

  Caps (0.0 to 5.0): How round is the end of the stroke. This can be used to produce round end-caps when the fixation is small (i.e., simulating a round brush). Note that the cap extends beyond where the stroke would normally end.
  - 0.0: Flat end.
  - 1.0: Approximately semicircular.
  - 5.0: Elliptical, approximately five times longer than wide.
- Tremor (0.0 to 1.0): How much random shake should the stroke have. This parameter can be used to create a more realistic looking calligraphic stroke by adding some randomness to the thickness of the

stroke. It works by adding randomness to the node handle orientations.
  - 0.0: Smooth
  - 1.0: Chaotic.

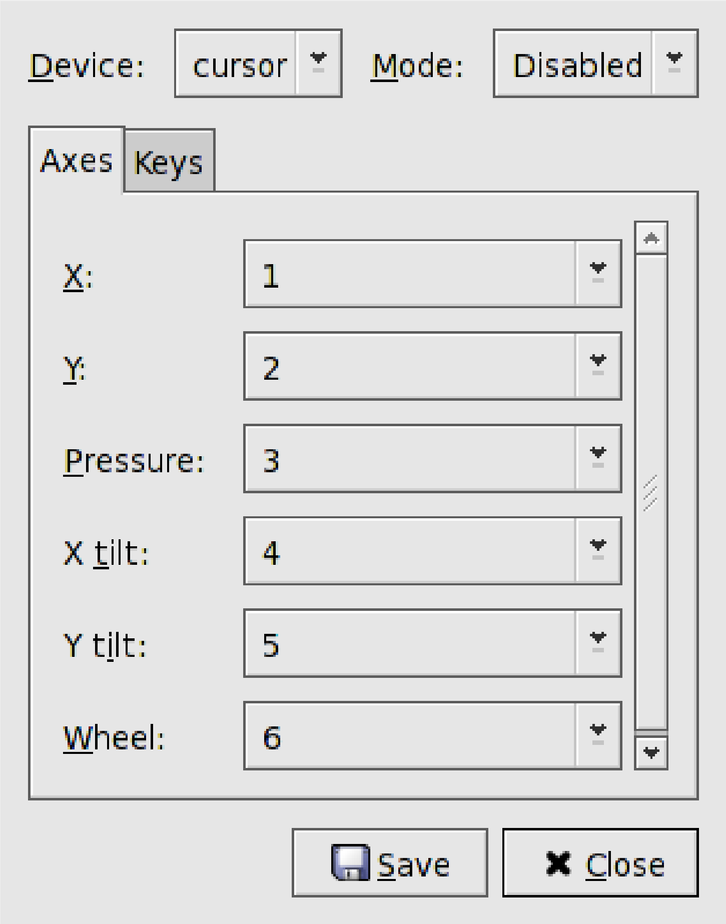
- *Renamed from Drag (with the numerical values inverted) in v0.45.*

  Wiggle (0.0 to 1.0): How resistant the pen is to movement. With a value of 1.0, the pen will *wiggle* all over the paper.
- Mass (0.0 to 1.0): How the line follows pen movement. The more *massive* the pen, the smoother the stroke but the less responsive the pen will be. Try a value of 0.10 for a good compromise between smoothness and responsiveness.
- Restore default values.

Two options require the use a tablet (e.g., Wacom): using tablet pressure to control the width of a stroke and using tablet tilt to control the orientation of the pen's nib. This is discussed in the next section.

Using a Tablet

Inkscape can make good use of a tablet for input. This is especially true with the *Calligraphy Tool* where stroke width and nib orientation can be controlled directly with a pen. Before using a tablet, the *extended input devices* must be configured and enabled with the *Input Devices* dialog (File → Input Devices...).



Device: cursor
Mode: Disabled

X: 1
Y: 2
Pressure: 3
X tilt: 4
Y tilt: 5
Wheel: 6

Input Devices dialog.

This dialog is the same as used by Gimp; however, not all features may work the same.

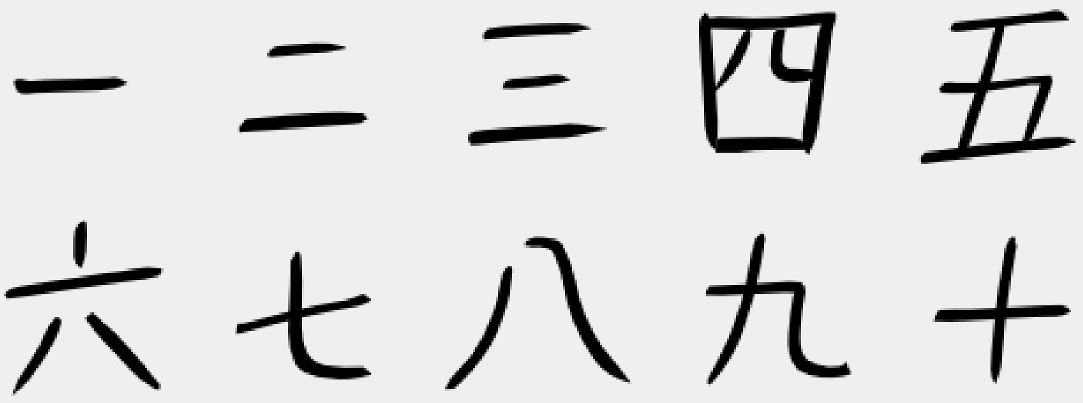
To enable pressure and tilt control, select the device to be used from the pull-down menu at the top left and enable it with the pull-down menu on the top right. In principle, there are three options for the latter pull-down menu:

- Ignore: Don't enable special features of extended input device. Treat as a regular mouse.
- Screen: Extended device will work as regular mouse outside of Canvas region.
- Window: Extended device works only in Canvas region.

The *Axes* tab allows you to swap input assignments; i.e., the *x*-axis for the *y*-axis if you rotate the tablet by 90 degrees. This doesn't seem to work. (Bug?)

The *Keys* tab allows you to assign key combinations to the macro keys, if any, on the tablet.

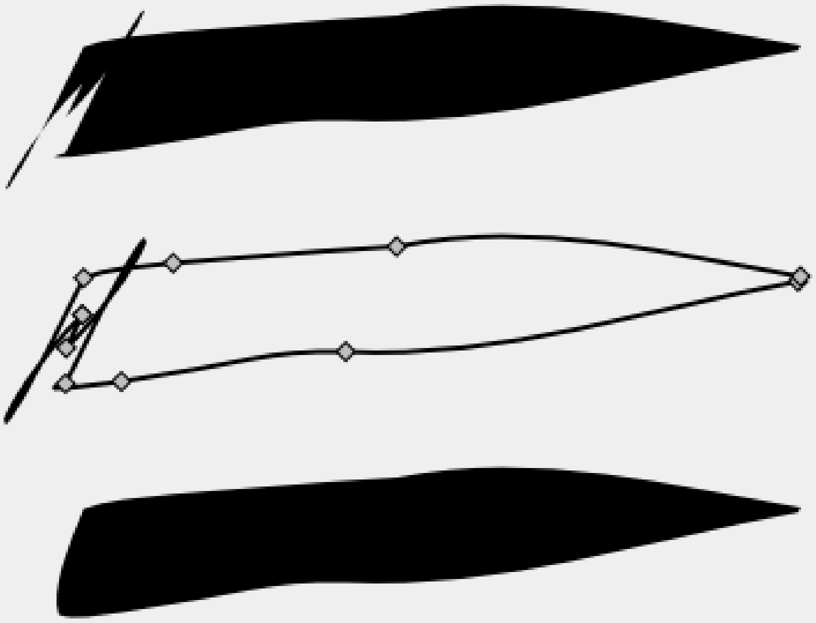
Pressure and tilt do not work properly on OS X due to problems with the X11 implementation.



An example of calligraphy using a Wacom tablet. The figures are the numbers 1 through 10 written (poorly) in the characters common to Chinese, Japanese, and Korean. The tablet pressure was used to control the stroke width.

Updated for v0.45.

A stroke made using a Wacom tablet with pressure sensitivity may create a mess at the beginning of the stroke or where there is a quick change in direction. This is especially true when the *Fixation* is set to a small number. Below is an example of this. By removing some of the nodes, the stroke can be cleaned up. The significance of this problem has been dramatically reduced as of v0.45.



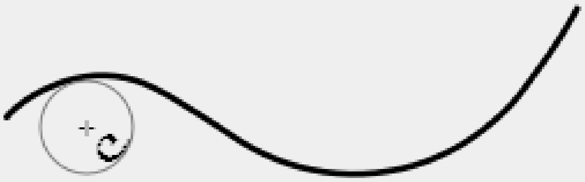
An example of cleaning up a calligraphic stroke. Top: Original stroke. Middle: Original stroke showing path and nodes. Bottom: Cleaned-up stroke after removing several nodes and adjusting some handles.

Hatchings

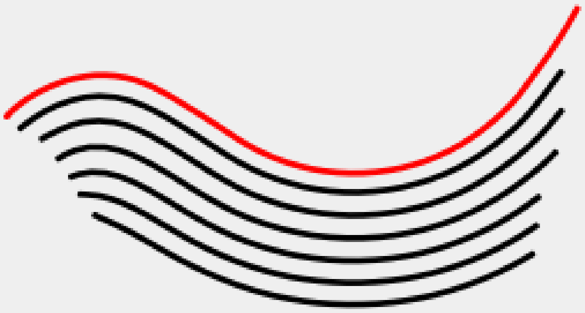
New in v0.46.

The *Calligraphy Tool* has an option that allows parallel lines (hatchings), as found in engravings, to be easily drawn. To use this option, first select a guide path. Any path, *Shape* or text can be used as a guide. Then with the **Ctrl** key held down, drag the pointer along a line parallel to the guide path. The closest distance between the start of the drag and the guide path determines the distance the new line will be away from the guide. This is indicated by the gray circle around the pointer. As you begin to drag, the circle turns green. This indicates that the cursor is tracking the guide. When you finish drawing a line, don't release the **Ctrl**! Just start drawing (**Left Mouse Drag**) another line. As long as you hold the **Ctrl** key down, each new line will use the last drawn line as a guide and the spacing will remain the same.

If you deviate too far from the guide, the pointer will break free. This is indicated by a red circle. This is an intentional design decision to allow one to continue a line past the end of the guide. If you accidently break free, you can delete the last path and start again; but you must reset the interline spacing. A slow steady hand works best. If the cursor is consistently closer or farther away from the guide path than the set spacing, the interline spacing will gradually decrease or increase. This is a subtle effect.

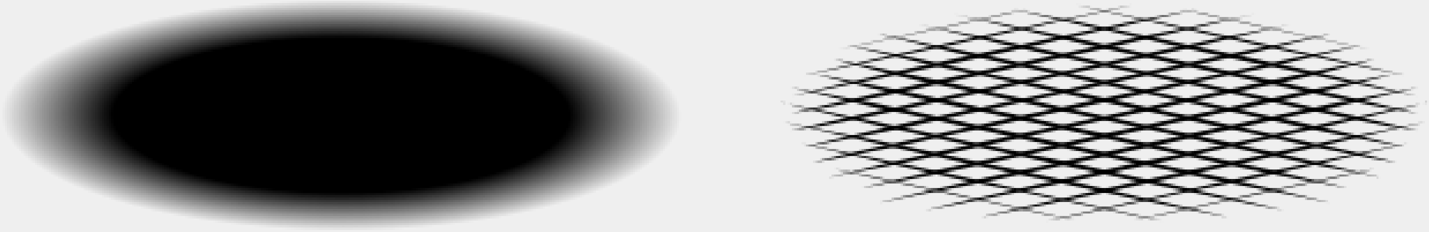


The *Calligraphy Tool* cursor near a guide path when the **Ctrl** key is held down.



A series of “engraved” lines created with the *Calligraphy Tool* while holding down the **Ctrl** key. The red line is the guide path.

The *Trace Background* (◪) option can be used to generate hatchings with pen width reflecting the background darkness as shown below.



A tracing of the shadow on the left is shown on the right. The *Tweak Tool* was used to clean up the ends.

By default, each new line uses the previous line as its guide path. To use the original guide path for each line, turn off the option *Select new path* found under the *Calligraphy Tool* section in the *Inkscape Preferences* dialog. The author finds it easier to make hatchings with this option turned off.

Tip

Use *Touch selection* to select the lines in the hatchings. It is often easier to keep hatchings in a separate *Layer*.

Paths from Other Objects

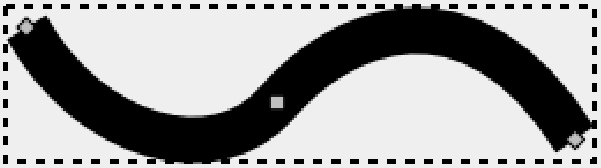
There are several ways to generate paths indirectly. One is to *convert* a regular shape or text object into a path. Another is to use *Stroke to Path*, which converts a path into a closed path with two parallel sub-paths. A third way to generate a path is to trace a bitmap image. This method is considered in Chapter 16, *Tracing Bitmaps*.

Object to Path

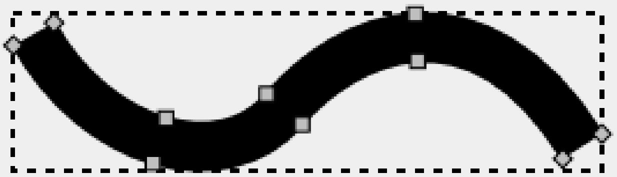
To convert a regular shape or text object to a path, use Path → Object to Path (**Shift**+**Ctrl**+**C**). Once an object is converted, the object loses any special knowledge associated with its previous existence. For example, the text font cannot be changed. But once an object is converted to a path, it can be modified in any arbitrary way, as shown in the section called “A Hiking Club Logo - An Exercise in Paths”.

Stroke to Path

A stroked path can be converted to a filled object consisting of two parallel sub-paths using Path → Stroke to Path (**Ctrl**+**Alt**+**C**). The path should have a non-zero thickness. The before and after objects look the same but have different structure and behavior. See the difference in the nodes in the figures below.

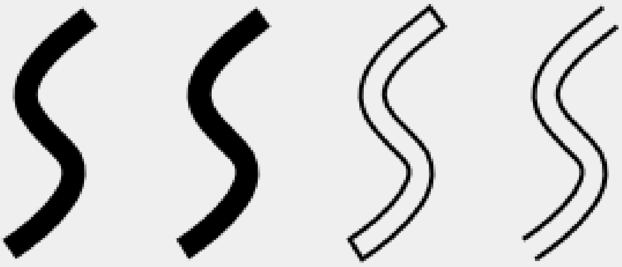


A stroked path consisting of three nodes.



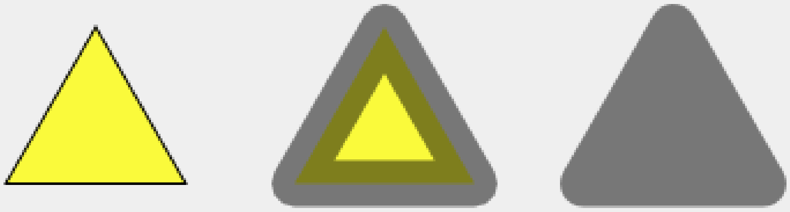
A filled path made from the stroked path by using the *Stroke to Path* command. It consists of ten nodes.

The *Stroke to Path* command can be used to make a sets of parallel lines. Simply draw the path you desire, setting the width to the desired gap plus the desired final stroke width. Convert the stroke to path, remove the fill, and add the stroke paint. The line segments at the ends can be removed if desired by selecting each pair of end nodes and using the *Split path between two non-end nodes* command in the *Node Tool*-*Tool Controls* (see next section). The *Path Offset* commands (for closed paths) or *Complex Strokes* are alternative ways of creating parallel lines.



Creating parallel lines. From left to right: Single line, Filled path after *Stroke to Path* command, Fill removed - Stroke painted, Paths at ends removed.

The *Stroke to Path* command can also be used to make polygons with uniformly rounded corners, as shown below. The trick is to use a wide *Stroke* and set the *Join* style to *Round*.



From left to right: A simple triangle; the triangle with a wide *Stroke* (the *opacity* of the *Stroke* has been set to 50% to allow the triangle shape to be seen); the finished triangle with rounded corners after using the *Stroke to Path* command and after removing the inner triangle.

Editing Paths

The primary means for editing paths is to use the *Node Tool* to modify one or more path nodes. A few exceptions are covered at the end of this section (simplifying, reversing, and offsetting paths).

Using the Node Tool

Paths are normally edited using the *Node Tool*. This flexible tool enables the addition, deletion, and movement of nodes. A *Sculpting* mode allows easy fine tuning of paths with many nodes.

Select the *Node Tool* by clicking on the icon (**F2** or **n**) in the *Tool Box*. Then click on the path you wish to edit. All of the path's nodes will be shown.

> **Note**
> Many of the things you can do with objects, you can do with nodes using the same methods. This is especially true for selecting and moving nodes. For example, the **Arrow** keys move selected objects by the *Nudge factor* when the *Select Tool* is active; they move selected nodes by the *Nudge factor* when the *Node Tool* is active. (The *Nudge factor* is a parameter that can be set in *Inkscape Preferences* dialog (File → Inkscape Preferences... (**Shift+Ctrl+P**)) dialog under the *Steps* tab). Knowing this should make learning to use the *Node Tool* quicker.

Selecting Nodes

Nodes must be selected before they can be edited. Selected nodes are indicated by a change in color as well as a slight enlargement in size. The handles of the selected nodes are shown, as are the handles for adjacent nodes. If the handles get in the way of selecting nodes, they can be toggled off by clicking on the *Hide Handles* icon in the *Tool Controls*. Clicking the icon a second time toggles the handles back on.

Nodes can be selected for editing a number of ways:

- **Left Mouse Click** on a node to select that node. A node turns red when the pointer hovers over it and it can be selected (or deselected).
- **Left Mouse Click** on the path to select the nearest node on each side of the place where you clicked. The hand symbol is added to the pointer when hovering over a clickable path. Note that a node can be selected and moved in one step by click-dragging on the node.
- **Left Mouse Drag** will select all nodes within the rubber-band box. The drag must not begin on a path unless the **Shift** is used. Using the **Shift** key allows the drag to begin on a path except over a node.

Nodes can be added (or removed) from the selection by holding down the **Shift** key while using one of the above methods.

New in v0.45.

Nodes can also be added to or removed from the selection by hovering the cursor over a node and using the **Mouse Wheel**, moving “up” to add nodes and “down” to remove nodes. The **Page Up** and **Page Down** keys can be used in place of the **Mouse**

Wheel. This selection technique is especially useful in conjunction with *node sculpting*.

Two modes are possible: The default mode adds nodes based on the *spatial* distance from the cursor. If the **Ctrl** key is held down, the nodes are selected based on the *linear* distance measured *along the path*. In this latter case, only nodes in the same sub-path can be selected.

Tab selects the *next* node in a path if one is already selected. This is usually the adjacent node in the direction the path was drawn. If no node is selected, it will select the first node. **Shift**+**Tab** will select the *previous* node in a path. **Ctrl**+**A** selects *all* nodes in a selected path. **!** inverts the node selection for any sub-path with at least one node selected. **Alt**+**!** inverts the node selection for the entire path.

Editing Nodes with the Mouse

The mouse can be used to move nodes and handles by dragging them. It can also be used to alter the shape of a path between two nodes by dragging the path. And finally, nodes can be inserted anywhere along a path by double clicking the path or by clicking the path while holding down the **Ctrl**+**Alt** keys.

Nodes

- **Left Mouse Drag**: Move selected nodes: If the pointer starts over a selected node, all selected nodes will move. If the pointer starts over an unselected node, that node will be selected and moved.
- **Ctrl**+**Left Mouse Drag**: Move selected nodes in either the horizontal or vertical direction.
- **Ctrl**+**Alt**+**Left Mouse Drag**: Move selected nodes along a line collinear with a node handle or to its perpendicular (passing through the node). The handles used are those belonging to the node where the pointer begins the drag.
- **Shift**+**Left Mouse Drag**: Temporarily disable snapping nodes to the *Grid* or to *Guide Lines* (if snapping of nodes enabled).
- **Left Mouse Drag**+**Space**: While dragging, drop an unlinked copy of the nodes. The copy is of the entire path even if only a few nodes are selected.

Handles

A handle becomes active when the mouse hovers over its control point. The control point will turn red. The **Shift**, **Ctrl**, and **Alt** keys can be used in combination for the options listed below.

- **Left Mouse Drag**: Move handle.
- **Shift**+**Left Mouse Drag**: Rotate both handles of a node together.
- **Ctrl**+**Left Mouse Drag**: Snap handle to either a multiple of the *Rotation snap angle* (15 degrees by default), or a line collinear with or orthogonal to the original handle.
- **Alt**+**Left Mouse Drag**: Allow only angle and not length to change as handle is dragged.

Editing Nodes with the Keyboard

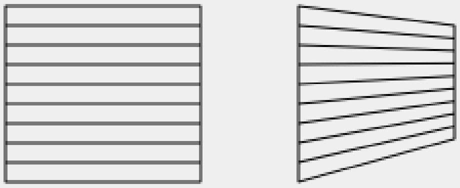
This section covers using the keyboard to move nodes and to adjust their handles. The keyboard can also be used to add and delete nodes, change the type of node, and to join or break paths. For these latter uses, see the keyboard shortcuts in the following section on the *Node Tool*-*Tool Controls*.

In this section, **Left-** and **Right-** applied to the **Ctrl** and **Alt** modifying keys refers to the keys on the left and right side of the **Space** bar. Using a left modifying key causes the left handle of a node to be modified; using a right modifying key modifies the rightmost handle. The definition of which handle is left or right is not always completely obvious as when one handle is directly above the other or when the leftmost handle is moved to the right of the former rightmost handle.

Note

Updated for v0.46.

The scaling and rotating operations described below are different if one node is selected as compared to two or more nodes. If two or more nodes are selected, the nodes act like an object and scale or rotate around the center of the selection, as described in the section called “Transforms with the Keyboard” in Chapter 5, *Positioning and Transforming*. If the mouse is over a node, then that node is used as the center of rotation. It is also possible to flip the nodes horizontally and vertically by using keyboard shortcuts.



An example of editing multiple nodes. The rectangles on the right are formed by one path. By selecting all the nodes on the right side of the path and using the normal transformation commands a perspective effect can be achieved. Use **<** to move the nodes closer together and the **Arrow** keys to move the nodes to the left and up.

Translations

Same as for translating objects.

- **Arrow**: Move selected nodes by the *Nudge factor* (2 *SVG* pixels by default).
- **Shift**+**Arrow**: Move selected nodes by ten times the *Nudge factor*.
- **Alt**+**Arrow**: Move selected nodes one *Screen pixel*.
- **Alt**+**Shift**+**Arrow**: Move selected nodes ten *Screen pixel*s.

Scaling Handles

Scaling applies to the the node handles and is not directly mappable to scaling objects. These items only apply when *one* node is selected.

- **.** or **>**: Expand handles on both sides of selected node by *Scale step* (2 *SVG* pixels by default).
- **,** or **<**: Shorten handles on both sides of selected node by *Scale step* (2 *SVG* pixels by default).
- **Left-Ctrl**+**.** , **Left-Ctrl**+**>** , **Right-Ctrl**+**.** , **Right-Ctrl**+**>** : Expand handle on one side of selected node by *Scale step*. **Left-Ctrl** selects the leftmost handle while **Right-Ctrl** selects the rightmost handle.
- **Left-Ctrl**+**,** , **Left-Ctrl**+**<** , **Right-Ctrl**+**,** , **Right-Ctrl**+**<** : Shorten handles on one side of selected nodes by *Scale step*. See above for usage of Left vs. Right **Ctrl** keys.
- **Left-Alt**+**.** , **Left-Alt**+**>** , **Right-Alt**+**.** , **Right-Alt**+**>** : Expand handle on one side of selected node by one *Screen pixel*. See above for usage of Left vs. Right **Alt** keys.
- **Left-Alt**+**,** , **Left-Alt**+**<** , **Right-Alt**+**,** , **Right-Alt**+**<** : Shorten handle on one side of selected node by one *Screen pixel*. See above for usage of Left vs. Right **Alt** keys.

Rotating Handles

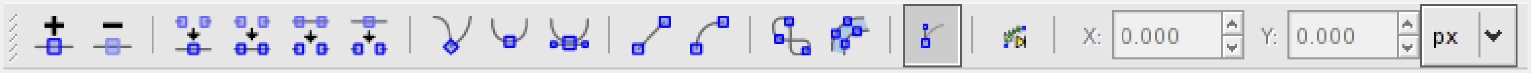
Rotating applies to the the node handles and is not directly mappable to rotating objects. These items only apply when *one* node is selected.

- **[** : Rotate handles of selected node counter-clockwise by the *Rotation snap angle* (15 degrees by default).
- **]** : Rotate handles of selected node clockwise by the *Rotation snap angle* (15 degrees by default).
- **Left-Ctrl**+**[**, **Right-Ctrl**+**[**: Rotate handle on one side of selected node counter-clockwise by the *Rotation snap angle*. **Left-Ctrl** selects the leftmost handle while **Right-Ctrl** selects the rightmost handle, as described earlier.

- **Left-Ctrl**+**]**, **Right-Ctrl**+**]**: Rotate handle on one side of selected node clockwise by the *Rotation snap angle*. See above for usage of Left vs. Right **Ctrl** keys.
- **Left-Alt**+**[**, **Right-Alt**+**[**: Rotate handle on one side of selected node counter-clockwise by one *Screen pixel*. See above for usage of Left vs. Right **Alt** keys.
- **Left-Alt**+**]**, **Right-Alt**+**]**: Rotate handle on one side of selected node clockwise by one *Screen pixel*. See above for usage of Left vs. Right **Alt** keys.

Using the Node Tool-Tool Controls

The *Node Tool*-*Tool Controls* provides an easy way to access many of the methods of editing nodes.



The *Node Tool*-*Tool Controls*.

- (**Insert**): Insert node between adjacent selected nodes. (Note: Clicking on the path will select the nearest node on both sides of the point where the path was clicked. Double clicking on the path or clicking the path once using the **Ctrl**+**Alt** keys will also insert a node, in this case under the pointer. Inkscape will try to add the node without changing the shape of the path by adjusting the handles of the nodes adjacent to the new node.)
- (**Backspace**, **Delete**, or **Ctrl**+**Alt**+**Left Mouse Click**): Delete selected nodes. Inkscape will attempt to preserve the shape of the path when nodes are removed by adjusting the handles of adjacent nodes. If you wish to remove a node and *not* change the handles of adjacent nodes use **Ctrl**+**Backspace** or **Ctrl**+**Delete**.
- (**Shift**+**J**): Join (merge) two selected end nodes. Normally the merged node is placed at the midpoint between the end nodes. With the keyboard shortcut, hovering the mouse over one of the end nodes will result in the merged node being placed at the position of that end node.
- : Join two selected end nodes with a path segment.
- : Delete segment between selected nodes.
- (**Shift**+**B**): Break path into sub-paths at selected nodes. Each selected node is converted into two end nodes.
- (**Shift**+**C**): Change selected nodes to corner or cusp nodes.
- (**Shift**+**S**): Change selected nodes to smooth nodes. When the keyboard shortcut is used, placing the mouse over one of the handles will preserve the position of that handle, rotating the partner handle of the node to match.
- (**Shift**+**Y**): Change selected nodes to symmetric smooth nodes.
- (**Shift**+**L**): Change selected segments to straight lines. One or more segments must be selected (by selecting nodes on both ends of the segment).
- (**Shift**+**U**): Change selected segments to curved lines. One or more segments must be selected.
- (**Shift**+**Ctrl**+**C**): Convert selected objects to paths.
- (**Ctrl**+**Alt**+**C**): Convert stroke of selected objects to path.
- : Toggle on/off display of handles.
- *New in v0.46.*

  : Toggle though parameter list for an *LPE*.
- *New in v0.46.*

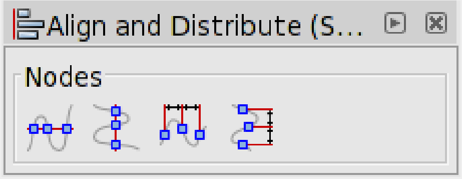
  Entry box for *x* coordinate of selected nodes.
- *New in v0.46.*

  Entry box for *y* coordinate of selected nodes.
- *New in v0.46.*

  Units for *x* and *y* coordinates.

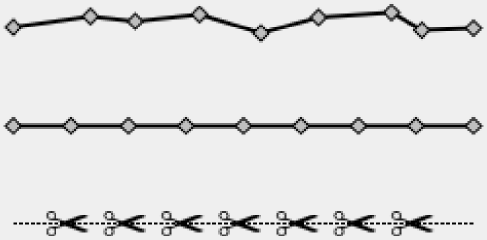
Editing Nodes with the Align and Distribute Dialog

The *Align and Distribute* dialog (Object → Align and Distribute... (**Shift**+**Ctrl**+**A**)) has special commands for editing nodes when the *Node Tool* is in use.



Align and Distribute dialog when *Node Tool* is active.

The commands will align or distribute selected nodes and can be useful to evenly place markers on a straight line (as shown in the figure below).



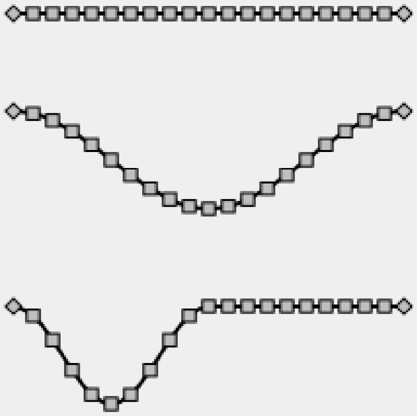
A path composed of nodes connected by straight line segments as drawn (top), aligned and distributed (middle), and with scissor markers (bottom). (*Markers* can be added by using the *Stroke style* tab of the *Fill and Stroke* dialog (Object → Fill and Stroke... (**Shift**+**Ctrl**+**F**)). See the section called “Markers” in Chapter 9, *Attributes*.

The commands available are:

- Align selected nodes along a horizontal line.
- Align selected nodes along a vertical line.
- Distribute selected nodes horizontally.
- Distribute selected nodes vertically.

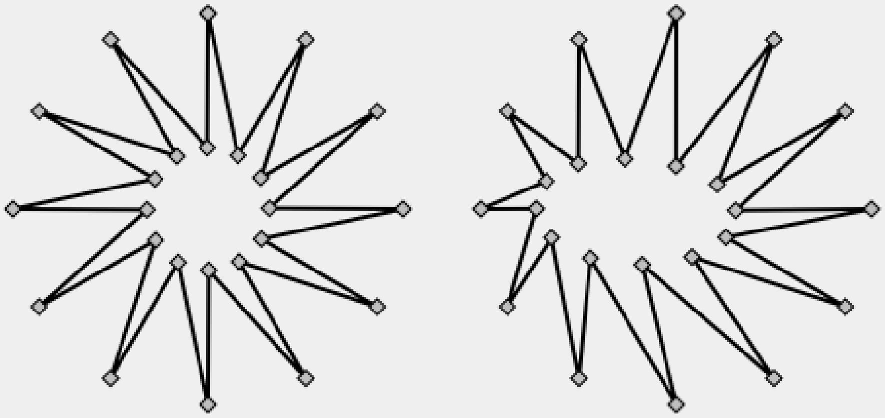
Sculpting Nodes

The *Sculpting* mode of the *Node Tool* allows one to easily manipulate a complex path, adjusting multiple nodes at the same time. The basic use is to select a group of nodes and then drag one of the selected nodes with the mouse while holding down the **Alt** key. Only the dragged node moves the full amount. The selected nodes at the end remain fixed and all the other selected nodes will move a distance that is a function of how far they are from the dragged node. The function takes the form of a *Bell Curve* distribution. This is best illustrated by the following diagram.



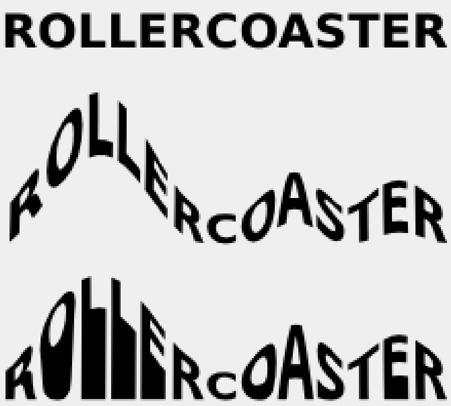
Adjusting a group of nodes using the *Sculpting* mode. Top: A straight line with many nodes. Middle: The line after selecting all nodes and dragging the middlemost node with the mouse while holding down the **Alt** key. Bottom: The line after selecting only the leftmost nodes and dragging the center of the selected nodes down while holding down the **Alt** key.

As usual with the *Node Tool*, only paths may be sculpted. Any other objects must be converted to a path first. In the following illustration, a star has been converted to a path and then the innermost nodes were selected and one was dragged.



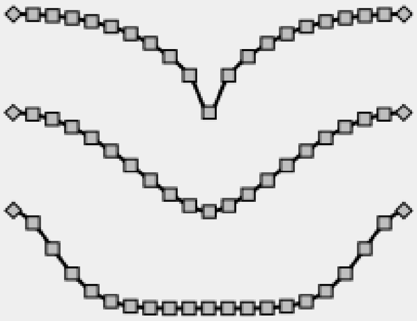
Sculpting a star. Left: A star converted to a path. Right: After selecting the center nodes and **Alt** dragging one node to the left.

The uses of *Sculpting* are endless. One can easily manipulate text into interesting shapes. In the following example, the text is sculpted in two different ways.



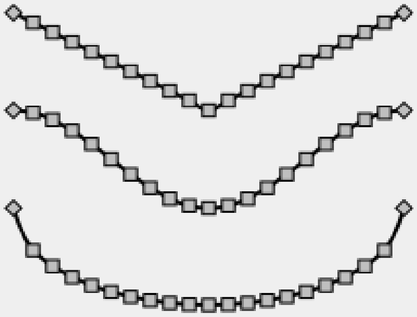
Top: Regular text. Middle: Text converted to a path with additional nodes added. First the leftmost characters were selected and one of the middle nodes dragged upward. This was repeated twice for each three letter group on the right. Bottom: The same text as above, but this time the bottommost nodes of each letter were excluded from the selection.

With a tablet, pressure sensitivity can be used to control the extent to which neighbor points are dragged. Neighbor nodes will move farther if the pressure used is greater. The tablet input device must be enabled; see the *Calligraphy Tool* section.



Sculpting of a straight line using the pressure sensitivity of a tablet. Applied pressure increases from top to bottom.

A number of different “profiles” are available. As of now, it is not possible to switch between the profiles using the GUI. You can switch by editing the *sculpting_profile* parameter in the file `preferences.xml` in your Inkscape profile directory.

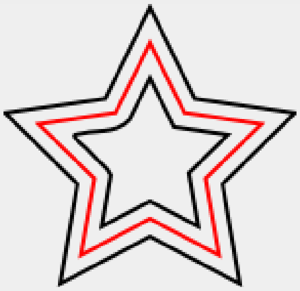


Available profiles for *Sculpting*. Top to bottom: 0, 1 (default), 2.

Path Offset Commands

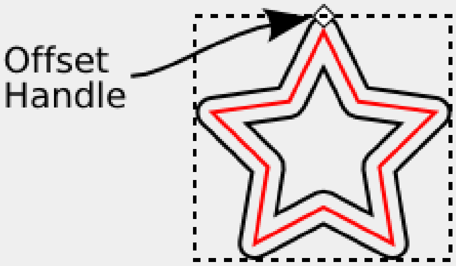
There are four commands grouped under this category, although one of them might be better thought of as a cloning tool. Each allows a path to be enlarged or reduced by moving each point perpendicular to a line tangent to the path at that point. A regular shape or text object is converted to a path automatically, except for the *Linked offset* command. The new paths are all closed, even if the original is open.

- Path → Inset (**Ctrl+(**) Inset path: Moves path inward by the *Inset/Outset* step (default 2 px).
- Path → Outset (**Ctrl+)**) Outset path: Moves path outward by the *Inset/Outset* step (default 2 px).



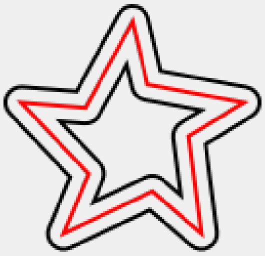
A star with an inset and an outset. The original star is red.

- Path → Dynamic Offset (**Ctrl+J**) Dynamic offset: Moves path inward or outward. A handle (viewable when *Node Tool* selected) controls the magnitude of the offset. The original path is stored so that further changes in the offset do not degrade the path. The original path is not editable after conversion. To edit, convert the dynamic offset path to a normal path with the Path → Object to Path (**Shift+Ctrl+C**) command.



A star with both a dynamic inset and a dynamic outset. The original star is red. Note that the shape of the outset star is different than in the simple outset example above.

- Path → Linked Offset (**Ctrl+Alt+J**) Linked offset: Makes a copy of a path that can then be enlarged or shrunk. A handle controls the magnitude of the offset. The original object is *not* converted to a path and remains editable, and the changes are reflected in linked copies. More than one link can be made.



A star with both a linked inset and a linked outset. The original star (red) was modified after the creation of the linked paths.

Miscellaneous Path Commands

The commands have in common that they act on the entire path, and not on a subset of a path's nodes.

- Path → Combine (**Ctrl+K**) Combine paths: Combine selected paths into a compound path.
- Path → Break Apart (**Shift+Ctrl+K**) Break apart paths: Break selected compound path(s) into simple paths.
- Path → Reverse (**Shift+R**) (keyboard shortcut only works with *Node Tool*): Reverse path: Reversing the direction of a path will affect things like the order in which nodes are selected by **Tab** and in the direction of *Markers* (e.g., arrows).
- Path → Simplify (**Ctrl+L**) Simplify path: This command reduces the number of nodes in a path while keeping the shape of the path almost the same. The larger the selection, the more aggressive the simplification. The command may be repeated. If repeated within a set time period (0.5 seconds), the simplification also becomes more aggressive. The *Simplification threshold* can be changed under the *Misc* tab in the *Inkscape Preferences* dialog (File → Inkscape Preferences... (**Shift+Ctrl+P**)).

Creating Paths

Table of Contents

Path Operations

Inkscape: Guide to a Vector Drawing Program

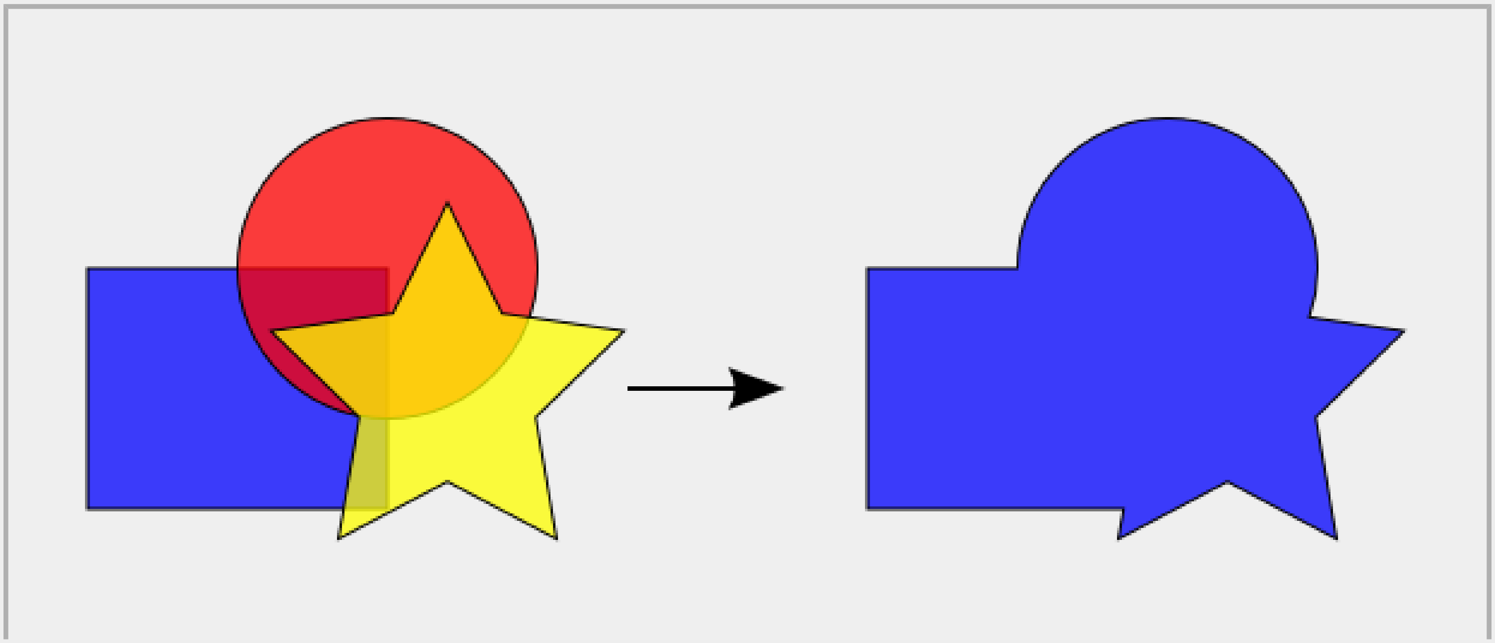
Get the book.

Path Operations

Inkscape has a number of commands to form new paths from two or more preexisting paths. The *z-order* (see the section called “Ordering Objects (Z-order)”) of the paths is important. In all cases except for the *Cut Path* command, the *Fill and Stroke* of the new path is inherited from the *bottom* path. For some operations, the *top* path can be thought of as operating on the *bottom* path; that is, part of the *bottom* path remains and the *top* path is thrown away. This is explained in more detail for each operation that it applies to below. All commands are accessible under the *Path* menu.

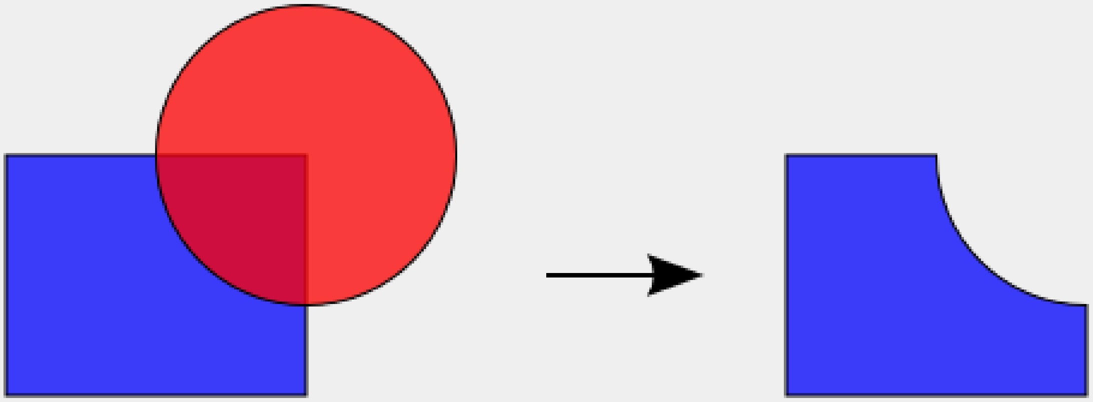
Any open paths are, for the purpose of these commands, closed by a line between the path's end points. Shape objects and text objects are automatically converted to paths.

- Union (**Ctrl**+**+**): *Modified in v0.45.* Union of one or more paths. One new path is created, containing all the areas of the original paths. A union of one path removes self-intersections, creating individual sub-paths for each section. Note that this is different from the *Path Combine* command where no nodes are lost or created.



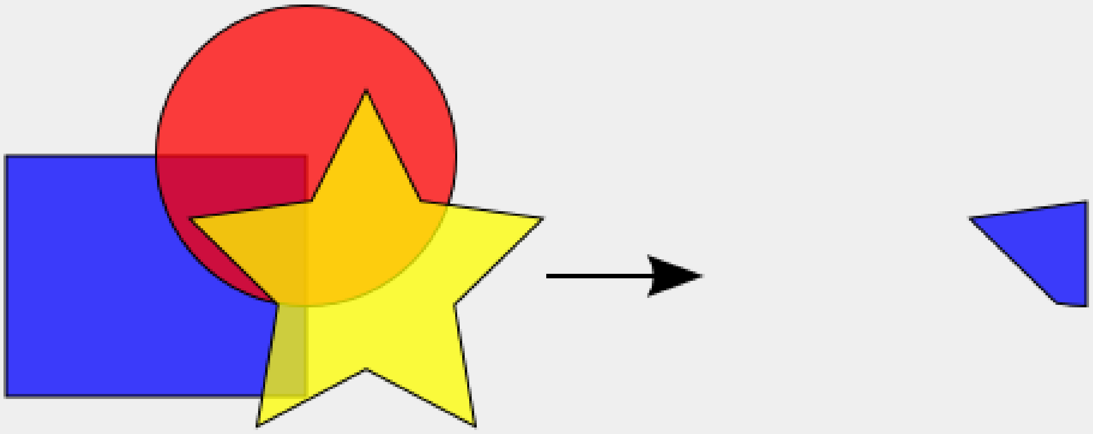
The path *Union* operation.

- Difference (**Ctrl**+**-**): Difference of two paths. The area of the top path is removed from the bottom path.



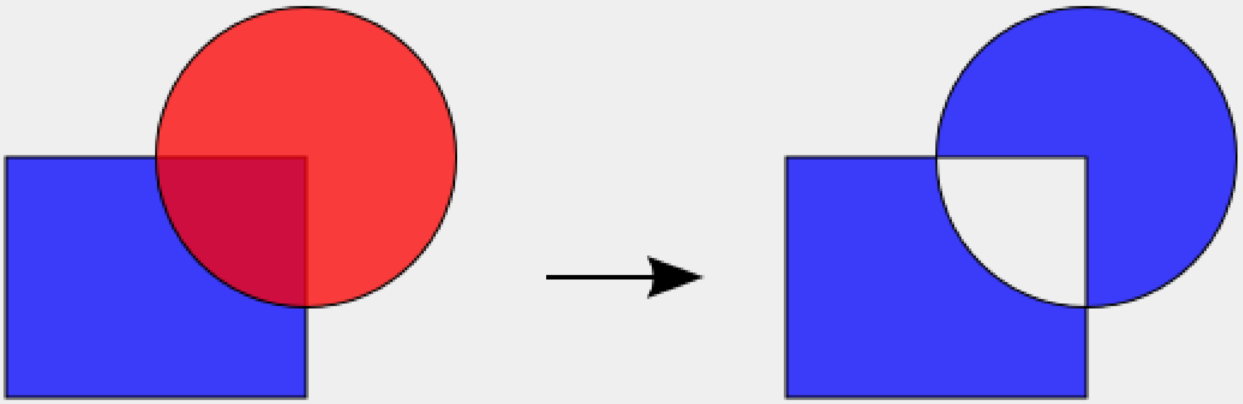
The path *Difference* operation.

- Intersection (**Ctrl**+*****): Intersection of two or more paths. The new path encloses the common area of the original paths.



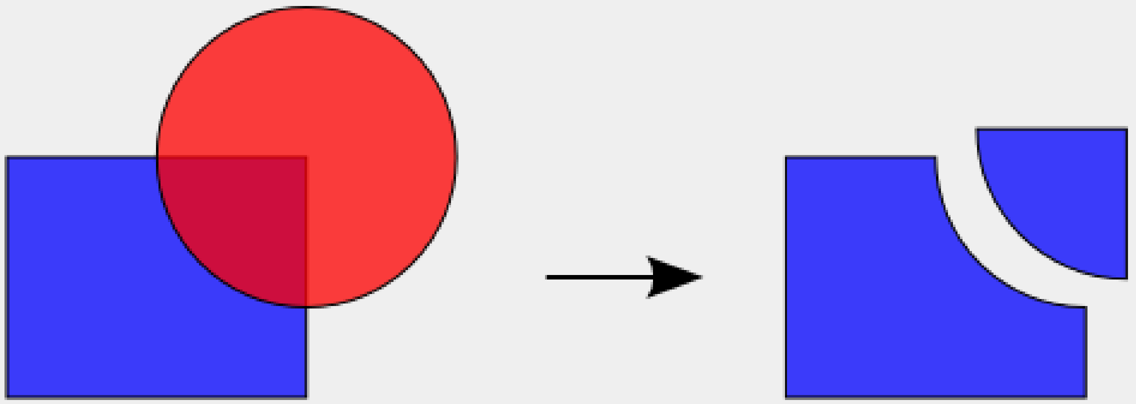
The path *Intersection* operation.

- Exclusion (**Ctrl**+**^**): Exclusion of two paths. One new path is created containing the non-overlapping areas of the original paths.



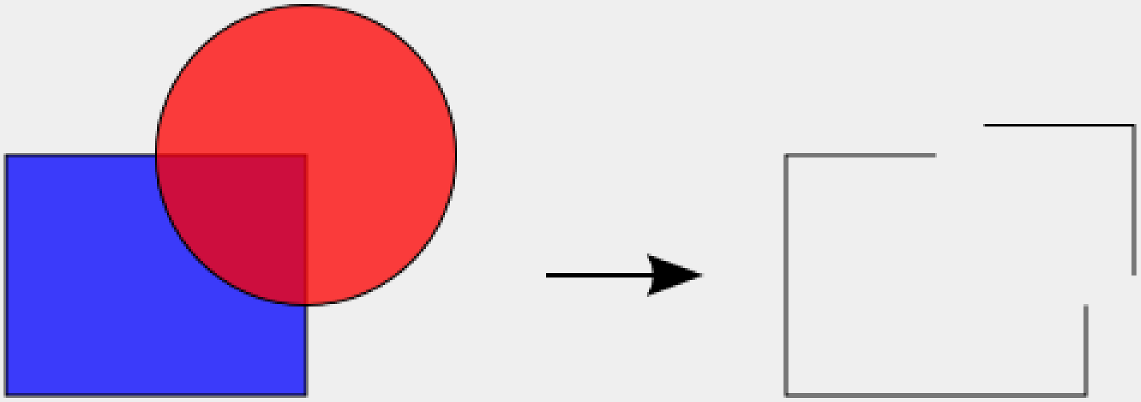
The path *Exclusion* operation.

- Division (**Ctrl**+***/***): Division of two paths. The first path is split by the second path. Two or more new paths are created.



The path *Division* operation. The upper-right corner of the “After” illustration has been shifted to show the two new paths clearly.

- Cut Path (**Ctrl**+**Alt**+***/***): Cutting by two paths. The first path is cut by the second path. Two or more new paths are created. The new paths do not have any *Fill*.



The path *Cut Path* operation. The path in the upper-right corner of the “After” illustration has been shifted to show the two new paths clearly.

Editing Paths

Table of Contents

Live Path Effects (LPE)

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Paths » Live Path Effects (LPE)

Index

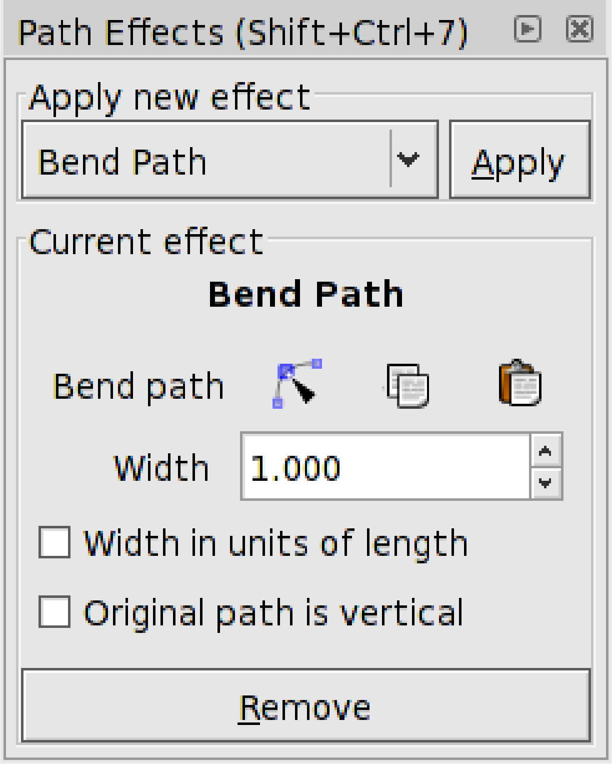
Live Path Effects (LPE)

New in v0.46.

Live Path Effects, or *LPE* for short, is a system for applying some kind of effect to a path. Inkscape stores the path so that it can be modified at a later time with the effect automatically updating. The original path and attached data are stored in the *SVG* file in the Inkscape name space so it doesn't display in other *SVG* renderers. The result will display.

Inkscape v0.46 only includes a few LPEs. With a solid framework in place, it is planned to migrate most of the path effects found under the *Effects* menu to LPEs. This has two advantages: The effects will be faster to render and the original path is stored for future editing.

To use an *LPE*, select a path (the "skeleton" path), call up the *Path Effects* dialog (Path → Path Effects... (**Shift**+**Ctrl**+**7**)), select the desired effect from the drop-down menu in the dialog, and then click the *Apply* button.



Path Effects (Shift+Ctrl+7)

Apply new effect

Bend Path ▾ | Apply

Current effect

Bend Path

Bend path

Width 1.000

☐ Width in units of length

☐ Original path is vertical

Remove

Path Effects dialog with a path selected.

When an object is selected that was generated by an LPE and the *Node Tool* is enabled, the original skeleton path is shown in red. This path can be edited like any other path.

Several of the effects require an additional path. In this case, a path is automatically generated (a straight green line). To see this path, either click on the node editing icon ( ) in the dialog or press **7** while the object is selected and the *Node Tool* is active. The path is fully editable like any other path. The path can be replaced by another through pasting from the *clipboard* using the paste icon ( ) in the dialog. It can also be copied to the *clipboard* by using the copy icon ( ) in the dialog.

An LPE can be converted to regular paths by using the Path → Object to Path (**Shift+Ctrl+C**) command.

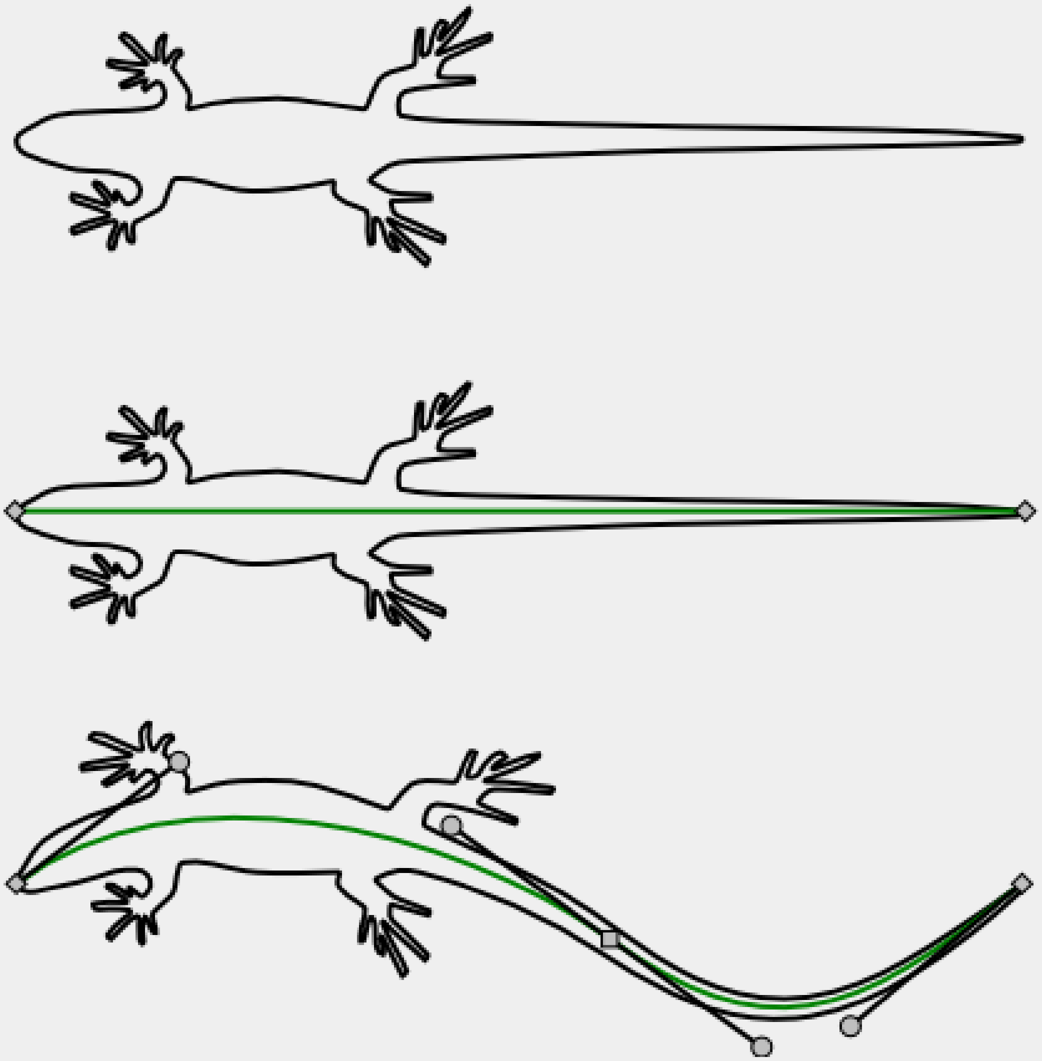
Bend Path

This effect takes an existing path and allows one to “bend” it in a well defined way via a second control path. The control path is automatically created. The style of the original path is used for the style of the bent path.

To apply the effect, with the source path selected, select the *Bend Path* option from the drop-down menu in the *Path Effects* dialog. Click on the *Apply* button. The path will turn red. A red path always corresponds to the original source or skeleton path. Next, click on the node icon ( ) in the *Path Effects* dialog. A green, horizontal path will appear in place of the red path. This path controls the bending. It can be manipulated in all the ways that a regular path can be including adding new nodes and dragging the path.

Warning
It is easy to apply this *Path Effect* multiple times... leading to bizarre behavior. Check the SVG tree with the XML Editor dialog if strange things are happening.

The effect assumes that the path to distort is orientated in the horizontal direction. If it is in the vertical orientation, one can check the *Original path is vertical box*. This distorts the aspect ratio. Restore the aspect ratio by checking the *Width in units of length* box. The size of the bent path can be changed by changing the size of the control path.



Demonstration of the *Bend Path* effect. Top: Source path. Middle: After applying the effect and enabling editing of the control path. Bottom: After adding a node and adjusting the control path.

The width of the bent path can be altered with the *Width* entry box.

Pattern Along Path

This effect puts one or more copies of one path ("pattern") along a second, control or skeleton path. The resulting object takes the attributes (*Fill*, etc.) of the skeleton path.

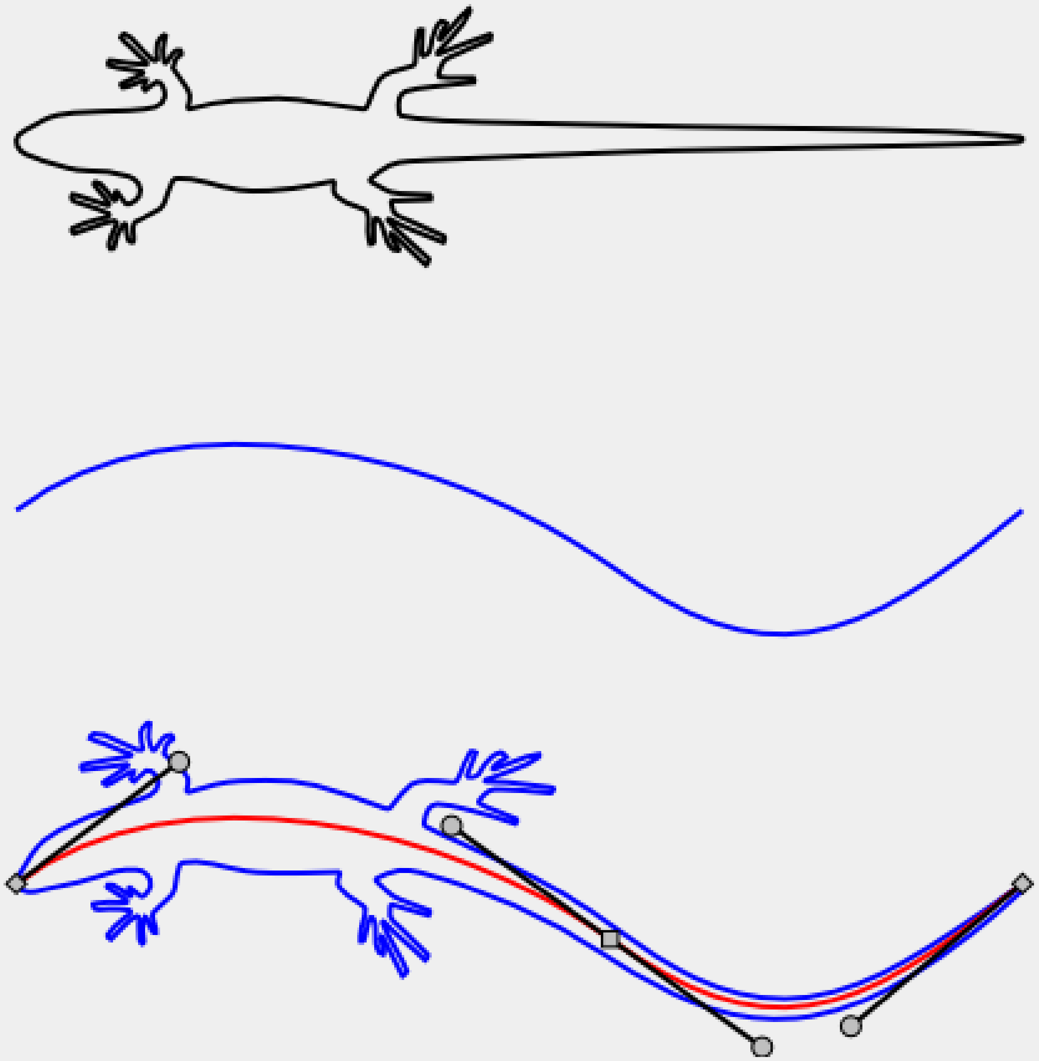
This LPE duplicates much of the functionality of the *Pattern along Path* effect. The advantage of using the LPE version is that both the pattern and the skeleton path can be edited at a later time. The disadvantages are that only paths can be used for the pattern and that there are fewer options. One subtle difference is that the LPE version will bend straight lines drawn with two nodes while the other version leaves them straight.

> **Note**
> Only a path can be used as a pattern. Many objects such as *Rectangles*, *Ellipses* and text must first be converted to a path (Path → Object to Path (**Shift**+**Ctrl**+**C**)).

To put a pattern on a path:

1. **Copy the pattern:** Select the pattern and copy it to the *clipboard* (Edit → Copy (**Ctrl**+**C**)). The pattern must be a single path.
2. **Select the skeleton path:** Only one can be selected.
3. **Apply the effect to skeleton path:** In the *Path Effects* dialog, select *Pattern Along Path* and click on the *Apply* button.
4. **Paste pattern:** Click on the *Paste* ( ) icon in dialog.

The *bounding box* of the pattern is used for placing the pattern along the path, with the *bounding box* of one pattern copy touching the *bounding box* of the next copy (if no additional spacing is specified).



Demonstration of the *Pattern Along Path* effect. Top: Pattern path. Middle: Skeleton or control path. Bottom: After pasting pattern to skeleton path and enabling editing of the skeleton path. Note that the created object takes the attributes of the skeleton path.

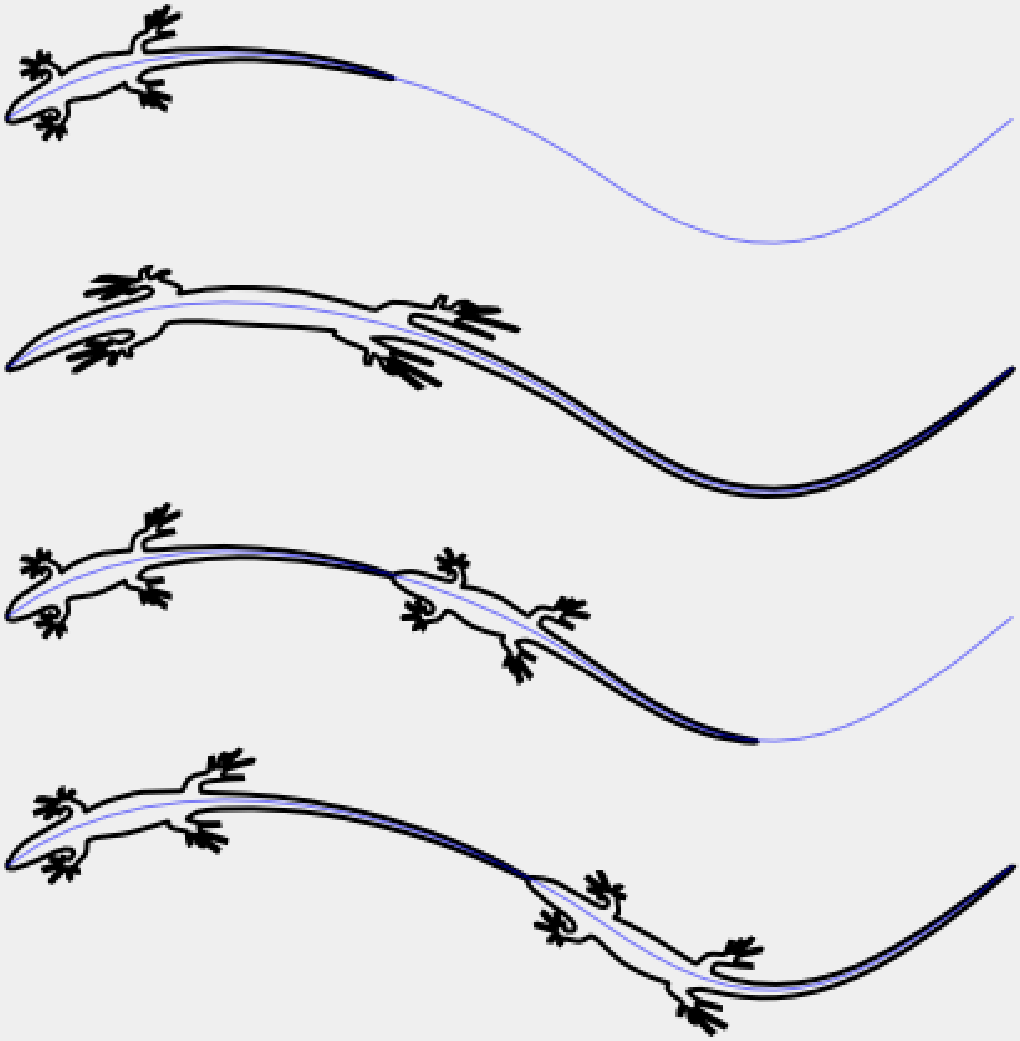
To edit the pattern, click on the node editing icon ( ) in the dialog. A temporary green copy of the pattern will appear at the pattern's original location. Any edit to this copy will be reflected in the final object.

To edit the skeleton path, select the object with the *Node Tool*. A temporary red copy of the skeleton path will appear. This can be edited as any other path.

A different pattern can be applied to the skeleton path by copying the pattern to the *clipboard* ((Edit → Cut (**Ctrl+X**) or Edit → Copy (**Ctrl+C**)) and then clicking on the *Paste* ( ) button in the dialog.

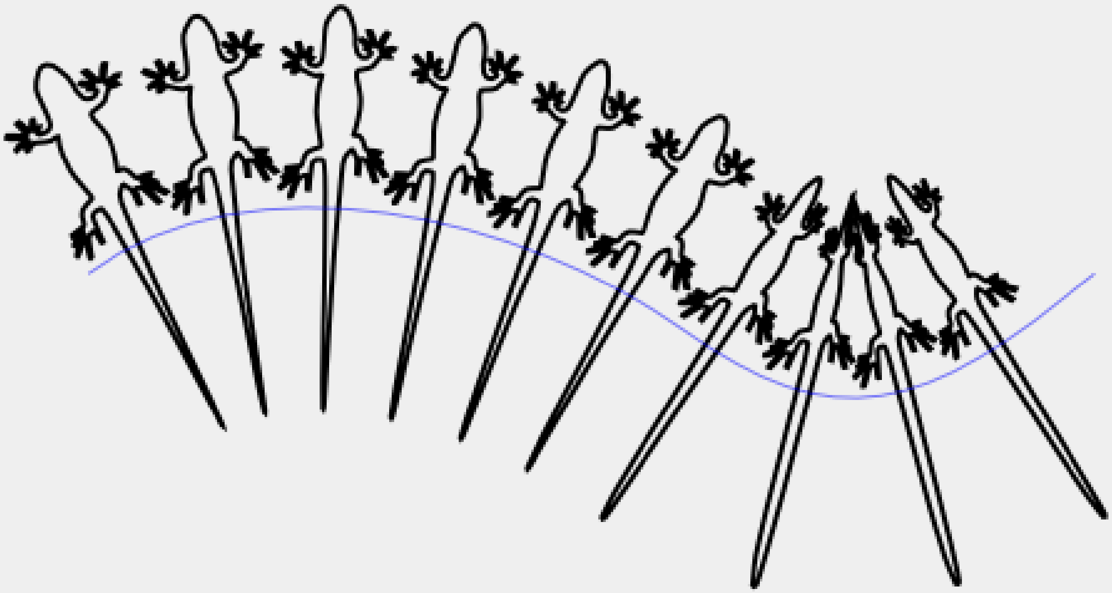
A copy of the original pattern can be placed on the *clipboard* by clicking on the *Copy* ( ) button in the dialog. The copy will have all attributes unset.

The *Pattern copies* drop-down menu has options to stretch the pattern to the path length and/or to put multiple copies along the skeleton path.



A small lizard is put on a path with *Pattern copies* set to, from top to bottom: *Single*; *Single, stretched*; *Repeated*; *Repeated, stretched*.

The LPE assumes that the pattern is drawn horizontally. This can be changed to vertical by checking the *Pattern is vertical* box.

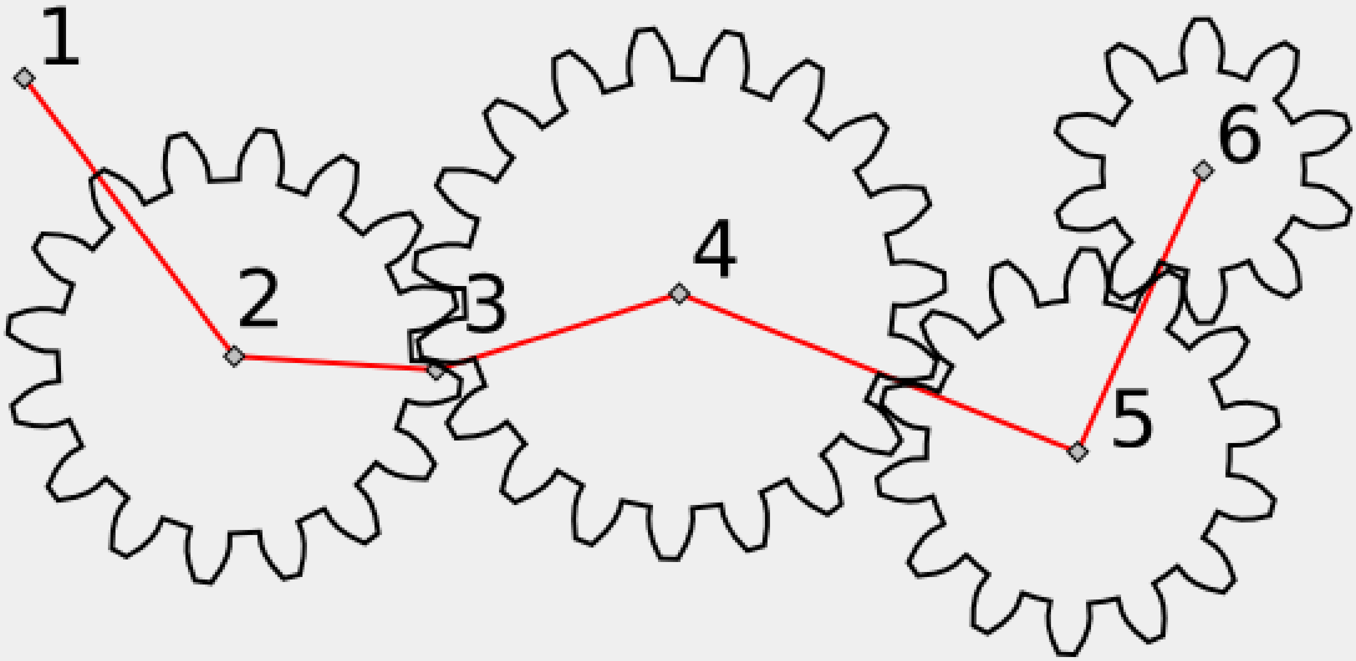


A small lizard is put on a path with the *Pattern is vertical* box checked.

Gears

This effect draws a series of intermeshed gears. It is more of a toy effect, designed to demonstrate the possibilities of LPEs. The *Gear Effect* can also be used to draw gears with a bit more control.

The effect uses the nodes of a path to determine how the gears are drawn. At least three nodes are needed to specify the first gear. Additional gears require one additional node each. Some nodes may be skipped if they would result in impractical gears.



An example of using the *Gear* LPE. The original path is shown in red. Point 1 defines the orientation of the first gear. Point 2 defines the center of the first gear. Point 3 defines the radius of the first gear. Points 4, 5, and 6 determine the centers of additional gears. The radius of the second gear is determined by subtracting the radius of the first gear from the distance between the centers of the first and second gears.

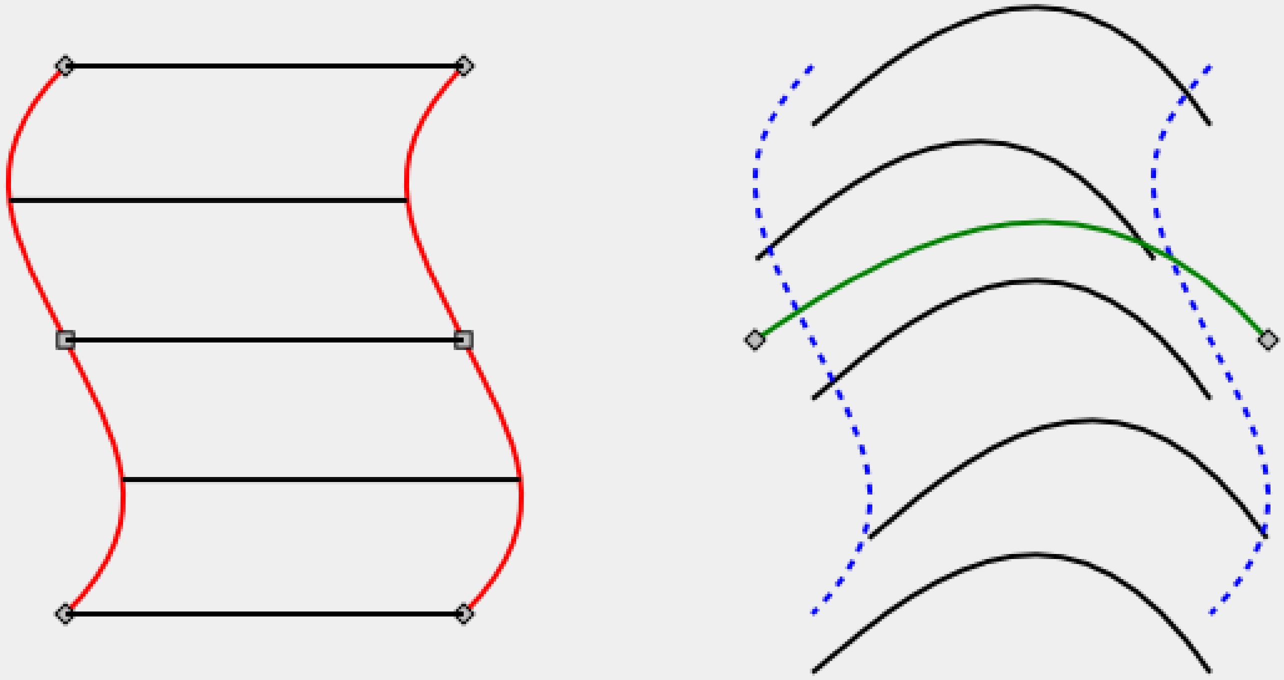
Two parameters are available: *Teeth* determines the number of teeth on the first gear. *Phi* determines the *Pressure Angle* of the gears. For real gears the *Pressure Angle* is typically 14.5, 20, or 25 degrees. Note: The default angle is 5 degrees, not a very realistic value.

Stitch Sub-Paths

This effect draws a series of *Stroke paths* between points on sub-paths. Some of the things it is useful for are drawing hatched shading and for drawing hair.

To stitch a sub-path:

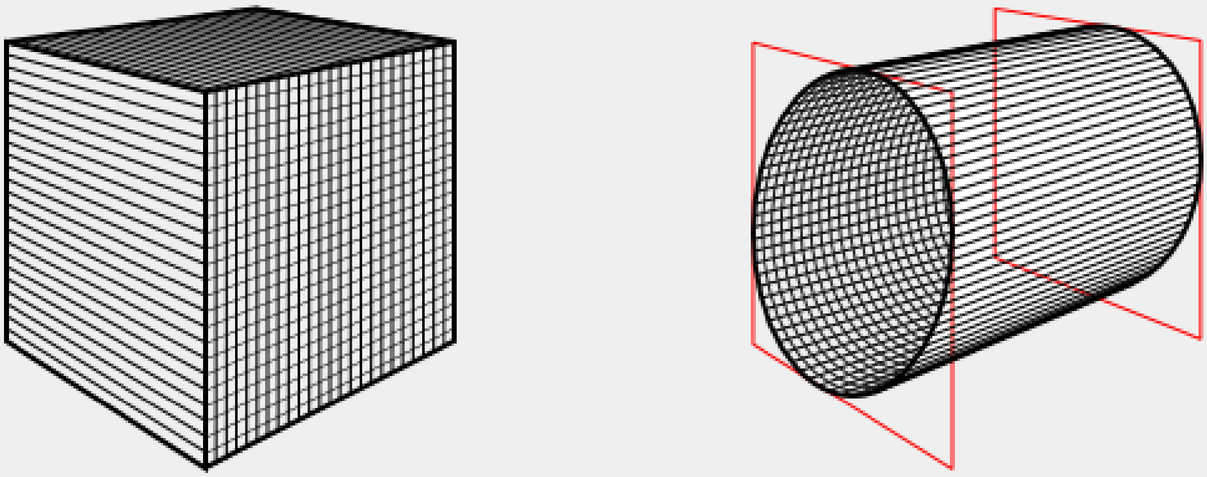
1. **Draw the sub-paths:** Draw two simple paths. Combine into a compound path consisting of two sub-paths using Path → Combine (**Ctrl+K**). The two sub-paths should be drawn in the same direction. If not, use the Path → Reverse (**Shift+R**) command on one of the sub-paths (prior to combining) to reverse its direction.
2. **Apply the effect to compound path:** In the *Path Effects* dialog, select *Stitch Sub-Paths* from the *Apply new effect* menu and click on the *Apply* button.
3. **Adjust *Stroke path*:** Click on the node editing icon ( ) to edit the *Stroke path*.



A basic example of using the *Stitch Sub-Paths* LPE. Left: A simple path was drawn and duplicated.

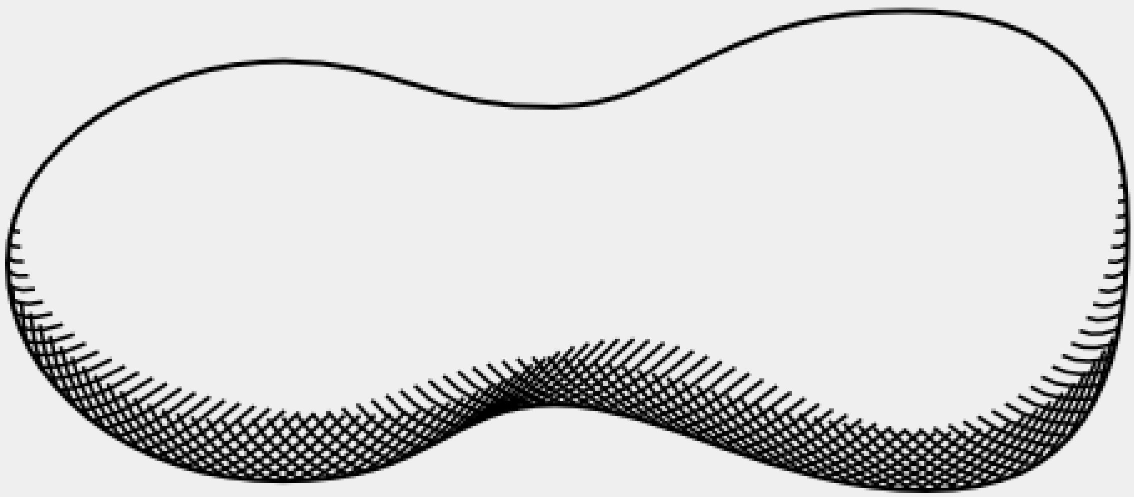
The two paths were then combined into a compound path (Path → Combine (**Ctrl+K**)). The effect was then applied. The red lines are shown when the LPE object is selected with the *Node Tool* enabled. Right: The *Stroke path* (green) has been enabled via clicking on the node-editing icon ( ) and the path adjusted. The original sub-paths (not normally visible) are shown by blue-dashed lines. Note how the ends of the *Stroke path* are no longer on the original sub-paths. This is because Inkscape uses the center on the left and right of the *bounding box* to place the *Stroke path*.

The *Stitch Sub-Paths* effect can be used to create the hatchings typically used in engravings as shown in the following example. While the *Interpolate Effect* could be used to created some of the shadings, it can not create the horizontal shadings on the cylinder below (likewise, the *Stitch Sub-Paths* effect can not easily create the precise circular hatching inside the cylinder).



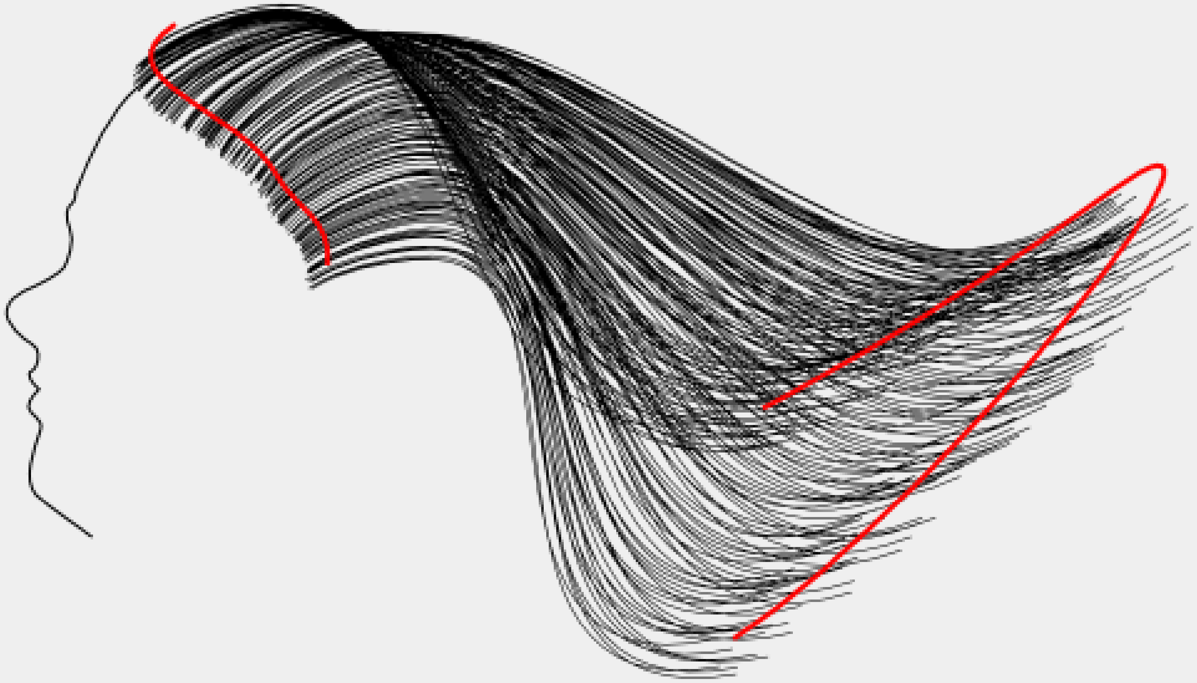
A simple example of using the effect for hatchings. Left: The *Box Tool* was used to draw a box. Then the *Bezier Tool* was used with snapping to draw paths on both sides of a box face. The paths were combined and the effect applied. For the right side of the box, the effect was used twice, once for the horizontal lines and once for the vertical lines. Right: Two ovals were drawn to fit in two opposite side of a box. The ovals were converted to paths and split into two sections. The left sections were used for the inside hatching and the right sections for the outside. The circular hatching inside the cylinder was done by using the *Interpolate Effect*.

By varying both the sub-paths and the *Stroke path* quite complicated hatchings can be created. The hatchings can be clipped to limit their range. The *Tweak Tool* could also be used to refine the hatchings if the hatchings are converted to stroked paths (see Chapter 10, *Tweak Tool*).



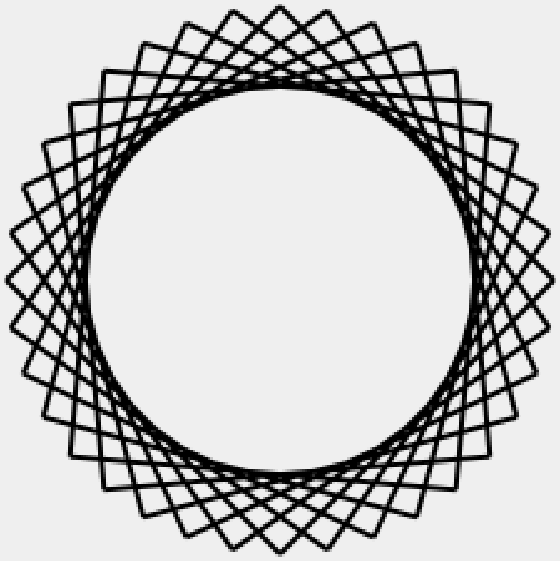
Hatching created by: 1. Copying the bottom half of the object's path. 2. Duplicating the copy with an offset to the upper left. 3. Applying the *Stitch Sub-Paths* LPE with 50 paths. 4. Adjusting the sub-paths and *Stroke path*. 5. Repeating with the duplicate offset to the upper right. 6. Grouping the hatchings and clipping with a copy of the original path.

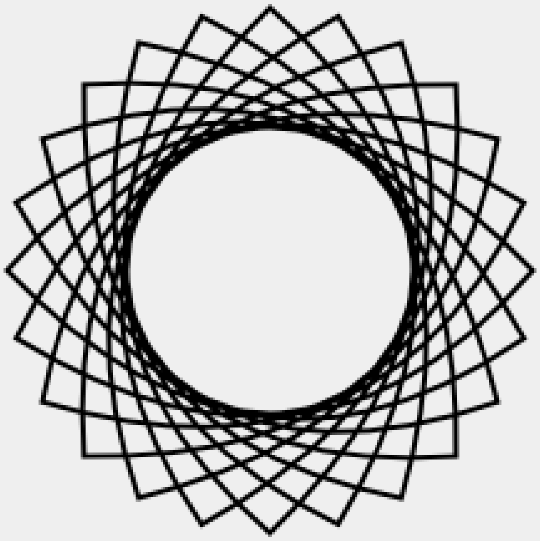
The *Stroke Sub-Paths* LPE has options that add random shifts to the start and end of each stitching path. The “variance” options can be used to draw hair as shown below. Each variance has a dice icon (🎲) next to it which, if clicked, sets a new starting random number seed. This will change the random shifts but keep the average shift the same.



Drawing hair: The sub-paths are shown in red. The number of paths was set to 200 and the following variances were used: Start edge: 0.02, Start spacing: 0.10, End edge: 0.10, End spacing: 0.10.

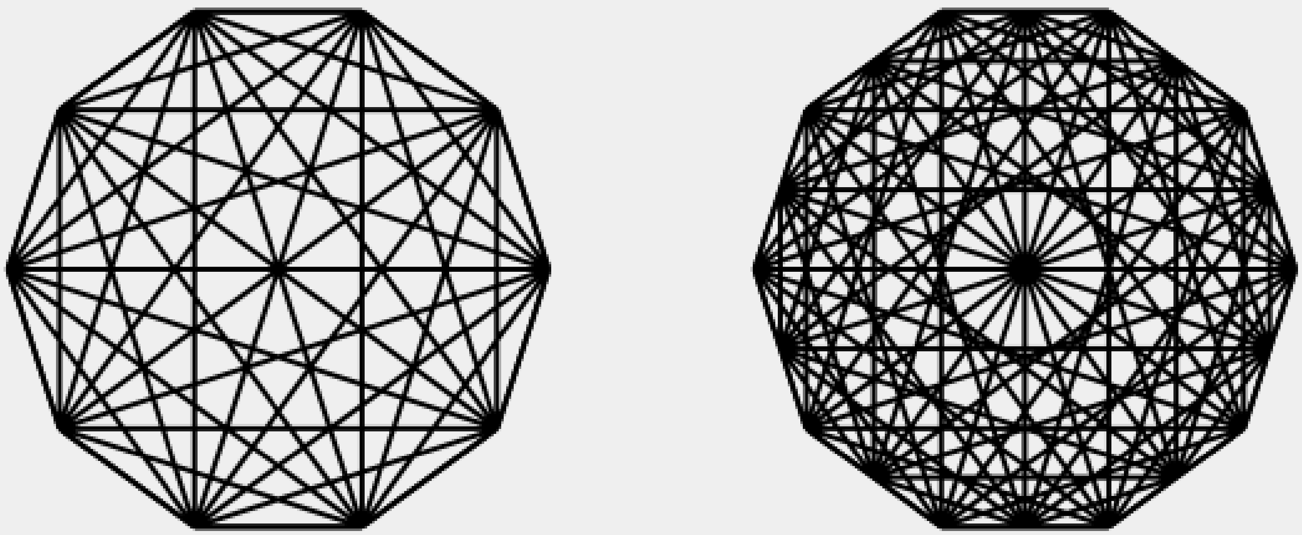
Interesting geometric patterns can be created with this effect as shown below.





For these designs, a circle was converted to a path and then duplicated. The path copy was rotated and then the two paths combined into a compound path. Left: The duplicate path was rotated 45° and the *Number of paths* was set to 37 (the first and last path are on top of each other). Right: The duplicate path was rotated 150° and the *Number of paths* set to 25. The *Stroke path* was bent until the path ends joined new paths.

What if multiple sub-paths are used? Each sub-path will be connected to every other sub-path by the specified *Number of paths*. This can be used to created some interesting patterns.



An example of using the multiple sub-paths. A decagon was drawn with the *Star Tool* tool. The decagon was converted to a path (Path → Object to Path (**Shift+Ctrl+C**)). The path was broken into 10 sub-paths by using the *Node Tool* (select node, click on in *Tool Controls*). Finally, the LPE *Stitch Sub-Paths* was applied. Left: *Number of paths* set to two. Right: *Number of paths* was set to three. Higher numbers yield rendering errors in Inkscape (but display correctly in Firefox 3 and Opera 9.26).

Path Operations

Table of Contents

Chapter 8. Text

Creating Text

Text is created with the *Text Tool*. To add text to a document, select the tool by clicking on the icon (**F8** or **t**) in the *Tool Box*.

> **Note**
>
> The *Text and Font* dialog has an entry box for entering text on the *Text* tab. This entry box will not be active unless some preexisting text object is selected or at least one character is entered into a new text object using the *Text Tool*. This is because a text object must be created first. Simply clicking on the screen with the *Text Tool* does not create an object.

There are two ways to add text to an Inkscape drawing. The first is as *regular* text. In this case, as text is typed, the *text box* grows to accommodate the text. Line breaks must be manually added.

The second way to enter text is as a *flowed* text object. The text is typed into a presized rectangular *text box*. Line breaks are automatically made. Once the text fills the box, no more text can be entered unless the box is enlarged. The *flowed* text object includes both the box and text and is thus moved and transformed as such. For a discussion of *flowed* text in an arbitrary shape, see the last section in this chapter.

A few things to note that apply to both entry methods:

- If you click on an already existing text object with the *Text Tool*, the text object will be selected and the cursor will be placed between the letters closest to where you clicked. The text can then be edited.
- Many of the keyboard shortcuts will not work while in the enter text mode. The **+** and – keys on the numeric keypad will type the corresponding characters if **Num Lock** is on; otherwise, they retain their zoom functions.
- Special characters can be entered by switching to *Unicode* mode. To toggle between normal and *Unicode* modes use **Ctrl**+**U**. Typing any non-hexadecimal character or **Esc** while in Unicode mode will return you to normal mode. Once in *Unicode* mode, you can enter by typing the two to four hexadecimal digits corresponding to the character you desire. Hit the **Space** bar to register the

character and to start entering another *Unicode* character.

This is "Japanese" written in Japanese: 日本語.

A sentence with Kanji characters. The characters were entered by typing: **Ctrl**+**U** 65e5 **Space** 672c **Space** 8a9e **Esc**. (You must have a Kanji font installed to see the characters.)

- One problem you may encounter with saving text in *SVG* files as text objects is that if the font is not available on the computer where the file is to be viewed, it may not be visible. To avoid this problem, you can convert the text to a path. This, though, will prevent the text from being edited as a text object.

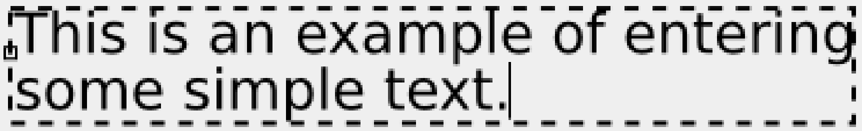
  Tip

  Make a duplicate of the text before you convert the text to a path. Put the copy on a separate layer and make that layer invisible. Then if you ever need to edit the text as a text object, you will have a copy available.

- When a text object is selected, a small square is shown at the left of the first line's baseline. This is the *baseline anchor*, which is used for snapping and alignment.

Entering Regular Text

To add *regular* text, click on the document where you desire the text to start. You should see a cursor (blinking bar) indicating you are in the text enter mode and showing where the text will start. To add text, just start typing. You can enter multi-line text by inserting a carriage return. The *text box* will grow as text is entered.



This is an example of entering
some simple text.

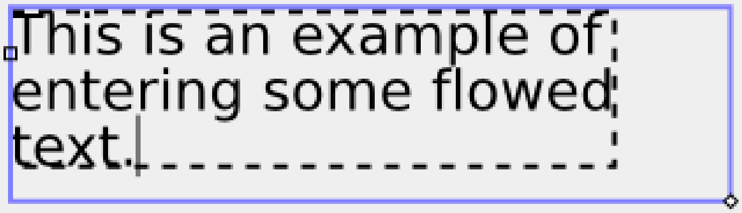
Entering *regular* text.

Entering Flowed Text

Warning

Flowed text was a draft *SVG* 1.2 specification that will not in the end be adopted. The text is not likely to be viewable by other renderers. In addition, some programs will not render any of a file with *flowed* text (Squiggle, for example). You can convert the *flowed* text to a *regular* text object before saving to avoid this problem.

To add *flowed* text, click-drag on the document with the *Text Tool* to create a blue rectangle box for the text. Once the box is drawn, you can move the cursor into the box area and begin to type. Carriage returns are automatically made. If the text fills the box, you can not add more text with out enlarging the box. The box can be enlarged or the proportions changed by dragging on the handle at the lower-right side with the *Text Tool*, *Node Tool*, or any of the *shape tools*; however, the *text box* will only be shown when using the *Text Tool*. Use the **Ctrl** key while dragging to constrain the change in box size to a horizontal or vertical direction. The box and text can be moved together.



This is an example of entering some flowed text.

Entering *flowed* text.

Chapter 8. Text

Table of Contents

Selecting Text

Inkscape » Text » Selecting Text

← Index →

Selecting Text

Editing and applying attributes to text requires positioning the cursor or selecting text. The following methods are available when using the *Text Tool*:

- Moving text cursor:
  - **Left**+**Mouse Click**: Position cursor under pointer.
  - **Arrows**: Move forward/backward one character, move up/down one line.
  - **Ctrl**+**Arrows**: Move forward/backward one word, move up/down one line.
  - **Home**: Move to start of line.
  - **End**: Move to end of line.
  - **Ctrl**+**Up Arrow**: Move up one paragraph.
  - **Ctrl**+**Down Arrow**: Move down one paragraph.
  - **Ctrl**+**Home**: Move to start of text.
  - **Ctrl**+**End**: Move to end of text.
- Selecting text:
  - **Left**+**Mouse Drag**: Select text under drag.
  - **Double Left**+**Mouse Click**: Select word under pointer.
  - **Triple Left**+**Mouse Click**: Select text line under pointer.
  - **Shift**+**Arrows**: Move start or end of selection one character in arrow direction.
  - **Shift**+**Ctrl**+**Left/Right Arrows**: Move start or end of selection to beginning of word in arrow direction.
  - **Shift**+**Ctrl**+**Up/Down Arrows**: Move start or end of selection to character in line above or below current line.
  - **Ctrl**+**A**: Select all text in current text object.

← Creating Text

↑ Table of Contents

→ Editing Text

Get the book.

Inkscape » Text » Editing Text

Index

Editing Text

Editing text is done with the *Text Tool*. As already mentioned, clicking on existing text will select that text object and enable editing. All the normal text editing keys function as expected: **Backspace**, **Arrows**, **Enter**, etc. Cut and paste also work (see above for selecting text): Cut: **Ctrl**+**X**, Copy: **Ctrl**+**C**, and Paste: **Ctrl**+**V**.

An alternative way to edit text is to use the *Text* tab of the *Text and Font* dialog (Text → Text and Font... (**Shift+Ctrl+T**)). Editing text in the tab has a couple of advantages, the most important of which is that the text is spell checked as it is typed. This option requires that *GtkSpell* be installed and it must be compiled in (make sure that you have the libgtkspell development library installed if you compile the program yourself).

Selecting Text

Table of Contents

Formatting Text

Get the book.

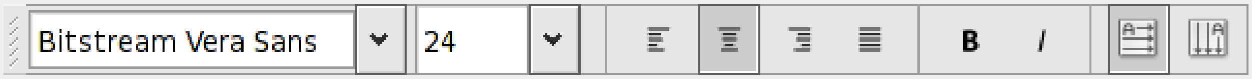
Formatting Text

Text in text objects can be formatted. This section covers changing the font, style, size, orientation, justification, and inter-line spacing. Positioning of individual characters (e.g., *kerning*) is discussed in the following section. Specifying the fill (color, pattern, etc.) of text is covered in Chapter 9, *Attributes*.

There are three methods to format text. The first is to use the items in the *Text Tool*-*Tool Controls*, the second is to use keyboard shortcuts, and the third is to use the *Text and Font* dialog.

Formatting with the Tool Controls

The font family, font size, justification, font style, and text direction can be changed from the *Text Tool*-*Tool Controls*. If characters within a text object are selected, the changes apply only to those characters. Otherwise, the changes apply to all selected text objects (to select more than one text object, switch temporarily to the *Select Tool*). Changes made when no text object is selected (or a new blank text object is created) change the default style. (Except flowed-text: Bug?)


Bitstream Vera Sans

24

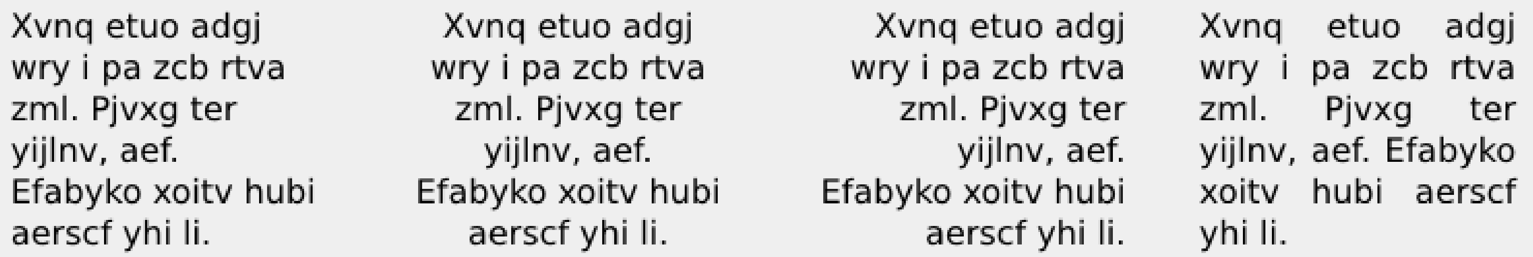
The *Text Tool*-*Tool Controls*.

Font Family: The leftmost drop-down menu allows one to select the font family. When activated, the menu shows samples of the various fonts available to Inkscape. (The text used for the sample can be customized by editing your `.preference.xml` file.)

Font Size: The adjacent drop-down menu sets the font size in pixels. To select a size that is not in the menu, simply type the number in. The mouse scroll wheel can also be used to select the size when the cursor is over the menu (doesn't always work: Bug).

Justification: Text can be justified by clicking on the following icons:

- ≡ Left justified.
- ≡ Centered.
- ≡ Right justified.
- ≡ Left and right justified.


Xvnq etuo adgj wry i pa zcb rtva zml. Pjvxg ter yijlnv, aef. Efabyko xoitv hubi aerscf yhi li.

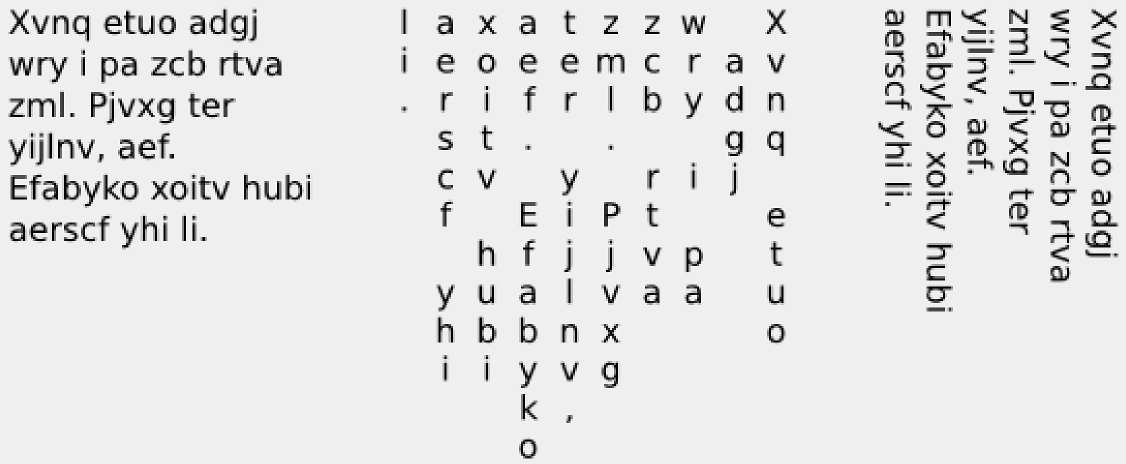
Left, center, right, and both left and right justified text.

> **Note**
>
> Only *flowed* text can be both left and right justified at the same time.

Style: Text can made **Bold** or *Italicized* (if the chosen font supports these options). These are the styles supported by the [SVG](SVG) standard.

Orientation: The orientation of the text can be chosen by clicking on one of the following icons:

- Horizontal.
- Vertical.


Xvnq etuo adgj wry i pa zcb rtva zml. Pjvxg ter yijlnv, aef. Efabyko xoitv hubi aerscf yhi li.

Horizontal (left), vertical (center), and rotated (right) text.

The vertical choice is mostly applicable to languages written from top to bottom and from right to left, such as Chinese, Japanese, or Korean. See the above figure to see the difference between vertical text and text rotated by 90 degrees. A block of text can only have one orientation.

Formatting with the Keyboard

Font style, interline spacing, and inter-character spacing for a line of text are changeable via the keyboard.

Text attributes: Applies to selected text.

- **Ctrl**+**B**: Toggle **Bold** on/off. Text must be selected.
- **Ctrl**+**I**: Toggle *Italics* on/off. Text must be selected.

Letter spacing: Commands apply to line with cursor or selection for *regular* text and applies to paragraph for *flowed* text. The adjustments are specified in *Screen pixels* and thus depend on the zoom level.

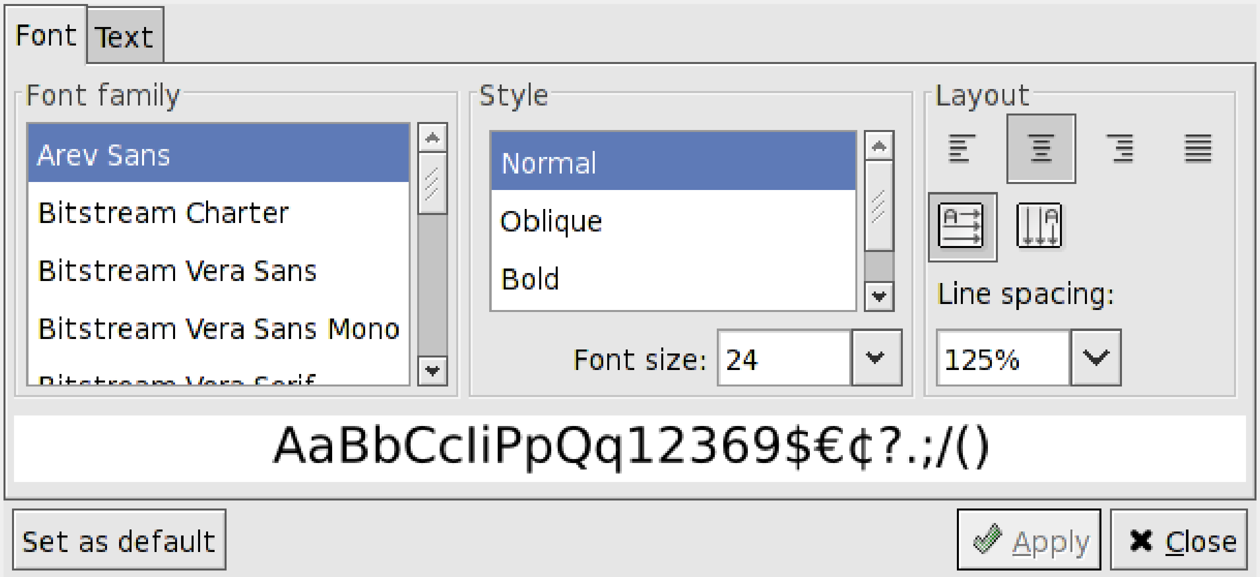
- **Alt**+**>**: Expand line or paragraph by one *Screen pixel*.
- **Alt**+**<**: Contract line or paragraph by one *Screen pixel*.
- **Shift**+**Alt**+**>**: Expand line or paragraph by ten *Screen pixels*.
- **Shift**+**Alt**+**<**: Contract line or paragraph by ten *Screen pixels*.

Line spacing: Commands apply to entire text object. The adjustments are specified in *Screen pixels* and thus depend on the zoom level.

- **Ctrl**+**Alt**+**>**: Make text object one *Screen pixel* taller.
- **Ctrl**+**Alt**+**<**: Make text object one *Screen pixel* shorter.
- **Shift**+**Ctrl**+**Alt**+**>**: Make text object ten *Screen pixels* taller.
- **Shift**+**Ctrl**+**Alt**+**<**: Make text object ten *Screen pixels* shorter.

Formatting with the Text and Font Dialog

Font family, style, and size; justification and line spacing can be changed via the *Text and Font* dialog (Text → **T** Text and Font... (**Shift+Ctrl+T**)). Changing line spacing is only accessible from the dialog. The dialog is split into three parts, the first for selecting the *Font Family*, the second for selecting the *Style* and *Size*, and the third for selecting the *Layout*. A preview of the proposed changes is shown on the dialog. Changes are not made to the selected text until the *Apply* button is clicked. Changes apply to the whole text object unless a portion of the text is selected in which case only the selected text is affected. New text can be forced to use a specified format by setting the default with the *Set as Default* button.



Arev Sans
Bitstream Charter
Bitstream Vera Sans
Bitstream Vera Sans Mono
Bitstream Vera Serif

Normal
Oblique
Bold

Font size: 24

Line spacing: 125%

AaBbCcIiPpQq12369$€¢?.;/()

Text and Font dialog.

Font Family: The names of all the Font Families installed on your computer should be shown. Click on the name to select a new Font Family.

Style and Font Size: After a Font Family is selected, the Style box will show the available styles for that family. Not all families have all styles. *Oblique* is often used instead of *Italics*.[12]

The Font Size can be selected from the drop-down menu. Font size is measured in *points*. A font size that is not listed can be typed into the Entry box. If you choose a large font, the preview window may not be large enough to show the entire text. In this case, the preview window can be made larger by increasing the size of the dialog window.

Layout: The alignment and orientation of text can be changed by clicking on the appropriate icon. See the previous section on using *Tool Controls* for more details on the use of the justification and orientation options.

Text can be:

- Left justified.
- Centered.
- Right justified.
- Left and right justified.

The orientation of the text can be:

- Horizontal.
- Vertical.

Line spacing (the distance between text *baselines*) can be changed by the *Line spacing* entry box.

[12] An *Oblique* font is usually a regular font that has been skewed. A true *Italic* font has several characters (e.g., 'a' and '*a*') that have different designs.

Editing Text

Table of Contents

Kerning, Shifting, and Rotating Characters

Get the book.

Inkscape: Guide to a Vector Drawing Program

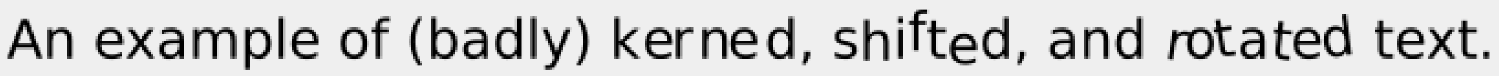
Get the book.

Inkscape » Text » Kerning, Shifting, and Rotating Characters

Index

Kerning, Shifting, and Rotating Characters

Individual characters in a line of *regular* (but not *flowed*) text may be shifted left or right to change their *kerning*, shifted up or down, or rotated. (Both *regular* and *flowed* text do utilize the internal *kerning* that is included with fonts.)



An example of (badly) kerned, shifted, and rotated text.

Text showing individual character manipulation.

Kerning and Shifting

Changing the *kerning* and shifting characters up and down are treated the same in Inkscape. If no characters are selected, the following commands apply to the characters in a line of text after the current cursor point. If characters are selected, the commands apply to only those characters. The shifts are in *Screen pixels*; thus, the zoom level will affect the magnitude of the shift.

- **Alt**+**Arrows Key**: Shift character(s) by one *Screen pixel* in arrow direction.
- **Shift**+**Alt**+**Arrows Key**: Shift character(s) by ten *Screen pixels* in arrow direction.

All *manual* kerning can be removed with the Text → Remove Manual Kerns command.

Rotating

If no characters are selected, the following commands apply the character after the current cursor point. If characters are selected, the commands apply to those characters. Some of the rotations are in *Screen pixels*; thus, the zoom level will affect the magnitude of the rotation.

- **Alt**+**[**, **Alt**+**]**: Rotate character(s) counter-clockwise, clockwise by one *Screen pixel*.
- **Ctrl**+**[**, **Alt**+**]**: Rotate character(s) counter-clockwise, clockwise by 90 degrees.

← Formatting Text

↑ Table of Contents

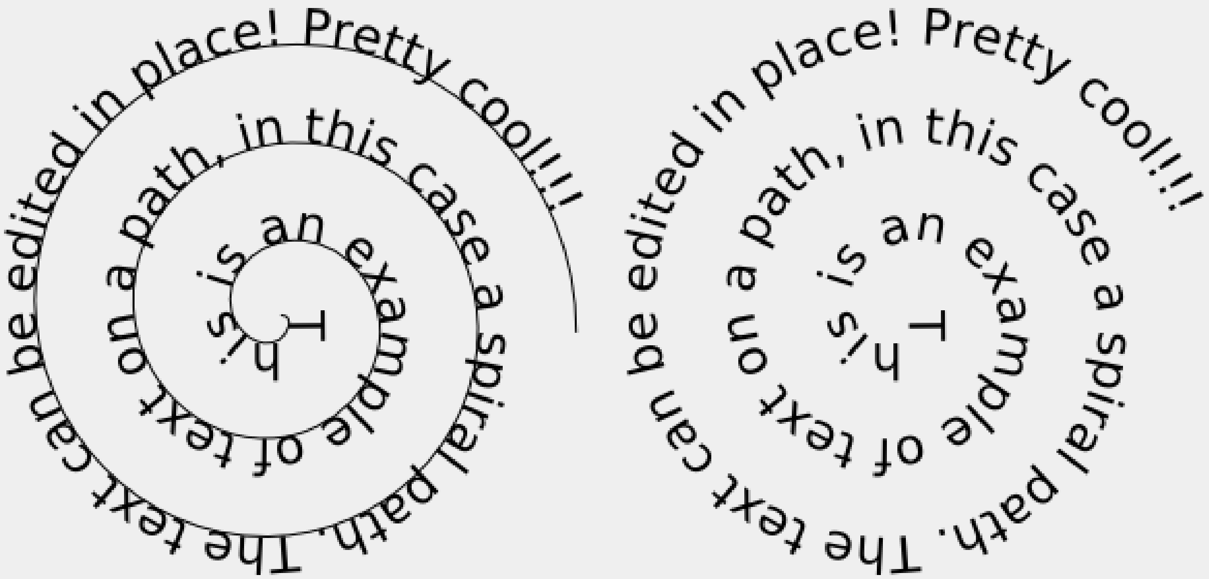
→ Text on a Path

[Inkscape](#) » [Text](#) » Text on a Path

Index

Text on a Path

Text can be put along an arbitrary path.


This is an example of text on a path, in this case a spiral path. The text can be edited in place! Pretty cool!!!
This is an example of text on a path, in this case a spiral path. The text can be edited in place! Pretty cool!!!

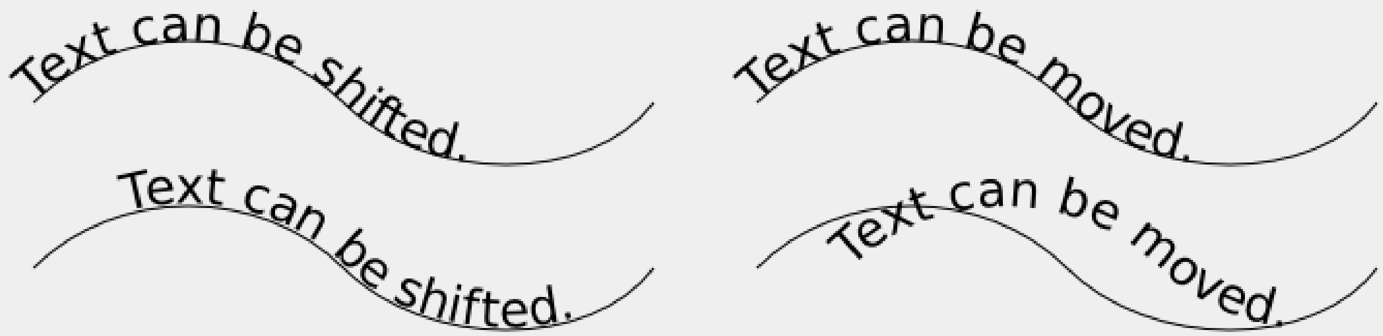
Text on *Spiral* paths. On the right, the path has been hidden by turning off the [*Stroke paint*](#).

Modified in v0.45. (No longer need to convert flowed text to regular text.)

To place text on a path, enter the text as a *Regular* text or *Flowed* text. Draw the desired path. Select both text and path, then use the Text → Put on Path command. The text should now appear along the path. Note that [*Shapes*](#) except for [*Rectangles*](#) are described internally by Inkscape as paths and thus don't require converting to a path.

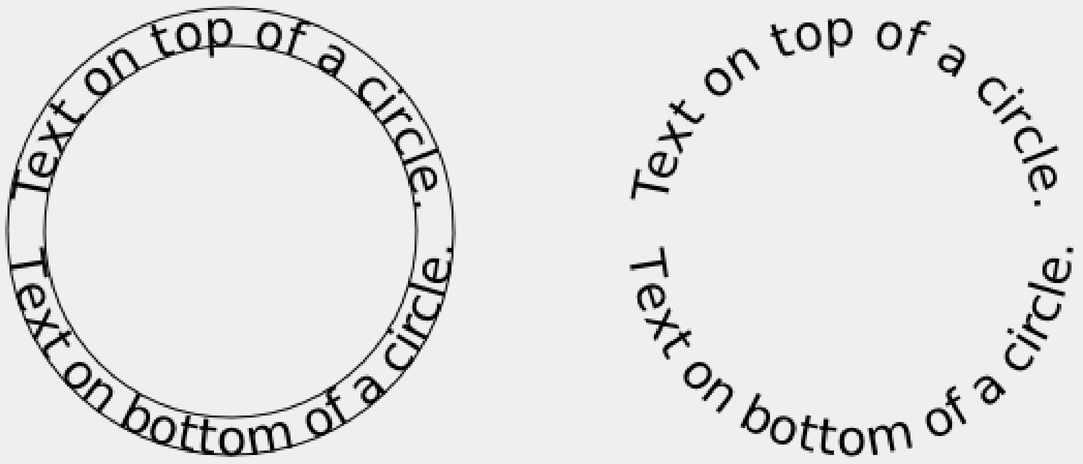
Both the text and the path can be edited in place. The text should adjust to any changes in the path. The path can be made invisible by selecting only the path, then removing the *Stroke paint* with the [*Fill and Stroke*](#) dialog. To select an invisible path for editing, select the text and use Edit → Clone → Select Original (**Shift+D**). To remove text from a path, use Text → Remove from Path.

Parts of the text can be selected and the style, [*kerning*](#), etc. can be adjusted as for *regular* text. Text can also be moved independently of the path by selecting the text only and using the normal means for moving objects.



Text can be adjusted or moved relative to the path. Left: the text has been adjusted and kerned with the **Alt**+**arrow** keys. Placing the cursor at the beginning of the text and using the **Alt**+**arrow** keys will move the starting position of the text. Right: the text has been moved independently of the path by selecting the text only and dragging it to its new position with the mouse.

Text on a path is initially placed on the “left” side of the path (referenced from the path direction) starting at the beginning of the path. One can change the direction of the text (and the side it is placed on) by reversing the direction of the path (e.g., Path → Reverse (**Shift+R**)).



To place text on a circle so that it reads from left to right on both the top and bottom, two circles must be used as seen on the left. The larger circle has been flipped horizontally (Object → Flip Horizontal (**H**)) so the text is placed inside the circle starting from the left. The smaller circle has been rotated 180 degrees. (The path of a circle when first drawn starts from the rightmost point. The circle must be rotated or flipped to move the starting point of the path to the left.) Note that *kerning* may be necessary as the characters of text placed on the outside of a curve will be too far apart while those on the inside of a curve will be too close together.

See the *Generate from Path* effect for an alternative way of putting text on a path.

Kerning, Shifting, and Rotating Characters

Table of Contents

Text in a Shape

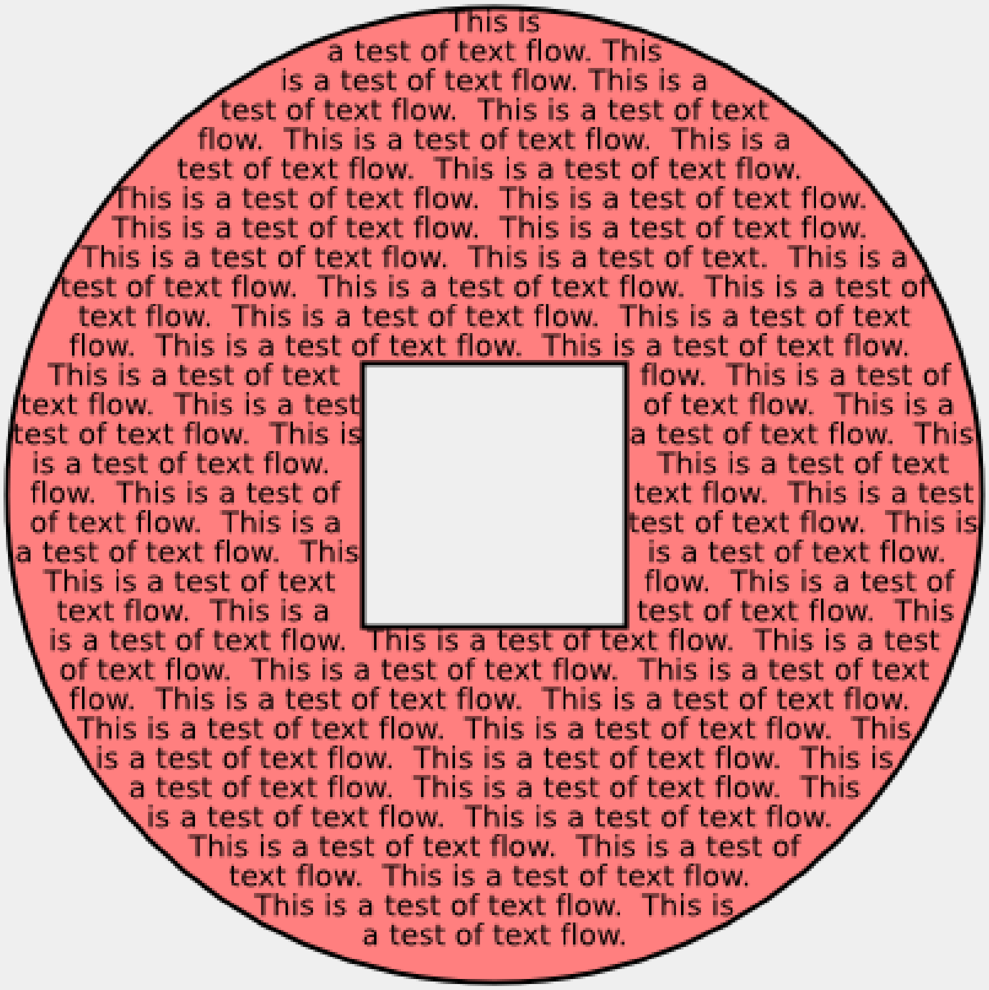
Inkscape » Text » Text in a Shape

Index

Text in a Shape

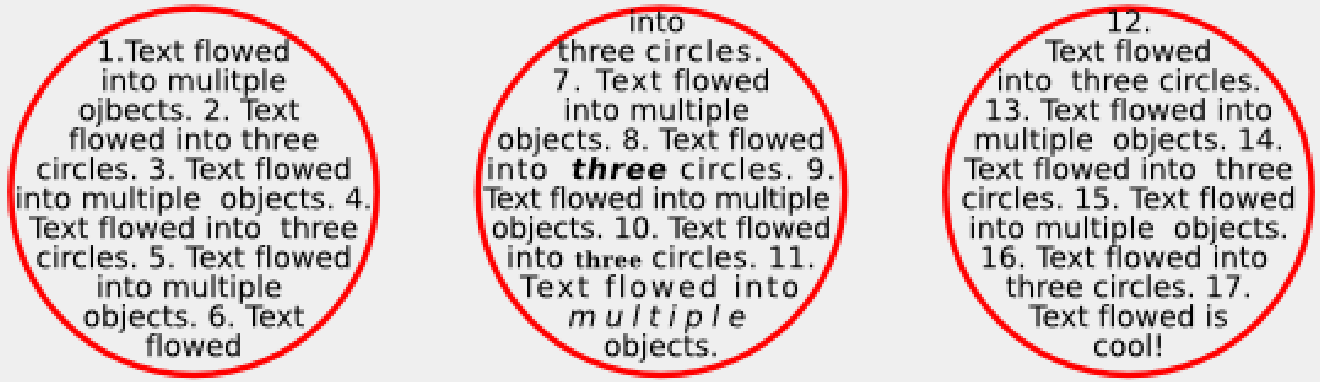
Modified in v0.45. (No longer need to convert flowed text to regular text.)

Text can be flowed inside any arbitrary shape by linking a text object to a shape or path. (This “Link Flowed Text” was a draft *SVG* 1.2 specification and will not work with *SVG* 1.1 viewers.)


This is
a test of text flow. This
is a test of text flow. This is a
test of text flow. This is a test of text
flow. This is a test of text flow. This is a
test of text flow. This is a test of text flow.
This is a test of text flow. This is a test of text flow.
This is a test of text flow. This is a test of text flow.
This is a test of text flow. This is a test of text. This is a
test of text flow. This is a test of text flow. This is a test of
text flow. This is a test of text flow. This is a test of text
flow. This is a test of text flow. This is a test of text flow.
This is a test of text
flow. This is a test of
text flow. This is a test
of text flow. This is a
test of text flow. This is
a test of text flow. This
is a test of text flow.
This is a test of text
flow. This is a test of
text flow. This is a test
of text flow. This is a
test of text flow. This is
a test of text flow. This
is a test of text flow.
This is a test of text
flow. This is a test of
text flow. This is a
test of text flow. This
is a test of text flow. This is a test of text flow. This is a test
of text flow. This is a test of text flow. This is a test of text
flow. This is a test of text flow. This is a test of text flow.
This is a test of text flow. This is a test of text flow. This
is a test of text flow. This is a test of text flow. This is
a test of text flow. This is a test of text flow. This
is a test of text flow. This is a test of text flow.
This is a test of text flow. This is a test of
text flow. This is a test of text flow.
This is a test of text flow. This is
a test of text flow.

Text flowed into a path with the shape of an old Chinese coin. The path consists of both the outer circle and inner square.

To create a linked flowed text object, select a text object and one or more shape/path objects. Then use the Text → Flow into Frame (**Alt+W**) command. If multiple shape/path objects are selected, the text will flow into the *last* object selected first.



1.Text flowed
into mulitple
ojbects. 2. Text
flowed into three
circles. 3. Text flowed
into multiple objects. 4.
Text flowed into three
circles. 5. Text flowed
into multiple
objects. 6. Text
flowed

into
three circles.
7. Text flowed
into multiple
objects. 8. Text flowed
into **three** circles. 9.
Text flowed into multiple
objects. 10. Text flowed
into **three** circles. 11.
Text flowed into
multiple
objects.

12.
Text flowed
into three circles.
13. Text flowed into
multiple objects. 14.
Text flowed into three
circles. 15. Text flowed
into multiple objects.
16. Text flowed into
three circles. 17.
Text flowed is
cool!

Text flowed into three circles. The circles were selected from right to left so that the text would flow first in the left hand circle.

Flowed text can be edited in place, including selecting part of the text and changing the style or changing the letter and line spacings. The center circle above has had some adjustments made to the style and the letter spacing. To select all the text, use Edit → Select All (**Ctrl+A**) command while a text object is selected with the *Text Tool*.

If the flowed text is selected, the Edit → Clone → Select Original (**Shift+D**) command will select the first shape or path object (especially useful if the path has been made invisible). **Tab** will then rotate the selection one by one through any other shape or path objects that are part of the *linked-flowed* text object, as well as the text itself.

Flowed text can be converted back to a *regular* text object with the Text → Unflow (**Shift+Alt+W**). The resulting text will be on a single line.

The Text → Convert to Text command converts *link-flowed* text to a *regular* text object while preserving the appearance of the text. The text is still editable but will no longer reflow inside the shape or path frame. This is necessary for display of the drawing in another *SVG* renderer.

Text on a Path
Table of Contents
Chapter 9. Attributes

Inkscape: Guide to a Vector Drawing Program

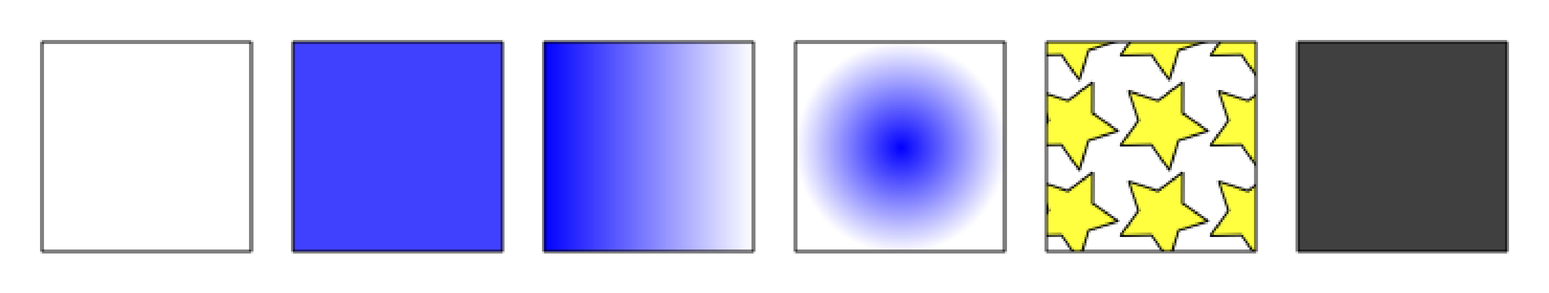
Get the book.

Inkscape » Attributes » Fill and Stroke Paint

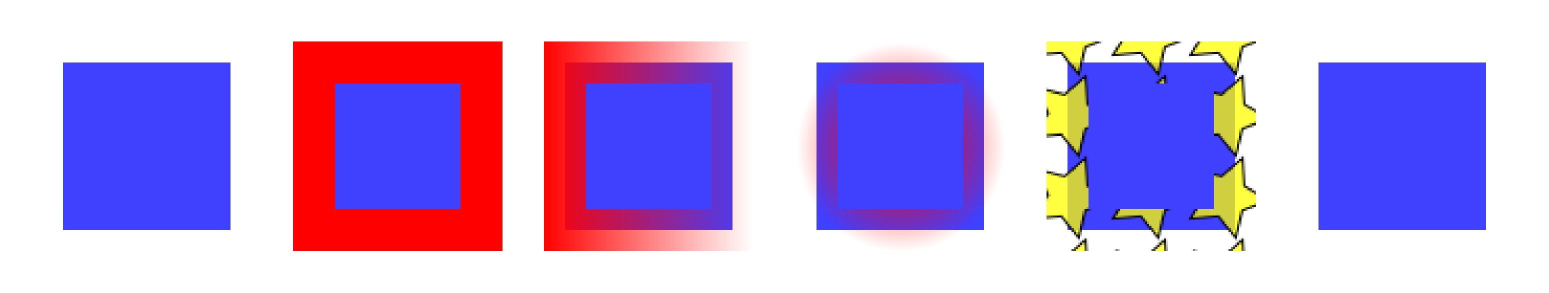
Index

Fill and Stroke Paint

There are a number of different options for the *Fill* and *Stroke paint* of an object. Examples of the different options are shown below. The use of these options for the *Fill* and the *Stroke paint* is basically the same, so we'll use the word *fill* to talk about both at the same time.



Choices for the *Fill* of an object, from left to right: No fill, Flat color, Linear gradient, Radial gradient, Pattern, Unset.



Choices for the *Stroke paint* of an object, from left to right: No fill, Flat color, Linear gradient, Radial gradient, Pattern, Unset. The stroke has been widened to make it easier to see the effect of the different options.

The fill type can be set using the *Fill and Stroke* dialog under the *Fill* and *Stroke paint* tabs. The fill type can be one of the following choices (set by clicking an icon):

- No paint (transparent).
- Flat (solid) color.
- Linear *Gradient* (a smooth transition between two or more colors).
- Radial *Gradient* (a smooth transition between two or more colors in a radial direction).
- *Pattern* (filled with a repeating pattern).
- Unset (necessary for giving different attributes to cloned copies of an object).

A *Gradient* fill type can also be selected by using the *Gradient Tool*.

Each of the options (except the *No paint* and *Unset* options) is discussed below.

Flat (Solid) Colors

Color can be the simplest or the most complicated aspect of a drawing depending on your needs. Color is stored internally in Inkscape as a six-digit *hexadecimal* number consisting of three pairs of digits. Each pair of digits corresponds to the amount of Red, Green, Blue (*RGB*). This matches the *SVG* specification for describing color. For example, a color defined as #FF7F00 has red, blue, and green components of 100%, 50%, and 0%, respectively, of the maximum values.

In some cases, a fourth pair of digits is added to describe *Alpha* (*RGBA*). The A or *Alpha* attribute may not be familiar to many people. This attribute specifies how transparent the fill should be. It can range from 0 for complete *transparency* to 255 (*hexadecimal* FF) for complete *opacity*. The term *opacity* is often used in place of *Alpha*. Its value ranges from 0% (0.0) for a transparent object to 100% (1.0) for an opaque object.

Modified in v0.46 (CMS interface):

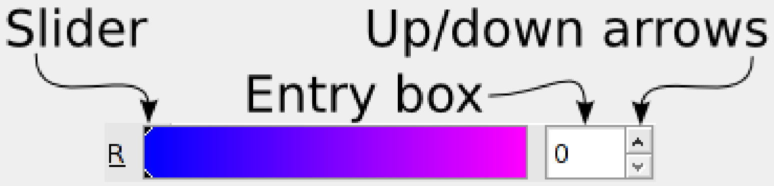
In principle, this is a simple description for specifying any color. The complexity comes from assuring that the color reproduced on a display or in printing matches the color the artist envisioned. Various color “systems” have been developed to facilitate this. Inkscape supports base ICC profile functionality through the use of LittleCMS. Setting up color management can be done under the *Color management* section in the *Inkscape Preferences* dialog. Color managements support is only available under Linux and Macintosh OS X. Profiles internal to a *bitmap* are also not handled. A complete discussion of this topic is beyond the scope of this book.

The fill color of an object can be modified a variety of ways including using the *Fill and Stroke* dialog, the *Palette*, the *Swatches* dialog, and the *Dropper Tool*. Some of these methods can also be used to change the color of a *Gradient* stop when a *Gradient* handle is selected.

Fill and Stroke Dialog - Color

When the use of a flat (or solid) color is specified for the *Fill* and *Stroke paint* of an object, the corresponding tab of the *Fill and Stroke* dialog will show five sub-tabs, each one corresponding to a different method of specifying the color plus one for color management. Each method is described next in its own section.

Except for the *Wheel* tab, each color parameter can be set by either dragging a slider (small triangles), typing the desired value into the entry box, using the up/down arrows in the widget (**Right Mouse**+**Click** on an arrow causes the value to change to the minimum or maximum, **Middle Mouse**+**Click** cause the value to increment or decrement by 10), or the **Up/Down Arrow** keys after the entry box is selected. The slider bar shows the current value (triangles) and what the color will look like as that slider is dragged.

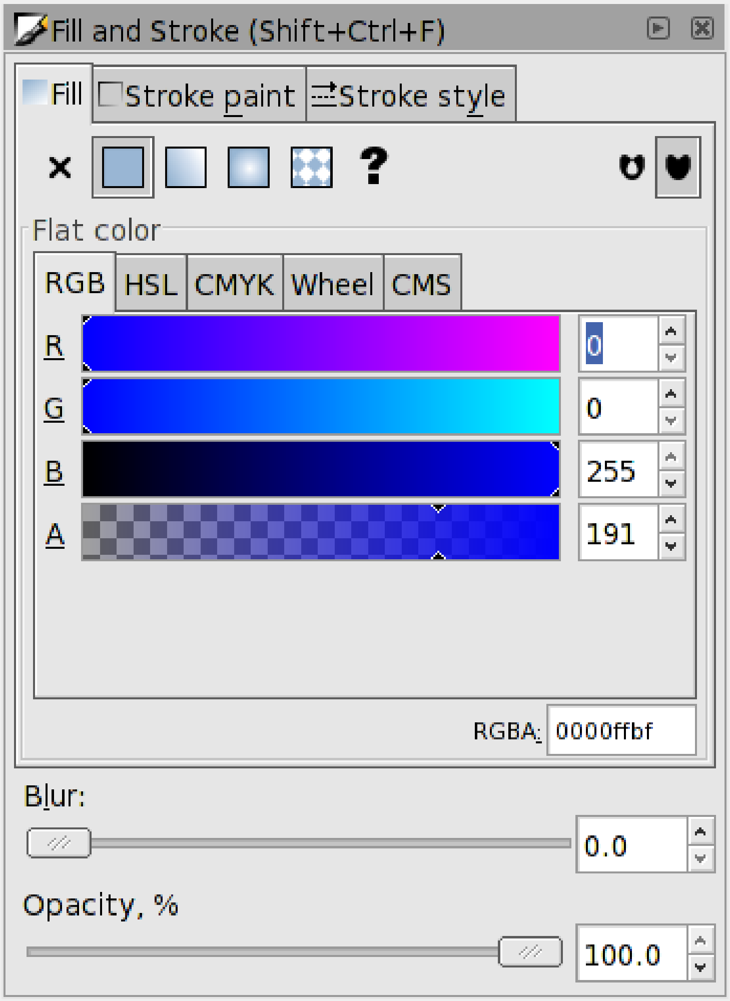


Methods for setting a color parameter.

The A or *Alpha* attribute specifies how transparent the fill should be, 0 for completely transparent and 255 (100) for completely opaque in the case of the *RGB*, *HSL*, and *Wheel* (*CMYK*) methods.

RGB

RGB (Red, Green, Blue) is a method for specifying a color in terms of the three additive primary colors. This is the native method for computer screens. Range of allowed values is from 0 to 255 (0 to FF in *hexadecimal*).

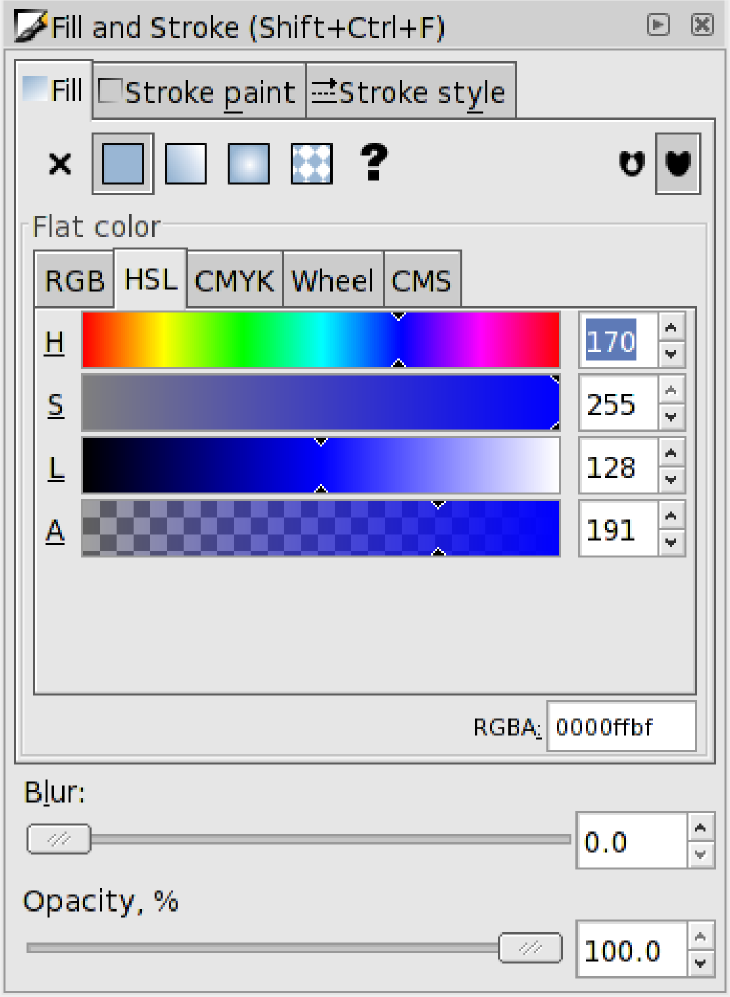


R: 0
G: 0
B: 255
A: 191
RGBA: 0000ffbf
Blur: 0.0
Opacity, %: 100.0

RGB tab for setting *Fill* color.

HSL

HSL (*Hue*, *Saturation*, *Lightness*) is a method for specifying color in terms of hue (color in optical spectrum), saturation (intensity-purity), and lightness. The range for saturation is from a pure color to gray. The range for lightness is from black to pure color to white. Range of allowed values is from 0 to 255 (0 to FF in *hexadecimal*).



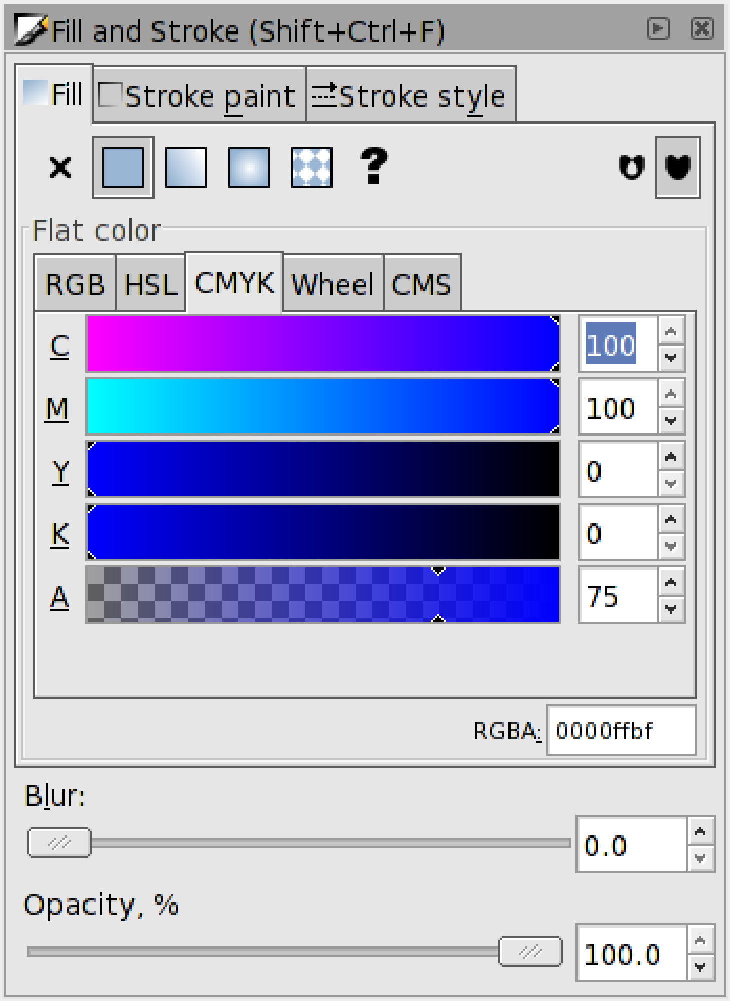
H 170
S 255
L 128
A 191
RGBA: 0000ffbf
Blur: 0.0
Opacity, % 100.0

HSL tab for setting *Fill* color.

CMYK

CMYK (Cyan, Magenta, Yellow, Key [Black]), is a method for specifying color in terms of *subtractive* primary colors and is commonly used in printing. Range of allowed values is from 0 to 100.

Warning
Inkscape stores color internally in the RGB format. This is the only color specification supported by SVG. Furthermore, the entry boxes are set up so that the value in one is always zero. (Any color in *RGB* color space can be defined using only three of the *CMYK* terms. The definition is not unique.) Better support for *CMYK* is planned.



Fill and Stroke (Shift+Ctrl+F)

C 100
M 100
Y 0
K 0
A 75

RGBA: 0000ffbf

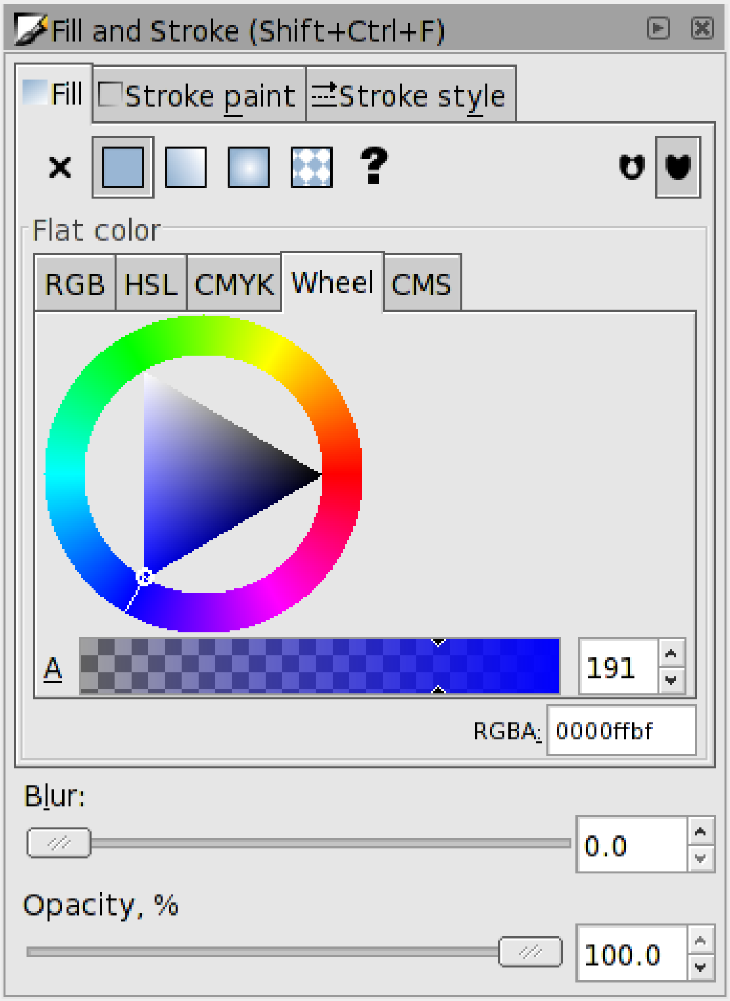
Blur: 0.0

Opacity, % 100.0

CMYK tab for setting *Fill* color.

Wheel

The *Wheel* is an alternative way of changing color in the HSL paradigm. Dragging the line around the circle changes the *Hue*. Dragging the small circle within the triangle parallel to the edge that varies from white to black changes the *Lightness* and dragging perpendicular to that edge changes the *Saturation*.



A 191

RGBA: 0000ffbf

Blur: 0.0

Opacity, % 100.0

Wheel tab for setting *Fill* color.

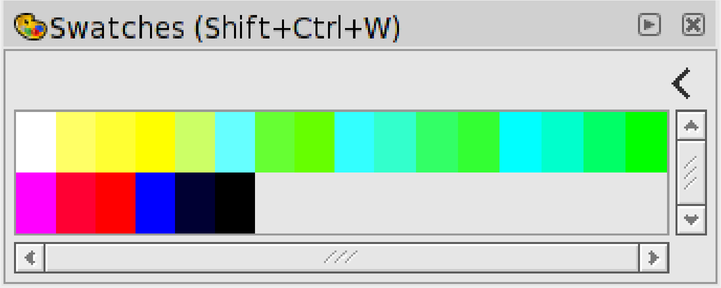
CMS

New in v0.46.

This tab allows editing of colors managed by an *icc profile* if enabled.

Palette and Swatches Dialog

The color *Palette*, located near the bottom of the main Inkscape window, and the *Swatches* dialog (View → Swatches... (**Shift+Ctrl+W**)) allows one to quickly set the color of an object's *Fill* or *Stroke* or to set the color of a *Gradient* *Stop*. They can also be used to set the *Current style*. There use is essentially identical so they will be treated together. The visibility of the *Palette* can be toggled via a check box in the View → Show/Hide submenu.



Swatches dialog.

The following methods are available for both the *Palette* and the *Swatches* dialog:

- **Left Mouse Click** on a swatch to change the *Fill* of selected objects or the color of selected *GradientStops* to the color of the swatch. **Shift**+**Left Mouse Click** on a swatch to change the *Stroke* of selected objects. The *Current style* will also change (no object need be selected).
- **Left Mouse Drag** from a swatch to an object's *Fill* or *Stroke* or to a *GradientStop* to change the corresponding attribute to the swatch's color. **Shift**+**Left Mouse Drag** to anywhere on an object to set the *Stroke* color (except to a *GradientStop*). The target object need not be selected. The *Current style* will not change.
- **Left Mouse Drag** from a swatch to the *Fill* or *Stroke* part of the *Style Indicator* to change the *Fill* or *Stroke* of selected objects. The *Current style* will also change.
- **Right Mouse Click** on a swatch to open a small dialog which allows you to assign the color to the *Fill* or *Stroke* of selected objects. The *Current style* will also change.

You can also drag colors to or from other applications that support *Drag and Drop*.

Inkscape has a variety of built-in palettes (some copied from Gimp). More palettes can be added by installing palette files in the directory `share/palettes`. The files use the Gimp palette file structure where colors are defined in terms of a triplet of numbers in a "*RGB*" format. See the section called “Custom Swatches or Palettes” in Chapter 20, *Customization*, for details.

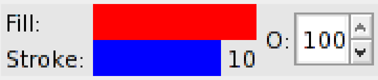
Both the *Palette* and the *Swatches* dialog have a pull-down menu (far right) where you can set the size and shape of the swatches, if the colors should be displayed in one row or in multiple rows, and which palette should be used. Hovering the pointer over a swatch will display a color's name in a *tool tip*. A scroll bar gives access to colors in a palette that are not displayed when there are too many colors to fit.

Style Indicator

The *Style Indicator* located on the left side of the *Status Bar* displays information on selected objects, text fragments, or *Gradient* stops. The indicator includes a number of methods to alter style including: pop-up menus, targets for Drag and Drop colors, and Color Gestures.

Display

The *Style Indicator* has three parts showing *Fill*, *Stroke paint*, and *opacity* (O), which show attributes for selected objects or text fragments. The *Fill* and *Stroke paint* parts are referred to as the *fill indicators*.



The *Style Indicator* showing the attributes of an object with a red *Fill*, a blue *Stroke paint*, a *Stroke* width of ten pixels, and a Master *opacity* of 100%.

A displayed *fill attribute* can be one of:

- Color swatch: Shows color with (left) and without (right) *Alpha* (*Alpha* refers to the *Fill* and *Stroke paint* attributes and not the object's Master *opacity*).
- N/A: Not Applicable (i.e., no object selected).
- None: No fill defined.
- Unset: fill is unset.
- L Gradient: fill is a linear *Gradient*.

- R Gradient: fill is a radial *Gradient*.
- Pattern: fill is a pattern.
- Different: More than one object selected with different fill.

When multiple objects are selected and all of the selected objects have a color fill, then one of the following letters will be shown:

- m: Selected objects have same fill color.
- a: Selected objects have different fill colors. The color displayed is an average of the colors in the selected objects.

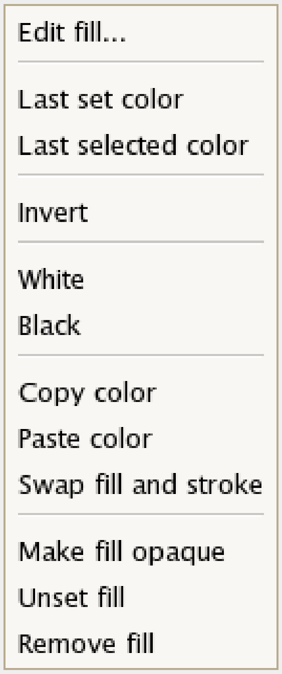
For *Gradient* handles, both parts (*Fill* and *Stroke paint*) show the handle color.

The *Style Indicator* has a number of features that depend on the part.

- Fill/Stroke Paint indicators:
  - A **Left Mouse Click** opens the *Fill and Stroke* dialog with the corresponding tab selected.
  - A **Middle Mouse Click** on a bar removes the fill from the selected objects if a fill is defined. If no fill is defined, it sets the fill to black.
  - A **Right Mouse Click** on a bar opens a pop-up menu as discussed below.
  - A color from the *Palette* or *Swatches* dialog can be dragged and dropped onto one of the *fill indicators* to change the fill of all selected objects.
- Stroke indicator:
  - A **Left Mouse Click** opens the *Fill and Stroke* dialog to the *Stroke style* tab.
  - A **Right Mouse Click** opens the a pop-up menu that allows the stroke width unit to be changed as well as a preset width to be selected. The stroke can also be removed with this menu.
- Opacity indicator:
  - A **Right Mouse Click** on the numeric field opens up a pop-up menu with preset opacity values.
  - A **Middle Mouse Click** on the "O:" label cycles through the opacity values 0%, 50%, and 100% (0.0, 0.5, and 1.0).

Fill Indicator Pop-up Menu

A **Right Mouse Click** on either the *Fill* or *Stroke paint* bar opens a pop-up menu with entries that act on the *Fill* or *Stroke paint* of the selected objects or text fragments, depending on which bar was clicked. If a *Gradient* handle is selected, the menu entries apply to that handle.



The *Style Indicator* pop-up menu.

- Edit fill... (Edit stroke...): Opens *Fill and Stroke* dialog. (The dialog can also be opened directly by a **Left Mouse Click** on the *Fill* or *Stroke paint* part of the *Style Indicator*.)
- Last set color: Applies the last *set* color to the *Fill* or *Stroke paint* of selected objects. A color is set when a color is applied to any object or when a color is selected from the *Palette*.
- Last selected color: Applies the last *selected* color to selected objects. The last selected color is the color of the previously selected object(s) prior to selecting the object(s) whose color is to be changed. If the color is to be applied to multiple objects, they must be selected together using a *rubber-band* selection.
- Invert: Inverts the color of the *Fill* or *Stroke paint* of selected objects. If more than one object is selected, the colors of those objects are averaged before the color is inverted. The *opacity* is not changed.
- White: Set *Fill* or *Stroke paint* to white.
- Black: Set *Fill* or *Stroke paint* to black.
- Copy color: Copies color of selected objects to the clipboard in *hexadecimal* format. If more than one object is selected, the colors of those objects are averaged.
- Paste color: Pastes color to selected objects from the clipboard.
- Swap fill and stroke: Exchanges *Fill* and *Stroke paint* colors.
- Make fill (stroke) opaque: Sets *Fill* or *Stroke paint* to full *opacity*. (Does not affect *Master opacity*.)
- Unset fill (stroke): Unsets the *Fill* or *Stroke paint* of selected objects.
- Remove fill (stroke): Removes the *Fill* or *Stroke paint* of selected objects.

Color Gestures

New in v0.46.

Color Gestures is the name give to changing the color of a *Fill*, *Stroke*, or *Gradient Stop* by dragging the mouse from a fill indicator into the Inkscape window. The principle is that as you drag the mouse, the color will change proportionally to the distance from a 45° line from the indicator. The farther away you are, the more subtle the changes can be. Changes are made in the *HSL* color space.

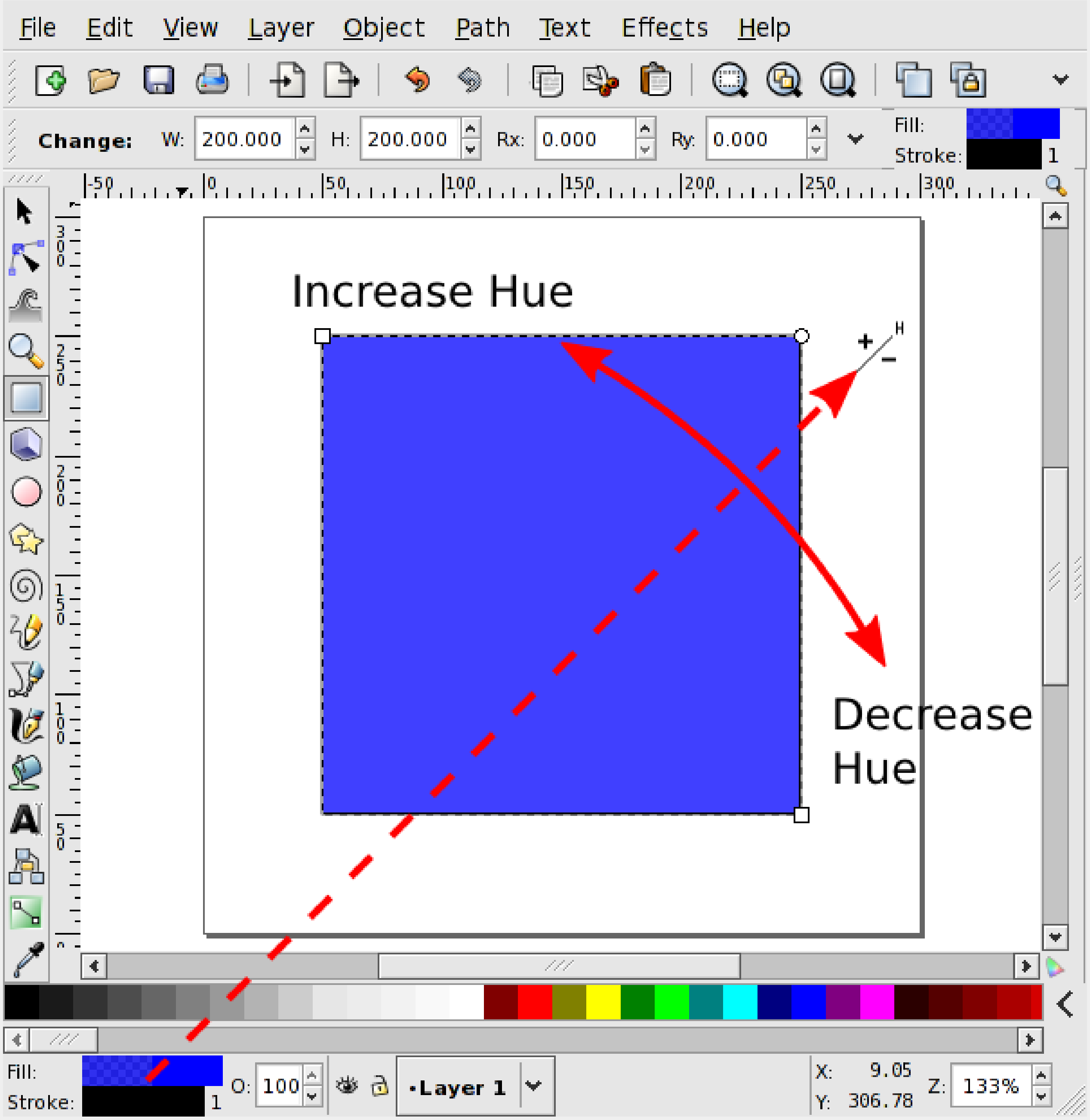


Diagram for *Color Gestures*. Dragging the mouse away from the *Fill* indicator to the left or up from the dash line increases hue (blue to red) while dragging to the right or down decreases hue (blue to green). Dragging straight up or straight across results in the maximal change possible (blue to yellow). Note the small H on the cursor. This indicates that hue is being altered.

With out any modifier key, changes are made to hue. With the **Shift** changes are made to saturation while with the **Ctrl** changes are made to lightness. Note that the letter next to the cursor will change to indicate the mode. Key modifiers can be changed while dragging. When a key modifier is changed, the “zero” line (normally at 45°) changes to pass through the current cursor position. This is to avoid abrupt changes in color when changing modifiers. The **Alt** modifier disables changes to the color so that the cursor can be repositioned.

If more than one object or *Gradient Stop* is selected, the starting color will be the average color of the selected items and the final color will be the same. If you wish to shift the color in the same way for a number of objects but preserve the relative differences use the *Tweak Tool*.

Color gestures are very useful once you get the hang of them. It is well worth spending a little time to play with them!

Dropper Tool

The *Fill* and *Stroke paint* color of an object can be changed by using the *Dropper Tool* to grab an existing color in the drawing. Options allow for grabbing the average color over a circular region, inverting the grabbed color, and saving the grabbed color to the system *clipboard* (as a *RGBA* hexadecimal number).

To use the *Dropper Tool*, first select the object that you want to modify with a tool other than the *Dropper Tool*. Recall that you can switch temporarily to the *Select Tool* by using the **Space Bar**.

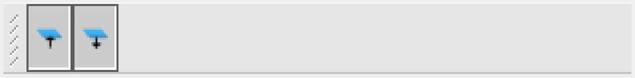
Then select the *Dropper Tool* by clicking on the icon (**F7** or **D**) in the *Tool Controls*. Finally click with the *Dropper Tool* on the desired color. The shortcut **D** will toggle between the *Dropper Tool* and any other tool (as of v0.46).

The **Shift** causes the chosen color to be applied to the object's *Stroke paint* rather than the *Fill*. The **Alt** causes the inverse color to be applied. The **Shift** and **Alt** keys can be used in combination. However, neither of the modifier keys are useful when copying a color to the *clipboard*.

- **Left Mouse Click**: Pick *Fill* color.
- **Left Mouse Drag**: Pick average *Fill* color (color is averaged over circle created during drag).
- **Ctrl**+**C**: Copy color under cursor to system *clipboard* in the form of an 8-digit hexadecimal number (two digits for each of *RGBA*).

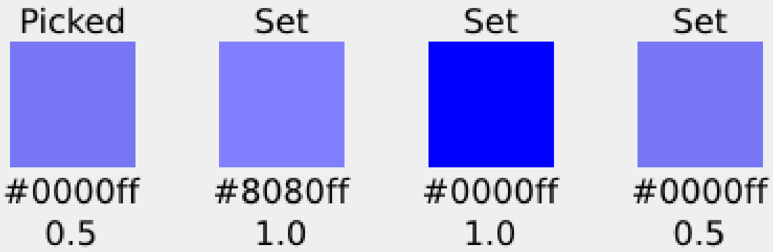
Changed in v0.45 and again in v0.46:

The *Dropper Tool* *Tool Controls* has two buttons (check boxes in v0.45) that determine if the *Alpha* of a color should be *picked* and/or *set*. These settings affect the way a color is picked if the “picked” object has an *opacity* different from 1.0 (or 100%).



The *Dropper Tool-Tool Controls*: : Pick alpha. : Set alpha.

- **Pick alpha disabled:** The color picked is as shown on the screen. For example, picking the color from an object with a dark blue fill but an opacity of 0.5 would result in a light blue color with an opacity of 1.0. Opacity of set object not changed.
- **Pick alpha enabled, Set alpha disabled:** The color picked is the color that the object would have if its opacity was 1.0. A dark blue object with an opacity of 0.5 would result in a dark blue color (an opacity of 1.0). Opacity of set object not changed.
- **Pick alpha enabled, Set alpha enabled:** The color and opacity are both copied from the picked object. A dark blue object with an opacity of 0.5 would result in a light blue color composed of a dark blue fill with an opacity of 0.5. Opacity of set object changed. Note: This is only applicable if the color is picked from an object with *transparency* that is not over another object.



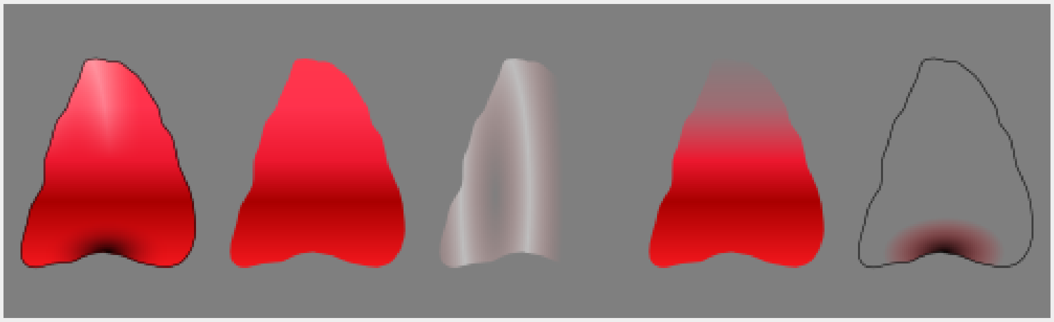
Color squares shown with the numerical values of their color (*hexadecimal*) and opacity. From left to right: Picked color, set color with *Pick alpha* box not checked, set color with *Pick alpha* box checked but *Set alpha* box not set, set color with both *Pick alpha* and *Set alpha* boxes checked. In all cases, the original opacity of the “Set” squares was 1.0.

Note: v0.45 has a bug. If the color is not picked correctly with a **Left Mouse Click**, use a small **Left Mouse Drag**.

Gradients

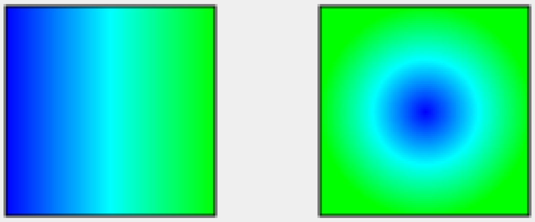
Onscreen creation and editing of intermediate Stops added in v0.46.

Gradients are a smooth blending from one color to another. *Gradients* can be used to build up complex shading of an object as shown in the flower petal below. Note: As of v0.45, Inkscape supports the *Gaussian Blur* filter, which may be an easier way to produce complex shadings.



A flower petal consisting of four layers. From left to right: The finished petal. The base layer. A highlight. A duplicate of the base layer with partial transparency to mask part of the highlight layer. Top shadow layer. The background is gray to show the transparency in some layers.

In Inkscape, *Gradients* can be *linear* or *radial* and can consist of transitions between two or more well-defined colors referred to as *Stops*.



An example of a linear (left) and a radial *Gradient* (right), both with three defined colors or *Stops*.

Inkscape does not support non-linear *Gradients*. Non-linear *Gradients* can be simulated by adding extra *Stops*.

There are three parts to using a *Gradient*; each treated in the next three sections:

1. Attach a *Gradient* to an object.
2. Edit the *Stops*.
3. Adjust the orientation and extent of the *Gradient*.

The use of *linear* and *radial Gradients* is essentially the same and both will be treated together.

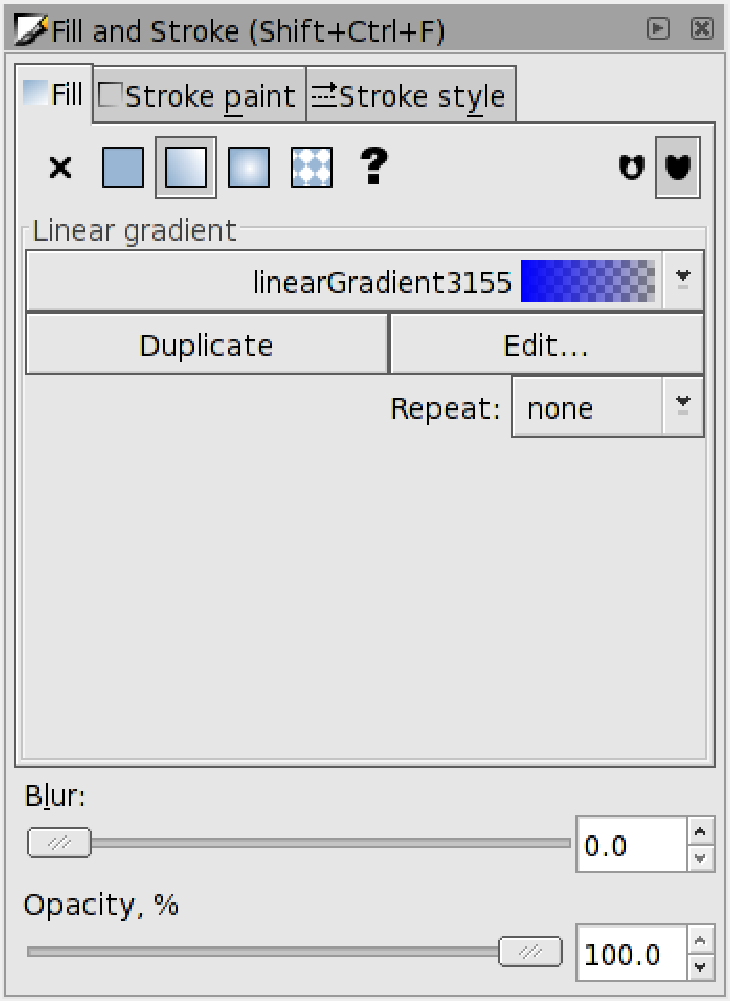
Note

If you want a *Gradient* to transform with an object, you must toggle on this option using the ⧉ icon that is in the *Tool Controls* when the *Select Tool* is in use.

Attaching Gradients to Objects

Gradients can be attached to an object either with the *Fill and Stroke* dialog or through the use of the *Gradient Tool*.

To attach a *Gradient* with the *Fill and Stroke* dialog, simply select an object and click on either the linear □ or radial *Gradient* □ icons in the dialog. A *Gradient* with two *Stops* will automatically by created and applied to the object. The *Stops* will have the color of the previous *Fill* with one *Stop* having full *opacity* and the other full *transparency*. The following figure shows the dialog after attaching a *Gradient* this way.



Linear gradient

linearGradient3155

Repeat: none

Blur: 0.0

Opacity, % 100.0

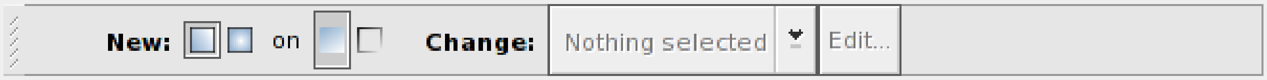
Gradient selected in *Fill and Stroke* dialog.

An already defined *Gradient* can be assigned to the object by selecting the *Gradient* from the drop-down menu under the *Fill* tab of the *Fill and Stroke* dialog. A *Gradient* can also be assigned to the *Stroke* of an object under the *Stroke paint* tab.

To attach a *Gradient* with the *Gradient Tool*, select the tool by clicking on the ⧉ icon (**Ctrl**+**F1** or **g**) in the *Tool Box*.

The *Gradient Tool* *Tool Controls* has options to choose a linear □ or a radial □ *Gradient* and the application of the

Gradient to the *Fill* ▨ or *Stroke* □ of an object. Once the options are selected, **Left Mouse Drag** across an object to attach a *Gradient*. The start and stop point of the drag will define the range of the *Gradient* (where the start and end *Stops* are placed, see below). If an already defined *Gradient* has been chosen from the drop-down menu in the *Tool Controls* it will be applied to the object. Otherwise a two *Stop Gradient* will automatically be created with both *Stops* the color of the objects existing *Fill* and with one *Stop* full *opacity* and the other with full *transparency*.



The *Gradient Tool*-*Tool Controls*.

Editing Stops

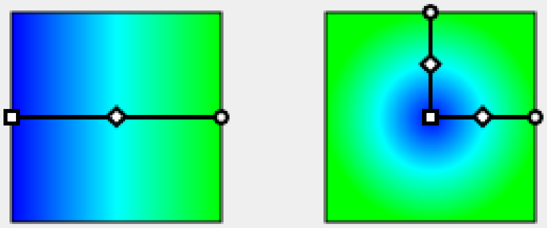
Gradients can be modified by adding, deleting, moving, or changing the color and *opacity* of *Stops*.

As of Inkscape v0.46, *Gradients* can be edited onscreen. This is much more convenient than using the *Gradient Editor* dialog.

Onscreen Editing

An object with a *Gradient* displays *Gradient* handles when the *Gradient Tool*, *Node Tool*, or one of the *shape tools* is active (the latter two if enabled in the *Inkscape Preferences* dialog). Some editing actions work when any of these tools is active, others work only with the *Gradient Tool*.

Circle, diamond, and square handles represent start, intermediate, and end *Stops* respectively. Editing *Stops* has many parallels to editing nodes.



Linear and radial *Gradients* showing the *Gradient* handles.

Stops can be selected by clicking on them with the *Gradient Tool*, the *Node Tool*, or one of the *shape tools*. To select more than one use **Shift**+**Left Mouse Click**. With the *Gradient Tool* you can use a *rubber-band* selection by using **Shift** +**Left Mouse Drag** or you can select all the *Stops* by using **Ctrl**+**A**.

To add a new intermediate *Stop*, with the object selected and the *Gradient Tool* active:

- Double click on the *Gradient* path. A new *Stop* will be added where you clicked. The *Stop* will take the existing color under the path. Note that at least one *Stop* must already be selected.
- **Ctrl**+**Alt**+**Left Mouse Click** on the *Gradient*. A new *Stop* will be added where you clicked. The *Stop* will take the existing color under the cursor.
- Select two adjacent *Stops* and press **Insert**. A new *Stop* will be added halfway between the selected *Stops* and with a color halfway in-between.
- Drag a color from the *Palette* or the *Swatches* dialog unto the *Gradient* path. A new *Stop* will be created with the

dragged color at the point of the drop (drop too far from the path and the *Fill* will be changed to solid with the dragged color). This also works if the *Node Tool* or one of the *shape tools* is active.

To remove an intermediate *Stop*, with the *Gradient Tool* active:

- Click on it using **Ctrl**+**Alt**+**Left Mouse Click**. This also works if the *Node Tool* or any of the *shape tools* are active.
- Use the **Del** to remove selected *Stops*. If there is at least one intermediate *Stop* then deleting the start or end *Stops* will shorten the *Gradient* with the nearest intermediate *Stop* becoming a start or end *Stop*. If there is no intermediate *Stop* then deleting a start or end *Stop* will replace the *Gradient* with a solid *Fill* of the color of the remaining *Stop*.
- Using **Ctrl**+**L** will attempt to simplify the *Gradient* over the a region defined by selected *Stops* by adjusting and removing some *Stops*. This is particularly useful for removing redundant *Stops*.

To move an intermediate *Stop*:

- Drag it with the *Gradient Tool*, *Node Tool*, or one of the *shape tools*. If more than one *Stop* is selected they will all move together. Dragging with the **Ctrl** snaps the *Stops* at points that are multiples of 1/10th of the distance between the nearest neighboring unselected *Stops*. Dragging with the **Alt** moves selected *Stops* according to how far away they are from the dragged *Stop*.
- With one or more *Stops* selected and the *Gradient Tool* active, use the **Arrow** keys. If multiple intermediate *Stops* are selected, they will move together. Using the **Shift** with the **Arrow** keys accelerates the shift by a factor of 10. Using the **Alt** moves the selected *Stops* one screen pixel at a time. Using **Shift**+**Alt** moves the *Stops* ten screen pixels at a time.

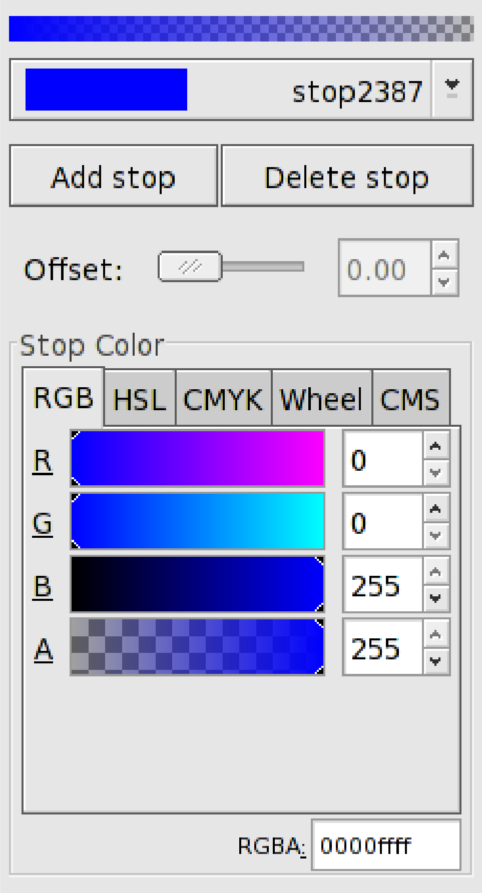
Note: You cannot move a *Stop* past an adjacent *Stop*.

There are several ways to see and change the style (color and/or *transparency*) of one or more *Stops*. In general, if no *Stop* is selected, indicators and changes apply to the whole object; if one *Stop* is selected, indicators and changes apply to that *Stop*; and if multiple *Stops* are selected, indicators show an average value for the selected *Stops* and changes apply to all selected *Stops*.

- The *Fill and Stroke* dialog: If no *Stop* is selected the *Gradient* is previewed at the top under the *Fill* or *Stroke* tab. Otherwise the current color and *opacity* values for selected *Stops* are shown. Changes apply to all selected *Stops*.
- The *Style Indicator*: If no *Stop* is selected the indicators show previews of the *Gradients* (*Fill* and *Stroke*). Otherwise the current values for selected *Stops* are shown. Changes apply to all selected *Stops*. See the section called “Style Indicator”, earlier in this chapter for more details.
- Drag-and-Drop: Colors can be dragged from either the *Palette* or from the *Swatches* dialog onto *Stops* (or onto the *Gradient* path to add a new *Stop*).
- Copy-and-Paste: Colors can be copied to and from the *clipboard*. Copying to the *clipboard* (Edit → Copy (**Ctrl+C**)) will copy a *Stop* color and *opacity* if one *Stop* is selected or the average color if more than one is selected. Pasting the style (Edit → Paste Style (**Shift+Ctrl+V**)) copies from the *clipboard* the color and *opacity* to all selected *Stops*.

Using the Gradient Editor Dialog

There is a dedicated *Gradient Editor* dialog for editing *Gradients*. It is envisioned that this dialog be removed as redundant in the future. To call up the dialog, click on the *Edit...* button, either in the *Fill and Stroke* dialog or the *Gradient Tool*-*Tool Controls*. Note that which *Gradient* is being edited does not change automatically when you select an object with a different *Gradient*.



stop2387

Offset: 0.00

Stop Color

R 0

G 0

B 255

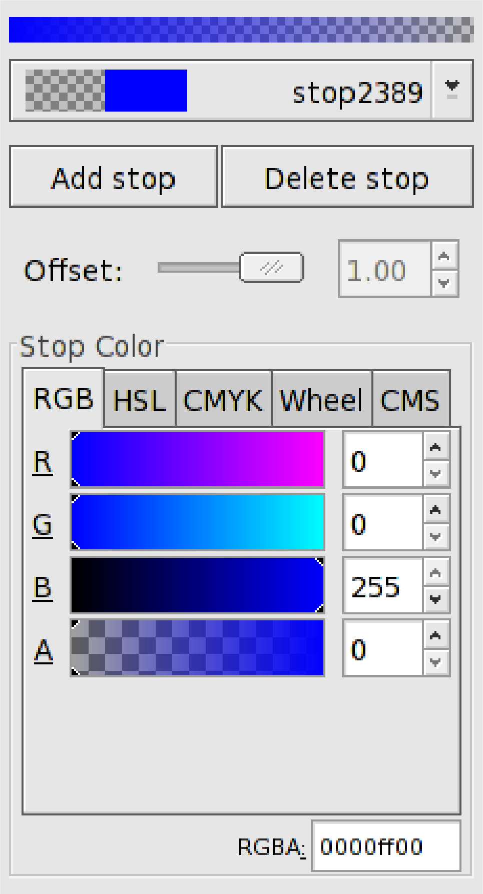
A 255

RGBA: 0000ffff

Gradient Editor dialog showing first default *Stop*.

In this dialog, the *Gradient* is shown at the very top. Next down is the current *Stop*. You can see that the *Stop* shown above (stop2387) is a solid blue, fully opaque with *Alpha* = 255. You can also see from the *Offset* slider that the *Stop* shown is set all the way to the left (and as the *Stop* is at one limit of the *Gradient*, its position can't be moved). The color and transparency of this *Stop* can be changed in the *Stop Color* section. The tabs work the same as those described in the *Flat Colors* section above. The *Dropper Tool* can also be used to change one of the end *Gradient* colors by first selecting the corresponding *Gradient* handle. (To modify a non-end *Gradient* color, copy the color to the *clipboard* by using **Ctrl**+**C** with the desired color selected with the *Dropper Tool* and then paste into the *Gradient Editor* *RGBA* entry box.)

To edit another *Stop*, select that *Stop* in the pull-down menu. In the default case described above, the second *Stop* (stop2389) has the same color as the first but is fully transparent (e.g., *Alpha* = 0). This is shown in the current *Stop* section by the divided box. The left half shows the color with *Alpha* and the right half without *Alpha*. Note also that the *Offset* slider is fully to the right.

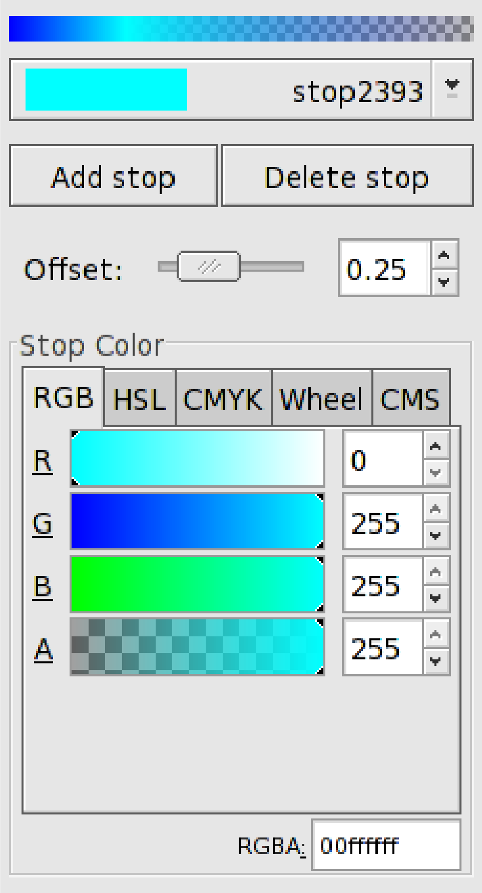


R: 0
G: 0
B: 255
A: 0
RGBA: 0000ff00

Gradient Editor dialog showing second default *Stop*.

To add a *Stop*, click on the *Add stop* button. The new *Stop* will be added to the right of whichever *Stop* was selected, unless that *Stop* was the farthest to the right, in which case the new *Stop* will be added to the left. The position, color, and transparency for the new *Stop* will be set to halfway between its neighbors.

The following figures show a third *Stop* added to a *Gradient* after its color and position have been adjusted.



stop2393

Offset: 0.25

Stop Color

R 0

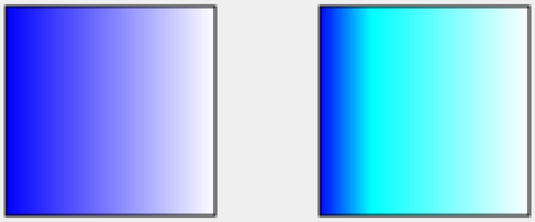
G 255

B 255

A 255

RGBA: 00ffffff

Gradient Editor dialog showing added third *Stop*. Note that the new *Stop* has an offset of 0.25, that is the center of the *Stop* will be one quarter of the *Gradient* width from the left.

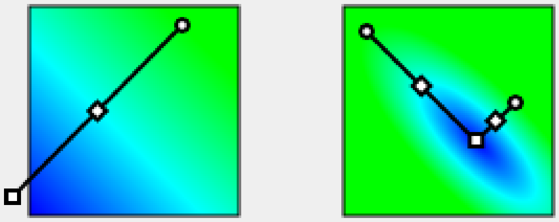


An example of a *Gradient*, before and after adding a third *Stop*.

To delete a *Stop*, select the *Stop*, then click on the *Delete stop* button. As at least two *Stops* are required to define a *Gradient*, you cannot delete a *Stop* if only two are defined.

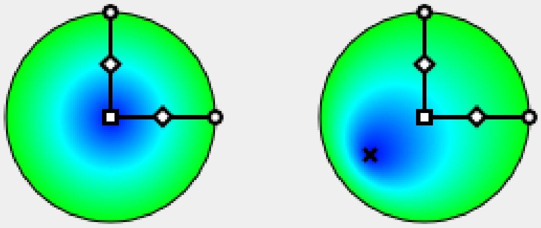
Adjusting Gradients

Once a *Gradient* has been applied to an object, the orientation and extent of the *Gradient* can be changed via dragging the outer two *Gradient Stops* indicated by the square and circle handles. The handles appear when the *Gradient Tool* is selected. They will also appear by default when many of the other tools are selected (controlled by the *Enable Gradient editing* option in the *Inkscape Preferences* dialog under each tool tab). For linear *Gradients*, one set of handles define the range of the *Gradient*. The *Gradient* is parallel to the line connecting the two handles. For radial *Gradients*, there are two sets of handles (or *Stops*) at right angles to each other, sharing the square center handle. The center, handle controls the origin of the *Gradient* (one “edge”), while the two circular handles control the range of the *Gradient* in orthogonal directions. This allows a radial *Gradient* to have an elliptical shape.



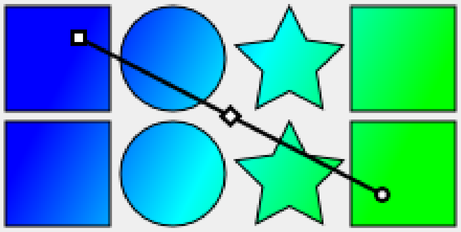
Linear and radial *Gradients* showing the *Gradient* handles after adjusting the orientation and range of the *Gradient*.

A radial *Gradient* can be made asymmetric by dragging the center handle (diamond) while holding down the **Shift** key. A cross will appear where the center of the *Gradient* is located. The cross can be dragged to make further adjustments.



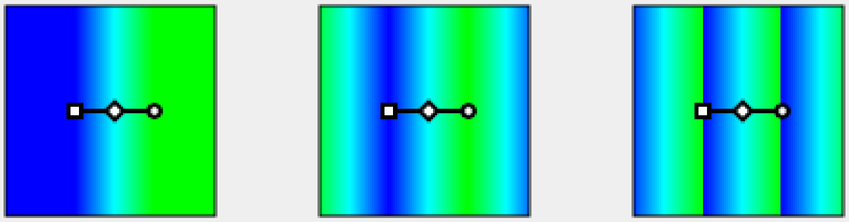
A radial *Gradient* with a symmetric fill (left) and an asymmetric fill (right).

Gradient handles from two different objects will snap together if one is placed over the other. This facilitates aligning *Gradients* between different objects. The handles will then move together. If multiple objects are selected when a *Gradient* is created, all the objects will share a common *Gradient*.

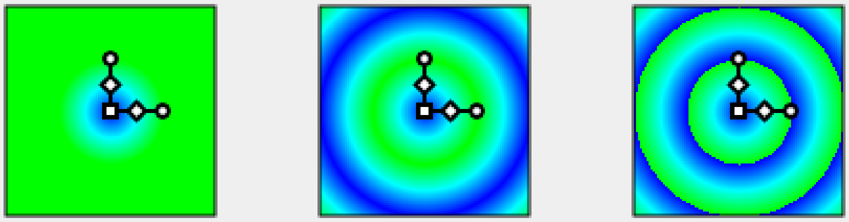


The eight objects share a common *Gradient*.

An option only accessible through the *Gradient* tabs of the *Fill and Stroke* dialog is defining how the area outside the *Gradient* range is filled. The three options are: fill with the solid color of the edge *Stops* (None), fill with a reflection of the same *Gradient* (Reflect), and fill with a translation of the same *Gradient* (Direct). The effect of these three options is shown next.



An example of a linear *Gradients* with different repeat options. From left to right: None, Reflected, and Direct.



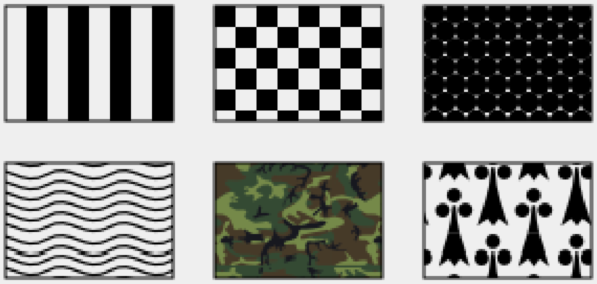
An example of a radial *Gradient* with different repeat options. From left to right: None, Reflected, and Direct.

The keyboard shortcut **Shift**+**R** reverses the *Gradient* direction when the *Gradient Tool* is active. This is especially useful for radial *Gradients* where one cannot just drag the *Gradient* handles to reverse the *Gradient*.

Patterns

Updated for v0.46.

Any object or set of objects can be turned into a *Pattern* and used in the fill of an object. The *Pattern* can be shifted, rotated, and stretched as necessary. Inkscape, as of v04.6, includes a set of *Patterns* accessible through the *Fill and Stroke* dialog. The *Vine Design* tutorial covers creating and using *Patterns*.



A few of the *Patterns* included in Inkscape.

Warning

Check the licenses of the bitmap patterns included with Inkscape to see if they are compatible with your artwork. They are not public domain! (Look inside the *SVG* file or use the *XML Editor* to look in the “defs” section.)

The *Patterns* included with Inkscape are defined in the file `patterns/patterns.svg` located in the Inkscape “share” directory. Edit or replace this file to include your own stock *Patterns*.

To use a *Pattern*, two or three steps are necessary. The first, optional step, is to create a *Pattern*. The second is to apply the newly created *Pattern* or an Inkscape provided *Pattern* to an object. And the third is to adjust the *Pattern* as necessary.

Creating Patterns

Patterns are very easy to create. Simply select the object or objects you wish to use as a *Pattern* and then use the Object ➝ Pattern ➝ Objects to Pattern (**Alt+I**) command.

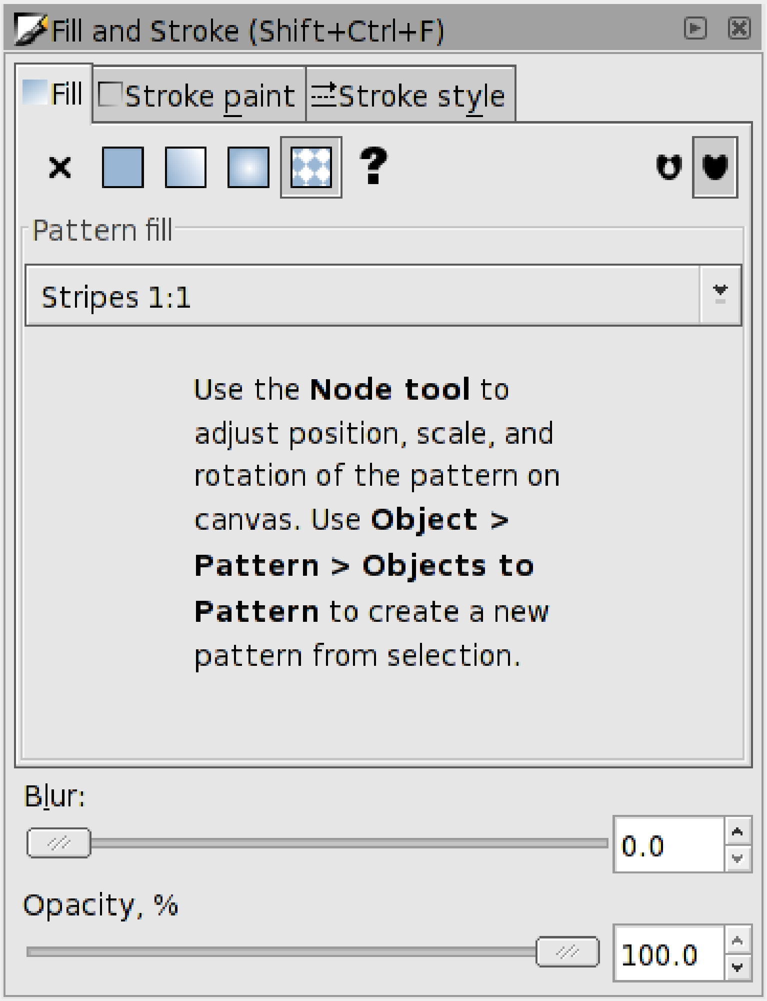
After converting the selection to a *Pattern*, the original selection is replaced by a *Rectangle* shape object filled with the new *Pattern* (and with an invisible stroke). This new object can be deleted but the *Pattern* will remain. *Patterns* have a life of their own.

The “tile” size of the *Pattern* is the total *bounding box* of the objects in the *Pattern*. If space is required around the objects, a non-visible rectangle object can be added or the *Pattern* size can be edited with the *XML Editor* dialog.

The object(s) in any *Pattern* can be edited by selecting an object that is filled with that *Pattern* and then using the Object ➝ Pattern ➝ Pattern to Objects (**Shift+Alt+I**) command. The original objects will reappear in their original place (built-in *Patterns* will appear in the upper-left corner). After editing, the objects can be again turned into a *Pattern*. Both the old and new *Patterns* are available for use. Other objects filled with the original *Pattern* remain unchanged. The new *Pattern* must be “reapplied” to the object the *Pattern* came from.

Using Patterns

To change the fill of an object to a *Pattern*, simply click on the *Pattern* icon (▦) in the *Fill and Stroke* dialog. Then select the required *Pattern* from the pull-down menu. User created *Patterns* will be listed first.



Stripes 1:1

Use the **Node tool** to adjust position, scale, and rotation of the pattern on canvas. Use **Object > Pattern > Objects to Pattern** to create a new pattern from selection.

Blur: 0.0

Opacity, % 100.0

Pattern section of *Fill and Stroke* dialog.

Unfortunately, there is no preview of the *Patterns* (as with the *Gradient* tab).

Note

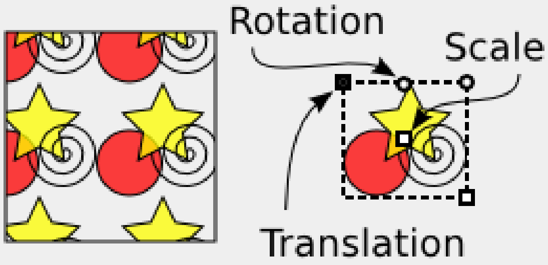
If you want a *Pattern* to transform with an object, you must toggle on this option using the icon that is in the *Tool Controls* when the *Select Tool* is in use.

Warning

Patterns are not exported to PS and EPS files via the old export routines and are exported incorrectly to PS and PDF files via the Cairo routines (Bug).

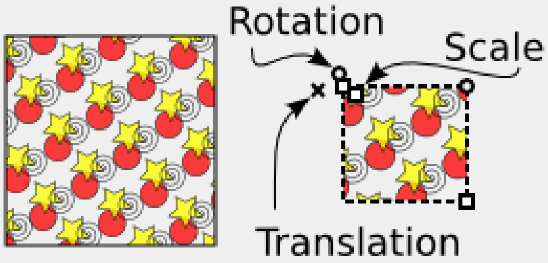
Adjusting Patterns

Adjusting *Patterns* is done by a set of handles. The handles will appear when an object with a *Pattern* fill is selected and the node or one of the shape tools is active. The handles will appear on the original objects that defined the *Pattern*, or the former location of those objects if they have been moved or deleted (unless the *Pattern* has been previously adjusted). In the case of built-on *Patterns* they will appear in the upper-left corner of the canvas. The following figure shows the location of the handles for a *Pattern* that has not been adjusted.



A *Pattern* (right) with an object filled with that *Pattern* (left). The translation handle is a "×" overlaying a *Rectangle* shape handle. The size handle is a square and the rotation handle a circle.

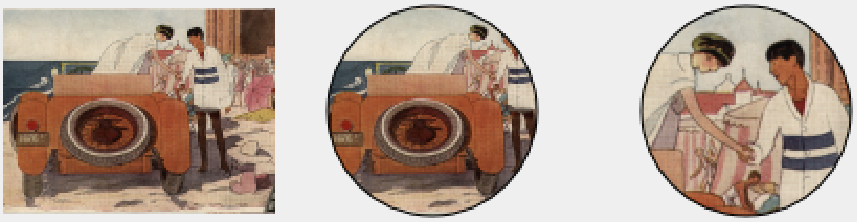
To adjust the origin, scale, and orientation of the *Pattern*, drag the translation handle (×), scale handle (square), and rotation handle (circle). The translation handle can conveniently be dragged over the object with the *Pattern* fill. Holding the **Ctrl** key down while dragging will restrict the movement to the horizontal or vertical direction. The scale is governed by the distance between the translation handle and scale handle, the orientation by the relative position of the rotation handle with the translation handle. The following figure shows how the fill changed after the handles have been adjusted.



A *Pattern* filled object (left) after adjusting the *Pattern* shown on the right. The *Pattern* was adjusted by dragging the handles shown.

Tip

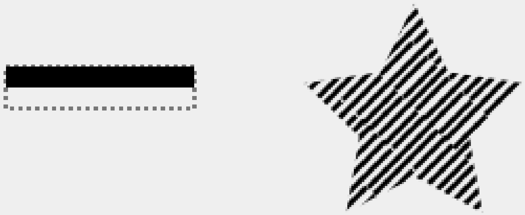
For *SVG* viewers that don't support clipping, you can crop a bitmap by turning the bitmap into a *Pattern* and using it to fill an arbitrary path.



A demonstration of cropping a bitmap image. Left: Original image. Center: Image after converting to *Pattern* and using as *Pattern* for circle. Right: Image after enlarging and shifting *Pattern*.

Hatchings

Inkscape now includes many *Patterns* that can be used as hatchings. If you need to create a new hatching here is the general procedure. The simplest hatching is to group two rectangular boxes, one with black fill and one with no fill as shown below.



The hatching *Pattern* on the left (surrounded by the dotted line) has been used for the star's *Fill*. The *Pattern* consists of two rectangles in a *Group*.

Warning

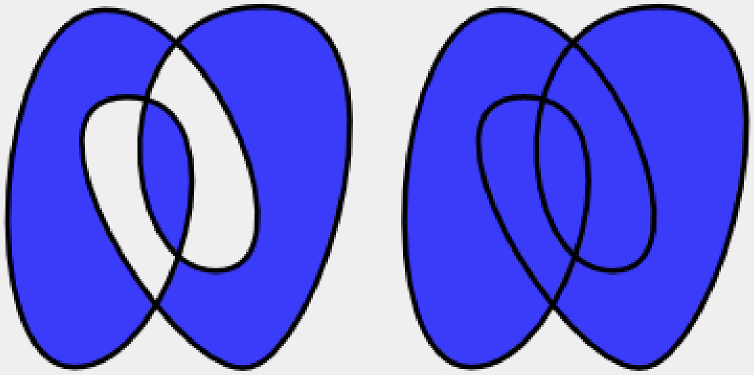
Inkscape has problems properly displaying *Patterns* at the *Pattern* boundaries. Inkscape export to a *PNG* also has problems. To work around this problem, you can make the *Pattern* rectangles wider than the maximum width needed to fill an object. Firefox, Opera, and Batik will display patterns without artifacts. Batik can be used for producing a high-quality *PNG*.

Fill Rule

The *Fill Rule* dictates what areas are filled when a path overlaps itself or one part of a complex path surrounds another part. The rule applies only to the *Fill* of an object (and not the *Stroke paint*). One can choose the *Fill Rule* from the two choices:

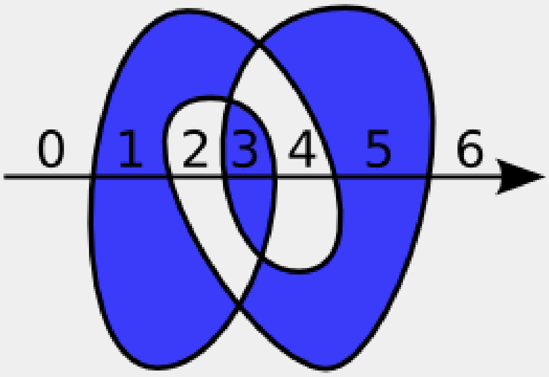
- Even-odd.
- Non-zero.

The difference between these rules is demonstrated in the following figures.



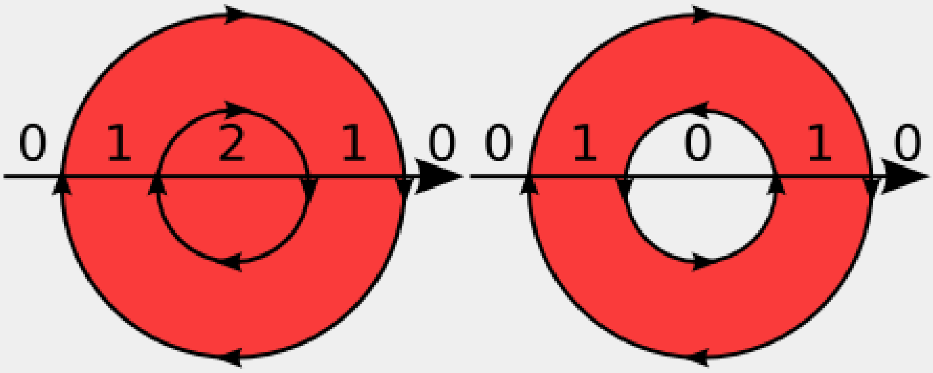
Fill rule: Even-odd vs. Non-zero.

Even-odd rule. In this rule, you start outside the path with the number zero. Every time you cross the object's path, you add one to the number. If the current number is odd, the region is inside the path and therefore colored. If the current number is even, the region is outside the path and is not colored.



Fill rule: a demonstration of counting. Even regions are *outside* the path, odd regions are *inside*.

Non-zero rule. In this rule, you start outside the path with the number zero. Every time you cross the object's path with the path going to the left, you add one to the number. If the path is going to the right, you subtract one. If the number is non-zero, the region is inside the path; if it is zero, the region is outside the path.



Non-zero fill rule applied to a path consisting of two sub-paths. Zero regions are *outside* the path; non-zero regions are *inside* the path. The arrows on the paths show the paths' direction.

Chapter 9. Attributes

Table of Contents

Stroke Style

Get the book.

Inkscape: Guide to a Vector Drawing Program

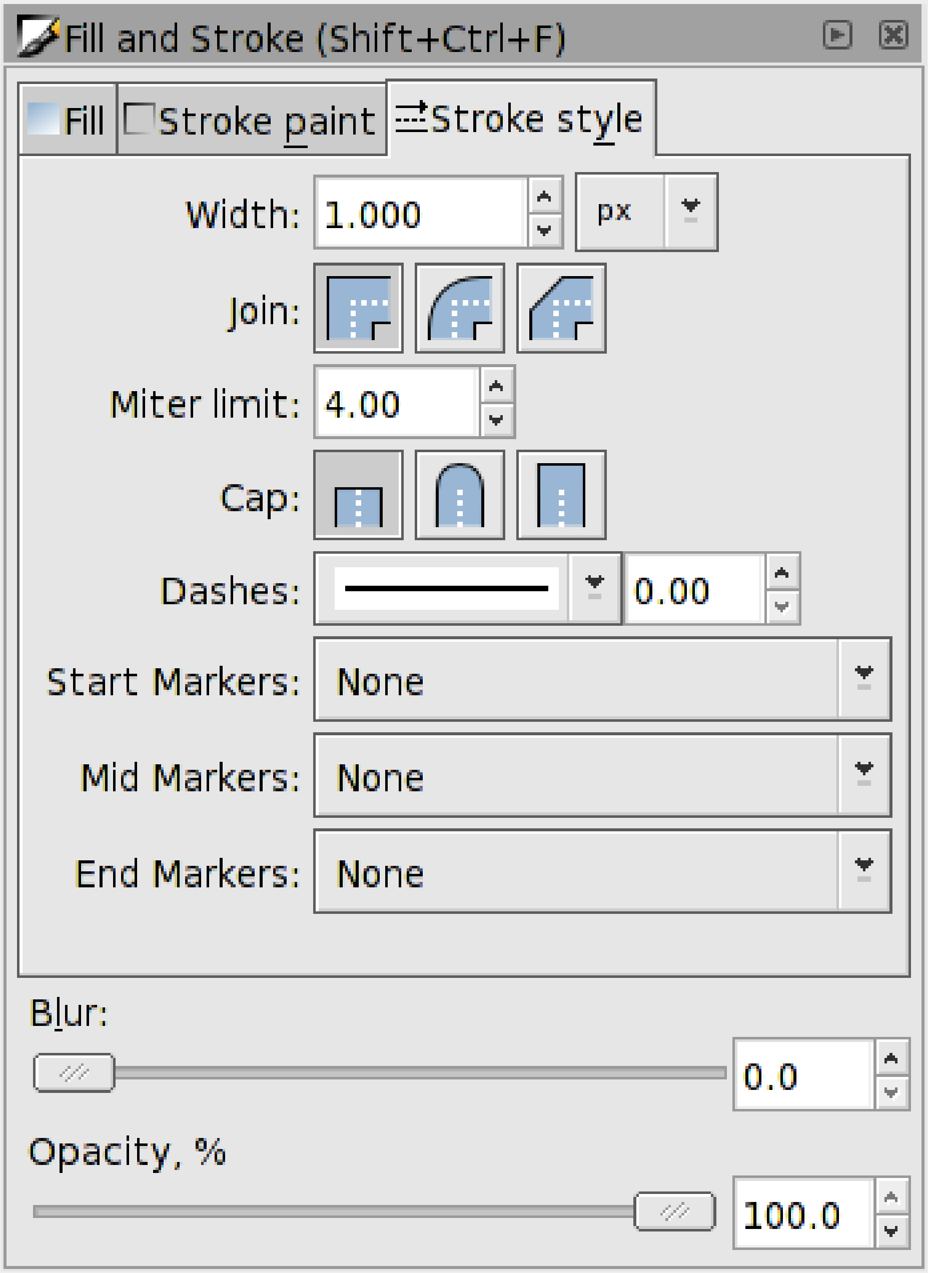
Get the book.

Inkscape » Attributes » Stroke Style

⬅ Index ➡

Stroke Style

The *Stroke style* tab has options to change the style of an object's path. This includes the stroke width, *Join* style, *Cap* style, *Dash Pattern*, and *Markers*.



Fill and Stroke (Shift+Ctrl+F)

Width: 1.000 px

Join:

Miter limit: 4.00

Cap:

Dashes: 0.00

Start Markers: None

Mid Markers: None

End Markers: None

Blur: 0.0

Opacity, % 100.0

Stroke style tab.

Stroke Width

Stroke width can be changed by the *Width* entry box. The units are specified by the drop-down menu on the right.

Note

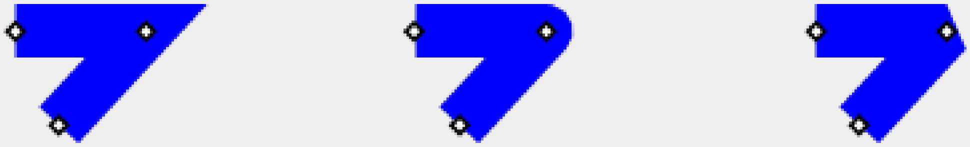
If you want the line width to transform with an object, you must toggle on this option using the ⇉ icon that is in the *Tool Controls* when the *Select Tool* is in use.

Join Style

The *Join* style is how two lines meeting at a corner should be joined together. The options are:

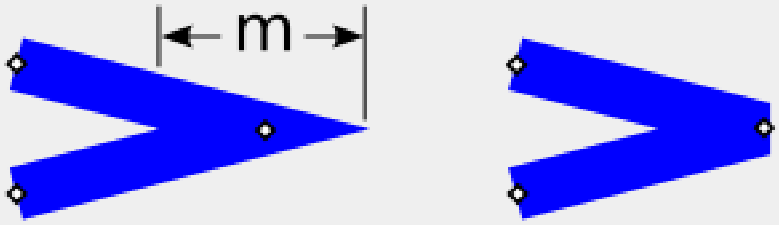
- Miter join.
- Round join.
- Bevel join.

The different *Join* styles are shown in the figure following.



Join styles: Miter (left), Round (center), Bevel (right). Note how the node is at the center of curvature for the Round style.

When the *Miter* option is selected, the length of the projection can become quite long if the two lines intersect at a small angle. In this case, it may be preferable to use the *Bevel* style. The *Miter limit* controls the point at which a *Miter* join automatically is converted to a *Bevel* join. If the distance from the inner intersection point to the tip of the triangle ("m" in the following figure) measured in stroke widths is more than the *Miter limit*, the join will be drawn in the *Bevel* style.



On the left is a Miter join. If the distance "m" as measured in stroke widths is more that the Miter limit, the corner will be drawn with a Bevel join as shown on the right. (The nodes are shown as diamonds.)

Note

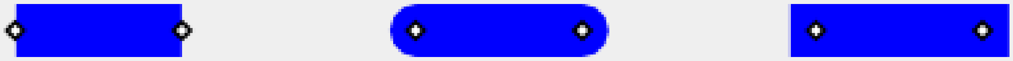
The bounding box is determined by assuming that the *Join* style is *Round*.

Cap Style

The *Cap* style determines how the end of a line is drawn. The options are:

- Butt cap.
- Round cap.
- Square cap.

The different *Cap* styles are shown in the figure below.



Cap styles: Butt (left), Round (center), Square (right). The nodes are shown as diamonds for reference.

Note

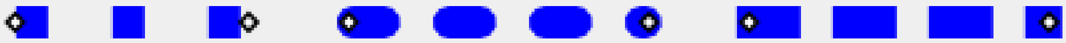
The bounding box is determined by assuming the *Cap* style is *Round*.

Dashes

A wide selection of *Dash* patterns are available from the drop-down *Dash* menu. The patterns scale with the stroke width.

Note

Each *Dash* takes on the *Cap* style as shown in the figure below.

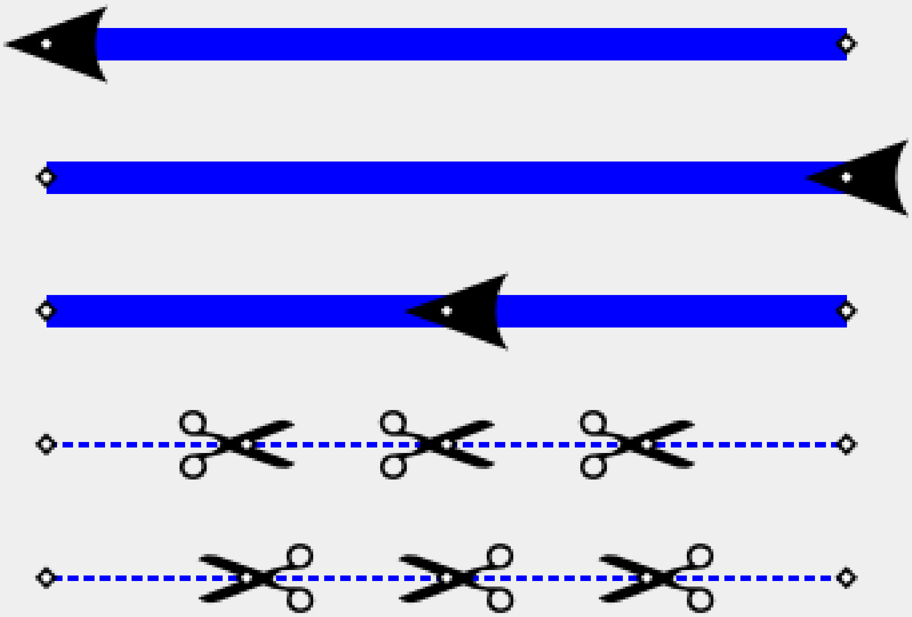


The same *Dash* style but with different *Cap* styles: Butt (left), Round (center), Square (right). The nodes are shown as diamonds for reference.

The entry box next to the *Dash* menu is for the *Dash offset*. The offset shifts the *Dash* pattern parallel to the path. The units are in stroke width.

Markers

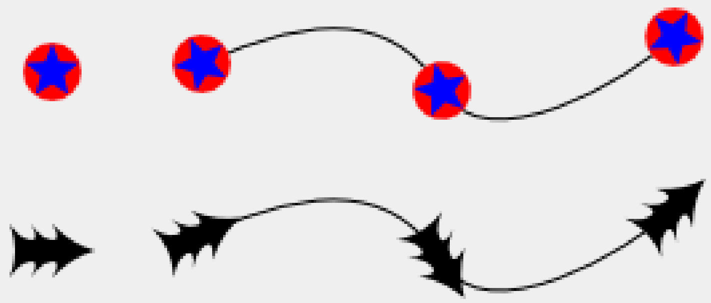
Markers are objects like arrow heads placed along a path. Different *Markers* can be specified for the start, middle, and end of a path. Middle *Markers* are placed at the location of every non-end node.



Examples of *Markers*. From the top down: *Start arrow* at the start of a line. *Start arrow* at the end of a line. *Start arrow* in the middle of a line (a node has been added). Scissors at middle nodes. Scissors at middle nodes, path reversed.

New in v0.46:

A custom *Marker* can be created by selecting the object or objects that you wish to use as a *Marker* and then using the Object → Object to Marker command. The selected objects will disappear and a new entry will appear in the *Marker* pull-down menus of the [*Fill and Stroke*]() dialog. The *Marker* is created assuming a horizontal orientation for the path. The point of attachment to a node is the center of the *bounding box* for the *Marker*. Warning: While the marker will display fine in Inkscape, only a fourth of it will be displayed in most other *SVG* renderers. Adding the attribute *style="overflow:visible"* to the *Marker* definition will fix the problem (Bug). Note, that custom *Markers* can be added to Inkscape; see the section called "Custom Markers" in Chapter 20, *Customization*.



Examples of custom *Markers*. The objects on the left were converted to *Markers* by first duplicating and then using the Object → Object to Marker command. They were then applied to the paths on the right.

Note: The Object → Object to Marker procedure is a bit buggy. The new *Marker* may not show up in the list until another *Marker* is applied to the path. If multiple objects are converted, the *z-order* is reversed. *Group* the objects first to avoid this problem.

Two problems exist with *Markers*. The first is that *Markers* do not take the color of the stroke. This can be worked around by using the *Color Markers to Match Stroke* effect, by editing the *SVG* file with the *XML Editor*, or adding a custom, pre-colored *Marker*. An alternative solution is to convert the path with the *Markers* to a path (Path → Object to Path (**Shift+Ctrl+C**)). This creates a *Group* with the *Markers* converted to separate objects, which can be colored independently.

The second problem is that *Markers* scale with line width. The line width had to be reduced in the above figure for the scissors examples, to give the scissors a reasonable size. Again, one could edit the *SVG* file to adjust the *Marker* size.

Tip

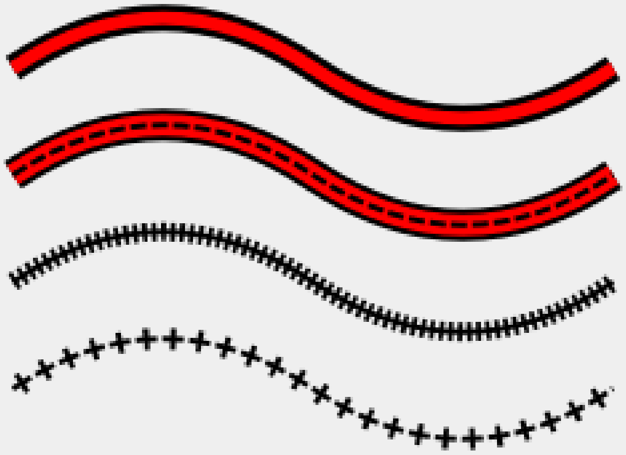
> To place middle *Markers* evenly along a path you need to have evenly spaced nodes. For straight horizontal and vertical lines, nodes can be distributed evenly using the *Align and Distribute* dialog. To add just one node halfway between two existing nodes click on the *Insert* icon in the *Node Tool*-*Tool Controls*. To add one node anywhere on an existing path, double click on the path where you want the node (or single click while holding down the **Ctrl**+**Alt** keys). To add multiple nodes evenly spaced between existing nodes, use the *Add Nodes* effect.

Another thing to note is that *Markers* are included in the *Visual* bounding box calculation.

Complex Strokes

A complex *Stroke* can be created by overlaying two or more paths with different *Stroke* attributes. The easiest way to make exact copies of a path is to use the Edit → Duplicate (**Ctrl+D**) command.

If one uses *Clones* of a path (Edit → Clone → Clone (**Alt+D**)), then one can adjust the original path at a later time and all the *Clones* will change too. This requires unsetting the *Stroke* attributes of the original path (use the XML editor to unset the *Stroke* width). Since the original path's attributes are unset, it will not be visible and cannot be used as part of the visible *Stroke*.



Examples of complex *Strokes*. From top to bottom: Road: consists of a 5 pixel-wide red path over a 10 pixel-wide black path. Divided road: consists of a 2 pixel-wide dashed black path over an 8 pixel-wide red path, over a 12 pixel-wide black path. Railroad tracks: consists of a 7 pixel-wide dashed path, with dash pattern (1.75, 1.75) over a 2 pixel-wide solid path. Border: consists of an 8 pixel-wide dashed path, with dash pattern (2, 8) and an offset of 7 pixels (use XML editor to set) over a 2 pixel-wide dashed path with dash pattern

(8, 2).

Fill and Stroke Paint

Table of Contents

Chapter 10. Tweak Tool

Inkscape » Tweak Tool » Tweaking Paths

Index

Tweaking Paths

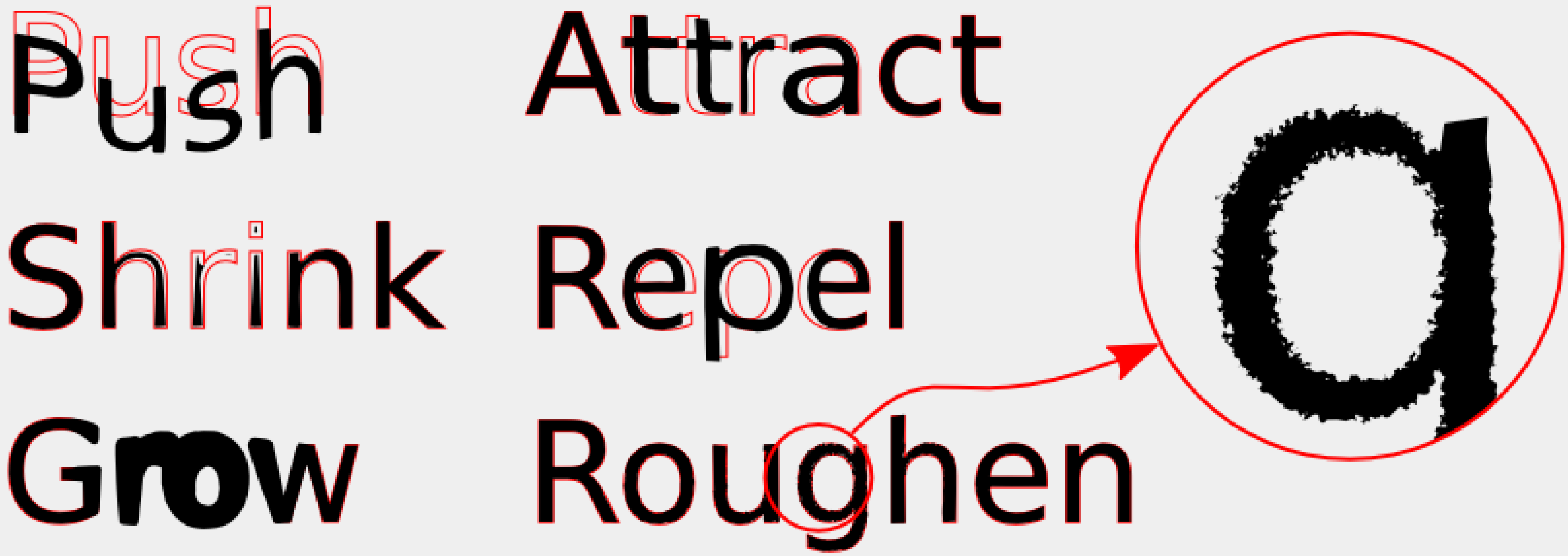
A variety of *Tweak Tool* modes modify paths. If an object is not a path (i.e. *Rectangles*, *Ellipses*, text) it is first converted to a path. Unlike the *Node Tool*, nodes do not need to be selected.

All path modes share the *Fidelity* parameter. The range for the parameter is from 1 to 100. A low value gives a rough distortion using few nodes, a high value gives a smoother distortion but at the cost of creating large numbers of nodes. Note that any path distortion will affect the entire path, even the parts that are far away from the cursor.

The tool has several known problems. If used on an open path, the path will become closed. It doesn't work well on straight lines. Thin calligraphic type strokes will suddenly disappear or change drastically in shape. A number of possible work-arounds: 1. Increase *Fidelity*, 2. Use the Path → Simplify (**Ctrl+L**) command (which results in a new set of nodes which may avoid the problem), 3. Use multiple “soft” strokes (i.e. reduce the *Force* parameter). The problems are from faults in the library routines used to manipulate the paths. A new library is currently under development.

The following path modes are available:

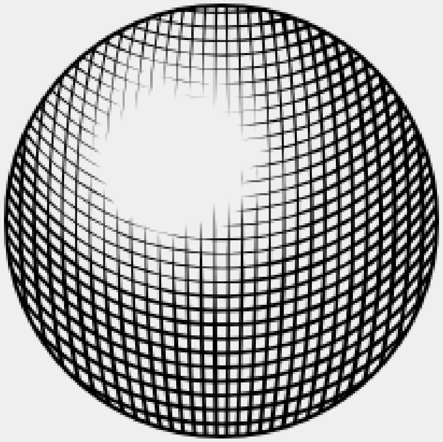
- **Push.** (**Shift**+**P**) Default. Displaces path in direction of drag.
- **Shrink.** (**Shift**+**S**) Insets path near cursor while dragging.
- **Grow.** (**Shift**+**G**) Outsets path near cursor while dragging.
- **Attract.** (**Shift**+**A**) Displaces path inward toward cursor while dragging.
- **Repel.** (**Shift**+**E**) Displaces path outward from cursor while dragging.
- **Roughen.** (**Shift**+**R**) Roughens path near dragged cursor.



The *Tweak Tool* was applied with a different mode to each of the text objects. In each case the cursor was moved from top to bottom. The mode is indicated by the text. The original text is outlined in red.

A few short cuts exist: **Ctrl** switches temporarily to *Shrink* mode. **Ctrl**+**Shift** switches temporarily to *Grow* mode.

The *Tweak Tool* is very useful for manipulating hatchings.



The hatching was created by using the *Interpolate Effect* twice to create two quasi-orthogonal sets of lines that were then clipped by a circle. When created, each set of lines belonged to one compound path. The compound paths were broken apart (Path → Break Apart (**Shift+Ctrl+K**)) and then the strokes converted to paths (Path → Stroke to Path (**Ctrl+Alt+C**)). Finally, the *Tweak Tool* was used in *Shrink* mode to narrow the width of the lines to create a 3D shading effect.

Chapter 10. Tweak Tool

Table of Contents

Tweaking Colors

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Tweak Tool » Tweaking Colors

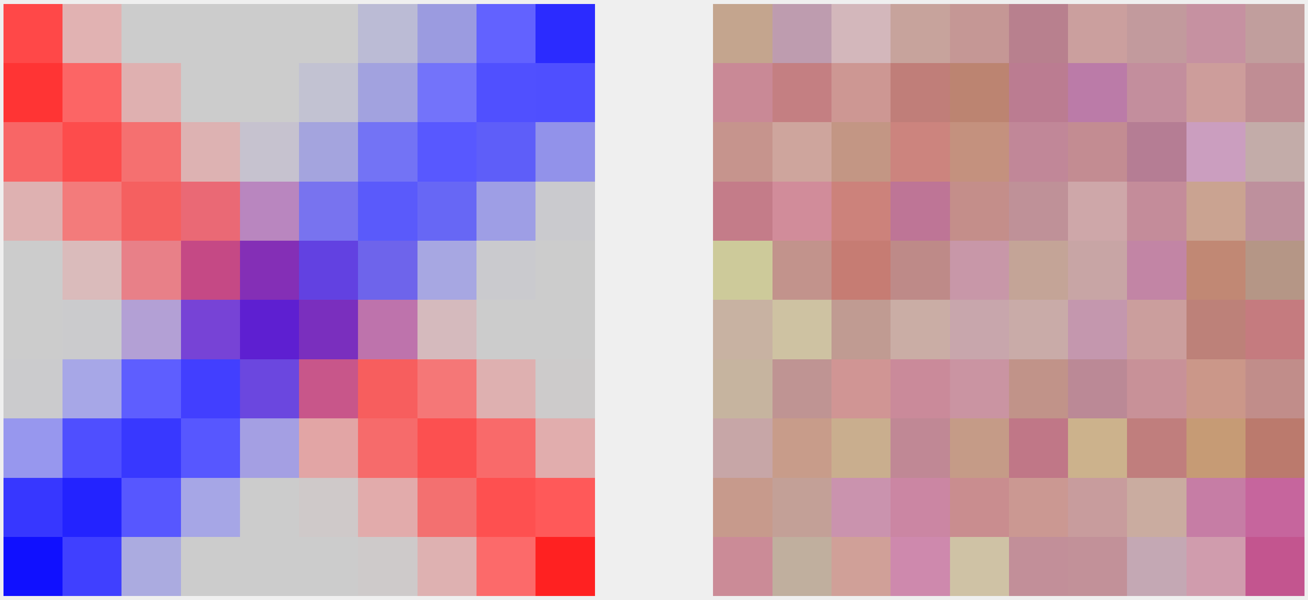
Index

Tweaking Colors

The *Tweak Tool* can be used to make small color changes to objects. In doing so, it changes the *Fill* and/or *Stroke* color of an entire object. While it can be used to change the color of *Gradient* stop, it can not create *Gradients* nor can it modify part of an object leaving the rest alone. This is due to the fundamental nature of vector objects. It can, however, be used to adjust the color of a group of objects, with the objects closer to the cursor changing the most.

Two modes with several options exist for tweaking colors.

- **Paint.** (**Shift**+**C**) Paint with current *Fill* and *Stroke* style. When the tool is selected, changing the *Fill* or *Stroke* by using the *Fill and Stroke* dialog or the *Palette* will change only the *Current style* and not any objects. If either *Fill* or *Stroke* are not set, the *Fill* or *Stroke* of tweaked objects will remain unchanged.
- **Jitter.** (**Shift**+**J**) Randomize colors. The *Current style* is not used.



The *Tweak Tool* was used to modify a grid of gray squares. Left: In *Paint* mode the tool was used to paint a diagonal red and blue lines. Right: In *Jitter* mode the tool was used to add a random color to the squares.

The *Tweak Tool* in a color mode has four options. It can act independently on a color's hue, saturation, lightness (*HSL*) and *opacity*. The options can be toggled on independently by the H, S, L, and O icons next to the *Channel* label in the *Tool Controls*.

Tweaking Paths

Table of Contents

Chapter 11. Paint Bucket Tool

Inkscape: Guide to a Vector Drawing Program

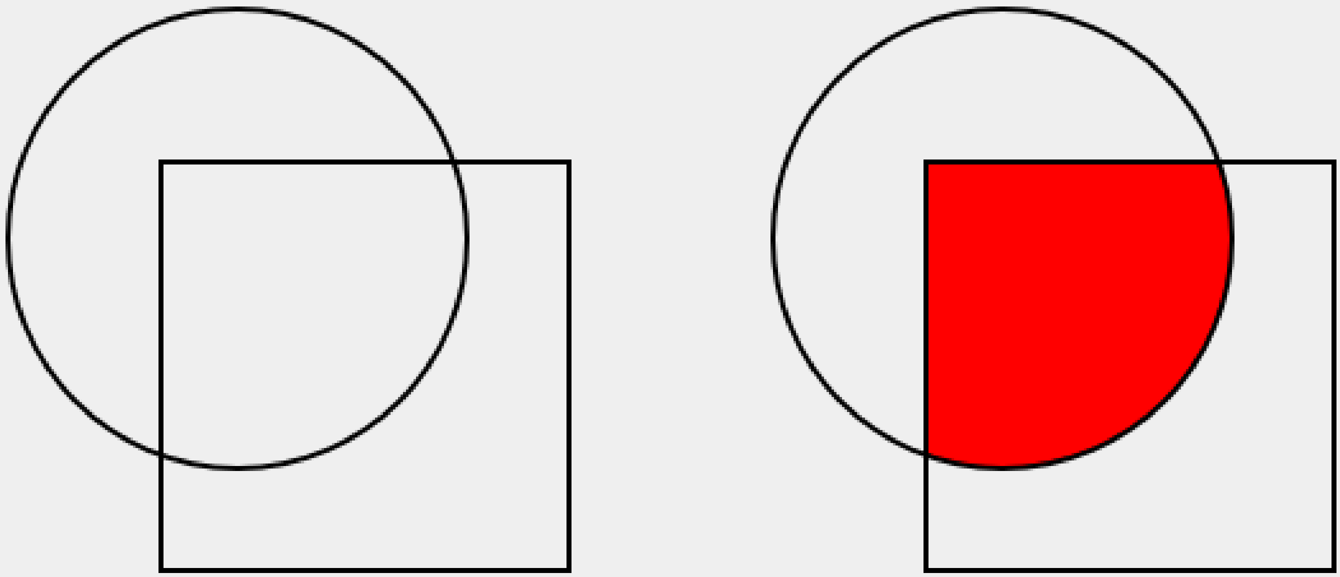
Get the book.

Inkscape » Paint Bucket Tool » Simple Use

← Index →

Simple Use

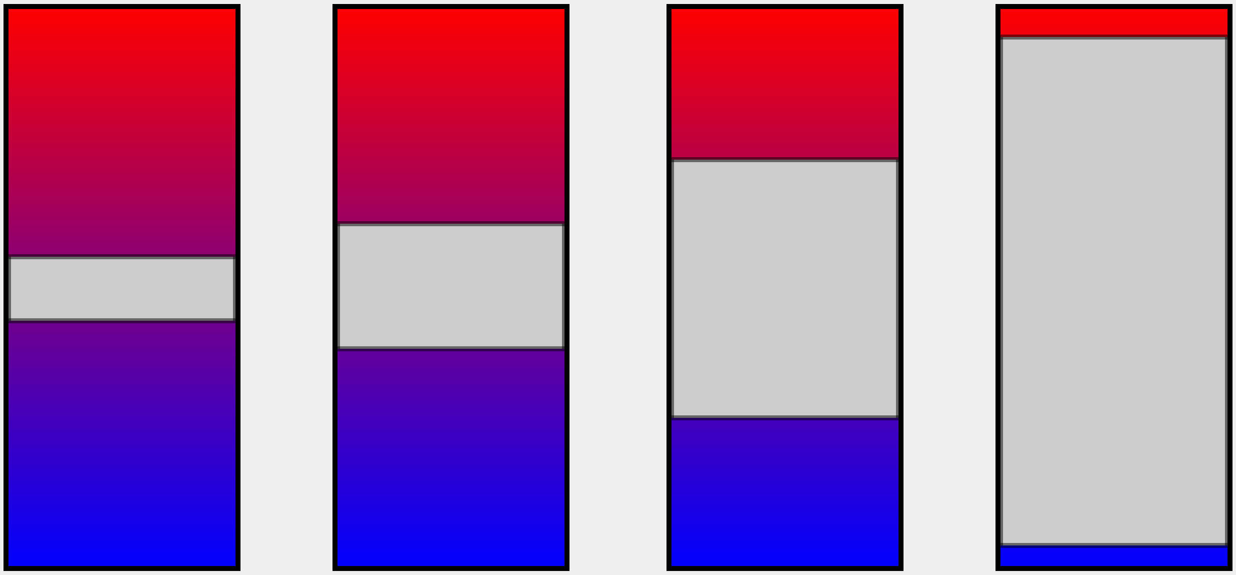
The simplest use is to fill an area defined by two different objects. Select the *Paint Bucket Tool* (, **Shift**+**F7**) from the *Tool Box* and then click in the overlapping region. A new object is created with the current *Fill* color.



Left: Before *Paint Bucket* fill. Right: After clicking on the overlapping region with the *Paint Bucket Tool* (with the *Fill* color set to red).

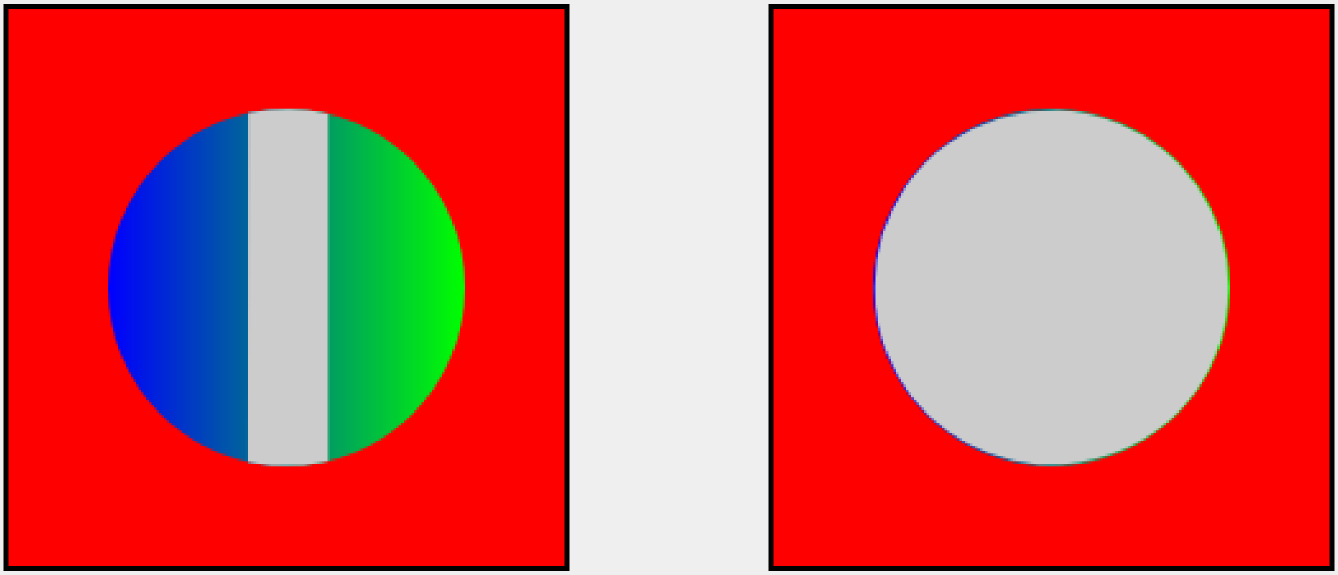
The cursor position when clicking determines the starting point of the fill region. The filling algorithm recursively adds neighboring areas to the region until pixels are found that don't meet the criteria determined by the parameters in the *Tool Controls*.

The *Threshold* parameter controls how close a color must match the color (or *Alpha*) under the cursor. Lower thresholds will match a smaller range.



The gray areas have been filled by clicking on the centers of the rectangles with the *Paint Bucket Tool*. From left to right, the *Threshold* parameter was set to 5, 10, 20, and 40.

The *Fill by* parameter controls what color value is used to determine the the color matching. Choices are: *Visible Colors* (default), *Red*, *Blue*, *Green*, *Hue*, *Saturation*, *Lightness*, and *Alpha*.



The circles were first filled with a blue to green gradient. The gray areas were created with the *Paint Bucket Tool* by clicking on the centers of the circles. In both cases, the *Threshold* parameter was set to 10. The *Fill by* parameter was set to *Visible Colors* on the left and *Red* on the right. With a value of *Red* the blue and green of the circle are ignored in the color matching, thus the bucket fill area reaches the circle's edge where there is a large change in the red level.

Holding the **Ctrl** down while clicking on an object with the *Paint Bucket Tool* will set the *Fill* and *Stroke* to the current style without preforming any filling.

Chapter 11. Paint Bucket Tool

Table of Contents

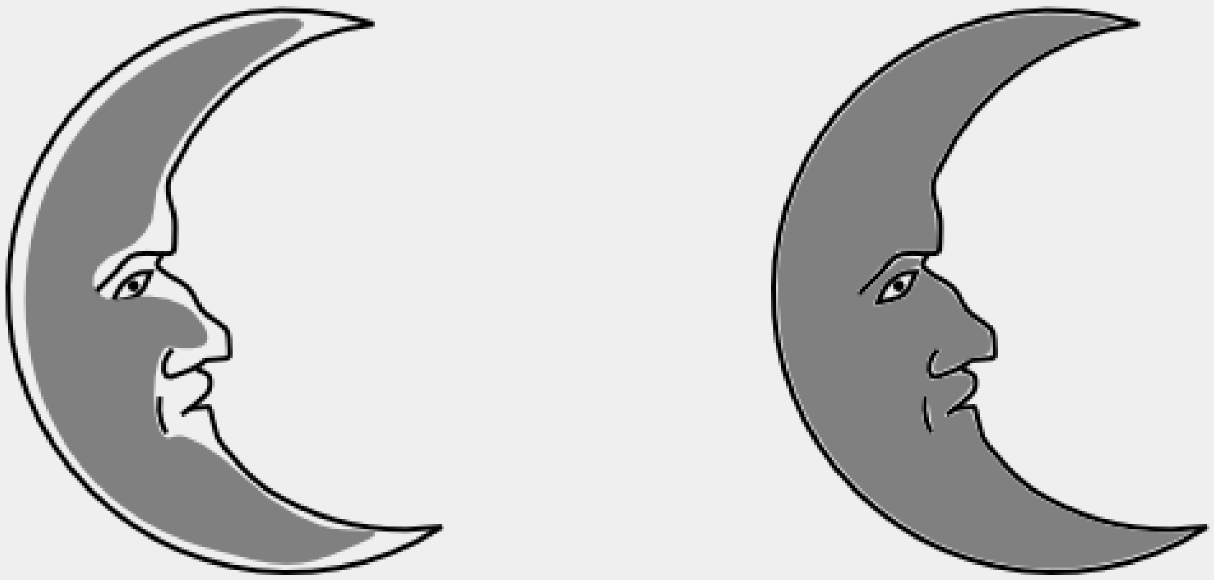
Filling Fidelity

Inkscape » Paint Bucket Tool » Filling Fidelity

Index

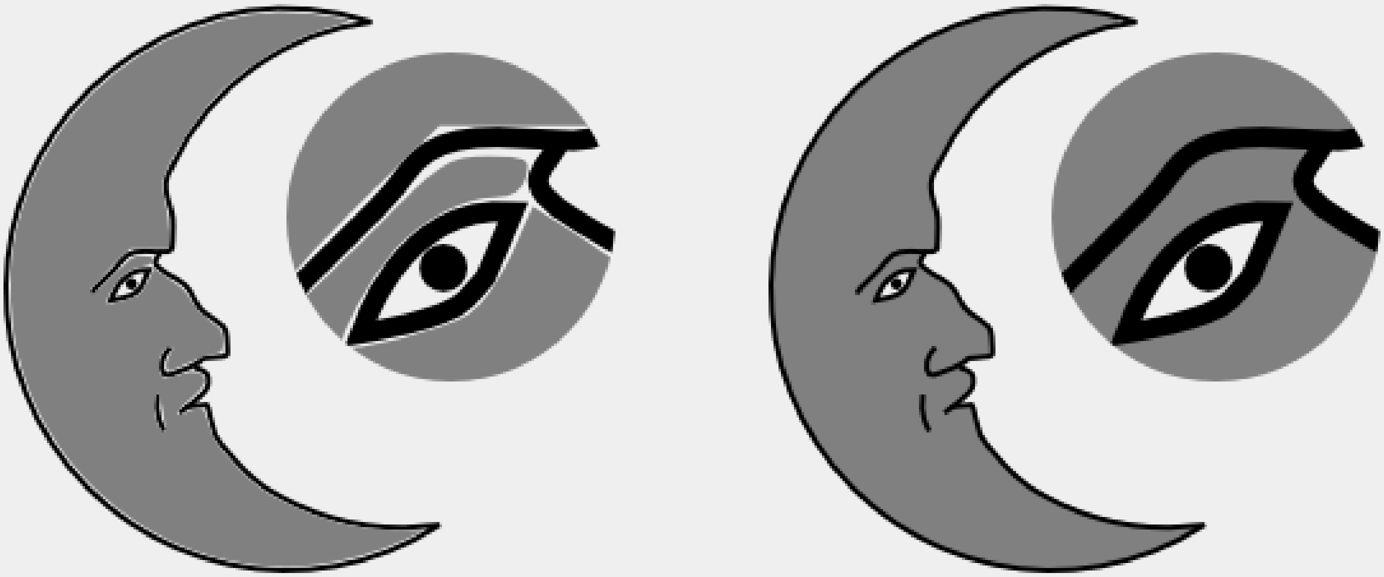
Filling Fidelity

The filling process works first by determining which pixels should be filled and then tracing those pixels to produce a vectorized path. The tracing process has limited precision which can result in inaccuracies in the filled region. You can improve the filling accuracy several ways. The first way is to zoom in on the region you are filling. Zooming in increases the number of screen pixels in the filled region which results in a more accurate tracing.



The moon man was given a gray fill using the *Paint Bucket Tool* while the zoom level was 25% on the left and 200% on the right.

The second way to improve the filling accuracy is to expand the fill region slightly in a process akin to “trapping” that printers use to account for small misalignments in their printing plates. This works especially well for cartoons where the fills can be put on a separate *Layer* beneath a *Layer* containing the black lines. The amount of expansion is controlled by the *Grow/shrink by* parameter. As the name suggests, one can both expand and reduce the fill area.



The moon man was given a gray fill using the *Paint Bucket Tool* while the zoom level was 200%. On the left, no “trapping” was used. On the right the *Grow/shrink by* parameter was set to 1 pixel. The fill was then moved behind the line drawing. On both sides, the region around the eye has been expanded by a factor of four.

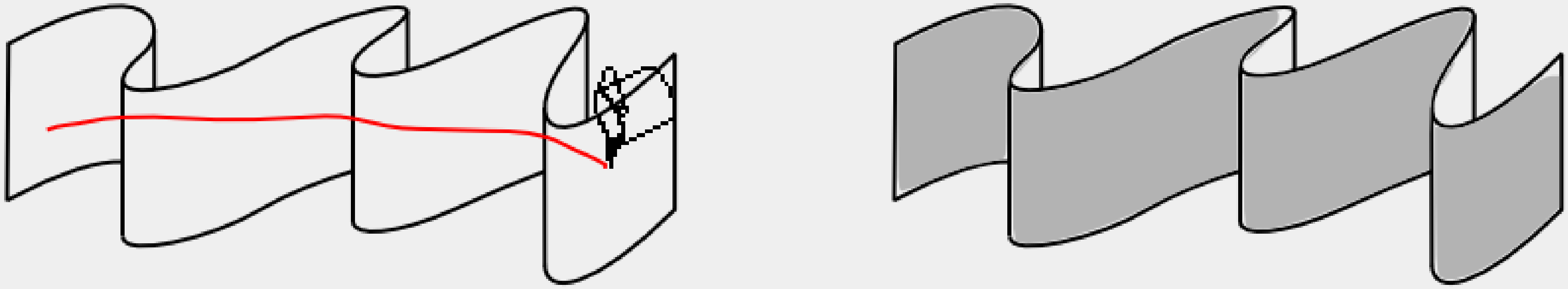
Simple Use

Table of Contents

Filling Multiple Regions

Filling Multiple Regions

Click-dragging the *Paint Bucket Tool* while holding the **Alt** key down across several noncontiguous regions will cause all the regions to be filled. (Not holding the **Alt** down will cause the borders to also be filled.)



The *Paint Bucket Tool* was click-dragged with the **Alt** key down across the object on the left as shown by the red line. This resulted in the figure on the right.

Filling Fidelity

Table of Contents

Closing Gaps

Get the book.

Inkscape: Guide to a Vector Drawing Program

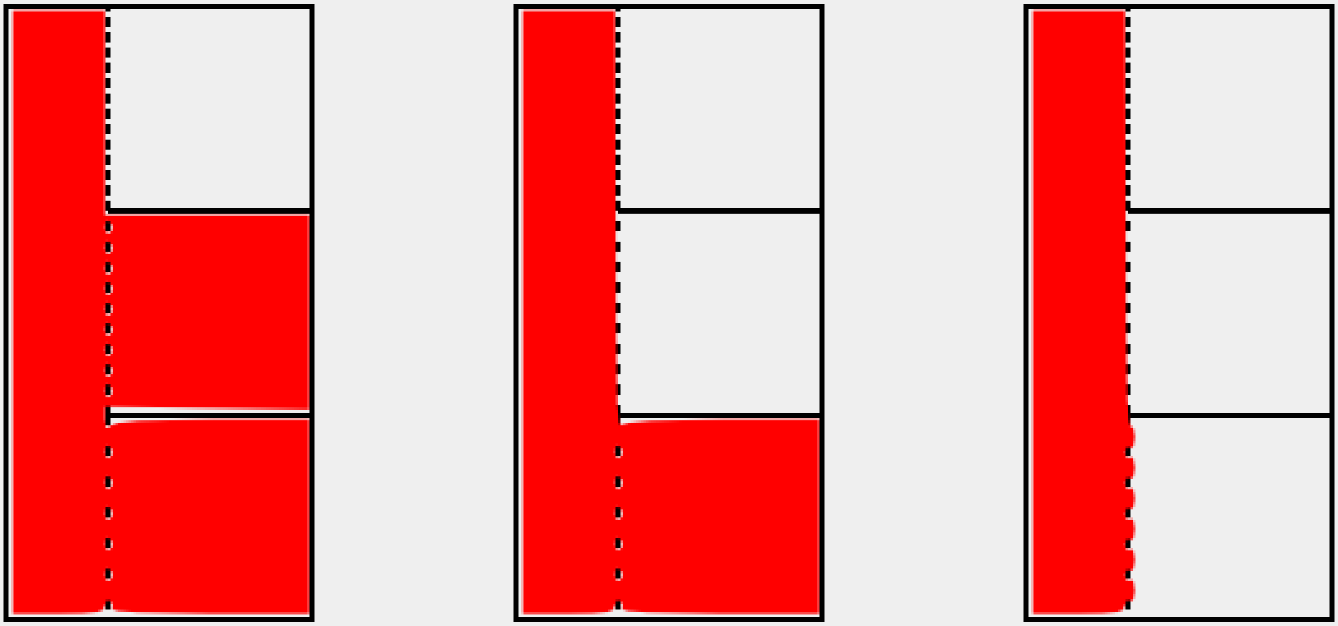
Get the book.

Inkscape » Paint Bucket Tool » Closing Gaps

⬅ Index ➡

Closing Gaps

Small gaps in borders can be “filled” by setting the *Close gaps* to a value other than *None*. This prevents fills from leaking into undesired areas just because there is a small break in a line (as might happen in tracing a cartoon). Note that the gaps are defined in terms of screen pixels so different zoom levels will give different results. Also note, calculating the fill when there are gaps can take some time.



The *Paint Bucket Tool* was applied to the leftmost region in the three figures. The square areas are separated from this region by dashed lines with gaps of 1, 2, and 4 pixels from top to bottom. The *Close gaps* parameter was set, from left to right, to *Small*, *Medium*, *Large*. The filling was done at a zoom of 100%.

← Filling Multiple Regions

↑ Table of Contents

→ Adding to a Fill

Inkscape: Guide to a Vector Drawing Program

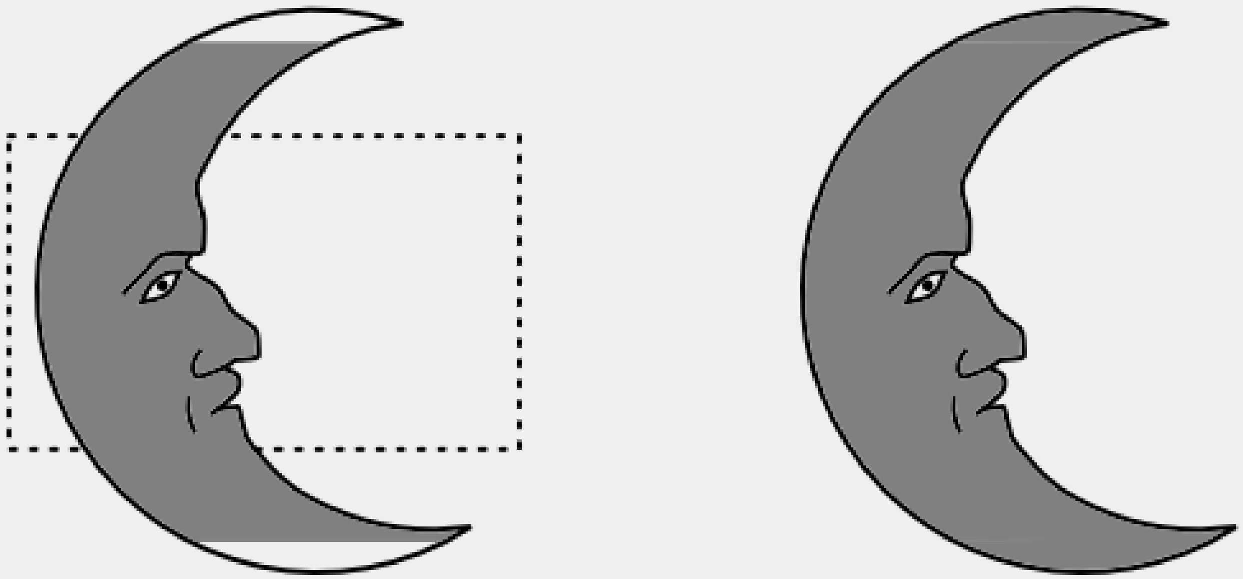
Get the book.

Inkscape » Paint Bucket Tool » Adding to a Fill

Index

Adding to a Fill

Since the *Paint Bucket Tool* uses pixels to calculate the area to be enclosed, Inkscape will clip the region offscreen to prevent the number of pixels that go into the filling algorithm from becoming too large. If this happens you can either zoom out to do the fill or you can do the fill in pieces. By holding the **Shift** down while clicking you can add to an existing fill. Unfortunately, this can lead to rendering artifacts between adjacent pieces even though they are part of the same path. Two solutions: 1. Set the *Grow/shrink by* parameter to 0.10. This will ensure a slight overlap during the filling process. The overlap is removed in the unioning step. 2. Use the *Node Tool* to adjust the nodes to overlap areas and then use the Path → Union (**Ctrl++**) command to remove the overlap.



The *Paint Bucket Tool* was applied to the left moon under high zoom. The part visible onscreen is shown by the dotted rectangle. Note the missing fill at the top and bottom. On the right, the missing sections have been added by scrolling in turn so that each missing section was visible onscreen and then with the first filled section selected, **Shift**-clicking on the missing area.

Closing Gaps

Table of Contents

Chapter 12. Clipping and Masking

Clipping

Any path, regular shape, or regular text object can be used as a clipping path.

To clip an object (or *Group*), select both the object and the clipping path. The clipping path must be above the object to be clipped in *z-order*. Then use the Object → Clip → Set command. To unclip a clipped object, select the object and use the Object → Clip → Release command. The clipping path is then restored as a regular object, placed just above the formerly clipped object in *z-order*.

More than one object (or *Group*) can be clipped at the same time. Just follow the above instructions but include all the objects to be clipped in the selection (with the clipping path on top). Inkscape will store one copy of the clipping path in the *<defs>* section of the *SVG* file for each clipped object; thus, the clipped objects can be edited separately.

When a clipped object is selected, the *Status Bar* will display the type of object clipped along with the word “clipped.” The *bounding box* of the clipped object is defined by the intersection of the *bounding box* of the unclipped object with the *bounding box* of the clipping path. (However, if an object inside a clipped *Group* is selected, the unclipped *bounding box* of that object will be displayed.)

Chapter 12. Clipping and Masking

Table of Contents

Masking

Masking

Any object can be used to mask another object. The opacity and lightness of the mask determines the opacity of the masked objects. A masked object will be fully opaque only at places that are: inside the mask path, where the mask has maximum lightness (i.e., white), and the mask has maximum *Alpha*. So to summarize:

- Regions with minimum lightness (i.e., black) will be fully transparent.
- Regions with minimum *Alpha* (zero alpha) will be fully transparent.
- Regions outside the mask will be fully transparent.

To mask an object (or *Group*), select both the object to be masked and the object to be used as a mask. The mask must be above the object to be masked in *z-order*. Then use the Object → Mask → Set command. To unmask a masked object, select the object and use the Object → Mask → Release command. The mask is then restored as a regular object and is placed just above the formerly masked object in *z-order*.

More than one object (or *Group*) can be masked at the same time. Just follow the above instructions but include all the objects to be masked in the selection (with the mask on top). Inkscape will store one copy of the mask in the *<defs>* section of the *SVG* file for each masked object; thus, the masked objects can be edited separately.

When a masked object is selected, the *Status Bar* will display the type of object clipped along with the word “masked.” The *bounding box* of the masked object is the same as that of the unmasked object.

Clipping

Table of Contents

Chapter 13. Filter Effects

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Filter Effects » Basic Use

← Index →

Basic Use

All filters can be defined and applied through the *Filter Effects* dialog. The *Gaussian Blur* filter can also be used directly through the *Fill and Stroke* dialog. The *Blend* filter can also be applied to an entire *Layer* through the *Layers* dialog.

The *Fill and Stroke* dialog has a *Blur* slider that when moved from zero, creates and attaches a *Gaussian Blur* filter to selected objects. If the slider is reset to zero, the filter is automatically removed and deleted. The same principle works on the *Layers* dialog where a choosing anything but *Normal* creates a *Blend* filter and attaches it to the *Layer*. Setting the drop-down menu back to *Normal* deletes the filter.

All *Filters* primitives can be used through the *Filter Effects* dialog. In this dialog, *Filters* must first be declared and then applied to selected objects. Complex filters can be composed by combining *Filter* primitives to create new effects such as a “Drop-Shadow” filter.

The source of the graphic used as an input for a *Filter* primitive is either the output of another *Filter* primitive or one of the following sources:

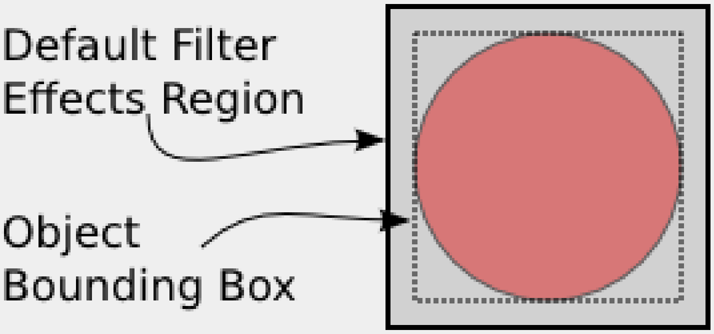
- Source Graphic. Use the object as the source for the *Filter* primitive.
- Source Alpha. Use the *Alpha* of the object as the source for the *Filter* primitive.
- Background Image. Use the region under the *Filter* at the time the *Filter* is invoked.
- Background Alpha. Use the *Alpha* of the region under the *Filter* at the time the *Filter* is invoked.
- Fill Paint: *Not implemented.* Use the *Fill* of the target object as the input to a *Filter*

primitive as if the object had an infinite extent. Useful if the Fill is a Gradient or Pattern with transparent or semi-transparent regions.

- Stroke Paint: *Not implemented.* Use the Stroke of the target object as the input to a *Filter* primitive (see Fill Paint above).

> **Warning**
>
> Whenever an input to a *Filter* is specified to be *Background Image* or *Background Alpha*, the SVG file must include the “enableBackground” tag to tell the SVG renderer it must keep a copy of the background around. Inkscape does not add this tag for you except in the case of using the Layer dialog to add a blend. A work-around is to use the Layer dialog to temporary add and then remove a Blend filter; this will leave the necessary tag in place

A filter effect is applied to a region defined relative to the object to which it is attached. By default, the *Filter Effects Region* ranges from -0.1 to 1.1 in units of the object's bounding box width and height. This area may not always be sufficient. For example, if a large shift is prescribed in the Offset filter, the area must be increased. The area can be adjusted under the *Filter General Settings* tab of the *Filter Effects* dialog. (It can also be set through the XML Editor dialog, attributes *x*, *y*, *width*, and *height*.)



The default *Filter Effects Region* is shown for the red circle.

The *Filter Effects Region* can be defined in terms of the object's bounding box or by units in the current user coordinate system by setting the “filterUnits” parameter to “objectBoundingBox” or “userSpaceOnUse”. Inkscape currently supports only the use of the first in the GUI.

Inkscape » Filter Effects » Filter Effects Dialog

Index

Filter Effects Dialog

Adding a Filter

Click on the *New* button at the bottom left to add a new filter. Alternatively, right-click on an existing filter in the *Filter* list to duplicate it. The name of the filter can also be changed by clicking on the name.

Defining a Filter

A filter is defined by selecting filter primitives from the drop-down list next to the *Add Effect* button. A short description of the filter is displayed when a filter is selected. The description can be toggled on or off under the *Filter* tab of the *Inkscape Preferences* dialog.

Once a primitive is selected, click on the *Add Effect* button to add it to the filter. The input(s) of the filters are automatically attached to the *Source Graphic* or the output of a previously added filter primitive. The default connections are shown as gray lines originating from a triangle right of the filter primitive name. Explicitly defined connections are shown in black.

The inputs of a filter can be reassigned by click-dragging from the triangle at the right of the filter name to one of the columns on the right (e.g. *Source Graphic*), or by click-dragging to another of the filter primitives.

The filter primitives can be reordered by click-dragging a filter primitive in the list to another place in the list.

Note

Inkscape frequently fails to update the display when tweaking *Filter Effects* parameters. Nudging the object up and down is sufficient to force an update.

Applying a Filter

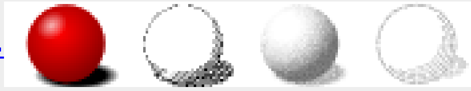
To apply a filter to an object(s), select the object(s) and check the box next to the filter name. Uncheck the box to remove a filter. If multiple objects are selected with different filters, all the boxes corresponding to those filters will be checked.

Basic Use

Table of Contents

Mini Tutorial - A Drop Shadow

Get the book.



Inkscape: Guide to a Vector Drawing Program

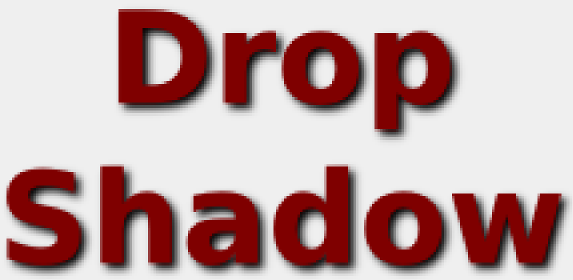
Get the book.

Inkscape » Filter Effects » Mini Tutorial - A Drop Shadow

← Index →

Mini Tutorial - A Drop Shadow

Drop shadows are a perfect use for filters. The shadow automatically updates when the source object changes. Here is a step-by-step tutorial on creating a drop shadow:



A drop shadow created using only filters.

Procedure 13.1. Drop Shadow

We will construct a compound filter that creates a drop shadow when applied to a text object.

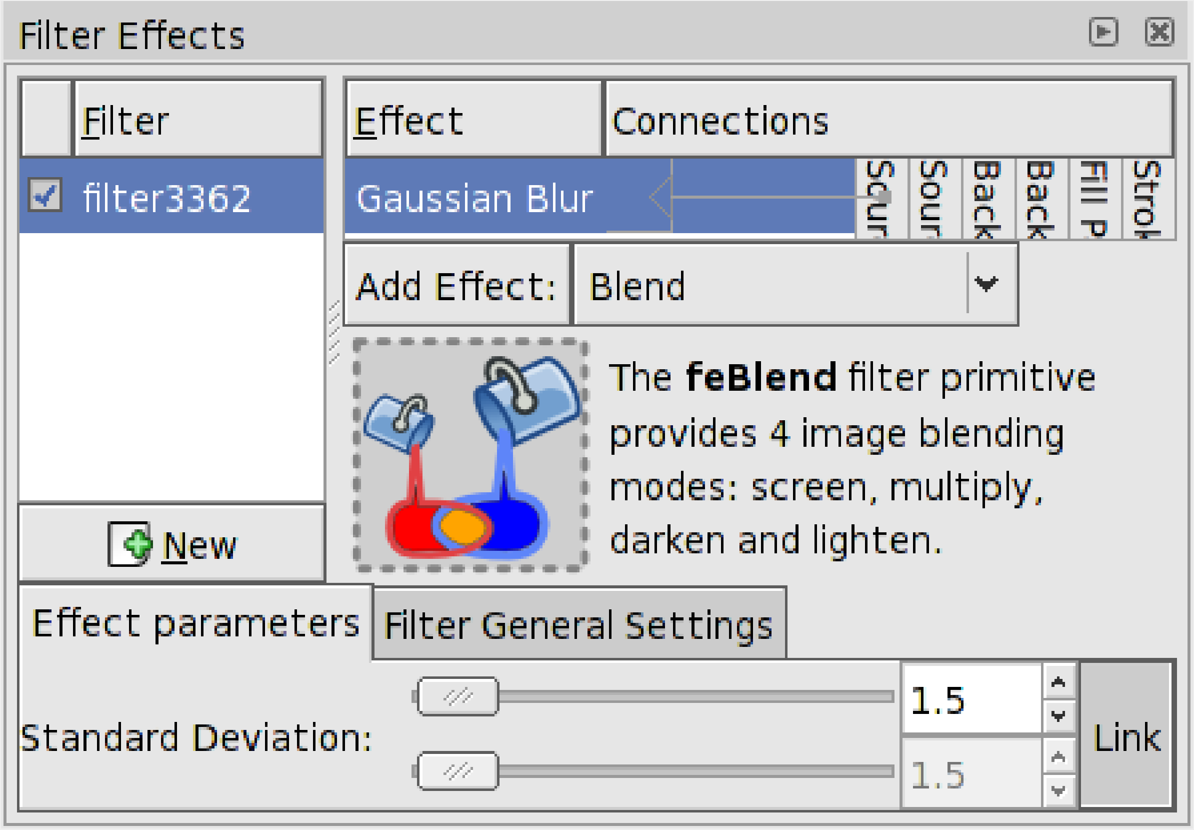
1. **Add Text**

   Create a text object. In the example, I've used the text “Drop Shadow”. Give the text a color. I've used dark red.
2. **Add Gaussian Blur**

Select the text. Use the *Fill and Stroke* dialog to add a *Gaussian Blur* by dragging the *Blur* slider to a non-zero value. Doing so automatically creates a *Gaussian Blur* filter and attaches it to the text.

The blurred text has the attributes of the original text, in this case, the dark red color. As the blurred text will become the shadow, we'll change the color to black in a moment.

We used the *Fill and Stroke* dialog to rapidly create a filter object for the text. Further work will need to be done with the *Filter Effects* dialog. Open the dialog. Select the new filter (with a name like “filter3362”) by clicking on it. You should see something like:



Add Effect: Blend

The **feBlend** filter primitive provides 4 image blending modes: screen, multiply, darken and lighten.

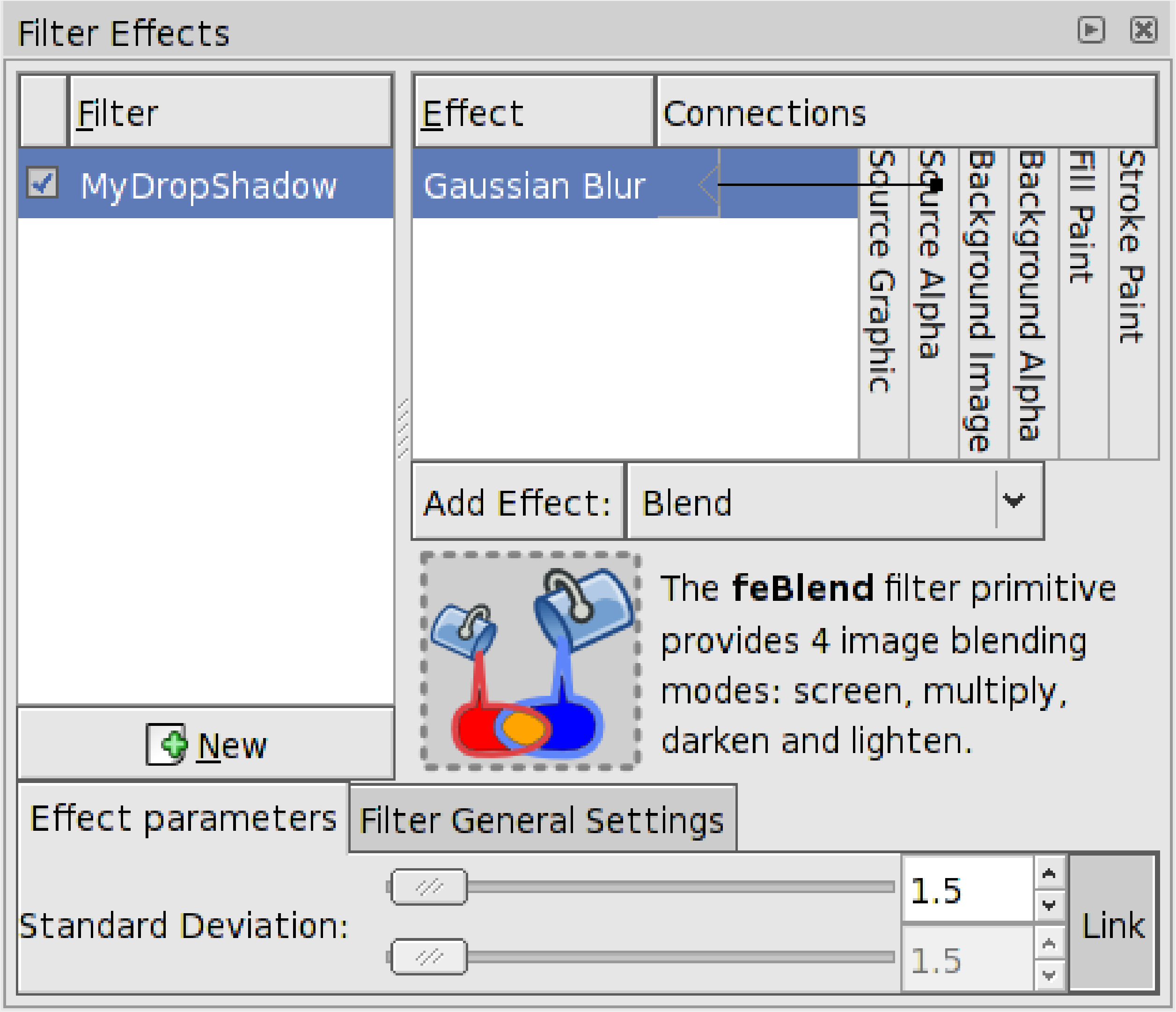
Standard Deviation: 1.5 1.5

Filter Effects dialog after adding *Gaussian Blur* filter.

Let's first give our filter a new name: “MyDropShadow”. Clicking on the filter name when selected allows editing the name.

Our filter consists at the moment of a single *Gaussian Blur*, as shown under the *Effect* column. The line connecting a triangle to the right of “Gaussian Blur” label to a square dot in the *Source Graphics* column shows that the input for the *Gaussian Blur* filter is the original source object (in this case the text). The default height of the dialog is too short to show the complete column labels. The dialog can be stretched so that the labels are completely visible.

To get a black shadow, we'll use *Source Alpha* for the input to the Gaussian Blur primitive. To make the link, left-drag the mouse from the triangle next to the *Gaussian Blur* label to the *Source Alpha* column. The *Filter Effects* dialog should look like:



Filter Effects

Filter

☑ MyDropShadow

New

Effect | Connections

Gaussian Blur

Add Effect: Blend

The **feBlend** filter primitive provides 4 image blending modes: screen, multiply, darken and lighten.

Effect parameters | Filter General Settings

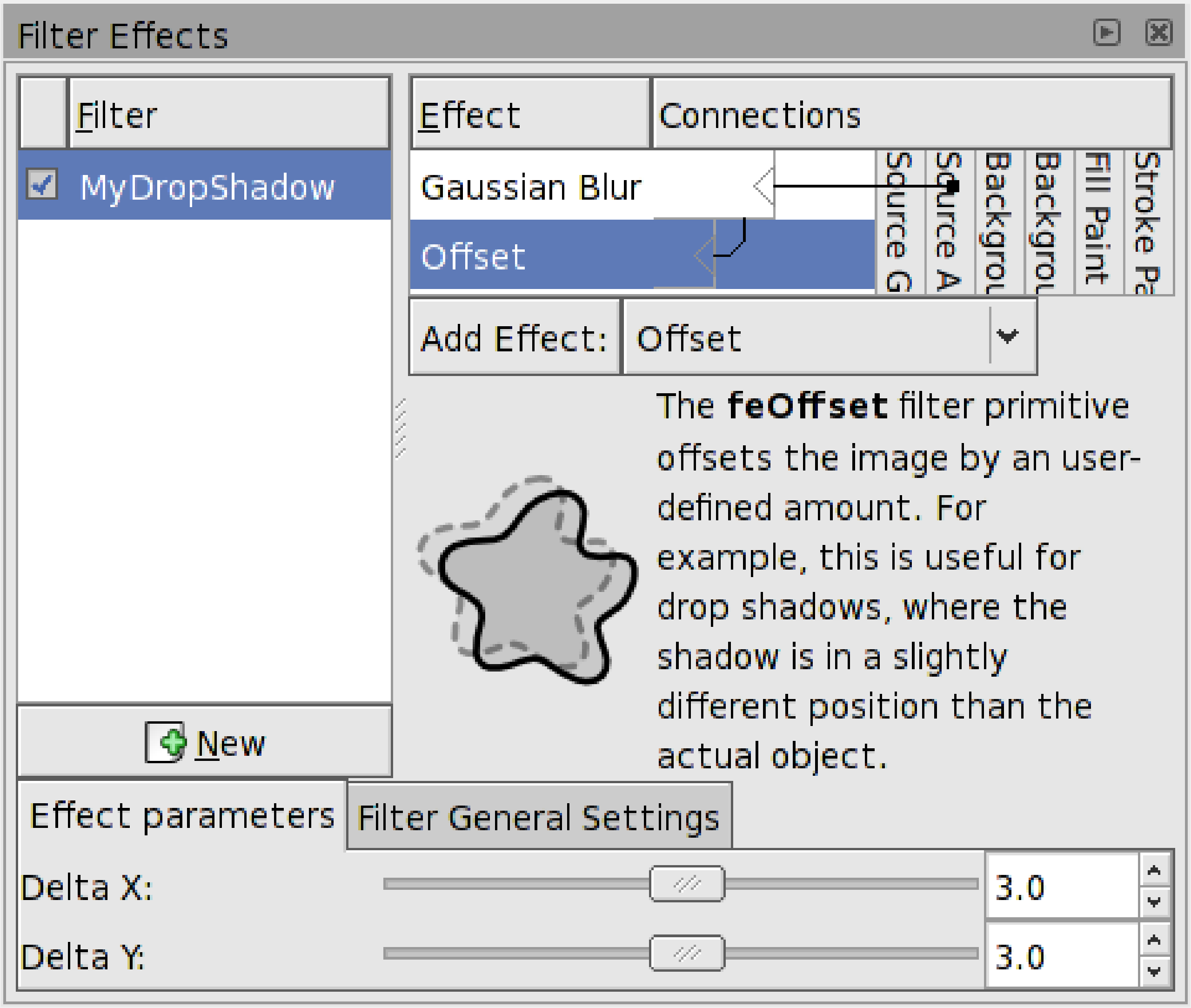
Standard Deviation: 1.5 1.5 Link

Filter Effects dialog after changing the input of the *Gaussian Blur* filter to *Source Alpha*.

The amount of blurring can be changed by using the *Standard Deviation* sliders at the bottom of *Filter Effects* dialog (when the Gaussian Blur primitive is selected). There are two sliders: the top for *x* and the bottom for *y*. By default, the sliders are linked together so the blur is the same in both directions. Note that the blur amount (*Standard Deviations*) is defined in terms of pixels in this dialog while it is defined in terms of a percentage of 1/8 of the perimeter of the bounding box in the Fill and Stroke dialog.

3. **Add Offset**

The shadow should be shifted relative to the text. This can be achieved through the use of the *Offset* primitive. To add such the primitive, select it from the drop-down menu next to the *Add Effect:* button and then click on the button to make the addition. Note the bent line connecting the triangle to the right of the “Offset” entry under the *Effect* column to the “Gaussian Blur” entry above. This (the default) indicates that the *Offset* primitive is using the output of the *Gaussian Blur* primitive as its input. The amount of the offset can be changed by the *Delta X* and *Delta Y* sliders at the bottom of the *Filter Effects* dialog when the *Offset* label is highlighted. Set the offsets both to 3 (pixels). You should see the shadow text shift as the sliders are dragged.



Add Effect: Offset

The **feOffset** filter primitive offsets the image by an user-defined amount. For example, this is useful for drop shadows, where the shadow is in a slightly different position than the actual object.

Delta X: 3.0

Delta Y: 3.0

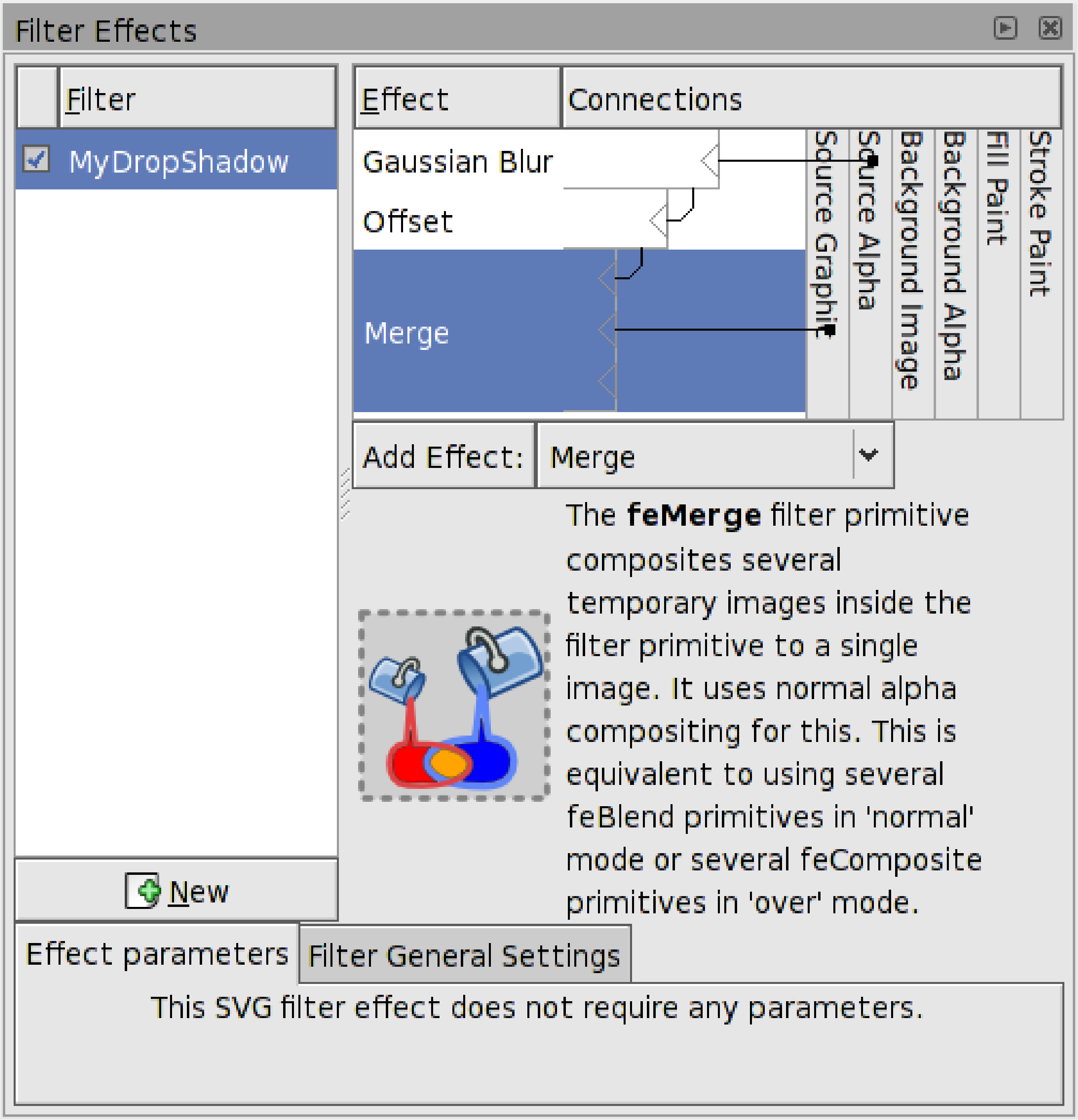
Filter Effects dialog after adding the *Offset* primitive.

4. **Add Merge**

Now that we have the shadow, we need to add back in the original text. This can be done with the *Merge* primitive which merges graphics from multiple inputs. Select the *Merge* primitive in the drop-down menu list of *Filter* primitives. Click on the *Add Effect:* button to add the primitive.

When first created, there are no inputs defined. Left-dragging the mouse from the triangle at the right of the “Merge” label to the “Offset” label above, to create a link from the output of the *Offset* primitive to the input of the *Merge* primitive.

When an input link to a *Merge* primitive is made, a new, empty input node (triangle) is created. To add back in the original text, unmodified, left-drag the mouse from the empty triangle to the “Source Graphic” column.



Add Effect: Merge

The **feMerge** filter primitive composites several temporary images inside the filter primitive to a single image. It uses normal alpha compositing for this. This is equivalent to using several feBlend primitives in 'normal' mode or several feComposite primitives in 'over' mode.

This SVG filter effect does not require any parameters.

The completed drop-shadow filter.

The drop shadow is now complete. You can modify the amount of blur or the amount of offset by selecting the appropriate *Filter* primitive under the *Effects:* column and using the sliders at the bottom of the *Filter Effects* dialog.

Filter Effects Dialog

Table of Contents

Color Filter Primitives

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Filter Effects » Color Filter Primitives

← Index →

Color Filter Primitives

Two *Filter* primitives allow the manipulation of colors.

Color Matrix

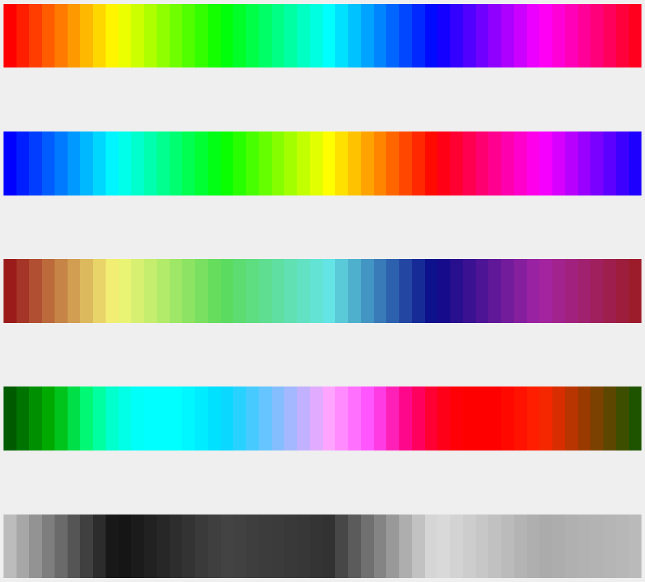
The *Color Matrix* primitive maps each *RGB* and *Alpha* value to a new value. The transformation is described by a 5×5 matrix with the bottom row fixed, thus a general transformation is described by a 5×4 matrix. The fifth column adds a value that is independent of *RGB* or *Alpha*, allowing for nonlinear color correction.

$$\begin{pmatrix} R' \\ G' \\ B' \\ A' \\ 1 \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \\ A \\ 1 \end{pmatrix}$$

Four types of transformations are defined, of which three are special classes of the first.

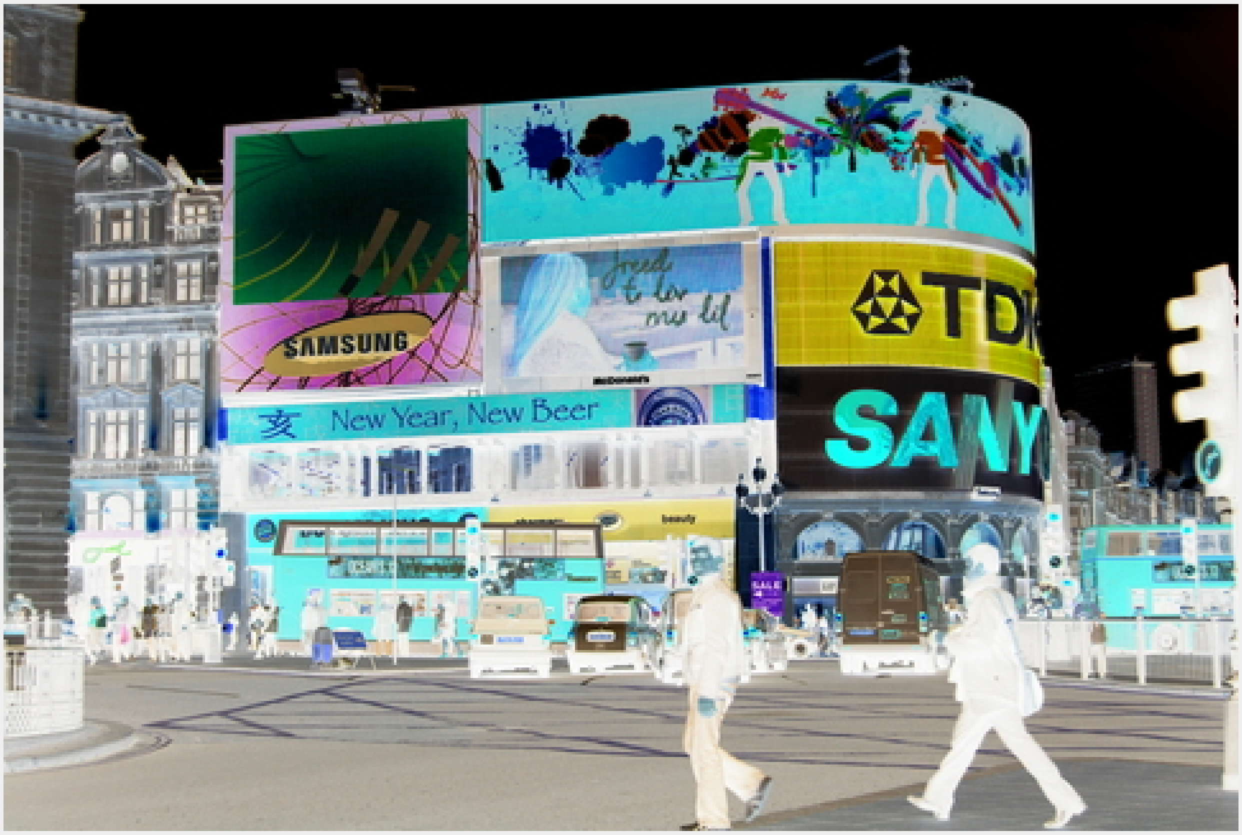
- Matrix: The full 5×4 matrix is defined. This is the most general case.

- Saturate: The saturation is reduced by specifying one number, s. The range of s is 0.0 (completely desaturated) to 1.0 (unchanged). Only the RGB values are changed. The exact formula is: $R' = (0.213+0.787s)R + 0.715\times(1-s)G + 0.072\times(1-s)B$; $G' = 0.213\times(1-s)R + (0.715+0.285s)G + 0.072\times(1-s)B$; $B' = 0.213\times(1-s)R + 0.715\times(1-s)G + (0.072+0.928s)B$.
- Hue Rotate: The hue is shifted by specifying one number. Like the *Saturate* case, only RGB values are changed. The exact formula is quite complicated. It is not just a red to yellow to green etc. rotation.
- Luminance to Alpha: The luminance is converted to *Alpha* via a fixed formula: *Alpha*= 0.2125 × R + 0.7154 × G + 0.0721 × B (from ITU-R Recommendation BT709, the HDTV color standard).



Examples of using the *Color Matrix* primitive. From top to bottom: Source object. *Matrix* mode set to swap red and blue. *Saturate* mode with input of 0.5. *Hue Rotate* mode with input of 90°. *Luminance to Alpha* mode.

A “negative” can be made by setting the RGB diagonal matrix elements (a_{00}, a_{11}, a_{22}) to -1.00 and the top three elements of the fifth column (a_{04}, a_{14}, a_{24}) to 1.00.



Creation of a “negative” using the *Matrix* mode.

Component Transfer

Partially implemented, No user interface.

The *Component Transfer* primitive changes the *RGB* and *Alpha* of an object by applying independent functions to each of the *RGB* and *Alpha* input values. The following modes for defining the functions are available:

- Identity.
- Table.
- Discrete.
- Linear.
- Gamma.

Mini Tutorial - A Drop Shadow
Table of Contents
Compositing Filter Primitives

Get the book.

Compositing Filter Primitives

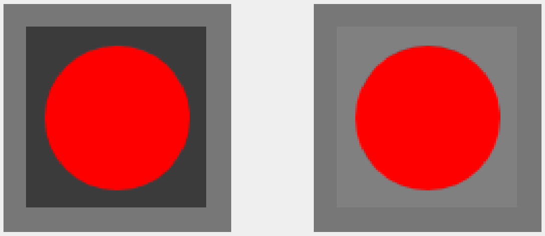
These primitives composite two or more graphics. The graphics may be from an object, a background, or the output of another primitive.

Note

Inkscape v0.46 has a problem in using one of these filters. When using either *Background Image* or *Background Alpha* as an input to the filter, the “enabled-background” tag must be added to the *SVG* file (this tells *SVG* renderers to keep a copy of the background in memory). This is not done. A work-around is to use the *Layers* dialog to add a *Blend* filter to a *Layer*. The *Layer* blend can then be removed, leaving the necessary tag in place.

Note

The *SVG* 1.1 specification has a problem when an object is composited with a background that is not fully opaque. The background is included twice (once with the composited image and once as a background). There are three ways to deal with this problem. The first is to avoid using a *Background Image* or *Background Alpha* as a filter input. The second is to replace a transparent background with a solid background (you can use the *Dropper Tool* to replace a transparent *Fill* with an equivalent solid *Fill* [turn off “Pick alpha” in the *Tool Controls*]). The third is to use the *Flood* filter to create a solid white background and include this as the first input to a *Merge* filter (if using a *Merge* filter, include the flood first; if using a *Blend* or *Composite* filter, add a *Merge* filter with the first input being the output from the *Flood* filter and the second input being the output from the *Blend* or *Composite* filter). This solution runs into trouble when it is desired that the overall image have transparency. The *SVG* 1.2 standard corrects this deficiency.



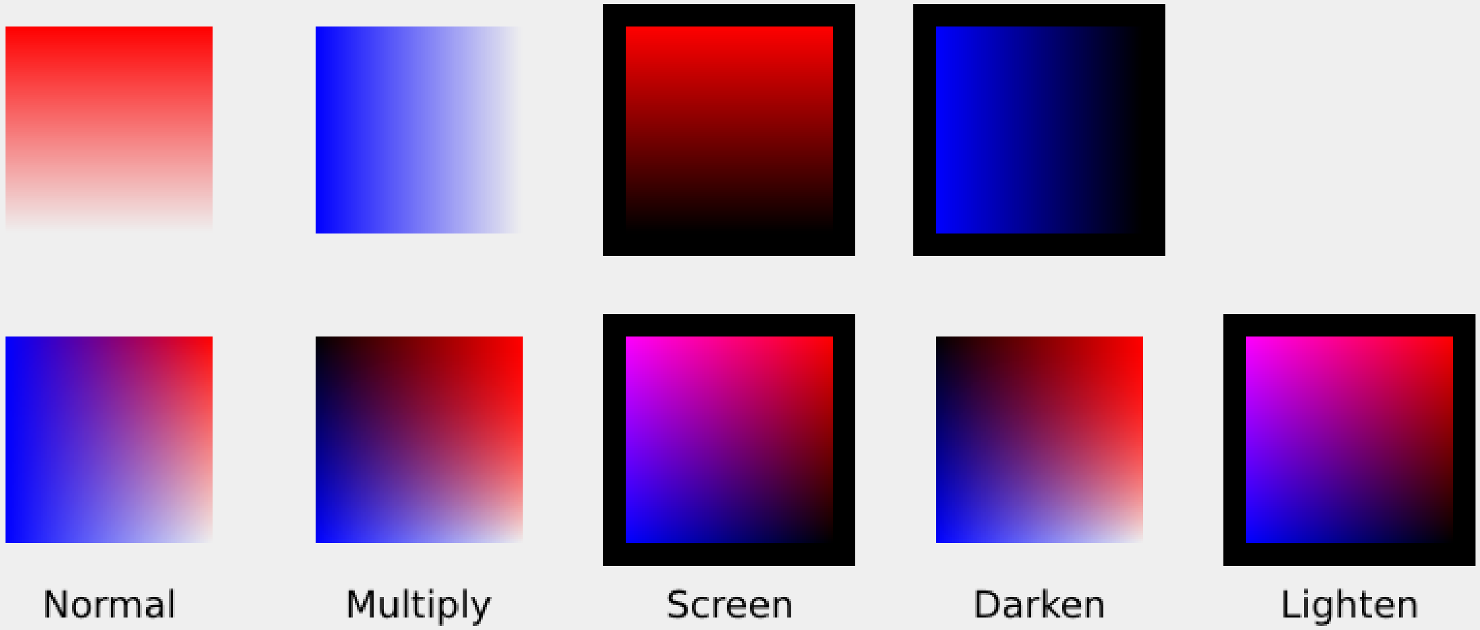
The red circle is combined with the background using the *Merge* filter. The background gray squares have transparency of 50%. On the left, the area within the filter region is too dark, a result of the background being added in twice. On the right, the filter region was first filled with white using the *Flood* filter.

Blend

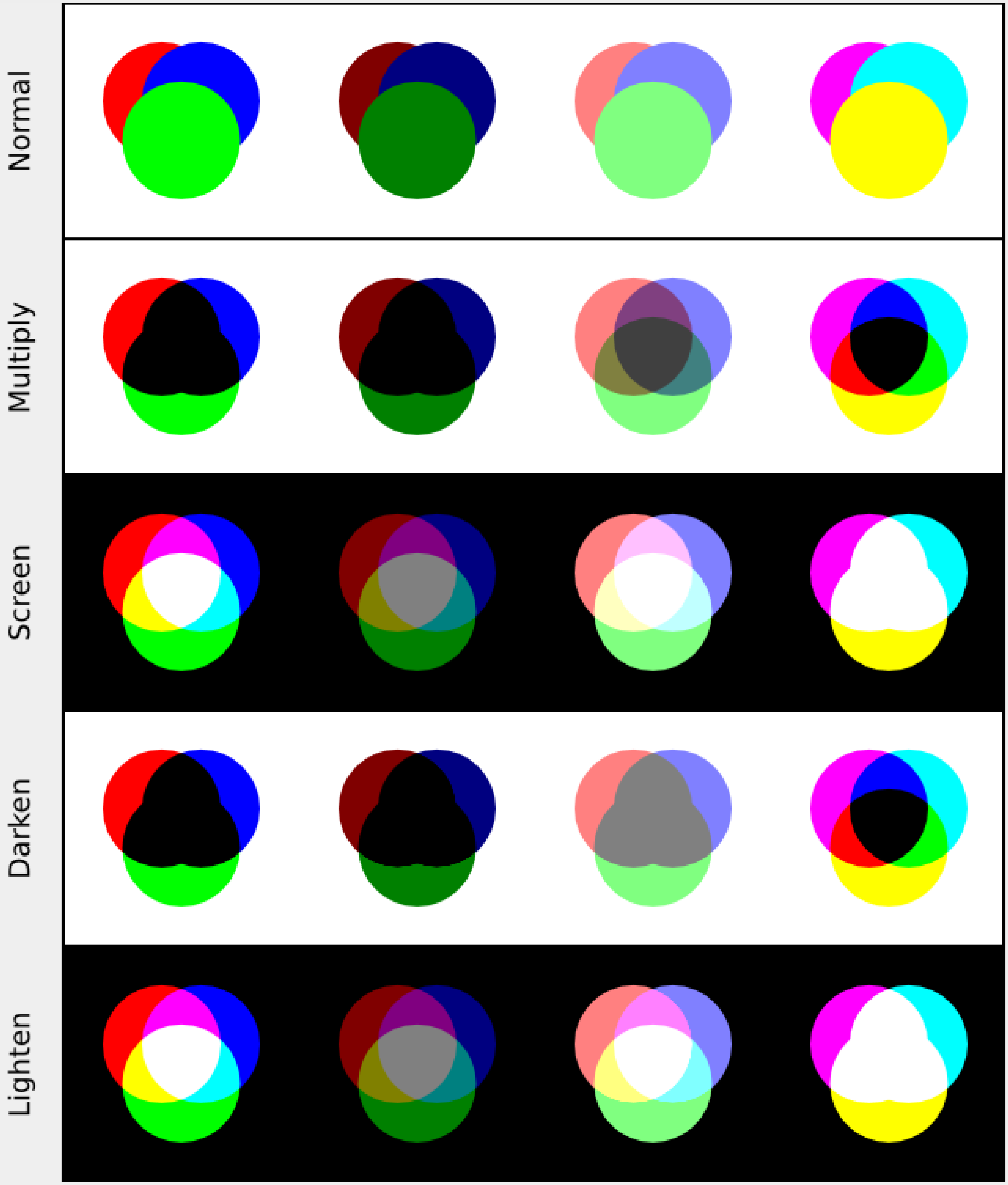
The *Blend* primitive blends two overlapping objects or an object with its background by doing a pixel-by-pixel combination using one of five defined blend modes. The five modes are listed below. Except for the *Normal* mode, the result is independent of which object is on top.

For each mode, the mathematical definition is given. In the definitions, *a* corresponds to an object on top of *b*. *c* is the RGB color of the object while *q* is the opacity. Both *c* and *q* range from 0 to 1. Each of the RGB colors is combined independently.

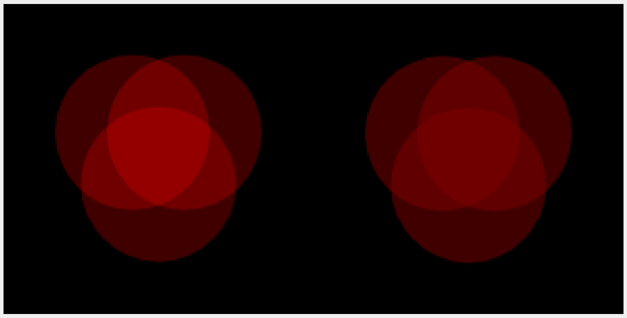
- Normal: Top object in front of bottom object as if filter not present. (In fact Inkscape removes the *Blend* filter primitive when “Normal” is selected for a blend added to a *Layer* with the *Layers* dialog.) $cr = (1-qa)\times cb + ca)$.
- Multiply: Top object filters light from bottom object/image. (Like looking through a transparency with color of top object at bottom object.) Examples: Blue object over red object yields black since the blue object filters out all the red light. Cyan object over purple object yields blue since the cyan filters out the red but allows the blue to pass. $cr = (1-qa)\times cb + (1-qb)\times ca + ca\times cb$.
- Screen: Top object adds light to bottom object. (As if both top and bottom objects are projected independently onto a screen.) Examples: Blue object over red object yields purple. Cyan object over purple object yields white since cyan contains green and blue and purple contains red and blue. Thus red, green, and blue are present in equal amounts. (Why doesn't the result contain more blue? Because both cyan and purple already contain the maximum amount of blue.) $cr = cb + ca - ca\times cb$.
- Darken: Top object darkens bottom object. $cr = Minimum\ (\ (1-qa)\times cb + ca,\ (1-qb)\times ca + cb\ )$.
- Lighten: Top object lightens bottom object. $cr = Maximum\ (\ (1-qa)\times cb + ca,\ (1-qb)\times ca + cb\ )$.



Top: The blue and red squares contain linear *Gradients* that range from full *opacity* to full *transparency* over a white or black background. Bottom: Blue squares overlaying red squares with different *Blend modes*.



Three circles on top of each other. The same *Blend* has been applied individually to each circle in a set. Note that the circles are on either a white or black background with maximum *opacity*.



An illustration of the difference between the *Screen* (left) and *Lighten* (right) *Blend modes*. Each circle has a red value of 128 (50%) and an *opacity* of 128 (50%). If the *opacity* was set to 255 (100%), the two figures would be identical.

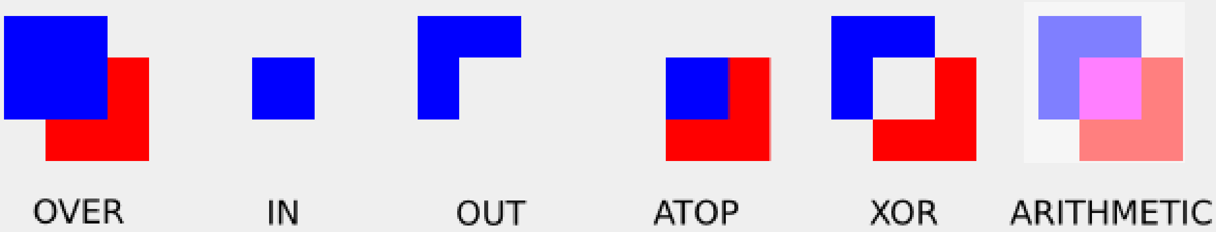
The *Blend* filter primitive can also be applied to a *Layer* through the *Layers* dialog. In this case, input one is the selected object while input two is set to the *Background Image*.

Composite

The *Composite* filter primitive allows two overlapping objects or an object and background to be merged pixel-by-pixel according to a mode-dependent rule. See the introduction to this section for problems when using a background as one of the inputs.

The possible modes are:

- Over: The upper object is placed over the lower object. This is equivalent to the normal way overlapping objects are drawn.
- In: The bottom object determines which part of the top object is visible.
- Out: The bottom object determines which part of the top object is hidden.
- Atop: The bottom object determines what part of the top object is visible. The bottom object is also visible.
- XOR: The non-overlapping regions of the top and bottom objects are visible.
- Arithmetic. The inputs *K1*, *K2*, *K3*, and *K4* are used in the equation: result = $K1 \times i_1 \times i_2 + K2 \times i_1 + K3 \times i_2 + K4$ to determine the output; i_1 and i_2 are the input values of the two source objects.



Examples of the different modes available in the *Composite* filter primitive. The first input is the blue square. The second input, the red square, is derived from the blue square using the *Color Matrix* and *Offset* filter primitives. The parameters for the *Arithmetic* mode are all set to 0.5.

Merge

The *Merge* filter allows the combining of two or more objects or outputs of filter primitives. It works by layering one image on top of another, much as regular objects are layered on top of each other in *z-order*, or, for the case of two inputs, as the *Composite* filter primitive using the “Over” mode.

When the *Merge* filter is added to a complex filter an unassigned input node is created. As each input is assigned, another empty input node is created. This empty node is not included in the *SVG* tree structure.

Color Filter Primitives

Table of Contents

Fill Filter Primitives

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Filter Effects » Fill Filter Primitives

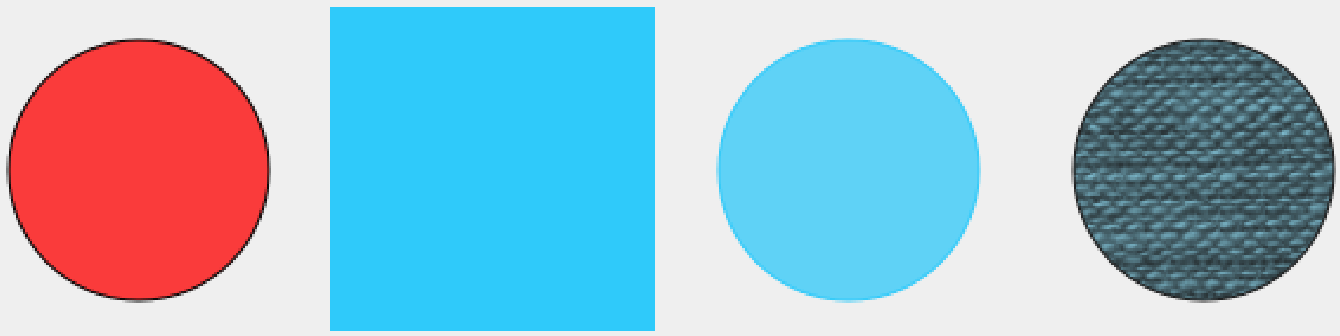
← Index →

Fill Filter Primitives

These primitives provide some type of *Fill* for the filter region.

Flood

The *Flood* primitive fills the *Filter Effects Region* with a specified color and *opacity*. This filter primitive is most useful when combined with other filters primitives.



An example of using the *Flood* filter. From left to right: 1. A simple circle. 2. After adding the *Flood* filter with a light blue color. Note that the default *Filter Effects Region* is larger than the bounding box of the circle. 3. After the addition of a *Composite* filter primitive using the *In* operator. The first input is the output of the *Flood* fitter and the second is the *Source Graphic*. 4. The *Fill* of the circle has been changed to the *Cloth* pattern (with the *Fill and Stroke* dialog) and the *Composite* type has been changed to *Arithmetic* with K1=0.5, K2=0.0, K3=0.5, and K4=0.0.

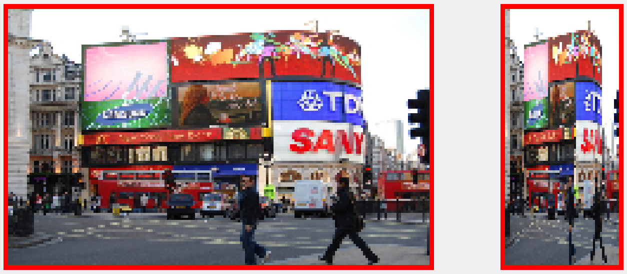
Image

Partially Implemented.

The *Image* primitive renders an external graphics file or an internal *SVG* object. It allows more than one object to be referenced in a complex filter (the first being the object attached to the filter).

Unfortunately, this very useful filter primitive is not yet fully implemented in Inkscape with only external images supported. The *GUI* will create a reference with an absolute path to the external image. Use the *XML Editor* to change an absolute path to a relative one if required.

By default, the image is shrunk or stretched to fit inside the *bounding box* of the object to which the filter is attached.

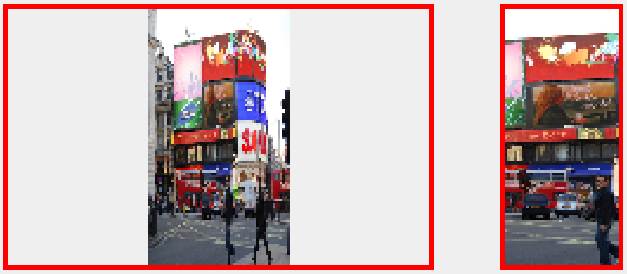


An image applied to two objects with the *Image* filter primitive. The red rectangles show the *bounding box* of the objects.

The placement of the image within the *bounding box* can be controlled using the parameters *x*, *y*, *width*, and *height*. The coordinate system is the same as used for the object. The *XML Editor* must be used to modify these parameters. An additional parameter, *preserveAspectRatio*, is not supported by Inkscape.

Note

Both Firefox and Opera render the image differently than Inkscape. They use the default *Filter Effects Region* to determine the image placement (even if set to a different value). They also, by default, preserve the aspect ratio of the image.



An image applied to two objects with the *Image* filter primitive specifying different image regions. On the left, *x* is 60 px and *width* 60 px. On the right, *x* is -60 px and *width* 180 px.

Tile

Not implemented.

The *Tile* primitive fills a rectangular region with a repeated input image.

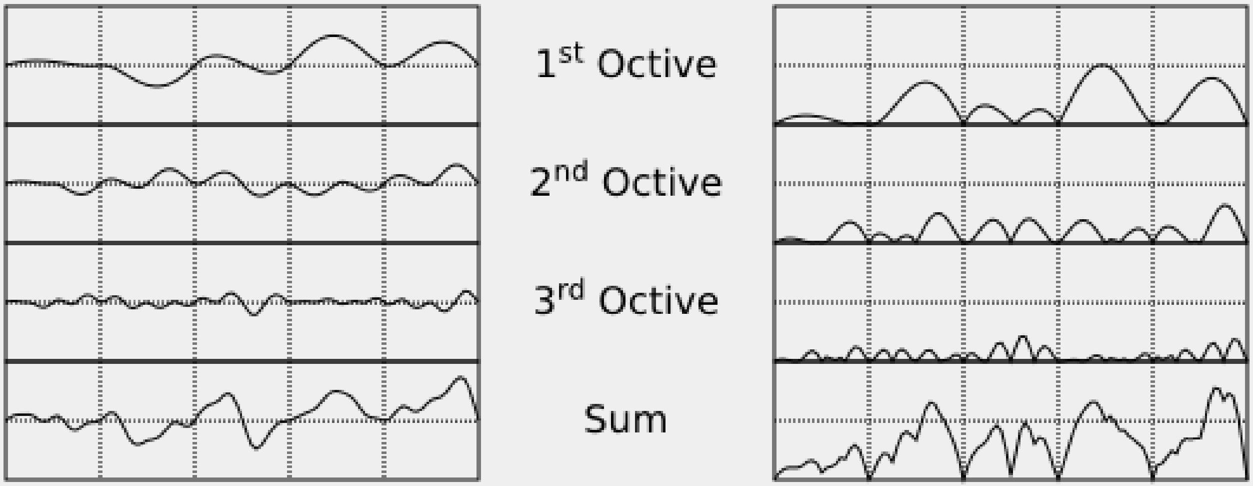
No options.

Turbulence

The *Turbulence* primitive allows the creation of artificial textures such as marble surfaces or clouds. It is based on the work of Ken Perlin who won an Academy Award for creating realistic textures with computers. Perlin solved the problem of using random numbers to produce *smooth* random fluctuations in color. A simplified version of his method is to generate random intensities at given intervals and then connect these points smoothly together.

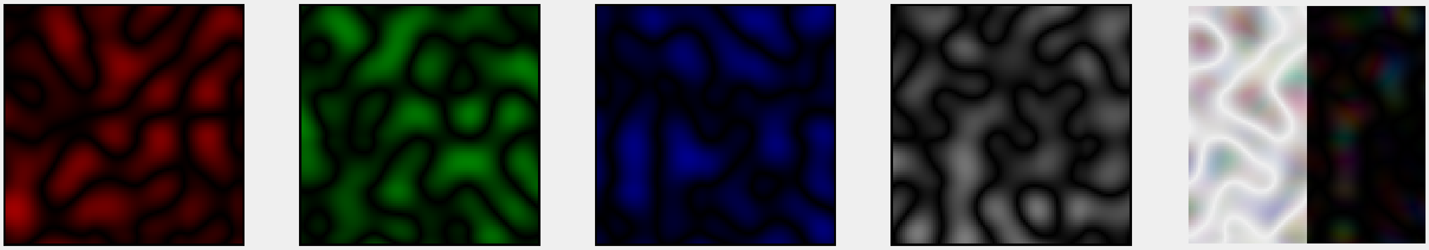
Note that both Firefox 3 and Batik render this filter differently. Opera 9.26 renders it the same as Inkscape.

The following figure gives an idea of how the filter would be implemented in one-dimension:



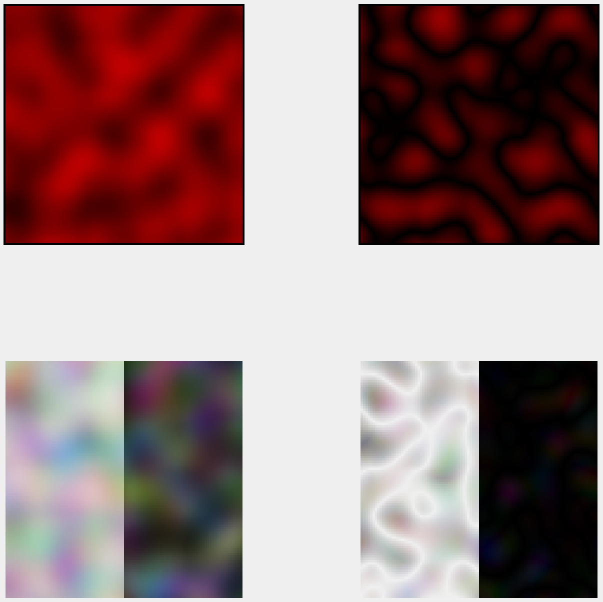
A one-dimension illustration of the Perlin method of generating noise. *Fractal Noise* is shown on the left and *Turbulence* on the right. The *Base Frequency* is set to 0.05 giving a “wave length” of 20 pixels (indicated by the vertical dotted lines). The contributions from three *Octaves* are shown individually and then summed.

The *RGB* and *Alpha* components are each derived separately.



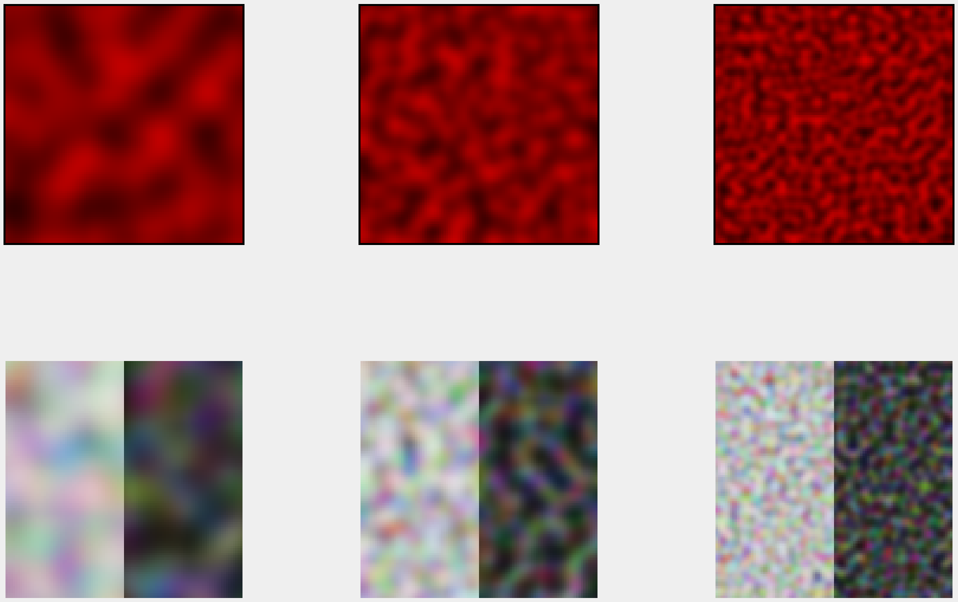
The components of a *Fractal NoiseTurbulence* filter. From left to right: Red. Green. Blue. Alpha. Combined, there is a black rectangle behind the right half. The Color Matrix primitive was used to extract the individual components.

The *Type* can either be *Fractal Noise* or *Turbulence*. *Turbulence* tends to have more dramatic dark regions as the absolute values of the noise terms are used, creating “visual cusps”.[13]



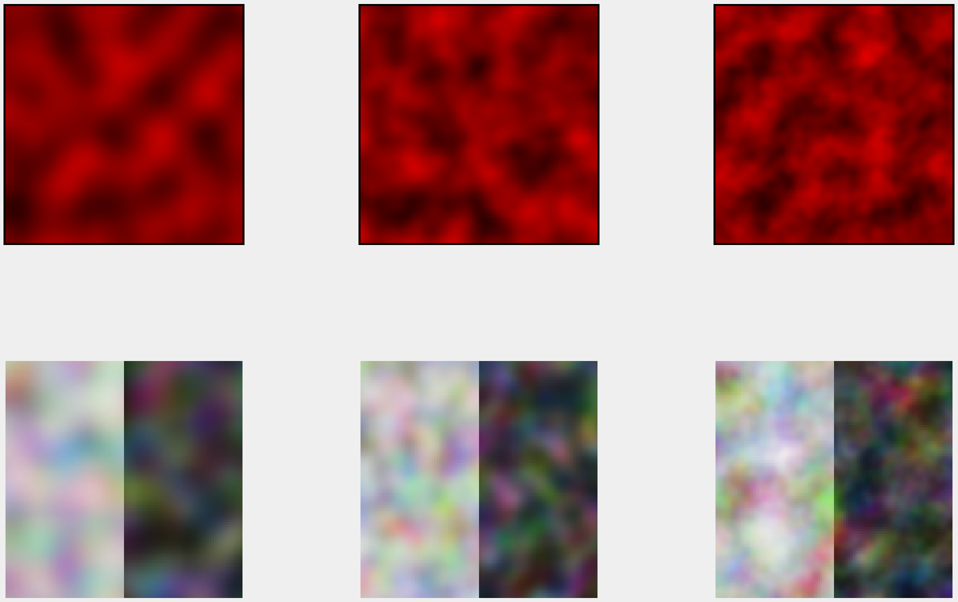
The two types of noise available: Left: *Fractal Noise*. Right: *Turbulence*. Top: Only red channel. Bottom: All channels. There are black rectangles behind the right halves.

The *Base Frequency* is defined by default in terms of inverse *Screen pixels*. For example, a frequency of 0.1 would have a “wave length” of 10 pixels. Inkscape cannot yet create resolution independent noise. Reasonable values for this attribute range from 0.01 (a slowly varying texture) to 0.2 (a rapidly varying texture).



Fractal Noise as a function of *Base Frequency*. From left to right: 0.05. 0.10. 0.20. Top: Only red channel. Bottom: All channels, a black rectangle has been placed behind the right halves.

The number of *Octaves* determines the complexity of the noise, the more *Octaves*, the more complex. Each *Octave* adds a term with twice the *Frequency* but half the amplitude. Using more than four or five *Octaves* isn't so useful as the spatial and color variations become too small to be seen (and increases the CPU load).



Fractal Noise as a function of *Octaves*. All with *Base Frequency* of 0.05. From left to right: 1 octave. 2 octaves. 3 octaves. Top: Only red channel. Bottom: All channels, a black rectangle has been placed behind the right halves.

The *Turbulence* primitive uses a pseudo-random number generator. In principle, renderings with different *SVG* viewers should use the same generator and produce the same textures (at the same scale). If you use the filter twice on two identical objects, the textures should be the same. Changing the seed will force the random-number sequence to be different and thus the textures will be different too.

[13] See http://www.noisemachine.com/talk1/22.html.

←
Compositing Filter Primitives

↑
Table of Contents

→
Lighting Filters Primitives

Get the book.

Inkscape: Guide to a Vector Drawing Program

Get the book.

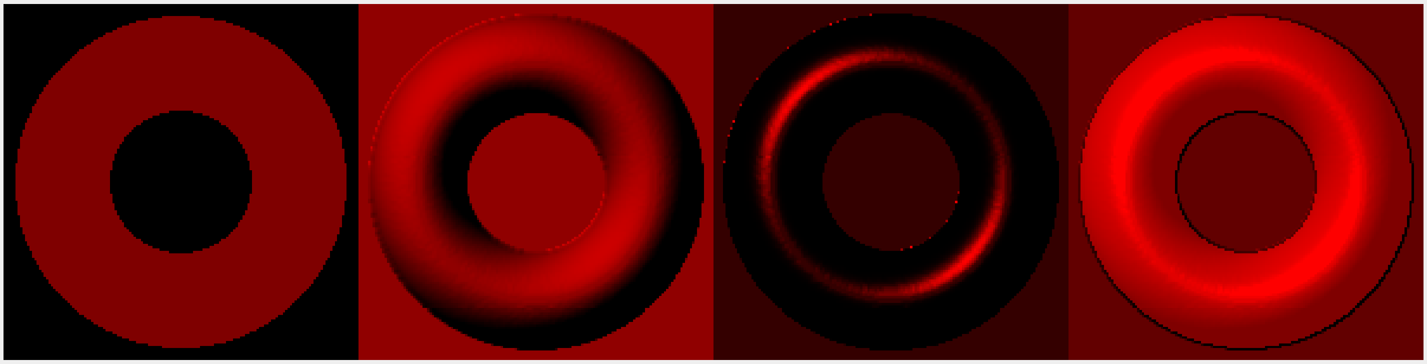
Inkscape » Filter Effects » Lighting Filters Primitives

Index

Lighting Filters Primitives

Two primitives, *Diffuse Lighting* and *Specular Lighting*, are included to simulate light shining on objects. They represent two of the three parts of the Phong reflection model for modeling light in computer graphics. The three parts of the model are:

- Ambient light: The light present everywhere in a scene. In *SVG* this would be represented by a solid *Fill*.
- Diffuse light: The reflection of a light source off a surface that does not depend on the location of the observer. The intensity is a function of the angle between a line connecting the surface to a light source and a line normal to the surface.
- Specular light: The reflection of a light source off a surface that does depend on the location of the observer. For example, the highlights seen on shiny surfaces. The intensity is maximum when the angle between the normal to a surface and the light source is equal to the angle between the normal and the viewer.



An illustration of the components of the Phong model. From left to right: Ambient light. Diffuse light. Specular light. All light sources combined. The diffuse and specular light source is the same, a red distant light coming from the upper left. The specular image has had a black background added to set off the specular light.

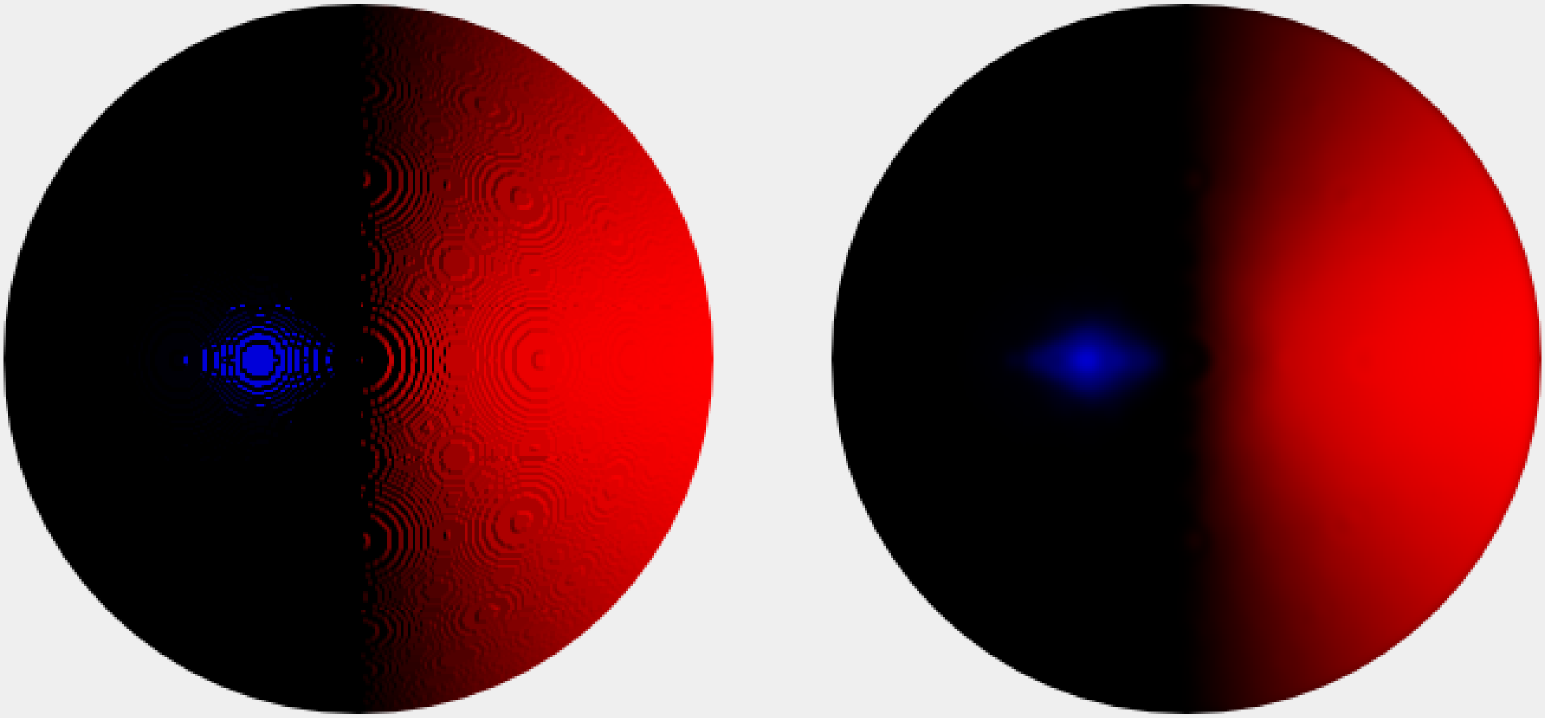
The Phong model does not take into account shadows that would be caused by one area of an object on another, i.e., it does not do ray tracing.

The contour of an object in the *z* (out of the drawing) direction is described by a bump map that is defined by the Alpha channel of an object. The values of the pixel and the neighboring pixels in the *bump map* define the normal to the surface for the pixel.

The two lighting filters share in common most of their attributes such as the type of light source, its color, and its position; thus we'll discuss them together.

- *Diffuse Color*, *Specular Color*: The color of the light source.
- *Surface Scale*: Sets the scale of the surface in order to calculate normals to the surface. The number represents the maximum height (corresponding to Alpha=1.0) of the surface in the same units as *x* and *y*.
- *Constant*: Diffuse or Specular Reflection Constant: How much of the light that hits the surface is reflected diffusely or specularly. A value greater than 1.0 oversaturates the object.
- *Exponent* (Specular Lighting only): Determines how sharp or narrow the specular reflections are. A value of 1.0 (the minimum) suggests a dull surface with wide reflections; as the value increases, the surface appears more polished with narrower reflections.
- *Kernel Unit Length* (Unused): Sets the pixel size used to calculate the reflections (screen pixels are used by default).
- *Light Source*: One of three types of light: *Distant Light*, *Point Light*, or *Spot Light* (see below).

When applying a lighting filter with a large *Surface Scale*, the limited resolution of the bump map may create artifacts. These can be removed by applying a small amount of Gaussian blur to the image.

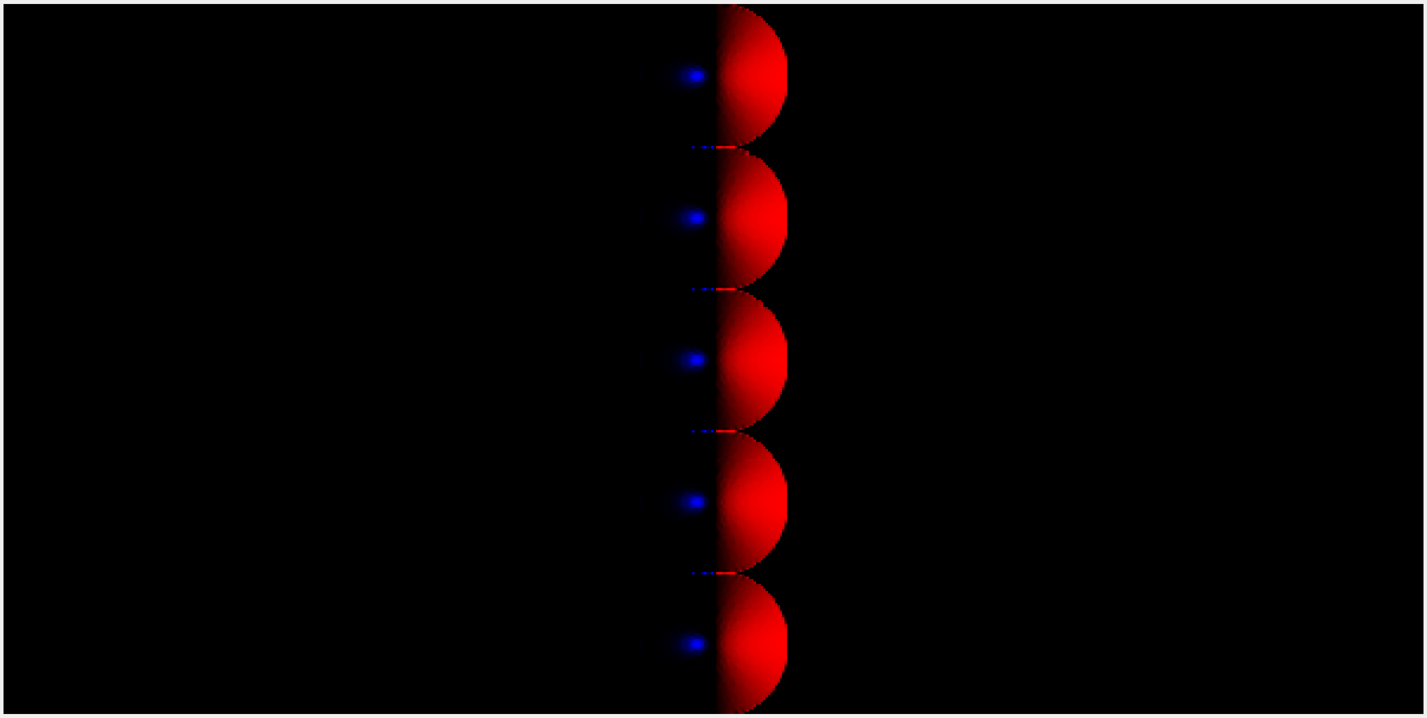


The limited resolution of the bump map has resulted in artifacts on the sphere on the left. A small amount of blurring has removed the artifacts on the sphere on the right (the sphere has also been clipped).

Distant Light Source

This light source simulates a light at a large (infinite) distance from the illuminated object. The required attributes are:

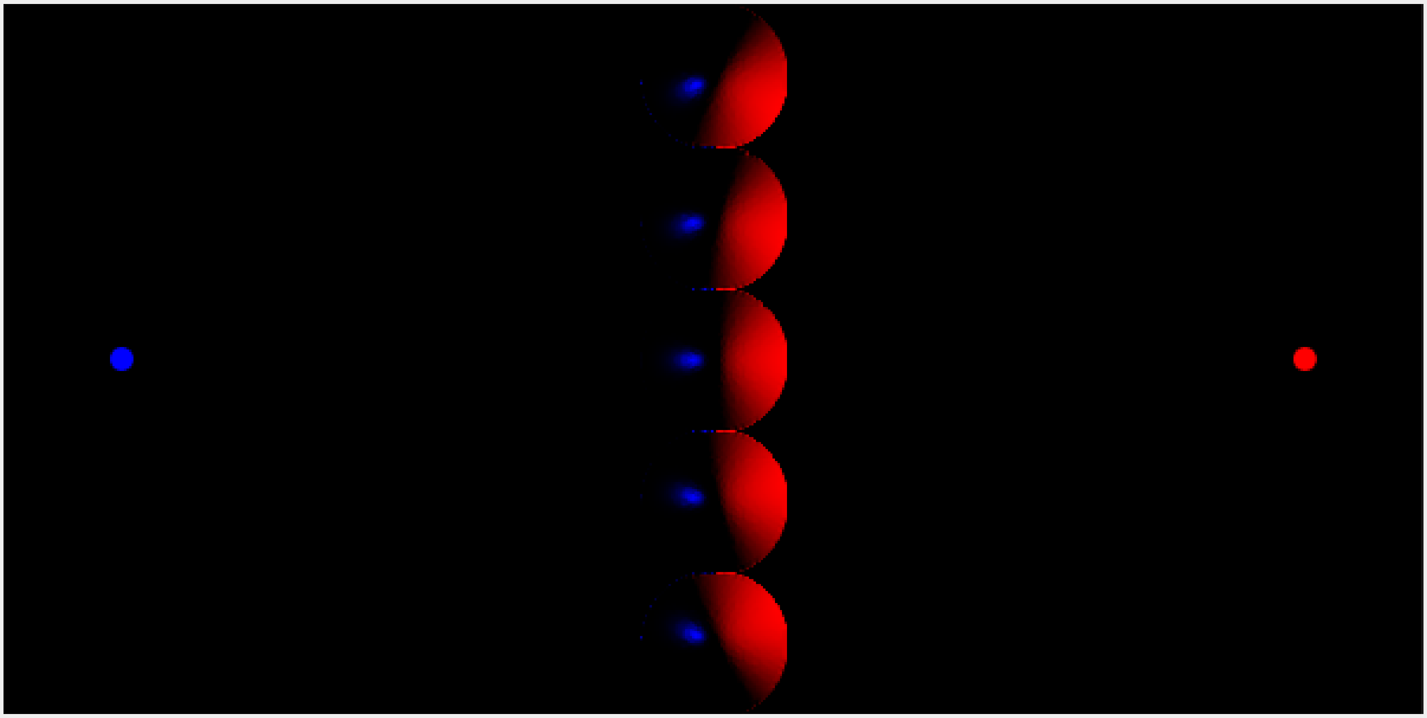
- *Azimuth*: The direction (angle) of the light source in the drawing plane. The angle is defined in degrees from the horizontal (*x*) axis in the clockwise direction. Note that this does not match either angle definition used by Inkscape (see the section called “Transformations”). Recall that *SVG* uses a left-handed coordinate system where the positive *y* direction is down.
- *Elevation*: The direction (angle) of the light source above the drawing plane (in degrees).



Five spheres illuminated by *Distant Light* sources. The light from the left is a *Specular* reflection while that on the right is a *Diffuse* reflection.

Point Light Source

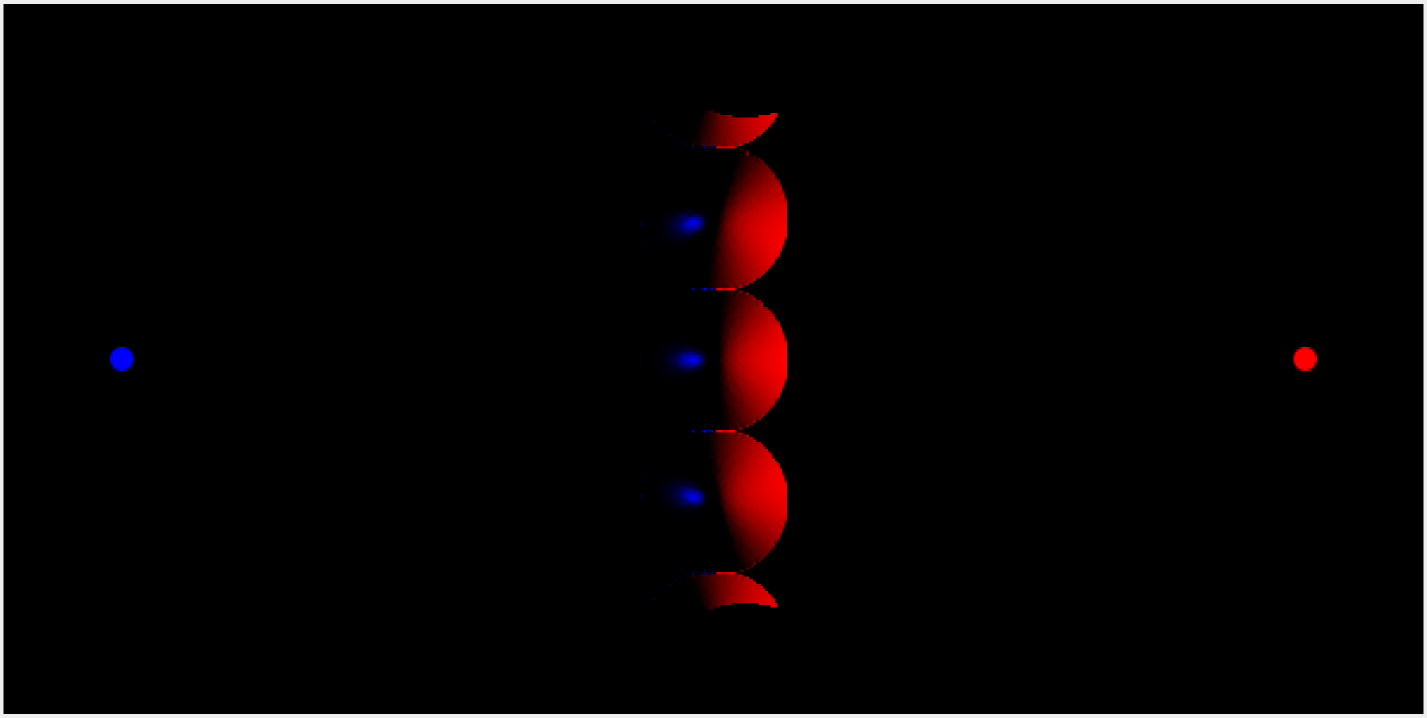
This light source simulates a point light source near an illuminated object. One triple set of numbers (*x*, *y*, *z*) is required to set the *Location* of the light. The units are in the coordinate system of the lit object. Note: *z* represents the distance out of the plane (toward the viewer of the *SVG* drawing) if *x* or *y* are not inverted.



Five spheres illuminated by *Point Light* sources. The light from the left is a *Specular* reflection while that on the right is a *Diffuse* reflection. The small red and blue circles indicate the positions of the light sources.

Spot Light Source

This light source simulates a point light source near an illuminated object but with a limited cone of light. One triple set of numbers (*x*, *y*, *z*) is required to set the *Location* of the light and another to set the direction the center of the cone points (*Points At*). See above for the definition of the coordinate system. The *Specular Exponent* sets how well focused is the light; the higher the value, the more sharply focused the light. The *Cone Angle* (degrees) defines the maximum angle for the light. The lights are directed at each other.



Five spheres illuminated by *Spot Light* sources. The light from the left is a *Specular* reflection while that on the right is a *Diffuse* reflection. The small red and blue circles indicate the positions of the light sources. The cone angle is 25°.

Fill Filter Primitives

Table of Contents

Pixel Manipulation Filter Primitives

Get the book.

Pixel Manipulation Filter Primitives

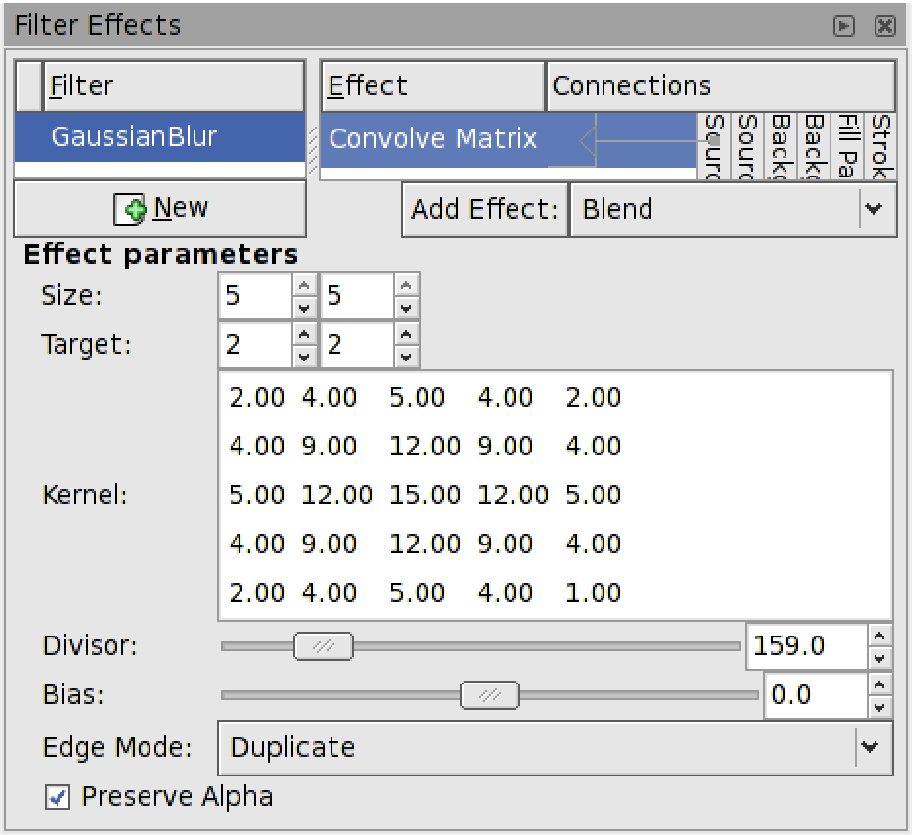
These primitives move pixels or blend adjacent pixels.

Convolve Matrix

The *Convolve Matrix* primitive uses neighboring pixels to modify the color of a pixel. How the pixel is changed is determined by an *N*×*M* matrix with one entry for each neighboring pixel.

$$result = \left(\sum_{i=0}^{i<N} \sum_{j=0}^{j<M} source_{i,j} \times k_{i,j} \right) / divisor + bias$$

The following is an example of a “Gaussian Blur” that uses a 5×5 matrix around the center pixel. The *Kernel* is a integer representation of a 2-dimension Gaussian with a standard deviation of 1.4 pixels. It is normalized by the *Divisor*.[14]



Filter: GaussianBlur

Effect: Convolve Matrix

Add Effect: Blend

Effect parameters

Size: 5 5

Target: 2 2

Kernel:

2.00	4.00	5.00	4.00	2.00
4.00	9.00	12.00	9.00	4.00
5.00	12.00	15.00	12.00	5.00
4.00	9.00	12.00	9.00	4.00
2.00	4.00	5.00	4.00	1.00

Divisor: 159.0

Bias: 0.0

Edge Mode: Duplicate

☑ Preserve Alpha

Filter Effects dialog after defining a *Gaussian Blur* using the *Convolve Matrix* filter primitive.

The above *Gaussian Blur- Convolve Matrix* filter is applied to a photograph.

The parameters for the effect are:

- *Size*: Size of matrix (*x*×*y*).
- *Target*: Which matrix element corresponds to the target pixel, default is to center *Kernel* on target pixel.
- *Kernel*: The matrix.
- *Divisor*: Scale factor after *Kernel* applied.
- *Bias*: Value added after *Kernel* applied, default 0.
- *Edge Mode*: The method that the input image is extended so that pixels at the edge can be evaluated. The options are: *Duplicate* (the edge pixels are duplicated), *Wrap* (the pixels are taken from the opposite side of the input image), *none* (the extended pixels are given RGB and Alpha values of zero). At the moment Inkscape does not use this parameter despite it being in the user interface.
- *Preserve Alpha*: If box is checked, Alpha will be copied directly from the input graphics. Otherwise, it will be calculated just like RGB.

The *Convolve Matrix* primitive is necessarily linked to evaluating pixels. By default, the pixel size is that of the display. This means that the resulting image is not resolution independent. The SVG standard provides ways to avoid this through the *filterRes* and *kernelUnitLength* attributes but Inkscape does not yet support them.

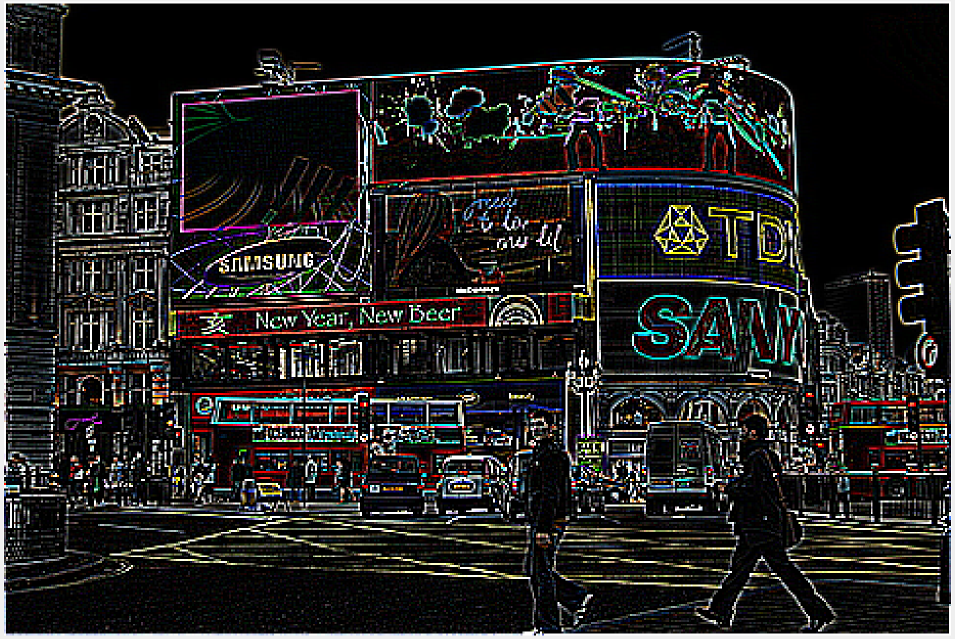
Examples

Edge Detection

Kernel:

$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

Divisor = 1.0, *Bias* = 0.0, *Preserve Alpha* selected.



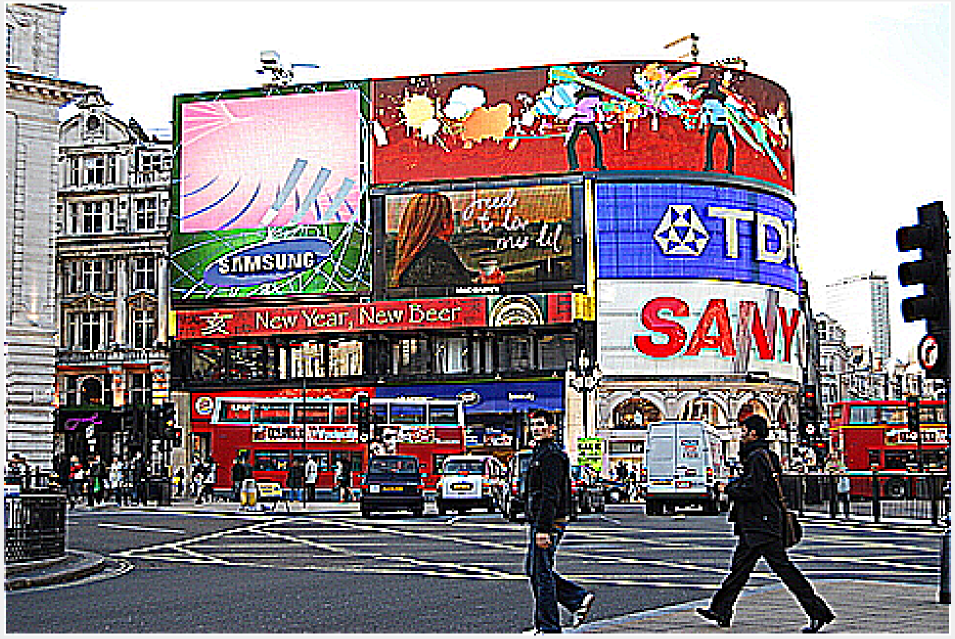
Edge detection.

Sharpen

Kernel:

$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

Divisor = 1.0, *Bias* = 0.0.



Sharpen.

Unsharpen

The above “Sharpen” filter is a bit extreme. This “Unsharp” filter is a bit more subtle.

Kernel:

$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 17 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

Divisor = 9.0, *Bias* = 0.0.

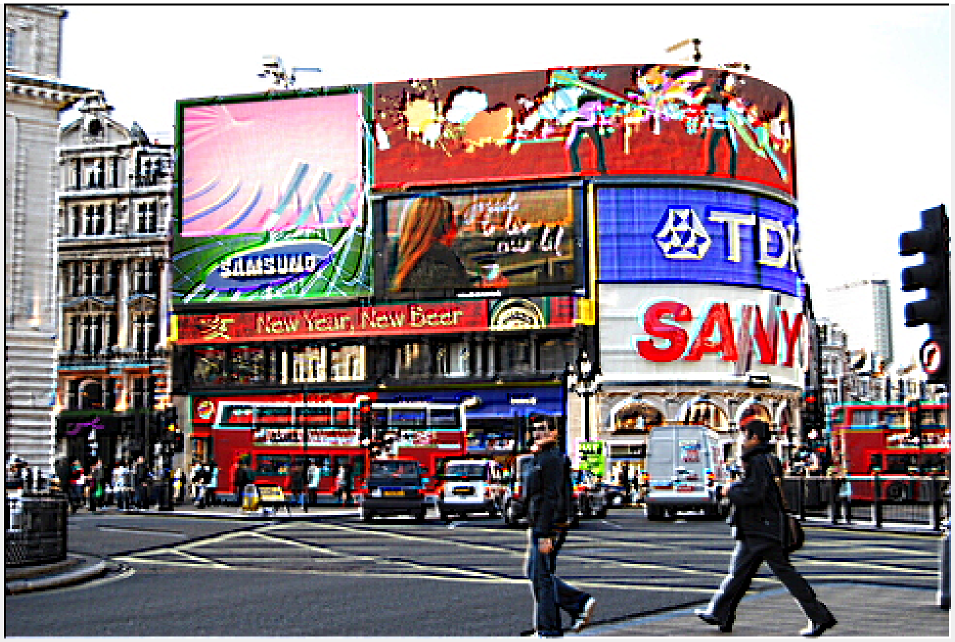
Unsharp.

Emboss

Kernel:

$$\begin{pmatrix} -2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

Divisor = 1.0, *Bias* = 0.0.

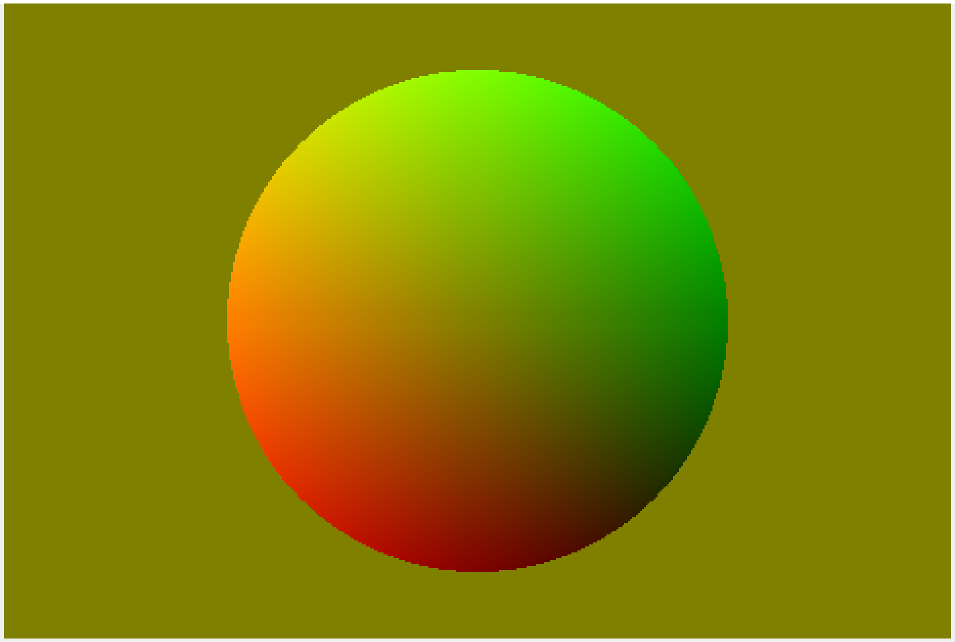


Emboss.

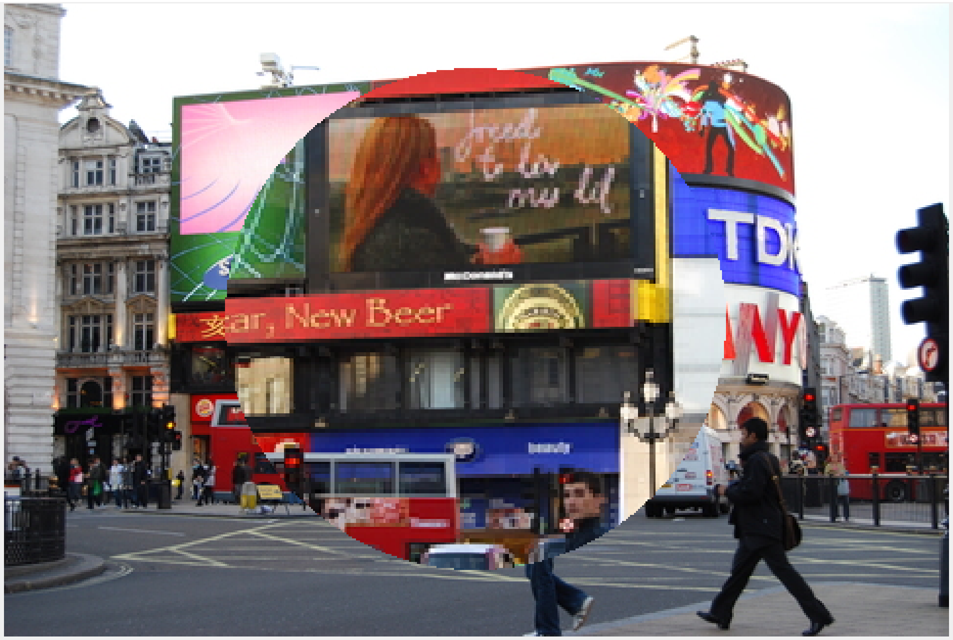
Displacement Map

The *Displacement Map* primitive distorts one *bitmap* using another as input. A pixel in the source *bitmap* is translated to a new coordinate via the equations *x'* = *scale* × (*CX(x,y)-0.5*) and *y'* = *scale* × (*CY(x,y)-0.5*) where *CX* and *CY* are the any of the *RGB* components or *Alpha*, selectable by the *X Channel* and *Y Channel* attributes. The *X Channel* and *Y Channel* can be mapped with different colors.

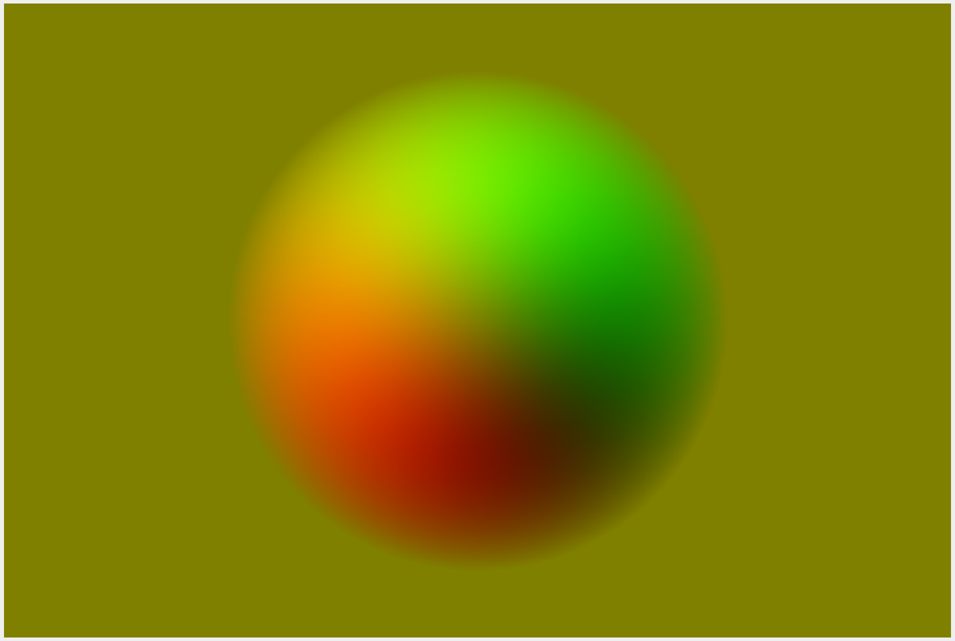
The *Displacement Map* filter can be used to produce some interesting effects... but figuring out a correct map can be difficult. In the following examples, the *x* displacement is set to red and the *y* displacement is set to green. The olive green background corresponds to red and green values of 127... which corresponds (almost) to no displacement.



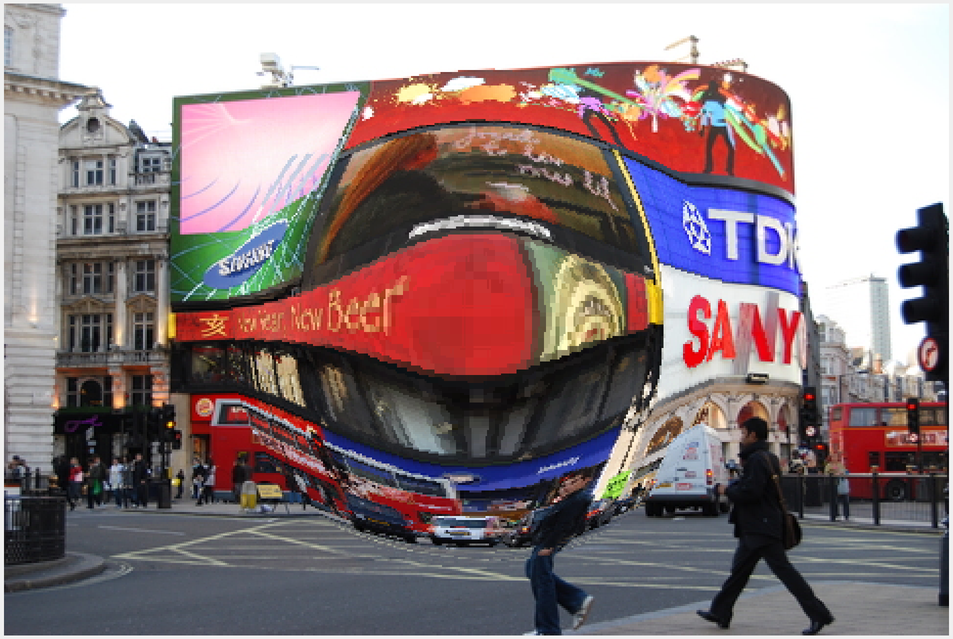
Magnify map (uniform gradients inside of circle).



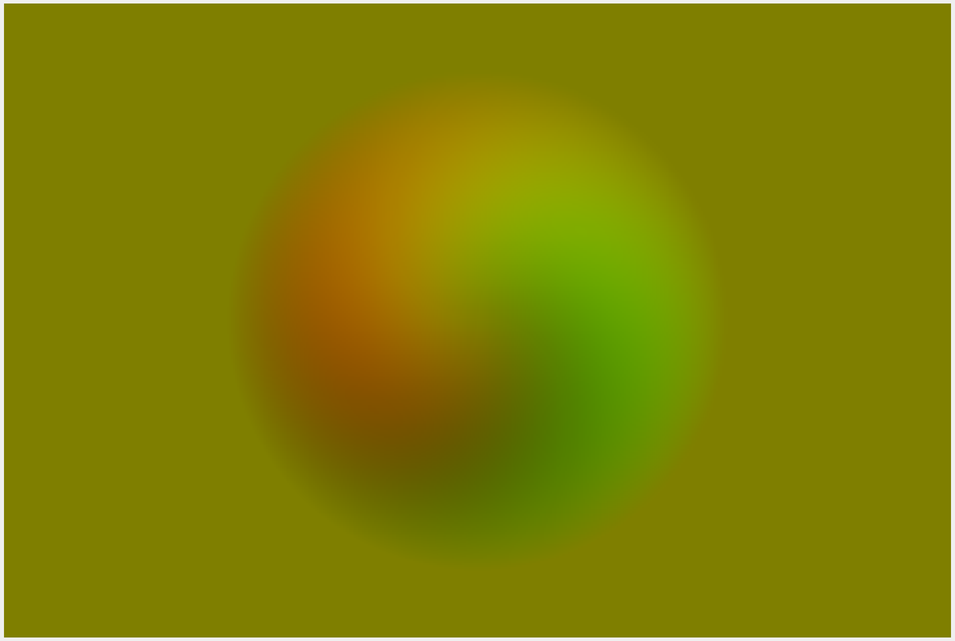
Magnify.



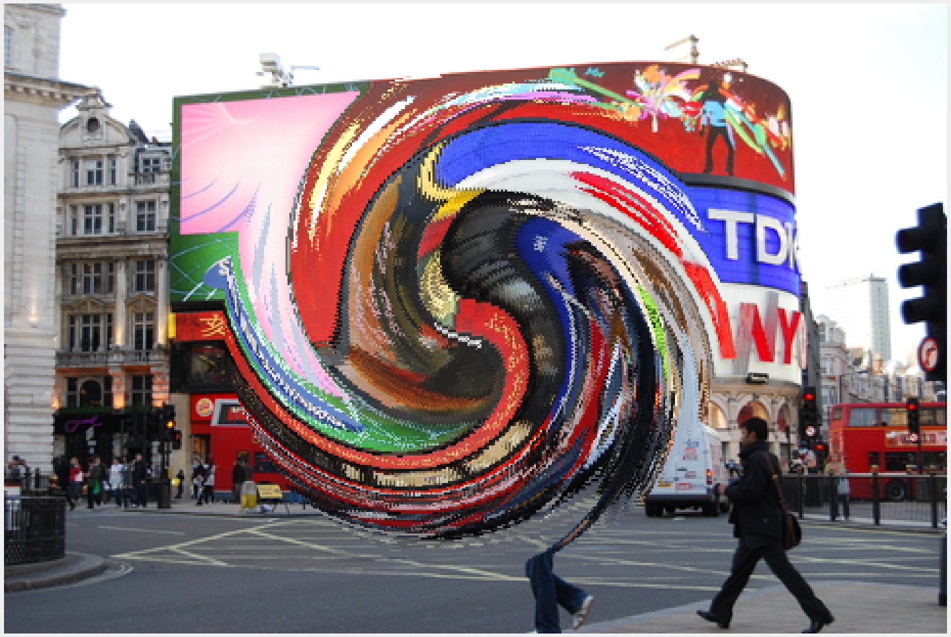
Bubble map.



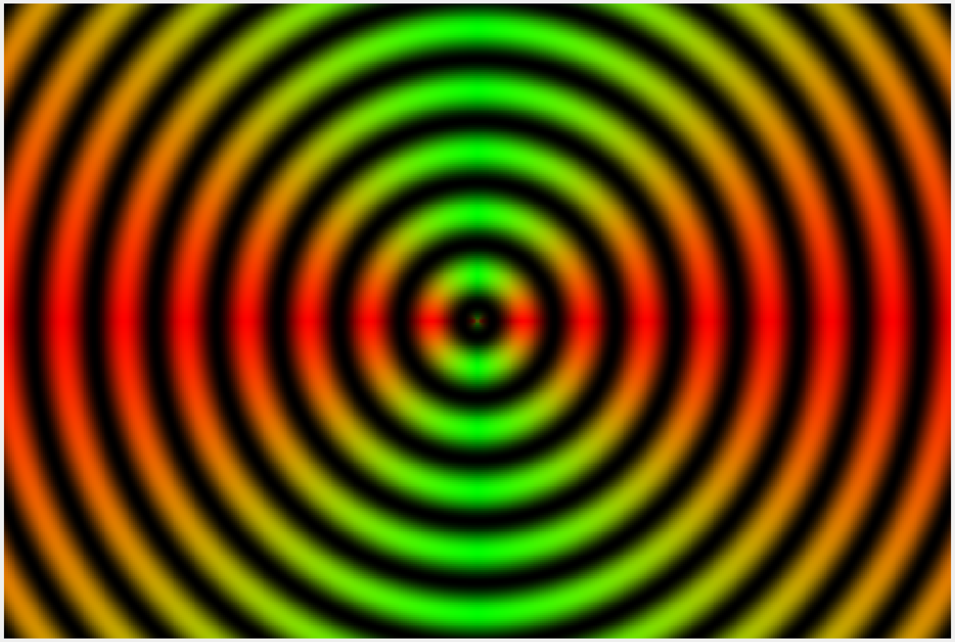
Bubble.



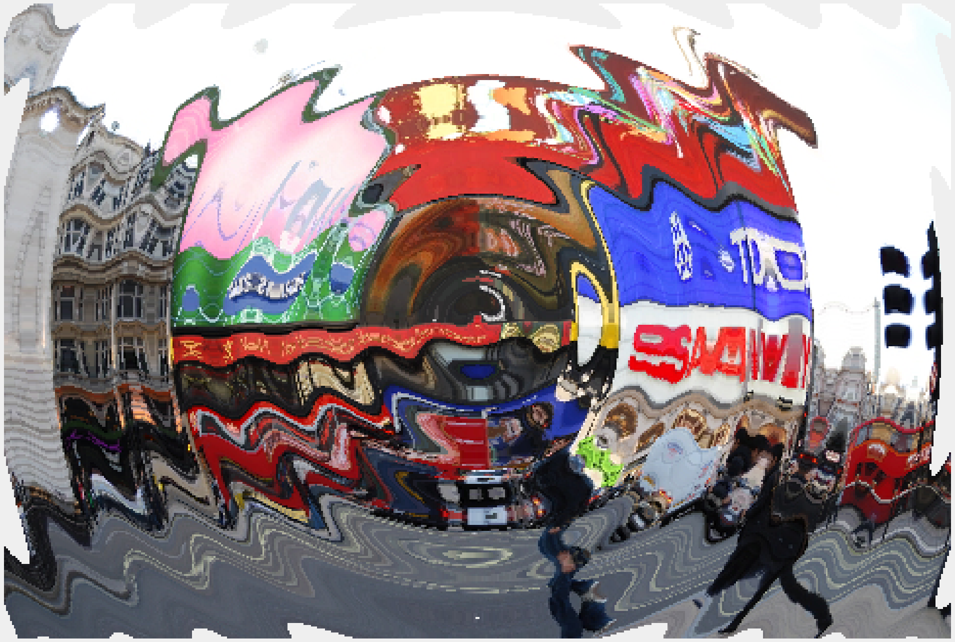
Twirl map.



Twirl.



Ripple map.



Ripple.

There are a couple of problems with the implementation of this filter in Inkscape v0.46. The first is that the implementation using a *Background Image* is buggy. This forces you to use an external image (via the *Image* filter) or use the output of another filter primitive as the displacement map. Another problem is the leakage of the image outside the intended area. This can be dealt with by setting the *Filter Effects Region* to the *bounding box* of the object.

See the Stereoscopic Picture example in the next section for another example of using this filter.

> **Note**
>
> Zero displacement? Not possible in the *SVG* standard. Color and *Alpha* are described by a byte (8 bits). This corresponds to a range of 0 to 255. A zero displacement would be a value halfway between 0 and 255, or 127.5. But these are integer numbers and you can't have a value of 127.5. Take your pick, 127 or 128. In most practical cases, the shift won't be noticable.

Gaussian Blur

The *Gaussian Blur* primitive blurs objects. Realistic highlights and reflections can be added to drawings as well as making objects out-of-focus. The primitive creates an output image by using a Gaussian weighted average of the input pixels around the location of each corresponding output pixel.

Internally, the amount of blur is defined in terms of the *blur radius* which for the mathematically inclined is just the standard deviation of the Gaussian. Technically, a Gaussian function extends to infinity. For practical reasons, the limit of an object's blur is two times the *blur radius* outside the *bounding box* at maximum blur.

The *Gaussian Blur* primitive is highly CPU intensive. The output is a trade off between speed and quality. One can set the *Blur quality* for the screen display in the *Inkscape Preferences* dialog (File → Inkscape Preferences... (**Shift+Ctrl+P**)) under the *Filter* entry. Choosing a low-quality option will affect blurring of thin objects the most. Bitmap export is always done at the highest quality (and thus may be slow).

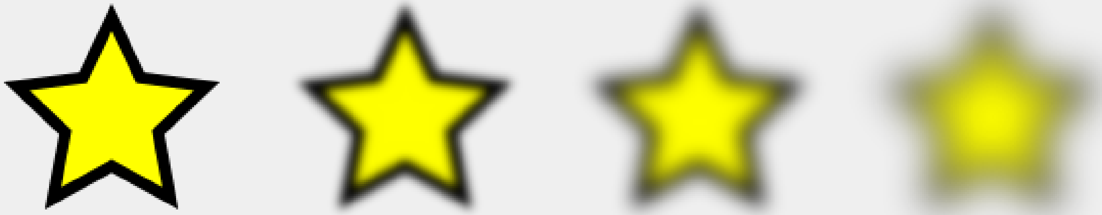
A *Gaussian Blur* filter can be created through both the *Filter Effects* and the *Fill and Stroke* dialogs.

Blurring with the Fill and Stroke Dialog

Using the *Fill and Stroke* dialog to create a blur is fast and easy. The dialog automatically creates the filter for you (and removes it if the blur is removed). In this dialog, the amount of blurring is defined in terms of a percentage. A blurring of 100% (the maximum blurring allowed) is equivalent to a *blur radius* of 1/8 of the *bounding box* perimeter (see above). For a square *bounding box*, this would be half of a side.

To apply a *Gaussian Blur* to an object, select the object and then adjust the blur with the *Blur* slider near the bottom of the dialog. Only a symmetric blur can be applied with this dialog.

> **Tip**
>
> Blurs created through the *Fill and Stroke* dialog depend on the size of the blurred object. To get the exact same amount of blur on different size objects, you can either use the Edit → Paste Style (**Shift+Ctrl+V**) command (if all the attributes are to be the same) or use the *Filter Effects* dialog to set the *blur radius* (*standard deviation*) to the same values.



Example of using the *Gaussian Blur* filter. The star on the left has been blurred respectively by 5%, 10%, and 20% to the right.

Blurring with the Filter Effects dialog

For more sophisticated use of the *Gaussian Blur* filter it is necessary to use the *Filter Effects* dialog. Through this dialog you can create asymmetric blurs as well as have precise control over the *blur radius*. You can also build more complicated filters as demonstrated in the *Drop Shadow* example earlier in this chapter.

Blurring Examples

Drop Shadow

As demonstrated in the introduction, Inkscape can be used to easily make an auto-updating drop shadow. However, prior to v0.46, the necessary *Offset* and *Merge* primitives were not available. Here is a work-around for Inkscape v0.45 that makes creating an auto-updating drop shadow easy:

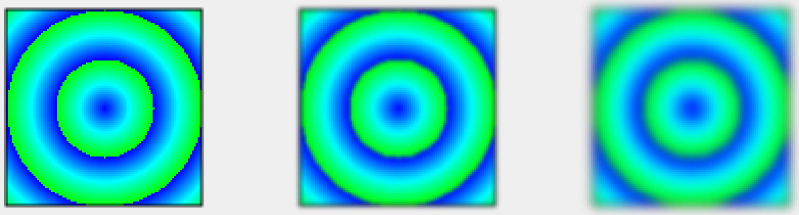
- Clone the object twice (Edit → Clone → Clone (**Alt+D**)).
- Remove the *Fill* and *Stroke* of the original object (select with Edit → Clone → Select Original (**Shift+D**) and use the *Fill and Stroke* dialog or the *Style Indicator* to *Unset* the *Fill* and *Stroke*).
- Add a *Fill* color to the top copy.
- Add a *Fill* color and shift the bottom copy. Blur with the *Fill and Stroke* dialog.

Drop
Shadow

A drop shadow created with the *Gaussian Blur* primitive and cloning. The shadow will automatically update if the original text is edited. The *Blur* amount was set to 2%.

Gradient Blurring

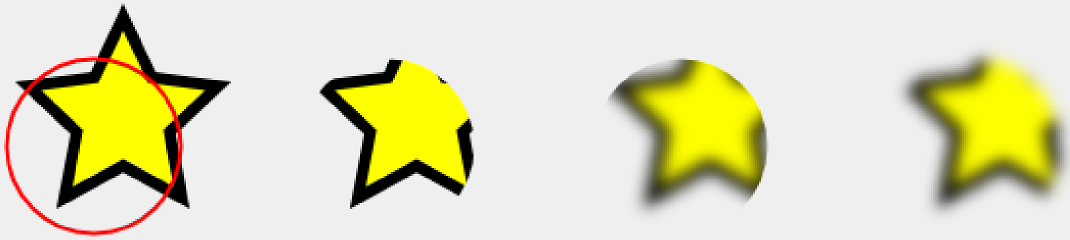
Blurring an object with a *Gradient* softens the color transitions.



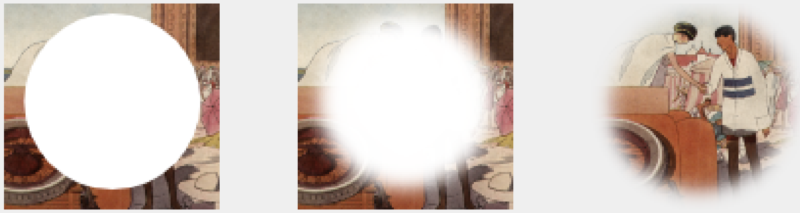
A *Rectangle* with a radial *Gradient*. Left: no blurring; Middle: 2% blurring; Right: 5% blurring.

Clipping and Masking

The *Gaussian Blur* primitive is applied to an object before any *Clipping or Masking*. This will give a sharp edge along the clipping path to a blurred object. If you wish the clipped edge to be blurred, put it in a *Group* by itself and then blur the *Group*. If you want a feathered edge to an object like a bitmap, create a white *transparency* mask with the edge blurred.



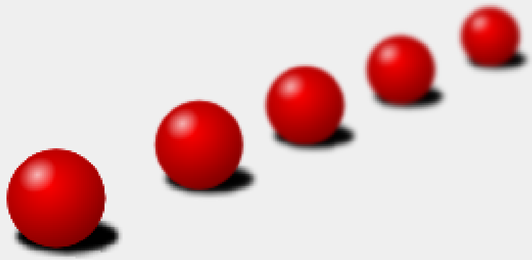
From left to right: Star with circular *Clip* path not yet applied; *Clip* path applied to star; star blurred; star put into *Group* with *Group* blurred.



From left to right: White circle mask over bitmap image; blurred circle mask over image; mask applied to image.

Tile Clones

The *Create Tiled Clones* dialog has an option to vary the *Blur Radius* under the *Blur and opacity* tab.

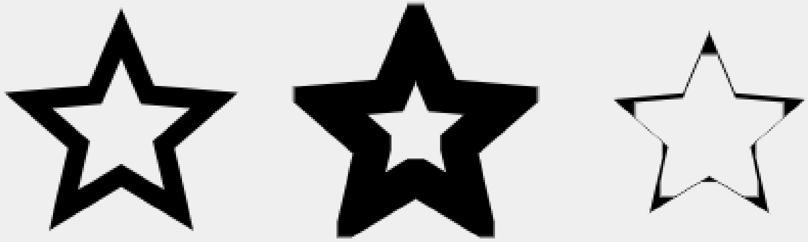


A sphere cloned with the *Create Tiled Clones* dialog, with *Shift*, *Scale*, and *Blur* changed from their default values.

Morphology

The *Morphology* primitive “fattens” or “thins” an object. The *Operator* attribute can either be *Dilate* or *Erode*. The amount of change is controlled by the *Radius* attribute. It can have independent *x* and *y* values.

Note: Inkscape seems to under do the transform.



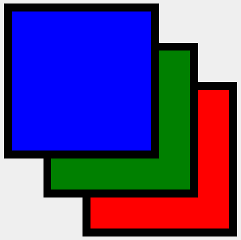
Examples of the *Morphology* primitive. From left to right: Unfiltered object. Dilated object with *Radius* of 4. Eroded object with *Radius* of 4.

Offset

The *Offset* primitive shifts a graphic by the specified amounts in *x* and *y*. The classic example is the use of this primitive to create a shadow. See the Drop Shadow example earlier in this section.

This primitive has two parameters: *Delta X*, the offset in the horizontal direction and *Delta Y* the offset in the vertical direction. Note that the positive *y* direction is in the downward direction (as defined by the *SVG* standard).

If the specified offset is large, the filter region needs to be enlarged. You can increase the filter region under the *Filter General Settings* tab at the bottom of the *Filter Effects* dialog.



Examples of the *Offset* primitive. The three squares are drawn on top of each other. The green and red squares are then shifted with the *Offset* primitive. The filter region for the green and red squares had to be enlarged.

[14] Why define a “Gaussian Blur” using the *Convolve Matrix* filter when a *Gaussian Blur* filter exists? The *Gaussian Blur* filter is designed to blur on a large scale. The *Convolve Matrix* filter works on a short scale, reducing noise from neighboring pixels.

Lighting Filters Primitives

Table of Contents

Complex Examples

Get the book.

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Filter Effects » Complex Examples

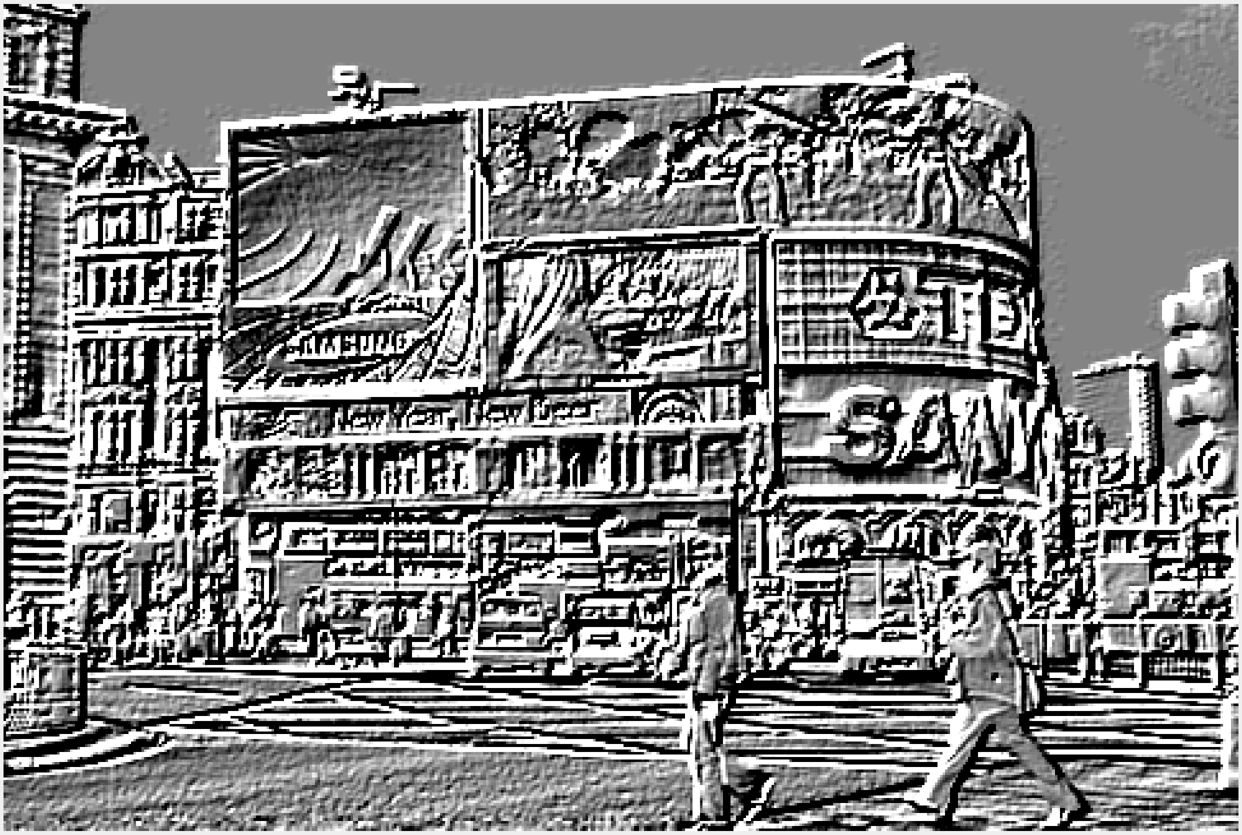
← Index →

Complex Examples

This section features some examples of complex filters built out of primitives.

Emboss

This example uses the *Color Matrix* to convert a photograph into an *Alpha* layer. The *Alpha* layer is then embossed by the *Diffuse Lighting* filter.



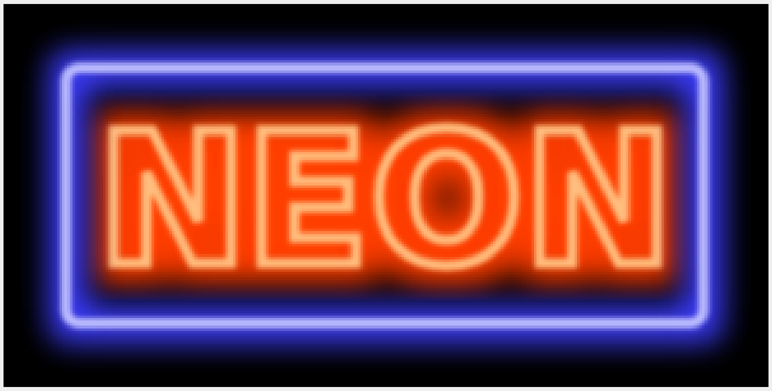
An embossed photograph.

Settings: *Color Matrix*: *In*: Source Graphics, *Mode*: Luminance to Alpha. *Diffuse Lighting*: *In*: Color Matrix output, *Diffuse Color*: White, *Surface Scale*: 10, *Constant*: 2, *Light Source*: Distant Light, *Azimuth*: 45, *Elevation*: 15.

Neon

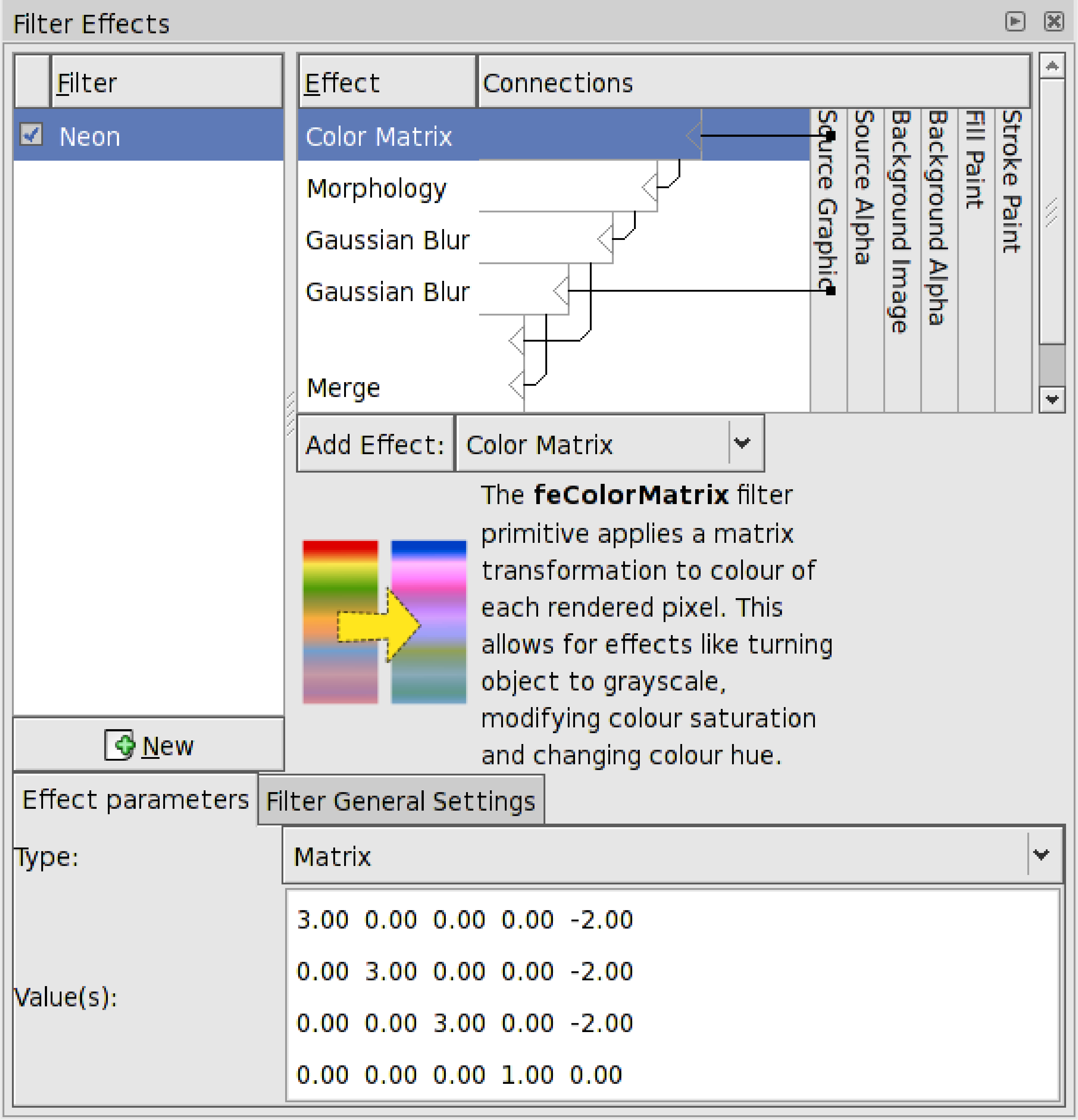
This filter effect is broken in Inkscape v0.46. It displays properly in Opera 9 and Firefox 3.

This example uses the *Morphology* primitive to create the glow around a neon tube. The glow color is derived from the neon color using the *Color Matrix* primitive. A couple *Gaussian Blur* primitives create the soft feel of the neon and a *Merge* primitive combines the neon and glow together.



A neon sign.

Settings: See below for overall structure and for *Color Matrix* primitive inputs. *Morphology*: *Operator*: Dilate, *Radius*: 6 (*x* and *y*); *Gaussian Blur*: *Standard Deviation*: 1st: 6, 2nd: 1.



Add Effect: Color Matrix

The **feColorMatrix** filter primitive applies a matrix transformation to colour of each rendered pixel. This allows for effects like turning object to grayscale, modifying colour saturation and changing colour hue.

Type: Matrix

Value(s):

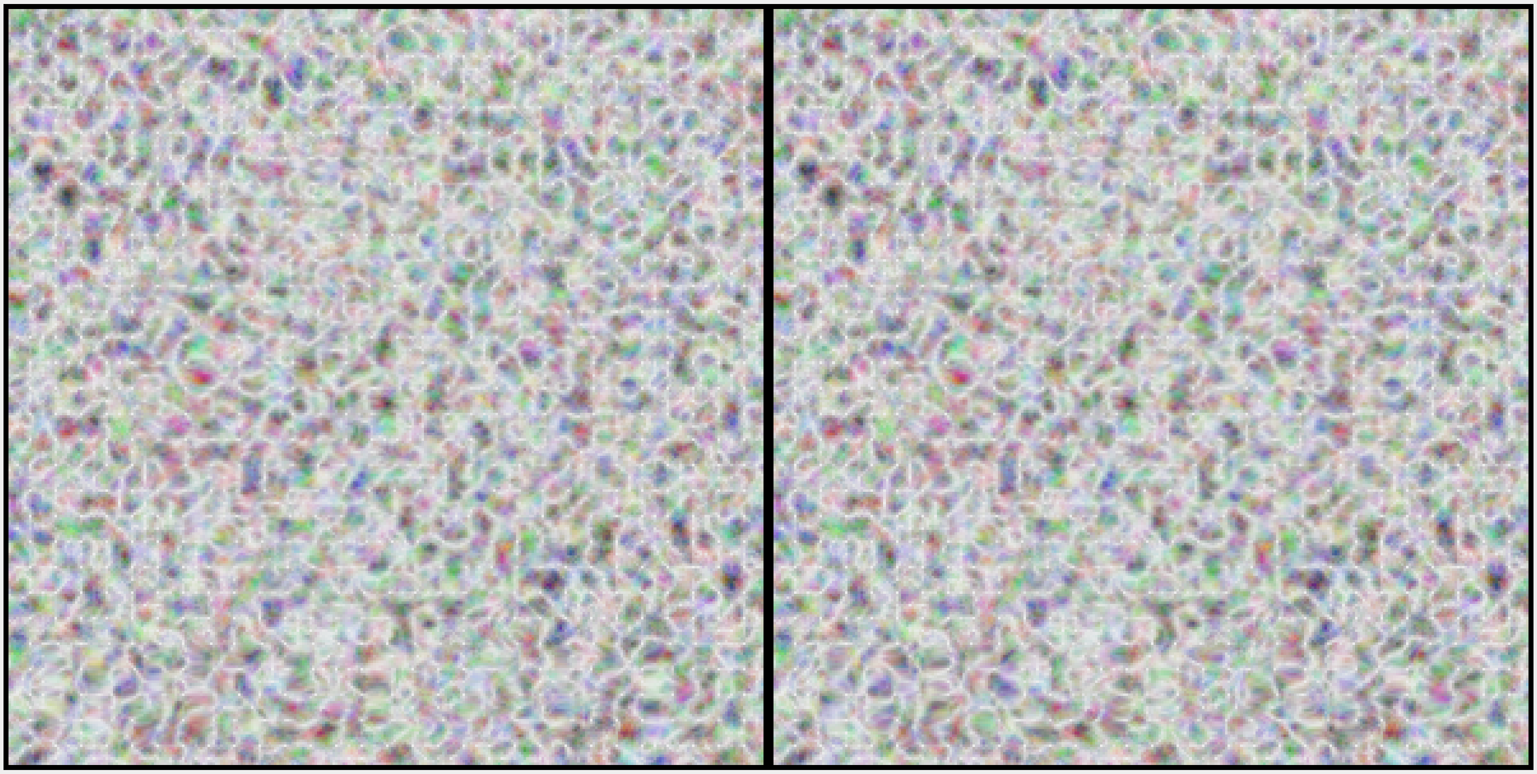
```
3.00  0.00  0.00  0.00  -2.00
0.00  3.00  0.00  0.00  -2.00
0.00  0.00  3.00  0.00  -2.00
0.00  0.00  0.00  1.00  0.00
```

Filter Effects dialog for neon effect, showing *Color Matrix* parameters.

Stereoscopic Pictures

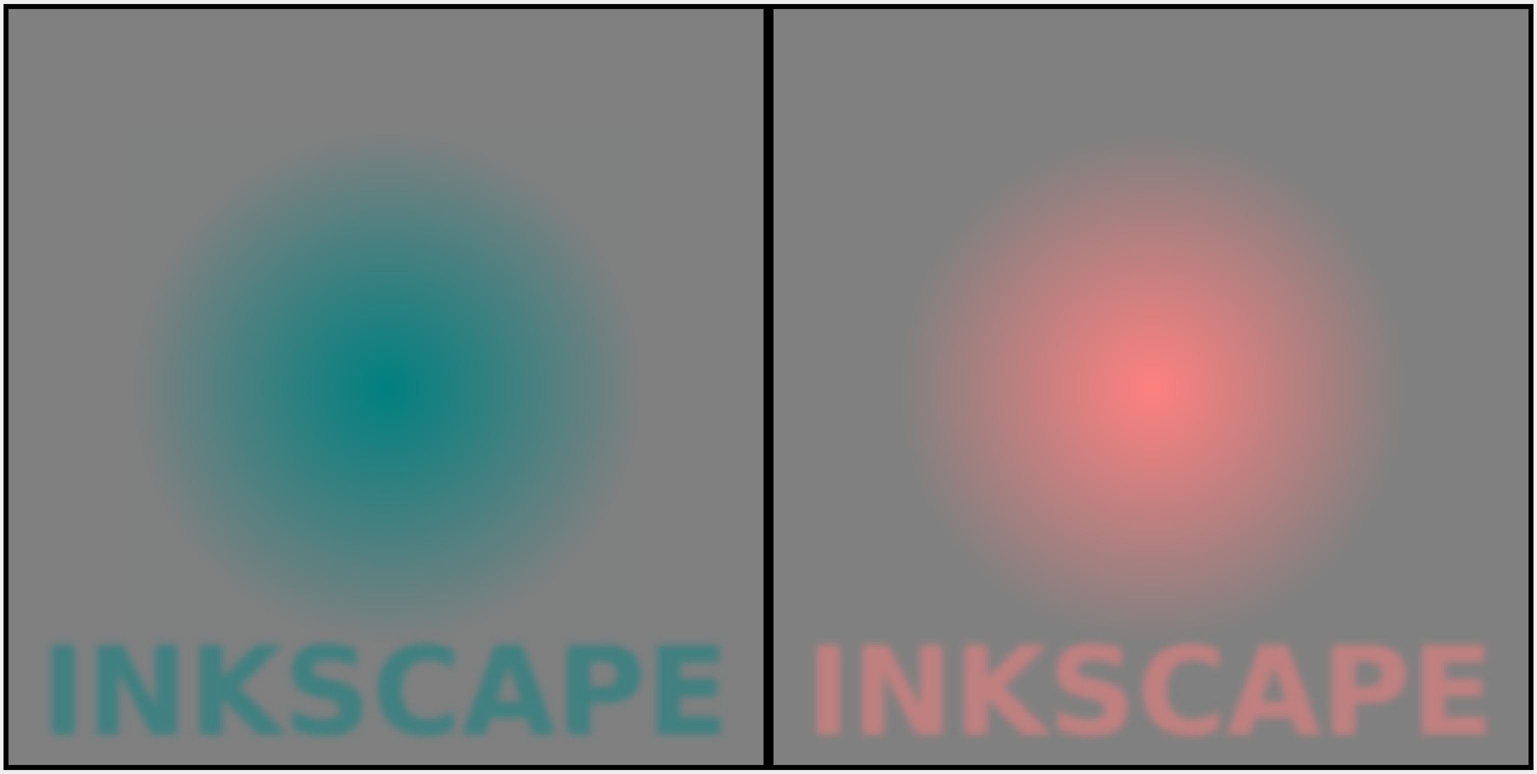
This filter effect is partially broken in Inkscape. It displays properly in Opera 9 but not Firefox 3.

This example uses the *Displacement Map* filter primitive to create a stereoscopic picture. The *Turbulence* primitive is used to generate a picture that is distorted with the *Displacement Map* primitive.



A stereoscopic image. Look at the left picture with the left eye and the right picture with the right eye. It may help to place a sheet of paper vertical and extending from your nose to the line between the two pictures. When your brain combines the two images, a shape should pop out.

Settings: Turbulence: *Input*: Source Alpha, *Type*: Turbulence, *Base Frequency*: 0.1, *Octaves* 5, *Seed* 0. Displacement Map: *Input 1*: Turbulence output, *Input 2*: Source Graphic, *Scale:*: 10, *X Channel*: Red, *Y Channel*: Green. Composite: *Input 1*: Displacement Map, *Input 2*: Source Graphic, *Operator*: In.

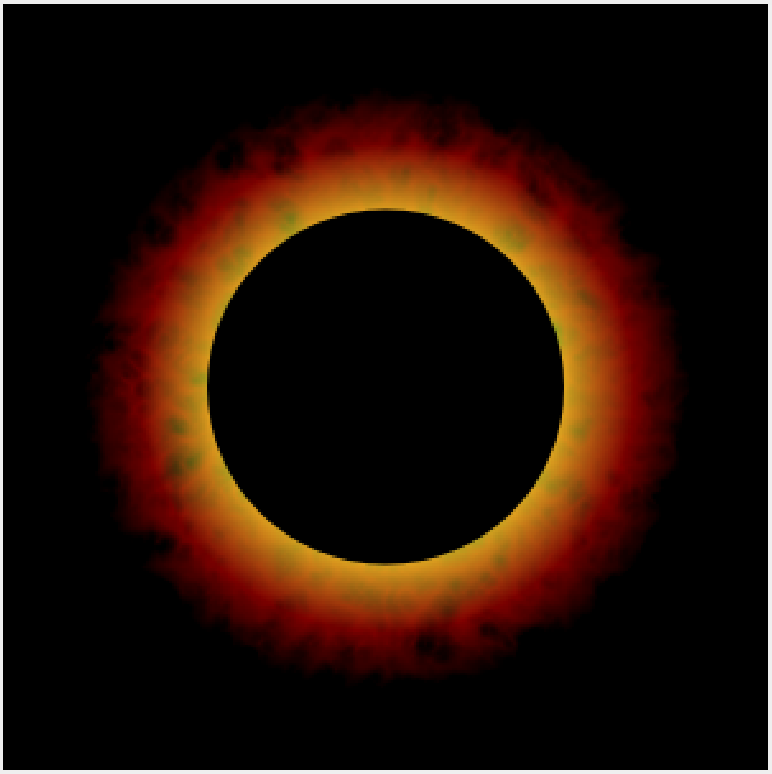


The *Source Graphic*. The red levels control the displacement in the *x* direction.

Solar Flare

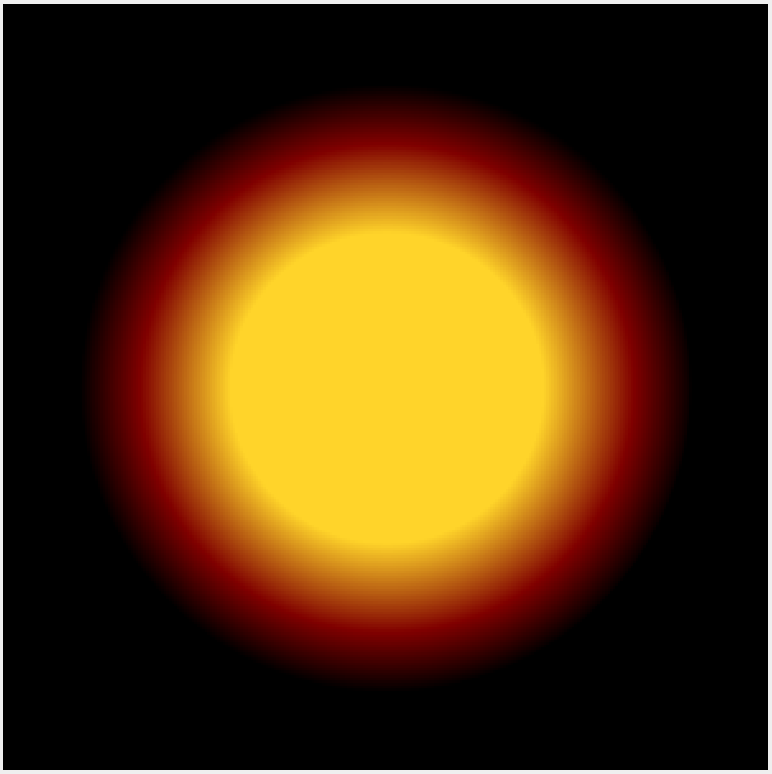
This filter effect displays in Firefox 3 but not Opera 9.

This example uses the *Turbulence* primitive to modify a radial *Gradient* and thus simulating a solar flare during an eclipse.



A solar flare seen during an eclipse.

Settings: Turbulence: *Input*: Source Alpha, *Type*: Turbulence, *Base Frequency*: 0.5, *Octaves* 3, *Seed* 0. Color Matrix: *Input*: Turbulence output, *Matrix*: All 0 except a_{00} and a_{33} are 1. Composite: *Input 1*: Source Graphic, *Input 2*: Color Matrix, *Operator*: Arithmetic, *K*: 0, 1, -0.75, 0.



The *Source Graphic*.

Pixel Manipulation Filter Primitives

Table of Contents

Chapter 14. XML Editor

previous next contents elements attributes properties index

14 January 2003

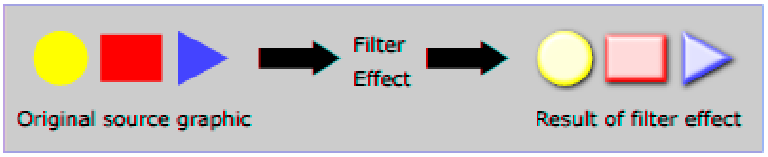
15 Filter Effects

Contents

15.1 Introduction

This chapter describes SVG's declarative filter effects feature set, which when combined with the 2D power of SVG can describe much of the common artwork on the Web in such a way that client-side generation and alteration can be performed easily. In addition, the ability to apply filter effects to SVG graphics elements and container elements helps to maintain the semantic structure of the document, instead of resorting to images which aside from generally being a fixed resolution tend to obscure the original semantics of the elements they replace. This is especially true for effects applied to text.

A filter effect consists of a series of graphics operations that are applied to a given **source graphic** to produce a modified graphical result. The result of the filter effect is rendered to the target device instead of the original source graphic. The following illustrates the process:



View this example as SVG (SVG-enabled browsers only)

Filter effects are defined by **'filter'** elements. To apply a filter effect to a graphics element or a container element, you set the value of the **'filter'** property on the given element such that it references the filter effect.

Each **'filter'** element contains a set of **filter primitives** as its children. Each filter primitive performs a single fundamental graphical operation (e.g., a blur or a lighting effect) on one or more inputs, producing a graphical result. Because most of the filter primitives represent some form of image processing, in most cases the output from a filter primitive is a single RGBA image.

The original source graphic or the result from a filter primitive can be used as input into one or more other filter primitives. A common application is to use the source graphic multiple times. For example, a simple filter could replace one graphic by two by adding a black copy of original source graphic offset to create a drop shadow. In effect, there are now two layers of graphics, both with the same original source graphics.

When applied to container elements such as **'g'**, the **'filter'** property applies to the contents of the group as a whole. The group's children do not render to the screen directly; instead, the graphics commands necessary to render the children are stored temporarily. Typically, the graphics commands are executed as part of the processing of the referenced **'filter'** element via use of the keywords **SourceGraphic** or **SourceAlpha**. Filter effects can be applied to container elements with no content (e.g., an empty **'g'** element), in which case the **SourceGraphic** or **SourceAlpha** consist of a transparent black rectangle that is the size of the **filter effects region**.

Sometimes filter primitives result in undefined pixels. For example, filter primitive **'feOffset'** can shift an image down and to the right, leaving undefined pixels at the top and left. In these cases, the undefined pixels are set to transparent black.

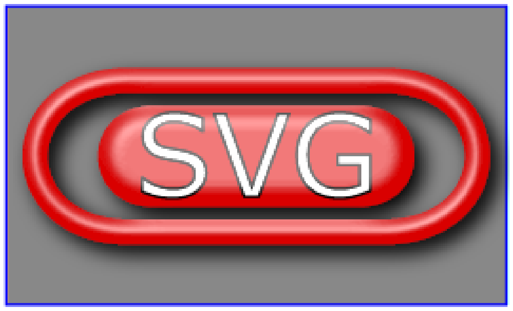
15.2 An example

The following shows an example of a filter effect.

Example filters01 - introducing filter effects.

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
              "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="7.5cm" height="5cm" viewBox="0 0 200 120"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <title>Example filters01.svg - introducing filter effects</title>
  <desc>An example which combines multiple filter primitives
        to produce a 3D lighting effect on a graphic consisting
        of the string "SVG" sitting on top of oval filled in red
        and surrounded by an oval outlined in red.</desc>
  <defs>
```

```
        <filter id="MyFilter" filterUnits="userSpaceOnUse" x="0" y="0" width="200" height="120">
          <feGaussianBlur in="SourceAlpha" stdDeviation="4" result="blur"/>
          <feOffset in="blur" dx="4" dy="4" result="offsetBlur"/>
          <feSpecularLighting in="blur" surfaceScale="5" specularConstant=".75"
                              specularExponent="20" lighting-color="#bbbbbb"
                              result="specOut">
            <fePointLight x="-5000" y="-10000" z="20000"/>
          </feSpecularLighting>
          <feComposite in="specOut" in2="SourceAlpha" operator="in" result="specOut"/>
          <feComposite in="SourceGraphic" in2="specOut" operator="arithmetic"
                       k1="0" k2="1" k3="1" k4="0" result="litPaint"/>
          <feMerge>
            <feMergeNode in="offsetBlur"/>
            <feMergeNode in="litPaint"/>
          </feMerge>
        </filter>
      </defs>
      <rect x="1" y="1" width="198" height="118" fill="#888888" stroke="blue" />
      <g filter="url(#MyFilter)" >
          <g>
          <path fill="none" stroke="#D90000" stroke-width="10"
                d="M50,90 C0,90 0,30 50,30 L150,30 C200,30 200,90 150,90 z" />
          <path fill="#D90000"
                d="M60,80 C30,80 30,40 60,40 L140,40 C170,40 170,80 140,80 z" />
          <g fill="#FFFFFF" stroke="black" font-size="45" font-family="Verdana" >
            <text x="52" y="76">SVG</text>
          </g>
        </g>
      </g>
    </svg>
```



Example filters01

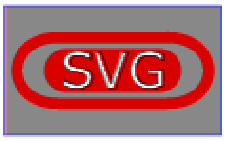
[View this example as SVG (SVG-enabled browsers only)]

The filter effect used in the example above is repeated here with reference numbers in the left column before each of the six filter primitives:

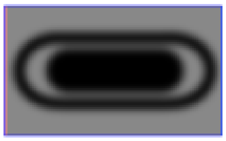
```
          <filter id="MyFilter" filterUnits="userSpaceOnUse" x="0" y="0" width="200" height="120">
            <desc>Produces a 3D lighting effect.</desc>
1           <feGaussianBlur in="SourceAlpha" stdDeviation="4" result="blur"/>
2           <feOffset in="blur" dx="4" dy="4" result="offsetBlur"/>
3           <feSpecularLighting in="blur" surfaceScale="5" specularConstant=".75"
                                specularExponent="20" lighting-color="#bbbbbb"
                                result="specOut">
              <fePointLight x="-5000" y="-10000" z="20000"/>
            </feSpecularLighting>
4           <feComposite in="specOut" in2="SourceAlpha" operator="in" result="specOut"/>
5           <feComposite in="SourceGraphic" in2="specOut" operator="arithmetic"
                         k1="0" k2="1" k3="1" k4="0" result="litPaint"/>
6           <feMerge>
```

```
        <feMergeNode in="offsetBlur"/>
        <feMergeNode in="litPaint"/>
      </feMerge>
    </filter>
```

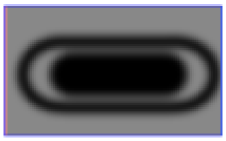
The following pictures show the intermediate image results from each of the six filter elements:



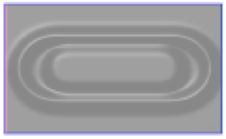
Source graphic



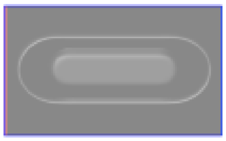
After filter primitive 1



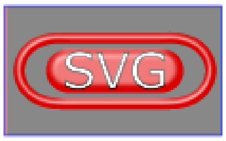
After filter primitive 2



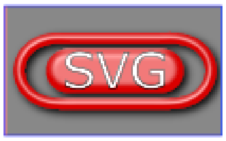
After filter primitive 3



After filter primitive 4



After filter primitive 5



After filter primitive 6

1. Filter primitive **'feGaussianBlur'** takes input **SourceAlpha**, which is the alpha channel of the source graphic. The result is stored in a temporary buffer named "blur". Note that "blur" is used as input to both filter primitives 2 and 3.
2. Filter primitive **'feOffset'** takes buffer "blur", shifts the result in a positive direction in both x and y, and creates a new buffer named "offsetBlur". The effect is that of a drop shadow.
3. Filter primitive **'feSpecularLighting'**, uses buffer "blur" as a model of a surface elevation and generates a lighting effect from a single point source. The result is stored in buffer "specOut".
4. Filter primitive **'feComposite'** masks out the result of filter primitive 3 by the original source graphics alpha channel so that the intermediate result is no bigger than the original source graphic.
5. Filter primitive **'feComposite'** composites the result of the specular lighting with the original source graphic.
6. Filter primitive **'feMerge'** composites two layers together. The lower layer consists of the drop shadow result from filter primitive 2. The upper layer consists of the specular lighting result from filter primitive 5.

15.3 The 'filter' element

The description of the **'filter'** element follows:

```
<!ENTITY % SVG.filter.extra.content "" >
<!ENTITY % SVG.filter.element "INCLUDE" >
<![%SVG.filter.element;[
<!ENTITY % SVG.filter.content
    "(( %SVG.Description.class; )*, ( %SVG.animate.qname; | %SVG.set.qname;
        %SVG.FilterPrimitive.class; %SVG.filter.extra.conten\
t; )*)"
>
<!ELEMENT %SVG.filter.qname; %SVG.filter.content; >
<!-- end of SVG.filter.element -->]]>
<!ENTITY % SVG.filter.attlist "INCLUDE" >
<![%SVG.filter.attlist;[
<!ATTLIST %SVG.filter.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.XLink.attrib;
    %SVG.External.attrib;
    x %Coordinate.datatype; #IMPLIED
    y %Coordinate.datatype; #IMPLIED
    width %Length.datatype; #IMPLIED
    height %Length.datatype; #IMPLIED
    filterRes %NumberOptionalNumber.datatype; #IMPLIED
    filterUnits ( userSpaceOnUse | objectBoundingBox ) #IMPLIED
    primitiveUnits ( userSpaceOnUse | objectBoundingBox ) #IMPLIED
>
```

Attribute definitions:

`filterUnits` = "*userSpaceOnUse* | *objectBoundingBox*"

See Filter effects region.

`primitiveUnits` = "*userSpaceOnUse* | *objectBoundingBox*"

Specifies the coordinate system for the various length values within the filter primitives and for the attributes that define the filter primitive subregion.

If **primitiveUnits="userSpaceOnUse"**, any length values within the filter definitions represent values in the current user coordinate system in place at the time when the **'filter'** element is referenced (i.e., the user coordinate system for the element referencing the **'filter'** element via a **'filter'** property).

If **primitiveUnits="objectBoundingBox"**, then any length values within the filter definitions represent fractions or percentages of the bounding box on the referencing element (see Object bounding box units).

If attribute **primitiveUnits** is not specified, then the effect is as if a value of **userSpaceOnUse** were specified.

Animatable: yes.

`x` = "*<coordinate>*"

See Filter effects region.

`y` = "*<coordinate>*"

See Filter effects region.

`width` = "*<length>*"

See Filter effects region.

`height` = "*<length>*"

See Filter effects region.

`filterRes` = "*<number-optional-number>*"

See Filter effects region.

`xlink:href` = "<uri>"

A URI reference to another **'filter'** element within the current SVG document fragment. Any attributes which are defined on the referenced **'filter'** element which are not defined on this element are inherited by this element. If this element has no defined filter nodes, and the referenced element has defined filter nodes (possibly due to its own **href** attribute), then this element inherits the filter nodes defined from the referenced **'filter'** element. Inheritance can be indirect to an arbitrary level; thus, if the referenced **'filter'** element inherits attributes or its filter node specification due to its own **href** attribute, then the current element can inherit those attributes or filter node specifications.

Animatable: yes.

Properties inherit into the **'filter'** element from its ancestors; properties do *not* inherit from the element referencing the **'filter'** element.

'filter' elements are never rendered directly; their only usage is as something that can be referenced using the **'filter'** property. The **'display'** property does not apply to the **'filter'** element; thus, **'filter'** elements are not directly rendered even if the **'display'** property is set to a value other than **none**, and **'filter'** elements are available for referencing even when the **'display'** property on the **'filter'** element or any of its ancestors is set to **none**.

15.4 The 'filter' property

The description of the **'filter'** property is as follows:

'filter'

Value: <uri> | none | inherit

Initial:	none
Applies to:	container elements and graphics elements
Inherited:	no
Percentages:	N/A
Media:	visual
Animatable:	yes

<uri>
A URI reference to a **'filter'** element which defines the filter effects that shall be applied to this element.

none
Do not apply any filter effects to this element.

15.5 Filter effects region

A **'filter'** element can define a region on the canvas to which a given filter effect applies and can provide a resolution for any intermediate continuous tone images used to process any raster-based filter primitives. The **'filter'** element has the following attributes which work together to define the filter effects region:

- **filterUnits={ userSpaceOnUse | objectBoundingBox }**.
  Defines the coordinate system for attributes **x, y, width, height**.
  If **filterUnits="userSpaceOnUse"**, **x, y, width, height** represent values in the current user coordinate system in place at the time when the **'filter'** element is referenced (i.e., the user coordinate system for the element referencing the **'filter'** element via a **'filter'** property).
  If **filterUnits="objectBoundingBox"**, then **x, y, width, height** represent fractions or percentages of the bounding box on the referencing element (see Object bounding box units).
  If attribute **filterUnits** is not specified, then the effect is as if a value of **objectBoundingBox** were specified.
  Animatable: yes.
- **x, y, width, height**, which define a rectangular region on the canvas to which this filter applies.
  The amount of memory and processing time required to apply the filter are related to the size of this rectangle and the **filterRes** attribute of the filter.
  The coordinate system for these attributes depends on the value for attribute **filterUnits**.
  Negative values for **width** or **height** are an error (see Error processing). Zero values disable rendering of the element which referenced the filter.
  The bounds of this rectangle act as a hard clipping region for each filter primitive included with a given **'filter'** element; thus, if the effect of a given filter primitive would extend beyond the bounds of the rectangle (this sometimes happens when using a **'feGaussianBlur'** filter primitive with a very large **stdDeviation**), parts of the effect will get clipped.
  If **x** or **y** is not specified, the effect is as if a value of "-10%" were specified.
  If **width** or **height** is not specified, the effect is as if a value of "120%" were specified.
  Animatable: yes.
- **filterRes** (which has the form `x-pixels [y-pixels]`) indicates the width and height of the intermediate images in pixels. If not provided, then a reasonable default resolution appropriate for the target device will be used. (For displays, an appropriate display resolution, preferably the current display's pixel resolution, is the default. For printing, an appropriate common printer resolution, such as 400dpi, is the default.)
  Care should be taken when assigning a non-default value to this attribute. Too small of a value may result in unwanted pixelation in the result. Too large of a value may result in slow processing and large memory usage.
  Negative values are an error (see Error processing). Zero values disable rendering of the element which referenced the filter.
  Animatable: yes.

Note that both of the two possible value for **filterUnits** (i.e., **objectBoundingBox** and **userSpaceOnUse**) result in a filter region whose coordinate system has its X-axis and Y-axis each parallel to the X-axis and Y-axis, respectively, of the user coordinate system for the element to which the filter will be applied.

Sometimes implementers can achieve faster performance when the filter region can be mapped directly to device pixels; thus, for best performance on display devices, it is suggested that authors define their region such that SVG user agent can align the filter region pixel-for-pixel with the background. In particular, for best filter effects performance, avoid rotating or skewing the user coordinate system. Explicit values for attribute **filterRes** can either help or harm performance. If **filterRes** is smaller than the automatic (i.e., default) filter resolution, then filter effect might have faster performance (usually at the expense of quality). If **filterRes** is larger than the automatic (i.e., default) filter resolution, then filter effects performance will usually be slower.

It is often necessary to provide padding space because the filter effect might impact bits slightly outside the tight-fitting bounding box on a given object. For these purposes, it is possible to provide negative percentage values for **x, y** and percentages values greater than 100% for **width, height**. This, for example, is why the defaults for the filter effects region are x="-10%" y="-10%" width="120%" height="120%".

15.6 Accessing the background image

Two possible pseudo input images for filter effects are BackgroundImage and BackgroundAlpha, which each represent an image snapshot of the canvas under the filter region at the time that the **'filter'** element is invoked. BackgroundImage represents both the color values and alpha channel of the canvas (i.e., RGBA pixel values), whereas BackgroundAlpha represents only the alpha channel.

Implementations of SVG user agents often will need to maintain supplemental background image buffers in order to support the BackgroundImage and BackgroundAlpha pseudo input images. Sometimes, the background image buffers will contain an in-memory copy of the accumulated painting operations on the current canvas.

Because in-memory image buffers can take up significant system resources, SVG content must explicitly indicate to the SVG user agent that the document needs access to the background image before BackgroundImage and BackgroundAlpha pseudo input images can be used. The property which enables access to the background image is **'enable-background'**:

'enable-background'

Value:	accumulate | new [<x> <y> <width> <height>] | inherit
Initial:	accumulate
Applies to:	container elements
Inherited:	no
Percentages:	N/A
Media:	visual
Animatable:	no

'enable-background' is only applicable to container elements and specifies how the SVG user agents manages the accumulation of the background image.

A value of **new** indicates two things:

- It enables the ability of children of the current container element to access the background image.
- It indicates that a new (i.e., initially transparent black) background image canvas is established and that (in effect) all children of the current container element shall be rendered into the new background image canvas in addition to being rendered onto the target device.

A meaning of **enable-background: accumulate** (the initial/default value) depends on context:

- If an ancestor container element has a property value of 'enable-background:new', then all graphics elements within the current container element are rendered both onto the parent container element's background image canvas and onto the target device.
- Otherwise, there is no current background image canvas, so it is only necessary to render graphics elements onto the target device. (No need to render to the background image canvas.)

If a filter effect specifies either the BackgroundImage or the BackgroundAlpha pseudo input images and no ancestor container element has a property value of 'enable-background:new', then the background image request is technically in error. Processing will proceed without interruption (i.e., no error message) and a transparent black image shall be provided in response to the request.

The optional **<x>,<y>,<width>,<height>** parameters on the **new** value indicate the subregion of the container element's user space where access to the background image is allowed to happen. These parameters enable the SVG user agent potentially to allocate smaller temporary image buffers than the default values, which might require the SVG user agent to allocate buffers as large as the current viewport. Thus, the values <x>,<y>,<width>,<height> act as a clipping rectangle on the background image canvas. Negative values for <width> or <height> are an error (see Error processing). If more than zero but less than four of the values <x>,<y>,<width> and <height> are specified or if zero values are specified for <width> or <height>, BackgroundImage and BackgroundAlpha are processed as if background image processing were not enabled.

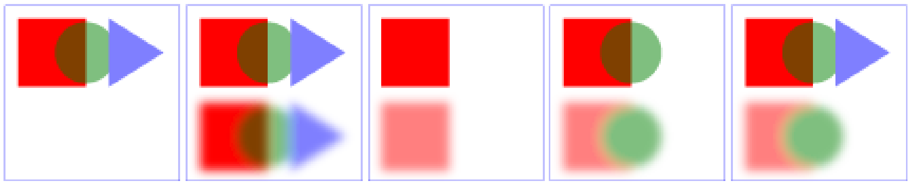
Assume you have an element E in the document and that E has a series of ancestors A_1 (its immediate parent), A_2, etc. (Note: A_0 is E.) Each ancestor A_i will have a corresponding temporary background image offscreen buffer BUF_i. The contents of the *background image* available to a **'filter'** referenced by E is defined as follows:

- Find the element A_i with the smallest subscript i (including A_0=E) for which the **'enable-background'** property has the value **new**. (Note: if there is no such ancestor element, then there is no background image available to E, in which case a transparent black image will be used as E's background image.)
- For each A_i (from i=n to 1), initialize BUF_i to transparent black. Render all children of A_i up to but not including A_{i-1} into BUF_i. The children are painted, then filtered, clipped, masked and composited using the various painting, filtering, clipping, masking and object opacity settings on the given child. Any filter effects, masking and group opacity that might be set on A_i do *not* apply when rendering the children of A_i into BUF_i.

  (Note that for the case of A_0=E, the graphical contents of E are not rendered into BUF_1 and thus are not part of the background image available to E. Instead, the graphical contents of E are available via the SourceGraphic and SourceAlpha pseudo input images.)
- Then, for each A_i (from i=1 to n-1), composite BUF_i into BUF_{i+1}.
- The accumulated result (i.e., BUF_n) represents the background image available to E.

Example enable-background-01 illustrates the rules for background image processing.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="13.5cm" height="2.7cm" viewBox="0 0 1350 270"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <title>Example enable-background01</title>
  <desc>This test case shows five pictures which illustrate the rules
        for background image processing.</desc>
  <defs>
    <filter id="ShiftBGAndBlur"
            filterUnits="userSpaceOnUse" x="0" y="0" width="1200" height="400">
      <desc>
        This filter discards the SourceGraphic, if any, and just produces
        a result consisting of the BackgroundImage shifted down 125 units
        and then blurred.
      </desc>
      <feOffset in="BackgroundImage" dx="0" dy="125" />
      <feGaussianBlur stdDeviation="8" />
    </filter>
    <filter id="ShiftBGAndBlur_WithSourceGraphic"
            filterUnits="userSpaceOnUse" x="0" y="0" width="1200" height="400">
      <desc>
        This filter takes the BackgroundImage, shifts it down 125 units, blurs it,
        and then renders the SourceGraphic on top of the shifted/blurred background.
      </desc>
      <feOffset in="BackgroundImage" dx="0" dy="125" />
      <feGaussianBlur stdDeviation="8" result="blur" />
      <feMerge>
        <feMergeNode in="blur"/>
        <feMergeNode in="SourceGraphic"/>
      </feMerge>
    </filter>
  </defs>
  <g transform="translate(0,0)">
    <desc>The first picture is our reference graphic without filters.</desc>
    <rect x="25" y="25" width="100" height="100" fill="red"/>
    <g opacity=".5">
      <circle cx="125" cy="75" r="45" fill="green"/>
      <polygon points="160,25 160,125 240,75" fill="blue"/>
    </g>
    <rect x="5" y="5" width="260" height="260" fill="none" stroke="blue"/>
  </g>
  <g enable-background="new" transform="translate(270,0)">
    <desc>The second adds an empty 'g' element which invokes ShiftBGAndBlur.</desc>
    <rect x="25" y="25" width="100" height="100" fill="red"/>
    <g opacity=".5">
      <circle cx="125" cy="75" r="45" fill="green"/>
      <polygon points="160,25 160,125 240,75" fill="blue"/>
    </g>
    <g filter="url(#ShiftBGAndBlur)"/>
    <rect x="5" y="5" width="260" height="260" fill="none" stroke="blue"/>
  </g>
  <g enable-background="new" transform="translate(540,0)">
    <desc>The third invokes ShiftBGAndBlur on the inner group.</desc>
    <rect x="25" y="25" width="100" height="100" fill="red"/>
    <g filter="url(#ShiftBGAndBlur)" opacity=".5">
      <circle cx="125" cy="75" r="45" fill="green"/>
      <polygon points="160,25 160,125 240,75" fill="blue"/>
    </g>
    <rect x="5" y="5" width="260" height="260" fill="none" stroke="blue"/>
  </g>
  <g enable-background="new" transform="translate(810,0)">
    <desc>The fourth invokes ShiftBGAndBlur on the triangle.</desc>
    <rect x="25" y="25" width="100" height="100" fill="red"/>
    <g opacity=".5">
      <circle cx="125" cy="75" r="45" fill="green"/>
      <polygon points="160,25 160,125 240,75" fill="blue"
               filter="url(#ShiftBGAndBlur)"/>
    </g>
    <rect x="5" y="5" width="260" height="260" fill="none" stroke="blue"/>
  </g>
  <g enable-background="new" transform="translate(1080,0)">
    <desc>The fifth invokes ShiftBGAndBlur_WithSourceGraphic on the triangle.</desc>
    <rect x="25" y="25" width="100" height="100" fill="red"/>
    <g opacity=".5">
      <circle cx="125" cy="75" r="45" fill="green"/>
      <polygon points="160,25 160,125 240,75" fill="blue"
               filter="url(#ShiftBGAndBlur_WithSourceGraphic)"/>
    </g>
    <rect x="5" y="5" width="260" height="260" fill="none" stroke="blue"/>
  </g>
</svg>
```



Example enable-background-01

View this example as SVG (SVG-enabled browsers only)

The example above contains five parts, described as follows:

1. The first set is the reference graphic. The reference graphic consists of a red rectangle followed by a 50% transparent **'g'** element. Inside the **'g'** is a green circle that partially overlaps the rectangle and a a blue triangle that partially overlaps the circle. The three objects are then outlined by a rectangle stroked with a thin blue line. No filters are applied to the reference graphic.
2. The second set enables background image processing and adds an empty **'g'** element which invokes the ShiftBGAndBlur filter. This filter takes the current accumulated background image (i.e., the entire reference graphic) as input, shifts its offscreen down, blurs it, and then writes the result to the canvas. Note that the offscreen for the filter is initialized to transparent black, which allows the already rendered rectangle, circle and triangle to show through after the filter renders its own result to the canvas.
3. The third set enables background image processing and instead invokes the ShiftBGAndBlur filter on the inner **'g'** element. The accumulated background at the time the filter is applied contains only the red rectangle. Because the children of the inner **'g'** (i.e., the circle and triangle) are not part of the inner **'g'** element's background and because ShiftBGAndBlur ignores SourceGraphic, the children of the inner **'g'** do not appear in the result.
4. The fourth set enables background image processing and invokes the ShiftBGAndBlur on the **'polygon'** element that draws the triangle. The accumulated background at the time the filter is applied contains the red rectangle plus the green circle ignoring the effect of the **'opacity'** property on the inner **'g'** element. (Note that the blurred green circle at the bottom does not let the red rectangle show through on its left side. This is due to ignoring the effect of the **'opacity'** property.) Because the triangle itself is not part of the accumulated background and because ShiftBGAndBlur ignores SourceGraphic, the triangle does not appear in the result.
5. The fifth set is the same as the fourth except that filter ShiftBGAndBlur_WithSourceGraphic is invoked instead of ShiftBGAndBlur. ShiftBGAndBlur_WithSourceGraphic performs the same effect as ShiftBGAndBlur, but then

renders the SourceGraphic on top of the shifted, blurred background image. In this case, SourceGraphic is the blue triangle; thus, the result is the same as in the fourth case except that the blue triangle now appears.

15.7 Filter primitives overview

15.7.1 Overview

This section describes the various filter primtives that can be assembled to achieve a particular filter effect.

Unless otherwise stated, all image filters operate on premultiplied RGBA samples. Filters which work more naturally on non-premultiplied data (feColorMatrix and feComponentTransfer) will temporarily undo and redo premultiplication as specified. All raster effect filtering operations take 1 to N input RGBA images, additional attributes as parameters, and produce a single output RGBA image.

The RGBA result from each filter primitive will be clamped into the allowable ranges for colors and opacity values. Thus, for example, the result from a given filter primitive will have any negative color values or opacity values adjusted up to color/opacity of zero.

The color space in which a particular filter primitive performs its operations is determined by the value of property **'color-interpolation-filters'** on the given filter primitive. A different property, **'color-interpolation'** determines the color space for other color operations. Because these two properties have different initial values (**'color-interpolation-filters'** has an initial value of **linearRGB** whereas **'color-interpolation'** has an initial value of **sRGB**), in some cases to achieve certain results (e.g., when coordinating gradient interpolation with a filtering operation) it will be necessary to explicitly set **'color-interpolation'** to **linearRGB** or **'color-interpolation-filters'** to **sRGB** on particular elements. Note that the examples below do not explicitly set either **'color-interpolation'** or **'color-interpolation-filters'**, so the initial values for these properties apply to the examples.

15.7.2 Common attributes

The following attributes are available for most of the filter primitives:

```
<!ENTITY % SVG.FilterPrimitive.extra.attrib "" >
<!ENTITY % SVG.FilterPrimitive.attrib
    "x %Coordinate.datatype; #IMPLIED
     y %Coordinate.datatype; #IMPLIED
     width %Length.datatype; #IMPLIED
     height %Length.datatype; #IMPLIED
     result CDATA #IMPLIED
     %SVG.FilterPrimitive.extra.attrib;"
>
<!-- SVG.FilterPrimitiveWithIn.attrib ..................\
 -->
<!ENTITY % SVG.FilterPrimitiveWithIn.extra.attrib "" >
<!ENTITY % SVG.FilterPrimitiveWithIn.attrib
    "%SVG.FilterPrimitive.attrib;
     in CDATA #IMPLIED
     %SVG.FilterPrimitiveWithIn.extra.attrib;"
>
```

Attribute definitions:

x = "*<coordinate>*"

The minimum x coordinate for the subregion which restricts calculation and rendering of the given filter primitive. See filter primitive subregion.

Animatable: yes.

y = "*<coordinate>*"

The minimum y coordinate for the subregion which restricts calculation and rendering of the given filter primitive. See filter primitive subregion. *Animatable: yes.*

`width` = "*<length>*"

The width of the subregion which restricts calculation and rendering of the given filter primitive. See filter primitive subregion.

A negative value is an error (see Error processing). A value of zero disables the effect of the given filter primitive (i.e., the result is a transparent black image).

Animatable: yes.

`height` = "*<length>*"

The height of the subregion which restricts calculation and rendering of the given filter primitive. See filter primitive subregion.

A negative value is an error (see Error processing). A value of zero disables the effect of the given filter primitive (i.e., the result is a transparent black image).

Animatable: yes.

`result` = "*<filter-primitive-reference>*"

Assigned name for this filter primitive. If supplied, then graphics that result from processing this filter primitive can be referenced by an **in** attribute on a subsequent filter primitive within the same **'filter'** element. If no value is provided, the output will only be available for re-use as the implicit input into the next filter primitive if that filter primitive provides no value for its **in** attribute.

Note that a *<filter-primitive-reference>* is not an XML ID; instead, a *<filter-primitive-reference>* is only meaningful within a given **'filter'** element and thus have only local scope. It is legal for the same *<filter-primitive-reference>* to appear multiple times within the same **'filter'** element. When referenced, the *<filter-primitive-reference>* will use the closest preceding filter primitive with the given result.

Animatable: yes.

`in` = "*SourceGraphic* | *SourceAlpha* | *BackgroundImage* | *BackgroundAlpha* | *FillPaint* | *StrokePaint* | *<filter-primitive-reference>*"

Identifies input for the given filter primitive. The value can be either one of six keywords or can be a string which matches a previous **result** attribute value within the same **'filter'** element. If no value is provided and this is the first filter primitive, then this filter primitive will use **SourceGraphic** as its input. If no value is provided and this is a subsequent filter primitive, then this filter primitive will use the result from the previous filter primitive as its input.

If the value for **result** appears multiple times within a given **'filter'** element, then a reference to that result will use the closest preceding filter primitive with the given value for attribute **result**. Forward references to results are an error.

Definitions for the six keywords:

SourceGraphic

This keyword represents the graphics elements that were the original input into the **'filter'** element. For raster effects filter primitives, the graphics elements will be rasterized into an initially clear RGBA raster in image space. Pixels left untouched by the original graphic will be left clear. The image is specified to be rendered in linear RGBA pixels. The alpha channel of this image captures any anti-aliasing specified by SVG. (Since the raster is linear, the alpha channel of this image will represent the exact percent coverage of each pixel.)

SourceAlpha

This keyword represents the graphics elements that were the original input into the **'filter'** element. **SourceAlpha** has all of the same rules as **SourceGraphic** except that only the alpha channel is used. The input image is an RGBA image consisting of implicitly black color values for the RGB channels, but whose alpha channel is the same as **SourceGraphic**. If this option is used, then some implementations might need to rasterize the graphics elements in order to extract the alpha channel.

BackgroundImage

This keyword represents an image snapshot of the canvas under the filter region at the time that the **'filter'** element was invoked. See Accessing the background image.

BackgroundAlpha

Same as BackgroundImage except only the alpha channel is used. See **SourceAlpha** and Accessing the background image.

FillPaint

This keyword represents the value of the **'fill'** property on the target element for the filter effect. The FillPaint image has conceptually infinite extent. Frequently this image is opaque everywhere, but it might not be if the "paint" itself has alpha, as in the case of a gradient or pattern which itself includes transparent or semi-transparent parts.

StrokePaint

This keyword represents the value of the **'stroke'** property on the target element for the filter effect. The StrokePaint image has conceptually infinite extent. Frequently this image is opaque everywhere, but it might not be if the "paint" itself has alpha, as in the case of a gradient or pattern which itself includes transparent or semi-transparent parts.

Animatable: yes.

15.7.3 Filter primitive subregion

All filter primitives have attributes **x**, **y**, **width** and **height** which identify a subregion which restricts calculation and rendering of the given filter primitive. These attributes are defined according to the same rules as other filter primitives' coordinate and length attributes and thus represent values in the coordinate system established by attribute **primitiveUnits** on the **'filter'** element.

x, **y**, **width** and **height** default to the union (i.e., tightest fitting bounding box) of the subregions defined for all referenced nodes. If there are no referenced nodes (e.g., for **'feImage'** or **'feTurbulence'**), or one or more of the referenced nodes is a standard input (one of **SourceGraphic**, **SourceAlpha**, **BackgroundImage**, **BackgroundAlpha**, **FillPaint** or **StrokePaint**), or for **'feTile'** (which is special because its principal function is to replicate the referenced node in X and Y and thereby produce a usually larger result), the default subregion is 0%,0%,100%,100%, where percentages are relative to the dimensions of the filter region.

x, **y**, **width** and **height** act as a hard clip clipping rectangle.

All intermediate offscreens are defined to not exceed the intersection of **x**, **y**, **width** and **height** with the filter region. The filter region and any of the **x**, **y**, **width** and **height** subregions are to be set up such that all offscreens are made big enough to accommodate any pixels which even partly intersect with either the filter region or the x,y,width,height subregions.

'feTile' references a previous filter primitive and then stitches the tiles together based on the **x**, **y**, **width** and **height** values of the referenced filter primitive in order to fill its own filter primitive subregion.

15.8 Light source elements and properties

15.8.1 Introduction

The following sections define the elements that define a light source, **'feDistantLight'**, **'fePointLight'** and **'feSpotLight'**, and property **'lighting-color'**, which defines the color of the light.

15.8.2 Light source 'feDistantLight'

```
<!ENTITY % SVG.feDistantLight.extra.content "" >
<!ENTITY % SVG.feDistantLight.element "INCLUDE" >
<![%SVG.feDistantLight.element;[
<!ENTITY % SVG.feDistantLight.content
    "( %SVG.animate.qname; | %SVG.set.qname;
       %SVG.feDistantLight.extra.content; )*"
>
<!ELEMENT %SVG.feDistantLight.qname; %SVG.feDistan\
tLight.content; >
<!-- end of SVG.feDistantLight.element -->]]>
<!ENTITY % SVG.feDistantLight.attlist "INCLUDE" >
<![%SVG.feDistantLight.attlist;[
<!ATTLIST %SVG.feDistantLight.qname;
    %SVG.Core.attrib;
    azimuth %Number.datatype; #IMPLIED
    elevation %Number.datatype; #IMPLIED
>
```

Attribute definitions:

`azimuth` = "*<number>*"

Direction angle for the light source on the XY plane, in degrees.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
Animatable: yes.

`elevation` = "*<number>*"

Direction angle for the light source on the YZ plane, in degrees.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
Animatable: yes.

15.8.3 Light source 'fePointLight'

```
<!ENTITY % SVG.fePointLight.extra.content "" >
<!ENTITY % SVG.fePointLight.element "INCLUDE" >
<![%SVG.fePointLight.element;[
<!ENTITY % SVG.fePointLight.content
    "( %SVG.animate.qname; | %SVG.set.qname;
       %SVG.fePointLight.extra.content; )*"
>
<!ELEMENT %SVG.fePointLight.qname; %SVG.fePointLight\
.content; >
<!-- end of SVG.fePointLight.element -->]]>
<!ENTITY % SVG.fePointLight.attlist "INCLUDE" >
<![%SVG.fePointLight.attlist;[
<!ATTLIST %SVG.fePointLight.qname;
    %SVG.Core.attrib;
    x %Number.datatype; #IMPLIED
    y %Number.datatype; #IMPLIED
    z %Number.datatype; #IMPLIED
>
```

Attribute definitions:

x = "*<number>*"

X location for the light source in the coordinate system established by attribute **primitiveUnits** on the **'filter'** element.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
Animatable: yes.

y = "*<number>*"

Y location for the light source in the coordinate system established by attribute **primitiveUnits** on the **'filter'** element.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
Animatable: yes.

z = "*<number>*"

Z location for the light source in the coordinate system established by attribute **primitiveUnits** on the **'filter'** element, assuming that, in the initial coordinate system, the positive Z-axis comes out towards the person viewing the content and assuming that one unit along the Z-axis equals one unit in X or Y.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
Animatable: yes.

15.8.4 Light source 'feSpotLight'

```
<!ENTITY % SVG.feSpotLight.extra.content "" >
<!ENTITY % SVG.feSpotLight.element "INCLUDE" >
<![%SVG.feSpotLight.element;[
<!ENTITY % SVG.feSpotLight.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feSpotLight.extra.content; )*"
>
<!ELEMENT %SVG.feSpotLight.qname; %SVG.feSpotLight.co\
ntent; >
<!-- end of SVG.feSpotLight.element -->]]>
<!ENTITY % SVG.feSpotLight.attlist "INCLUDE" >
<![%SVG.feSpotLight.attlist;[
<!ATTLIST %SVG.feSpotLight.qname;
    %SVG.Core.attrib;
    x %Number.datatype; #IMPLIED
    y %Number.datatype; #IMPLIED
    z %Number.datatype; #IMPLIED
    pointsAtX %Number.datatype; #IMPLIED
    pointsAtY %Number.datatype; #IMPLIED
    pointsAtZ %Number.datatype; #IMPLIED
    specularExponent %Number.datatype; #IMPLIED
    limitingConeAngle %Number.datatype; #IMPLIED
>
```

Attribute definitions:

x = "*<number>*"

X location for the light source in the coordinate system established by attribute **primitiveUnits** on the **'filter'** element.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
Animatable: yes.

y = "*<number>*"

Y location for the light source in the coordinate system established by attribute **primitiveUnits** on the **'filter'** element.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
Animatable: yes.

z = "*<number>*"

Z location for the light source in the coordinate system established by attribute **primitiveUnits** on the **'filter'** element, assuming that, in the initial coordinate system, the positive Z-axis comes out towards the person viewing the content and assuming that one unit along the Z-axis equals one unit in X or Y.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
Animatable: yes.

pointsAtX = "*<number>*"

X location in the coordinate system established by attribute **primitiveUnits** on the **'filter'** element of the point at which the light source is pointing.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
Animatable: yes.

pointsAtY = "*<number>*"

Y location in the coordinate system established by attribute **primitiveUnits** on the **'filter'** element of the point at which the light source is pointing.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
Animatable: yes.

pointsAtZ = "*<number>*"

Z location of the point at which the light source is pointing, assuming that, in the initial coordinate system, the positive Z-axis comes out towards the person viewing the content and assuming that one unit along the Z-axis equals one unit in X or Y.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
Animatable: yes.

specularExponent = "*<number>*"

Exponent value controlling the focus for the light source.
If the attribute is not specified, then the effect is as if a value of **1** were specified.

Animatable: yes.

`limitingConeAngle` = "*<number>*"

A limiting cone which restricts the region where the light is projected. No light is projected outside the cone. **limitingConeAngle** represents the angle between the spot light axis (i.e. the axis between the light source and the point to which it is pointing at) and the spot light cone. User agents should apply a smoothing technique such as anti-aliasing at the boundary of the cone.
If no value is specified, then no limiting cone will be applied.
Animatable: yes.

15.8.5 The 'lighting-color' property

The **'lighting-color'** property defines the color of the light source for filter primitives **'feDiffuseLighting'** and **'feSpecularLighting'**.

'lighting-color'

Value:	currentColor | <color> [icc-color(<name>[,<icccolorvalue>]*)] | inherit
Initial:	white
Applies to:	**'feDiffuseLighting'** and **'feSpecularLighting'** elements
Inherited:	no
Percentages:	N/A
Media:	visual
Animatable:	yes

15.9 Filter primitive 'feBlend'

This filter composites two objects together using commonly used imaging software blending modes. It performs a pixel-wise combination of two input images.

```
<!ENTITY % SVG.feBlend.extra.content "" >
<!ENTITY % SVG.feBlend.element "INCLUDE" >
<![%SVG.feBlend.element;[
<!ENTITY % SVG.feBlend.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feBlend.extra.content; )*"
>
<!ELEMENT %SVG.feBlend.qname; %SVG.feBlend.content; >
<!-- end of SVG.feBlend.element -->]]>
<!ENTITY % SVG.feBlend.attlist "INCLUDE" >
<![%SVG.feBlend.attlist;[
<!ATTLIST %SVG.feBlend.qname;
    %SVG.Core.attrib;
    %SVG.FilterColor.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
    in2 CDATA #REQUIRED
    mode ( normal | multiply | screen | darken | lighten ) 'normal'
>
```

Attribute definitions:

`mode` = "*normal | multiply | screen | darken | lighten*"

One of the image blending modes (see table below). Default is: normal.
Animatable: yes.

`in2` = "*(see in attribute)*"

The second input image to the blending operation. This attribute can take on the same values as the **in** attribute.
Animatable: yes.

For all feBlend modes, the result opacity is computed as follows:

```
qr = 1 - (1-qa)*(1-qb)
```

For the compositing formulas below, the following definitions apply:

```
cr = Result color (RGB) - premultiplied
qa = Opacity value at a given pixel for image A
qb = Opacity value at a given pixel for image B
ca = Color (RGB) at a given pixel for image A - premultiplied
cb = Color (RGB) at a given pixel for image B - premultiplied
```

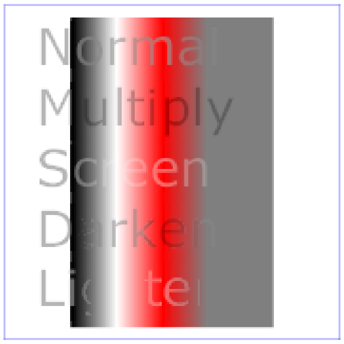
The following table provides the list of available image blending modes:

Image Blending Mode	Formula for computing result color
normal	cr = (1 - qa) * cb + ca
multiply	cr = (1-qa)*cb + (1-qb)*ca + ca*cb
screen	cr = cb + ca - ca * cb
darken	cr = Min ((1 - qa) * cb + ca, (1 - qb) * ca + cb)
lighten	cr = Max ((1 - qa) * cb + ca, (1 - qb) * ca + cb)

'normal' blend mode is equivalent to **operator="over"** on the **'feComposite'** filter primitive, matches the blending method used by **'feMerge'** and matches the simple alpha compositing technique used in SVG for all compositing outside of filter effects.

Example feBlend shows examples of the five blend modes.

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
          "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="5cm" height="5cm" viewBox="0 0 500 500" version="1.1"
     xmlns="http://www.w3.org/2000/svg">
  <title>Example feBlend - Examples of feBlend modes</title>
  <desc>Five text strings blended into a gradient,
        with one text string for each of the five feBlend modes.</desc>
  <defs>
    <linearGradient id="MyGradient" gradientUnits="userSpaceOnUse"
            x1="100" y1="0" x2="300" y2="0">
      <stop offset="0" stop-color="#000000" />
      <stop offset=".33" stop-color="#ffffff" />
      <stop offset=".67" stop-color="#ff0000" />
      <stop offset="1" stop-color="#808080" />
    </linearGradient>
    <filter id="Normal">
      <feBlend mode="normal" in2="BackgroundImage" in="SourceGraphic"/>
    </filter>
    <filter id="Multiply">
      <feBlend mode="multiply" in2="BackgroundImage" in="SourceGraphic"/>
    </filter>
    <filter id="Screen">
      <feBlend mode="screen" in2="BackgroundImage" in="SourceGraphic"/>
    </filter>
    <filter id="Darken">
      <feBlend mode="darken" in2="BackgroundImage" in="SourceGraphic"/>
    </filter>
    <filter id="Lighten">
      <feBlend mode="lighten" in2="BackgroundImage" in="SourceGraphic"/>
    </filter>
  </defs>
  <rect fill="none" stroke="blue"
        x="1" y="1" width="498" height="498"/>
  <g enable-background="new" >
    <rect x="100" y="20" width="300" height="460" fill="url(#MyGradient)" />
    <g font-family="Verdana" font-size="75" fill="#888888" fill-opacity=".6" >
      <text x="50" y="90" filter="url(#Normal)" >Normal</text>
      <text x="50" y="180" filter="url(#Multiply)" >Multiply</text>
      <text x="50" y="270" filter="url(#Screen)" >Screen</text>
      <text x="50" y="360" filter="url(#Darken)" >Darken</text>
      <text x="50" y="450" filter="url(#Lighten)" >Lighten</text>
    </g>
  </g>
</svg>
```



Example feBlend

[View this example as SVG (SVG-enabled browsers only)]

15.10 Filter primitive 'feColorMatrix'

This filter applies a matrix transformation:

```
| R' |     | a00 a01 a02 a03 a04 |   | R |
| G' |     | a10 a11 a12 a13 a14 |   | G |
| B' |  =  | a20 a21 a22 a23 a24 | * | B |
| A' |     | a30 a31 a32 a33 a34 |   | A |
| 1  |     |  0   0   0   0   1  |   | 1 |
```

on the RGBA color and alpha values of every pixel on the input graphics to produce a result with a new set of RGBA color and alpha values.

The calculations are performed on non-premultiplied color values. If the input graphics consists of premultiplied color values, those values are automatically converted into non-premultiplied color values for this operation.

These matrices often perform an identity mapping in the alpha channel. If that is the case, an implementation can avoid the costly undoing and redoing of the premultiplication for all pixels with A = 1.

```
<!ENTITY % SVG.feColorMatrix.extra.content "" >
<!ENTITY % SVG.feColorMatrix.element "INCLUDE" >
<![%SVG.feColorMatrix.element;[
<!ENTITY % SVG.feColorMatrix.content
    "( %SVG.animate.qname; | %SVG.set.qname;
       %SVG.feColorMatrix.extra.content; )*"
>
<!ELEMENT %SVG.feColorMatrix.qname; %SVG.feColorMat\
rix.content; >
<!-- end of SVG.feColorMatrix.element -->]]>
<!ENTITY % SVG.feColorMatrix.attlist "INCLUDE" >
<![%SVG.feColorMatrix.attlist;[
<!ATTLIST %SVG.feColorMatrix.qname;
    %SVG.Core.attrib;
    %SVG.FilterColor.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
    type ( matrix | saturate | hueRotate | luminanceToAlpha ) 'matrix'
    values CDATA #IMPLIED
>
```

Attribute definitions:

`type` = "*matrix | saturate | hueRotate | luminanceToAlpha*"

Indicates the type of matrix operation. The keyword **matrix** indicates that a full 5x4 matrix of values will be provided. The other keywords represent convenience shortcuts to allow commonly used color operations to be performed without specifying a complete matrix.
Animatable: yes.

`values` = "*list of <number>s*"

The contents of **values** depends on the value of attribute **type**:

- For **type="matrix"**, **values** is a list of 20 matrix values (a00 a01 a02 a03 a04 a10 a11 ... a34), separated by whitespace and/or a comma. For example, the identity matrix could be expressed as:

```
type="matrix"
values="1 0 0 0 0  0 1 0 0 0  0 0 1 0 0  0 0 0 1 0"
```

- For **type="saturate"**, **values** is a single real number value (0 to 1). A **saturate** operation is equivalent to the following matrix operation:

```
| R' |     |0.213+0.787s  0.715-0.715s  0.072-0.072s 0  0 |   | R |
| G' |     |0.213-0.213s  0.715+0.285s  0.072-0.072s 0  0 |   | G |
| B' |  =  |0.213-0.213s  0.715-0.715s  0.072+0.928s 0  0 | * | B |
| A' |     |           0            0             0  1  0 |   | A |
| 1  |     |           0            0             0  0  1 |   | 1 |
```

- For **type="hueRotate"**, **values** is a single one real number value (degrees). A **hueRotate** operation is equivalent to the following matrix operation:

```
| R' |     | a00  a01  a02  0  0 |   | R |
| G' |     | a10  a11  a12  0  0 |   | G |
| B' |  =  | a20  a21  a22  0  0 | * | B |
| A' |     | 0    0    0    1  0 |   | A |
| 1  |     | 0    0    0    0  1 |   | 1 |
```

where the terms a00, a01, etc. are calculated as follows:

```
| a00 a01 a02 |    [+0.213 +0.715 +0.072]
| a10 a11 a12 | =  [+0.213 +0.715 +0.072] +
| a20 a21 a22 |    [+0.213 +0.715 +0.072]
                        [+0.787 -0.715 -0.072]
cos(hueRotate value) *  [-0.213 +0.285 -0.072] +
                        [-0.213 -0.715 +0.928]
                        [-0.213 -0.715+0.928]
sin(hueRotate value) *  [+0.143 +0.140-0.283]
                        [-0.787 +0.715+0.072]
```

Thus, the upper left term of the hue matrix turns out to be:

```
.213 + cos(hueRotate value)*.787 - sin(hueRotate value)*.213
```

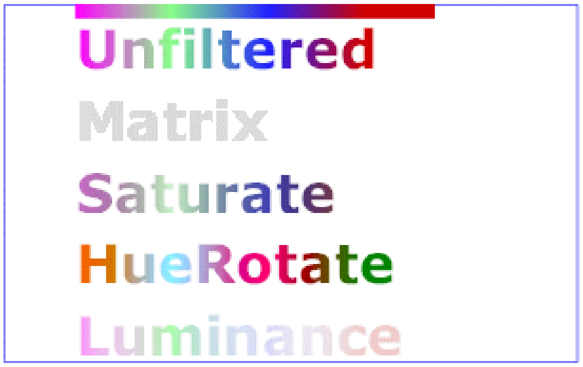
- For **type="luminanceToAlpha"**, **values** is not applicable. A **luminanceToAlpha** operation is equivalent to the following matrix operation:

```
   | R' |     |      0        0        0  0  0 |   | R |
   | G' |     |      0        0        0  0  0 |   | G |
   | B' |  =  |      0        0        0  0  0 | * | B |
   | A' |     | 0.2125   0.7154   0.0721  0  0 |   | A |
   | 1  |     |      0        0        0  0  1 |   | 1 |
```

If the attribute is not specified, then the default behavior depends on the value of attribute **type**. If **type="matrix"**, then this attribute defaults to the identity matrix. If **type="saturate"**, then this attribute defaults to the value **1**, which results in the identify matrix. If **type="hueRotate"**, then this attribute defaults to the value **0**, which results in the identify matrix.
Animatable: yes.

Example feColorMatrix shows examples of the four types of feColorMatrix operations.

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
          "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="8cm" height="5cm" viewBox="0 0 800 500" version="1.1"
     xmlns="http://www.w3.org/2000/svg">
  <title>Example feColorMatrix - Examples of feColorMatrix operations</title>
  <desc>Five text strings showing the effects of feColorMatrix:
        an unfiltered text string acting as a reference,
        use of the feColorMatrix matrix option,
        use of the feColorMatrix saturate option,
        use of the feColorMatrix hueRotate option,
        and use of the feColorMatrix luminanceToAlpha option.</desc>
  <defs>
    <linearGradient id="MyGradient" gradientUnits="userSpaceOnUse"
            x1="100" y1="0" x2="500" y2="0">
      <stop offset="0" stop-color="#ff00ff" />
      <stop offset=".33" stop-color="#88ff88" />
      <stop offset=".67" stop-color="#2020ff" />
      <stop offset="1" stop-color="#d00000" />
    </linearGradient>
    <filter id="Matrix" filterUnits="objectBoundingBox"
            x="0%" y="0%" width="100%" height="100%">
      <feColorMatrix type="matrix" in="SourceGraphic"
           values=".33 .33 .33 0 0
                   .33 .33 .33 0 0
                   .33 .33 .33 0 0
                   .33 .33 .33 0 0"/>
    </filter>
    <filter id="Saturate40" filterUnits="objectBoundingBox"
            x="0%" y="0%" width="100%" height="100%">
      <feColorMatrix type="saturate" in="SourceGraphic" values="0.4"/>
    </filter>
    <filter id="HueRotate90" filterUnits="objectBoundingBox"
            x="0%" y="0%" width="100%" height="100%">
      <feColorMatrix type="hueRotate" in="SourceGraphic" values="90"/>
    </filter>
    <filter id="LuminanceToAlpha" filterUnits="objectBoundingBox"
            x="0%" y="0%" width="100%" height="100%">
      <feColorMatrix type="luminanceToAlpha" in="SourceGraphic" result="a"/>
      <feComposite in="SourceGraphic" in2="a" operator="in" />
    </filter>
  </defs>
  <rect fill="none" stroke="blue"
        x="1" y="1" width="798" height="498"/>
  <g font-family="Verdana" font-size="75"
            font-weight="bold" fill="url(#MyGradient)" >
    <rect x="100" y="0" width="500" height="20" />
    <text x="100" y="90">Unfiltered</text>
    <text x="100" y="190" filter="url(#Matrix)" >Matrix</text>
    <text x="100" y="290" filter="url(#Saturate40)" >Saturate</text>
    <text x="100" y="390" filter="url(#HueRotate90)" >HueRotate</text>
    <text x="100" y="490" filter="url(#LuminanceToAlpha)" >Luminance</text>
  </g>
</svg>
```



Example feColorMatrix

View this example as SVG (SVG-enabled browsers only)

15.11 Filter primitive 'feComponentTransfer'

This filter primitive performs component-wise remapping of data as follows:

```
R' = feFuncR( R )
```

```
G' = feFuncG( G )
B' = feFuncB( B )
A' = feFuncA( A )
```

for every pixel. It allows operations like brightness adjustment, contrast adjustment, color balance or thresholding.

The calculations are performed on non-premultiplied color values. If the input graphics consists of premultiplied color values, those values are automatically converted into non-premultiplied color values for this operation. (Note that the undoing and redoing of the premultiplication can be avoided if feFuncA is the identity transform and all alpha values on the source graphic are set to 1.)

```
lt;!ENTITY % SVG.feComponentTransfer.extra.content "" >
<!ENTITY % SVG.feComponentTransfer.element "INCLUDE" >
<![%SVG.feComponentTransfer.element;[
<!ENTITY % SVG.feComponentTransfer.content
    "( %SVG.feFuncR.qname;?, %SVG.feFuncG.qname;?, %SVG.feFuncB.qname;?,
       %SVG.feFuncA.qname;? %SVG.feComponentTransfer.e\
xtra.content; )"
>
<!ELEMENT %SVG.feComponentTransfer.qname; %SV\
G.feComponentTransfer.content; >
<!-- end of SVG.feComponentTransfer.element -->]]>
<!ENTITY % SVG.feComponentTransfer.attlist "INCLUDE" >
<![%SVG.feComponentTransfer.attlist;[
<!ATTLIST %SVG.feComponentTransfer.qname;
    %SVG.Core.attrib;
    %SVG.FilterColor.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
>

<!-- feFuncR: Filter Effect Function Red Element ....... -->
<!ENTITY % SVG.feFuncR.extra.content "" >
<!ENTITY % SVG.feFuncR.element "INCLUDE" >
<![%SVG.feFuncR.element;[
<!ENTITY % SVG.feFuncR.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feFuncR.extra.content; )*"
>
<!ELEMENT %SVG.feFuncR.qname; %SVG.feFuncR.content; >
<!-- end of SVG.feFuncR.element -->]]>
<!ENTITY % SVG.feFuncR.attlist "INCLUDE" >
<![%SVG.feFuncR.attlist;[
<!ATTLIST %SVG.feFuncR.qname;
    %SVG.Core.attrib;
    type ( identity | table | discrete | linear | gamma ) #REQUIRED
    tableValues CDATA #IMPLIED
    slope %Number.datatype; #IMPLIED
    intercept %Number.datatype; #IMPLIED
    amplitude %Number.datatype; #IMPLIED
    exponent %Number.datatype; #IMPLIED
   offset %Number.datatype; #IMPLIED
>
<!-- end of SVG.feFuncR.attlist -->]]>
<!-- feFuncG: Filter Effect Function Green Element ..... -->
<!ENTITY % SVG.feFuncG.extra.content "" >
<!ENTITY % SVG.feFuncG.element "INCLUDE" >
<![%SVG.feFuncG.element;[
<!ENTITY % SVG.feFuncG.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feFuncG.extra.content; )*"
>
<!ELEMENT %SVG.feFuncG.qname; %SVG.feFuncG.content; >
<!-- end of SVG.feFuncG.element -->]]>
<!ENTITY % SVG.feFuncG.attlist "INCLUDE" >
<![%SVG.feFuncG.attlist;[
<!ATTLIST %SVG.feFuncG.qname;
    %SVG.Core.attrib;
    type ( identity | table | discrete | linear | gamma ) #REQUIRED
    tableValues CDATA #IMPLIED
    slope %Number.datatype; #IMPLIED
    intercept %Number.datatype; #IMPLIED
    amplitude %Number.datatype; #IMPLIED
    exponent %Number.datatype; #IMPLIED
    offset %Number.datatype; #IMPLIED
>
<!-- end of SVG.feFuncG.attlist -->]]>
<!-- feFuncB: Filter Effect Function Blue Element ...... -->
<!ENTITY % SVG.feFuncB.extra.content "" >
<!ENTITY % SVG.feFuncB.element "INCLUDE" >
<![%SVG.feFuncB.element;[
<!ENTITY % SVG.feFuncB.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feFuncB.extra.content; )*"
>
<!ELEMENT %SVG.feFuncB.qname; %SVG.feFuncB.content; >
<!-- end of SVG.feFuncB.element -->]]>
<!ENTITY % SVG.feFuncB.attlist "INCLUDE" >
<![%SVG.feFuncB.attlist;[
<!ATTLIST %SVG.feFuncB.qname;
    %SVG.Core.attrib;
    type ( identity | table | discrete | linear | gamma ) #REQUIRED
    tableValues CDATA #IMPLIED
    slope %Number.datatype; #IMPLIED
    intercept %Number.datatype; #IMPLIED
    amplitude %Number.datatype; #IMPLIED
    exponent %Number.datatype; #IMPLIED
    offset %Number.datatype; #IMPLIED
>
<!-- end of SVG.feFuncB.attlist -->]]>
<!-- feFuncA: Filter Effect Function Alpha Element ..... -->
<!ENTITY % SVG.feFuncA.extra.content "" >
<!ENTITY % SVG.feFuncA.element "INCLUDE" >
<![%SVG.feFuncA.element;[
<!ENTITY % SVG.feFuncA.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feFuncA.extra.content; )*"
>
<!ELEMENT %SVG.feFuncA.qname; %SVG.feFuncA.content; >
<!-- end of SVG.feFuncA.element -->]]>
<!ENTITY % SVG.feFuncA.attlist "INCLUDE" >
<![%SVG.feFuncA.attlist;[
<!ATTLIST %SVG.feFuncA.qname;
    %SVG.Core.attrib;
    type ( identity | table | discrete | linear | gamma ) #REQUIRED
    tableValues CDATA #IMPLIED
    slope %Number.datatype; #IMPLIED
    intercept %Number.datatype; #IMPLIED
    amplitude %Number.datatype; #IMPLIED
    exponent %Number.datatype; #IMPLIED
    offset %Number.datatype; #IMPLIED
>
```

The specification of the transfer functions is defined by the sub-elements to 'feComponentTransfer':

'feFuncR', transfer function for red component of the input graphic
'feFuncG', transfer function for green component of the input graphic
'feFuncB', transfer function for blue component of the input graphic
'feFuncA', transfer function for alpha component of the input graphic

The attributes below apply to sub-elements **'feFuncR'**, **'feFuncG'**, **'feFuncB'** and **'feFuncA'** define the transfer functions.

Attribute definitions:

`type` = "*identity* | *table* | *discrete* | *linear* | *gamma*"

Indicates the type of component transfer function. The type of function determines the applicability of the other attributes.

- For **identity**:

  C' = C

- For **table**, the function is defined by linear interpolation into a lookup table by attribute **tableValues**, which provides a list of *n+1* values (i.e., v_0 to v_n) in order to identify *n* interpolation ranges. Interpolations use the following formula.

  For a value C pick a k such that:

  k/N <= C < (k+1)/N

  The result C' is given by:

  $$C' = v_k + (C - k/N)*N * (v_{k+1} - v_k)$$

- For **discrete**, the function is defined by the step function defined by attribute **tableValues**, which provides a list of *n* values (i.e., v_0 to v_{n-1}) in order to identify a step function consisting of *n* steps. The step function is defined by the following formula.

  For a value C pick a k such that:

  k/N <= C < (k+1)/N

  The result C' is given by:

  $$C' = v_k$$

- For **linear**, the function is defined by the following linear equation:

  C' = **slope** * C + **intercept**

- For **gamma**, the function is defined by the following exponential function:

  C' = **amplitude** * pow(C, **exponent**) + **offset**

Animatable: yes.

`tableValues` = "*(list of <number>s)*"

When **type="table"**, the list of <number>s *v0,v1,...vn*, separated by white space and/or a comma, which define the lookup table. An empty list results in an identity transfer function. If the attribute is not specified, then the effect is as if an empty list were provided. If the attribute is not specified, then the effect is as if an empty list were provided.
Animatable: yes.

`slope` = "*<number>*"

When **type="linear"**, the slope of the linear function.
If the attribute is not specified, then the effect is as if a value of **1** were specified.
Animatable: yes.

`intercept` = "*<number>*"

When **type="linear"**, the intercept of the linear function.
If the attribute is not specified, then the effect is as if a value of **0** were specified.

Animatable: yes.

`amplitude` = "*<number>*"

When **type="gamma"**, the amplitude of the gamma function.
If the attribute is not specified, then the effect is as if a value of **1** were specified.
Animatable: yes.

`exponent` = "*<number>*"

When **type="gamma"**, the exponent of the gamma function.
If the attribute is not specified, then the effect is as if a value of **1** were specified.
Animatable: yes.

`offset` = "*<number>*"

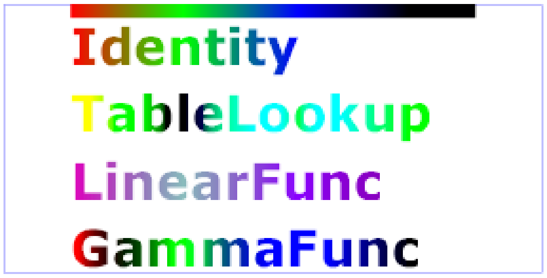
When **type="gamma"**, the offset of the gamma function.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
Animatable: yes.

Attributes defined elsewhere:

%stdAttrs;, %filter_primitive_attributes_with_in;, %PresentationAttributes-FilterPrimitives;.

Example feComponentTransfer shows examples of the four types of feComponentTransfer operations.

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
          "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="8cm" height="4cm" viewBox="0 0 800 400" version="1.1"
     xmlns="http://www.w3.org/2000/svg">
  <title>Example feComponentTransfer - Examples of feComponentTransfer operations</title>
  <desc>Four text strings showing the effects of feComponentTransfer:
        an identity function acting as a reference,
        use of the feComponentTransfer table option,
        use of the feComponentTransfer linear option,
        and use of the feComponentTransfer gamma option.</desc>
  <defs>
    <linearGradient id="MyGradient" gradientUnits="userSpaceOnUse"
            x1="100" y1="0" x2="600" y2="0">
      <stop offset="0" stop-color="#ff0000" />
      <stop offset=".33" stop-color="#00ff00" />
      <stop offset=".67" stop-color="#0000ff" />
      <stop offset="1" stop-color="#000000" />
    </linearGradient>
    <filter id="Identity" filterUnits="objectBoundingBox"
            x="0%" y="0%" width="100%" height="100%">
      <feComponentTransfer>
        <feFuncR type="identity"/>
        <feFuncG type="identity"/>
        <feFuncB type="identity"/>
        <feFuncA type="identity"/>
      </feComponentTransfer>
    </filter>
    <filter id="Table" filterUnits="objectBoundingBox"
            x="0%" y="0%" width="100%" height="100%">
      <feComponentTransfer>
        <feFuncR type="table" tableValues="0 0 1 1"/>
        <feFuncG type="table" tableValues="1 1 0 0"/>
        <feFuncB type="table" tableValues="0 1 1 0"/>
      </feComponentTransfer>
    </filter>
    <filter id="Linear" filterUnits="objectBoundingBox"
            x="0%" y="0%" width="100%" height="100%">
      <feComponentTransfer>
        <feFuncR type="linear" slope=".5" intercept=".25"/>
        <feFuncG type="linear" slope=".5" intercept="0"/>
        <feFuncB type="linear" slope=".5" intercept=".5"/>
      </feComponentTransfer>
    </filter>
    <filter id="Gamma" filterUnits="objectBoundingBox"
            x="0%" y="0%" width="100%" height="100%">
      <feComponentTransfer>
        <feFuncR type="gamma" amplitude="2" exponent="5" offset="0"/>
        <feFuncG type="gamma" amplitude="2" exponent="3" offset="0"/>
        <feFuncB type="gamma" amplitude="2" exponent="1" offset="0"/>
      </feComponentTransfer>
    </filter>
  </defs>
  <rect fill="none" stroke="blue"
        x="1" y="1" width="798" height="398"/>
  <g font-family="Verdana" font-size="75"
            font-weight="bold" fill="url(#MyGradient)" >
    <rect x="100" y="0" width="600" height="20" />
    <text x="100" y="90">Identity</text>
    <text x="100" y="190" filter="url(#Table)" >TableLookup</text>
    <text x="100" y="290" filter="url(#Linear)" >LinearFunc</text>
    <text x="100" y="390" filter="url(#Gamma)" >GammaFunc</text>
  </g>
</svg>
```



Example feComponentTransfer

View this example as SVG (SVG-enabled browsers only)

15.12 Filter primitive 'feComposite'

This filter performs the combination of the two input images pixel-wise in image space using one of the Porter-Duff [PORTERDUFF] compositing operations: *over, in, atop, out, xor*. Additionally, a component-wise *arithmetic* operation (with the result clamped between [0..1]) can be applied.

The *arithmetic* operation is useful for combining the output from the **'feDiffuseLighting'** and **'feSpecularLighting'** filters with texture data. It is also useful for implementing *dissolve*. If the *arithmetic* operation is chosen, each result pixel is computed using the following formula:

```
result = k1*i1*i2 + k2*i1 + k3*i2 + k4
```

For this filter primitive, the extent of the resulting image might grow as described in the section that describes the filter primitive subregion.

```
<!ENTITY % SVG.feComposite.extra.content "" >
<!ENTITY % SVG.feComposite.element "INCLUDE" >
<![%SVG.feComposite.element;[
<!ENTITY % SVG.feComposite.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feComposite.extra.content; )*"
>
<!ELEMENT %SVG.feComposite.qname; %SVG.feComposite.co\
ntent; >
<!-- end of SVG.feComposite.element -->]]>
<!ENTITY % SVG.feComposite.attlist "INCLUDE" >
<![%SVG.feComposite.attlist;[
<!ATTLIST %SVG.feComposite.qname;
    %SVG.Core.attrib;
    %SVG.FilterColor.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
    in2 CDATA #REQUIRED
    operator ( over | in | out | atop | xor | arithmetic ) 'over'
    k1 %Number.datatype; #IMPLIED
    k2 %Number.datatype; #IMPLIED
    k3 %Number.datatype; #IMPLIED
    k4 %Number.datatype; #IMPLIED
>
```

Attribute definitions:

`operator` = "*over* | *in* | *out* | *atop* | *xor* | *arithmetic*"

The compositing operation that is to be performed. All of the **operator** types except **arithmetic** match the

correspond operation as described in [PORTERDUFF]. The **arithmetic** operator is described above.
Animatable: yes.

k1 = "*<number>*"
Only applicable if **operator="arithmetic"**.
If the attribute is not specified, the effect is as if a value of "0" were specified.
Animatable: yes.

k2 = "*<number>*"
Only applicable if **operator="arithmetic"**.
If the attribute is not specified, the effect is as if a value of "0" were specified.
Animatable: yes.

k3 = "*<number>*"
Only applicable if **operator="arithmetic"**.
If the attribute is not specified, the effect is as if a value of "0" were specified.
Animatable: yes.

k4 = "*<number>*"
Only applicable if **operator="arithmetic"**.
If the attribute is not specified, the effect is as if a value of "0" were specified.
Animatable: yes.

in2 = "*(see in attribute)*"
The second input image to the compositing operation. This attribute can take on the same values as the **in** attribute.
Animatable: yes.

Example feComposite shows examples of the six types of feComposite operations. It also shows two different techniques to using the BackgroundImage as part of the compositing operation.

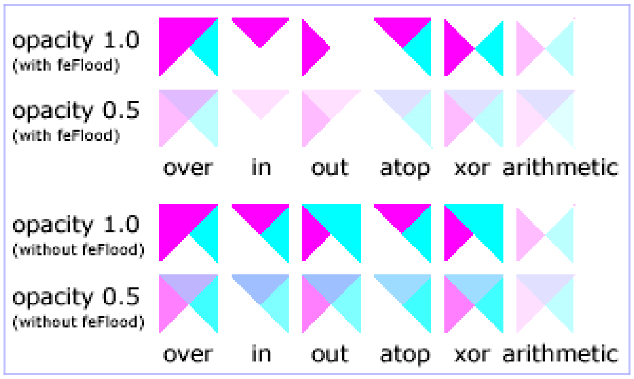
The first two rows render bluish triangles into the background. A filter is applied which composites reddish triangles into the bluish triangles using one of the compositing operations. The result from compositing is drawn onto an opaque white temporary surface, and then that result is written to the canvas. (The opaque white temporary surface obliterates the original bluish triangle.)

The last two rows apply the same compositing operations of reddish triangles into bluish triangles. However, the compositing result is directly blended into the canvas (the opaque white temporary surface technique is not used). In some cases, the results are different than when a temporary opaque white surface is used. The original bluish triangle from the background shines through wherever the compositing operation results in completely transparent pixel. In other cases, the result from compositing is blended into the bluish triangle, resulting in a different final color value.

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
          "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="330" height="195" viewBox="0 0 1100 650" version="1.1"
     xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Example feComposite - Examples of feComposite operations</title>
  <desc>Four rows of six pairs of overlapping triangles depicting
        the six different feComposite operators under different
        opacity values and different clearing of the background.</desc>
    <defs>
    <desc>Define two sets of six filters for each of the six compositing operators.
          The first set wipes out the background image by flooding with opaque white.
          The second set does not wipe out the background, with the result
          that the background sometimes shines through and is other cases
          is blended into itself (i.e., "double-counting").</desc>
    <filter id="overFlood" filterUnits="objectBoundingBox" x="-5%" y="-5%" width="110%" height="110%">
      <feFlood flood-color="#ffffff" flood-opacity="1" result="flood"/>
      <feComposite in="SourceGraphic" in2="BackgroundImage" operator="over" result="comp"/>
      <feMerge> <feMergeNode in="flood"/> <feMergeNode in="comp"/> </feMerge>
    </filter>
    <filter id="inFlood" filterUnits="objectBoundingBox" x="-5%" y="-5%" width="110%" height="110%">
      <feFlood flood-color="#ffffff" flood-opacity="1" result="flood"/>
      <feComposite in="SourceGraphic" in2="BackgroundImage" operator="in" result="comp"/>
      <feMerge> <feMergeNode in="flood"/> <feMergeNode in="comp"/> </feMerge>
    </filter>
    <filter id="outFlood" filterUnits="objectBoundingBox" x="-5%" y="-5%" width="110%" height="110%">
      <feFlood flood-color="#ffffff" flood-opacity="1" result="flood"/>
      <feComposite in="SourceGraphic" in2="BackgroundImage" operator="out" result="comp"/>
      <feMerge> <feMergeNode in="flood"/> <feMergeNode in="comp"/> </feMerge>
    </filter>
    <filter id="atopFlood" filterUnits="objectBoundingBox" x="-5%" y="-5%" width="110%" height="110%">
      <feFlood flood-color="#ffffff" flood-opacity="1" result="flood"/>
      <feComposite in="SourceGraphic" in2="BackgroundImage" operator="atop" result="comp"/>
      <feMerge> <feMergeNode in="flood"/> <feMergeNode in="comp"/> </feMerge>
    </filter>
    <filter id="xorFlood" filterUnits="objectBoundingBox" x="-5%" y="-5%" width="110%" height="110%">
```

```
      <feFlood flood-color="#ffffff" flood-opacity="1" result="flood"/>
      <feComposite in="SourceGraphic" in2="BackgroundImage" operator="xor" result="comp"/>
      <feMerge> <feMergeNode in="flood"/> <feMergeNode in="comp"/> </feMerge>
    </filter>
    <filter id="arithmeticFlood" filterUnits="objectBoundingBox"
            x="-5%" y="-5%" width="110%" height="110%">
      <feFlood flood-color="#ffffff" flood-opacity="1" result="flood"/>
      <feComposite in="SourceGraphic" in2="BackgroundImage" result="comp"
                   operator="arithmetic" k1=".5" k2=".5" k3=".5" k4=".5"/>
      <feMerge> <feMergeNode in="flood"/> <feMergeNode in="comp"/> </feMerge>
    </filter>
    <filter id="overNoFlood" filterUnits="objectBoundingBox" x="-5%" y="-5%" width="110%" height="110%">
      <feComposite in="SourceGraphic" in2="BackgroundImage" operator="over" result="comp"/>
    </filter>
    <filter id="inNoFlood" filterUnits="objectBoundingBox" x="-5%" y="-5%" width="110%" height="110%">
      <feComposite in="SourceGraphic" in2="BackgroundImage" operator="in" result="comp"/>
    </filter>
    <filter id="outNoFlood" filterUnits="objectBoundingBox" x="-5%" y="-5%" width="110%" height="110%">
      <feComposite in="SourceGraphic" in2="BackgroundImage" operator="out" result="comp"/>
    </filter>
    <filter id="atopNoFlood" filterUnits="objectBoundingBox" x="-5%" y="-5%" width="110%" height="110%">
      <feComposite in="SourceGraphic" in2="BackgroundImage" operator="atop" result="comp"/>
    </filter>
    <filter id="xorNoFlood" filterUnits="objectBoundingBox" x="-5%" y="-5%" width="110%" height="110%">
      <feComposite in="SourceGraphic" in2="BackgroundImage" operator="xor" result="comp"/>
    </filter>
    <filter id="arithmeticNoFlood" filterUnits="objectBoundingBox"
            x="-5%" y="-5%" width="110%" height="110%">
      <feComposite in="SourceGraphic" in2="BackgroundImage" result="comp"
                   operator="arithmetic" k1=".5" k2=".5" k3=".5" k4=".5"/>
    </filter>
    <path id="Blue100" d="M 0 0 L 100 0 L 100 100 z" fill="#00ffff" />
    <path id="Red100" d="M 0 0 L 0 100 L 100 0 z" fill="#ff00ff" />
    <path id="Blue50" d="M 0 125 L 100 125 L 100 225 z" fill="#00ffff" fill-opacity=".5" />
    <path id="Red50" d="M 0 125 L 0 225 L 100 125 z" fill="#ff00ff" fill-opacity=".5" />
    <g id="TwoBlueTriangles">
      <use xlink:href="#Blue100"/>
      <use xlink:href="#Blue50"/>
    </g>
    <g id="BlueTriangles">
      <use transform="translate(275,25)" xlink:href="#TwoBlueTriangles"/>
      <use transform="translate(400,25)" xlink:href="#TwoBlueTriangles"/>
      <use transform="translate(525,25)" xlink:href="#TwoBlueTriangles"/>
      <use transform="translate(650,25)" xlink:href="#TwoBlueTriangles"/>
      <use transform="translate(775,25)" xlink:href="#TwoBlueTriangles"/>
      <use transform="translate(900,25)" xlink:href="#TwoBlueTriangles"/>
    </g>
  </defs>
  <rect fill="none" stroke="blue" x="1" y="1" width="1098" height="648"/>
  <g font-family="Verdana" font-size="40" shape-rendering="crispEdges">
    <desc>Render the examples using the filters that draw on top of
          an opaque white surface, thus obliterating the background.</desc>
    <g enable-background="new">
      <text x="15" y="75">opacity 1.0</text>
      <text x="15" y="115" font-size="27">(with feFlood)</text>
      <text x="15" y="200">opacity 0.5</text>
      <text x="15" y="240" font-size="27">(with feFlood)</text>
      <use xlink:href="#BlueTriangles"/>
      <g transform="translate(275,25)">
        <use xlink:href="#Red100" filter="url(#overFlood)" />
        <use xlink:href="#Red50" filter="url(#overFlood)" />
        <text x="5" y="275">over</text>
      </g>
      <g transform="translate(400,25)">
        <use xlink:href="#Red100" filter="url(#inFlood)" />
        <use xlink:href="#Red50" filter="url(#inFlood)" />
        <text x="35" y="275">in</text>
      </g>
      <g transform="translate(525,25)">
        <use xlink:href="#Red100" filter="url(#outFlood)" />
        <use xlink:href="#Red50" filter="url(#outFlood)" />
        <text x="15" y="275">out</text>
      </g>
      <g transform="translate(650,25)">
        <use xlink:href="#Red100" filter="url(#atopFlood)" />
        <use xlink:href="#Red50" filter="url(#atopFlood)" />
        <text x="10" y="275">atop</text>
      </g>
      <g transform="translate(775,25)">
        <use xlink:href="#Red100" filter="url(#xorFlood)" />
        <use xlink:href="#Red50" filter="url(#xorFlood)" />
        <text x="15" y="275">xor</text>
      </g>
      <g transform="translate(900,25)">
        <use xlink:href="#Red100" filter="url(#arithmeticFlood)" />
        <use xlink:href="#Red50" filter="url(#arithmeticFlood)" />
        <text x="-25" y="275">arithmetic</text>
      </g>
    </g>
    <g transform="translate(0,325)" enable-background="new">
    <desc>Render the examples using the filters that do not obliterate
          the background, thus sometimes causing the background to continue
          to appear in some cases, and in other cases the background
          image blends into itself ("double-counting").</desc>
      <text x="15" y="75">opacity 1.0</text>
      <text x="15" y="115" font-size="27">(without feFlood)</text>
      <text x="15" y="200">opacity 0.5</text>
      <text x="15" y="240" font-size="27">(without feFlood)</text>
```

```
          <use xlink:href="#BlueTriangles"/>
          <g transform="translate(275,25)">
            <use xlink:href="#Red100" filter="url(#overNoFlood)" />
            <use xlink:href="#Red50" filter="url(#overNoFlood)" />
            <text x="5" y="275">over</text>
          </g>
          <g transform="translate(400,25)">
            <use xlink:href="#Red100" filter="url(#inNoFlood)" />
            <use xlink:href="#Red50" filter="url(#inNoFlood)" />
            <text x="35" y="275">in</text>
          </g>
          <g transform="translate(525,25)">
            <use xlink:href="#Red100" filter="url(#outNoFlood)" />
            <use xlink:href="#Red50" filter="url(#outNoFlood)" />
            <text x="15" y="275">out</text>
          </g>
          <g transform="translate(650,25)">
            <use xlink:href="#Red100" filter="url(#atopNoFlood)" />
            <use xlink:href="#Red50" filter="url(#atopNoFlood)" />
            <text x="10" y="275">atop</text>
          </g>
          <g transform="translate(775,25)">
            <use xlink:href="#Red100" filter="url(#xorNoFlood)" />
            <use xlink:href="#Red50" filter="url(#xorNoFlood)" />
            <text x="15" y="275">xor</text>
          </g>
          <g transform="translate(900,25)">
            <use xlink:href="#Red100" filter="url(#arithmeticNoFlood)" />
            <use xlink:href="#Red50" filter="url(#arithmeticNoFlood)" />
            <text x="-25" y="275">arithmetic</text>
          </g>
        </g>
      </g>
    </svg>
```



Example feComposite

[View this example as SVG (SVG-enabled browsers only)]

15.13 Filter primitive 'feConvolveMatrix'

feConvolveMatrix applies a matrix convolution filter effect. A convolution combines pixels in the input image with neighboring pixels to produce a resulting image. A wide variety of imaging operations can be achieved through convolutions, including blurring, edge detection, sharpening, embossing and beveling.

A matrix convolution is based on an n-by-m matrix (the convolution kernel) which describes how a given pixel value in the input image is combined with its neighboring pixel values to produce a resulting pixel value. Each result pixel is determined by applying the kernel matrix to the corresponding source pixel and its neighboring pixels. The basic convolution formula which is applied to each color value for a given pixel is:

```
RESULT$_{X,Y}$ = (
                 SUM $_{I=0 \text{ to } [orderY-1]}$ {
                   SUM $_{J=0 \text{ to } [orderX-1]}$ {
                     SOURCE $_{X-targetX+J,\ Y-targetY+I}$ *  kernelMatrix$_{orderX-J-1,\ orderY-I-1}$
                   }
                 }
```

```
                  ) /  divisor +  bias
```

where "orderX" and "orderY" represent the X and Y values for the **order** attribute, "targetX" represents the value of the **targetX** attribute, "targetY" represents the value of the **targetY** attribute, "kernelMatrix" represents the value of the **kernelMatrix** attribute, "divisor" represents the value of the **divisor** attribute, and "bias" represents the value of the **bias** attribute.

Note in the above formulas that the values in the kernel matrix are applied such that the kernel matrix is rotated 180 degrees relative to the source and destination images in order to match convolution theory as described in many computer graphics textbooks.

To illustrate, suppose you have a input image which is 5 pixels by 5 pixels, whose color values for one of the color channels are as follows:

```
  0  20  40 235 235
100 120 140 235 235
200 220 240 235 235
225 225 255 255 255
225 225 255 255 255
```

and you define a 3-by-3 convolution kernel as follows:

```
1 2 3
4 5 6
7 8 9
```

Let's focus on the color value at the second row and second column of the image (source pixel value is 120). Assuming the simplest case (where the input image's pixel grid aligns perfectly with the kernel's pixel grid) and assuming default values for attributes **divisor**, **targetX** and **targetY**, then resulting color value will be:

```
(9*  0 + 8* 20 + 7* 40 +
6*100 + 5*120 + 4*140 +
3*200 + 2*220 + 1*240) / (9+8+7+6+5+4+3+2+1)
```

Because they operate on pixels, matrix convolutions are inherently resolution-dependent. To make **'feConvolveMatrix** produce resolution-independent results, an explicit value should be provided for either the **filterRes** attribute on the **'filter'** element and/or attribute **kernelUnitLength**.

kernelUnitLength, in combination with the other attributes, defines an implicit pixel grid in the filter effects coordinate system (i.e., the coordinate system established by the **primitiveUnits** attribute). If the pixel grid established by **kernelUnitLength** is not scaled to match the pixel grid established by attribute **filterRes** (implicitly or explicitly), then the input image will be temporarily rescaled to match its pixels with **kernelUnitLength**. The convolution happens on the resampled image. After applying the convolution, the image is resampled back to the original resolution.

When the image must be resampled to match the coordinate system defined by **kernelUnitLength** prior to convolution, or resampled to match the device coordinate system after convolution, it is recommended that high quality viewers make use of appropriate interpolation techniques, for example bilinear or bicubic. Depending on the speed of the available interpolents, this choice may be affected by the **'image-rendering'** property setting. Note that implementations might choose approaches that minimize or eliminate resampling when not necessary to produce proper results, such as when the document is zoomed out such that **kernelUnitLength** is considerably smaller than a device pixel.

```
<!ENTITY % SVG.feConvolveMatrix.extra.content "" >
<!ENTITY % SVG.feConvolveMatrix.element "INCLUDE" >
<![%SVG.feConvolveMatrix.element;[
<!ENTITY % SVG.feConvolveMatrix.content
    "( %SVG.animate.qname; | %SVG.set.qname;
       %SVG.feConvolveMatrix.extra.content; )*"
>
<!ELEMENT %SVG.feConvolveMatrix.qname; %SVG.feCo\
nvolveMatrix.content; >
<!-- end of SVG.feConvolveMatrix.element -->]]>
<!ENTITY % SVG.feConvolveMatrix.attlist "INCLUDE" >
<![%SVG.feConvolveMatrix.attlist;[
<!ATTLIST %SVG.feConvolveMatrix.qname;
    %SVG.Core.attrib;
    %SVG.FilterColor.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
    order %NumberOptionalNumber.datatype; #REQUIRED
    kernelMatrix CDATA #REQUIRED
    divisor %Number.datatype; #IMPLIED
    bias %Number.datatype; #IMPLIED
    targetX %Integer.datatype; #IMPLIED
    targetY %Integer.datatype; #IMPLIED
    edgeMode ( duplicate | wrap | none ) 'duplicate'
    kernelUnitLength %NumberOptionalNumber.datatype; #IMPLIED
    preserveAlpha %Boolean.datatype; #IMPLIED
>
```

Attribute definitions:

`order` = "<number-optional-number>"

Indicates the number of cells in each dimension for **kernelMatrix**. The values provided must be <integer>s greater than zero. The first number, <orderX>, indicates the number of columns in the matrix. The second number, <orderY>, indicates the number of rows in the matrix. If <orderY> is not provided, it defaults to <orderX>.
A typical value is order="3". It is recommended that only small values (e.g., 3) be used; higher values may result in very high CPU overhead and usually do not produce results that justify the impact on performance.
If the attribute is not specified, the effect is as if a value of "3" were specified.
Animatable: yes.

`kernelMatrix` = "<list of numbers>"

The list of <number>s that make up the kernel matrix for the convolution. Values are separated by space characters and/or a comma. The number of entries in the list must equal <orderX> times <orderY>.
Animatable: yes.

`divisor` = "<number>"

After applying the **kernelMatrix** to the input image to yield a number, that number is divided by **divisor** to yield the final destination color value. A divisor that is the sum of all the matrix values tends to have an evening effect on the overall color intensity of the result. It is an error to specify a divisor of zero. The default value is the sum of all values in kernelMatrix, with the exception that if the sum is zero, then the divisor is set to 1.
Animatable: yes.

`bias` = "<number>"

After applying the **kernelMatrix** to the input image to yield a number and applying the **divisor**, the **bias** attribute is added to each component. One application of **bias** is when it is desirable to have .5 gray value be the zero response of the filter. If **bias** is not specified, then the effect is as if a value of zero were specified.
Animatable: yes.

`targetX` = "<integer>"

Determines the positioning in X of the convolution matrix relative to a given target pixel in the input image. The leftmost column of the matrix is column number zero. The value must be such that: 0 <= targetX < orderX. By default, the convolution matrix is centered in X over each pixel of the input image (i.e., targetX = floor (orderX / 2)).
Animatable: yes.

`targetY` = "<integer>"

Determines the positioning in Y of the convolution matrix relative to a given target pixel in the input image. The topmost row of the matrix is row number zero. The value must be such that: 0 <= targetY < orderY. By default, the convolution matrix is centered in Y over each pixel of the input image (i.e., targetY = floor (orderY / 2)).
Animatable: yes.

edgeMode = "duplicate | wrap | none"

Determines how to extend the input image as necessary with color values so that the matrix operations can be applied when the kernel is positioned at or near the edge of the input image.

"duplicate" indicates that the input image is extended along each of its borders as necessary by duplicating the color values at the given edge of the input image.

```
Original N-by-M image, where m=M-1 and n=N-1:
          11 12 ... 1m 1M
          21 22 ... 2m 2M
          .. .. ... .. ..
          n1 n2 ... nm nM
          N1 N2 ... Nm NM
Extended by two pixels using "duplicate":
  11 11   11 12 ... 1m 1M   1M 1M
  11 11   11 12 ... 1m 1M   1M 1M
  11 11   11 12 ... 1m 1M   1M 1M
  21 21   21 22 ... 2m 2M   2M 2M
  .. ..   .. .. ... .. ..   .. ..
  n1 n1   n1 n2 ... nm nM   nM nM
  N1 N1   N1 N2 ... Nm NM   NM NM
  N1 N1   N1 N2 ... Nm NM   NM NM
  N1 N1   N1 N2 ... Nm NM   NM NM
```

"wrap" indicates that the input image is extended by taking the color values from the opposite edge of the image.

```
Extended by two pixels using "wrap":
  nm nM   n1 n2 ... nm nM   n1 n2
  Nm NM   N1 N2 ... Nm NM   N1 N2
  1m 1M   11 12 ... 1m 1M   11 12
  2m 2M   21 22 ... 2m 2M   21 22
  .. ..   .. .. ... .. ..   .. ..
  nm nM   n1 n2 ... nm nM   n1 n2
  Nm NM   N1 N2 ... Nm NM   N1 N2
  1m 1M   11 12 ... 1m 1M   11 12
  2m 2M   21 22 ... 2m 2M   21 22
```

"none" indicates that the input image is extended with pixel values of zero for R, G, B and A.

Animatable: yes.

kernelUnitLength = "<number-optional-number>"

The first number is the <dx> value. The second number is the <dy> value. If the <dy> value is not specified, it defaults to the same value as <dx>. Indicates the intended distance in current filter units (i.e., units as determined by the value of attribute **primitiveUnits**) between successive columns and rows, respectively, in the **kernelMatrix**. By specifying value(s) for **kernelUnitLength**, the kernel becomes defined in a scalable, abstract coordinate system. If **kernelUnitLength** is not specified, the default value is one pixel in the offscreen bitmap, which is a pixel-based coordinate system, and thus potentially not scalable. For some level of consistency across display media and user agents, it is necessary that a value be provided for at least one of **filterRes** and **kernelUnitLength**. In some implementations, the most consistent results and the fastest performance will be achieved if the pixel grid of the temporary offscreen images aligns with the pixel grid of the kernel.
A negative or zero value is an error (see Error processing).
Animatable: yes.

preserveAlpha = "false | true"

A value of **false** indicates that the convolution will apply to all channels, including the alpha channel.
A value of **true** indicates that the convolution will only apply to the color channels. In this case, the filter will temporarily unpremultiply the color component values, apply the kernel, and then re-premultiply at the end.

If **preserveAlpha** is not specified, then the effect is as if a value of **false** were specified.
Animatable: yes.

15.14 Filter primitive 'feDiffuseLighting'

This filter primitive lights an image using the alpha channel as a bump map. The resulting image is an RGBA opaque image based on the light color with alpha = 1.0 everywhere. The lighting calculation follows the standard diffuse component of the Phong lighting model. The resulting image depends on the light color, light position and surface geometry of the input bump map.

The light map produced by this filter primitive can be combined with a texture image using the multiply term of the *arithmetic* **'feComposite'** compositing method. Multiple light sources can be simulated by adding several of these light maps together before applying it to the texture image.

The formulas below make use of 3x3 filters. Because they operate on pixels, such filters are inherently resolution-dependent. To make **'feDiffuseLighting'** produce resolution-independent results, an explicit value should be provided for either the **filterRes** attribute on the **'filter'** element and/or attribute **kernelUnitLength**.

kernelUnitLength, in combination with the other attributes, defines an implicit pixel grid in the filter effects coordinate system (i.e., the coordinate system established by the **primitiveUnits** attribute). If the pixel grid established by **kernelUnitLength** is not scaled to match the pixel grid established by attribute **filterRes** (implicitly or explicitly), then the input image will be temporarily rescaled to match its pixels with **kernelUnitLength**. The 3x3 filters are applied to the resampled image. After applying the filter, the image is resampled back to its original resolution.

When the image must be resampled, it is recommended that high quality viewers make use of appropriate interpolation techniques, for example bilinear or bicubic. Depending on the speed of the available interpolents, this choice may be affected by the **'image-rendering'** property setting. Note that implementations might choose approaches that minimize or eliminate resampling when not necessary to produce proper results, such as when the document is zoomed out such that **kernelUnitLength** is considerably smaller than a device pixel.

For the formulas that follow, the $\mathrm{Norm}(A_x,A_y,A_z)$ function is defined as:

$$\mathrm{Norm}(A_x,A_y,A_z) = \mathrm{sqrt}(A_x\hat{}2+A_y\hat{}2+A_z\hat{}2)$$

The resulting RGBA image is computed as follows:

$$D_r = k_d * N.L * L_r$$
$$D_g = k_d * N.L * L_g$$
$$D_b = k_d * N.L * L_b$$
$$D_a = 1.0$$

where

k_d = diffuse lighting constant
N = surface normal unit vector, a function of x and y
L = unit vector pointing from surface to light, a function of x and y in the point and spot light cases
L_r,L_g,L_b = RGB components of light, a function of x and y in the spot light case

N is a function of x and y and depends on the surface gradient as follows:

The surface described by the input alpha image A_{in} (x,y) is:

$$Z\ (x,y) = \mathrm{surfaceScale} * A_{in}\ (x,y)$$

Surface normal is calculated using the Sobel gradient 3x3 filter. Different filter kernels are used depending on whether the given pixel is on the interior or an edge. For each case, the formula is:

```
Nx (x,y)= - surfaceScale * FACTORx *
           (K x(0,0)*I(x-dx,y-dy) + Kx(1,0)*I(x,y-dy) + Kx(2,0)*I(x+dx,y-dy) +
            K x(0,1)*I(x-dx,y)    + Kx(1,1)*I(x,y)    + Kx(2,1)*I(x+dx,y)    +
            K x(0,2)*I(x-dx,y+dy) + Kx(1,2)*I(x,y+dy) + Kx(2,2)*I(x+dx,y+dy))
Ny (x,y)= - surfaceScale * FACTORy *
           (K y(0,0)*I(x-dx,y-dy) + Ky(1,0)*I(x,y-dy) + Ky(2,0)*I(x+dx,y-dy) +
            K y(0,1)*I(x-dx,y)    + Ky(1,1)*I(x,y)    + Ky(2,1)*I(x+dx,y)    +
            K y(0,2)*I(x-dx,y+dy) + Ky(1,2)*I(x,y+dy) + Ky(2,2)*I(x+dx,y+dy))
Nz (x,y) = 1.0

N = (Nx, Ny, Nz) / Norm((Nx,Ny,Nz))
```

In these formulas, the dx and dy values (e.g., I(x-dx,y-dy)), represent deltas relative to a given (x,y) position for the purpose of estimating the slope of the surface at that point. These deltas are determined by the value (explicit or implicit) of attribute **kernelUnitLength**.

Top/left corner: FACTOR$_x$=2/(3*dx) K$_x$ = $\begin{vmatrix} 0 & 0 & 0 \\ 0 & -2 & 2 \\ 0 & -1 & 1 \end{vmatrix}$ FACTOR$_y$=2/(3*dy) K$_y$ = $\begin{vmatrix} 0 & 0 & 0 \\ 0 & -2 & -1 \\ 0 & 2 & 1 \end{vmatrix}$	Top row: FACTOR$_x$=1/(3*dx) K$_x$ = $\begin{vmatrix} 0 & 0 & 0 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{vmatrix}$ FACTOR$_y$=1/(2*dy) K$_y$ = $\begin{vmatrix} 0 & 0 & 0 \\ -1 & -2 & -1 \\ 1 & 2 & 1 \end{vmatrix}$	Top/right corner: FACTOR$_x$=2/(3*dx) K$_x$ = $\begin{vmatrix} 0 & 0 & 0 \\ -2 & 2 & 0 \\ -1 & 1 & 0 \end{vmatrix}$ FACTOR$_y$=2/(3*dy) K$_y$ = $\begin{vmatrix} 0 & 0 & 0 \\ -1 & -2 & 0 \\ 1 & 2 & 0 \end{vmatrix}$
Left column: FACTOR$_x$=1/(2*dx) K$_x$ = $\begin{vmatrix} 0 & -1 & 1 \\ 0 & -2 & 2 \\ 0 & -1 & 1 \end{vmatrix}$ FACTOR$_y$=1/(3*dy) K$_y$ = $\begin{vmatrix} 0 & -2 & -1 \\ 0 & 0 & 0 \\ 0 & 2 & 1 \end{vmatrix}$	Interior pixels: FACTOR$_x$=1/(4*dx) K$_x$ = $\begin{vmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{vmatrix}$ FACTOR$_y$=1/(4*dy) K$_y$ = $\begin{vmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{vmatrix}$	Right column: FACTOR$_x$=1/(2*dx) K$_x$ = $\begin{vmatrix} -1 & 1 & 0 \\ -2 & 2 & 0 \\ -1 & 1 & 0 \end{vmatrix}$ FACTOR$_y$=1/(3*dy) K$_y$ = $\begin{vmatrix} -1 & -2 & 0 \\ 0 & 0 & 0 \\ 1 & 2 & 0 \end{vmatrix}$

Bottom/left corner:	Bottom row:	Bottom/right corner:
$FACTOR_x$=2/(3*dx) K_x = | 0 -1 1 | | 0 -2 2 | | 0 0 0 | $FACTOR_y$=2/(3*dy) K_y = | 0 -2 -1 | | 0 2 1 | | 0 0 0 |	$FACTOR_x$=1/(3*dx) K_x = | -1 0 1 | | -2 0 2 | | 0 0 0 | $FACTOR_y$=1/(2*dy) K_y = | -1 -2 -1 | | 1 2 1 | | 0 0 0 |	$FACTOR_x$=2/(3*dx) K_x = | -1 1 0 | | -2 2 0 | | 0 0 0 | $FACTOR_y$=2/(3*dy) K_y = | -1 -2 0 | | 1 2 0 | | 0 0 0 |

L, the unit vector from the image sample to the light, is calculated as follows:

For Infinite light sources it is constant:

L_x = cos(azimuth)*cos(elevation)

L_y = sin(azimuth)*cos(elevation)

L_z = sin(elevation)

For Point and spot lights it is a function of position:

L_x = $Light_x$ - x

L_y = $Light_y$ - y

L_z = $Light_z$ - Z(x,y)

L = (L_x, L_y, L_z) / Norm(L_x, L_y, L_z)

where $Light_x$, $Light_y$, and $Light_z$ are the input light position.

L_r,L_g,L_b, the light color vector, is a function of position in the spot light case only:

L_r = $Light_r$*pow((-L.S),specularExponent)

L_g = $Light_g$*pow((-L.S),specularExponent)

L_b = $Light_b$*pow((-L.S),specularExponent)

where S is the unit vector pointing from the light to the point (pointsAtX, pointsAtY, pointsAtZ) in the x-y plane:

S_x = pointsAtX - $Light_x$

S_y = pointsAtY - $Light_y$

S_z = pointsAtZ - $Light_z$

S = (S_x, S_y, S_z) / Norm(S_x, S_y, S_z)

If L.S is positive, no light is present. ($L_r = L_g = L_b = 0$)

```
<!ENTITY % SVG.feDiffuseLighting.extra.content "" >
<!ENTITY % SVG.feDiffuseLighting.element "INCLUDE" >
<![%SVG.feDiffuseLighting.element;[
<!ENTITY % SVG.feDiffuseLighting.content
    "(( %SVG.feDistantLight.qname; | %SVG.fePointLight.qname;
      | %SVG.feSpotLight.qname; ), ( %SVG.animate.qname; | %SVG.set.qname;
      | %SVG.animateColor.qname; %SVG.feDiffuseLig\
hting.extra.content; )*)"
>
<!ELEMENT %SVG.feDiffuseLighting.qname; %SVG.fe\
DiffuseLighting.content; >
<!-- end of SVG.feDiffuseLighting.element -->]]>
<!ENTITY % SVG.feDiffuseLighting.attlist "INCLUDE" >
<![%SVG.feDiffuseLighting.attlist;[
<!ATTLIST %SVG.feDiffuseLighting.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Color.attrib;
    %SVG.FilterColor.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
    lighting-color %SVGColor.datatype; #IMPLIED
    surfaceScale %Number.datatype; #IMPLIED
    diffuseConstant %Number.datatype; #IMPLIED
    kernelUnitLength %NumberOptionalNumber.datatype; #IMPLIED
>
```

Attribute definitions:

surfaceScale = "*<number>*"

height of surface when A_{in} = 1.

If the attribute is not specified, then the effect is as if a value of **1** were specified.
Animatable: yes.

diffuseConstant = "*<number>*"

kd in Phong lighting model. In SVG, this can be any non-negative number.
If the attribute is not specified, then the effect is as if a value of **1** were specified.
Animatable: yes.

kernelUnitLength = "<number-optional-number>"

The first number is the <dx> value. The second number is the <dy> value. If the <dy> value is not specified, it defaults to the same value as <dx>. Indicates the intended distance in current filter units (i.e., units as determined by the value of attribute **primitiveUnits**) for dx and dy, respectively, in the surface normal calculation formulas. By specifying value(s) for **kernelUnitLength**, the kernel becomes defined in a scalable, abstract coordinate system. If **kernelUnitLength** is not specified, the dx and dy values should represent very small deltas relative to a given (x, y) position, which might be implemented in some cases as one pixel in the intermediate image offscreen bitmap, which is a pixel-based coordinate system, and thus potentially not scalable. For some level of consistency across display media and user agents, it is necessary that a value be provided for at least one of **filterRes** and **kernelUnitLength**. Discussion of intermediate images are in the Introduction and in the description of attribute **filterRes**.
A negative or zero value is an error (see Error processing).
Animatable: yes.

The light source is defined by one of the child elements **'feDistantLight'**, **'fePointLight'** or **'feSpotLight'**. The light color is specified by property **'lighting-color'**.

15.15 Filter primitive 'feDisplacementMap'

This filter primitive uses the pixels values from the image from **in2** to spatially displace the image from **in**. This is the transformation to be performed:

```
P'(x,y) <- P( x + scale * (XC(x,y) - .5), y + scale * (YC(x,y) - .5))
```

where P(x,y) is the input image, **in**, and P'(x,y) is the destination. XC(x,y) and YC(x,y) are the component values of the designated by the *xChannelSelector* and *yChannelSelector*. For example, to use the R component of **in2** to control displacement in x and the G component of Image2 to control displacement in y, set *xChannelSelector* to "R" and *yChannelSelector* to "G".

The displacement map defines the inverse of the mapping performed.

The calculations using the pixel values from **in2** are performed using non-premultiplied color values. If the image from **in2** consists of premultiplied color values, those values are automatically converted into non-premultiplied color values before performing this operation.

This filter can have arbitrary non-localized effect on the input which might require substantial buffering in the processing pipeline. However with this formulation, any intermediate buffering needs can be determined by *scale* which represents the maximum range of displacement in either x or y.

When applying this filter, the source pixel location will often lie between several source pixels. In this case it is recommended that high quality viewers apply an interpolent on the surrounding pixels, for example bilinear or bicubic, rather than simply selecting the nearest source pixel. Depending on the speed of the available interpolents, this choice may be affected by the **'image-rendering'** property setting.

The **'color-interpolation-filters'** property only applies to the **in2** source image and does not apply to the **in** source image.

```
<!ENTITY % SVG.feDisplacementMap.extra.content "" >
<!ENTITY % SVG.feDisplacementMap.element "INCLUDE" >
<![%SVG.feDisplacementMap.element;[
<!ENTITY % SVG.feDisplacementMap.content
    "( %SVG.animate.qname; | %SVG.set.qname;
       %SVG.feDisplacementMap.extra.content; )*"
>
<!ELEMENT %SVG.feDisplacementMap.qname; %SVG.fe\
DisplacementMap.content; >
<!-- end of SVG.feDisplacementMap.element -->]]>
<!ENTITY % SVG.feDisplacementMap.attlist "INCLUDE" >
<![%SVG.feDisplacementMap.attlist;[
<!ATTLIST %SVG.feDisplacementMap.qname;
    %SVG.Core.attrib;
    %SVG.FilterColor.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
    in2 CDATA #REQUIRED
    scale %Number.datatype; #IMPLIED
    xChannelSelector ( R | G | B | A ) 'A'
    yChannelSelector ( R | G | B | A ) 'A'
>
```

Attribute definitions:

`scale` = "*<number>*"

Displacement scale factor. The amount is expressed in the coordinate system established by attribute **primitiveUnits** on the **'filter'** element.
When the value of this attribute is **0**, this operation has no effect on the source image.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
Animatable: yes.

`xChannelSelector` = "*R* | *G* | *B* | *A*"

Indicates which channel from **in2** to use to displace the pixels in **in** along the x-axis.
Animatable: yes.

`yChannelSelector` = "*R* | *G* | *B* | *A*"

Indicates which channel from **in2** to use to displace the pixels in **in** along the y-axis.

Animatable: yes.

in2 = "*(see* in *attribute)*"

The second input image, which is used to displace the pixels in the image from attribute **in**. This attribute can take on the same values as the **in** attribute.

Animatable: yes.

15.16 Filter primitive 'feFlood'

This filter primitive creates a rectangle filled with the color and opacity values from properties **'flood-color'** and **'flood-opacity'**. The rectangle is as large as the filter primitive subregion established by the **x**, **y**, **width** and **height** attributes on the **'feFlood'** element.

```
<!ENTITY % SVG.feFlood.extra.content "" >
<!ENTITY % SVG.feFlood.element "INCLUDE" >
<![%SVG.feFlood.element;[
<!ENTITY % SVG.feFlood.content
    "( %SVG.animate.qname; | %SVG.set.qname; | %SVG.animateColor.qname;
       %SVG.feFlood.extra.content; )*"
>
<!ELEMENT %SVG.feFlood.qname; %SVG.feFlood.content; >
<!-- end of SVG.feFlood.element -->]]>
<!ENTITY % SVG.feFlood.attlist "INCLUDE" >
<![%SVG.feFlood.attlist;[
<!ATTLIST %SVG.feFlood.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Color.attrib;
    %SVG.FilterColor.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
    flood-color %SVGColor.datatype; #IMPLIED
    flood-opacity %OpacityValue.datatype; #IMPLIED
>
```

The **'flood-color'** property indicates what color to use to flood the current filter primitive subregion. The keyword **currentColor** and ICC colors can be specified in the same manner as within a <paint> specification for the **'fill'** and **'stroke'** properties.

'flood-color'

Value:	currentColor | <color> [icc-color(<name>[,<icccolorvalue>]*)] | inherit
Initial:	black
Applies to:	**'feFlood'** elements
Inherited:	no
Percentages:	N/A
Media:	visual
Animatable:	yes

The **'flood-opacity'** property defines the opacity value to use across the entire filter primitive subregion.

'flood-opacity'

Value:	<opacity-value> | inherit
Initial:	1
Applies to:	**'feFlood'** elements
Inherited:	no
Percentages:	N/A

Media: visual
Animatable: yes

15.17 Filter primitive 'feGaussianBlur'

This filter primitive performs a Gaussian blur on the input image.

The Gaussian blur kernel is an approximation of the normalized convolution:

$$H(x) = exp(-x^2/ (2s^2)) / sqrt(2* pi*s^2)$$

where 's' is the standard deviation specified by **stdDeviation**.

The value of **stdDeviation** can be either one or two numbers. If two numbers are provided, the first number represents a standard deviation value along the x-axis of the current coordinate system and the second value represents a standard deviation in Y. If one number is provided, then that value is used for both X and Y.

Even if only one value is provided for **stdDeviation**, this can be implemented as a separable convolution.

For larger values of 's' (s >= 2.0), an approximation can be used: Three successive box-blurs build a piece-wise quadratic convolution kernel, which approximates the Gaussian kernel to within roughly 3%.

```
let d = floor(s * 3*sqrt(2*pi)/4 + 0.5)
```

... if d is odd, use three box-blurs of size 'd', centered on the output pixel.

... if d is even, two box-blurs of size 'd' (the first one centered on the pixel boundary between the output pixel and the one to the left, the second one centered on the pixel boundary between the output pixel and the one to the right) and one box blur of size 'd+1' centered on the output pixel.

Frequently this operation will take place on alpha-only images, such as that produced by the built-in input, **SourceAlpha**. The implementation may notice this and optimize the single channel case. If the input has infinite extent and is constant, this operation has no effect. If the input has infinite extent and is a tile, the filter is evaluated with periodic boundary conditions.

```
<!ENTITY % SVG.feGaussianBlur.extra.content "" >
<!ENTITY % SVG.feGaussianBlur.element "INCLUDE" >
<![%SVG.feGaussianBlur.element;[
<!ENTITY % SVG.feGaussianBlur.content
    "( %SVG.animate.qname; | %SVG.set.qname;
       %SVG.feGaussianBlur.extra.content; )*"
>
<!ELEMENT %SVG.feGaussianBlur.qname; %SVG.feGaussi\
anBlur.content; >
<!-- end of SVG.feGaussianBlur.element -->]]>
<!ENTITY % SVG.feGaussianBlur.attlist "INCLUDE" >
<![%SVG.feGaussianBlur.attlist;[
<!ATTLIST %SVG.feGaussianBlur.qname;
    %SVG.Core.attrib;
    %SVG.FilterColor.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
    stdDeviation %NumberOptionalNumber.datatype; #IMPLIED
>
```

Attribute definitions:

stdDeviation = "*<number-optional-number>*"
The standard deviation for the blur operation. If two <number>s are provided, the first number represents a

standard deviation value along the x-axis of the coordinate system established by attribute **primitiveUnits** on the **'filter'** element. The second value represents a standard deviation in Y. If one number is provided, then that value is used for both X and Y.
A negative value is an error (see Error processing). A value of zero disables the effect of the given filter primitive (i.e., the result is a transparent black image).
If the attribute is not specified, then the effect is as if a value of **0** were specified.
Animatable: yes.

The example at the start of this chapter makes use of the **feGaussianBlur** filter primitive to create a drop shadow effect.

15.18 Filter primitive 'feImage'

This filter primitive refers to a graphic external to this filter element, which is loaded or rendered into an RGBA raster and becomes the result of the filter primitive.

This filter primitive can refer to an external image or can be a reference to another piece of SVG. It produces an image similar to the built-in image source **SourceGraphic** except that the graphic comes from an external source.

If the **xlink:href** references a stand-alone image resource such as a JPEG, PNG or SVG file, then the image resource is rendered according to the behavior of the **'image'** element; otherwise, the referenced resource is rendered according to the behavior of the **'use'** element. In either case, the current user coordinate system depends on the value of attribute **primitiveUnits** on the **'filter'** element. The processing of the **preserveAspectRatio** attribute on the **'feImage'** element is identical to that of the **'image'** element.

When the referenced image must be resampled to match the device coordinate system, it is recommended that high quality viewers make use of appropriate interpolation techniques, for example bilinear or bicubic. Depending on the speed of the available interpolents, this choice may be affected by the **'image-rendering'** property setting.

```
<!ENTITY % SVG.feImage.extra.content "" >
<!ENTITY % SVG.feImage.element "INCLUDE" >
<![%SVG.feImage.element;[
<!ENTITY % SVG.feImage.content
    "( %SVG.animate.qname; | %SVG.set.qname; | %SVG.animateTransform.qname;
       %SVG.feImage.extra.content; )*"
>
<!ELEMENT %SVG.feImage.qname; %SVG.feImage.content; >
<!-- end of SVG.feImage.element -->]]>
<!ENTITY % SVG.feImage.attlist "INCLUDE" >
<![%SVG.feImage.attlist;[
<!ATTLIST %SVG.feImage.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Presentation.attrib;
    %SVG.FilterPrimitive.attrib;
    %SVG.XLinkEmbed.attrib;
    %SVG.External.attrib;
    preserveAspectRatio %PreserveAspectRatioSpec.datatype; 'xMidYMid meet'
>
```

15.19 Filter primitive 'feMerge'

This filter primitive composites input image layers on top of each other using the *over* operator with *Input1* (corresponding to the first **'feMergeNode'** child element) on the bottom and the last specified input, *InputN* (corresponding to the last **'feMergeNode'** child element), on top.

Many effects produce a number of intermediate layers in order to create the final output image. This filter allows us to collapse those into a single image. Although this could be done by using n-1 Composite-filters, it is more convenient to have this common operation available in this form, and offers the implementation some additional flexibility.

Each 'feMerge' element can have any number of 'feMergeNode' subelements, each of which has an **in** attribute.

The canonical implementation of feMerge is to render the entire effect into one RGBA layer, and then render the resulting layer on the output device. In certain cases (in particular if the output device itself is a continuous tone device), and since merging is associative, it might be a sufficient approximation to evaluate the effect one layer at a time and render each layer individually onto the output device bottom to top.

If the topmost image input is **SourceGraphic** and this **'feMerge'** is the last filter primitive in the filter, the implementation is encouraged to render the layers up to that point, and then render the **SourceGraphic** directly from its vector description on top.

```
<!ENTITY % SVG.feMerge.extra.content "" >
<!ENTITY % SVG.feMerge.element "INCLUDE" >
<![%SVG.feMerge.element;[
<!ENTITY % SVG.feMerge.content
    "( %SVG.feMergeNode.qname; %SVG.feMerge.extra.\
content; )*"
>
<!ELEMENT %SVG.feMerge.qname; %SVG.feMerge.content; >
<!-- end of SVG.feMerge.element -->]]>
<!ENTITY % SVG.feMerge.attlist "INCLUDE" >
<![%SVG.feMerge.attlist;[
<!ATTLIST %SVG.feMerge.qname;
    %SVG.Core.attrib;
    %SVG.FilterColor.attrib;
    %SVG.FilterPrimitive.attrib;
>
<!-- end of SVG.feMerge.attlist -->]]>
<!-- feMergeNode: Filter Effect Merge Node Element ..... -->
<!ENTITY % SVG.feMergeNode.extra.content "" >
<!ENTITY % SVG.feMergeNode.element "INCLUDE" >
<![%SVG.feMergeNode.element;[
<!ENTITY % SVG.feMergeNode.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feMergeNode.extra.content; )*"
>
<!ELEMENT %SVG.feMergeNode.qname; %SVG.feMergeNode.co\
ntent; >
<!-- end of SVG.feMergeNode.element -->]]>
<!ENTITY % SVG.feMergeNode.attlist "INCLUDE" >
<![%SVG.feMergeNode.attlist;[
<!ATTLIST %SVG.feMergeNode.qname;
    %SVG.Core.attrib;
    in CDATA #IMPLIED
>
```

The example at the start of this chapter makes use of the **feMerge** filter primitive to composite two intermediate filter results together.

15.20 Filter primitive 'feMorphology'

This filter primitive performs "fattening" or "thinning" of artwork. It is particularly useful for fattening or thinning an alpha channel.

The dilation (or erosion) kernel is a rectangle with a width of 2**x-radius* and a height of 2**y-radius*. In dilation, the output pixel is the individual component-wise maximum of the corresponding R,G,B,A values in the input image's kernel rectangle. In erosion, the output pixel is the individual component-wise minimum of the corresponding R,G,B,A values in the input image's kernel rectangle.

Frequently this operation will take place on alpha-only images, such as that produced by the built-in input, **SourceAlpha**. In that case, the implementation might want to optimize the single channel case.

If the input has infinite extent and is constant, this operation has no effect. If the input has infinite extent and is a tile, the filter is evaluated with periodic boundary conditions.

Because **'feMorphology'** operates on premultipied color values, it will always result in color values less than or equal to the alpha channel.

```
<!ENTITY % SVG.feMorphology.extra.content "" >
<!ENTITY % SVG.feMorphology.element "INCLUDE" >
<![%SVG.feMorphology.element;[
<!ENTITY % SVG.feMorphology.content
    "( %SVG.animate.qname; | %SVG.set.qname;
       %SVG.feMorphology.extra.content; )*"
>
<!ELEMENT %SVG.feMorphology.qname; %SVG.feMorphology\
.content; >
<!-- end of SVG.feMorphology.element -->]]>
<!ENTITY % SVG.feMorphology.attlist "INCLUDE" >
<![%SVG.feMorphology.attlist;[
<!ATTLIST %SVG.feMorphology.qname;
    %SVG.Core.attrib;
    %SVG.FilterColor.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
    operator ( erode | dilate ) 'erode'
    radius %NumberOptionalNumber.datatype; #IMPLIED
>
```

Attribute definitions:

`operator` = "*erode | dilate*"

A keyword indicating whether to erode (i.e., thin) or dilate (fatten) the source graphic.
Animatable: yes.

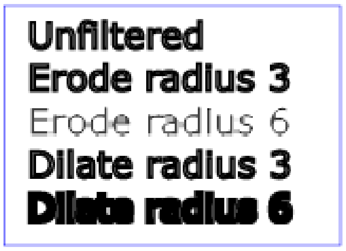
`radius` = "*<number-optional-number>*"

The radius (or radii) for the operation. If two <number>s are provided, the first number represents a x-radius and the second value represents a y-radius. If one number is provided, then that value is used for both X and Y. The values are in the coordinate system established by attribute **primitiveUnits** on the **'filter'** element.
A negative value is an error (see Error processing). A value of zero disables the effect of the given filter primitive (i.e., the result is a transparent black image).
If the attribute is not specified, then the effect is as if a value of `0` were specified.
Animatable: yes.

Example feMorphology shows examples of the four types of feMorphology operations.

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
          "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="5cm" height="7cm" viewBox="0 0 700 500" version="1.1"
     xmlns="http://www.w3.org/2000/svg">
  <title>Example feMorphology - Examples of erode and dilate</title>
  <desc>Five text strings drawn as outlines.
        The first is unfiltered. The second and third use 'erode'.
        The fourth and fifth use 'dilate'.</desc>
  <defs>
    <filter id="Erode3">
      <feMorphology operator="erode" in="SourceGraphic" radius="3" />
    </filter>
    <filter id="Erode6">
      <feMorphology operator="erode" in="SourceGraphic" radius="6" />
    </filter>
    <filter id="Dilate3">
      <feMorphology operator="dilate" in="SourceGraphic" radius="3" />
    </filter>
```

```
        <filter id="Dilate6">
          <feMorphology operator="dilate" in="SourceGraphic" radius="6" />
        </filter>
      </defs>
      <rect fill="none" stroke="blue" stroke-width="2"
            x="1" y="1" width="698" height="498"/>
      <g enable-background="new" >
        <g font-family="Verdana" font-size="75"
                  fill="none" stroke="black" stroke-width="6" >
          <text x="50" y="90">Unfiltered</text>
          <text x="50" y="180" filter="url(#Erode3)" >Erode radius 3</text>
          <text x="50" y="270" filter="url(#Erode6)" >Erode radius 6</text>
          <text x="50" y="360" filter="url(#Dilate3)" >Dilate radius 3</text>
          <text x="50" y="450" filter="url(#Dilate6)" >Dilate radius 6</text>
        </g>
      </g>
    </svg>
```



Unfiltered
Erode radius 3
Erode radius 6
Dilate radius 3
Dilate radius 6

Example feMorphology

[View this example as SVG (SVG-enabled browsers only)]

15.21 Filter primitive 'feOffset'

This filter primitive offsets the input image relative to its current position in the image space by the specified vector.

This is important for effects like drop shadows.

When applying this filter, the destination location may be offset by a fraction of a pixel in device space. In this case a high quality viewer should make use of appropriate interpolation techniques, for example bilinear or bicubic. This is especially recommended for dynamic viewers where this interpolation provides visually smoother movement of images. For static viewers this is less of a concern. Close attention should be made to the **'image-rendering'** property setting to determine the authors intent.

```
<!ENTITY % SVG.feOffset.extra.content "" >
<!ENTITY % SVG.feOffset.element "INCLUDE" >
<![%SVG.feOffset.element;[
<!ENTITY % SVG.feOffset.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feOffset.extra.content; )*"
>
<!ELEMENT %SVG.feOffset.qname; %SVG.feOffset.content; >
<!-- end of SVG.feOffset.element -->]]>
<!ENTITY % SVG.feOffset.attlist "INCLUDE" >
<![%SVG.feOffset.attlist;[
<!ATTLIST %SVG.feOffset.qname;
    %SVG.Core.attrib;
    %SVG.FilterColor.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
    dx %Number.datatype; #IMPLIED
    dy %Number.datatype; #IMPLIED
>
```

Attribute definitions:

dx = "*<number>*"

The amount to offset the input graphic along the x-axis. The offset amount is expressed in the coordinate system established by attribute **primitiveUnits** on the **'filter'** element.

If the attribute is not specified, then the effect is as if a value of **0** were specified.

Animatable: yes.

dy = "*<number>*"

The amount to offset the input graphic along the y-axis. The offset amount is expressed in the coordinate system established by attribute **primitiveUnits** on the **'filter'** element.

If the attribute is not specified, then the effect is as if a value of **0** were specified.

Animatable: yes.

The example at the start of this chapter makes use of the **feOffset** filter primitive to offset the drop shadow from the original source graphic.

15.22 Filter primitive 'feSpecularLighting'

This filter primitive lights a source graphic using the alpha channel as a bump map. The resulting image is an RGBA image based on the light color. The lighting calculation follows the standard specular component of the Phong lighting model. The resulting image depends on the light color, light position and surface geometry of the input bump map. The result of the lighting calculation is added. The filter primitive assumes that the viewer is at infinity in the z direction (i.e., the unit vector in the eye direction is (0,0,1) everywhere).

This filter primitive produces an image which contains the specular reflection part of the lighting calculation. Such a map is intended to be combined with a texture using the *add* term of the *arithmetic* **'feComposite'** method. Multiple light sources can be simulated by adding several of these light maps before applying it to the texture image.

The resulting RGBA image is computed as follows:

$S_r = k_s * \mathrm{pow}(N.H, \mathrm{specularExponent}) * L_r$
$S_g = k_s * \mathrm{pow}(N.H, \mathrm{specularExponent}) * L_g$
$S_b = k_s * \mathrm{pow}(N.H, \mathrm{specularExponent}) * L_b$
$S_a = \max(S_r, S_g, S_b)$

where

k_s = specular lighting constant

N = surface normal unit vector, a function of x and y
H = "halfway" unit vector between eye unit vector and light unit vector

L_r, L_g, L_b = RGB components of light

See **'feDiffuseLighting'** for definition of N and (L_r, L_g, L_b).

The definition of H reflects our assumption of the constant eye vector E = (0,0,1):

```
H = (L + E) / Norm(L+E)
```

where L is the light unit vector.

Unlike the **'feDiffuseLighting'**, the **'feSpecularLighting'** filter produces a non-opaque image. This is due to the fact that the specular result (S_r,S_g,S_b,S_a) is meant to be added to the textured image. The alpha channel of the result is the max of the color components, so that where the specular light is zero, no additional coverage is added to the image and a fully white highlight will add opacity.

The **'feDiffuseLighting'** and **'feSpecularLighting'** filters will often be applied together. An implementation may detect this and calculate both maps in one pass, instead of two.

```
<!ENTITY % SVG.feSpecularLighting.extra.content "" >
<!ENTITY % SVG.feSpecularLighting.element "INCLUDE" >
<![%SVG.feSpecularLighting.element;[
<!ENTITY % SVG.feSpecularLighting.content
    "(( %SVG.feDistantLight.qname; | %SVG.fePointLight.qname;
      | %SVG.feSpotLight.qname; ), ( %SVG.animate.qname; | %SVG.set.qname;
      | %SVG.animateColor.qname; %SVG.feSpecularLi\
ghting.extra.content; )*)"
>
<!ELEMENT %SVG.feSpecularLighting.qname; %SVG.\
feSpecularLighting.content; >
<!-- end of SVG.feSpecularLighting.element -->]]>
<!ENTITY % SVG.feSpecularLighting.attlist "INCLUDE" >
<![%SVG.feSpecularLighting.attlist;[
<!ATTLIST %SVG.feSpecularLighting.qname;
    %SVG.Core.attrib;
    %SVG.Style.attrib;
    %SVG.Color.attrib;
    %SVG.FilterColor.attrib;
    %SVG.Color.attrib;
    %SVG.FilterColor.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
    lighting-color %SVGColor.datatype; #IMPLIED
    surfaceScale %Number.datatype; #IMPLIED
    specularConstant %Number.datatype; #IMPLIED
    specularExponent %Number.datatype; #IMPLIED
    kernelUnitLength %NumberOptionalNumber.datatype; #IMPLIED
>
```

Attribute definitions:

`surfaceScale` = "*<number>*"

height of surface when A_{in} = 1.

If the attribute is not specified, then the effect is as if a value of **1** were specified.
Animatable: yes.

`specularConstant` = "*<number>*"

ks in Phong lighting model. In SVG, this can be any non-negative number.
If the attribute is not specified, then the effect is as if a value of **1** were specified.
Animatable: yes.

`specularExponent` = "*<number>*"

Exponent for specular term, larger is more "shiny". Range 1.0 to 128.0.
If the attribute is not specified, then the effect is as if a value of **1** were specified.
Animatable: yes.

The light source is defined by one of the child elements **'feDistantLight'**, **'fePointLight'** or **'feDistantLight'**. The light color is specified by property **'lighting-color'**.

The example at the start of this chapter makes use of the **feSpecularLighting** filter primitive to achieve a highly reflective, 3D glowing effect.

15.23 Filter primitive 'feTile'

This filter primitive fills a target rectangle with a repeated, tiled pattern of an input image. The target rectangle is as large as the filter primitive subregion established by the **x**, **y**, **width** and **height** attributes on the **'feTile'** element.

Typically, the input image has been defined with its own filter primitive subregion in order to define a reference tile. **'feTile'** replicates the reference tile in both X and Y to completely fill the target rectangle. The top/left corner of each given tile is at location `(x+i*width,y+j*height)`, where `(x,y)` represents the top/left of the input image's filter primitive subregion, `width` and `height` represent the width and height of the input image's filter primitive subregion, and `i` and `j` can be any integer value. In most cases, the input image will have a smaller filter primitive subregion than the **'feTile'** in order to achieve a repeated pattern effect.

Implementers must take appropriate measures in constructing the tiled image to avoid artifacts between tiles, particularly in situations where the user to device transform includes shear and/or rotation. Unless care is taken, interpolation can lead to edge pixels in the tile having opacity values lower or higher than expected due to the interaction of painting adjacent tiles which each have partial overlap with particular pixels.

```
<!ENTITY % SVG.feTile.extra.content "" >
<!ENTITY % SVG.feTile.element "INCLUDE" >
<![%SVG.feTile.element;[
<!ENTITY % SVG.feTile.content
    "( %SVG.animate.qname; | %SVG.set.qname; %SVG.feTile.extra.content; )*"
>
<!ELEMENT %SVG.feTile.qname; %SVG.feTile.content; >
<!-- end of SVG.feTile.element -->]]>
<!ENTITY % SVG.feTile.attlist "INCLUDE" >
<![%SVG.feTile.attlist;[
<!ATTLIST %SVG.feTile.qname;
    %SVG.Core.attrib;
    %SVG.FilterColor.attrib;
    %SVG.FilterPrimitiveWithIn.attrib;
>
```

15.24 Filter primitive 'feTurbulence'

This filter primitive creates an image using the Perlin turbulence function. It allows the synthesis of artificial textures like clouds or marble. For a detailed description the of the Perlin turbulence function, see "Texturing and Modeling", Ebert et al, AP Professional, 1994. The resulting image will fill the entire filter primitive subregion for this filter primitive.

It is possible to create bandwidth-limited noise by synthesizing only one octave.

The C code below shows the exact algorithm used for this filter effect.

For fractalSum, you get a turbFunctionResult that is aimed at a range of -1 to 1 (the actual result might exceed this range in some cases). To convert to a color value, use the formula `colorValue = ((turbFunctionResult * 255) + 255) / 2`, then clamp to the range 0 to 255.

For turbulence, you get a turbFunctionResult that is aimed at a range of 0 to 1 (the actual result might exceed this range in some cases). To convert to a color value, use the formula `colorValue = (turbFunctionResult * 255)`, then clamp to the range 0 to 255.

The following order is used for applying the pseudo random numbers. An initial seed value is computed based on attribute **seed**. Then the implementation computes the lattice points for R, then continues getting additional pseudo random numbers relative to the last generated pseudo random number and computes the lattice points for G, and so on for B and A.

The generated color and alpha values are in the color space determined by the value of property **'color-interpolation-**

filters':

```
/* Produces results in the range [1, 2**31 - 2].
Algorithm is: r = (a * r) mod m
where a = 16807 and m = 2**31 - 1 = 2147483647
See [Park & Miller], CACM vol. 31 no. 10 p. 1195, Oct. 1988
To test: the algorithm should produce the result 1043618065
as the 10,000th generated number if the original seed is 1.
*/
#define RAND_m 2147483647 /* 2**31 - 1 */
#define RAND_a 16807 /* 7**5; primitive root of m */
#define RAND_q 127773 /* m / a */
#define RAND_r 2836 /* m % a */
long setup_seed(long lSeed)
{
  if (lSeed <= 0) lSeed = -(lSeed % (RAND_m - 1)) + 1;
  if (lSeed > RAND_m - 1) lSeed = RAND_m - 1;
  return lSeed;
}
long random(long lSeed)
{
  long result;
  result = RAND_a * (lSeed % RAND_q) - RAND_r * (lSeed / RAND_q);
  if (result <= 0) result += RAND_m;
  return result;
}
#define BSize 0x100
#define BM 0xff
#define PerlinN 0x1000
#define NP 12 /* 2^PerlinN */
#define NM 0xfff
static uLatticeSelector[BSize + BSize + 2];
static double fGradient[4][BSize + BSize + 2][2];
struct StitchInfo
{
  int nWidth; // How much to subtract to wrap for stitching.
  int nHeight;
  int nWrapX; // Minimum value to wrap.
  int nWrapY;
};
static void init(long lSeed)
{
  double s;
  int i, j, k;
  lSeed = setup_seed(lSeed);
  for(k = 0; k < 4; k++)
  {
    for(i = 0; i < BSize; i++)
    {
      uLatticeSelector[i] = i;
      for (j = 0; j < 2; j++)
        fGradient[k][i][j] = (double)(((lSeed = random(lSeed)) % (BSize + BSize)) - BSize) / BSize;
      s = double(sqrt(fGradient[k][i][0] * fGradient[k][i][0] + fGradient[k][i][1] * fGradient[k][i][1]));
      fGradient[k][i][0] /= s;
      fGradient[k][i][1] /= s;
    }
  }
  while(--i)
  {
    k = uLatticeSelector[i];
    uLatticeSelector[i] = uLatticeSelector[j = (lSeed = random(lSeed)) % BSize];
    uLatticeSelector[j] = k;
  }
  for(i = 0; i < BSize + 2; i++)
  {
    uLatticeSelector[BSize + i] = uLatticeSelector[i];
    for(k = 0; k < 4; k++)
      for(j = 0; j < 2; j++)
        fGradient[k][BSize + i][j] = fGradient[k][i][j];
  }
}
#define s_curve(t) ( t * t * (3. - 2. * t) )
#define lerp(t, a, b) ( a + t * (b - a) )
double noise2(int nColorChannel, double vec[2], StitchInfo *pStitchInfo)
{
  int bx0, bx1, by0, by1, b00, b10, b01, b11;
  double rx0, rx1, ry0, ry1, *q, sx, sy, a, b, t, u, v;
  register i, j;
  t = vec[0] + PerlinN;
  bx0 = (int)t;
  bx1 = bx0+1;
  rx0 = t - (int)t;
  rx1 = rx0 - 1.0f;
  t = vec[1] + PerlinN;
  by0 = (int)t;
  by1 = by0+1;
  ry0 = t - (int)t;
  ry1 = ry0 - 1.0f;
  // If stitching, adjust lattice points accordingly.
  if(pStitchInfo != NULL)
  {
    if(bx0 >= pStitchInfo->nWrapX)
      bx0 -= pStitchInfo->nWidth;
    if(bx1 >= pStitchInfo->nWrapX)
      bx1 -= pStitchInfo->nWidth;
    if(by0 >= pStitchInfo->nWrapY)
      by0 -= pStitchInfo->nHeight;
    if(by1 >= pStitchInfo->nWrapY)
      by1 -= pStitchInfo->nHeight;
  }
  bx0 &= BM;
  bx1 &= BM;
  by0 &= BM;
  by1 &= BM;
  i = uLatticeSelector[bx0];
  j = uLatticeSelector[bx1];
  b00 = uLatticeSelector[i + by0];
  b10 = uLatticeSelector[j + by0];
  b01 = uLatticeSelector[i + by1];
  b11 = uLatticeSelector[j + by1];
  sx = double(s_curve(rx0));
  sy = double(s_curve(ry0));
  q = fGradient[nColorChannel][b00]; u = rx0 * q[0] + ry0 * q[1];
  q = fGradient[nColorChannel][b10]; v = rx1 * q[0] + ry0 * q[1];
  a = lerp(sx, u, v);
  q = fGradient[nColorChannel][b01]; u = rx0 * q[0] + ry1 * q[1];
  q = fGradient[nColorChannel][b11]; v = rx1 * q[0] + ry1 * q[1];
  b = lerp(sx, u, v);
  return lerp(sy, a, b);
}
double turbulence(int nColorChannel, double *point, double fBaseFreqX, double fBaseFreqY,
          int nNumOctaves, bool bFractalSum, bool bDoStitching,
          double fTileX, double fTileY, double fTileWidth, double fTileHeight)
{
  StitchInfo stitch;
  StitchInfo *pStitchInfo = NULL; // Not stitching when NULL.
  // Adjust the base frequencies if necessary for stitching.
  if(bDoStitching)
  {
    // When stitching tiled turbulence, the frequencies must be adjusted
    // so that the tile borders will be continuous.
    if(fBaseFreqX != 0.0)
    {
      double fLoFreq = double(floor(fTileWidth * fBaseFreqX)) / fTileWidth;
      double fHiFreq = double(ceil(fTileWidth * fBaseFreqX)) / fTileWidth;
      if(fBaseFreqX / fLoFreq < fHiFreq / fBaseFreqX)
        fBaseFreqX = fLoFreq;
      else
        fBaseFreqX = fHiFreq;
    }
    if(fBaseFreqY != 0.0)
    {
      double fLoFreq = double(floor(fTileHeight * fBaseFreqY)) / fTileHeight;
      double fHiFreq = double(ceil(fTileHeight * fBaseFreqY)) / fTileHeight;
      if(fBaseFreqY / fLoFreq < fHiFreq / fBaseFreqY)
        fBaseFreqY = fLoFreq;
      else
        fBaseFreqY = fHiFreq;
    }
    // Set up initial stitch values.
    pStitchInfo = &stitch;
    stitch.nWidth = int(fTileWidth * fBaseFreqX + 0.5f);
    stitch.nWrapX = fTileX * fBaseFreqX + PerlinN + stitch.nWidth;
    stitch.nHeight = int(fTileHeight * fBaseFreqY + 0.5f);
    stitch.nWrapY = fTileY * fBaseFreqY + PerlinN + stitch.nHeight;
  }
  double fSum = 0.0f;
  double vec[2];
  vec[0] = point[0] * fBaseFreqX;
  vec[1] = point[1] * fBaseFreqY;
  double ratio = 1;
  for(int nOctave = 0; nOctave < nNumOctaves; nOctave++)
  {
    if(bFractalSum)
      fSum += double(noise2(nColorChannel, vec, pStitchInfo) / ratio);
    else
      fSum += double(fabs(noise2(nColorChannel, vec, pStitchInfo)) / ratio);
    vec[0] *= 2;
    vec[1] *= 2;
    ratio *= 2;
    if(pStitchInfo != NULL)
    {
      // Update stitch values. Subtracting PerlinN before the multiplication and
      // adding it afterward simplifies to subtracting it once.
      stitch.nWidth *= 2;
      stitch.nWrapX = 2 * stitch.nWrapX - PerlinN;
      stitch.nHeight *= 2;
      stitch.nWrapY = 2 * stitch.nWrapY - PerlinN;
    }
  }
  return fSum;
}
```

```
<!ENTITY % SVG.feTurbulence.extra.content "" >
<!ENTITY % SVG.feTurbulence.element "INCLUDE" >
<![%SVG.feTurbulence.element;[
<!ENTITY % SVG.feTurbulence.content
    "( %SVG.animate.qname; | %SVG.set.qname;
       %SVG.feTurbulence.extra.content; )*"
>
<!ELEMENT %SVG.feTurbulence.qname; %SVG.feTurbulence\
.content; >
<!-- end of SVG.feTurbulence.element -->]]>
<!ENTITY % SVG.feTurbulence.attlist "INCLUDE" >
<![%SVG.feTurbulence.attlist;[
<!ATTLIST %SVG.feTurbulence.qname;
    %SVG.Core.attrib;
    %SVG.FilterColor.attrib;
    %SVG.FilterPrimitive.attrib;
    baseFrequency %NumberOptionalNumber.datatype; #IMPLIED
    numOctaves %Integer.datatype; #IMPLIED
    seed %Number.datatype; #IMPLIED
    stitchTiles ( stitch | noStitch ) 'noStitch'
    type ( fractalNoise | turbulence ) 'turbulence'
>
```

Attribute definitions:

baseFrequency = "*<number-optional-number>*"

The base frequency (frequencies) parameter(s) for the noise function. If two <number>s are provided, the first number represents a base frequency in the X direction and the second value represents a base frequency in the Y direction. If one number is provided, then that value is used for both X and Y.
A negative value for base frequency is an error (see Error processing).
If the attribute is not specified, then the effect is as if a value of **0** were specified.
Animatable: yes.

numOctaves = "*<integer>*"

The numOctaves parameter for the noise function.
If the attribute is not specified, then the effect is as if a value of **1** were specified.
Animatable: yes.

seed = "*<number>*"

The starting number for the pseudo random number generator.
If the attribute is not specified, then the effect is as if a value of **0** were specified.
Animatable: yes.

stitchTiles = "*stitch* | *noStitch*"

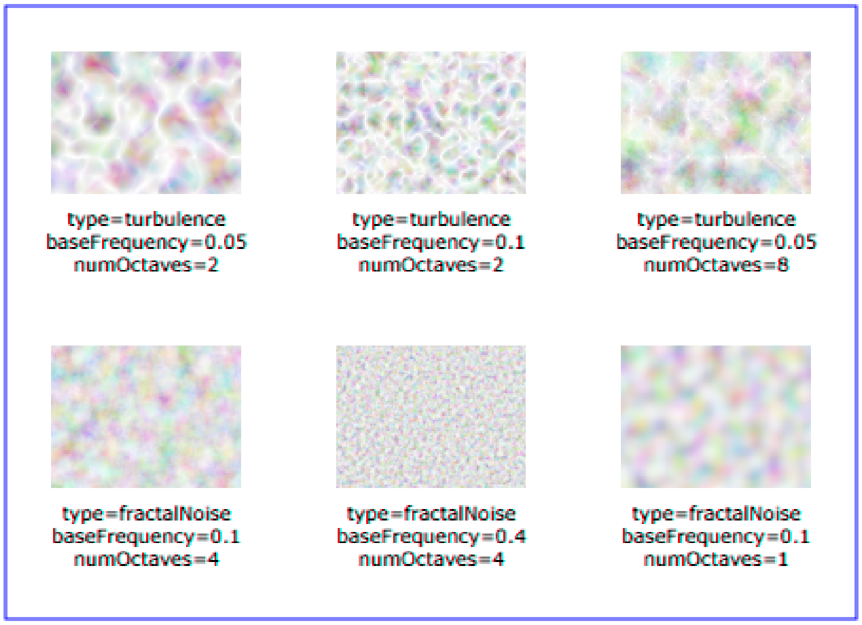
If **stitchTiles="noStitch"**, no attempt it made to achieve smooth transitions at the border of tiles which contain a turbulence function. Sometimes the result will show clear discontinuities at the tile borders.
If **stitchTiles="stitch"**, then the user agent will automatically adjust baseFrequency-x and baseFrequency-y values such that the feTurbulence node's width and height (i.e., the width and height of the current subregion) contains an integral number of the Perlin tile width and height for the first octave. The baseFrequency will be adjusted up or down depending on which way has the smallest relative (not absolute) change as follows: Given the frequency, calculate `lowFreq=floor(width*frequency)/width` and `hiFreq=ceil(width*frequency)/width`. If frequency/lowFreq < hiFreq/frequency then use lowFreq, else use hiFreq. While generating turbulence values, generate lattice vectors as normal for Perlin Noise, except for those lattice points that lie on the right or bottom edges of the active area (the size of the resulting tile). In those cases, copy the lattice vector from the opposite edge of the active area.
Animatable: yes.

type = "*fractalNoise* | *turbulence*"

Indicates whether the filter primitive should perform a noise or turbulence function.
Animatable: yes.

Example feTurbulence shows the effects of various parameter settings for feTurbulence.

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
          "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="450px" height="325px" viewBox="0 0 450 325" version="1.1"
     xmlns="http://www.w3.org/2000/svg">
  <title>Example feTurbulence - Examples of feTurbulence operations</title>
  <desc>Six rectangular areas showing the effects of
        various parameter settings for feTurbulence.</desc>
  <g  font-family="Verdana" text-anchor="middle" font-size="10" >
    <defs>
      <filter id="Turb1" filterUnits="objectBoundingBox"
              x="0%" y="0%" width="100%" height="100%">
        <feTurbulence type="turbulence" baseFrequency="0.05" numOctaves="2"/>
      </filter>
      <filter id="Turb2" filterUnits="objectBoundingBox"
              x="0%" y="0%" width="100%" height="100%">
        <feTurbulence type="turbulence" baseFrequency="0.1" numOctaves="2"/>
      </filter>
      <filter id="Turb3" filterUnits="objectBoundingBox"
              x="0%" y="0%" width="100%" height="100%">
        <feTurbulence type="turbulence" baseFrequency="0.05" numOctaves="8"/>
      </filter>
      <filter id="Turb4" filterUnits="objectBoundingBox"
              x="0%" y="0%" width="100%" height="100%">
        <feTurbulence type="fractalNoise" baseFrequency="0.1" numOctaves="4"/>
      </filter>
      <filter id="Turb5" filterUnits="objectBoundingBox"
              x="0%" y="0%" width="100%" height="100%">
        <feTurbulence type="fractalNoise" baseFrequency="0.4" numOctaves="4"/>
      </filter>
      <filter id="Turb6" filterUnits="objectBoundingBox"
              x="0%" y="0%" width="100%" height="100%">
        <feTurbulence type="fractalNoise" baseFrequency="0.1" numOctaves="1"/>
      </filter>
    </defs>
    <rect x="1" y="1" width="448" height="323"
          fill="none" stroke="blue" stroke-width="1"  />
    <rect x="25" y="25" width="100" height="75" filter="url(#Turb1)"  />
    <text x="75" y="117">type=turbulence</text>
    <text x="75" y="129">baseFrequency=0.05</text>
    <text x="75" y="141">numOctaves=2</text>
    <rect x="175" y="25" width="100" height="75" filter="url(#Turb2)"  />
    <text x="225" y="117">type=turbulence</text>
    <text x="225" y="129">baseFrequency=0.1</text>
    <text x="225" y="141">numOctaves=2</text>
    <rect x="325" y="25" width="100" height="75" filter="url(#Turb3)"  />
    <text x="375" y="117">type=turbulence</text>
    <text x="375" y="129">baseFrequency=0.05</text>
    <text x="375" y="141">numOctaves=8</text>
    <rect x="25" y="180" width="100" height="75" filter="url(#Turb4)"  />
    <text x="75" y="272">type=fractalNoise</text>
    <text x="75" y="284">baseFrequency=0.1</text>
    <text x="75" y="296">numOctaves=4</text>
    <rect x="175" y="180" width="100" height="75" filter="url(#Turb5)"  />
    <text x="225" y="272">type=fractalNoise</text>
    <text x="225" y="284">baseFrequency=0.4</text>
    <text x="225" y="296">numOctaves=4</text>
    <rect x="325" y="180" width="100" height="75" filter="url(#Turb6)"  />
    <text x="375" y="272">type=fractalNoise</text>
    <text x="375" y="284">baseFrequency=0.1</text>
    <text x="375" y="296">numOctaves=1</text>
  </g>
</svg>
```



Example feTurbulence

View this example as SVG (SVG-enabled browsers only)

15.25 Filter Module

Elements	Attributes	Content Model
filter	Core.attrib, XLink.attrib, External.attrib, Style.attrib, Presentation.attrib, filterUnits, primitiveUnits, x, y, width, height, filterRes	(Description.class | FilterPrimitive.class | Animation.class)*
feBlend	Core.attrib, FilterColor.attrib, FilterPrimitiveWithIn.attrib, in2, mode	(Animation.class)*
feColorMatrix	Core.attrib, FilterColor.attrib, FilterPrimitiveWithIn.attrib, type, values	(Animation.class)*
feComponentTransfer	Core.attrib, FilterColor.attrib, FilterPrimitiveWithIn.attrib	(feFuncR?, feFuncG?, feFuncB?, feFuncA?)
feComposite	Core.attrib, FilterColor.attrib, FilterPrimitiveWithIn.attrib, in2, operator, k1, k2, k3, k4	(Animation.class)*
feConvolveMatrix	Core.attrib, FilterColor.attrib, FilterPrimitiveWithIn.attrib, order, kernelMatrix, divisor, bias, targetX, targetY, edgeMode, kernelUnitLength, preserveAlpha	(Animation.class)*
feDiffuseLighting	Core.attrib, FilterColor.attrib, FilterPrimitiveWithIn.attrib, Style.attrib, Paint.attrib, lighting-color, surfaceScale, diffuseConstant, kernelUnitLength	((feDistantLight | fePointLight | feSpotLight), (Animation.class)*)
feDisplacementMap	Core.attrib, FilterColor.attrib, FilterPrimitiveWithIn.attrib, in2, scale, xChannelSelector, yChannelSelector	(Animation.class)*
feFlood	Core.attrib, FilterColor.attrib, FilterPrimitiveWithIn.attrib, Style.attrib, Paint.attrib, flood-color, flood-opacity	(Animation.class)*
feGaussianBlur	Core.attrib, FilterColor.attrib, FilterPrimitiveWithIn.attrib, stdDeviation	(Animation.class)*
feImage	Core.attrib, XLink.attrib, FilterColor.attrib, FilterPrimitive.attrib, External.attrib, Style.attrib, Presentation.attrib	(Animation.class)*
feMerge	Core.attrib, in	(Animation.class)*
feMergeNode	Core.attrib, FilterColor.attrib, FilterPrimitive.attrib	(feMergeNode)*
feMorphology	Core.attrib, FilterColor.attrib, FilterPrimitiveWithIn.attrib, operator, radius	(Animation.class)*
feOffset	Core.attrib, FilterColor.attrib, FilterPrimitiveWithIn.attrib, dx, dy	(Animation.class)*
feSpecularLighting	Core.attrib, FilterColor.attrib, FilterPrimitiveWithIn.attrib, Style.attrib, Paint.attrib, lighting-color, surfaceScale, specularConstant, specularExponent, kernelUnitLength	((feDistantLight | fePointLight | feSpotLight), (Animation.class)*)
feTile	Core.attrib, FilterColor.attrib, FilterPrimitiveWithIn.attrib	(Animation.class)*
feTurbulence	Core.attrib, FilterColor.attrib, FilterPrimitive.attrib, baseFrequency, numOctaves, seed, stitchTiles, type	(Animation.class)*
feDistantLight	Core.attrib, azimuth, elevation	(Animation.class)*
fePointLight	Core.attrib, x, y, z	(Animation.class)*

feSpotLight	Core.attrib, x, y, z, pointsAtX, pointsAtY, pointsAtZ, specularExponent, limitingConeAngle	(Animation.class)*
feFuncR	Core.attrib, type, tableValues, slope, intercept, amplitude, exponent, offset	(Animation.class)*
feFuncG	Core.attrib, type, tableValues, slope, intercept, amplitude, exponent, offset	(Animation.class)*
feFuncB	Core.attrib, type, tableValues, slope, intercept, amplitude, exponent, offset	(Animation.class)*
feFuncA	Core.attrib, type, tableValues, slope, intercept, amplitude, exponent, offset	(Animation.class)*

15.25.1 Filter Content Sets

The Filter Module defines the Filter.class and FilterPrimitive.class content sets.

Content Set Name	Elements in Content Set
Filter.class	filter
FilterPrimitive.class	feBlend, feFlood, feColorMatrix, feComponentTransfer, feComposite, feConvolveMatrix, feDiffuseLighting, feDisplacementMap, feGaussianBlur, feImage, feMerge, feMorphology, feOffset, feSpecularLighting, feTile, feTurbulence

15.25.2 Filter Attribute Sets

The Filter Module defines the Filter.attrib, FilterColor.attrib, FilterPrimitive.attrib and FilterPrimitiveWithIn.attrib attribute sets.

Collection Name	Attributes in Collection
Filter.attrib	filter
FilterColor.attrib	color-interpolation-filters
FilterPrimitive.attrib	x, y, width, height, result
FilterPrimitiveWithIn.attrib	FilterPrimitive.attrib, in

15.26 Basic Filter Module

Elements	Attributes	Content Model
filter	Core.attrib, XLink.attrib, External.attrib, Style.attrib, Presentation.attrib, filterUnits, primitiveUnits, x, y, width, height, filterRes	(Description.class | FilterPrimitive.class | Animation.class)*
feBlend	Core.attrib, FilterColor.attrib, FilterPrimitiveWithIn.attrib, in2, mode	(Animation.class)*
feColorMatrix	Core.attrib, FilterColor.attrib, FilterPrimitiveWithIn.attrib, type, values	(Animation.class)*
feComponentTransfer	Core.attrib, FilterColor.attrib, FilterPrimitiveWithIn.attrib	(feFuncR?, feFuncG?, feFuncB?, feFuncA?)
feComposite	Core.attrib, FilterColor.attrib, FilterPrimitiveWithIn.attrib, in2, operator, k1, k2, k3, k4	(Animation.class)*
feFlood	Core.attrib, FilterColor.attrib, FilterPrimitiveWithIn.attrib, Style.attrib, Paint.attrib, flood-color, flood-opacity	(Animation.class)*
feGaussianBlur	Core.attrib, FilterColor.attrib, FilterPrimitiveWithIn.attrib, stdDeviation	(Animation.class)*

feImage	Core.attrib, XLinkEmbed.attrib, FilterColor.attrib, FilterPrimitive.attrib, External.attrib, Style.attrib, Presentation.attrib	(Animation.class)*
feMerge	Core.attrib, in	(Animation.class)*
feMergeNode	Core.attrib, FilterColor.attrib, FilterPrimitive.attrib	(feMergeNode)*
feOffset	Core.attrib, FilterColor.attrib, FilterPrimitiveWithIn.attrib, dx, dy	(Animation.class)*
feTile	Core.attrib, FilterColor.attrib, FilterPrimitiveWithIn.attrib	(Animation.class)*
feFuncR	Core.attrib, type, tableValues, slope, intercept, amplitude, exponent, offset	(Animation.class)*
feFuncG	Core.attrib, type, tableValues, slope, intercept, amplitude, exponent, offset	(Animation.class)*
feFuncB	Core.attrib, type, tableValues, slope, intercept, amplitude, exponent, offset	(Animation.class)*
feFuncA	Core.attrib, type, tableValues, slope, intercept, amplitude, exponent, offset	(Animation.class)*

15.26.1 Basic Filter Content Sets

The Basic Filter Module defines the Filter.class and FilterPrimitive.class content sets.

Content Set Name	Elements in Content Set
Filter.class	filter
FilterPrimitive.class	feBlend, feFlood, feColorMatrix, feComponentTransfer, feComposite, feGaussianBlur, feImage, feMerge, feOffset, feTile

15.26.2 Basic Filter Attribute Sets

The Basic Filter Module defines the Filter.attrib, FilterColor.attrib, FilterPrimitive.attrib and FilterPrimitiveWithIn.attrib attribute sets.

Collection Name	Attributes in Collection
Filter.attrib	filter
FilterColor.attrib	color-interpolation-filters
FilterPrimitive.attrib	x, y, width, height, result
FilterPrimitiveWithIn.attrib	FilterPrimitive.attrib, in

15.27 DOM interfaces

The following interfaces are defined below: SVGFilterElement, SVGFilterPrimitiveStandardAttributes, SVGFEBlendElement, SVGFEColorMatrixElement, SVGFEComponentTransferElement, SVGComponentTransferFunctionElement, SVGFEFuncRElement, SVGFEFuncGElement, SVGFEFuncBElement, SVGFEFuncAElement, SVGFECompositeElement, SVGFEConvolveMatrixElement, SVGFEDiffuseLightingElement, SVGFEDistantLightElement, SVGFEPointLightElement, SVGFESpotLightElement, SVGFEDisplacementMapElement, SVGFEFloodElement, SVGFEGaussianBlurElement, SVGFEImageElement, SVGFEMergeElement, SVGFEMergeNodeElement, SVGFEMorphologyElement, SVGFEOffsetElement, SVGFESpecularLightingElement, SVGFETileElement, SVGFETurbulenceElement.

Interface SVGFilterElement

The **SVGFilterElement** interface corresponds to the **'filter'** element.

IDL Definition

```
interface SVGFilterElement :
                SVGElement,
                SVGURIReference,
                SVGLangSpace,
                SVGExternalResourcesRequired,
                SVGStylable,
                SVGUnitTypes {
  readonly attribute SVGAnimatedEnumeration filterUnits;
  readonly attribute SVGAnimatedEnumeration primitiveUnits;
  readonly attribute SVGAnimatedLength      x;
  readonly attribute SVGAnimatedLength      y;
  readonly attribute SVGAnimatedLength      width;
  readonly attribute SVGAnimatedLength      height;
  readonly attribute SVGAnimatedInteger    filterResX;
  readonly attribute SVGAnimatedInteger    filterResY;
  void setFilterRes ( in unsigned long filterResX, in unsigned long filterResY );
};
```

Attributes

readonly SVGAnimatedEnumeration filterUnits

Corresponds to attribute **filterUnits** on the given **'filter'** element. Takes one of the constants defined in **SVGUnitTypes**.

readonly SVGAnimatedEnumeration primitiveUnits

Corresponds to attribute **primitiveUnits** on the given **'filter'** element. Takes one of the constants defined in **SVGUnitTypes**.

readonly SVGAnimatedLength x

Corresponds to attribute **x** on the given **'filter'** element.

readonly SVGAnimatedLength y

Corresponds to attribute **y** on the given **'filter'** element.

readonly SVGAnimatedLength width

Corresponds to attribute **width** on the given **'filter'** element.

readonly SVGAnimatedLength height

Corresponds to attribute **height** on the given **'filter'** element.

readonly SVGAnimatedInteger filterResX

Corresponds to attribute **filterRes** on the given **'filter'** element. Contains the X component of attribute **filterRes**.

readonly SVGAnimatedInteger filterResY

Corresponds to attribute **filterRes** on the given **'filter'** element. Contains the Y component (possibly computed automatically) of attribute **filterRes**.

Methods

setFilterRes

Sets the values for attribute **filterRes**.

Parameters

in unsigned long filterResX	The X component of attribute **filterRes**.
in unsigned long filterResY	The Y component of attribute **filterRes**.

No Return Value

No Exceptions

Interface SVGFilterPrimitiveStandardAttributes

This interface defines the set of DOM attributes that are common across the filter interfaces.

IDL Definition

```
interface SVGFilterPrimitiveStandardAttributes : SVGStylable {
  readonly attribute SVGAnimatedLength x;
  readonly attribute SVGAnimatedLength y;
  readonly attribute SVGAnimatedLength width;
  readonly attribute SVGAnimatedLength height;
  readonly attribute SVGAnimatedString result;
};
```

Attributes

readonly SVGAnimatedLength x

Corresponds to attribute **x** on the given element.

readonly SVGAnimatedLength y

Corresponds to attribute **y** on the given element.

readonly SVGAnimatedLength width

Corresponds to attribute **width** on the given element.

readonly SVGAnimatedLength height

Corresponds to attribute **height** on the given element.

readonly SVGAnimatedString result

Corresponds to attribute **result** on the given element.

Interface SVGFEBlendElement

The **SVGFEBlendElement** interface corresponds to the **'feBlend'** element.

IDL Definition

```
interface SVGFEBlendElement :
                SVGElement,
                SVGFilterPrimitiveStandardAttributes {
  // Blend Mode Types
  const unsigned short SVG_FEBLEND_MODE_UNKNOWN  = 0;
  const unsigned short SVG_FEBLEND_MODE_NORMAL   = 1;
  const unsigned short SVG_FEBLEND_MODE_MULTIPLY = 2;
  const unsigned short SVG_FEBLEND_MODE_SCREEN   = 3;
  const unsigned short SVG_FEBLEND_MODE_DARKEN   = 4;
  const unsigned short SVG_FEBLEND_MODE_LIGHTEN  = 5;
  readonly attribute SVGAnimatedString      in1;
  readonly attribute SVGAnimatedString      in2;
  readonly attribute SVGAnimatedEnumeration mode;
};
```

Definition group Blend Mode Types

Defined constants

SVG_FEBLEND_MODE_UNKNOWN	The type is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type.
SVG_FEBLEND_MODE_NORMAL	Corresponds to value **normal**.
SVG_FEBLEND_MODE_MULTIPLY	Corresponds to value **multiply**.
SVG_FEBLEND_MODE_SCREEN	Corresponds to value **screen**.
SVG_FEBLEND_MODE_DARKEN	Corresponds to value **darken**.

SVG_FEBLEND_MODE_LIGHTEN	Corresponds to value **lighten**.

Attributes

readonly SVGAnimatedString in1

Corresponds to attribute **in** on the given **'feBlend'** element.

readonly SVGAnimatedString in2

Corresponds to attribute **in2** on the given **'feBlend'** element.

readonly SVGAnimatedEnumeration mode

Corresponds to attribute **mode** on the given **'feBlend'** element. Takes one of the Blend Mode Types.

Interface SVGFEColorMatrixElement

The **SVGFEColorMatrixElement** interface corresponds to the **'feColorMatrix'** element.

IDL Definition

```
interface SVGFEColorMatrixElement :
                SVGElement,
                SVGFilterPrimitiveStandardAttributes {
  // Color Matrix Types
  const unsigned short SVG_FECOLORMATRIX_TYPE_UNKNOWN          = 0;
  const unsigned short SVG_FECOLORMATRIX_TYPE_MATRIX           = 1;
  const unsigned short SVG_FECOLORMATRIX_TYPE_SATURATE         = 2;
  const unsigned short SVG_FECOLORMATRIX_TYPE_HUEROTATE        = 3;
  const unsigned short SVG_FECOLORMATRIX_TYPE_LUMINANCETOALPHA = 4;
  readonly attribute SVGAnimatedString      in1;
  readonly attribute SVGAnimatedEnumeration type;
  readonly attribute SVGAnimatedNumberList  values;
};
```

Definition group Color Matrix Types

Defined constants

SVG_FECOLORMATRIX_TYPE_UNKNOWN	The type is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type.
SVG_FECOLORMATRIX_TYPE_MATRIX	Corresponds to value **matrix**.
SVG_FECOLORMATRIX_TYPE_SATURATE	Corresponds to value **saturate**.
SVG_FECOLORMATRIX_TYPE_HUEROTATE	Corresponds to value **hueRotate**.
SVG_FECOLORMATRIX_TYPE_LUMINANCETOALPHA	Corresponds to value **luminanceToAlpha**.

Attributes

readonly SVGAnimatedString in1

Corresponds to attribute **in** on the given **'feColorMatrix'** element.

readonly SVGAnimatedEnumeration type

Corresponds to attribute **type** on the given **'feColorMatrix'** element. Takes one of the Color Matrix Types.

readonly SVGAnimatedNumberList values

Corresponds to attribute **values** on the given **'feColorMatrix'** element.

Provides access to the contents of the **values** attribute.

Interface SVGFEComponentTransferElement

The **SVGFEComponentTransferElement** interface corresponds to the **'feComponentTransfer'** element.

IDL Definition

```
interface SVGFEComponentTransferElement : 
                SVGElement,
                SVGFilterPrimitiveStandardAttributes {
  readonly attribute SVGAnimatedString in1;
};
```

Attributes

readonly SVGAnimatedString in1

Corresponds to attribute **in** on the given **'feComponentTransfer'** element.

Interface SVGComponentTransferFunctionElement

This interface defines a base interface used by the component transfer function interfaces.

IDL Definition

```
interface SVGComponentTransferFunctionElement : SVGElement {
  // Component Transfer Types
  const unsigned short SVG_FECOMPONENTTRANSFER_TYPE_UNKNOWN  = 0;
  const unsigned short SVG_FECOMPONENTTRANSFER_TYPE_IDENTITY = 1;
  const unsigned short SVG_FECOMPONENTTRANSFER_TYPE_TABLE    = 2;
  const unsigned short SVG_FECOMPONENTTRANSFER_TYPE_DISCRETE    = 3;
  const unsigned short SVG_FECOMPONENTTRANSFER_TYPE_LINEAR   = 4;
  const unsigned short SVG_FECOMPONENTTRANSFER_TYPE_GAMMA    = 5;
  readonly attribute SVGAnimatedEnumeration type;
  readonly attribute SVGAnimatedNumberList  tableValues;
  readonly attribute SVGAnimatedNumber      slope;
  readonly attribute SVGAnimatedNumber      intercept;
  readonly attribute SVGAnimatedNumber      amplitude;
  readonly attribute SVGAnimatedNumber      exponent;
  readonly attribute SVGAnimatedNumber      offset;
};
```

Definition group Component Transfer Types

Defined constants

SVG_FECOMPONENTTRANSFER_TYPE_UNKNOWN	The type is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type.
SVG_FECOMPONENTTRANSFER_TYPE_IDENTITY	Corresponds to value **identity**.
SVG_FECOMPONENTTRANSFER_TYPE_TABLE	Corresponds to value **table**.
SVG_FECOMPONENTTRANSFER_TYPE_DISCRETE	Corresponds to value **discrete**.
SVG_FECOMPONENTTRANSFER_TYPE_LINEAR	Corresponds to value **linear**.
SVG_FECOMPONENTTRANSFER_TYPE_GAMMA	Corresponds to value **gamma**.

Attributes

readonly SVGAnimatedEnumeration type

Corresponds to attribute **type** on the given element. Takes one of the Component Transfer Types.

readonly SVGAnimatedNumberList tableValues

Corresponds to attribute **tableValues** on the given element.

readonly SVGAnimatedNumber slope

Corresponds to attribute **slope** on the given element.

readonly SVGAnimatedNumber intercept

Corresponds to attribute **intercept** on the given element.

readonly SVGAnimatedNumber amplitude

Corresponds to attribute **amplitude** on the given element.

readonly SVGAnimatedNumber exponent

Corresponds to attribute **exponent** on the given element.

readonly SVGAnimatedNumber offset

Corresponds to attribute **offset** on the given element.

Interface SVGFEFuncRElement

The **SVGFEFuncRElement** interface corresponds to the **'feFuncR'** element.

IDL Definition

```
interface SVGFEFuncRElement : SVGComponentTransferFunctionElement {};
```

Interface SVGFEFuncGElement

The **SVGFEFuncGElement** interface corresponds to the **'feFuncG'** element.

IDL Definition

```
interface SVGFEFuncGElement : SVGComponentTransferFunctionElement {};
```

Interface SVGFEFuncBElement

The **SVGFEFuncBElement** interface corresponds to the **'feFuncB'** element.

IDL Definition

```
interface SVGFEFuncBElement : SVGComponentTransferFunctionElement {};
```

Interface SVGFEFuncAElement

The **SVGFEFuncAElement** interface corresponds to the **'feFuncA'** element.

IDL Definition

```
interface SVGFEFuncAElement : SVGComponentTransferFunctionElement {};
```

Interface SVGFECompositeElement

The **SVGFECompositeElement** interface corresponds to the **'feComposite'** element.

IDL Definition

```
interface SVGFECompositeElement : 
                SVGElement,
                SVGFilterPrimitiveStandardAttributes {
  // Composite Operators
  const unsigned short SVG_FECOMPOSITE_OPERATOR_UNKNOWN    = 0;
  const unsigned short SVG_FECOMPOSITE_OPERATOR_OVER       = 1;
  const unsigned short SVG_FECOMPOSITE_OPERATOR_IN         = 2;
  const unsigned short SVG_FECOMPOSITE_OPERATOR_OUT        = 3;
  const unsigned short SVG_FECOMPOSITE_OPERATOR_ATOP       = 4;
  const unsigned short SVG_FECOMPOSITE_OPERATOR_XOR        = 5;
  const unsigned short SVG_FECOMPOSITE_OPERATOR_ARITHMETIC = 6;
  readonly attribute SVGAnimatedString      in1;
  readonly attribute SVGAnimatedString      in2;
  readonly attribute SVGAnimatedEnumeration operator;
  readonly attribute SVGAnimatedNumber      k1;
  readonly attribute SVGAnimatedNumber      k2;
  readonly attribute SVGAnimatedNumber      k3;
  readonly attribute SVGAnimatedNumber      k4;
};
```

Definition group Composite Operators

Defined constants

SVG_FECOMPOSITE_OPERATOR_UNKNOWN	The type is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type.
SVG_FECOMPOSITE_OPERATOR_OVER	Corresponds to value **over**.
SVG_FECOMPOSITE_OPERATOR_IN	Corresponds to value **in**.
SVG_FECOMPOSITE_OPERATOR_OUT	Corresponds to value **out**.
SVG_FECOMPOSITE_OPERATOR_ATOP	Corresponds to value **atop**.
SVG_FECOMPOSITE_OPERATOR_XOR	Corresponds to value **xor**.
SVG_FECOMPOSITE_OPERATOR_ARITHMETIC	Corresponds to value **arithmetic**.

Attributes

readonly SVGAnimatedString in1

Corresponds to attribute **in** on the given **'feComposite'** element.

readonly SVGAnimatedString in2

Corresponds to attribute **in2** on the given **'feComposite'** element.

readonly SVGAnimatedEnumeration operator

Corresponds to attribute **operator** on the given **'feComposite'** element. Takes one of the Composite Operators.

readonly SVGAnimatedNumber k1

Corresponds to attribute **k1** on the given **'feComposite'** element.

readonly SVGAnimatedNumber k2

Corresponds to attribute **k2** on the given **'feComposite'** element.

readonly SVGAnimatedNumber k3

Corresponds to attribute **k3** on the given **'feComposite'** element.

readonly SVGAnimatedNumber k4

Corresponds to attribute **k4** on the given **'feComposite'** element.

Interface SVGFEConvolveMatrixElement

The **SVGFEConvolveMatrixElement** interface corresponds to the **'feConvolveMatrix'** element.

IDL Definition

```
interface SVGFEConvolveMatrixElement : 
                SVGElement,
                SVGFilterPrimitiveStandardAttributes {
  // Edge Mode Values
  const unsigned short SVG_EDGEMODE_UNKNOWN   = 0;
  const unsigned short SVG_EDGEMODE_DUPLICATE = 1;
  const unsigned short SVG_EDGEMODE_WRAP      = 2;
  const unsigned short SVG_EDGEMODE_NONE      = 3;
  readonly attribute SVGAnimatedInteger     orderX;
  readonly attribute SVGAnimatedInteger     orderY;
  readonly attribute SVGAnimatedNumberList  kernelMatrix;
  readonly attribute SVGAnimatedNumber      divisor;
  readonly attribute SVGAnimatedNumber      bias;
  readonly attribute SVGAnimatedInteger     targetX;
  readonly attribute SVGAnimatedInteger     targetY;
  readonly attribute SVGAnimatedEnumeration edgeMode;
  readonly attribute SVGAnimatedNumber      kernelUnitLengthX;
  readonly attribute SVGAnimatedNumber      kernelUnitLengthY;
  readonly attribute SVGAnimatedBoolean     preserveAlpha;
};
```

Definition group Edge Mode Values

Defined constants

SVG_EDGEMODE_UNKNOWN	The type is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type.
SVG_EDGEMODE_DUPLICATE	Corresponds to value **duplicate**.
SVG_EDGEMODE_WRAP	Corresponds to value **wrap**.
SVG_EDGEMODE_NONE	Corresponds to value **none**.

Attributes

readonly SVGAnimatedInteger orderX

Corresponds to attribute **order** on the given **'feConvolveMatrix'** element.

readonly SVGAnimatedInteger orderY

Corresponds to attribute **order** on the given **'feConvolveMatrix'** element.

readonly SVGAnimatedNumberList kernelMatrix

Corresponds to attribute **kernelMatrix** on the given element.

readonly SVGAnimatedNumber divisor

Corresponds to attribute **divisor** on the given **'feConvolveMatrix'** element.

readonly SVGAnimatedNumber bias

Corresponds to attribute **bias** on the given **'feConvolveMatrix'** element.

readonly SVGAnimatedInteger targetX

Corresponds to attribute **targetX** on the given **'feConvolveMatrix'** element.

readonly SVGAnimatedInteger targetY

Corresponds to attribute **targetY** on the given **'feConvolveMatrix'** element.

readonly SVGAnimatedEnumeration edgeMode

Corresponds to attribute **edgeMode** on the given **'feConvolveMatrix'** element. Takes one of the Edge Mode Types.

readonly SVGAnimatedNumber kernelUnitLengthX

Corresponds to attribute **kernelUnitLength** on the given **'feConvolveMatrix'** element.

readonly SVGAnimatedNumber kernelUnitLengthY

Corresponds to attribute **kernelUnitLength** on the given **'feConvolveMatrix'** element.

readonly SVGAnimatedBoolean preserveAlpha

Corresponds to attribute **preserveAlpha** on the given **'feConvolveMatrix'** element.

Interface SVGFEDiffuseLightingElement

The **SVGFEDiffuseLightingElement** interface corresponds to the **'feDiffuseLighting'** element.

IDL Definition

```
interface SVGFEDiffuseLightingElement : 
                SVGElement,
                SVGFilterPrimitiveStandardAttributes {
  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber surfaceScale;
  readonly attribute SVGAnimatedNumber diffuseConstant;
  readonly attribute SVGAnimatedNumber kernelUnitLengthX;
  readonly attribute SVGAnimatedNumber kernelUnitLengthY;
};
```

Attributes

readonly SVGAnimatedString in1

Corresponds to attribute **in** on the given **'feDiffuseLighting'** element.

readonly SVGAnimatedNumber surfaceScale

Corresponds to attribute **surfaceScale** on the given **'feDiffuseLighting'** element.

readonly SVGAnimatedNumber diffuseConstant

Corresponds to attribute **diffuseConstant** on the given **'feDiffuseLighting'** element.

readonly SVGAnimatedNumber kernelUnitLengthX

Corresponds to attribute **kernelUnitLength** on the given **'feDiffuseLighting'** element.

readonly SVGAnimatedNumber kernelUnitLengthY

Corresponds to attribute **kernelUnitLength** on the given **'feDiffuseLighting'** element.

Interface SVGFEDistantLightElement

The **SVGFEDistantLightElement** interface corresponds to the **'feDistantLight'** element.

IDL Definition

```
interface SVGFEDistantLightElement : SVGElement {
  readonly attribute SVGAnimatedNumber azimuth;
  readonly attribute SVGAnimatedNumber elevation;
};
```

Attributes

readonly SVGAnimatedNumber azimuth

Corresponds to attribute **azimuth** on the given **'feDistantLight'** element.

readonly SVGAnimatedNumber elevation

Corresponds to attribute **elevation** on the given **'feDistantLight'** element.

Interface SVGFEPointLightElement

The **SVGFEPointLightElement** interface corresponds to the **'fePointLight'** element.

IDL Definition

```
interface SVGFEPointLightElement : SVGElement {
  readonly attribute SVGAnimatedNumber x;
  readonly attribute SVGAnimatedNumber y;
  readonly attribute SVGAnimatedNumber z;
};
```

Attributes

readonly SVGAnimatedNumber x

Corresponds to attribute **x** on the given **'fePointLight'** element.

readonly SVGAnimatedNumber y

Corresponds to attribute **y** on the given **'fePointLight'** element.

readonly SVGAnimatedNumber z

Corresponds to attribute **z** on the given **'fePointLight'** element.

Interface SVGFESpotLightElement

The **SVGFESpotLightElement** interface corresponds to the **'feSpotLight'** element.

IDL Definition

```
interface SVGFESpotLightElement : SVGElement {
  readonly attribute SVGAnimatedNumber x;
  readonly attribute SVGAnimatedNumber y;
  readonly attribute SVGAnimatedNumber z;
  readonly attribute SVGAnimatedNumber pointsAtX;
  readonly attribute SVGAnimatedNumber pointsAtY;
  readonly attribute SVGAnimatedNumber pointsAtZ;
  readonly attribute SVGAnimatedNumber specularExponent;
  readonly attribute SVGAnimatedNumber limitingConeAngle;
};
```

Attributes

readonly SVGAnimatedNumber x

Corresponds to attribute **x** on the given **'feSpotLight'** element.

readonly SVGAnimatedNumber y

Corresponds to attribute **y** on the given **'feSpotLight'** element.

readonly SVGAnimatedNumber z

Corresponds to attribute **z** on the given **'feSpotLight'** element.

readonly SVGAnimatedNumber pointsAtX

Corresponds to attribute **pointsAtX** on the given **'feSpotLight'** element.

readonly SVGAnimatedNumber pointsAtY

Corresponds to attribute **pointsAtY** on the given **'feSpotLight'** element.

readonly SVGAnimatedNumber pointsAtZ

Corresponds to attribute **pointsAtZ** on the given **'feSpotLight'** element.

readonly SVGAnimatedNumber specularExponent

Corresponds to attribute **specularExponent** on the given **'feSpotLight'** element.

readonly SVGAnimatedNumber limitingConeAngle

Corresponds to attribute **limitingConeAngle** on the given **'feSpotLight'** element.

Interface SVGFEDisplacementMapElement

The **SVGFEDisplacementMapElement** interface corresponds to the **'feDisplacementMap'** element.

IDL Definition

```
interface SVGFEDisplacementMapElement : 
                SVGElement,
                SVGFilterPrimitiveStandardAttributes {
  // Channel Selectors
  const unsigned short SVG_CHANNEL_UNKNOWN = 0;
  const unsigned short SVG_CHANNEL_R       = 1;
  const unsigned short SVG_CHANNEL_G       = 2;
  const unsigned short SVG_CHANNEL_B       = 3;
  const unsigned short SVG_CHANNEL_A       = 4;
  readonly attribute SVGAnimatedString      in1;
  readonly attribute SVGAnimatedString      in2;
  readonly attribute SVGAnimatedNumber      scale;
  readonly attribute SVGAnimatedEnumeration xChannelSelector;
  readonly attribute SVGAnimatedEnumeration yChannelSelector;
};
```

Definition group Channel Selectors

Defined constants

SVG_CHANNEL_UNKNOWN	The type is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type.
SVG_CHANNEL_R	Corresponds to value **R**.
SVG_CHANNEL_G	Corresponds to value **G**.
SVG_CHANNEL_B	Corresponds to value **B**.
SVG_CHANNEL_A	Corresponds to value **A**.

Attributes

readonly SVGAnimatedString in1

Corresponds to attribute **in** on the given **'feDisplacementMap'** element.

readonly SVGAnimatedString in2

Corresponds to attribute **in2** on the given **'feDisplacementMap'** element.

readonly SVGAnimatedNumber scale

Corresponds to attribute **scale** on the given **'feDisplacementMap'** element.

readonly SVGAnimatedEnumeration xChannelSelector

Corresponds to attribute **xChannelSelector** on the given **'feDisplacementMap'** element. Takes one of the Channel Selectors.

readonly SVGAnimatedEnumeration yChannelSelector

Corresponds to attribute **yChannelSelector** on the given **'feDisplacementMap'** element. Takes one of the Channel Selectors.

Interface SVGFEFloodElement

The **SVGFEFloodElement** interface corresponds to the **'feFlood'** element.

IDL Definition

```
interface SVGFEFloodElement : 
                SVGElement,
                SVGFilterPrimitiveStandardAttributes {
  readonly attribute SVGAnimatedString      in1;
};
```

Attributes

readonly SVGAnimatedString in1

Corresponds to attribute **in** on the given **'feFlood'** element.

Interface SVGFEGaussianBlurElement

The **SVGFEGaussianBlurElement** interface corresponds to the **'feGaussianBlur'** element.

IDL Definition

```
interface SVGFEGaussianBlurElement : 
                SVGElement,
                SVGFilterPrimitiveStandardAttributes {
  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber stdDeviationX;
  readonly attribute SVGAnimatedNumber stdDeviationY;
  void setStdDeviation ( in float stdDeviationX, in float stdDeviationY );
};
```

Attributes

readonly SVGAnimatedString in1

Corresponds to attribute **in** on the given **'feGaussianBlur'** element.

readonly SVGAnimatedNumber stdDeviationX

Corresponds to attribute **stdDeviation** on the given **'feGaussianBlur'** element. Contains the X component of attribute **stdDeviation**.

readonly SVGAnimatedNumber stdDeviationY

Corresponds to attribute **stdDeviation** on the given **'feGaussianBlur'** element. Contains the Y component (possibly computed automatically) of attribute **stdDeviation**.

Methods

setStdDeviation

Sets the values for attribute **stdDeviation**.

Parameters

in float stdDeviationX	The X component of attribute **stdDeviation**.
in float stdDeviationY	The Y component of attribute **stdDeviation**.

No Return Value

No Exceptions

Interface SVGFEImageElement

The **SVGFEImageElement** interface corresponds to the **'feImage'** element.

IDL Definition

```
interface SVGFEImageElement : 
                SVGElement,
                SVGURIReference,
                SVGLangSpace,
                SVGExternalResourcesRequired,
                SVGFilterPrimitiveStandardAttributes {
readonly attribute SVGAnimatedPreserveAspectRatio preserveAspectRatio;
};
```

Attributes

readonly SVGAnimatedPreserveAspectRatio preserveAspectRatio

Corresponds to attribute **preserveAspectRatio** on the given element.

Interface SVGFEMergeElement

The **SVGFEMergeElement** interface corresponds to the **'feMerge'** element.

IDL Definition

```
interface SVGFEMergeElement : 
                SVGElement,
                SVGFilterPrimitiveStandardAttributes {};
```

Interface SVGFEMergeNodeElement

The **SVGFEMergeNodeElement** interface corresponds to the **'feMergeNode'** element.

IDL Definition

```
interface SVGFEMergeNodeElement : SVGElement {
  readonly attribute SVGAnimatedString in1;
};
```

Attributes

readonly SVGAnimatedString in1

Corresponds to attribute **in** on the given **'feMergeNode'** element.

Interface SVGFEMorphologyElement

The **SVGFEMorphologyElement** interface corresponds to the **'feMorphology'** element.

IDL Definition

```
interface SVGFEMorphologyElement : 
                SVGElement,
                SVGFilterPrimitiveStandardAttributes {
  // Morphology Operators
  const unsigned short SVG_MORPHOLOGY_OPERATOR_UNKNOWN = 0;
  const unsigned short SVG_MORPHOLOGY_OPERATOR_ERODE   = 1;
  const unsigned short SVG_MORPHOLOGY_OPERATOR_DILATE  = 2;
  readonly attribute SVGAnimatedString      in1;
  readonly attribute SVGAnimatedEnumeration operator;
  readonly attribute SVGAnimatedNumber      radiusX;
  readonly attribute SVGAnimatedNumber      radiusY;
};
```

Definition group Morphology Operators

Defined constants

SVG_MORPHOLOGY_OPERATOR_UNKNOWN	The type is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type.
SVG_MORPHOLOGY_OPERATOR_ERODE	Corresponds to value **erode**.
SVG_MORPHOLOGY_OPERATOR_DILATE	Corresponds to value **dilate**.

Attributes

readonly SVGAnimatedString in1

Corresponds to attribute **in** on the given **'feMorphology'** element.

readonly SVGAnimatedEnumeration operator

Corresponds to attribute **operator** on the given **'feMorphology'** element. Takes one of the Morphology Operators.

readonly SVGAnimatedNumber radiusX

Corresponds to attribute **radius** on the given **'feMorphology'** element.

readonly SVGAnimatedNumber radiusY

Corresponds to attribute **radius** on the given **'feMorphology'** element.

Interface SVGFEOffsetElement

The **SVGFEOffsetElement** interface corresponds to the **'feOffset'** element.

IDL Definition

```
interface SVGFEOffsetElement : 
                SVGElement,
                SVGFilterPrimitiveStandardAttributes {
  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber dx;
  readonly attribute SVGAnimatedNumber dy;
};
```

Attributes

readonly SVGAnimatedString in1

Corresponds to attribute **in** on the given **'feOffset'** element.

readonly SVGAnimatedNumber dx

Corresponds to attribute **dx** on the given **'feOffset'** element.

readonly SVGAnimatedNumber dy

Corresponds to attribute **dy** on the given **'feOffset'** element.

Interface SVGFESpecularLightingElement

The **SVGFESpecularLightingElement** interface corresponds to the **'feSpecularLighting'** element.

IDL Definition

```
interface SVGFESpecularLightingElement :
                SVGElement,
                SVGFilterPrimitiveStandardAttributes {
  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber surfaceScale;
  readonly attribute SVGAnimatedNumber specularConstant;
  readonly attribute SVGAnimatedNumber specularExponent;
};
```

Attributes

readonly SVGAnimatedString in1

Corresponds to attribute **in** on the given **'feSpecularLighting'** element.

readonly SVGAnimatedNumber surfaceScale

Corresponds to attribute **surfaceScale** on the given **'feSpecularLighting'** element.

readonly SVGAnimatedNumber specularConstant

Corresponds to attribute **specularConstant** on the given **'feSpecularLighting'** element.

readonly SVGAnimatedNumber specularExponent

Corresponds to attribute **specularExponent** on the given **'feSpecularLighting'** element.

Interface SVGFETileElement

The **SVGFETileElement** interface corresponds to the **'feTile'** element.

IDL Definition

```
interface SVGFETileElement :
                SVGElement,
                SVGFilterPrimitiveStandardAttributes {
  readonly attribute SVGAnimatedString in1;
};
```

Attributes

readonly SVGAnimatedString in1

Corresponds to attribute **in** on the given **'feTile'** element.

Interface SVGFETurbulenceElement

The **SVGFETurbulenceElement** interface corresponds to the **'feTurbulence'** element.

IDL Definition

```
interface SVGFETurbulenceElement : 
                SVGElement,
                SVGFilterPrimitiveStandardAttributes {
  // Turbulence Types
  const unsigned short SVG_TURBULENCE_TYPE_UNKNOWN      = 0;
  const unsigned short SVG_TURBULENCE_TYPE_FRACTALNOISE = 1;
  const unsigned short SVG_TURBULENCE_TYPE_TURBULENCE   = 2;
  // Stitch Options
  const unsigned short SVG_STITCHTYPE_UNKNOWN  = 0;
  const unsigned short SVG_STITCHTYPE_STITCH   = 1;
  const unsigned short SVG_STITCHTYPE_NOSTITCH = 2;
  readonly attribute SVGAnimatedNumber      baseFrequencyX;
  readonly attribute SVGAnimatedNumber      baseFrequencyY;
  readonly attribute SVGAnimatedInteger     numOctaves;
  readonly attribute SVGAnimatedNumber      seed;
  readonly attribute SVGAnimatedEnumeration stitchTiles;
  readonly attribute SVGAnimatedEnumeration type;
};
```

Definition group Turbulence Types

Defined constants

SVG_TURBULENCE_TYPE_UNKNOWN — The type is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type.

SVG_TURBULENCE_TYPE_FRACTALNOISE — Corresponds to value **fractalNoise**.

SVG_TURBULENCE_TYPE_TURBULENCE — Corresponds to value **turbulence**.

Definition group Stitch Options

Defined constants

SVG_STITCHTYPE_UNKNOWN — The type is not one of predefined types. It is invalid to attempt to define a new value of this type or to attempt to switch an existing value to this type.

SVG_STITCHTYPE_STITCH — Corresponds to value **stitch**.

SVG_STITCHTYPE_NOSTITCH — Corresponds to value **noStitch**.

Attributes

readonly SVGAnimatedNumber baseFrequencyX

Corresponds to attribute **baseFrequencyX** on the given **'feTurbulence'** element.

readonly SVGAnimatedNumber baseFrequencyY

Corresponds to attribute **baseFrequencyY** on the given **'feTurbulence'** element.

readonly SVGAnimatedInteger numOctaves

Corresponds to attribute **numOctaves** on the given **'feTurbulence'** element.

readonly SVGAnimatedNumber seed

Corresponds to attribute **seed** on the given **'feTurbulence'** element.

readonly SVGAnimatedEnumeration stitchTiles

Corresponds to attribute **stitchTiles** on the given **'feTurbulence'** element. Takes one of the Stitching Options.

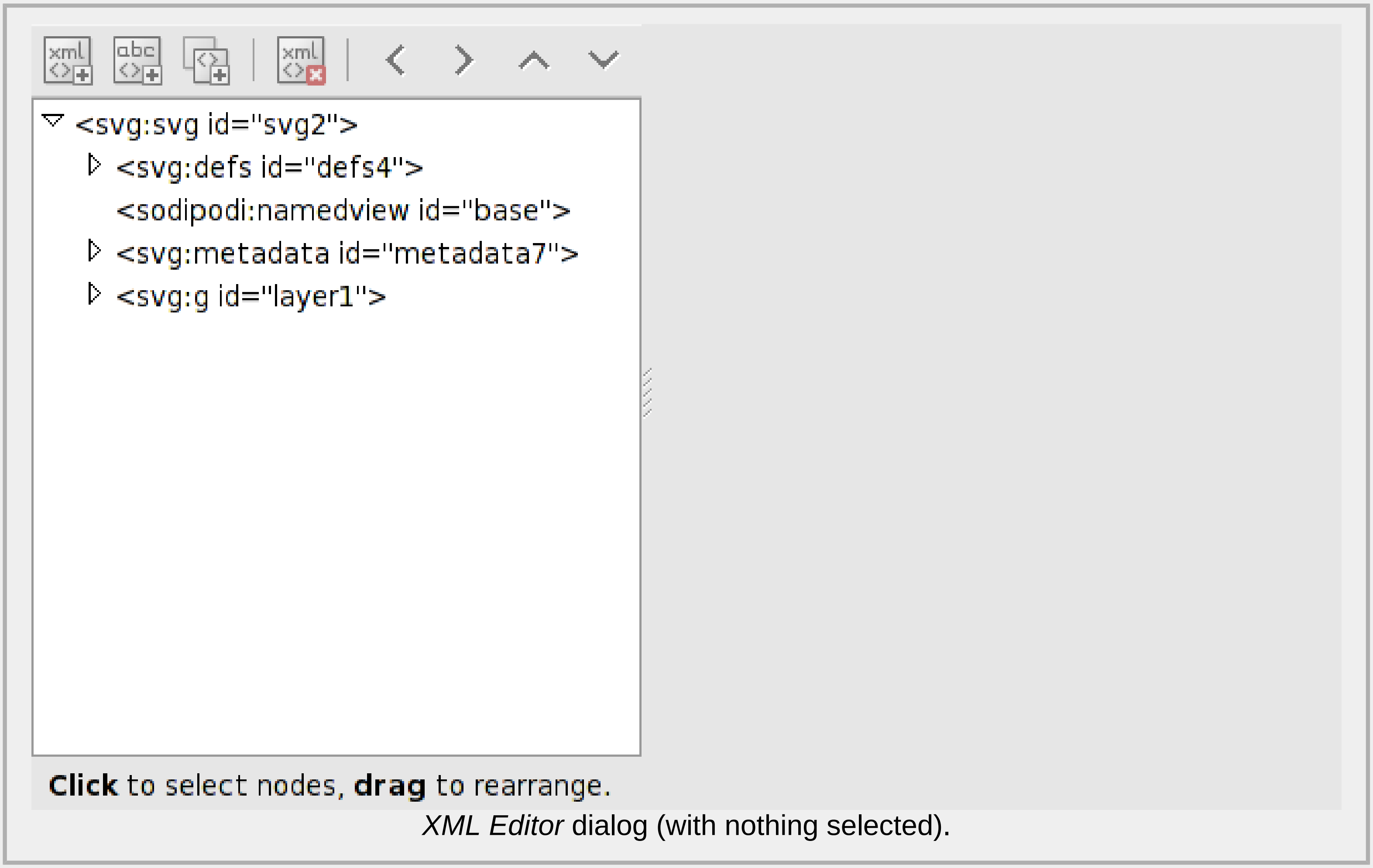
readonly SVGAnimatedEnumeration type

Corresponds to attribute **type** on the given **'feTurbulence'** element. Takes one of the Turbulence Types.

previous next contents elements attributes properties index

Basic Usage

To edit the *XML* file, open the *XML Editor* dialog (Edit → XML Editor... (**Shift**+**Ctrl**+**X**)). This will open a window like the following for an empty drawing.



```
▽ <svg:svg id="svg2">
    ▷ <svg:defs id="defs4">
      <sodipodi:namedview id="base">
    ▷ <svg:metadata id="metadata7">
    ▷ <svg:g id="layer1">
```

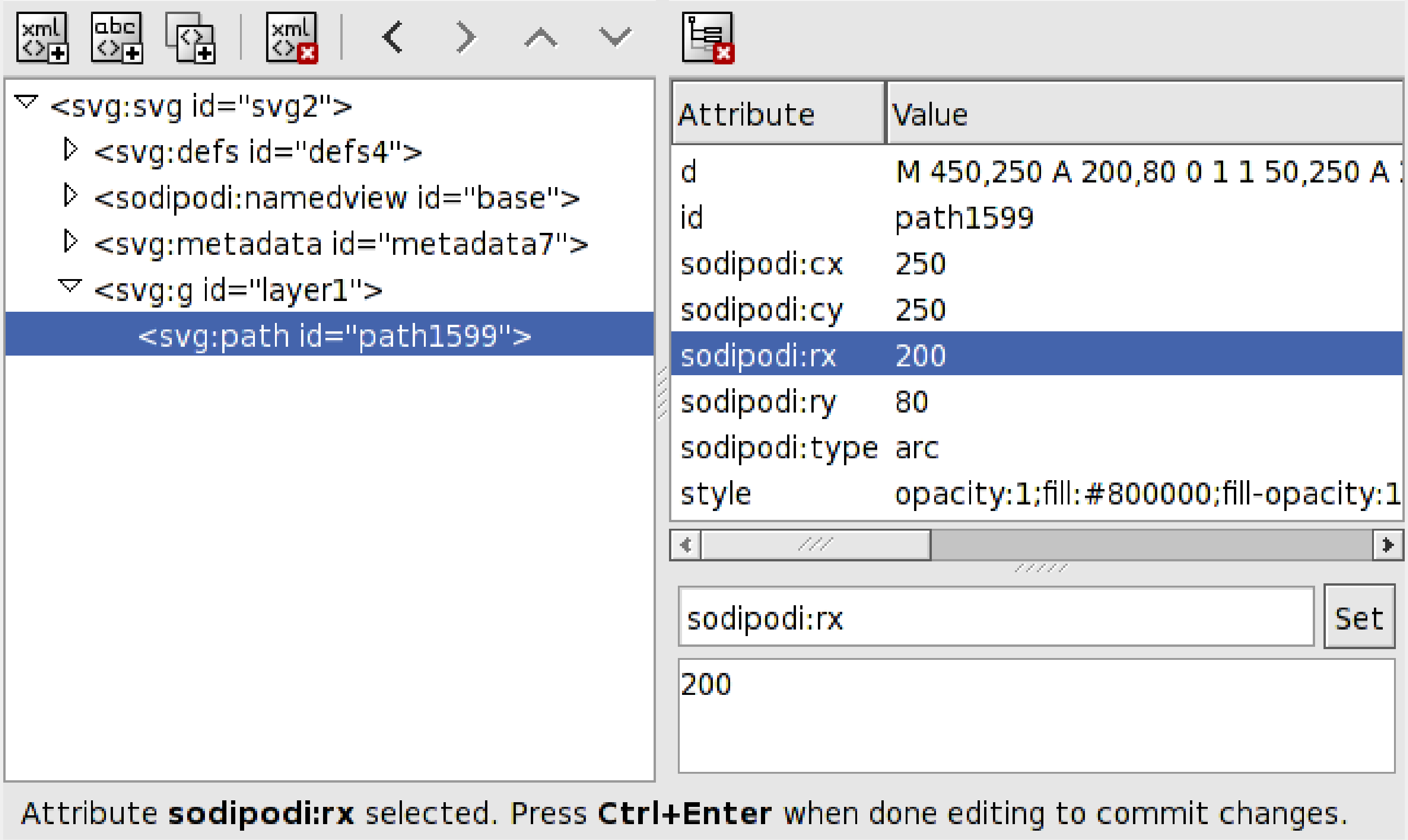
Click to select nodes, **drag** to rearrange.

XML Editor dialog (with nothing selected).

The “tree” structure of the empty drawing is shown on the right in the dialog. Even an empty Inkscape drawing contains information including an empty *Layer* ("layer1"). The *Layer*, like all the items listed, is represented by a “node” in the tree. If the layer contained objects, they would be represented by nodes under the layer's node. The objects under a particular node can be hidden in the tree view. To hide and unhide these objects click on the small triangles just in front of a node's name.

Upon adding an ellipse to the drawing, an entry (node) is added for the new ellipse under the formerly empty *Layer* (see next figure). The line is highlighted and the ellipse's parameters are shown on the left. Note that the name of the object is given in the highlighted line (in this case, "path1599").

Now, suppose you would like the ellipse to have a width of 400 pixels (i.e., a radius of 200 pixels in the x direction). You can specify this by clicking on the “sodipodi:rx” attribute. The attribute is shown below with the current value in the attribute entry box. Change that value to 200 and then click on the *Set* button (or use **Ctrl**+**Enter**).



```
▽ <svg:svg id="svg2">
    ▷ <svg:defs id="defs4">
    ▷ <sodipodi:namedview id="base">
    ▷ <svg:metadata id="metadata7">
    ▽ <svg:g id="layer1">
        <svg:path id="path1599">
```

Attribute	Value
d	M 450,250 A 200,80 0 1 1 50,250 A :
id	path1599
sodipodi:cx	250
sodipodi:cy	250
sodipodi:rx	200
sodipodi:ry	80
sodipodi:type	arc
style	opacity:1;fill:#800000;fill-opacity:1

sodipodi:rx

200

Attribute **sodipodi:rx** selected. Press **Ctrl+Enter** when done editing to commit changes.

XML Editor dialog after changing the width of the ellipse.

The ellipse should now be 400 pixels wide. You could also change the height (sodipodi:ry) and the location of the center of the ellipse (sodipodi:cx and sodipodi:cy). And you can change attributes like the *Fill* color and *Stroke style*.

Elements in the Inkscape file fall under two categories: those included in the *SVG* standard and those that are private to Inkscape. In the above example, the attributes with the “sodipodi” tag are internal to Inkscape (the “sodipodi” tag is the result of Inkscape being branched from the Sodipodi program). These attributes are used to calculate the “real” *SVG* path definition given by the 'd' attribute. (See the section called “Paths” in Appendix B, *File Format*.)

The Inkscape internal elements should be ignored by other *SVG* rendering programs. This may not always be true thus Inkscape includes the possibility to export a drawing without the Inkscape internal elements.

Sometimes it is useful to know what the allowed attributes are for a given type of object. The *SVG* standard is described in detail at the *Official W3C SVG* website. Note that not all the *SVG* standard is currently supported in Inkscape. It is possible, however, to add nonsupported attributes via the *XML Editor*. These attributes may not be displayed by Inkscape but will appear in any program that supports those attributes.

Chapter 14. XML Editor

Table of Contents

Editing XML Nodes

Inkscape » XML Editor » Editing XML Nodes

← Index →

Editing XML Nodes

The *XML Editor* dialog includes a number of clickable icons to manipulate the “nodes” in the *XML* tree. A description of each icon follows. Some of the things you can do with these commands may not be so sensible.

- Add XML element node. Add a node. For this to be useful, all the attributes appropriate to the type of node add must also be added.
- Add XML text node. Can also be used to edit the text in a text object.
- Duplicate XML node. Make a copy of the currently selected node including all its daughters. The new node will be placed at the same level and just after the original node. As each node must have a unique *ID*, Inkscape will assign a new *ID*. You can change the *ID* name if you wish.
- Delete XML element node. Delete a node and all its daughters.
- Unindent node. Move a node out one level. For a drawable object in a group, this is equivalent to removing that object from that group.
- Indent node. Move a node in one level. The node will be moved under the closest node above at the same level.
- Raise node. Move a node above the previous node with the same parent. Equivalent to changing the *z-order* when the two nodes are drawable objects.
- Lower node. Move a node below the next node with the same parent. Equivalent to changing the *z-order* when the two nodes are drawable objects.
- Delete the selected object attribute.

← Basic Usage

↑ Table of Contents

→ Examples

Examples

A few examples are given here to show the possibilities of “hand” editing the *XML* file.

Adding Color to a Marker Arrow

Markers on paths in Inkscape do not inherit the attributes of the path. This is most noticeable for colored paths where the markers are drawn in black. As of v0.45, Inkscape includes the *Color Markers to Match Stroke* effect for changing the color of markers to match the stroke color. This section is kept for pedagogical reasons.

To add color to a marker, open up the *XML Editor*. Select the path with the marker in the canvas window. In the “style” attribute for the line, locate the marker entry (marker-end:url (#Arrow2Lend) for example). Then expand the “<svg:defs>” line by clicking on the triangle at the beginning of the line (if not already expanded). You should see an entry for the marker. Select that entry. The attributes for the marker should be displayed on the right. Select the “style” attribute. Add “fill:#rrggbb” to the attributes in the entry box at the bottom right, where #rrggbb is the *RGB* color in *hexadecimal* form (obtainable from the attributes for the path).

The marker should change color. If is doesn't, then expand the "<svg:marker>" line. Select the path entry and remove any *Fill* and/or *Stroke paint*. For this change to show up, you must save and reopen the *SVG* file.

If you wish to have markers of the same type with different colors, then you must add copies in the <svg:defs> section. You can use the *Duplicate Node* icon to duplicate a marker entry.

Rename the new entry to a suitable name and change the reference to the marker in the path object you want the marker to be associated with. Again the file must be closed and reopened for the changes to be seen.

You are not limited to changing color. You can change other attributes such as the marker size.

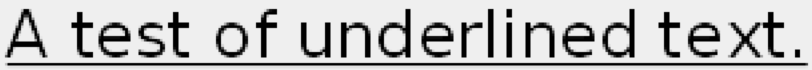
A red line with a red marker.

Underlined Text

Underlined text cannot be added through the normal Inkscape interface, nor will Inkscape display underlines. But you can add underlined text that will be displayed properly by another *SVG* program.

To underline text, open the *XML Editor*. Select the text you wish underlined. Go to the "<svg:tspan>" object found inside an "<svg:text>" tag. If you are selecting part of the text, you may need to add some attribute temporarily (color for example) to create a corresponding “tspan” object; the color can be removed later. Add to the style: “text-decoration: underline”.

Here is an example as rendered by the Squiggle program.



A test of underlined text.

Underlined text example.

Editing XML Nodes

Table of Contents

Chapter 15. Tiling

Get the book.

Inkscape: Guide to a Vector Drawing Program

[Get the book.](http://tavmjong.free.fr)

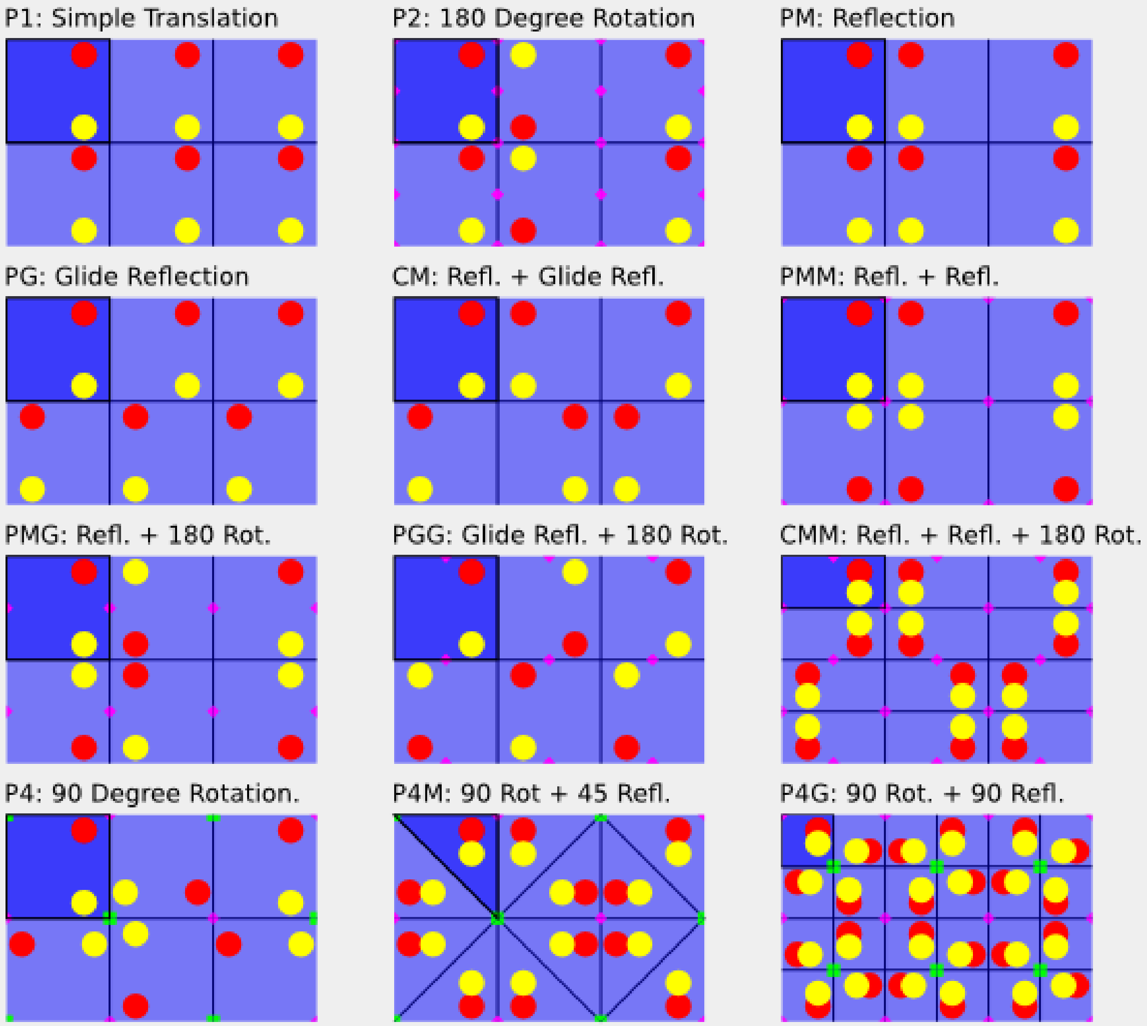
Inkscape » Tiling » Symmetry Tab

Index

Symmetry Tab

The *Symmetries* tab is at the heart of the tiling process. Understanding the different symmetries is necessary to have full control over the outcome of a tiling. The symmetry of the tiling is selected from the pull-down menu under the *Symmetries* tab (see above figure).

There are three regular geometric shapes that can be replicated to cover a surface completely (without gaps or overlaps). These shapes are: triangles, rectangles (parallelograms), and hexagons. A complete set of tiling symmetries requires taking these shapes and adding rotations and reflections. It is known that there are 17 such tiling symmetries. (See: Wikipedia entry.) All 17 symmetries are included in the Inkscape *Create Tiled Clones* dialog. The symmetries are shown next.



P1: Simple Translation

P2: 180 Degree Rotation

PM: Reflection

PG: Glide Reflection

CM: Refl. + Glide Refl.

PMM: Refl. + Refl.

PMG: Refl. + 180 Rot.

PGG: Glide Refl. + 180 Rot.

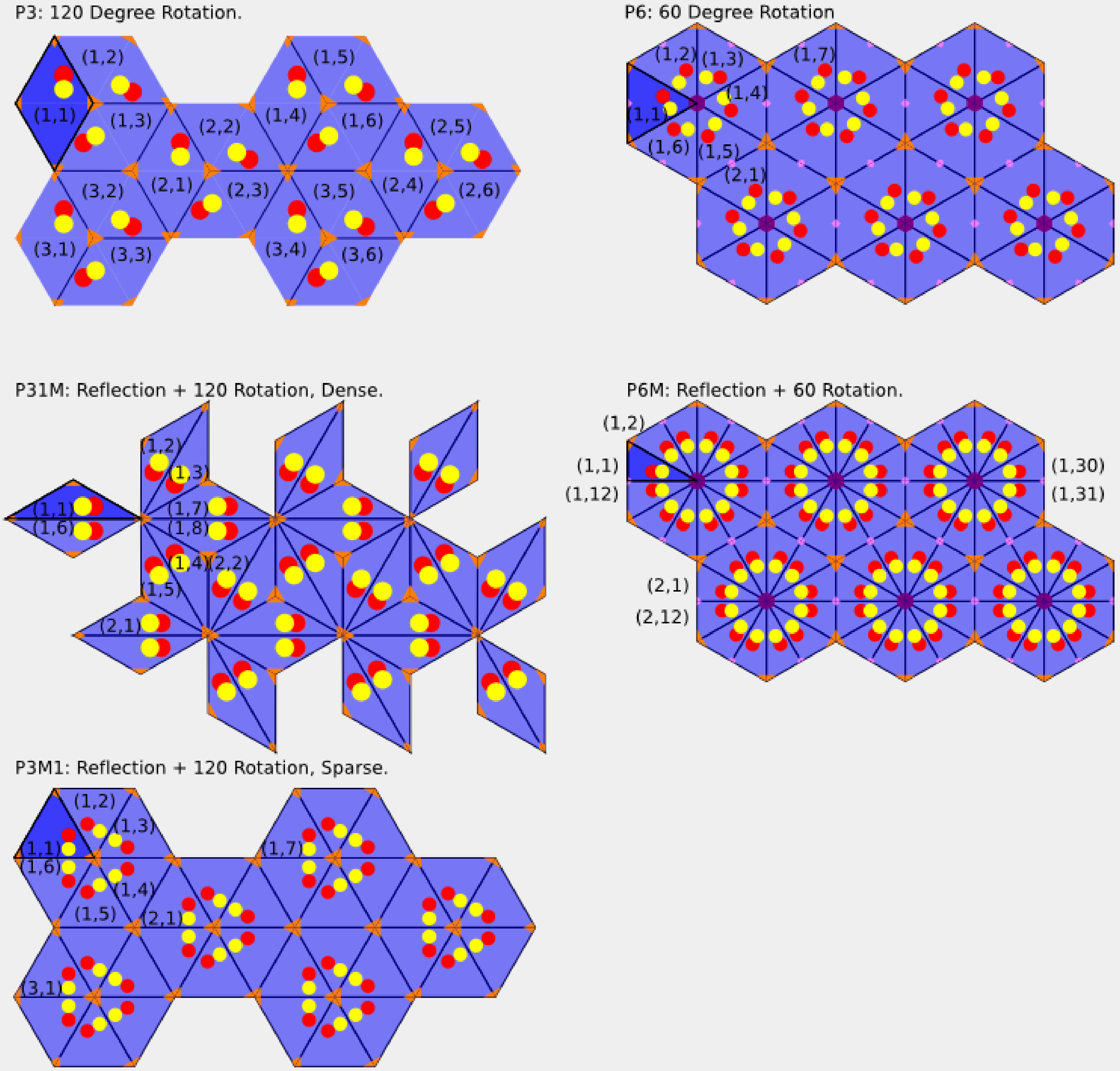
CMM: Refl. + Refl. + 180 Rot.

P4: 90 Degree Rotation.

P4M: 90 Rot + 45 Refl.

P4G: 90 Rot. + 90 Refl.

Tilings based on a rectangle tile (or 45-45-90 degree triangle). The outlined dark blue tile is the basic unit. Red and yellow dots show the reflection and rotation symmetries. Points of two-fold and four-fold rotational symmetry are shown by pink diamonds and green squares, respectively. The P1 and P2 symmetries also work with parallelograms.



P3: 120 Degree Rotation.

P6: 60 Degree Rotation

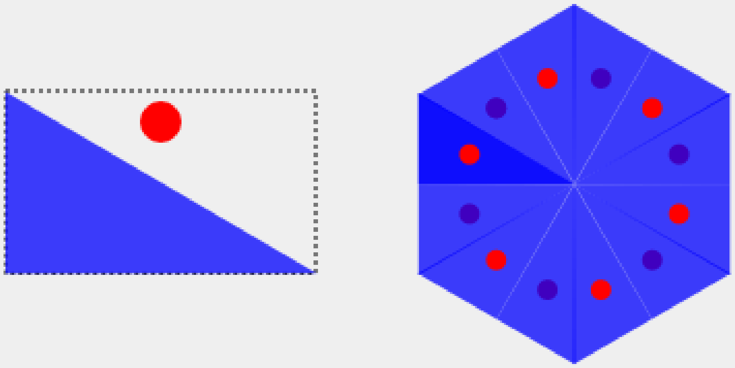
P31M: Reflection + 120 Rotation, Dense.

P6M: Reflection + 60 Rotation.

P3M1: Reflection + 120 Rotation, Sparse.

Tiling based on regular subdivisions of a hexagonal. The outlined dark blue tile is the basic unit. All tiling have points of three-fold rotational symmetry (orange triangles). Two also have two-fold and six-fold rotational symmetries (pink diamonds and purple hexagons). The pairs of numbers indicate the row and column numbers.

The basic tile for each of the 17 symmetries is shown in dark blue in the preceding figures. Inkscape uses the *bounding box* of an object to determine the basic tile size. For rectangular base tiles, the *bounding box* corresponds to the base tile. However, for triangular base tiles, the base tile covers only part of the *bounding box* area. This can result in tiles “overlapping” if an object extends outside the base tile shape (but is still within the *bounding box*) as in the tiling in the introduction to this chapter. Overlapping can also occur if the base tile is altered after the tiles are positioned.



On the left is a triangle and circle that are grouped together. The triangle corresponds to the base tile for a P6M symmetry. Note that the red circle is outside the base triangle but is still within the *bounding box* of the group (and triangle). On the right is a P6M tiling with the triangle and circle. Note how the red circle ends up above some but below other triangles as determined by the order in which the tiling is made.

As of v0.46, Inkscape always uses the *Geometric* bounding box to determine the tile size. This avoids problems when creating a triangular tile with a *Stroke* where the *Visual* bounding box doesn't have the same width to height ratio as the *Geometric* bounding box.

If you need to adjust the base tile size after having creating a tiling, you can use the *XML Editor* dialog to change the parameters “inkscape:tile-h” and “inkscape:tile-w” (these will appear after you have cloned the object and are used only if the *Use saved size and position of the tile* button is checked).

Chapter 15. Tiling

Table of Contents

Shift Tab

Inkscape » Tiling » Shift Tab

Index

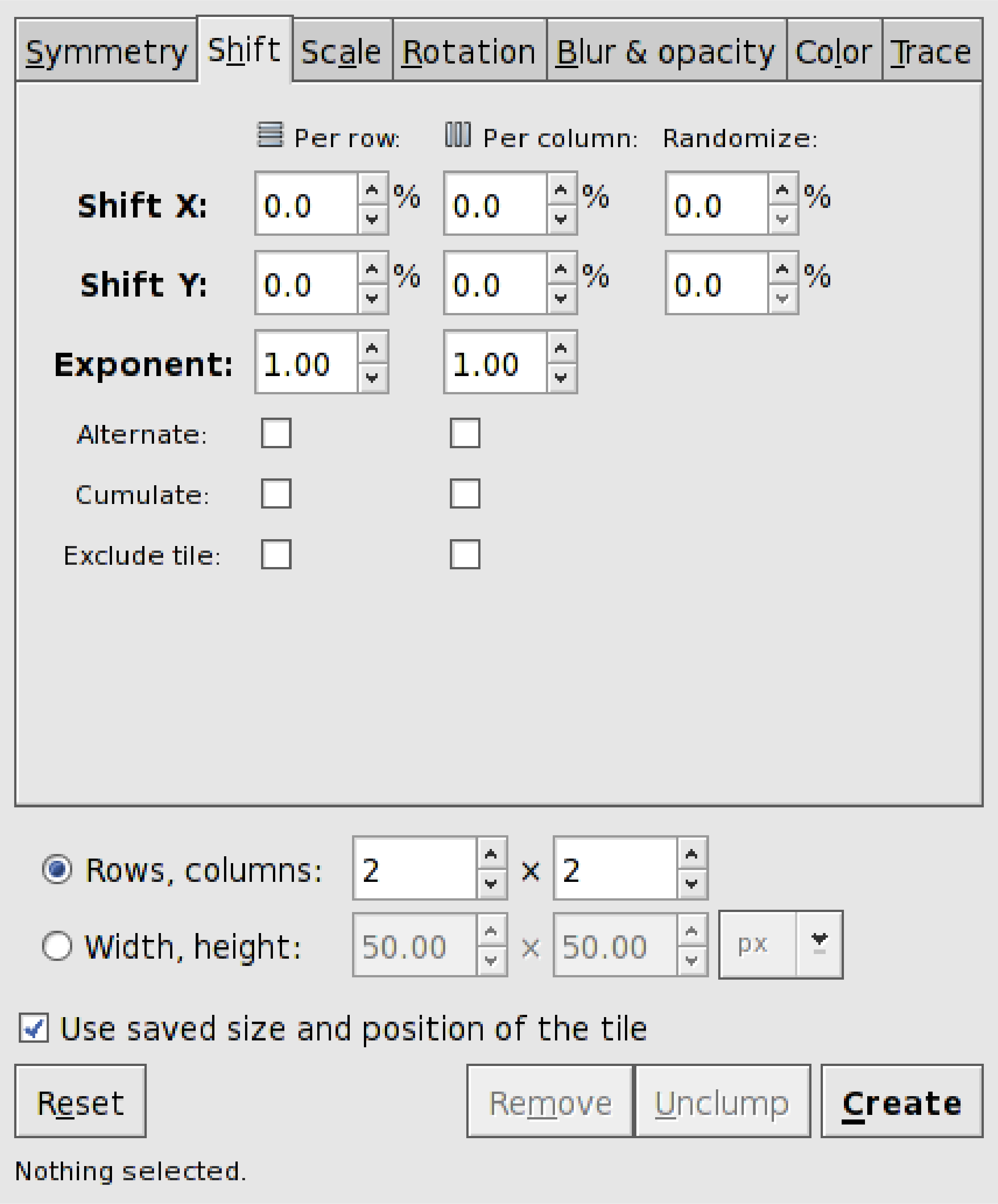
Shift Tab

The *Shift* tab allows one to vary the spacing between tiles. With the default parameters, rectangular tiles are arranged so that their *Geometric* bounding boxes are touching. The following options are available to add or subtract space between the tiles:

- *Shift X*, *Shift Y*: Adds (or subtracts) to the tile spacing in units of *bounding box* width and height. A random factor can also be added.
- *Exponent*: Changes the exponent factor *z* so that position of each tile is *x* (or *y*)= (1 + “shift”)^{z}.
- *Alternate*: The shift alternates between being added and subtracted.
- *New in v0.46:*

  Cumulate: The previous shift is added to the new shift. For example, if there was a *Shift X* of 10%, normally the space between subsequent tiles would be 10%, 20%, 30%, etc. With this option, the shifts become 10%, (10+20)% (10+20+30)%, etc. This is useful when one is also scaling the tiles to keep the tile spacing constant. (See the *Scale Tab* section for an example.)
- *New in v0.46:*

  Exclude tile: The tile width or height is excluded in the calculation of tile spacing. This is useful when using the *Rotation* option to put tiles on a circle. In this case, it is a shortcut for specifying a -100% shift.



	Per row:	Per column:	Randomize:
Shift X:	0.0 %	0.0 %	0.0 %
Shift Y:	0.0 %	0.0 %	0.0 %
Exponent:	1.00	1.00	
Alternate:	□	□	
Cumulate:	□	□	
Exclude tile:	□	□	

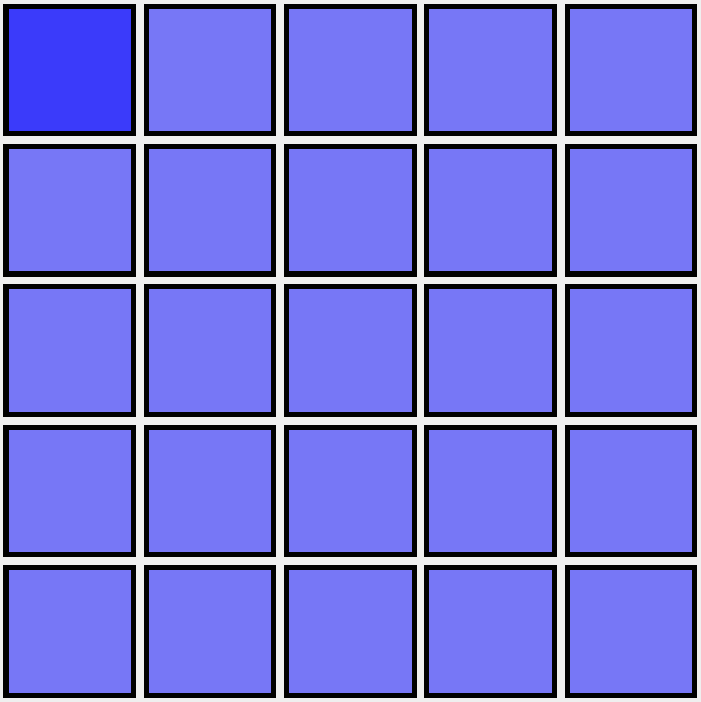
● Rows, columns: 2 × 2

○ Width, height: 50.00 × 50.00 px

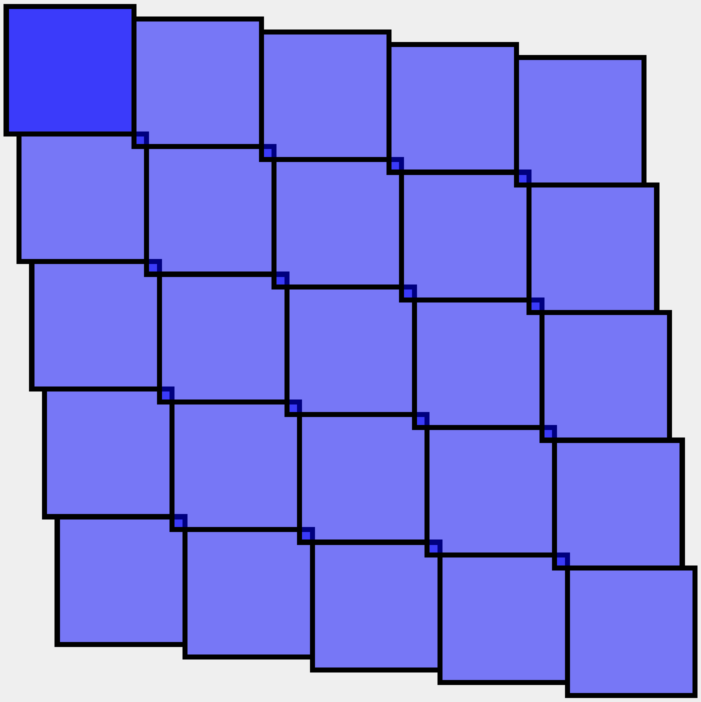
☑ Use saved size and position of the tile

Nothing selected.

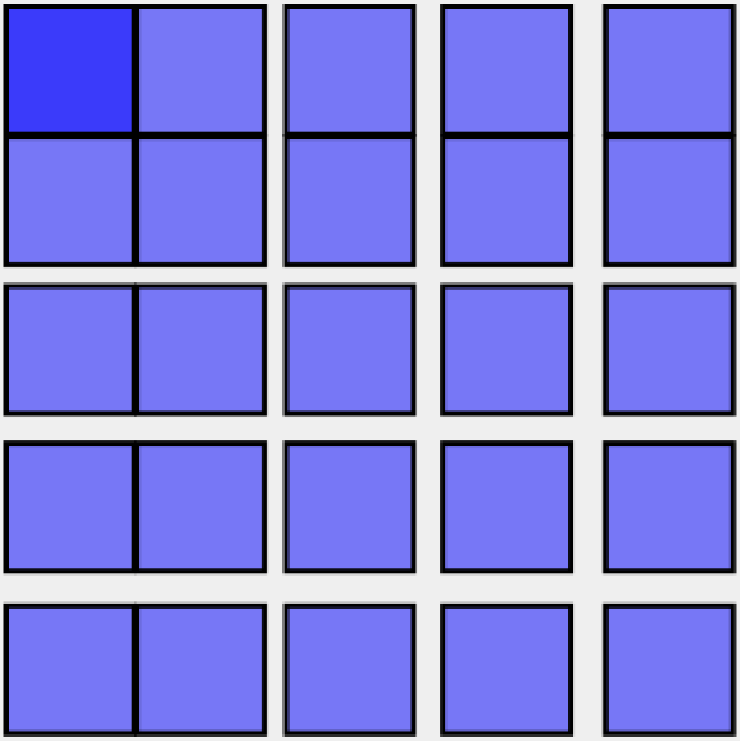
The *Shift* tab of the *Tile Clones* dialog.



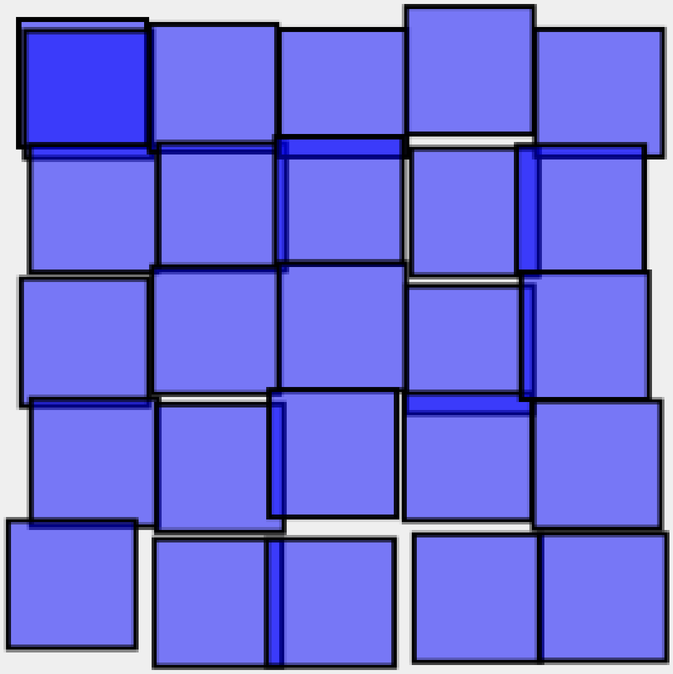
A P1 symmetry tiling with a constant shift of 10% (of the *bounding box*). There is an *x* shift for each column and a *y* shift for each row.



A P1 symmetry tiling with a constant shift of 10% (of the *bounding box*). There is a *y* shift for each column and an *x* shift for each row.

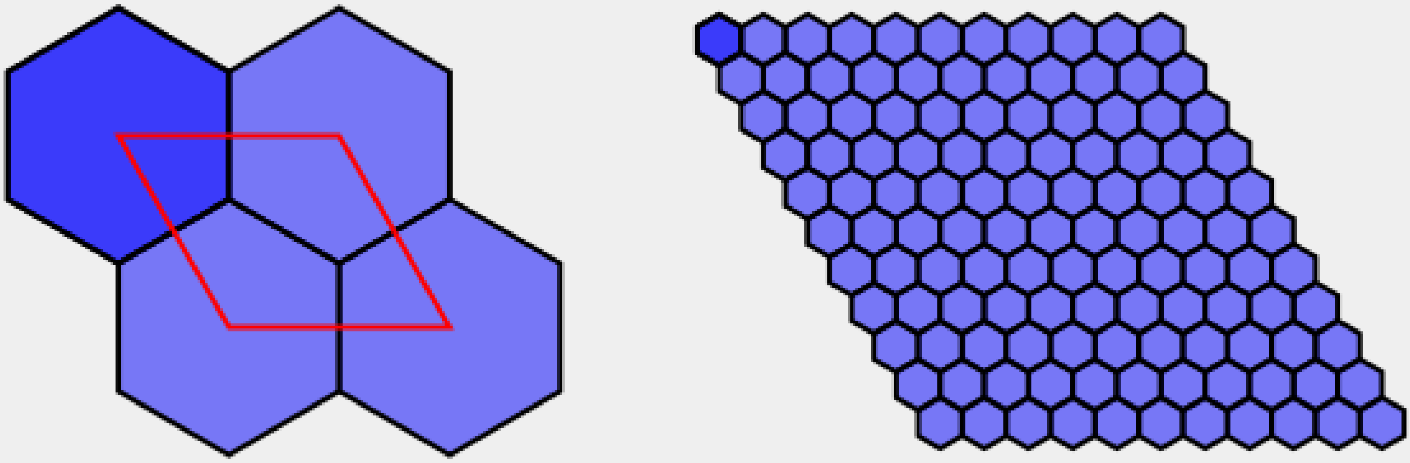


A P1 symmetry tiling with an exponential shift of 1.1 (2% shift in *x* and *y*).



A P1 symmetry tiling with a random shift of 10% (of the *bounding box*) in both *x* and *y*.

Question: What is the symmetry of closely packed hexagons? The answer is P1 as can be seen below. One can use this fact to trivially generate the board for the game Hex invented independently by the mathematicians Piet Hein and John Nash.



Closely packed hexagons have a P1 symmetry tiling as shown on the left. On the right is the board for the game Hex. To generate both tilings, a hexagon was tiled using a shift in *x* of 50% and a shift in *y* of -25% per row.

Symmetry Tab

Table of Contents

Scale Tab

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Tiling » Scale Tab

⬅ Index ➡

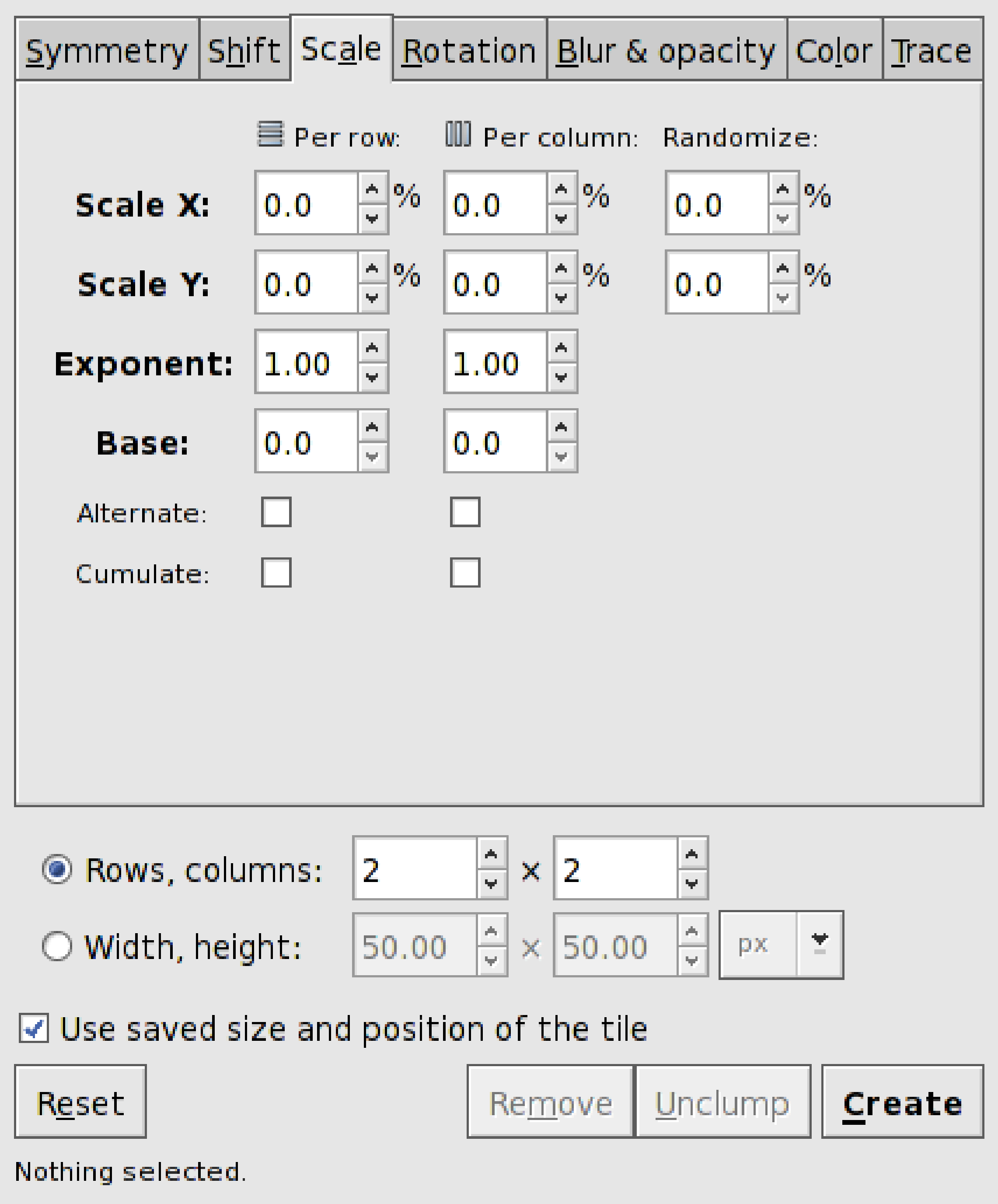
Scale Tab

The *Scale* tab allows one to increase or decrease the size of the tiles depending on the row and column position. The following options are available to scale tiles:

- *Scale X*, *Scale Y*: Scales each tile in terms of percentage. A random factor can be added.
- *Exponent*: Scale each tile with an exponential factor. The nominal scaling S becomes $S^{exponent}$.
- *New in v0.46:*

  Base: Used to create a logarithmic spiral along with *Rotation*. The nominal scaling S becomes $base^{S-1}$ unless *base* is one in which case scaling remains unchanged. Use a value less than one for a converging spiral and a value greater than one for a diverging spiral. A true logarithmic spiral would use a base of *e*=2.718 (or 1/*e*=0.368). See the *Tile Tricks* section for examples.
- *Alternate*: Alternate scaling up and scaling down tiles.
- *New in v0.46:*

  Cumulate: Scaling is cumulative.



	Per row:	Per column:	Randomize:
Scale X:	0.0 %	0.0 %	0.0 %
Scale Y:	0.0 %	0.0 %	0.0 %
Exponent:	1.00	1.00	
Base:	0.0	0.0	
Alternate:	□	□	
Cumulate:	□	□	

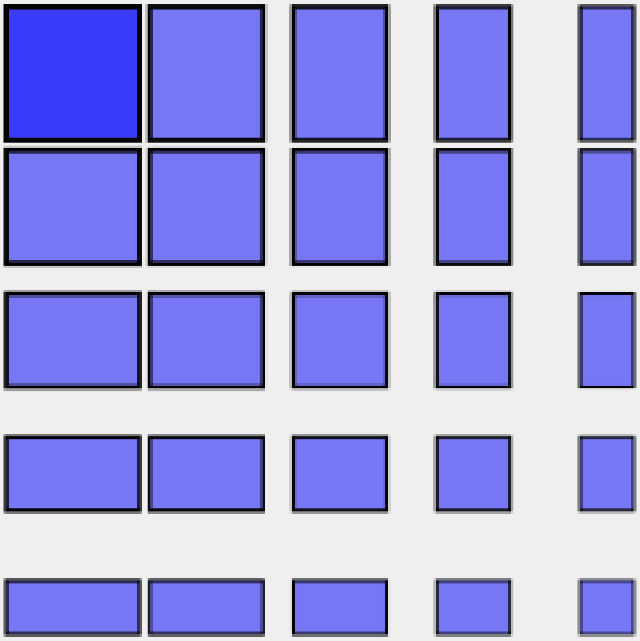
● Rows, columns: 2 × 2

○ Width, height: 50.00 × 50.00 px

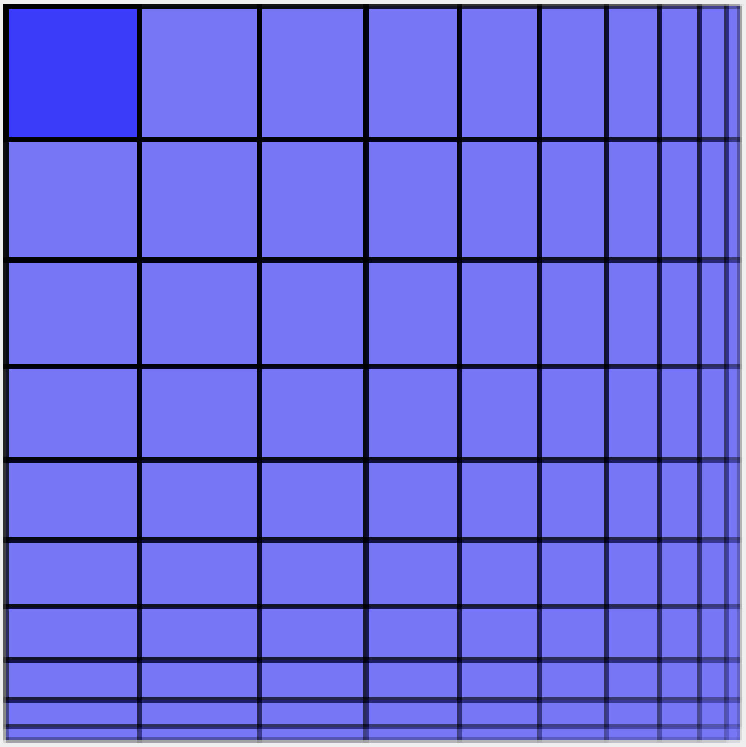
☑ Use saved size and position of the tile

Nothing selected.

The *Scale* tab of the *Tile Clones* dialog.



A P1 symmetry tiling with a negative scaling. There is an -15% *x* scaling for each column and a -15% *y* scaling for each row. The scaling is a percentage of the base tile *bounding box*. The spacing between the center of adjacent tiles remains fixed.



A P1 symmetry tiling with a cumulative negative scaling. There is a -10% *x* scaling for each column and a -10% *y* scaling for each row. There is also a -5% *x* shift for each column and a -5% *y* shift for each row. The *Cumulate* box is checked for both *x* and *y*. A general rule is that to keep scaled tiles just touching, specify a cumulative shift that is half of the scaling (in percent).

← Shift Tab

↑ Table of Contents

→ Rotation Tab

Inkscape » Tiling » Rotation Tab

Index

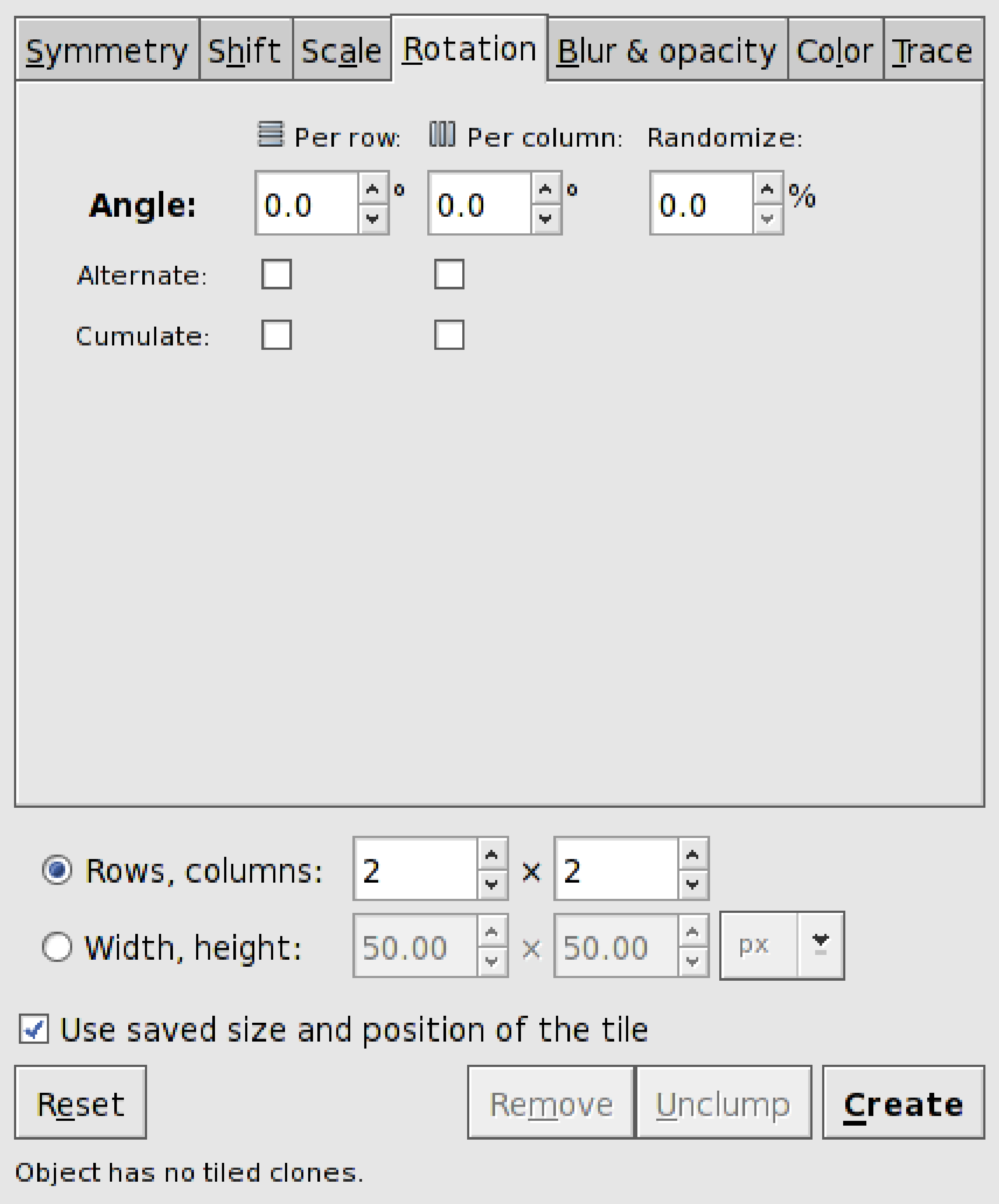
Rotation Tab

Updated for v0.46.

The *Rotation* tab allows one to rotate the tiles depending on the row and column position. As of v0.46, the *Rotation center* is used as the center of rotation. See the *Tile Tricks* section for examples of using a shifted *Rotation center*. The rotation is specified in degrees. The following options are available:

- *Angle*: Rotate by this amount around the *Rotation center*. A random factor can also be added.
- *Alternate*: The rotation alternates between being added and subtracted.
- *New in v0.46:*

  Cumulate: Rotation is cumulative.



Per row: Per column: Randomize:

Angle: 0.0 ° 0.0 ° 0.0 %

Alternate: □ □

Cumulate: □ □

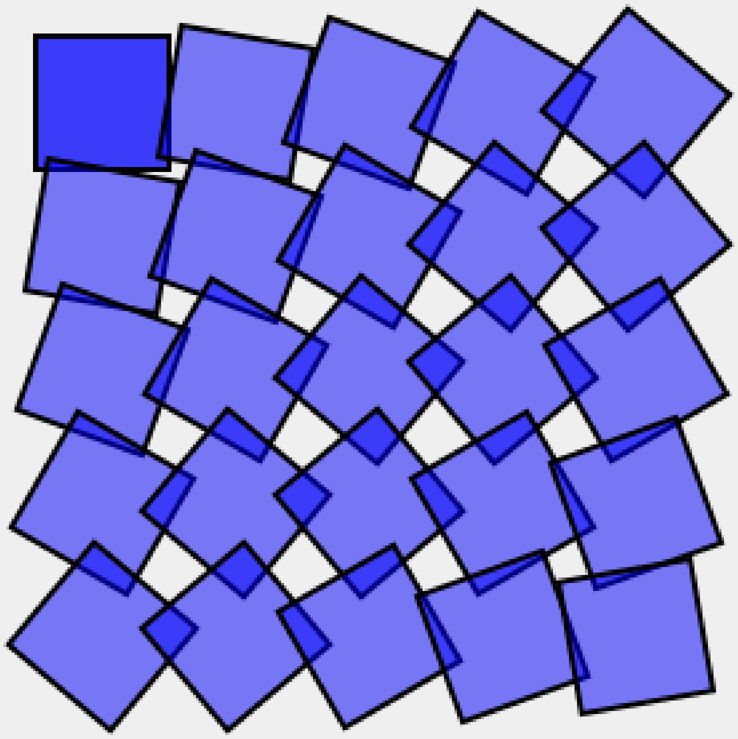
● Rows, columns: 2 × 2

○ Width, height: 50.00 × 50.00 px

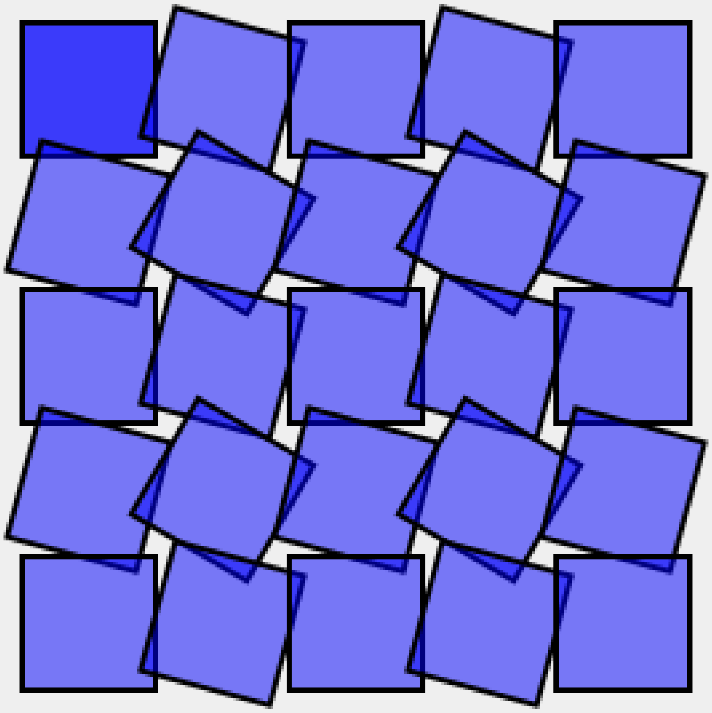
☑ Use saved size and position of the tile

Object has no tiled clones.

The *Rotation* tab of the *Tile Clones* dialog.



A P1 symmetry tiling with a 10° rotation for each row and column.



A P1 symmetry tiling with a 15° alternating rotation for each row and column.

Scale Tab

Table of Contents

Blur and Opacity Tab

Inkscape: Guide to a Vector Drawing Program

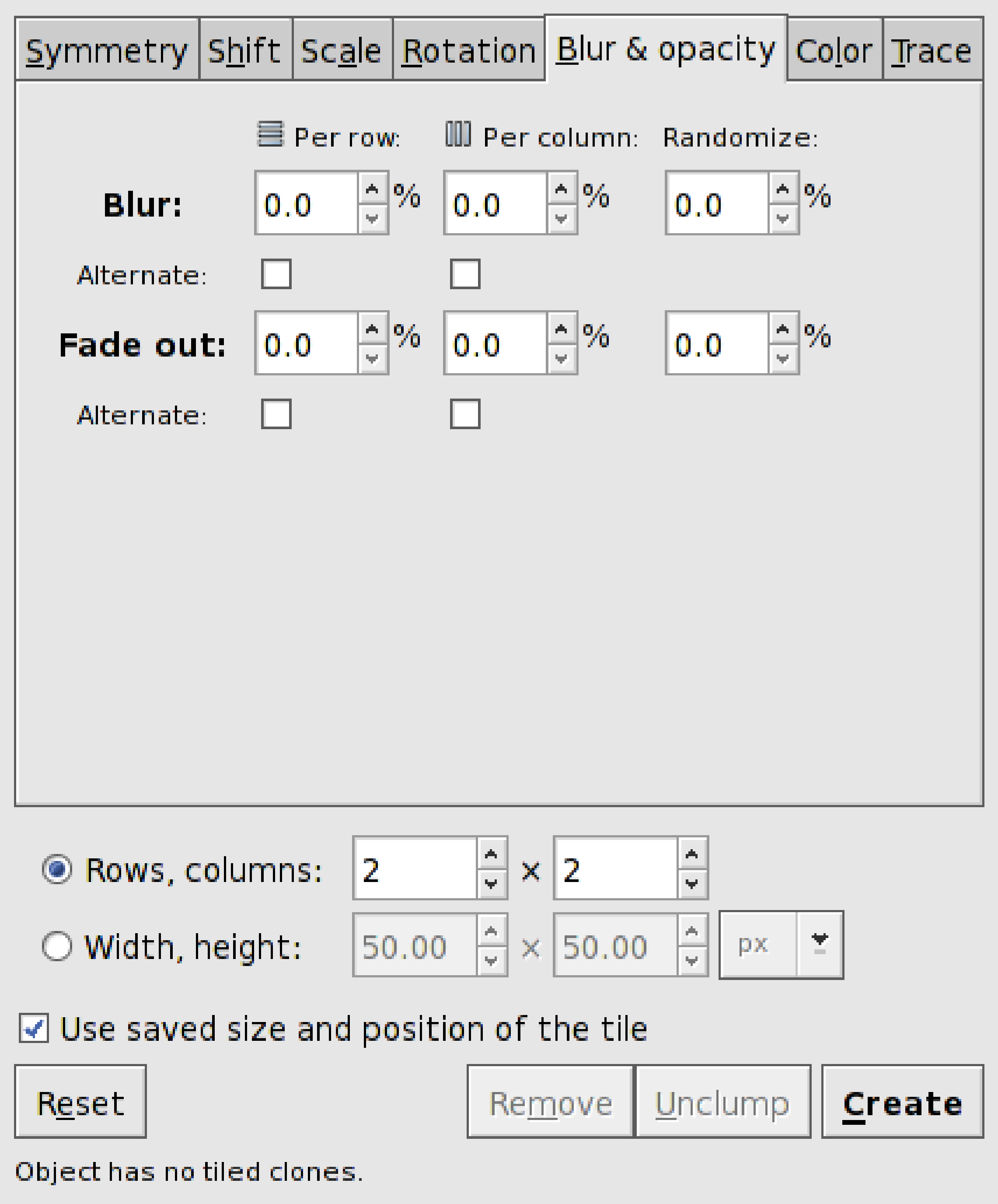
Get the book.

Inkscape » Tiling » Blur and Opacity Tab

← Index →

Blur and Opacity Tab

The *Blur and opacity* tab allows one to change the *blur* and/or *transparency* of each tile depending on the row and column position.



	Per row:	Per column:	Randomize:
Blur:	0.0 %	0.0 %	0.0 %
Alternate:	□	□	
Fade out:	0.0 %	0.0 %	0.0 %
Alternate:	□	□	

● Rows, columns: 2 × 2

○ Width, height: 50.00 × 50.00 px

☑ Use saved size and position of the tile

Object has no tiled clones.

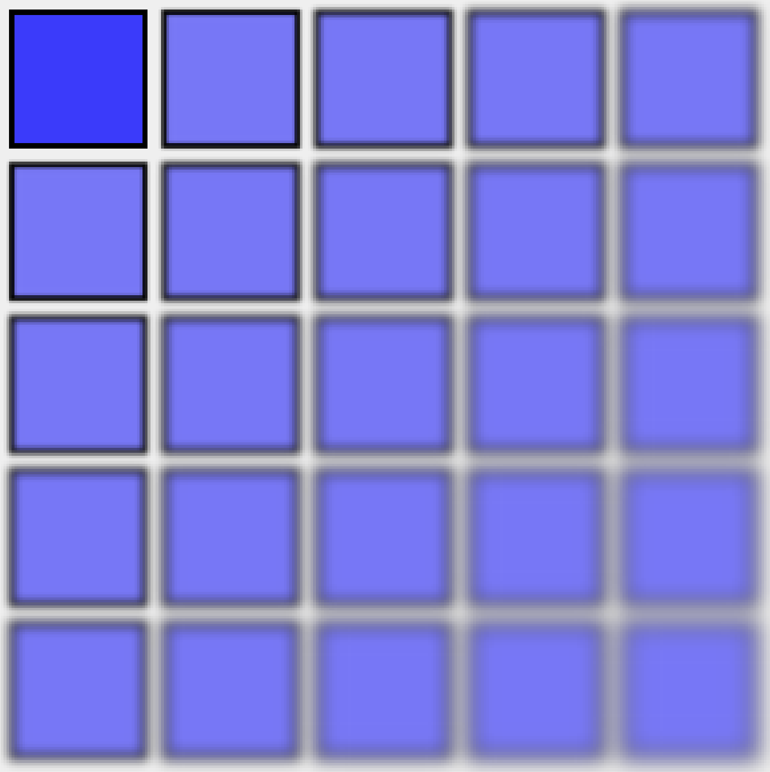
The *Blur and opacity* tab of the *Tile Clones* dialog.

Blur

New in v0.45.

A *Gaussian Blur* filter can be applied to each clone with different blurring values.

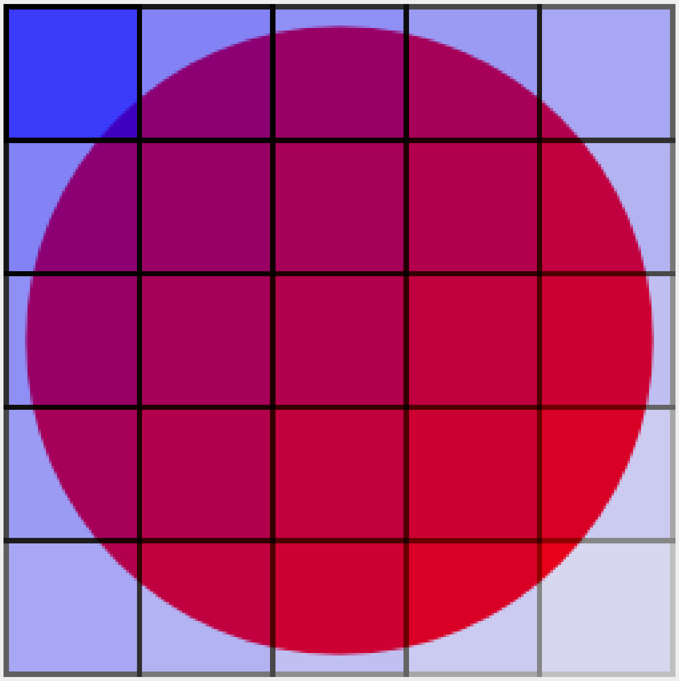
The blur change is specified in percent. The change in blur can be specified to *Alternate* between a positive and negative value; however, a negative blur value can be entered in the *Per row* and *Per column* boxes. A *Randomizer* factor can also be specified.



A P1 symmetry tiling with a 2% increase in blur for each row and column.

Opacity

The opacity change is specified in percent. The change in opacity can be specified to *Alternate* between a positive and negative value. A *Randomizer* factor can also be specified.



A P1 symmetry tiling with a 10% decrease in opacity for each row and column. A red circle has been placed under the tiling to illustrate the changes in opacity.

Rotation Tab

Table of Contents

Color Tab

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Tiling » Color Tab

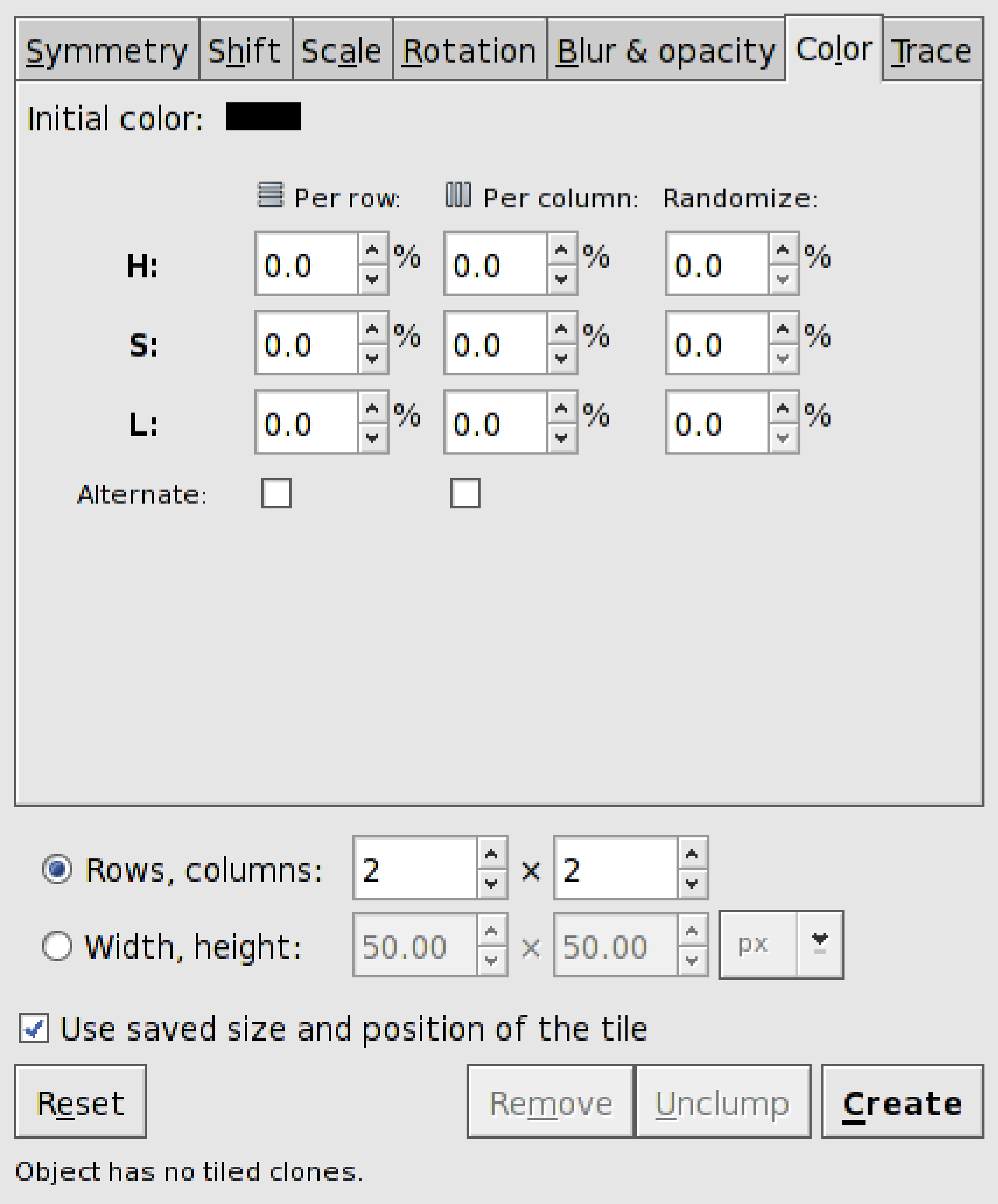
Index

Color Tab

The *Color* tab allows one to change the *Color* of each tile depending on the row and column position. The color change is specified in percent for each of the three components of a color specified with the *HSL* standard (see the section called “HSL”). The *Hue* repeats itself after a change of 100%. The full scale for *Saturation* and *Lightness* components are each 100%. The changes in the three parameters can be specified to *Alternate* between a positive and negative change. A *Randomizer* factor can also be specified.

Two key points: First, the *Fill* and/or *Stroke paint* must be specified as *Unset* (?) (see the section called “Fill and Stroke Paint”). Second, an *Initial color* must be specified by using the *Initial color of tiled clones* dialog accessible by clicking on the color button next to the *Initial Color* label.

Note that it is meaningless to have only a shift in *Hue* with a starting color of black or white. This is like trying to walk east from the north pole.



Initial color: ■

	Per row:	Per column:	Randomize:
H:	0.0 %	0.0 %	0.0 %
S:	0.0 %	0.0 %	0.0 %
L:	0.0 %	0.0 %	0.0 %
Alternate:	□	□	

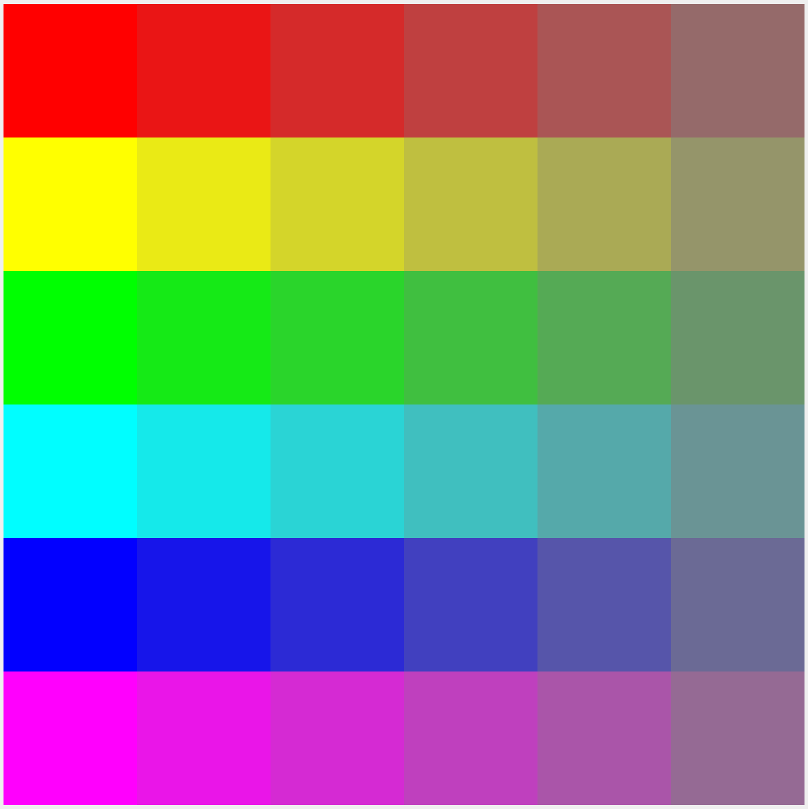
● Rows, columns: 2 × 2

○ Width, height: 50.00 × 50.00 px

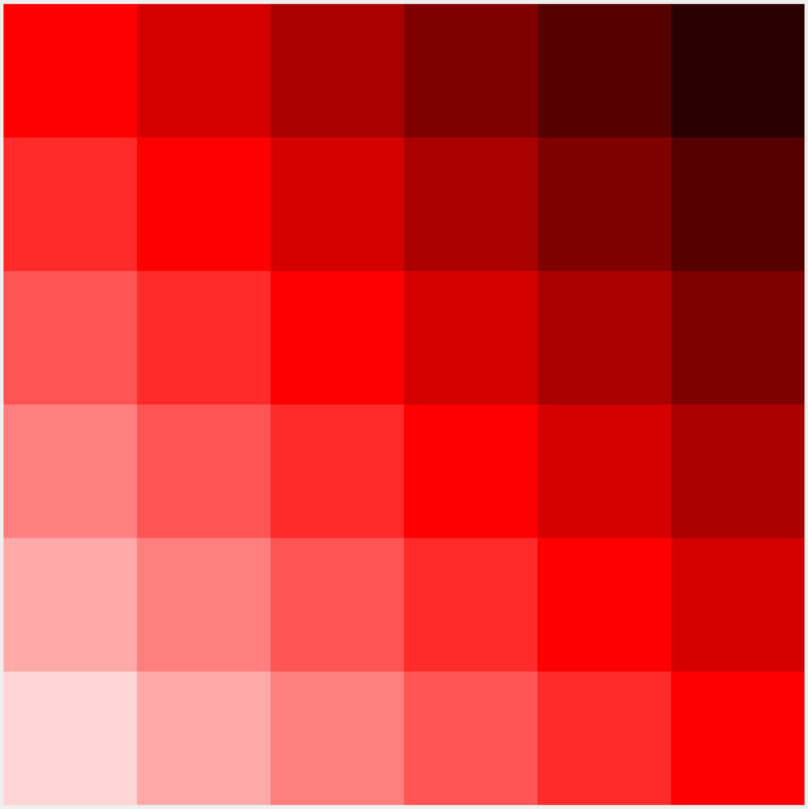
☑ Use saved size and position of the tile

Object has no tiled clones.

The *Color* tab of the *Tile Clones* dialog.



A P1 symmetry tiling with a 16.7% change in *Hue* per row and a -16.7% change in *Saturation* per column. The starting color is a red with 100% *Saturation* and 50% *Lightness*.



A P1 symmetry tiling with an 8.3% change in *Lightness* per row and a -8.3% change in *Lightness* per column. The starting color is a red with 100% *Saturation* and 50% *Lightness*.

Blur and Opacity Tab

Table of Contents

Trace

Inkscape » Tiling » Trace

Index

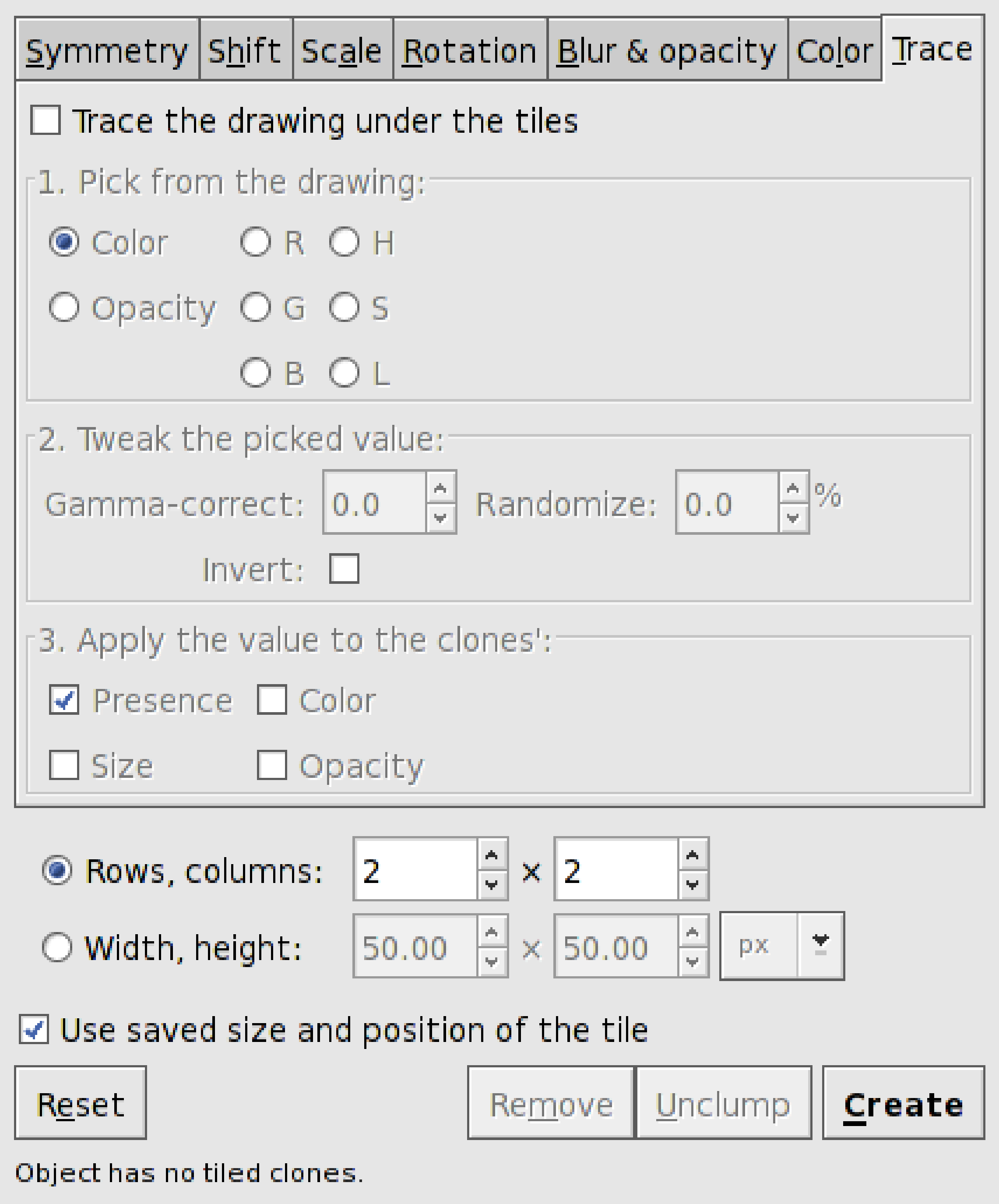
Trace

The *Trace* tab allows one to set the color, size, and transparency of the tiles by the color or transparency of the objects (including bitmaps) that are placed under the location of the tiling. To enable this feature, the *Trace the drawing under the tiles* box must be checked.

The *Trace* tab has three sections. At the top is a section for specifying what property of the underlying drawing should be used for input. Options include the color, one of the *RGB* components, or one of the *HSL* components. There is also the option to use the *Opacity*, which is the sum of the opacities (*Alpha*) of all objects under the tile.

In the middle of the tab is a section to modify the input value. One can specify a *Gamma*[15] correction or add a randomization factor to the input. One can also invert the input.

The bottom section is for specifying what should be affected by the input. Options include *Presence* (the probability that a given tile will be drawn), color, size, and opacity. The color will only be changed for regions of the base tile that have *Unset* fill.



□ Trace the drawing under the tiles

1. Pick from the drawing:

● Color ○ R ○ H

○ Opacity ○ G ○ S

○ B ○ L

2. Tweak the picked value:

Gamma-correct: 0.0 Randomize: 0.0 %

Invert: □

3. Apply the value to the clones':

☑ Presence □ Color

□ Size □ Opacity

● Rows, columns: 2 × 2

○ Width, height: 50.00 × 50.00 px

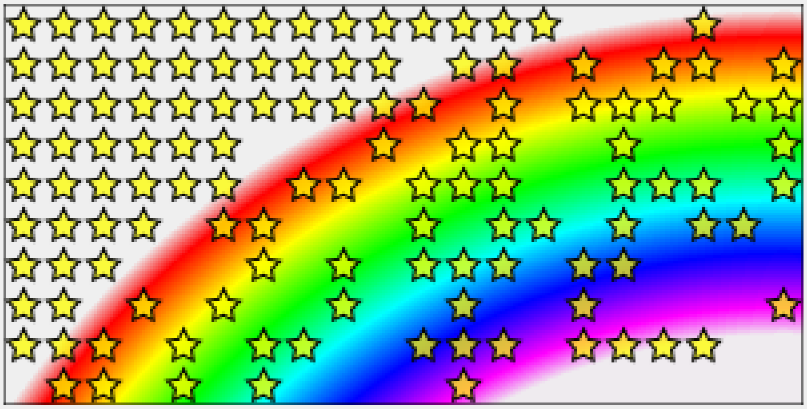
☑ Use saved size and position of the tile

Object has no tiled clones.

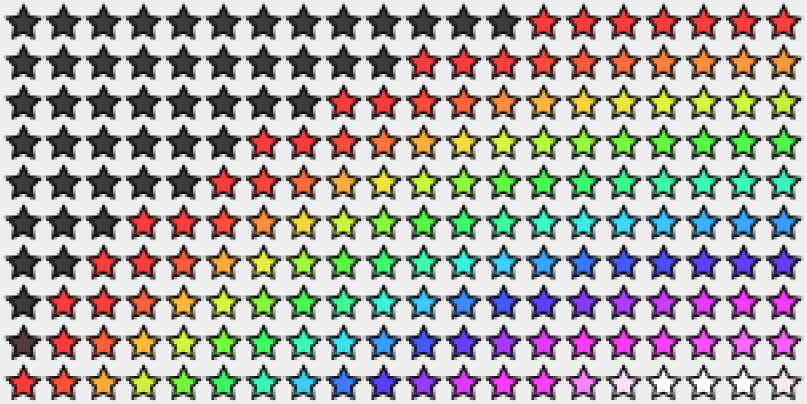
The *Trace* tab of the *Tile Clones* dialog.

The following figures show the effect of some of the possible combinations of input and output options. All the figures use the first rainbow figure as the input drawing. The rainbow is a *radial gradient* with multiple stops. The inside of the rainbow is defined as a white gradient stop with zero *Alpha*. The last outside stop is defined with a red color and with zero *Alpha*. For most figures, a star inside an unfilled rectangle is used as the base tile. The star has been given an *Unset* fill when color is selected in the output.

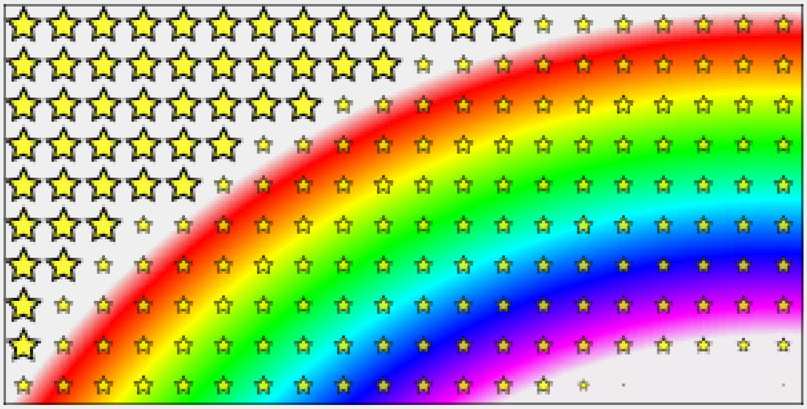
The rainbow pattern used for the background (a radial gradient).



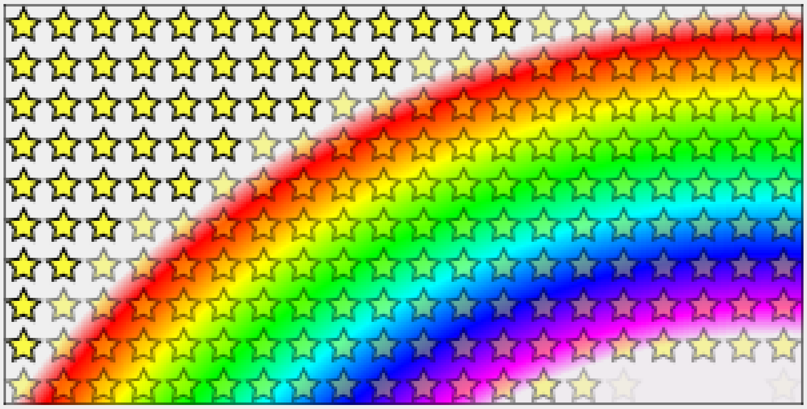
Input: Color. Output: Presence.



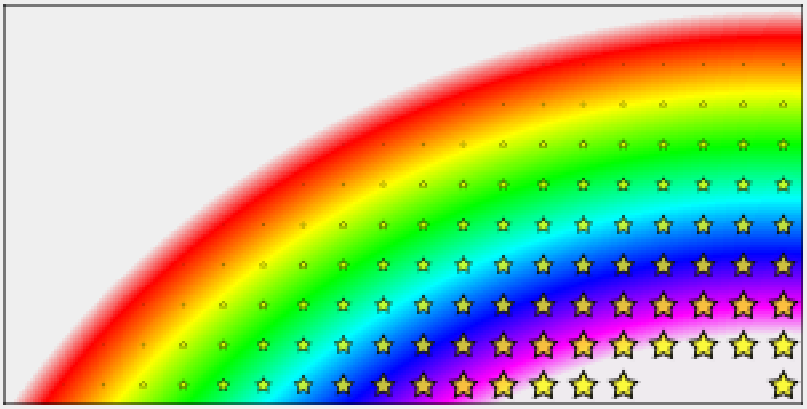
Input: Color. Output: Color. Background rainbow has been removed.



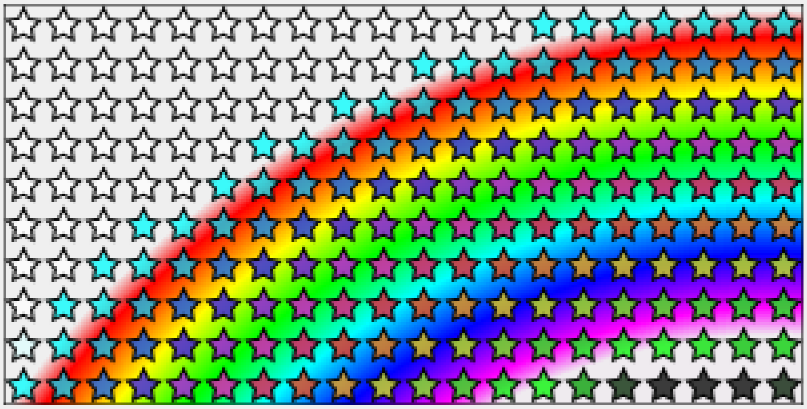
Input: Color. Output: Size.



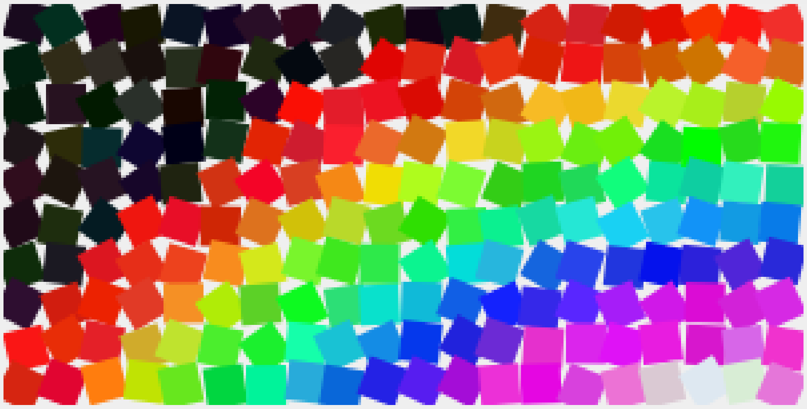
Input: Color. Output: Opacity.



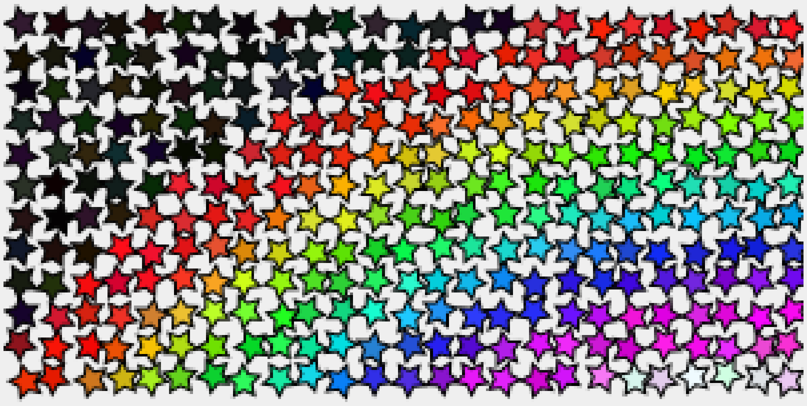
Input: Hue. Output: Size. Note how the red has a hue of zero and purple has the maximum value.



Input: Hue, inverted. Output: Color.



Input: Color, 10% random gamma. Output Color. Changes made to other tabs: Shift: random 10%, Rotation: random 20%. A square base tile with *Unset* fill has been used. The background rainbow has been deleted.



Input: Color, 10% random gamma. Output Color. Changes made to other tabs: Shift: -20%, random 10%, Rotation: random 20%. The number of rows and columns has been increased to compensate for the shift. The background rainbow has been deleted.

[15] See appendix for definition of *Gamma*.

←
Color Tab

↑
Table of Contents

→
Tricks

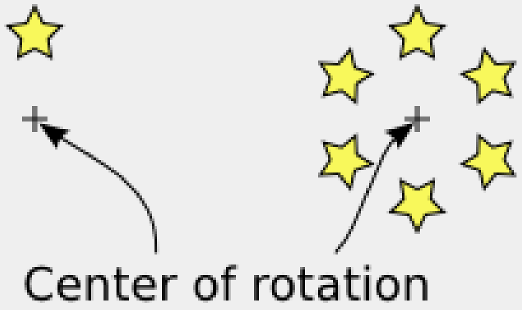
Get the book.

Tricks

Updated for v0.46.

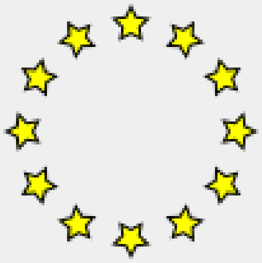
It is possible to exploit the *Tiling* dialog to produce a number of useful effects. The most interesting is placing tiles along an arc or spiral.

To put a tile along an arc use the P1 symmetry with one row of tiles. One use to have to resort to using a Group to put tiles along a curve. As of v0.46, the *Rotation center* is used as the center of rotation. One also used to have to specify a shift of -100%. Now one can just check the *Exclude tile* box.



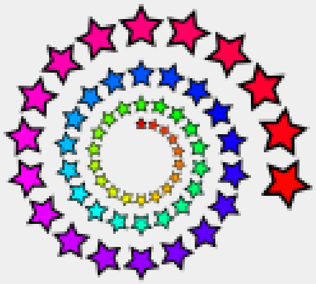
The base tile is drawn on the left, showing the *Rotation center* of the tile. On the right is after a P1 tiling with a per column shift removed by checking the *Exclude tile* box and with a rotation of 60%.

The next figure shows how 12 stars can be put in a circle. This would have been an alternative way of placing the stars in the European Union flag if the stars did not need to be placed with one of their points straight up.

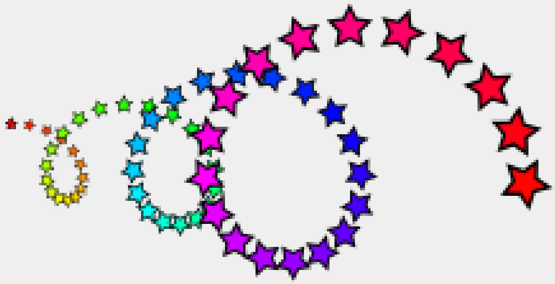


Twelve stars in a circle.

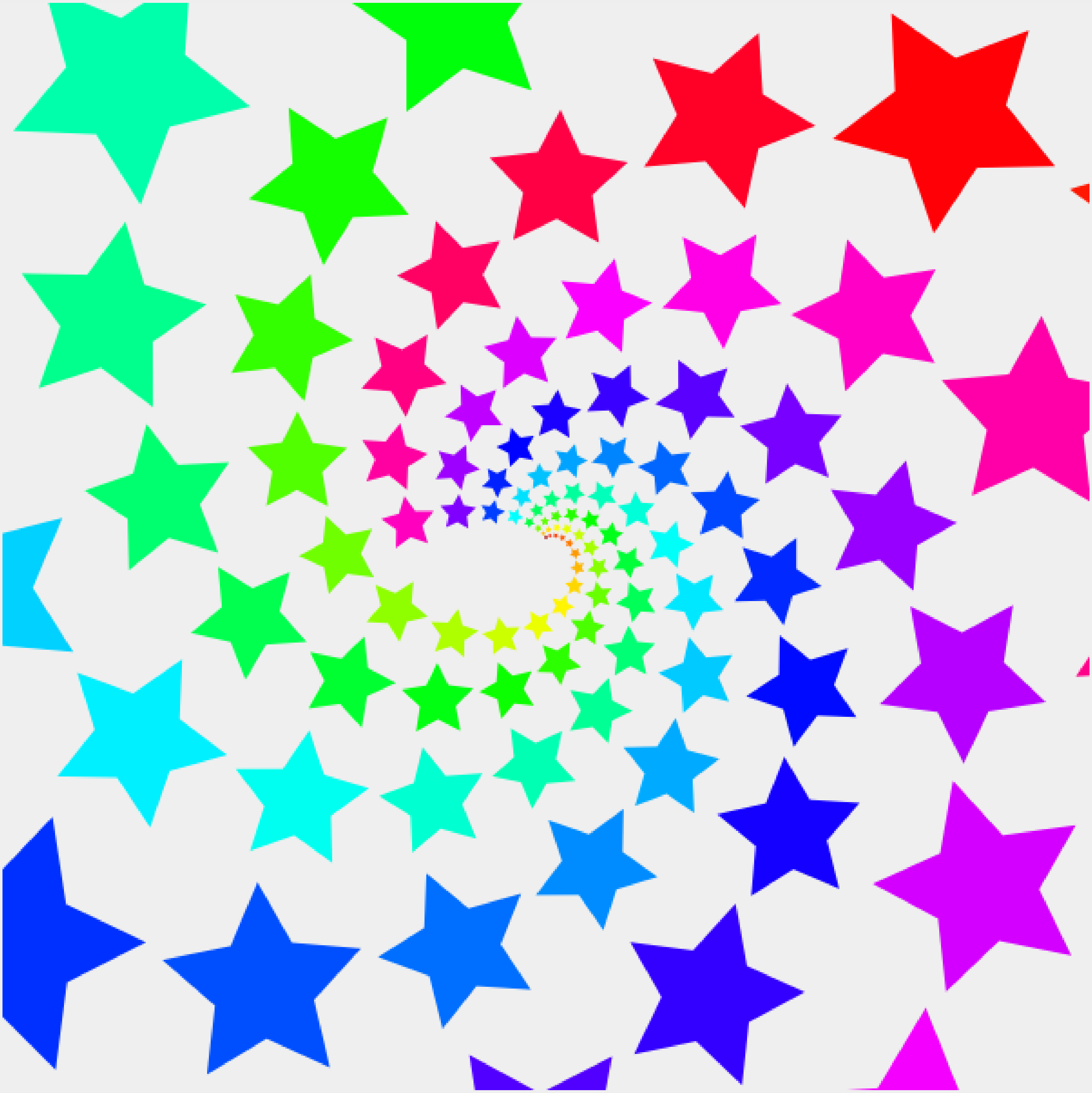
This trick can also place objects along a spiral by specifying that the tile should get larger with each column. As of v0.46, one can put the stars on a logarithmic spiral so that the stars don't run into each after several loops.



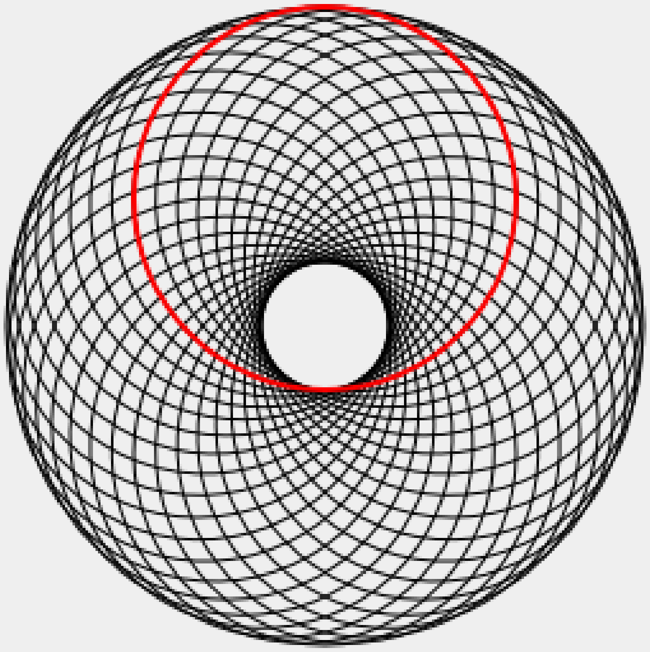
Stars on a logarithmic spiral. The tile size is increased by 2.5% with *Base* set to 2.7. Each tile is rotated 20°.



Stars on a logarithmic spiral. The tile size is increased by 2.5% with *Base* set to 2.7. Each tile is rotated 20°. The per column shift has been set to 60% (with the *Exclude tile* box checked).



A “P1 symmetry” tiling. 8 rows, 21 columns. Rotation of -11.5° per row and 20.6° per column, *Scale* of 39.3% per row and 24.2% per column with a *Base* of 2.7 for both *x* and *y*. There pattern matches that for a pine cone with 8 rows in one direction and 13 in the other. For the mathematicians: note that 13 times the per column scaling is equal to 8 times the per row scaling and that 13 times the per column rotation minus 8 times the per row rotation is equal to 360°. This is due to the constraint that the 14th star in the first row is the same as the 9th star in the first column.



A circle tiled on an arc. The red circle with the *Rotation center* moved off center was the source tile.

← ↑ →

Trace

Table of Contents

Chapter 16. Tracing Bitmaps

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Tracing Bitmaps » Single Scans

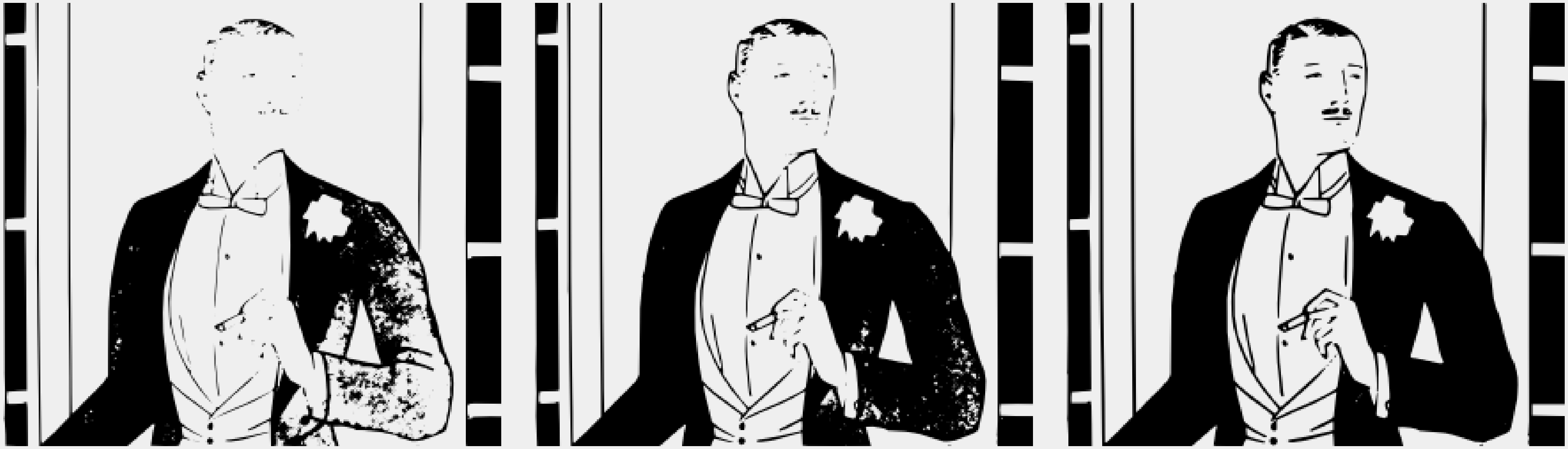
Index

Single Scans

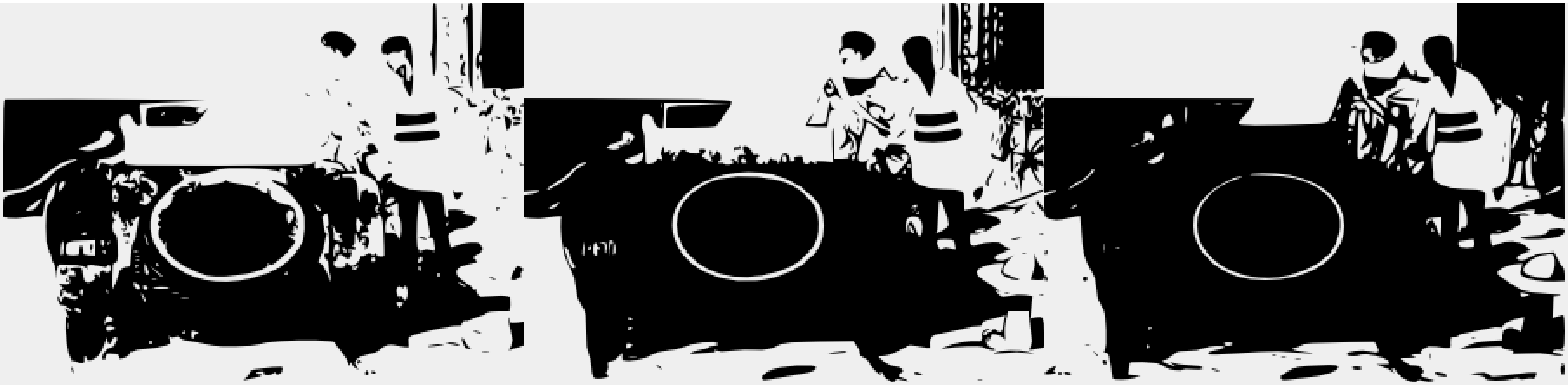
This section covers creating single *Paths* from bitmap images. One option common to all scanning strategies is available: Checking the *Invert image box* will invert the area the created *Path* encloses.

Brightness Cutoff

Output depends on the *brightness* of the pixels in the bitmap. Brightness is defined as the sum of the red, green, and blue values for a pixel (or the grayscale value for black and white drawings). One path is created containing all regions that are darker than the *Threshold* setting. This works well for black and white line art.



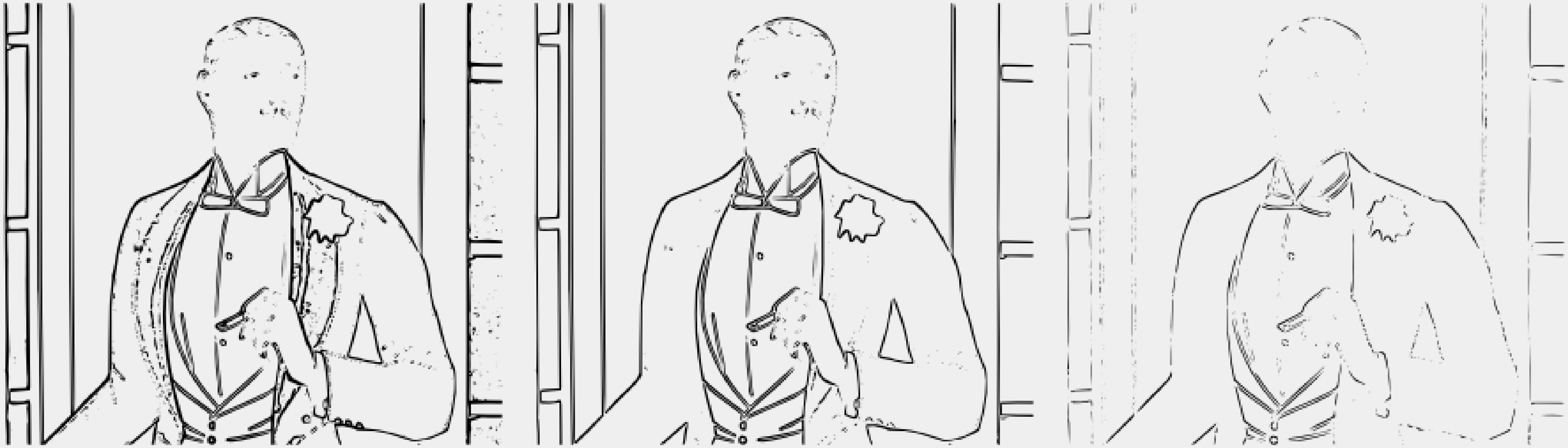
Brightness. From left to right, the *Threshold* is 0.4, 0.5, and 0.6.



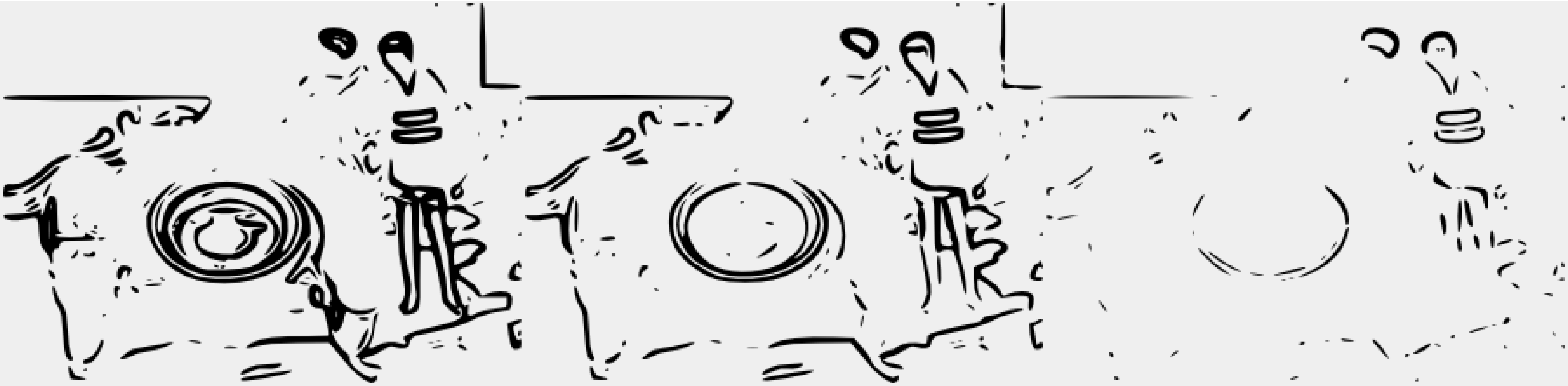
Brightness for a color drawing. From left to right, the *Threshold* is 0.4, 0.5, and 0.6.

Edge Detection

Output depends on differences in *brightness* between adjacent pixels. A path is created between areas with changes that exceed the *Threshold* setting.



Edge detection. From left to right, the *Threshold* is 0.1, 0.5, and 0.9.

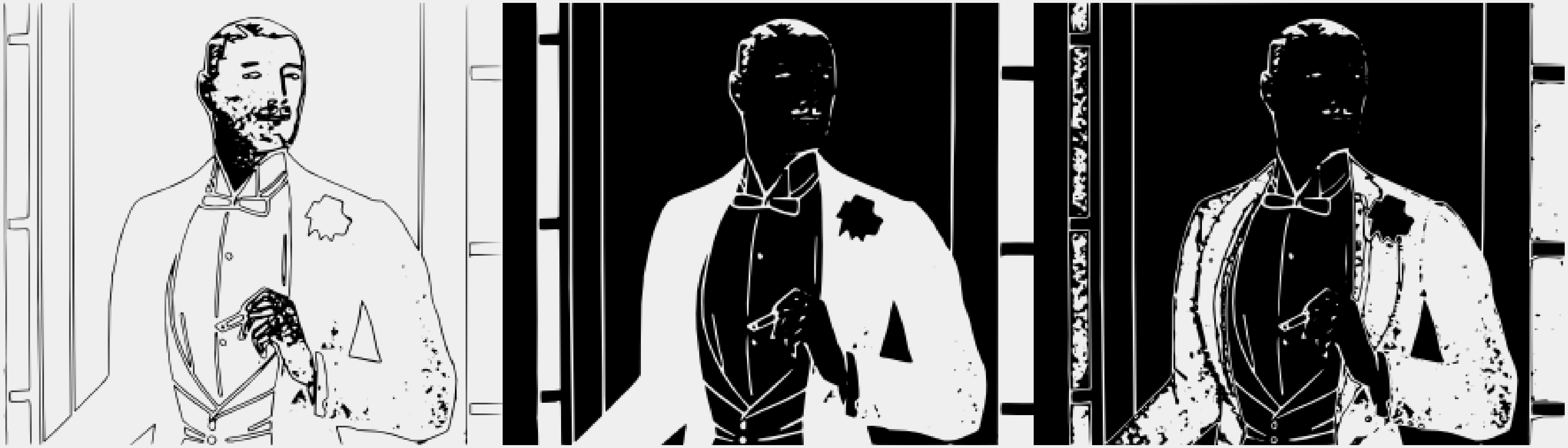


Edge detection for a color drawing. From left to right, the *Threshold* is 0.1, 0.5, and 0.9.

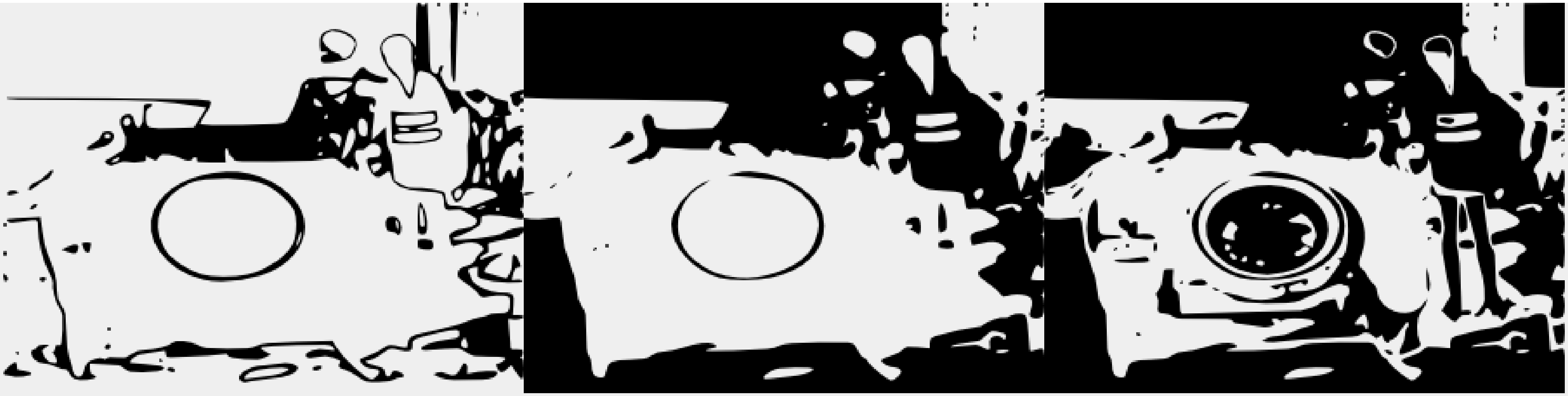
Color Quantization

New, faster algorithm in v0.45.

Output depends on changes in *color* between adjacent pixels. The *Number of Colors* gives the number of different colors that were used in looking for edges. Only one path is created; what is inside and outside of the path is based on if the index of the color is even or odd.



Color Quantization for 4, 6, and 8 colors (from right to left).



Color Quantization for 4, 6, and 8 colors (from right to left).

Chapter 16. Tracing Bitmaps

Table of Contents

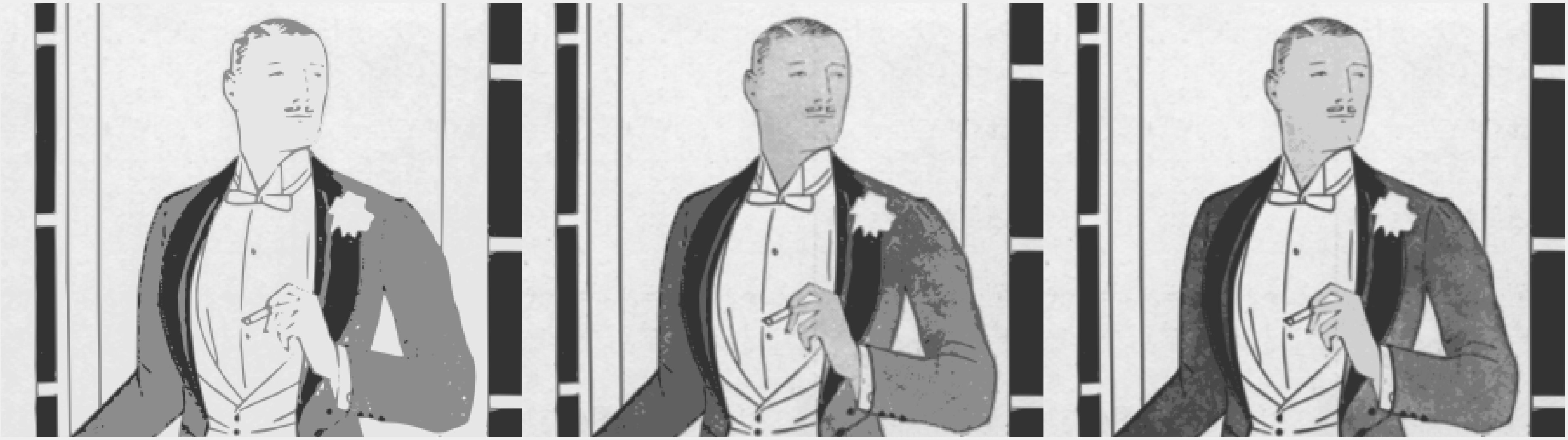
Multiple Scans

Multiple Scans

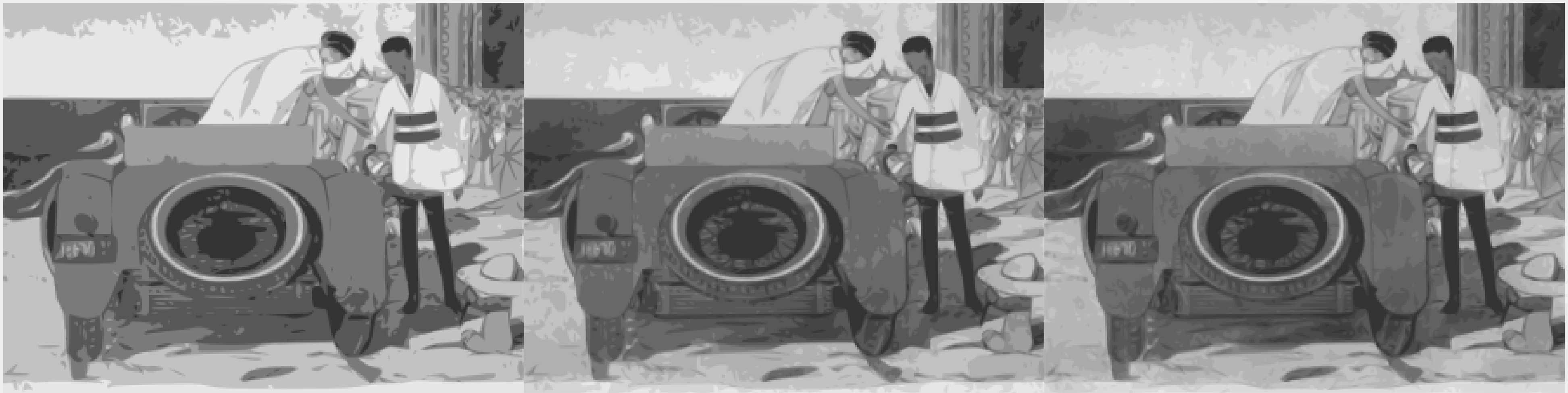
The bitmap is scanned multiple times, each time with a different setting. One path is created for each scan. The paths are stored in a group. The scanning criteria can be *Brightness*, *Colors*, or *Grays*. *Grays* uses the *Color* scanning criteria but assigns the resulting paths a grayscale rather than a color. The *Scans* entry box applies to all three scanning criteria. Three options, described in the Options section below, apply to all *Multiple Scans*.

Brightness Steps

The bitmap is scanned one more time than the specified number (Bug?). The values at which the scans are performed is nontrivial. The lowest (darkest) scan is always done at a brightness threshold of 0.2, the next scan is at a threshold of (0.2 + (0.9-0.2)/n). The output level of the darkest region is 0.2 and the lightest is 0.9. Other regions fall at evenly spaced positions in between. If the *Remove background* box is checked, the 90% region is removed.



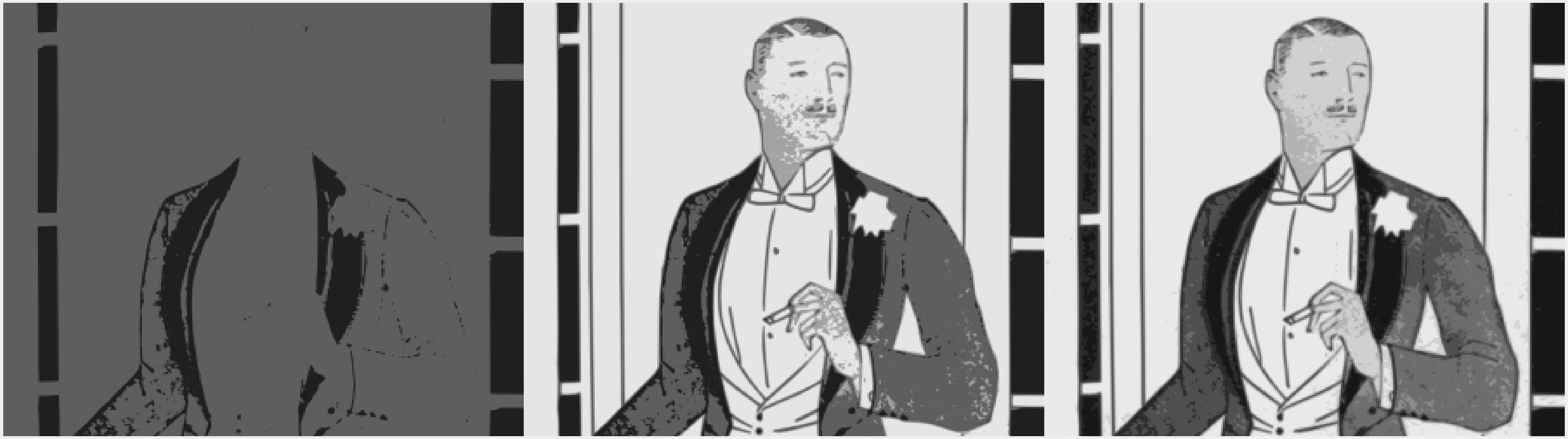
Multiple scans using the *Brightness* option. From left to right, there are 2, 4, and 8 scans.



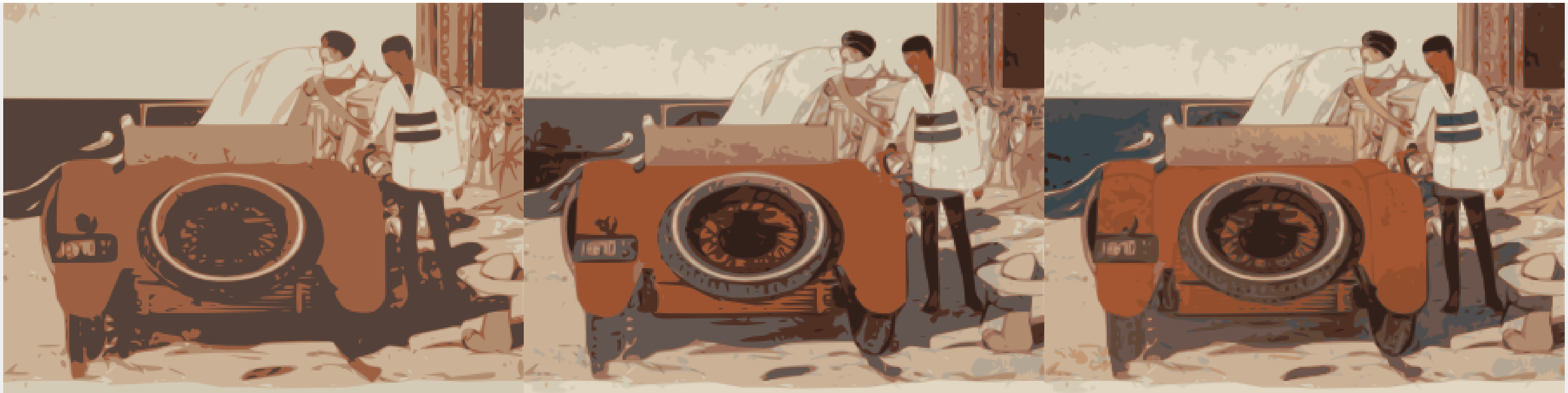
Multiple scans using the *Brightness* option. From left to right, there are 5, 10, and 15 scans.

Colors

The image is traced into the specified number of paths via the following procedure: The number of colors in the bitmap is reduced to the value in the *Scans* entry box using an optimal set of colors chosen via the *Octree Quantization* method. A black and white bitmap is created for each color, which is then sent to Potrace for tracing.



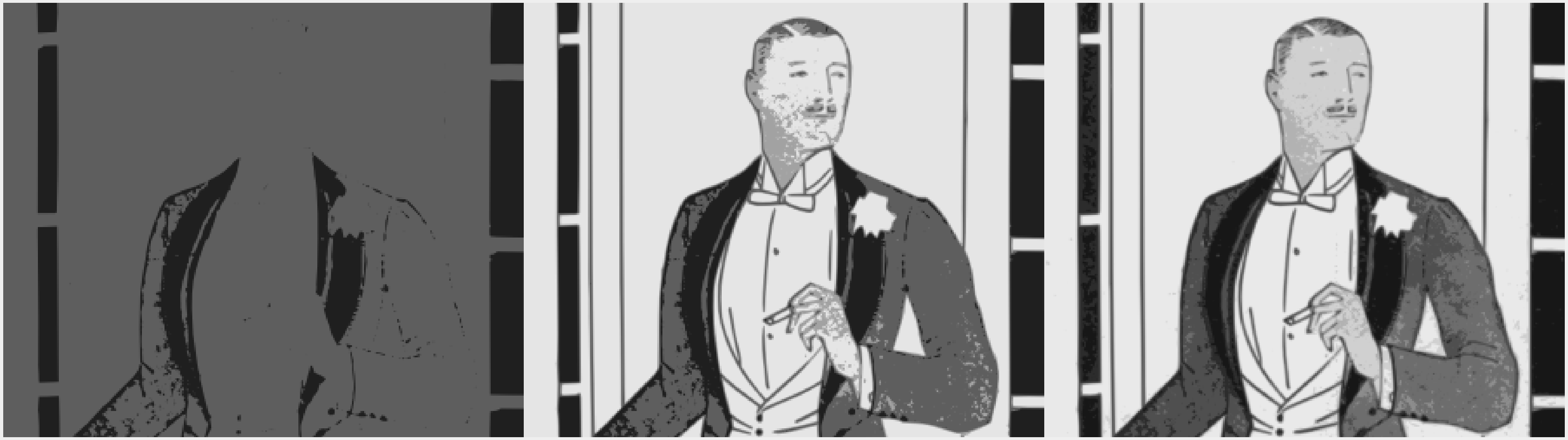
Multiple scans using the *Colors* option. From left to right, there are 2, 4, and 8 scans.



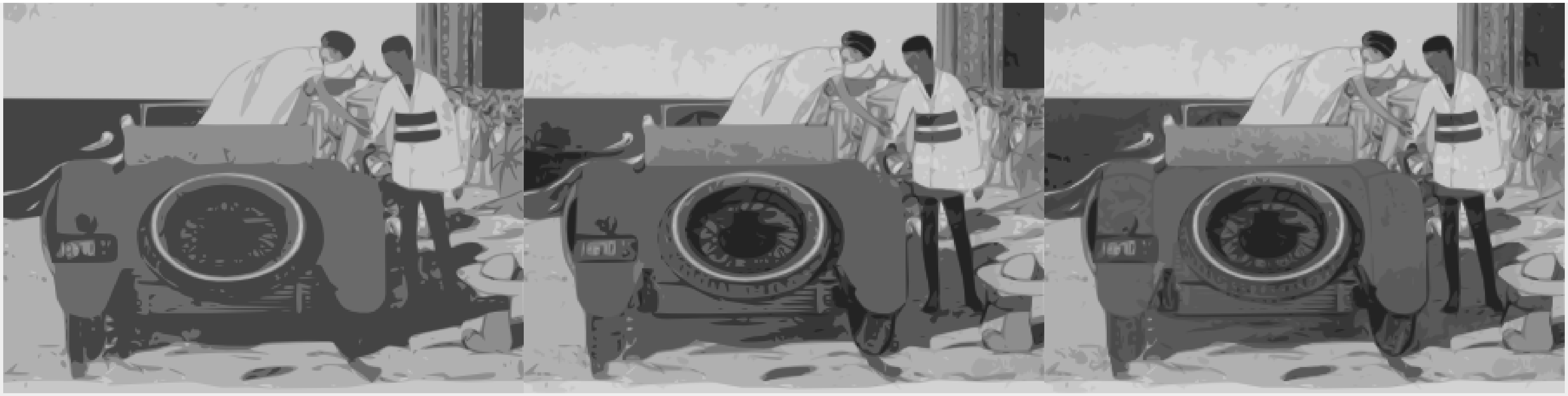
Multiple scans using the *Colors* option. From left to right, there are 5, 10, and 15 scans.

Grays

The tracing principle is the same as for *Colors* but the result is turned into a grayscale image.



Multiple scans using the *Grays* option. From left to right, there are 2, 4, and 8 scans.



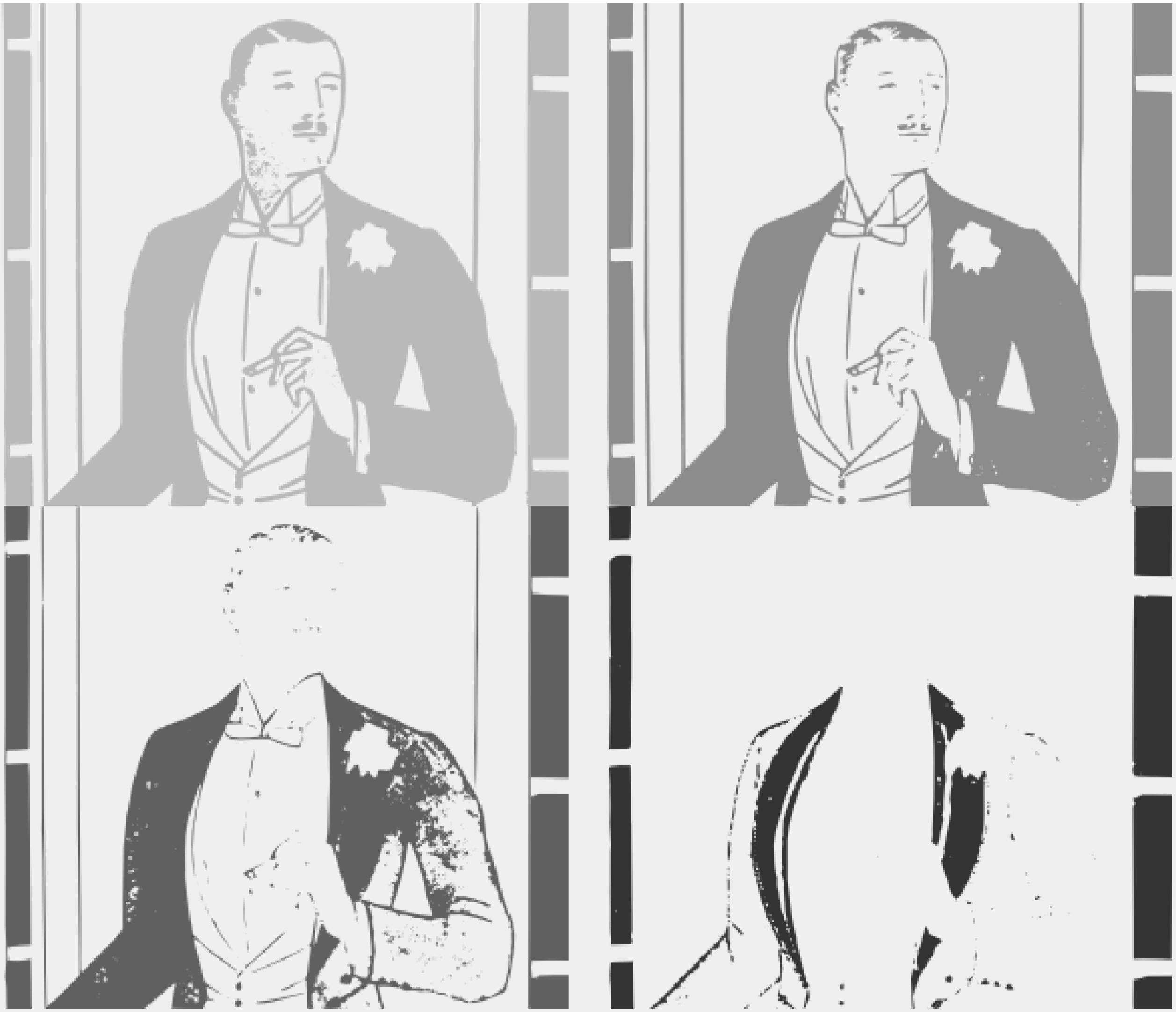
Multiple scans using the *Grays* option. From left to right, there are 5, 10, and 15 scans.

Options

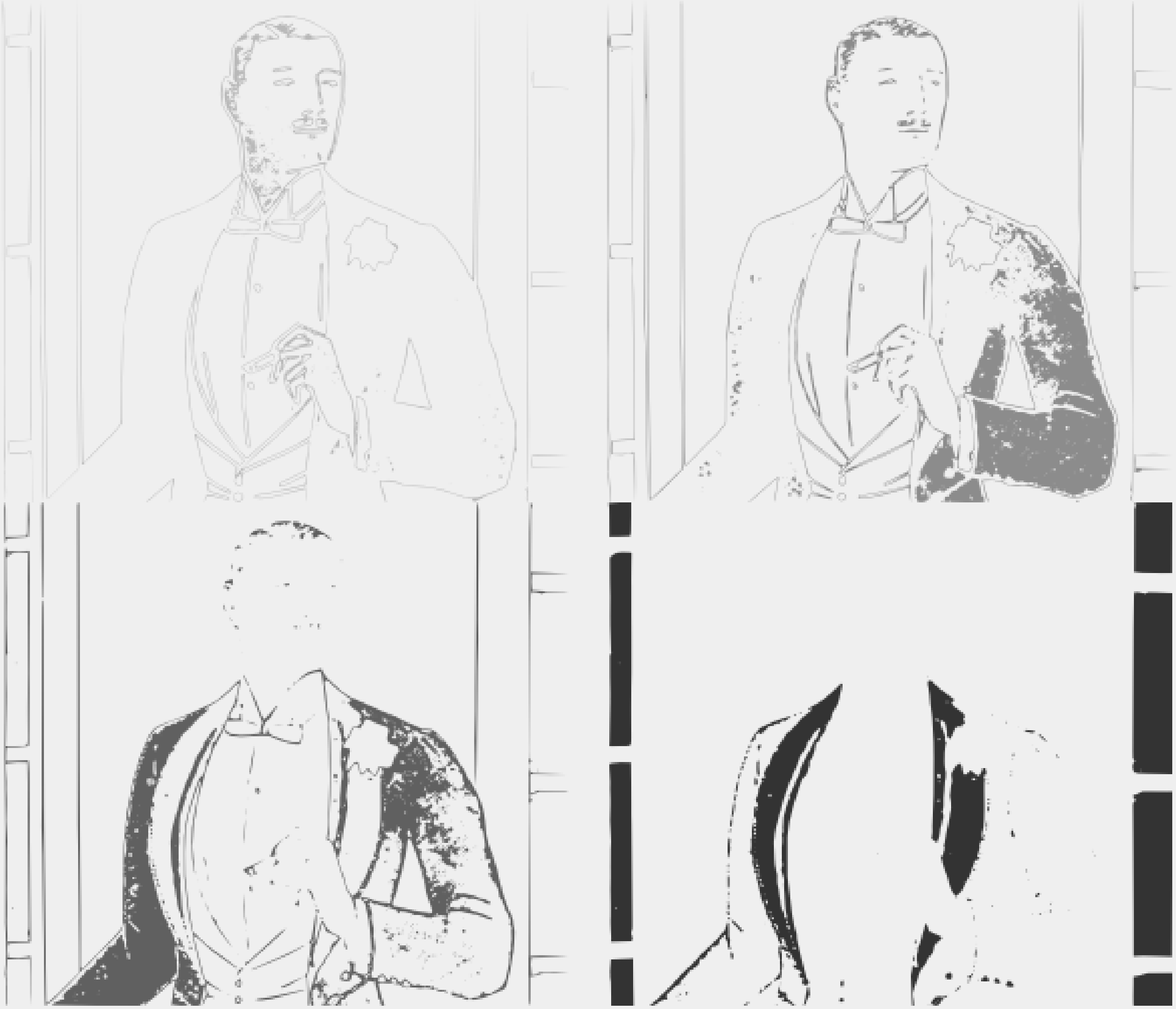
Three extra options are available with *Multiple scans*.

Smooth. Selecting the *Smooth* causes a *Gaussian Blur* to be applied to the **input** bitmap before tracing. This has the effect of smoothing out the difference between adjacent pixels and can be very useful, for example, with scans of screened prints. Too much filtering can lead to loss of detail. The preceding tracings were made without smoothing as the input scan had already been filtered with a *Gaussian* blur in Gimp.

Stack scans. The *Stack scans* option determines how the paths are defined. With the *Stack scans* box unchecked, the paths produced do not overlap; with the box checked, each path includes the area of the paths above it in *z-order*. The advantage of unstacked paths is that they are easier to divide into subpaths, while the advantage of stacked paths is that there are no “holes” between the coverage of the paths. The differences between the two situations is depicted below.



The four paths from a multiple scan with the *Stacked* box checked.



The four paths from a multiple scan with the *Stacked* box unchecked.

Remove background. The lowest path in *z-order* is defined as the *background*. Normally, this path has the lightest color. When the *Stack scans* option is chosen, the background path corresponds to a rectangle the size of the scanned image. Checking the *Remove background* box prior to scanning suppresses this path.

Single Scans

Table of Contents

Common Options

Get the book.

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Tracing Bitmaps » Common Options

Index

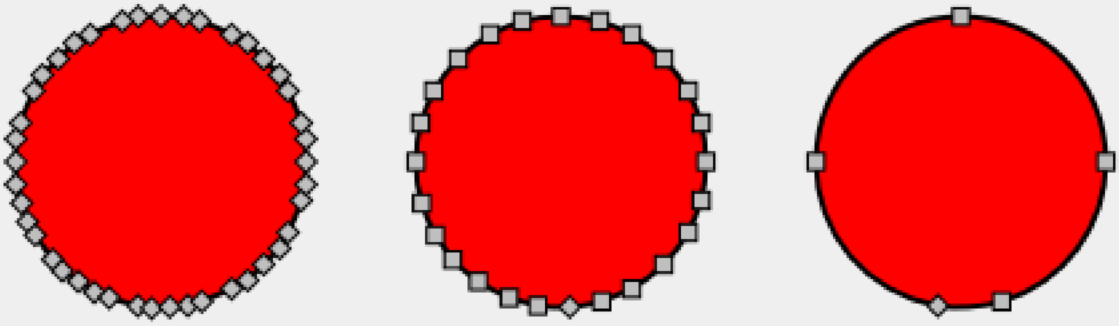
Common Options

This section covers options common to all scanning criteria that are found on the *Options* tab. Two of the options reduce file size while the third produces smoother paths. These are all options that are part of Potrace. Their usefulness will vary depending on the source bitmap.

Suppress speckles. Turning on this option removes all paths with a size less than the specified amount. See the *NP Logo* tutorial for an example of its use.

Smooth corners. This option produces rounded corners at nodes. The smaller the value, the sharper the corner. A value of 0 is equivalent to no smoothing (i.e., corner nodes connected by straight lines). As the value approaches 1, the number of nodes decreases with the percentage of smooth nodes increasing. The exact behavior is hard to predict.

Optimize paths. This option controls merging *Bezier* curves in a scan together, thereby reducing the number of nodes. The *Tolerance* value controls the allowed error in the resulting curve from the merging. The higher the tolerance, the more likely two *Bezier* curves can be merged into one. As *Bezier* curves are required for the merging, the *Smooth corners* option must be used (with a non-zero value).



The red circles are bitmap images and are traced using the *Brightness cutoff* method. From left to right: *Smooth corners* and *Optimize paths* not used: 42 corner nodes; *Smooth corners* used with a *Threshold* of 1.00: 24 smooth nodes and one corner node; *Smooth corners* used with a *Threshold* of 1.00, *Optimize paths* uses with a *Tolerance* of 0.20: 4 smooth nodes with 1 corner node.

Multiple Scans

Table of Contents

SIOX

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Tracing Bitmaps » SIOX

Index

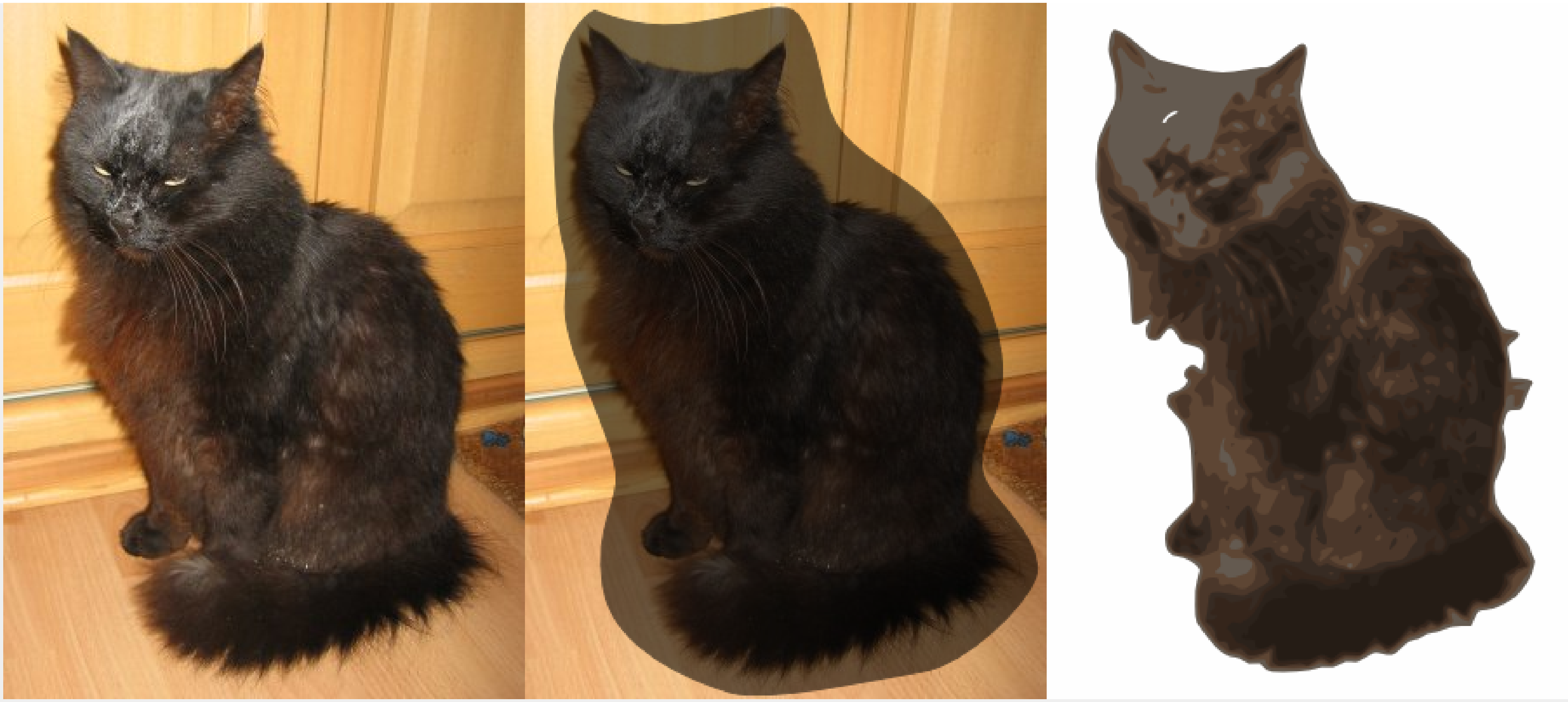
SIOX

Simple Interactive Object Extraction or SIOX allows one to separate an object from the background in a bitmap image. It acts as a preselection routine for normal tracing. The current implementation in Inkscape (v0.44 and v0.45) is rather primitive but it should improve in the future. At the moment, only a background region can be specified. In the future, a foreground region will also be definable.

The performance of SIOX depends greatly on the characteristics of the bitmap image. Colored bitmaps where the object is clearly distinguished in color from the background work best.

To use SIOX, check the *SIOX foreground selection* box in the *Trace Bitmap* dialog. The label is a bit misleading as the next step is to select a region that includes the entire object of interest and excludes most of the background. The SIOX algorithm uses the excluded region to characterize the background.

Create a closed path around the object you wish to extract. Give the path a fill if it doesn't already have one. Select both the bitmap image and the path and then trace as usual.



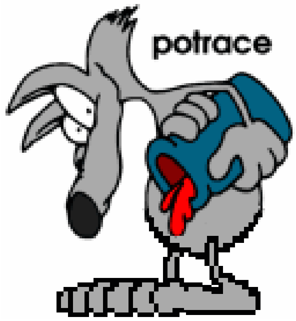
An example of using SIOX. Left, original bitmap image. Middle, image with background exclusion region added. Right, result of tracing. One can see that the background rejection is not perfect as there are similar colors in the background and foreground (e.g., reddish hair on chest).

Common Options

Table of Contents

Chapter 17. Connectors

Potrace



Transforming bitmaps into vector graphics

Contents

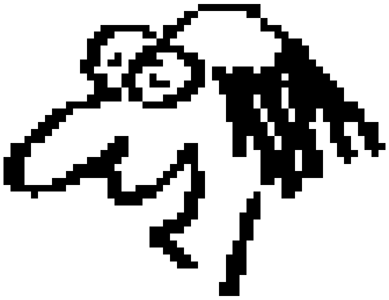
Description

Potrace(TM) is a utility for tracing a bitmap, which means, transforming a bitmap into a smooth, scalable image. The input is a bitmap (PBM, PGM, PPM, or BMP format), and the default output is an encapsulated PostScript file (EPS). A typical use is to create EPS files from scanned data, such as company or university logos, handwritten notes, etc. The resulting image is not "jaggy" like a bitmap, but smooth. It can then be rendered at any resolution.

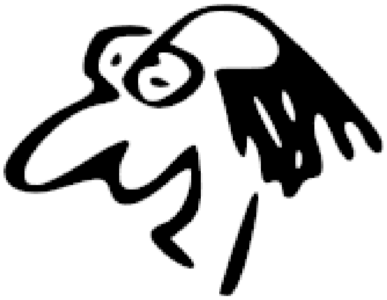
Potrace can currently produce the following output formats: EPS, PostScript, PDF, SVG (scalable vector graphics), Xfig, Gimppath, and PGM (for easy antialiasing). Additional backends might be added in the future.

Mkbitmap is a program distributed with Potrace which can be used to pre-process the input for better tracing behavior on greyscale and color images. See the mkbitmap examples page.

Example



Original image



Potrace output

More examples...

Mkbitmap examples...

Frequently asked questions

Trouble using Potrace? Here are the answers to some frequently asked questions.

News

For changes prior to version 1.6, see the file NEWS. For a more detailed list of changes, see the ChangeLog.

May 22, 2007: Updated XFig backend. The following patch to Potrace 1.8 improves the XFig output by adding a depth setting to the spline components. This prevents them from floating to the background when editing. Thanks to Rafael Laboissiere for suggesting this.

April 9, 2007: Release 1.8. This release contains minor bugfixes and portability improvements. Rotation is now implemented in the PDF backend.

March 6, 2005: Release 1.7. This is a bugfix release. A bug in the progress bar code, which caused arithmetic exceptions on some 64-bit architectures, has been fixed.

February 27, 2005: Release 1.6. This release contains an algorithm improvement that leads to a speedup of 20-60% over previous versions of Potrace. A new PDF backend was added, courtesy of Tor Andersson. An option --progress was added for displaying a progress bar. The Windows version of Potrace now uses MinGW instead of Cygwin, eliminating the need to install a special DLL alongside the executable programs, and solving some problems with wildcards and executable PostScript files. Some spurious "premature end of file" messages were eliminated. The core functionality of Potrace was separated into a library with a documented API, making it easier for developers to incorporate Potrace into other GPL-licensed software.

Downloading and installing

Potrace is built from sources using the standard configure/make commands. Please see the file INSTALL for generic installation instructions, and the file README for compile time configuration options specific to Potrace. Some pre-compiled binary distributions are also available. See the file README for instructions on how to install Potrace from a binary distribution. Additional instructions for Windows users are contained in the file README-WIN. If you would like to ensure the accuracy of the downloaded files, you can double-check their SHA1 sums.

Source distribution:		potrace-1.8.tar.gz
Precompiled distributions:	Linux (i386)	potrace-1.8.linux-i386.tar.gz
	Linux (Alpha)	potrace-1.7.linux-alpha.tar.gz
	Linux (AMD64)	potrace-1.8.linux-amd64.tar.gz
	Linux (Sparc)	potrace-1.1.linux-sparc.tar.gz
	Sun Solaris (Sparc)	potrace-1.7.solaris-sparc.tar.gz
	Sun Solaris (i386)	potrace-1.7.solaris2.9-i386.tar.gz
	FreeBSD (i386)	potrace-1.7.freebsd4.6-i386.tar.gz
	NetBSD (i386)	potrace-1.7.netbsdelf-i386.tar.gz
	OpenBSD (i386)	potrace-1.7.openbsd3.4-i386.tar.gz
	AIX	potrace-1.8.aix-rs6000.tar.gz
	Mac OS X (PowerPC)	potrace-1.7.darwin6.0-powerpc.tar.gz
	Mac OS X (i386)	potrace-1.8.mac-i386.tar.gz(1)
	Windows 95/98/2000/NT	potrace-1.8.win32-i386.zip potrace-1.8.win32-i386.tar.gz
	AROS (i386)	potrace-1.8.i386-aros.zip(2)
Packages:	Redhat Binary RPM (i386)	potrace-1.8-1.i386.rpm
	Redhat Source RPM	potrace-1.8-1.src.rpm
	Debian Package (i386)	potrace_1.7-1_i386.deb(3)
	Redhat Binary RPM (64-bit)	potrace-1.7-1.x86_64.rpm(4)
	SuSE Binary RPM (i586)	potrace_suse91-1.5-3.i586.rpm(5)
	SuSE Binary RPM (64-bit)	potrace_suse91-1.5-2.x86_64.rpm(5)

	Solaris 9 Package (i386)	potrace-1_7-i386-pc-solaris9.pkg.gz(6)
	Solaris 10 Package (i386)	potrace-1_8-i386-pc-solaris10.pkg.gz(6)
	Amiga (OS4)	potrace.lha (version 1.7)(7)
Package Management:	Debian	apt-get install potrace(3)
	ArchLinux	pacman -S potrace(8)
	FreeBSD	pkg_add -r potrace(9)
	Fink (Mac OS X)	fink install potrace(10)
	Macports (Mac OS X)	port install potrace(11)

(1) Intel binaries for Macintosh supplied by Arno Nuyts <arno at scoutsevere.be>.
(2) AROS binaries supplied by Matthias Rustler <mrustler at gmx.de>. See the AROS archives for sources and updates.
(3) Debian i386 Package maintained by Bartosz Fenski <fenio at debian.org>. Debian has a centralized package management and users may run "apt-get install potrace" as root to install.
(4) Redhat x86_64 Packages provided by Manuel Joriatti <manu-schwalbe at wanadoo.fr>.
(5) SuSE RPMs provided by Stanislav Brabec <sbrabec at suse.cz>. Potrace appears in the official SuSE distribution. Updates sometimes appear in the SuSE supplementary program.
(6) Solaris-i386 Packages provided by Apostolos Syropoulos <asyropoulos at gmail.com>.
(7) Amiga Package provided by Alfred Faust <alfred.j.faust at gmx.de>. See the OS4 Depot for updates.
(8) ArchLinux package maintained by Damir Perisa <damir at archlinux.org>. ArchLinux has a centralized package management and users need to run "pacman -S potrace" as root to install.
(9) FreeBSD Packages maintained by Piotr Smyrak <smyru at heron.pl>.
(10) Fink package maintained by Daniel Macks <dmacks at netspace.org>. Fink has a centralized package management and users need to run "fink install potrace" as root to install.
(11) Macports has a centralized package management and users need to run "port install potrace" as root to install.

Previous releases...

Usage

```
Potrace 1.8. Transforms bitmaps into vector graphics.

Usage: potrace [options] [file...]
General options:
 -h, --help                 - print this help message and exit
 -v, --version              - print version info and exit
 -l, --license              - print license info and exit
 -V, --show-defaults        - print compiled-in defaults and exit
 --progress                 - show progress bar
Input/output options:
 -o, --output file          - output to file
Backend selection:
 -e, --eps                  - EPS backend (encapsulated postscript) (default)
 -p, --postscript           - Postscript backend
 -s, --svg                  - SVG backend (scalable vector graphics)
 -g, --pgm                  - PGM backend (portable greymap)
 -b, --backend name         - select backend by name
Algorithm options:
 -z, --turnpolicy policy    - how to resolve ambiguities in path decomposition
 -t, --turdsize n           - suppress speckles of up to this size (default 2)
 -a, --alphamax n           - corner threshold parameter (default 1)
 -n, --longcurve            - turn off curve optimization
 -O, --opttolerance n       - curve optimization tolerance (default 0.2)
 -u, --unit n               - quantize output to 1/unit pixels (default 10)
 -d, --debug n              - produce debugging output of type n (n=1,2,3)
Scaling and placement options:
 -W, --width dim            - width of output image
 -H, --height dim           - height of output image
 -r, --resolution n[xn]     - resolution (in dpi)
 -x, --scale n[xn]          - scaling factor (pgm backend)
 -S, --stretch n            - yresolution/xresolution
 -A, --rotate angle         - rotate counterclockwise by angle
 -M, --margin dim           - margin
 -L, --leftmargin dim       - left margin
 -R, --rightmargin dim      - right margin
 -T, --topmargin dim        - top margin
 -B, --bottommargin dim     - bottom margin
Output options, supported by some backends:
 -C, --color #rrggbb        - set line color (default black)
 --fillcolor #rrggbb        - set fill color (default transparent)
 --opaque                   - make white shapes opaque
 --group                    - group related paths together
Postscript/EPS options:
 -P, --pagesize format      - page size (default is letter)
 -c, --cleartext            - do not compress the output
 -2, --level2               - use postscript level 2 compression (default)
 -3, --level3               - use postscript level 3 compression
 -q, --longcoding           - do not optimize for file size
PGM options:
 -G, --gamma n              - gamma value for anti-aliasing (default 2.2)
```

```
Frontend options:
 -k, --blacklevel n          - black/white cutoff in input file (default 0.5)
 -i, --invert                - invert bitmap

Dimensions can have optional units, e.g. 6.5in, 15cm, 100pt.
Default is inches (or pixels for pgm and gimppath backends).
Possible input file formats are: pnm (pbm, pgm, ppm), bmp.
Backends are: eps, postscript, ps, pdf, svg, pgm, gimppath, xfig.
```

For detailed usage information, see the potrace(1) man page.

Technical documentation

- For detailed usage information, see the man pages of potrace(1) and mkbitmap(1).
- A detailed technical description of how the Potrace algorithm works [ps, pdf].
- Technical documentation of the Potrace library API (for developers) [ps, pdf].

Support and reporting bugs

Potrace has a project page on SourceForge. There you will find facilities for reporting bugs, submitting patches, asking for support, asking for features, or discussing Potrace in general. You are encouraged to use these facilities. You can also send email to the author. Please also check the Frequently Asked Questions.

Dual licensing program

A non-GPL version of Potrace, called Potrace Professional(TM), is available for integration into proprietary software. Licenses are available from my company, Icosasoft Software Inc. If you wish to purchase a license, or for more information, please write to licensing@icosasoft.ca.

GUI's and related software

Commercial software:

- **Total Vectorize** by CoolUtils. This Windows(TM) application makes vectorizing images simple and fun. Get the power of Potrace Professional(TM) in an easy-to-install and easy-to-use package. With one click, you can either vectorize a single image or a batch of images. Try it free for 30 days! (Added Feb 27, 2008).

Graphical user interfaces:

- **Potrace GUI**, by Rasmus Andersson. A GUI for Potrace for Mac OS X. The source code is here. (Added Sept 22, 2003).

- **potracegui**, by Antonio Fasolato. A GUI for Potrace using KDE. (Added Nov 29, 2003).

- **delineate**, by Robert McKinnon. A GUI for Potrace and Autotrace using Java. There are versions for Mac OSX, GNU/Linux, and Windows. (Added Feb 11, 2003).

- **Rasterbater 1.0**, by Kevin Cole. A GUI for Potrace using C#. (Added December 20, 2007).

- **CR8tracer**, by Allan Murray. A Windows GUI for Potrace. (Added December 20, 2007).

Software that uses Potrace:

- **FontForge**, by George Williams. An outline font editor that lets you create and edit postscript and truetype fonts, among others. This program can invoke Autotrace or Potrace for converting bitmaps to vector fonts. It used to be called "PfaEdit". (Added Aug 21, 2003).

- **mftrace**, by Han-Wen Nienhuys. A small Python program that lets you trace a TeX bitmap font into a PFA or PFB font (A PostScript Type1 Scalable Font) or TTF (TrueType) font (Added Dec 13, 2003).

- **LilyPond**, by Han-Wen Nienhuys and Jan Nieuwenhuizen. An application that let you typeset beautiful sheet music. It uses Potrace indirectly via mftrace. (Added Apr 22, 2004).

- **TeXtrace**, by Péter Szabó. A collection of scripts that convert any TeX font into a Type1 .pfb outline font. The newest version (0.50) supports Potrace via the --potrace option. (Added Apr 22, 2004).

- **Inkscape**, by various authors. This is an excellent SVG editor. I just received news that Potrace will be integrated in the next release, 0.40. (Added Sep 16, 2004).

- **dvisvgm**, by Martin Gieseking. This is a utility for converting DVI files, as generated by TeX/LaTeX, to the XML-based Scalable Vector Graphics format. (Added Aug 24, 2005).

- **pocopo**, by Paul Yoon. An AppleScript for processing color images, using Potrace. (Added Oct 28, 2005).

Software that is similar to Potrace:

- **Autotrace**, by Martin Weber. This program performs a similar task as Potrace. In my opinion, the output is not as nice, but it supports a much larger number of file formats, and it has been integrated with a larger number of other software packages. (Added Nov 29, 2003).
- **Vector Magic**. A commercial tracing tool that is available through a web interface, for a fee. Unlike Potrace, Vector Magic works for color images.

Non tracing-based image enhancement software:

- **GREYCstoration**, by David Tschumperlé. If the examples on the webpage are representative, then this is the most astonishingly good image regularization filter that I have ever seen. It is based on a non-linear diffusion technique. It can be used for noise and artifact removal, resizing, and inpainting (which means filling in missing image regions). It works on color photographs and cartoons. (Added Feb 2, 2006).

SVG Editors:

- **Inkscape**.

Articles about Potrace

Here are some web articles, listings, and web logs that cover Potrace and its applications. I particularly recommend the article by C. Scott Ananian on how to make stained glass windows from color scans using open source software. It is an amazing, step-by-step description of the process used, with beautiful screen shots. Similar techniques might also be applicable for vectorizing color cartoons.

- Automatic Generation of Stained Glass from Scanned Photos by C. Scott Ananian.
- Trying Potrace [English] by Exocert.com.
- Potrace, rad image vectorizer by Will Benton.
- The power of Potrace [English] by Germán Poó Caamaño.
- Converting images from bitmap to vector format [English] by Mike Vargas.
- Compiling bitmaps to SVG and PDR using Potrace by Sarusekkei [English].
- Unixuser200403-3 by wiki.fdiary.net (Japanese only).
- Conference poster: "Emergency, full disclosure and free software" [English] by Mirko Maischberger.
- Creating Type 1 Fonts from METAFONT Sources: Comparison of Tools, Techniques and Results by Karel Piska (PDF only).
- The little things by daveg@dorja.
- Open Source Alternatives by Anders Rasmussen.

Version

1.8

Author

Logo

The Potrace logo and mascot was designed by Karol Krenski. Copyright © 2003 Karol Krenski and Peter Selinger. The logo is licensed under GPL.

License

Trademarks

"Potrace" is a trademark of Peter Selinger. "Potrace Professional" and "Icosasoft" are trademarks of Icosasoft Software Inc. Other trademarks belong to their respective owners.

To my other software.

Peter Selinger / Department of Mathematics and Statistics / Dalhousie University
selinger@users.sourceforge.net / PGP key

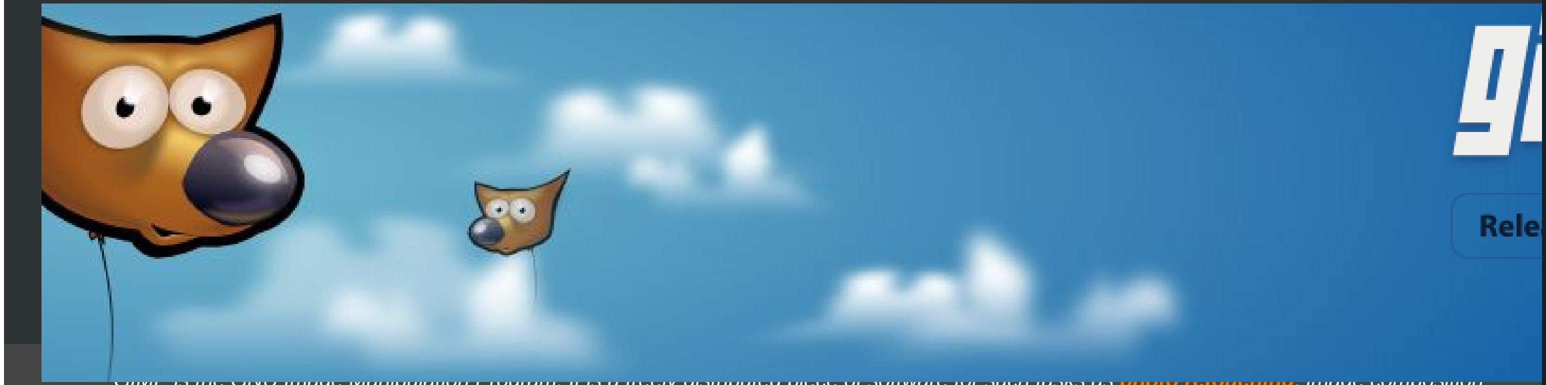
349844

GNU Image Manipulation Program

- News
- Screenshots
- Features
- Downloads
- Documentation
- Get Involved

- Plug-in Registry
- GIMP Development
- GIMP User Group

Make a Donation



GIMP is the GNU Image Manipulation Program. It is a freely distributed piece of software for such tasks as photo retouching, image composition and image authoring. It works on many operating systems, in many languages. (more...)

This is the official GIMP web site. It contains information about downloading, installing, using, and enhancing it. This site also serves as a distribution point for the latest releases. We try to provide as much information about the GIMP community and related projects as possible. Hopefully you will find what you need here. Grab a properly chilled beverage and enjoy.

GIMP 2.5.2 Development Release 2008-07-16

Getting closer to GIMP 2.6, the GIMP developers released another snapshot from the 2.5 development series. The NEWS file has a summary of the changes.

GIMP 2.5.1 Released 2008-06-15

GIMP 2.5.1 is another snapshot from the 2.5 development series. It gives developers and interested users a view into the current development towards GIMP 2.6. The changes in GIMP 2.5 are listed in the NEWS file.

If you want to give this development snapshot a try, please make sure you read the Release Notes for GIMP 2.5.

GIMP 2.4.6 Released 2008-05-30

GIMP 2.4.6 is a bug-fix release in the stable 2.4 series. Please see the NEWS file for a list of changes. The source can be downloaded from ftp.gimp.org. Binary packages should become available soon; please check the Downloads section.

GIMP 2.5 Development Snapshot 2008-04-10

GIMP 2.5.0 is the first release from the 2.5 development series. It gives developers and interested users a view into the current development towards GIMP 2.6. If you want to give this release a try, please make sure you read the Release Notes for GIMP 2.5.

GIMP User Manual 2.4.1 released 2008-04-09

A new release of the user manual is available:

- new translations: Lithuanian, Polish
- new content, spelling and grammar fixes for German, French, Italian, Norwegian, Korean, Spanish, Russian, English
- updated quick reference translations: German, English, French, Russian, Swedish, Italian

The source files of **gimp-help-2.4.1** can be downloaded from ftp.gimp.org. You will notice that the tarball is a lot smaller than the previous releases as we optimized the screen-shots.

Users should wait until this release has been packaged in a pre-compiled form for their platform. Find more information about our goals and how you can help at docs.gimp.org.

Libre Graphics Meeting 2008 2008-04-02

The 3rd annual Libre Graphics Meeting will be held in Wrocław, Poland May 8 - 11. The conference brings together developers and users of free software graphics applications, such as GIMP, Inkscape, Scribus, Blender, Krita, the Open Clipart Library and more.

DONATE pledgie.com $98.15 Over Goal! We are trying to raise USD$ 20,000 in the next 16 days before Friday, April 18th in order to support the conference and to help developers with travel and accommodation costs.

Google Summer of Code 2008

2008-03-18

Here is your chance to join GIMP development: GIMP has been accepted as a mentoring organization for Google Summer of Code 2008. Check the proposals, come up with your own ideas, talk to the developers on the mailing list or IRC, and sign up for work!

- Features
- Release Notes
- Wiki
- Screenshots
- Downloads
- Unix
- Windows
- Mac OS X
- Documentation
- FAQ
- Books
- Tutorials
- Mailing Lists
- IRC
- History
- Splash Archive
- Links
- Get Involved
- Donating
- Bug Reports
- GIMP Goods

GIMP News Feed | Contacting Us

Creating Connectors

Connectors are created by the *Connector Tool*. To create a connector, select the tool by clicking on the icon (**Ctrl**+**F2** or **o**) in the *Tool Box*. Then click-drag the mouse from one point on the canvas to another. When the pointer is over an object, a *Connection* handle is shown in the center of the object. Beginning or ending the click-drag on one of these handles will attach the connector to the corresponding object. Alternatively, one can begin a connector by clicking on an empty point on the canvas or on a *Connection* handle and end the connector by a second click.

Attached connectors will be drawn so they begin at the *bounding box* of the attached object and on a line that is radial to the center of the object. Future improvements will allow drawing of connectors that begin at the edge of an object.

By default, connectors will not attach to text objects. This facilitates connections between boxes that frame text. You can change this option under the *Connector* entry of the *Inkscape Preferences* dialog.

Modifying Connectors

Connectors can be modified several ways. The first is that the end points can be dragged around. Dragging can disconnect a connected end point or connect a disconnected one. If a connected object is moved, the connector will follow the object but it may no longer be radial to the center of the object. This can be fixed by dragging the corresponding connector end to the *Connection* handle.

A second way connectors can be modified is to change their line style, This can be done with the *Fill and Stroke* dialog (Object → Fill and Stroke... (**Shift+Ctrl+F**)).

Tip

Changing the *Cap style* to *Square Cap* may improve the look of a connection to a stroked object.

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Connectors » Routing Connectors ← Index →

Routing Connectors

Connectors are automatically routed. This includes routing around objects marked for avoidance. To mark an object that connectors should not cross:

1. Select the objects to be avoided (e.g., with *Select Tool*).
2. Select the *Connector Tool*.
3. Click on the *Avoid objects* ( ) icon in the *Tool Controls*.

To allow connectors to cross an object (the default), follow the above procedure but click the *Ignore objects* ( ) icon in the *Tool Controls*.

Connectors are automatically rerouted when either an object that the connector is attached to is moved or when objects that are marked to be avoided are modified.

The *Tool Controls* for the *Connector Tool* has an entry for *Spacing*. This adjusts the “padding” around objects that connectors must avoid.

The *Align and Distribute* dialog has a clickable icon ( ) for automatically rearranging connectors and connected objects in a “nice” arrangement. The placement is based on the Kamada-Kawai algorithm that treats the connectors as springs so that the distance between the *connector* handles are evenly spaced. Your mileage may vary.

Modifying Connectors

Table of Contents

Chapter 18. Effects

Inkscape: Guide to a Vector Drawing Program

Get the book.

Inkscape » Effects » Color

Index

Color

New in v0.45.

This set of effects manipulates the colors of an object or *Group* of objects. The effects are implemented through a set of Python scripts (located in the directory `share/inkscape/extensions`). The use of Python scripts is a temporary measure until the capability is incorporated natively into Inkscape. The color mapping is calculated in the *RGB* color space except for the effects that modify *HSL* values, which are calculated in *HSL* color space.

If no objects are selected, the color change will be applied to the entire drawing. An object's *Stroke* and any *Gradient* are also changed.

Desaturate

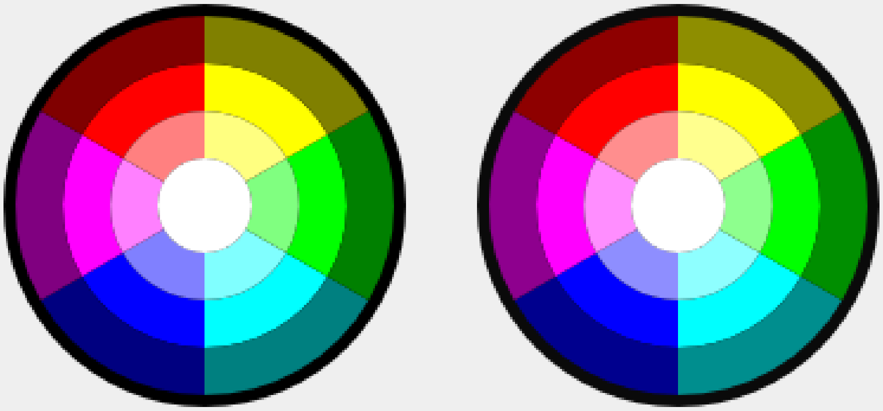
Desaturate the color of an object or *Group* of objects. This sets the values of R, G, and B to the average of the maximum of R, G, and B; and the minimum of R, G, and B. For example, R would be set to (max(R, G, B) + min(R, G, B))/2.

Color Desaturate example.

Left: Original colors. Right: Colors after applying the Color Desaturate effect.

Brighter

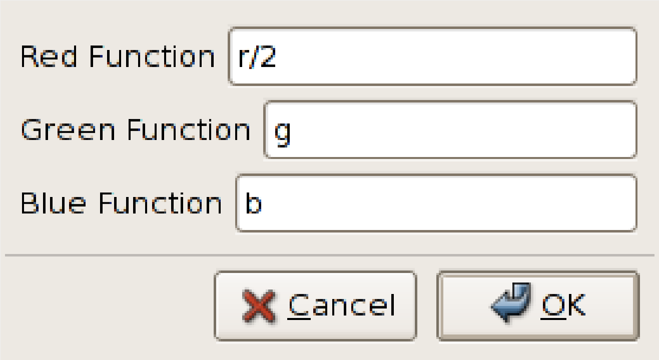
Brighten the color of an object or *Group* of objects. This effect has the property of making dark colors more intense but washing out light colors.



Left: Original colors. Right: Colors after applying the Color Brighter effect.

Custom

This effect allows color custom transformation functions to be defined. Standard math operations are allowed such as +, –, ×, and /. If a a resulting value is outside the allowed limits, it is set at the minimum or maximum allowed value.

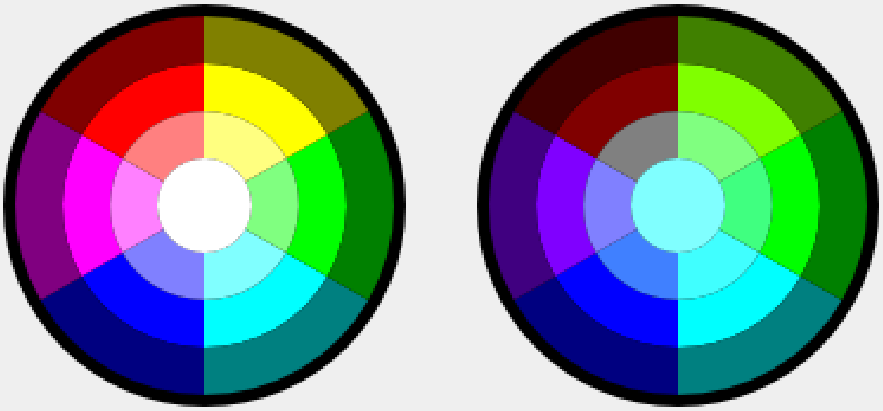


Red Function r/2

Green Function g

Blue Function b

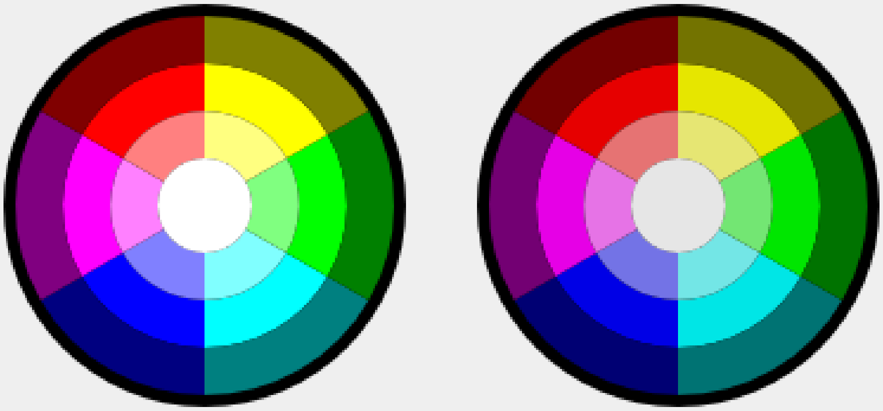
Custom Color Effect dialog set to reduce R (red) to one half.



Left: Original colors. Right: Colors after applying the Color Custom effect with the specification that R should be divided by 2.

Darker

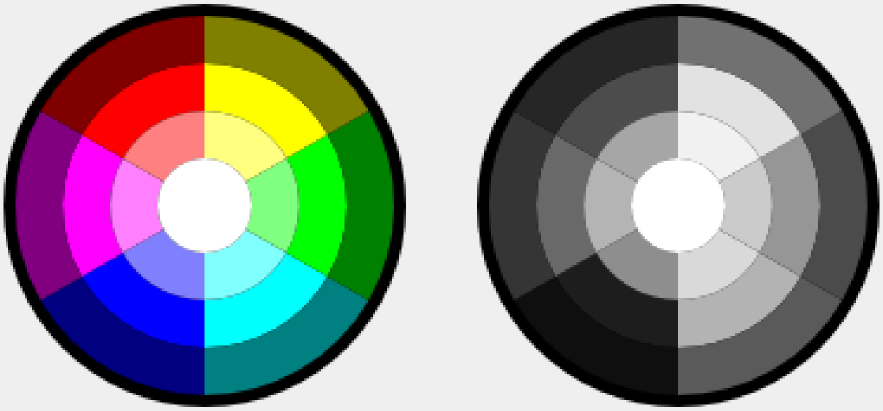
Darken the color of an object or *Group* of objects. Each R, G, and B component of a color is set to 90% of its previous value.



Left: Original colors. Right: Colors after applying the Color Darker effect.

Grayscale

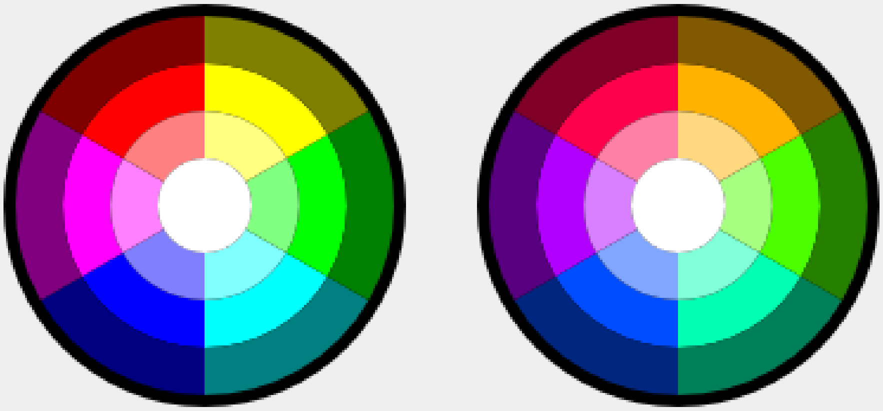
Change the color to a gray using the formula for Luminance used by the NTSC and PAL television standards. This sets the color to a *lightness* (Y) defined by: Y = 0.229 × R + 0.587; × G + 0.114 × B. See the Wikipedia entry for YUV for further information.



Left: Original colors. Right: Colors after applying the Color Grayscale effect.

Less Hue

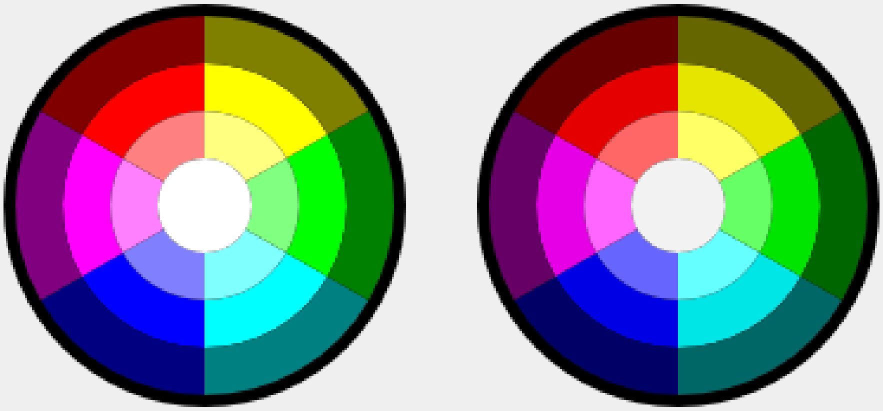
Decrease the hue (see the section called “HSL”) of a color. The hue is decreased by 5% (of the full hue range) or equivalently, a rotation of 18° around the color circle. This, for example, means that a pure red picks up a touch of blue in the *RGB* color space.



Left: Original colors. Right: Colors after applying the Color Less Hue effect.

Less Light

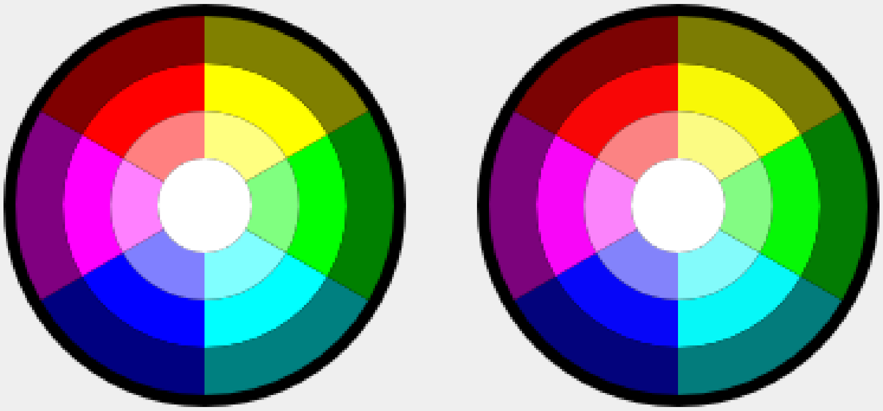
Decrease the lightness (see the section called “HSL”) of a color. The lightness is decreased by 5% (of the full lightness range). If the lightness is already less than 5%, it is set to 0%.



Left: Original colors. Right: Colors after applying the Color Less Light effect.

Less Saturation

Decrease the saturation (see the section called “HSL”) of a color. The saturation is decreased by 5% (of the full saturation range). If the saturation is already less than 5%, it is set to 0%.



Left: Original colors. Right: Colors after applying the Color Less Saturation effect.

More Hue

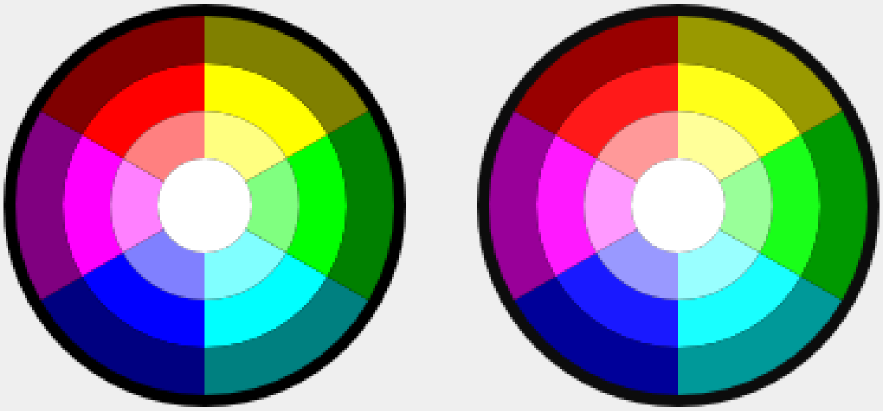
Increase the hue (see the section called “HSL”) of a color. The hue is increased by 5% (of the full hue range) or equivalently, a rotation of 18 degrees, around the color circle. This, for example, means that a pure red picks up a touch of green in the *RGB* color space.

Color MoreHue example.

Left: Original colors. Right: Colors after applying the Color More Hue effect.

More Light

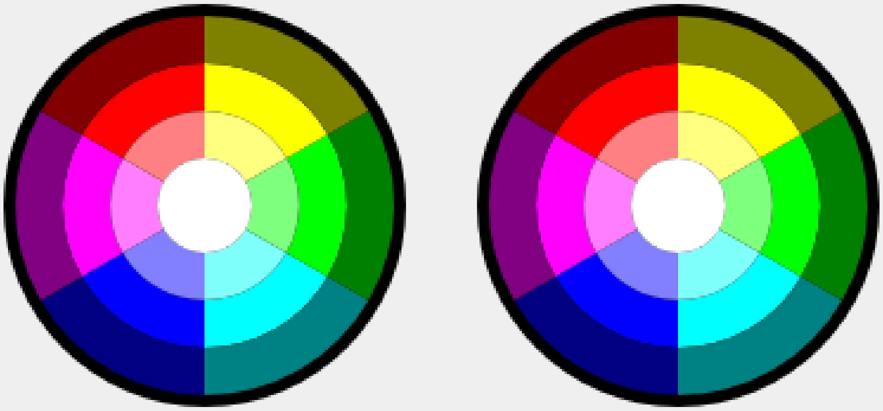
Increase the lightness (see the section called “HSL”) of a color. The lightness is increased by 5% (of the full lightness range). If the lightness is already more than 95%, it is set to 100%.



Left: Original colors. Right: Colors after applying the Color More Light effect.

More Saturation

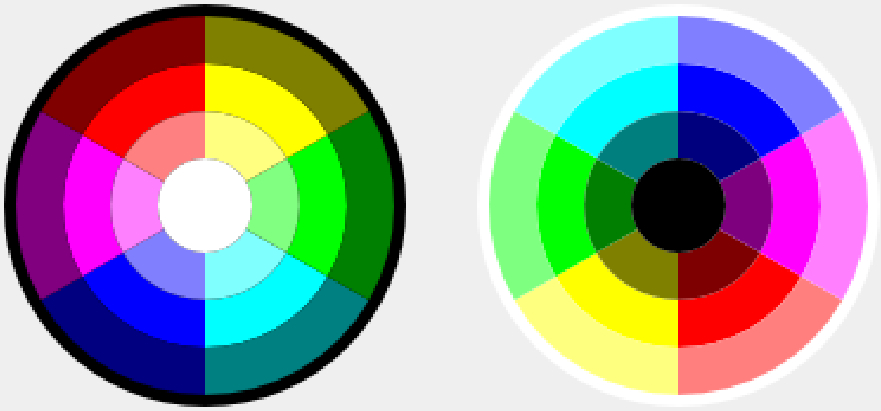
Increase the saturation (see the section called “HSL”) of a color. The saturation is increased by 5% (of the full saturation range). If the saturation is already more than 95%, it is set to 100%.



Left: Original colors. Right: Colors after applying the Color More Saturation effect.

Negative

Invert the color. For example, an R value of 64 (25%) becomes an R value of 191 (255–64, or 75%).

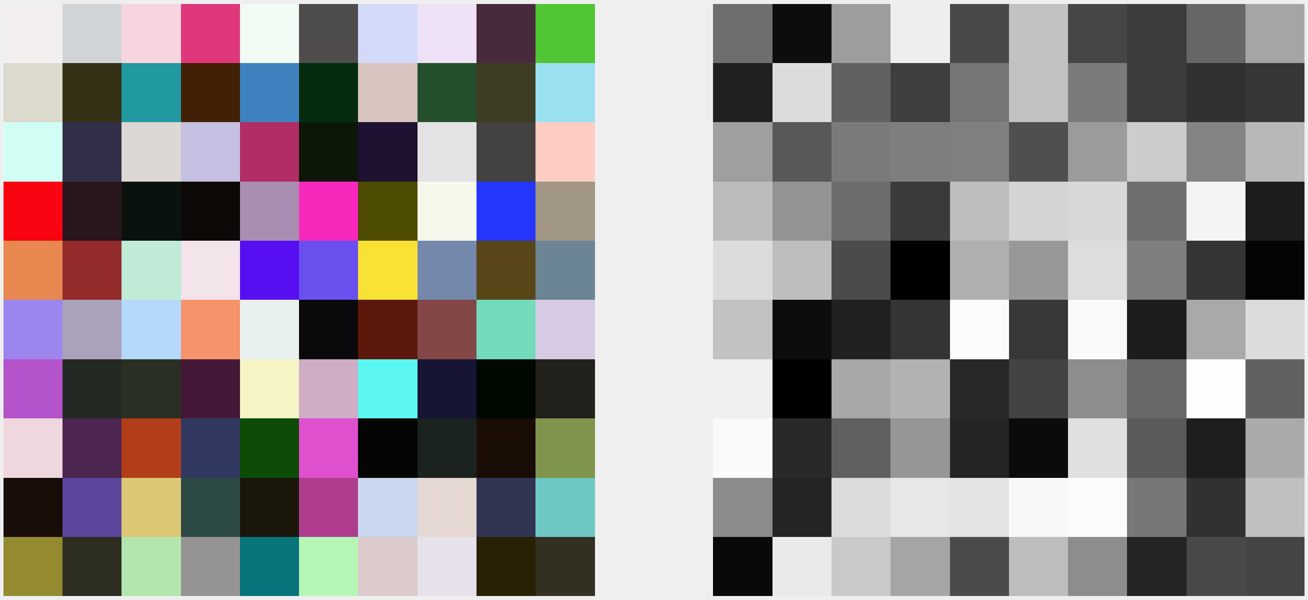


Left: Original colors. Right: Colors after applying the Color Negative effect.

Randomize

New in v0.46

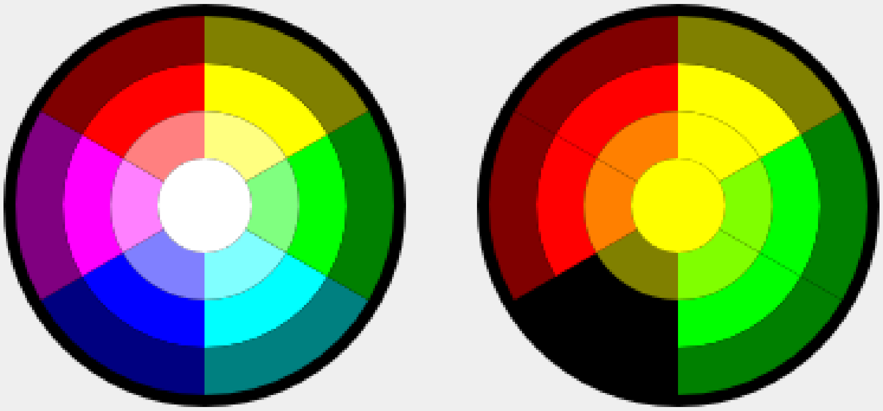
Randomize the color of selected objects or all objects if no object is selected. One can choose which of the *HSL* color parameters to randomize (hue, saturation, and/or lightness).



A grid of gray squares after randomizing their colors. Left: Randomizing hue, saturation, and lightness. Right: Randomizing only lightness.

Remove Blue

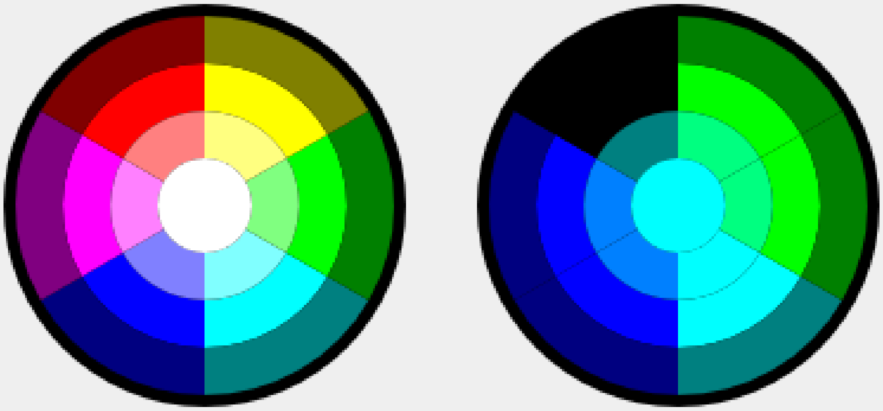
Set the B value in *RGB* to 0.



Left: Original colors. Right: Colors after applying the Color Remove Blue effect.

Remove Red

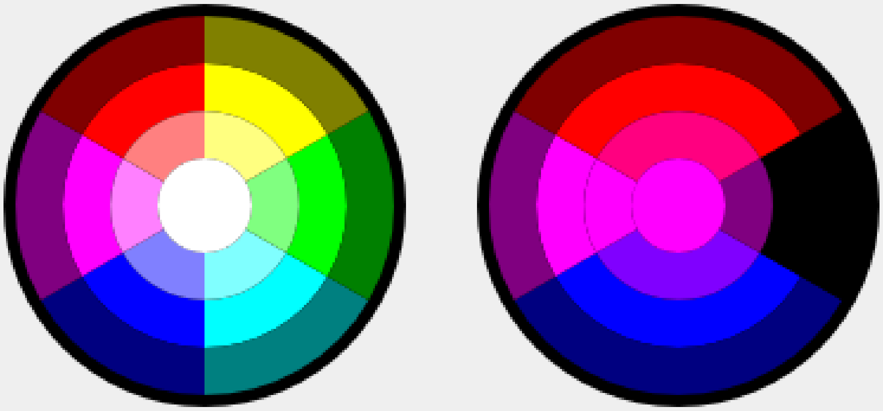
Set the R value in *RGB* to 0.



Left: Original colors. Right: Colors after applying the Color Remove Red effect.

Remove Green

Set the G value in *RGB* to 0.



Left: Original colors. Right: Colors after applying the Color Remove Green effect.

Replace

New in v0.46

Replace a color among selected objects or all objects if no object is selected. Color are specified in RRGGBB *hexadecimal* form. Color to be replaced must match exactly.

Chapter 18. Effects
Table of Contents
Export