Filter Effects Module Level 1

W3C Working Draft, 18 December 2018

This version:
https://www.w3.org/TR/2018/WD-filter-effects-1-20181218/
Latest published version:
https://www.w3.org/TR/filter-effects-1/
Editor's Draft:
https://drafts.fxtf.org/filter-effects-1/
Previous Versions:
https://www.w3.org/TR/2018/WD-filter-effects-1-20181124/
https://www.w3.org/TR/2014/WD-filter-effects-1-20141125/
https://www.w3.org/TR/2013/WD-filter-effects-1-20131126/
https://www.w3.org/TR/2013/WD-filter-effects-20130523/
https://www.w3.org/TR/2012/WD-filter-effects-20121025/
Test Suite:
http://test.csswg.org/suites/filter-effects/nightly-unstable/
Issue Tracking:
Inline In Spec
GitHub Issues
Editors:
Dirk Schulze (Adobe Inc.)
Dean Jackson (Apple Inc.)
Former Editors:
Vincent Hardy
Erik Dahlström (Invited Expert)
Suggest an Edit for this Spec:
GitHub Editor

Abstract

Filter effects are a way of processing an element's rendering before it is displayed in the document. Typically, rendering an element via CSS or SVG can conceptually be described as if the element, including its children, are drawn into a buffer (such as a raster image) and then that buffer is composited into the elements parent. Filters apply an effect before the compositing stage. Examples of such effects are blurring, changing color intensity and warping the image.

Although originally designed for use in SVG, filter effects are a set of operations to apply on an image buffer and therefore can be applied to nearly any presentational environment, including CSS. They are triggered by a style instruction (the 'filter' property). This specification describes filters in a manner that allows them to be used in content styled by CSS, such as HTML and SVG. It also defines a CSS property value function that produces a CSS <image> value.

CSS is a language for describing the rendering of structured documents (such as HTML and XML) on screen, on paper, etc.

Status of this document

GitHub Issues are preferred for discussion of this specification. When filing an issue, please put the text "filter-effects" in the title, preferably like this: "[filter-effects] *…summary of comment…*". All issues and comments are archived, and there is also a historical archive.

This document was produced by the CSS Working Group (part of the Style Activity).

Table of Contents

§ 1. Introduction

This section is not normative

A filter effect is a graphical operation that is applied to an element as it is drawn into the document. It is an image-based effect, in that it takes zero or more images as input, a number of parameters specific to the effect, and then produces an image as output. The output image is either rendered into the document instead of the original element, used as an input image to another filter effect, or provided as a CSS image value.

A simple example of a filter effect is a “flood”. It takes no image inputs but has a parameter defining a color. The effect produces an output image that is completely filled with the given color. A slightly more complex example is an “inversion” which takes a single image input (typically an image of the element as it would normally be rendered into its parent) and adjusts each pixel such that they have the opposite color values.

Filter effects are exposed with two levels of complexity:

1. A small set of canned filter functions that are given by name. While not particularly powerful, these are convenient and easily understood and provide a simple approach to achieving common effects, such as blurring. The canned filters can also be animated by [CSS3-ANIMATIONS].
2. A graph of individual filter effects described in markup that define an overall effect. The graph is agnostic to its input in that the effect can be applied to any content. While such graphs are the combination of effects that may be simple in isolation, the graph as a whole can produce complex effects. An example is given below.

EXAMPLE 1

In this example, an image is filtered with the <grayscale()> filter function.

```
#image {
    filter: grayscale(100%);
}
```

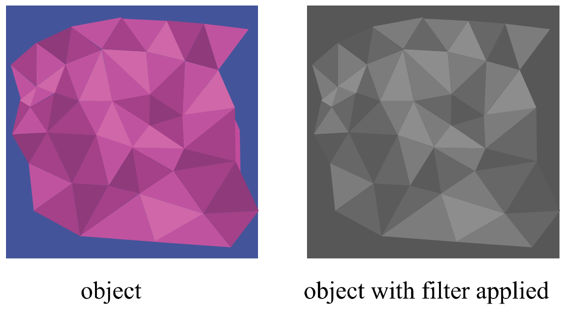


Figure 1. *An image without filter (left) and the same filter with a 100% grayscale filter (right).*

EXAMPLE 2
The following shows an example of graph of individual filter effects.

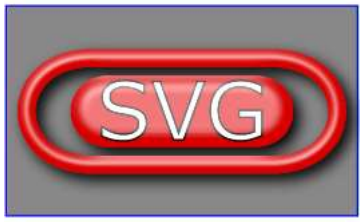


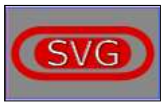
Figure 2. *Initial example for a filtered object.*

View this example as SVG

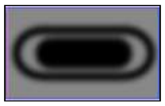
The filter effect used in the example above is repeated here with reference numbers in the left column before each of the six filter primitives:

```
    <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>
```

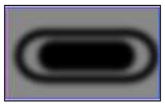
The following pictures show the intermediate image results from each of the six filter elements:



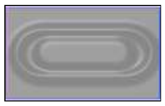
Source graphic



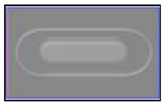
After filter primitive 1



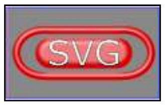
After filter primitive 2



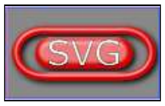
After filter primitive 3



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.

§ 2. Module interactions

This specification defines a set of CSS properties that affect the visual rendering of elements to which those properties are applied; these effects are applied after elements have been sized and positioned according to the Visual formatting model from [CSS21]. Some values of these properties result in the creation of a containing block, and/or the creation of a stacking context.

The compositing model follows the SVG compositing model [SVG11]: first any filter effect is applied, then any clipping, masking and opacity [CSS3COLOR]. These effects all apply after any other CSS effects such as 'border' [CSS3BG].

Some property and element definitions in this specification require an SVG 1.1 implementation [SVG11]. UAs without support for SVG must not implement the 'color-interpolation-filters', 'flood-color', 'flood-opacity' and 'lighting-color' properties as well as the `<filter>` element, the `<feMergeNode>` element, the transfer function elements and the filter primitive elements.

§ 3. Values

This specification follows the CSS property definition conventions from [CSS21]. Value types not defined in these specifications are defined in CSS Values and Units Module Level 3 [CSS3VAL].

In addition to the property-specific values listed in their definitions, all properties defined in this specification also accept the inherit keyword as their property value. For readability it has not been repeated explicitly.

§ 4. Terminology

When used in this specification, terms have the meanings assigned in this section.

***filter primitive*, `filter-primitive`**
: The set of elements that control the output of a `<filter>` element, particularly: `<feSpotLight>`, `<feBlend>`, `<feColorMatrix>`, `<feComponentTransfer>`, `<feComposite>`, `<feConvolveMatrix>`, `<feDiffuseLighting>`, `<feDisplacementMap>`, `<feDropShadow>`, `<feFlood>`, `<feGaussianBlur>`, `<feImage>`, `<feMerge>`, `<feMorphology>`, `<feOffset>`, `<feSpecularLighting>`, `<feTile>`, `<feTurbulence>`.

pass through filter
: The pass through filter output is equal to the primary input of the filter primitive.

§ 5. Graphic filters: the 'filter' property

The description of the 'filter' property is as follows:

Name:	*'filter'*
Value:	none | <filter-value-list>
Initial:	none
Applies to:	All elements. In SVG, it applies to container elements without the `<defs>` element, all graphics elements and the `<use>` element.
Inherited:	no
Percentages:	n/a
Computed value:	as specified
Canonical order:	per grammar
Media:	visual
Animatable:	See prose in Animation of Filters.

`<filter-value-list> = [ <filter-function> | <url> ]+`

'<url>'

A filter reference to a `<filter>` element. For example 'url(commonfilters.svg#filter)'. If the filter references a non-existent object or the referenced object is not a filter element, then the whole filter chain is ignored. No filter is applied to the object.

<filter-function>

See Filter Functions.

none

No filter effect gets applied.

A value other than 'none' for the 'filter' property results in the creation of a containing block for absolute and fixed positioned descendants unless the element it applies to is a document root element in the current browsing context. The list of functions are applied in the order provided.

The first filter function or `<filter>` reference in the list takes the element (SourceGraphic) as the input image. Subsequent operations take the output from the previous filter function or filter reference as the input image. filter element reference functions can specify an alternate input, but still uses the previous output as its SourceGraphic.

'color-interpolation-filters' has no affect for Filter Functions. Filter Functions must operate in the sRGB color space.

A computed value of other than 'none' results in the creation of a stacking context [CSS21] the same way that CSS 'opacity' does. All the elements descendants are rendered together as a group with the filter effect applied to the group as a whole.

The 'filter' property has no effect on the geometry of the target element's CSS boxes, even though 'filter' can cause painting outside of an element's border box.

Conceptually, any parts of the drawing are effected by filter operations. This includes any content, background, borders, text decoration, outline and visible scrolling mechanism of the element to which the filter is applied, and those of its descendants. The filter operations are applied in the element's local coordinate system.

The compositing model follows the SVG compositing model [SVG11]: first any filter effect is applied, then any clipping, masking and opacity. As per SVG, the application of 'filter' has no effect on hit-testing.

The 'filter' property is a presentation attribute for SVG elements.

ISSUE 1 How does filter behave on fixed background images? <https://github.com/w3c/csswg-drafts/issues/238>

§ 6. Filter Functions

§ 6.1. Supported Filter Functions

`<filter-function>` = `<blur()>` | `<brightness()>` | `<contrast()>` | `<drop-shadow()>` | `<grayscale()>` | `<hue-rotate()>` | `<invert()>` | `<opacity()>` | `<sepia()>` | `<saturate()>`

Unless defined otherwise, omitted values default to the initial value for interpolation.

> Note: For some filter functions the default value for omitted values differes from their initial value for interpolation. For the convenience of content creators, the default value for omitted values for <grayscale()>, <sepia()> and <invert()> is ‘1’ (apply the effect to 100%) while the initial value for interpolation is ‘0’ (no effect).

blur() **= blur(`<length>?`)**

Applies a Gaussian blur to the input image. The passed parameter defines the value of the standard deviation to the Gaussian function. The parameter is specified a CSS length, but does not accept percentage values. The markup equivalent of this function is given below.

Negative values are not allowed.

Default value when omitted is ‘0px’.

The initial value for interpolation is ‘0px’.

> Note: Standard deviation is different to ‘box-shadow’ s blur radius.

brightness() **= brightness(`<number-percentage>?`)**

Applies a linear multiplier to input image, making it appear more or less bright. A value of ‘0%’ will create an image that is completely black. A value of ‘100%’ leaves the input unchanged. Other values are linear multipliers on the effect. Values of amount over 100% are allowed, providing brighter results. The markup equivalent of this function is given below.

Negative values are not allowed.

Default value when omitted is ‘1’.

The initial value for interpolation is ‘1’.

contrast() **= contrast(`<number-percentage>?`)**

Adjusts the contrast of the input. A value of ‘0%’ will create an image that is completely gray. A value of ‘100%’ leaves the input unchanged. Values of amount over 100% are allowed, providing results with more contrast. The markup equivalent of this function is given below.

Negative values are not allowed.

Default value when omitted is ‘1’.

The initial value for interpolation is ‘1’.

drop-shadow() **= drop-shadow(`<color>?` && `<length>{2,3}`)**

Applies a drop shadow effect to the input image. A drop shadow is effectively a blurred, offset version of the input image’s alpha mask drawn in a particular color, composited below the image. Values are interpreted as for ‘box-shadow’ [CSS3BG] but with the optional 3rd <length> value being the standard deviation instead of blur radius. The markup equivalent of this function is given below.

The default value for omitted values is missing length values set to ‘0’ and the missing used color is taken from the color property.

The initial value for interpolation is all length values set to ‘0’ and the used color set to ‘transparent’.

> Note: Spread values or multiple shadows are not accepted for this level of the specification.

> Note: Standard deviation is different to ‘box-shadow’ s blur radius.

grayscale() **= grayscale(`<number-percentage>?`)**

Converts the input image to grayscale. The passed parameter defines the proportion of the conversion. A value of ‘100%’ is completely grayscale. A value of ‘0%’ leaves the input unchanged. Values between ‘0%’ and ‘100%’ are linear multipliers on the effect. Values of amount over ‘100%’ are allowed but UAs must clamp the values to ‘1’. The markup equivalent of this function is given below.

Negative values are not allowed.

Default value when omitted is ‘1’.

The initial value for interpolation is ‘0’.

hue-rotate() `= hue-rotate( [ <angle> | <zero> ]? )`

Applies a hue rotation on the input image. The passed parameter defines the number of degrees around the color circle the input samples will be adjusted. A value of ‘0deg’ leaves the input unchanged. Implementations must not normalize this value in order to allow animations beyond ‘360deg’. The markup equivalent of this function is given below.

The unit identifier may be omitted if the <angle> is zero.

Default value when omitted is ‘0deg’.

The initial value for interpolation is ‘0deg’.

invert() `= invert( <number-percentage>? )`

Inverts the samples in the input image. The passed parameter defines the proportion of the conversion. A value of 100% is completely inverted. A value of ‘0%’ leaves the input unchanged. Values between ‘0%’ and ‘100%’ are linear multipliers on the effect. Values of amount over ‘100%’ are allowed but UAs must clamp the values to ‘1’. The markup equivalent of this function is given below.

Negative values are not allowed.

Default value when omitted is ‘1’.

The initial value for interpolation is ‘0’.

opacity() `= opacity( <number-percentage>? )`

Applies transparency to the samples in the input image. The passed parameter defines the proportion of the conversion. A value of ‘0%’ is completely transparent. A value of ‘100%’ leaves the input unchanged. Values between ‘0%’ and ‘100%’ are linear multipliers on the effect. This is equivalent to multiplying the input image samples by amount. Values of amount over ‘100%’ are allowed but UAs must clamp the values to ‘1’. The markup equivalent of this function is given below.

Negative values are not allowed.

Default value when omitted is ‘1’.

The initial value for interpolation is ‘1’.

> Note: The opacity filter function is not meant to be a shorthand of the ‘opacity’ property. Furthermore, it allows setting the transparency of intermediate filter primitive results before passing to the next filter primitive. If the opacity filter function is set as last filter primitive, the value of the ‘opacity’ property is multiplied on top of the value of the filter function, which may result in a more transparent content.

saturate() `= saturate( <number-percentage>? )`

Saturates the input image. The passed parameter defines the proportion of the conversion. A value of ‘0%’ is completely un-saturated. A value of ‘100%’ leaves the input unchanged. Other values are linear multipliers on the effect. Values of amount over ‘100%’ are allowed, providing super-saturated results. The markup equivalent of this function is given below.

Negative values are not allowed.

Default value when omitted is ‘1’.

The initial value for interpolation is ‘1’.

sepia() `= sepia( <number-percentage>? )`

Converts the input image to sepia. The passed parameter defines the proportion of the conversion. A value of ‘100%’ is completely sepia. A value of ‘0%’ leaves the input unchanged. Values between 0% and 100% are linear multipliers on the effect. Values of amount over ‘100%’ are allowed but UAs must clamp the values to ‘1’. The markup equivalent of this function is given below.

Negative values are not allowed.

Default value when omitted is ‘1’.

The initial value for interpolation is ‘0’.

§ 6.2. Computed Values of Filter Functions

The values in a <filter-function> are computed as specified, with these exceptions:

- Omitted values are included and compute to their defaults.
- <drop-shadow()> starts with the computed value of <color> followed by the computed value of the <length> values.

§ 6.3. Serialization of Filter Functions

To serialize the <filter-function>, serialize as per their individual grammars, in the order the grammars are written in, avoiding <calc()> expressions where possible, serialize filter arguments as specified, avoiding <calc()> transformations, joining space-separated tokens with a single space, and following each serialized comma with a single space.

§ 6.4. Interpolation of Filter Functions

For interpolation of values in <filter-function>s, the steps corresponding to the first matching condition in the following list must be run:

↪ **<blur()>**
  Interpolate values as length by computed value.

↪ **<brightness()>**
↪ **<contrast()>**
↪ **<grayscale()>**
↪ **<invert()>**
↪ **<opacity()>**
↪ **<saturate()>**
↪ **<sepia()>**
  Convert percentage values to numbers with 0% being relative to 0 and 100% relative to 1. Interpolate values as number by computed value .

↪ **<hue-rotate()>**
  Interpolate values as number by computed value.

↪ **<drop-shadow()>**
  Interpolate values as shadow list as repeatable list.

§ 7. SVG Filter Sources: the `<filter>` element

Name:	*`filter`*
Categories:	None.
Content model:	Any number of the following elements, in any order: • descriptive — `<desc>`, `<title>`, `<metadata>` • filter primitive — `<feBlend>`, `<feFlood>`, `<feColorMatrix>`, `<feComponentTransfer>`, `<feComposite>`, `<feConvolveMatrix>`, `<feDiffuseLighting>`, `<feDisplacementMap>`, `<feDropShadow>`, `<feGaussianBlur>`, `<feImage>`, `<feMerge>`, `<feMorphology>`, `<feOffset>`, `<feSpecularLighting>`, `<feTile>`, `<feTurbulence>` • `<animate>` • `<script>` • `<set>`
Attributes:	• core attributes — id, xml:base, xml:lang, xml:space

- presentation attributes — ‘alignment-baseline’, ‘baseline-shift’, ‘clip’, ‘clip-path’, ‘clip-rule’, ‘color’, ‘color-interpolation’, ‘color-interpolation-filters’, ‘color-rendering’, ‘cursor’, ‘direction’, ‘display’, ‘dominant-baseline’, ‘enable-background’, ‘fill’, ‘fill-opacity’, ‘fill-rule’, ‘filter’, ‘flood-color’, ‘flood-opacity’, ‘font’, ‘font-family’, ‘font-size’, ‘font-size-adjust’, ‘font-stretch’, ‘font-style’, ‘font-variant’, ‘font-weight’, ‘glyph-orientation-horizontal’, ‘glyph-orientation-vertical’, ‘image-rendering’, ‘isolation’, ‘kerning’, ‘letter-spacing’, ‘lighting-color’, ‘marker’, ‘marker-end’, ‘marker-mid’, ‘marker-start’, ‘mask’, ‘opacity’, ‘overflow’, ‘pointer-events’, ‘shape-rendering’, ‘stop-color’, ‘stop-opacity’, ‘stroke’, ‘stroke-dasharray’, ‘stroke-dashoffset’, ‘stroke-linecap’, ‘stroke-linejoin’, ‘stroke-miterlimit’, ‘stroke-opacity’, ‘stroke-width’, ‘text-anchor’, ‘text-decoration’, ‘text-rendering’, ‘unicode-bidi’, ‘visibility’, ‘word-spacing’, ‘writing-mode’
- class
- style
- externalResourcesRequired
- `x`
- `y`
- `width`
- `height`
- `filterUnits`
- `primitiveUnits`

DOM Interfaces: SVGFilterElement

The description of the `<filter>` element follows:

‘<number-optional-number>’ = <number> <number>?

Attribute definitions:

`filterUnits` = "***`userSpaceOnUse`*** | ***`objectBoundingBox`***"
: See filter 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` is equal to userSpaceOnUse, any length values within the filter definitions represent values in the current local 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` is equal to 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). Note that if only one number was specified in a <number-optional-number> value this number is expanded out before the primitiveUnits computation takes place.

The initial value for `primitiveUnits` is userSpaceOnUse.

Animatable: yes.

`x` = "**<length-percentage>**"
: See filter region.

`y` = "**<length-percentage>**"
: See filter region.

`width` = "**<length-percentage>**"
: See filter region.

`height` = "**<length-percentage>**"
: See filter region.

`filterRes` = "**<number-optional-number>**"
: The *filterRes* attribute was removed from the specification. See SVG 1.1 specification for the defintion [SVG11].

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’.

§ 8. Filter Region

A `<filter>` element can define a ***filter 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 region:

`filterUnits`

Defines the coordinate system for attributes `x`, `y`, `width`, `height`.

If `filterUnits` is equal to 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` is equal to objectBoundingBox, then `x`, `y`, `width`, `height` represent fractions or percentages of the bounding box on the referencing element (see object bounding box units).

The initial value for `filterUnits` is objectBoundingBox.

Animatable: yes.

`x`, `y`, `width`, `height`

These attributes define a rectangular region on the canvas to which this filter applies.

The coordinate system for these attributes depends on the value for attribute `filterUnits`.

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.

The initial value for `x` and `y` is ‘-10%’.

The initial value for `width` and `height` is ‘120%’.

ng of the element which referenced the filter.

Animatable: yes.

Note: 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 local coordinate system for the element to which the filter will be applied.

Note: 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 the 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.

Note: 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 region are `x="-10%" y="-10%" width="120%" height="120%"`.

§ 9. Filter primitives

§ 9.1. Overview

This section describes the various filter primitives that can be assembled to achieve a particular filter effect.

Unless otherwise stated, all image filters operate on premultiplied RGBA samples. Some filters like `<feColorMatrix>` and `<feComponentTransfer>` work more naturally on non-premultiplied data. For the time of the filter operation, all color values must temporarily be transformed to the required color multiplication of the current filter.

> Note: All input images are assumed to be in premultiplied RGBA. User agents may optimize performance by using non-premultiplied data buffering.

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 the 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.

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.

To provide high quality rendering, all filter primitives should operate in a device dependent coordinate space, the ***operating coordinate space***, taking device pixel density, user space transformations and zooming into account. To provide a platform independent alignment, attribute and property values are often relative to a coordinate system described by the `primitiveUnits` attribute. User agents must scale these relative attributes and properties to the operating coordinate space.

> Note: On high resolution devices, attribute and property values that are relative to the `primitiveUnits` usually need to be scaled up. User agents may reduce the resolution of filter primitives on limited platform resources.

> Note: Some attribute or property values from the filter primitives `<feConvolveMatrix>` and light sources can not be mapped from the coordinate space defined by the `primitiveUnits` attribute to the operating coordinate space.

§ 9.2. Common filter primitive attributes

The following ***filter primitive attributes*** are available for all filter primitives:

Attribute definitions:

x **= "<length-percentage>"**

The minimum x coordinate for the subregion which restricts calculation and rendering of the given filter primitive. See filter primitive subregion.

The initial value for x is '0%'.

Animatable: yes.

y **= "<length-percentage>"**

The minimum y coordinate for the subregion which restricts calculation and rendering of the given filter primitive. See filter primitive subregion.

The initial value for y is '0%'.

Animatable: yes.

`width` = "**<length-percentage>**"

The width of the subregion which restricts calculation and rendering of the given filter primitive. See filter primitive subregion.

A negative or zero value disables the effect of the given filter primitive (i.e., the result is a transparent black image).

The initial value for `width` is '100%'.

Animatable: yes.

`height` = "**<length-percentage>**"

The height of the subregion which restricts calculation and rendering of the given filter primitive. See filter primitive subregion.

A negative or zero value must disable the effect of the given filter primitive (i.e., the result is a transparent black image).

The initial value for `height` is '100%'.

Animatable: yes.

`result` = "***'<filter-primitive-reference>'***"

<filter-primitive-reference> is an <custom-ident> [CSS3VAL] and an 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.

Most filter primitives take other filter primitives as input. The following attribute is representative for all input attributes to reference other filter primitives:

Attribute definitions:

`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 not allowed, and will be treated as if no result was specified.

References to non-existent results will be treated as if no result was specified.

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.

> Note: 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 the back drop defined by the current isolation group behind the filter region at the time that the `<filter>` element was invoked. See ‘isolation’ property [COMPOSITING-1].

`BackgroundAlpha`

Same as BackgroundImage except only the alpha channel is used. See SourceAlpha and the ‘isolation’ property [COMPOSITING-1].

`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. If ‘fill’ references a paint server, then the coordinate space of the paint server is the coordinate space defined for the filtered object. E.g if the paint server requires to use the objectBoundingBox of the object, the object bounding box of the filtered object defines the reference size of the paint server. If the paint server requires to use the userSpaceOnUse, the nearest viewport in the local coordinate system of the filtered object defines the reference size of the paint server.

`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. See FillPaint above for more details.

Animatable: yes.

§ 9.3. Filter primitive tree

Filter primitives with no or one filter primitive input can be linked together to a filter chain. E.g. the filter primitive representation of a <filter-value-list> with two or more <filter-function>s is an example of a filter chain. Every filter primitive takes the result of the previous filter primitive as input.

EXAMPLE 3

A simple example of a `<filter>` element with its filter primitive children.

```
<filter id="filter">
  <feColorMatrix type="hueRotate" values="45"/>
  <feOffset dx="10" dy="10"/>
  <feGaussianBlur stdDeviation="3"/>
</filter>
```

`<feColorMatrix>`, `<feOffset>` and `<feGaussianBlur>` create a filter chain.

`<feColorMatrix>` takes ‘SourceGraphic’ as input. The result is the input of `<feOffset>` with its result being the input of `<feGaussianBlur>`.

Some filter primitives may have more than one filter primitive inputs. With the use of the `in` and `result` attributes it is possible to combine multiple filter primitives to a complex filter structure. Due to the non-forward reference restriction of filter primitives, every filter structure can be represented as a tree, the ***filter primitive tree***. The root filter primitive of the filter primitive tree is the most subsequential primitive of `<filter>` elements filter primitive children.

A filter chain is one possible filter structure that can also be represented in a filter primitive tree. Therefore, filter chains are referred to as filter primitive trees onwards as well.

A `<filter>` element may have one or more filter primitive trees. The filter primitive tree whose subsequent filter primitive is the last filter primitive child of the filter elements is the ***primary filter primitive tree***.

Only the primary filter primitive tree contributes to the filter process. Implementations may chose to ignore all other possible filter primitive trees.

If a `<filter>` element has no filter primitive tree then the element the filter applies to does not get rendered.

EXAMPLE 4
An example of multiple filter primitive trees:

```
<filter id="filter">
  <-- The first filter primitive tree. Ignored for filter process. -->
  <feColorMatrix type="hueRotate" values="45"/>
  <feOffset dx="10" dy="10"/>
  <feGaussianBlur stdDeviation="3"/>
  <-- The primary filter primitive tree. -->
  <feFlood flood-color="green" result="flood"/>
  <feComposite operator="in" in="SourceAlpha" in2="flood"/>
</filter>
```

The above filter has 2 filter primitive trees with the filter primitives:

1. `<feColorMatrix>`, `<feOffset>` and `<feGaussianBlur>` (with feGaussianBlur being the root filter primitive of the tree) as well as
2. `<feFlood>` and `<feComposite>` (with feComposite as the root filter primitive of the tree).

Both filter primitive trees are not connected. Only the 2nd, the primary filter primitive tree contributes to the filter process. The first tree can get ignored by implementations.

§ 9.4. Filter primitive subregion

All filter primitives have attributes `x`, `y`, `width` and `height` which together identify a ***filter primitive subregion*** which restricts calculation and rendering of the given filter primitive. The x, y, width and height 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 as a special-case the percentages are relative to the dimensions of the filter region, thus making the default filter primitive subregion equal to the filter region.

If the filter primitive subregion has a negative or zero width or height, the effect of the filter primitive is disabled.

The filter region acts as a hard clip clipping rectangle on the filter primitive’s input image(s).

The filter primitive subregion acts as a hard clip clipping rectangle on the filter primitive result.

All intermediate offscreens are defined to not exceed the intersection of the filter primitive subregion with the filter region. The filter region and any of the filter primitive 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 filter primitive subregions.

EXAMPLE 5

`<feTile>` references a previous filter primitive and then stitches the tiles together based on the filter primitive subregion of the referenced filter primitive in order to fill its own filter primitive subregion.

```
<svg width="400" height="400" xmlns="http://www.w3.org/2000/svg">
  <defs>
    <filter id="flood" x="0" y="0" width="100%" height="100%" primitiveUnits="objectBoundingBox"
       <feFlood x="25%" y="25%" width="50%" height="50%"
          flood-color="green" flood-opacity="0.75"/>
    </filter>
    <filter id="blend" primitiveUnits="objectBoundingBox">
       <feBlend x="25%" y="25%" width="50%" height="50%"
          in2="SourceGraphic" mode="multiply"/>
    </filter>
    <filter id="merge" primitiveUnits="objectBoundingBox">
       <feMerge x="25%" y="25%" width="50%" height="50%">
        <feMergeNode in="SourceGraphic"/>
        <feMergeNode in="FillPaint"/>
       </feMerge>
    </filter>
  </defs>

  <g fill="none" stroke="blue" stroke-width="4">
     <rect width="200" height="200"/>
     <line x2="200" y2="200"/>
     <line x1="200" y2="200"/>
  </g>
  <circle fill="green" filter="url(#flood)" cx="100" cy="100" r="90"/>

  <g transform="translate(200 0)">
    <g fill="none" stroke="blue" stroke-width="4">
       <rect width="200" height="200"/>
       <line x2="200" y2="200"/>
       <line x1="200" y2="200"/>
    </g>
    <circle fill="green" filter="url(#blend)" cx="100" cy="100" r="90"/>
  </g>

  <g transform="translate(0 200)">
    <g fill="none" stroke="blue" stroke-width="4">
       <rect width="200" height="200"/>
       <line x2="200" y2="200"/>
       <line x1="200" y2="200"/>
    </g>
    <circle fill="green" fill-opacity="0.5" filter="url(#merge)" cx="100" cy="100" r="90"/>
  </g>
</svg>
```

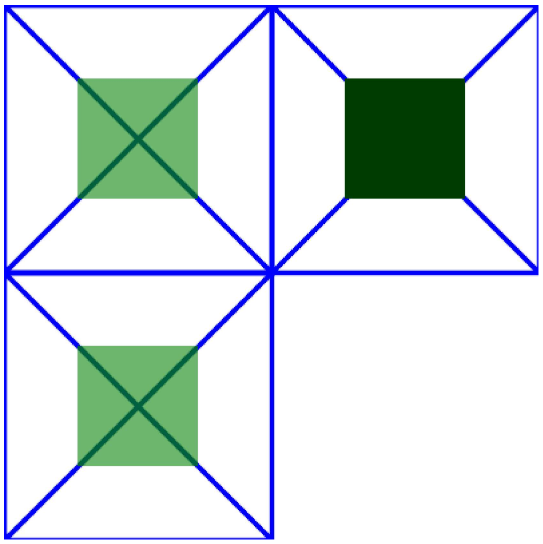


Figure 3. *Example for subregions*

View this example as SVG

In the example above there are three rectangles that each have a cross and a circle in them. The circle element in each one has a different filter applied, but with the same filter primitive subregion. The filter output should be limited to the filter primitive subregion so you should never see the circles themselves, just the rectangles that make up the filter primitive subregion.

- The upper left rectangle shows an `<feFlood>` with ‘flood-opacity: 75%’ so the cross should be visible through the green rect in the middle.
- The lower left rectangle shows an `<feMerge>` that merges SourceGraphic with FillPaint. Since the circle has `fill-opacity="0.5"` it will also be transparent so that the cross is visible through the green rect in the middle.
- The upper right rectangle shows an `<feBlend>` that has `mode="multiply"`. Since the circle in this case isn’t transparent the result is totally opaque. The rect should be dark green and the cross should not be visible through it.

§ 9.5. Filter primitive `<feBlend>`

Name: ***`feBlend`***

Categories: filter primitive

Content model: Any number of descriptive elements, `<animate>`, `<script>`, `<set>` elements, in any order.

Attributes:

- core attributes — id, xml:base, xml:lang, xml:space
- presentation attributes — ‘alignment-baseline’, ‘baseline-shift’, ‘clip’, ‘clip-path’, ‘clip-rule’, ‘color’, ‘color-interpolation’, ‘color-interpolation-filters’, ‘color-rendering’, ‘cursor’, ‘direction’, ‘display’, ‘dominant-baseline’, ‘enable-background’, ‘fill’, ‘fill-opacity’, ‘fill-rule’, ‘filter’, ‘flood-color’, ‘flood-opacity’, ‘font’, ‘font-family’, ‘font-size’, ‘font-size-adjust’, ‘font-stretch’, ‘font-style’, ‘font-variant’, ‘font-weight’, ‘glyph-orientation-horizontal’, ‘glyph-orientation-vertical’, ‘image-rendering’, ‘isolation’, ‘kerning’, ‘letter-spacing’, ‘lighting-color’, ‘marker’, ‘marker-end’, ‘marker-mid’, ‘marker-start’, ‘mask’, ‘opacity’, ‘overflow’, ‘pointer-events’, ‘shape-rendering’, ‘stop-color’, ‘stop-opacity’, ‘stroke’, ‘stroke-dasharray’, ‘stroke-dashoffset’, ‘stroke-linecap’, ‘stroke-linejoin’, ‘stroke-miterlimit’, ‘stroke-opacity’, ‘stroke-width’, ‘text-anchor’, ‘text-decoration’, ‘text-rendering’, ‘unicode-bidi’, ‘visibility’, ‘word-spacing’, ‘writing-mode’
- filter primitive attributes —`x`, `y`, `width`, `height`, `result`
- class
- style

- `in`
- `in2`
- `mode`

DOM Interfaces: SVGFEBlendElement

This filter blends two objects together using commonly used imaging software blending modes. It performs a pixel-wise combination of two input images. (See [COMPOSITING-1].)

Attribute definitions:

`mode` **= "<blend-mode>"**

One of the blend modes defined by “Compositing and Blending Level 1” [COMPOSITING-1] with the input `in` representing the source `Cs` and the second input `in2` representing the backdrop `Cb`. The output of this filter primitive `Cm` is the result of blending `Cs` with `Cb`.

The initial value for `mode` is ‘normal’.

Animatable: yes.

`no-composite` **= "*`no-composite`*"**

If the `no-composite` attribute is present, the specified blend mode must not apply alpha compositing. See Blending [COMPOSITING-1] for the "mixing" formula without compositing. Otherwise, implementations must combine the blend mode specified by `mode` with the Source Over composite operator. See Blending [COMPOSITING-1] for the "mixing" formula with compositing.

> Note: This attribute is an addition to the `<feBlend>` element defintion in SVG 1.1. `no-composite`, when specified, is meant to avoid "double-compositing" effects when blending an input source with the backdrop of the filtered object (E.g. using the BackgroundImage filter primitive). For the majority of use cases authors will not need to specify the no-composite attribute.

Animatable: no.

`in2` **= "*(see `in` attribute)*"**

The second input image to the blending operation.

Animatable: yes.

The ‘normal’ blend mode with alpha compositing 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 6

```
<svg width="5cm" height="5cm" viewBox="0 0 500 500"
     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 isolation="isolate" >
    <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>
```

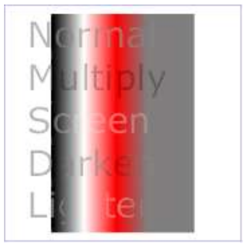


Figure 4. *Example of feBlend*

View this example as SVG

§ 9.6. Filter primitive `<feColorMatrix>`

Name:	*`feColorMatrix`*
Categories:	filter primitive
Content model:	Any number of descriptive elements, `<animate>`, `<script>`, `<set>` elements, in any order.
Attributes:	• core attributes — id, xml:base, xml:lang, xml:space • presentation attributes — ‘alignment-baseline’, ‘baseline-shift’, ‘clip’, ‘clip-path’, ‘clip-rule’, ‘color’, ‘color-interpolation’, ‘color-interpolation-filters’, ‘color-rendering’, ‘cursor’, ‘direction’, ‘display’, ‘dominant-baseline’, ‘enable-background’, ‘fill’, ‘fill-opacity’, ‘fill-rule’, ‘filter’, ‘flood-color’, ‘flood-opacity’, ‘font’, ‘font-family’, ‘font-size’, ‘font-size-adjust’, ‘font-stretch’, ‘font-style’, ‘font-variant’, ‘font-weight’, ‘glyph-orientation-horizontal’, ‘glyph-orientation-vertical’, ‘image-rendering’, ‘isolation’, ‘kerning’, ‘letter-spacing’, ‘lighting-color’, ‘marker’, ‘marker-end’, ‘marker-mid’, ‘marker-start’, ‘mask’, ‘opacity’, ‘overflow’, ‘pointer-events’, ‘shape-rendering’, ‘stop-color’, ‘stop-opacity’, ‘stroke’, ‘stroke-dasharray’, ‘stroke-dashoffset’, ‘stroke-linecap’, ‘stroke-linejoin’, ‘stroke-miterlimit’, ‘stroke-opacity’, ‘stroke-width’, ‘text-anchor’, ‘text-decoration’, ‘text-rendering’, ‘unicode-bidi’, ‘visibility’, ‘word-spacing’, ‘writing-mode’ • filter primitive attributes —`x`, `y`, `width`, `height`, `result` • class • style • `in` • `type` • `values`
DOM Interfaces:	SVGFEColorMatrixElement

This filter applies a matrix transformation:

$$\begin{bmatrix} R' \\ G' \\ B' \\ A' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \\ A \\ 1 \end{bmatrix}$$

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.

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.

The initial value for `type` is 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. A saturate operation is equivalent to the following matrix operation:

$$\begin{bmatrix} R' \\ G' \\ B' \\ A' \\ 1 \end{bmatrix} = \begin{bmatrix} 0.213+0.787s & 0.715-0.715s & 0.072-0.072s & 0 & 0 \\ 0.213-0.213s & 0.715+0.285s & 0.072-0.072s & 0 & 0 \\ 0.213-0.213s & 0.715-0.715s & 0.072+0.928s & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \\ A \\ 1 \end{bmatrix}$$

> Note: A value of '0' produces a fully desaturated (grayscale) filter result, while a value of '1' passes the filter input image through unchanged. Values outside the 0..1 range under- or oversaturates the filter input image respectively.

> Note: The precision of the luminance coefficients increased in comparison to previous specification texts [Cmam].

- For `type="hueRotate"`, `values` is a single one real number value (degrees). A hueRotate operation is equivalent to the following matrix operation:

$$\begin{bmatrix} R' \\ G' \\ B' \\ A' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & 0 & 0 \\ a_{10} & a_{11} & a_{12} & 0 & 0 \\ a_{20} & a_{21} & a_{22} & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \\ A \\ 1 \end{bmatrix}$$

where the terms a00, a01, etc. are calculated as follows:

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} +0.213 & +0.715 & +0.072 \\ +0.213 & +0.715 & +0.072 \\ +0.213 & +0.715 & +0.072 \end{bmatrix} + \cos\left(\mathit{hueRotate\ value}\right) \cdot \begin{bmatrix} +0.787 & -0.715 & -0.072 \\ -0.213 & +0.285 & -0.072 \\ -0.213 & -0.715 & +0.928 \end{bmatrix} + \sin\left(\mathit{hueRotate}\right.$$

Thus, the upper left term of the hue matrix turns out to be:

$$a_{00} = 0.2127 + \cos\left(\mathit{hueRotate\ value}\right) \cdot 0.7873 - \sin\left(\mathit{hueRotate\ value}\right) \cdot 0.2127$$

- For `type="luminanceToAlpha"`, `values` is not applicable. A luminanceToAlpha operation is equivalent to the following matrix operation:

$$\begin{bmatrix} R' \\ G' \\ B' \\ A' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0.2126 & 0.7152 & 0.0722 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \\ A \\ 1 \end{bmatrix}$$

The initial value for `values`

↪ **if `type="matrix"`**
defaults to the identity matrix

↪ **if `type="saturate"`**
defaults to the value '1'

↪ **if `type="hueRotate"`**
defaults to the value '0' which results in the identity matrix.

If the number of entries in the `values` list does not match the required number of entries by the `type`, the filter primitive acts as a pass through filter.

Animatable: yes.

EXAMPLE 7

```
<svg width="8cm" height="5cm" viewBox="0 0 800 500"
     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 to convert to grayscale,
        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>
```

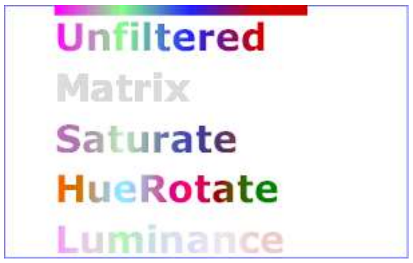


Figure 5. *Example of feColorMatrix*

View this example as SVG

§ 9.7. Filter primitive `<feComponentTransfer>`

Name:	***`feComponentTransfer`***
Categories:	filter primitive
Content model:	Any number of descriptive elements, `<feFuncR>`, `<feFuncG>`, `<feFuncB>`, `<feFuncA>`, `<script>` elements, in any order.
Attributes:	• core attributes — id, xml:base, xml:lang, xml:space • presentation attributes — ‘alignment-baseline’, ‘baseline-shift’, ‘clip’, ‘clip-path’, ‘clip-rule’, ‘color’, ‘color-interpolation’, ‘color-interpolation-filters’, ‘color-rendering’, ‘cursor’, ‘direction’, ‘display’, ‘dominant-baseline’, ‘enable-background’, ‘fill’, ‘fill-opacity’, ‘fill-rule’, ‘filter’, ‘flood-color’, ‘flood-opacity’, ‘font’, ‘font-family’, ‘font-size’, ‘font-size-adjust’, ‘font-stretch’, ‘font-style’, ‘font-variant’, ‘font-weight’, ‘glyph-orientation-horizontal’, ‘glyph-orientation-vertical’, ‘image-rendering’, ‘isolation’, ‘kerning’, ‘letter-spacing’, ‘lighting-color’, ‘marker’, ‘marker-end’, ‘marker-mid’, ‘marker-start’, ‘mask’, ‘opacity’, ‘overflow’, ‘pointer-events’, ‘shape-rendering’, ‘stop-color’, ‘stop-opacity’, ‘stroke’, ‘stroke-dasharray’, ‘stroke-dashoffset’, ‘stroke-linecap’, ‘stroke-linejoin’, ‘stroke-miterlimit’, ‘stroke-opacity’, ‘stroke-width’, ‘text-anchor’, ‘text-decoration’, ‘text-rendering’, ‘unicode-bidi’, ‘visibility’, ‘word-spacing’, ‘writing-mode’ • filter primitive attributes —`x`, `y`, `width`, `height`, `result` • class • style • `in`
DOM Interfaces:	SVGFEComponentTransferElement

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.

The child elements of a `<feComponentTransfer>` element specify the transfer functions for the four channels:

- `<feFuncR>` - transfer function for the red component of the input graphic
- `<feFuncG>` - transfer function for the green component of the input graphic
- `<feFuncB>` - transfer function for the blue component of the input graphic
- `<feFuncA>` - transfer function for the alpha component of the input graphic

The set of `<feFuncR>`, `<feFuncG>`, `<feFuncB>`, `<feFuncA>` elements are also called ***transfer function element***s.

The following rules apply to the processing of the `<feComponentTransfer>` element:

- If more than one transfer function element of the same kind is specified, the last occurrence is to be used.
- If any of the transfer function elements are unspecified, the `<feComponentTransfer>` must be processed as if those transfer function elements were specified with their `type` attributes set to ‘identity’.

§ 9.7.1. Transfer function `<feFuncR>`

Name:	***`feFuncR`***
Categories:	transfer function element
Content model:	Any number of descriptive elements, `<animate>`, `<script>`, `<set>` elements, in any order.
Attributes:	• core attributes — id, xml:base, xml:lang, xml:space • transfer function element attributes — `type`, `tableValues`, `slope`, `intercept`, `amplitude`, `exponent`, `offset`
DOM Interfaces:	SVGFEFuncRElement

The attributes below are the ***transfer function element attributes***, which apply to the transfer function elements.

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.

In the following, C is the initial component (e.g., `<feFuncR>`), C' is the remapped component; both in the closed interval [0,1].

- For ***`identity`***:

  `C' = C`

- For ***`table`***, the function is defined by linear interpolation between values given in the attribute `tableValues`. The table has *n+1* values (i.e., v_0 to v_n) specifying the start and end values for *n* evenly sized interpolation regions. Interpolations use the following formula:

  For a value `C < 1` find `k` such that:

  `k/n <= C < (k+1)/n`

  The result `C'` is given by:

  $C' = v_k + (C - k/n)*n * (v_{k+1} - v_k)$

  If `C = 1` then:

  $C' = v_n.$

- For ***`discrete`***, the function is defined by the step function given in the 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 < 1` find `k` such that:

  `k/n <= C < (k+1)/n`

  The result `C'` is given by:

  $C' = v_k$

If `C = 1` then:

`C' = v`$_{n-1}$`.`

- 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`

The initial value for `type` is identity.

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.

Animatable: yes.

`slope` = "***<number>***"

When `type="linear"`, the slope of the linear function.

The initial value for `slope` is ‘1’.

Animatable: yes.

`intercept` = "***<number>***"

When `type="linear"`, the intercept of the linear function.

The initial value for `intercept` is ‘0’.

Animatable: yes.

`amplitude` = "***<number>***"

When `type="gamma"`, the amplitude of the gamma function.

The initial value for `amplitude` is ‘1’.

Animatable: yes.

`exponent` = "***<number>***"

When `type="gamma"`, the exponent of the gamma function.

The initial value for `exponent` is ‘1’.

Animatable: yes.

`offset` = "***<number>***"

When `type="gamma"`, the offset of the gamma function.

The initial value for `offset` is ‘0’.

Animatable: yes.

EXAMPLE 8

```
<svg width="8cm" height="4cm" viewBox="0 0 800 400"
     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>
```

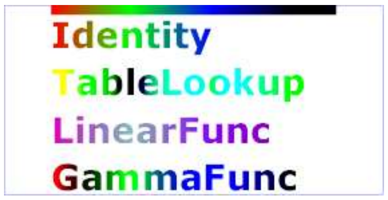


Figure 6. *Example for feComponentTransfer*

View this example as SVG

§ 9.7.2. Transfer function `<feFuncG>`

Name:	*`feFuncG`*
Categories:	transfer function element
Content model:	Any number of descriptive elements, `<animate>`, `<script>`, `<set>` elements, in any order.
Attributes:	• core attributes — id, xml:base, xml:lang, xml:space • transfer function element attributes — `type`, `tableValues`, `slope`, `intercept`, `amplitude`, `exponent`, `offset`
DOM Interfaces:	SVGFEFuncGElement

See `<feFuncR>` for the definitions of the attribute values.

§ 9.7.3. Transfer function `<feFuncB>`

Name:	*`feFuncB`*
Categories:	transfer function element
Content model:	Any number of descriptive elements, `<animate>`, `<script>`, `<set>` elements, in any order.
Attributes:	• core attributes — id, xml:base, xml:lang, xml:space • transfer function element attributes — `type`, `tableValues`, `slope`, `intercept`, `amplitude`, `exponent`, `offset`
DOM Interfaces:	SVGFEFuncBElement

See `<feFuncR>` for the definitions of the attribute values.

§ 9.7.4. Transfer function `<feFuncA>`

Name:	*`feFuncA`*
Categories:	transfer function element
Content model:	Any number of descriptive elements, `<animate>`, `<script>`, `<set>` elements, in any order.
Attributes:	• core attributes — id, xml:base, xml:lang, xml:space • transfer function element attributes — `type`, `tableValues`, `slope`, `intercept`, `amplitude`, `exponent`, `offset`
DOM Interfaces:	SVGFEFuncAElement

See `<feFuncR>` for the definitions of the attribute values.

§ 9.8. Filter primitive `<feComposite>`

Name:	***feComposite***
Categories:	filter primitive
Content model:	Any number of descriptive elements, `<animate>`, `<script>`, `<set>` elements, in any order.
Attributes:	• core attributes — id, xml:base, xml:lang, xml:space • presentation attributes — ‘alignment-baseline’, ‘baseline-shift’, ‘clip’, ‘clip-path’, ‘clip-rule’, ‘color’, ‘color-interpolation’, ‘color-interpolation-filters’, ‘color-rendering’, ‘cursor’, ‘direction’, ‘display’, ‘dominant-baseline’, ‘enable-background’, ‘fill’, ‘fill-opacity’, ‘fill-rule’, ‘filter’, ‘flood-color’, ‘flood-opacity’, ‘font’, ‘font-family’, ‘font-size’, ‘font-size-adjust’, ‘font-stretch’, ‘font-style’, ‘font-variant’, ‘font-weight’, ‘glyph-orientation-horizontal’, ‘glyph-orientation-vertical’, ‘image-rendering’, ‘isolation’, ‘kerning’, ‘letter-spacing’, ‘lighting-color’, ‘marker’, ‘marker-end’, ‘marker-mid’, ‘marker-start’, ‘mask’, ‘opacity’, ‘overflow’, ‘pointer-events’, ‘shape-rendering’, ‘stop-color’, ‘stop-opacity’, ‘stroke’, ‘stroke-dasharray’, ‘stroke-dashoffset’, ‘stroke-linecap’, ‘stroke-linejoin’, ‘stroke-miterlimit’, ‘stroke-opacity’, ‘stroke-width’, ‘text-anchor’, ‘text-decoration’, ‘text-rendering’, ‘unicode-bidi’, ‘visibility’, ‘word-spacing’, ‘writing-mode’ • filter primitive attributes —`x`, `y`, `width`, `height`, `result` • class • style • `in` • `in2` • `operator` • `k1` • `k2` • `k3` • `k4`
DOM Interfaces:	SVGFECompositeElement

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’, ‘lighter’ [COMPOSITING-1]. 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
```

where:

- `i1` and `i2` indicate the corresponding pixel channel values of the input image, which map to `in` and `in2` respectively
- `k1, k2, k3` and `k4` indicate the values of the attributes with the same name

For this filter primitive, the extent of the resulting image might grow as described in the section that describes the filter primitive subregion.

Attribute definitions:

***operator* = "*over* | *in* | *out* | *atop* | *xor* | *lighter* | *arithmetic*"**

The compositing operation that is to be performed. All of the `operator` types except ‘arithmetic’ match the corresponding operation as described in [COMPOSITING-1] with `in` representing the source and `in2` representing the destination. The ‘arithmetic’ operator is described above.

The initial value for `operator` is over.

Animatable: yes.

k1 = "**<number>**"
Only applicable if `operator="arithmetic"`.

The initial value for `k1` is ‘0’.

Animatable: yes.

k2 = "***<number>***"
Only applicable if `operator="arithmetic"`.

The initial value for `k2` is ‘0’.

Animatable: yes.

k3 = "**<number>**"
Only applicable if `operator="arithmetic"`.

The initial value for `k3` is ‘0’.

Animatable: yes.

k4 = "**<number>**"
Only applicable if `operator="arithmetic"`.

The initial value for `k4` is ‘0’.

Animatable: yes.

in2 = "***(see `in` attribute)***"
The second input image to the compositing operation.

Animatable: yes.

Note: Compositing and Blending [COMPOSITING-1] defines more compositing keywords. The functionality of the additional keywords can be archived by switching the input filter primitives `in` and `in2`.

EXAMPLE 9

```
<svg width="330" height="195" viewBox="0 0 1100 650"
     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="1
      <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="11
      <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="1
      <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="1
      <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="1
      <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=
      <feComposite in="SourceGraphic" in2="BackgroundImage" operator="over" result="comp"/>
    </filter>
    <filter id="inNoFlood" filterUnits="objectBoundingBox" x="-5%" y="-5%" width="110%" height="1
      <feComposite in="SourceGraphic" in2="BackgroundImage" operator="in" result="comp"/>
    </filter>
    <filter id="outNoFlood" filterUnits="objectBoundingBox" x="-5%" y="-5%" width="110%" height="
      <feComposite in="SourceGraphic" in2="BackgroundImage" operator="out" result="comp"/>
    </filter>
    <filter id="atopNoFlood" filterUnits="objectBoundingBox" x="-5%" y="-5%" width="110%" height=
      <feComposite in="SourceGraphic" in2="BackgroundImage" operator="atop" result="comp"/>
    </filter>
    <filter id="xorNoFlood" filterUnits="objectBoundingBox" x="-5%" y="-5%" width="110%" height="
      <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 isolation="isolate">
      <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)" isolation="isolate">
    <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>
```

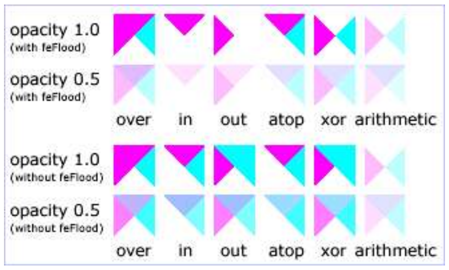


Figure 7. *Example of feComposite*

View this example as SVG

§ 9.9. Filter primitive `<feConvolveMatrix>`

Name: ***feConvolveMatrix***

Categories: filter primitive

Content model: Any number of descriptive elements, `<animate>`, `<script>`, `<set>` elements, in any order.

Attributes:

- core attributes — id, xml:base, xml:lang, xml:space
- presentation attributes — ‘alignment-baseline’, ‘baseline-shift’, ‘clip’, ‘clip-path’, ‘clip-rule’, ‘color’, ‘color-interpolation’, ‘color-interpolation-filters’, ‘color-rendering’, ‘cursor’, ‘direction’, ‘display’, ‘dominant-baseline’, ‘enable-background’, ‘fill’, ‘fill-opacity’, ‘fill-rule’, ‘filter’, ‘flood-color’, ‘flood-opacity’, ‘font’, ‘font-family’, ‘font-size’, ‘font-size-adjust’, ‘font-stretch’, ‘font-style’, ‘font-variant’, ‘font-weight’, ‘glyph-orientation-horizontal’, ‘glyph-orientation-vertical’, ‘image-rendering’, ‘isolation’, ‘kerning’, ‘letter-spacing’, ‘lighting-color’, ‘marker’, ‘marker-end’, ‘marker-mid’, ‘marker-start’, ‘mask’, ‘opacity’, ‘overflow’, ‘pointer-events’, ‘shape-rendering’, ‘stop-color’, ‘stop-opacity’, ‘stroke’, ‘stroke-dasharray’, ‘stroke-dashoffset’, ‘stroke-linecap’, ‘stroke-linejoin’, ‘stroke-miterlimit’, ‘stroke-opacity’, ‘stroke-width’, ‘text-anchor’, ‘text-decoration’, ‘text-rendering’, ‘unicode-bidi’, ‘visibility’, ‘word-spacing’, ‘writing-mode’
- filter primitive attributes —`x`, `y`, `width`, `height`, `result`
- class
- style
- `in`
- `order`
- `kernelMatrix`
- `divisor`
- `bias`
- `targetX`
- `targetY`
- `edgeMode`
- `kernelUnitLength`
- `preserveAlpha`

DOM Interfaces: SVGFEConvolveMatrixElement

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:

$$\mathbf{color}_{X,Y} = \frac{\sum_{i=0}^{\mathrm{orderY}-1} \sum_{j=0}^{\mathrm{orderX}-1} \mathbf{source}_{x-\mathrm{targetX}+j,\, y-targetY+i} \cdot \mathbf{kernalMatrix}_{\mathrm{orderX}-j-1,\, \mathrm{orderY}-i-1}}{\mathbf{divisor}} + \mathbf{bias} \cdot \mathbf{alpha}_{x,y}$$

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.

In the above formulas 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:

$$\begin{bmatrix} 0 & 20 & 40 & 235 & 235 \\ 100 & 120 & 140 & 235 & 235 \\ 200 & 220 & 240 & 235 & 235 \\ 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 255 \end{bmatrix}$$

and you define a 3-by-3 convolution kernel as follows:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

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:

$$\text{resultChannel}_{2,2} = \frac{9\cdot 0+8\cdot 20+7\cdot 40+6\cdot 100+5\cdot 120+4\cdot 140+3\cdot 200+2\cdot 220+1\cdot 240}{9+8+7+6+5+4+3+2+1}$$

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 the 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). 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.

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. Values that are not integers will be truncated, i.e. rounded to the closest integer value towards 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>.

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.

The initial value for `order` is ‘3’.

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>.

If the result of `orderX * orderY` is not equal to the the number of entries in the value list, the filter primitive acts as a pass through filter.

ISSUE 2 How to behave on invalid number of entries in the value list? <https://github.com/w3c/csswg-drafts/issues/237>

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. If the specified divisor is ‘0’ then the default value will be used instead.

The initial 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. The bias property shifts the range of the filter. This allows representation of values that would otherwise be clamped to 0 or 1.

The initial value for `bias` is ‘0’.

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:

$$\begin{bmatrix} 11 & 12 & ... & 1m & 1M \\ 21 & 22 & ... & 2m & 2M \\ ... & ... & ... & ... & ... \\ n1 & n2 & ... & \mathrm{nm} & \mathrm{nM} \\ N1 & N2 & ... & \mathrm{Nm} & \mathrm{NM} \end{bmatrix}$$

Extended by two pixels using ‘duplicate’:

$$\begin{bmatrix} 11 & 11 & 11 & 12 & \ldots & 1m & 1M & 1M & 1M \\ 11 & 11 & 11 & 12 & \ldots & 1m & 1M & 1M & 1M \\ 11 & 11 & 11 & 12 & \ldots & 1m & 1M & 1M & 1M \\ 21 & 21 & 21 & 22 & \ldots & 2m & 2M & 2M & 2M \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ n1 & n1 & n1 & n2 & \ldots & \mathrm{nm} & \mathrm{nM} & \mathrm{nM} & \mathrm{nM} \\ N1 & N1 & N1 & N2 & \ldots & \mathrm{Nm} & \mathrm{NM} & \mathrm{NM} & \mathrm{NM} \\ N1 & N1 & N1 & N2 & \ldots & \mathrm{Nm} & \mathrm{NM} & \mathrm{NM} & \mathrm{NM} \\ N1 & N1 & N1 & N2 & \ldots & \mathrm{Nm} & \mathrm{NM} & \mathrm{NM} & \mathrm{NM} \end{bmatrix}$$

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:

$$\begin{bmatrix} nm & nM & n1 & n2 & \ldots & nm & nM & n1 & n2 \\ Nm & NM & N1 & N2 & \ldots & Nm & NM & N1 & N2 \\ \mathrm{1m} & \mathrm{1M} & 11 & 12 & \ldots & \mathrm{1m} & \mathrm{1M} & 11 & 12 \\ \mathrm{2m} & \mathrm{2M} & 21 & 22 & \ldots & \mathrm{2m} & \mathrm{2M} & 21 & 22 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ nm & nM & n1 & n2 & \ldots & nm & nM & n1 & n2 \\ Nm & NM & N1 & N2 & \ldots & Nm & NM & N1 & N2 \\ \mathrm{1m} & \mathrm{1M} & 11 & 12 & \ldots & \mathrm{1m} & \mathrm{1M} & 11 & 12 \\ \mathrm{2m} & \mathrm{2M} & 21 & 22 & \ldots & \mathrm{2m} & \mathrm{2M} & 21 & 22 \end{bmatrix}$$

The value ‘none’ indicates that the input image is extended with pixel values of zero for R, G, B and A.

The initial value for `edgeMode` is ‘duplicate’.

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 kernelUnitLength. In some implementations, the most consistent results and the fastest performance will be achieved if the pixel grid of the temporary off-screen images aligns with the pixel grid of the kernel.

If a negative or zero value is specified the default value will be used instead.

> Note: This attribute is deprecated and will be removed. It does not provide a reliable way to create platform independent results. Future versions of this specification will cover this use case.

Animatable: yes.

`preserveAlpha` = "*false | true*"

A value of ‘false’ indicates that the convolution will apply to all channels, including the alpha channel. In this case the $ALPHA_{X,Y}$ of the convolution formula for a given pixel is:

```
ALPHA_{X,Y} = (
              SUM I=0 to [orderY-1] {
                SUM J=0 to [orderX-1] {
                  SOURCE X-targetX+J, Y-targetY+I *  kernelMatrix_{orderX-J-1,  orderY-I-1}
                }
              }
            ) /  divisor +  bias
```

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 and apply the kernel. In this case the $ALPHA_{X,Y}$ of the convolution formula for a given pixel is:

$ALPHA_{X,Y} = SOURCE_{X,Y}$

The initial value for `preserveAlpha` is ‘false’.

Animatable: yes.

§ 9.10. Filter primitive `<feDiffuseLighting>`

Name:	***`feDiffuseLighting`***
Categories:	filter primitive
Content model:	Any number of descriptive elements, `<script>` and exactly one light sources element, in any order.
Attributes:	

- core attributes — id, xml:base, xml:lang, xml:space
- presentation attributes — ‘alignment-baseline’, ‘baseline-shift’, ‘clip’, ‘clip-path’, ‘clip-rule’, ‘color’, ‘color-interpolation’, ‘color-interpolation-filters’, ‘color-rendering’, ‘cursor’, ‘direction’, ‘display’, ‘dominant-baseline’, ‘enable-background’, ‘fill’, ‘fill-opacity’, ‘fill-rule’, ‘filter’, ‘flood-color’, ‘flood-opacity’, ‘font’, ‘font-family’, ‘font-size’, ‘font-size-adjust’, ‘font-stretch’, ‘font-style’, ‘font-variant’, ‘font-weight’, ‘glyph-orientation-horizontal’, ‘glyph-orientation-vertical’, ‘image-rendering’, ‘isolation’, ‘kerning’, ‘letter-spacing’, ‘lighting-color’, ‘marker’, ‘marker-end’, ‘marker-mid’, ‘marker-start’, ‘mask’, ‘opacity’, ‘overflow’, ‘pointer-events’, ‘shape-rendering’, ‘stop-color’, ‘stop-opacity’, ‘stroke’, ‘stroke-dasharray’, ‘stroke-dashoffset’, ‘stroke-linecap’,

‘stroke-linejoin’, ‘stroke-miterlimit’, ‘stroke-opacity’, ‘stroke-width’, ‘text-anchor’, ‘text-decoration’, ‘text-rendering’, ‘unicode-bidi’, ‘visibility’, ‘word-spacing’, ‘writing-mode’

- filter primitive attributes —`x`, `y`, `width`, `height`, `result`
- class
- style
- `in`
- `surfaceScale`
- `diffuseConstant`
- `kernelUnitLength`

DOM Interfaces: SVGFEDiffuseLightingElement

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 the 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). 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.

Note: Depending on the speed of the available interpolates, this choice may be affected by the ‘image-rendering’ property setting.

Note: 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 $\mathtt{Norm}(A_x,A_y,A_z)$ function is defined as:

$$Norm\,(A_x,A_y,A_z) = \sqrt{A_x^2 + A_y^2 + A_z^2}$$

Note: User agents may use the the "fast inverse square root" to optimize the equation and avoid time differences on extrema color values. See Privacy and Security Considerations section for more details about timing attacks.

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 I(x,y) is:

```
Z (x,y) = surfaceScale * I(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:

```
N_x (x,y) = - surfaceScale * FACTOR_x *
           (K_x(0,0)*I(x-dx,y-dy) + K_x(1,0)*I(x,y-dy) + K_x(2,0)*I(x+dx,y-dy) +
            K_x(0,1)*I(x-dx,y)    + K_x(1,1)*I(x,y)    + K_x(2,1)*I(x+dx,y)    +
            K_x(0,2)*I(x-dx,y+dy) + K_x(1,2)*I(x,y+dy) + K_x(2,2)*I(x+dx,y+dy))
N_y (x,y) = - surfaceScale * FACTOR_y *
           (K_y(0,0)*I(x-dx,y-dy) + K_y(1,0)*I(x,y-dy) + K_y(2,0)*I(x+dx,y-dy) +
            K_y(0,1)*I(x-dx,y)    + K_y(1,1)*I(x,y)    + K_y(2,1)*I(x+dx,y)    +
            K_y(0,2)*I(x-dx,y+dy) + K_y(1,2)*I(x,y+dy) + K_y(2,2)*I(x+dx,y+dy))
N_z (x,y) = 1.0

N = (N_x, N_y, N_z) / Norm((N_x,N_y,N_z))
```

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:	**Top row:**	**Top/right corner:**
$FACTOR_x=2/(3*dx)$ $K_x =$ $\begin{vmatrix} 0 & 0 & 0 \\ 0 & -2 & 2 \\ 0 & -1 & 1 \end{vmatrix}$ $FACTOR_y=2/(3*dy)$ $K_y =$ $\begin{vmatrix} 0 & 0 & 0 \\ 0 & -2 & -1 \\ 0 & 2 & 1 \end{vmatrix}$	$FACTOR_x=1/(3*dx)$ $K_x =$ $\begin{vmatrix} 0 & 0 & 0 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{vmatrix}$ $FACTOR_y=1/(2*dy)$ $K_y =$ $\begin{vmatrix} 0 & 0 & 0 \\ -1 & -2 & -1 \\ 1 & 2 & 1 \end{vmatrix}$	$FACTOR_x=2/(3*dx)$ $K_x =$ $\begin{vmatrix} 0 & 0 & 0 \\ -2 & 2 & 0 \\ -1 & 1 & 0 \end{vmatrix}$ $FACTOR_y=2/(3*dy)$ $K_y =$ $\begin{vmatrix} 0 & 0 & 0 \\ -1 & -2 & 0 \\ 1 & 2 & 0 \end{vmatrix}$
Left column:	**Interior pixels:**	**Right column:**
$FACTOR_x=1/(2*dx)$ $K_x =$ $\begin{vmatrix} 0 & -1 & 1 \\ 0 & -2 & 2 \\ 0 & -1 & 1 \end{vmatrix}$ $FACTOR_y=1/(3*dy)$ $K_y =$ $\begin{vmatrix} 0 & -2 & -1 \\ 0 & 0 & 0 \\ 0 & 2 & 1 \end{vmatrix}$	$FACTOR_x=1/(4*dx)$ $K_x =$ $\begin{vmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{vmatrix}$ $FACTOR_y=1/(4*dy)$ $K_y =$ $\begin{vmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{vmatrix}$	$FACTOR_x=1/(2*dx)$ $K_x =$ $\begin{vmatrix} -1 & 1 & 0 \\ -2 & 2 & 0 \\ -1 & 1 & 0 \end{vmatrix}$ $FACTOR_y=1/(3*dy)$ $K_y =$ $\begin{vmatrix} -1 & -2 & 0 \\ 0 & 0 & 0 \\ 1 & 2 & 0 \end{vmatrix}$
Bottom/left corner:	**Bottom row:**	**Bottom/right corner:**
$FACTOR_x=2/(3*dx)$	$FACTOR_x=1/(3*dx)$	$FACTOR_x=2/(3*dx)$

$K_x =$ | 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 |	$K_x =$ | -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 |	$K_x =$ | -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. ($L_r = L_g = L_b = 0$). If `limitingConeAngle` is specified, -L.S < cos(limitingConeAngle) also indicates that no light is present.

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 kernelUnitLength.

If a negative or zero value is specified the default value will be used instead.

> Note: This attribute is deprecated and will be removed. It does not provide a reliable way to create platform independent results. Future versions of this specification will cover this use case.

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’.

§ 9.11. Filter primitive `<feDisplacementMap>`

Name: ***feDisplacementMap***

Categories: filter primitive

Content model: Any number of descriptive elements, `<animate>`, `<script>`, `<set>` elements, in any order.

Attributes:

- core attributes — id, xml:base, xml:lang, xml:space
- presentation attributes — ‘alignment-baseline’, ‘baseline-shift’, ‘clip’, ‘clip-path’, ‘clip-rule’, ‘color’, ‘color-interpolation’, ‘color-interpolation-filters’, ‘color-rendering’, ‘cursor’, ‘direction’, ‘display’, ‘dominant-baseline’, ‘enable-background’, ‘fill’, ‘fill-opacity’, ‘fill-rule’, ‘filter’, ‘flood-color’, ‘flood-opacity’, ‘font’, ‘font-family’, ‘font-size’, ‘font-size-adjust’, ‘font-stretch’, ‘font-style’, ‘font-variant’, ‘font-weight’, ‘glyph-orientation-horizontal’, ‘glyph-orientation-vertical’, ‘image-rendering’, ‘isolation’, ‘kerning’, ‘letter-spacing’, ‘lighting-color’, ‘marker’, ‘marker-end’, ‘marker-mid’, ‘marker-start’, ‘mask’, ‘opacity’, ‘overflow’, ‘pointer-events’, ‘shape-rendering’, ‘stop-color’, ‘stop-opacity’, ‘stroke’, ‘stroke-dasharray’, ‘stroke-dashoffset’, ‘stroke-linecap’, ‘stroke-linejoin’, ‘stroke-miterlimit’, ‘stroke-opacity’, ‘stroke-width’, ‘text-anchor’, ‘text-decoration’, ‘text-rendering’, ‘unicode-bidi’, ‘visibility’, ‘word-spacing’, ‘writing-mode’
- filter primitive attributes —`x`, `y`, `width`, `height`, `result`
- class
- style
- `in`
- `in2`
- `scale`
- `xChannelSelector`
- `yChannelSelector`

DOM Interfaces: SVGFEDisplacementMapElement

ISSUE 3 Implementations do not match specification. <https://github.com/w3c/csswg-drafts/issues/113>

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 channel 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, `in2`, defines the inverse of the mapping performed.

The input image `in` is to remain premultiplied for this filter primitive. The calculations using the pixel values from `in2` are performed using non-premultiplied color values.

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.

> Note: Depending on the speed of the available interpolents, this choice may be affected by the 'image-rendering' property setting.

> Note: A future version of this spec will define the interpolation method to be used when distorting the source image making UAs rendering result more interoperable.

The 'color-interpolation-filters' property only applies to the `in2` source image and does not apply to the `in` source image. The in source image must remain in its current color space.

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.

The initial value for `scale` is '0'.

Animatable: yes.

`xChannelSelector` **= "*R* | *G* | *B* | *A*"**

Indicates which channel from `in2` to use to displace the pixels in `in` along the x-axis.

The initial value for `xChannelSelector` is 'A'.

Animatable: yes.

`yChannelSelector` **= "*R* | *G* | *B* | *A*"**

Indicates which channel from `in2` to use to displace the pixels in `in` along the y-axis.

The initial value for `yChannelSelector` is 'A'.

Animatable: yes.

`in2` **= "*(see `in` attribute)*"**

The second input image, which is used to displace the pixels in the image from attribute `in`. See defintion for in attribute.

Animatable: yes.

§ 9.12. Filter primitive `<feDropShadow>`

Name:	*`feDropShadow`*
Categories:	filter primitive
Content model:	Any number of descriptive elements, `<animate>`, `<script>`, `<set>` elements, in any order.
Attributes:	

- core attributes — id, xml:base, xml:lang, xml:space
- presentation attributes — ‘alignment-baseline’, ‘baseline-shift’, ‘clip’, ‘clip-path’, ‘clip-rule’, ‘color’, ‘color-interpolation’, ‘color-interpolation-filters’, ‘color-rendering’, ‘cursor’, ‘direction’, ‘display’, ‘dominant-baseline’, ‘enable-background’, ‘fill’, ‘fill-opacity’, ‘fill-rule’, ‘filter’, ‘flood-color’, ‘flood-opacity’, ‘font’, ‘font-family’, ‘font-size’, ‘font-size-adjust’, ‘font-stretch’, ‘font-style’, ‘font-variant’, ‘font-weight’, ‘glyph-orientation-horizontal’, ‘glyph-orientation-vertical’, ‘image-rendering’, ‘isolation’, ‘kerning’, ‘letter-spacing’, ‘lighting-color’, ‘marker’, ‘marker-end’, ‘marker-mid’, ‘marker-start’, ‘mask’, ‘opacity’, ‘overflow’, ‘pointer-events’, ‘shape-rendering’, ‘stop-color’, ‘stop-opacity’, ‘stroke’, ‘stroke-dasharray’, ‘stroke-dashoffset’, ‘stroke-linecap’, ‘stroke-linejoin’, ‘stroke-miterlimit’, ‘stroke-opacity’, ‘stroke-width’, ‘text-anchor’, ‘text-decoration’, ‘text-rendering’, ‘unicode-bidi’, ‘visibility’, ‘word-spacing’, ‘writing-mode’
- filter primitive attributes —`x`, `y`, `width`, `height`, `result`
- class
- style
- `in`
- `stdDeviation`
- `dx`
- `dy`

DOM Interfaces: SVGFEDropShadowElement

This filter creates a drop shadow of the input image. It is a shorthand filter, and is defined in terms of combinations of other filter primitives. The expectation is that it can be optimized more easily by implementations.

The result of a `<feDropShadow>` filter primitive is equivalent to the following:

```
<feGaussianBlur in="alpha-channel-of-feDropShadow-in"   stdDeviation="stdDeviation-of-feDropShadow",
<feOffset dx="dx-of-feDropShadow"   dy="dy-of-feDropShadow" result="offsetblur"/>
<feFlood flood-color="flood-color-of-feDropShadow"  flood-opacity="flood-opacity-of-feDropShadow"/>
<feComposite in2="offsetblur" operator="in"/>
<feMerge>
  <feMergeNode/>
  <feMergeNode in="in-of-feDropShadow"/>
</feMerge>
```

The above divided into steps:

1. Take the alpha channel of the input to the `<feDropShadow>` filter primitive and the `stdDeviation` on the feDropShadow and do processing as if the following `<feGaussianBlur>` was applied:

   ```
   <feGaussianBlur in="alpha-channel-of-feDropShadow-in" stdDeviation="stdDeviation-of-feDropShado
   ```

2. Offset the result of step 1 by `dx` and `dy` as specified on the `<feDropShadow>` element, equivalent to applying an `<feOffset>` with these parameters:

   ```
   <feOffset dx="dx-of-feDropShadow" dy="dy-of-feDropShadow" result="offsetblur"/>
   ```

3. Do processing as if an `<feFlood>` element with ‘flood-color’ and ‘flood-opacity’ as specified on the `<feDropShadow>` was applied:

   ```
   <feFlood flood-color="flood-color-of-feDropShadow" flood-opacity="flood-opacity-of-feDropShadow
   ```

4. Composite the result of the `<feFlood>` in step 3 with the result of the `<feOffset>` in step 2 as if an `<feComposite>` filter primitive with `operator="in"` was applied:

```
<feComposite in2="offsetblur" operator="in"/>
```

5. Finally merge the result of the previous step, doing processing as if the following `<feMerge>` was performed:

```
<feMerge>
  <feMergeNode/>
  <feMergeNode in="in-of-feDropShadow"/>
</feMerge>
```

Note: that while the definition of the `<feDropShadow>` filter primitive says that it can be expanded into an equivalent tree it is not required that it is implemented like that. The expectation is that user agents can optimize the handling by not having to do all the steps separately.

Beyond the DOM interface SVGFEDropShadowElement there is no way of accessing the internals of the `<feDropShadow>` filter primitive, meaning if the filter primitive is implemented as an equivalent tree then that tree must not be exposed to the DOM.

Attribute definitions:

dx = "***<number>***"

The x offset of the drop shadow.

The initial value for `dx` is ‘2’.

This attribute is then forwarded to the `dx` attribute of the internal `<feOffset>` element.

Animatable: yes.

dy = "***<number>***"

The y offset of the drop shadow.

The initial value for `dy` is ‘2’.

This attribute is then forwarded to the `dy` attribute of the internal `<feOffset>` element.

Animatable: yes.

stdDeviation = "***<number-optional-number>***"

The standard deviation for the blur operation in the drop shadow.

The initial value for `stdDeviation` is ‘2’.

This attribute is then forwarded to the `stdDeviation` attribute of the internal `<feGaussianBlur>` element.

Animatable: yes.

§ 9.13. Filter primitive `<feFlood>`

Name:	*feFlood*
Categories:	filter primitive
Content model:	Any number of descriptive elements, `<animate>`, `<script>`, `<set>` elements, in any order.
Attributes:	• core attributes — id, xml:base, xml:lang, xml:space • presentation attributes — ‘alignment-baseline’, ‘baseline-shift’, ‘clip’, ‘clip-path’, ‘clip-rule’, ‘color’, ‘color-interpolation’, ‘color-interpolation-filters’, ‘color-rendering’, ‘cursor’, ‘direction’, ‘display’, ‘dominant-baseline’, ‘enable-background’, ‘fill’, ‘fill-opacity’, ‘fill-rule’, ‘filter’, ‘flood-color’, ‘flood-opacity’, ‘font’, ‘font-family’, ‘font-size’, ‘font-size-adjust’, ‘font-stretch’, ‘font-style’,

‘font-variant’, ‘font-weight’, ‘glyph-orientation-horizontal’, ‘glyph-orientation-vertical’, ‘image-rendering’, ‘isolation’, ‘kerning’, ‘letter-spacing’, ‘lighting-color’, ‘marker’, ‘marker-end’, ‘marker-mid’, ‘marker-start’, ‘mask’, ‘opacity’, ‘overflow’, ‘pointer-events’, ‘shape-rendering’, ‘stop-color’, ‘stop-opacity’, ‘stroke’, ‘stroke-dasharray’, ‘stroke-dashoffset’, ‘stroke-linecap’, ‘stroke-linejoin’, ‘stroke-miterlimit’, ‘stroke-opacity’, ‘stroke-width’, ‘text-anchor’, ‘text-decoration’, ‘text-rendering’, ‘unicode-bidi’, ‘visibility’, ‘word-spacing’, ‘writing-mode’

- filter primitive attributes —`x`, `y`, `width`, `height`, `result`
- class
- style

DOM Interfaces: SVGFEFloodElement

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 `<feFlood>` element.

§ 9.13.1. The ‘flood-color’ property

Name:	***‘flood-color’***
Value:	<color>
Initial:	black
Applies to:	`<feFlood>` and `<feDropShadow>` elements
Inherited:	no
Percentages:	n/a
Computed value:	as specified
Canonical order:	per grammar
Media:	visual
Animatable:	as by computed value

The ‘flood-color’ property indicates what color to used to flood the current filter primitive subregion.

The ‘flood-color’ property is a presentation attribute for SVG elements.

§ 9.13.2. The ‘flood-opacity’ property

Name:	*‘flood-opacity’*
Value:	<alpha-value>
Initial:	1
Applies to:	`<feFlood>` and `<feDropShadow>` elements
Inherited:	no
Percentages:	n/a
Computed value:	the specified value converted to a number, clamped to the range [0,1]
Canonical order:	per grammar
Media:	visual
Animatable:	by computed value

The ‘flood-opacity’ property defines the opacity value to use across the entire filter primitive subregion. If the ‘flood-color’ value includes an alpha channel, the alpha channel gets multiplied with the computed value of the ‘flood-opacity’ property.

The ‘flood-opacity’ property is a presentation attribute for SVG elements.

§ 9.14. Filter primitive `<feGaussianBlur>`

Name:	***`feGaussianBlur`***
Categories:	filter primitive
Content model:	Any number of descriptive elements, `<animate>`, `<script>`, `<set>` elements, in any order.
Attributes:	• core attributes — id, xml:base, xml:lang, xml:space • presentation attributes — ‘alignment-baseline’, ‘baseline-shift’, ‘clip’, ‘clip-path’, ‘clip-rule’, ‘color’, ‘color-interpolation’, ‘color-interpolation-filters’, ‘color-rendering’, ‘cursor’, ‘direction’, ‘display’, ‘dominant-baseline’, ‘enable-background’, ‘fill’, ‘fill-opacity’, ‘fill-rule’, ‘filter’, ‘flood-color’, ‘flood-opacity’, ‘font’, ‘font-family’, ‘font-size’, ‘font-size-adjust’, ‘font-stretch’, ‘font-style’, ‘font-variant’, ‘font-weight’, ‘glyph-orientation-horizontal’, ‘glyph-orientation-vertical’, ‘image-rendering’, ‘isolation’, ‘kerning’, ‘letter-spacing’, ‘lighting-color’, ‘marker’, ‘marker-end’, ‘marker-mid’, ‘marker-start’, ‘mask’, ‘opacity’, ‘overflow’, ‘pointer-events’, ‘shape-rendering’, ‘stop-color’, ‘stop-opacity’, ‘stroke’, ‘stroke-dasharray’, ‘stroke-dashoffset’, ‘stroke-linecap’, ‘stroke-linejoin’, ‘stroke-miterlimit’, ‘stroke-opacity’, ‘stroke-width’, ‘text-anchor’, ‘text-decoration’, ‘text-rendering’, ‘unicode-bidi’, ‘visibility’, ‘word-spacing’, ‘writing-mode’ • filter primitive attributes —`x`, `y`, `width`, `height`, `result` • class • style • `in` • `stdDeviation` • `edgeMode`
DOM Interfaces:	SVGFEGaussianBlurElement

This filter primitive performs a Gaussian blur on the input image.

The Gaussian blur kernel is an approximation of the normalized convolution:

```
G(x,y) = H(x)I(y)
```

where

`H(x) = exp(-x`^{2}`/ (2s`^{2}`)) / sqrt(2π * s`^{2}`)`

and

`I(y) = exp(-y`^{2}`/ (2t`^{2}`)) / sqrt(2π * t`^{2}`)`

with "s" being the standard deviation in the x direction and "t" being the standard deviation in the y direction, as 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 * π) / 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.

The approximation formula also applies correspondingly to "t".

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. This optimization must be omitted if it leads to privacy concerns of any matter. (See section Privacy and Security Considerations for more details about timing attacks.) If the input has infinite extent and is constant (e.g FillPaint where the fill is a solid color), this operation has no effect. If the input has infinite extent and the filter result where the fill is a solid color) is the input to an `<feTile>`, the filter is evaluated with periodic boundary conditions.

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 or a value of zero disables the effect of the given filter primitive (i.e., the result is the filter input image).

If `stdDeviation` is ‘0’ in only one of X or Y, then the effect is that the blur is only applied in the direction that has a non-zero value.

The initial value for `stdDeviation` is ‘0’.

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:

`wrap` indicates that the input image is extended by taking the color values from the opposite edge of the image.

The value ‘none’ indicates that the input image is extended with pixel values of zero for R, G, B and A.

The initial value for `edgeMode` is ‘none’.

Animatable: yes.

The example at the start of this chapter makes use of the `<feGaussianBlur>` filter primitive to create a drop shadow effect.

§ 9.15. Filter primitive `<feImage>`

Name:	***`feImage`***
Categories:	filter primitive
Content model:	Any number of descriptive elements, `<animate>`, `<animateTransform>`, `<script>`, `<set>` elements, in any order.
Attributes:	• core attributes — id, xml:base, xml:lang, xml:space • presentation attributes — ‘alignment-baseline’, ‘baseline-shift’, ‘clip’, ‘clip-path’, ‘clip-rule’, ‘color’, ‘color-interpolation’, ‘color-interpolation-filters’, ‘color-rendering’, ‘cursor’, ‘direction’, ‘display’, ‘dominant-baseline’, ‘enable-background’, ‘fill’, ‘fill-opacity’, ‘fill-rule’, ‘filter’, ‘flood-color’, ‘flood-opacity’, ‘font’, ‘font-family’, ‘font-size’, ‘font-size-adjust’, ‘font-stretch’, ‘font-style’, ‘font-variant’, ‘font-weight’, ‘glyph-orientation-horizontal’, ‘glyph-orientation-vertical’, ‘image-rendering’, ‘isolation’, ‘kerning’, ‘letter-spacing’, ‘lighting-color’, ‘marker’, ‘marker-end’, ‘marker-mid’, ‘marker-start’, ‘mask’, ‘opacity’, ‘overflow’, ‘pointer-events’, ‘shape-rendering’, ‘stop-color’, ‘stop-opacity’, ‘stroke’, ‘stroke-dasharray’, ‘stroke-dashoffset’, ‘stroke-linecap’, ‘stroke-linejoin’, ‘stroke-miterlimit’, ‘stroke-opacity’, ‘stroke-width’, ‘text-anchor’, ‘text-decoration’, ‘text-rendering’, ‘unicode-bidi’, ‘visibility’, ‘word-spacing’, ‘writing-mode’ • filter primitive attributes —`x`, `y`, `width`, `height`, `result` • class • style • externalResourcesRequired • `preserveAspectRatio` • `xlink:href` • `href` • `crossorigin`
DOM Interfaces:	SVGFEImageElement

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 `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.

A `href` reference that is an empty image (zero width or zero height), that fails to download, is non-existent, or that cannot be displayed (e.g. because it is not in a supported image format) fills the filter primitive subregion with transparent black.

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.

Attribute definitions:

`xlink:href` = "<url>"

See `href` attribute.

Animatable: yes.

> Note: This *xlink:href* attribute is deprecated and should not be used in new content, it’s included for backwards compatibility reasons only. Authors should use the `href` attribute instead.

`href` = "**<url>**"

An <url> to an image resource or to an element. If both, the `xlink:href` and the `href` attribute are specified, the latter overrides the first definition.

Animatable: yes.

`preserveAspectRatio` = "***[defer] <align> [<meetOrSlice>]***"

See `preserveAspectRatio`.

The initial value for `preserveAspectRatio` is ‘xMidYMid meet’.

Animatable: yes.

`crossorigin` = "***anonymous | use-credentials***"

The crossorigin attribute is a CORS settings attribute. Its purpose is to allow images from third-party sites that allow cross-origin access to be used with `<feDisplacementMap>`. For the defintion see crossorigin attribute for the `<img>` tag [HTML5] and the Privacy and Security Considerations section in this specification.

Animatable: no.

EXAMPLE 10

The following example illustrates how images are placed relative to an object. From left to right:

- The default placement of an image. Note that the image is centered in the filter region and has the maximum size that will fit in the region consistent with preserving the aspect ratio.
- The image stretched to fit the bounding box of an object.
- The image placed using user coordinates. Note that the image is first centered in a box the size of the filter region and has the maximum size that will fit in the box consistent with preserving the aspect ratio. This box is then shifted by the given x and y values relative to the viewport the object is in.

```
<svg width="600" height="250" viewBox="0 0 600 250"
     xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink">
  <title>Example feImage - Examples of feImage use</title>
  <desc>Three examples of using feImage, the first showing the
        default rendering, the second showing the image fit
        to a box and the third showing the image
        shifted and clipped.</desc>
  <defs>
    <filter id="Default">
      <feImage xlink:href="smiley.png" />
    </filter>
    <filter id="Fitted" primitiveUnits="objectBoundingBox">
      <feImage xlink:href="smiley.png"
         x="0" y="0" width="100%" height="100%"
         preserveAspectRatio="none"/>
    </filter>
    <filter id="Shifted">
      <feImage xlink:href="smiley.png"
         x="500" y="5"/>
    </filter>
  </defs>
  <rect fill="none" stroke="blue"
        x="1" y="1" width="598" height="248"/>
  <g>
    <rect x="50"  y="25" width="100" height="200" filter="url(#Default)"/>
    <rect x="50"  y="25" width="100" height="200" fill="none" stroke="green"/>
    <rect x="250" y="25" width="100" height="200" filter="url(#Fitted)"/>
    <rect x="250" y="25" width="100" height="200" fill="none" stroke="green"/>
    <rect x="450" y="25" width="100" height="200" filter="url(#Shifted)"/>
    <rect x="450" y="25" width="100" height="200" fill="none" stroke="green"/>
  </g>
</svg>
```

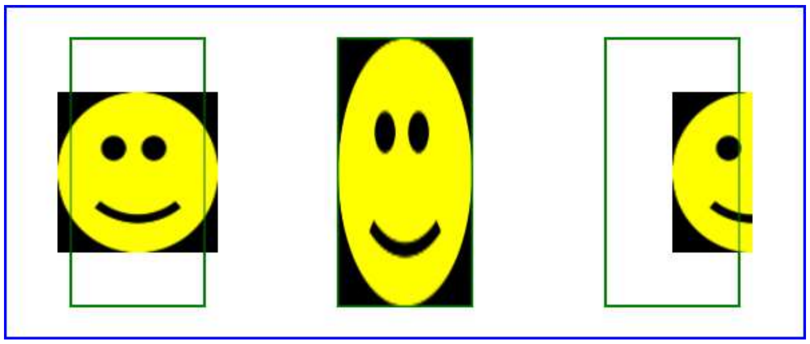


Figure 8. *Example of feImage*

View this example as SVG

§ 9.16. Filter primitive `<feMerge>`

Name:	*`feMerge`*
Categories:	filter primitive
Content model:	Any number of descriptive elements, `<feMergeNode>`, `<script>` elements, in any order.
Attributes:	• core attributes — id, xml:base, xml:lang, xml:space • presentation attributes — ‘alignment-baseline’, ‘baseline-shift’, ‘clip’, ‘clip-path’, ‘clip-rule’, ‘color’, ‘color-interpolation’, ‘color-interpolation-filters’, ‘color-rendering’, ‘cursor’, ‘direction’, ‘display’, ‘dominant-baseline’, ‘enable-background’, ‘fill’, ‘fill-opacity’, ‘fill-rule’, ‘filter’, ‘flood-color’, ‘flood-opacity’, ‘font’, ‘font-family’, ‘font-size’, ‘font-size-adjust’, ‘font-stretch’, ‘font-style’, ‘font-variant’, ‘font-weight’, ‘glyph-orientation-horizontal’, ‘glyph-orientation-vertical’, ‘image-rendering’, ‘isolation’, ‘kerning’, ‘letter-spacing’, ‘lighting-color’, ‘marker’, ‘marker-end’, ‘marker-mid’, ‘marker-start’, ‘mask’, ‘opacity’, ‘overflow’, ‘pointer-events’, ‘shape-rendering’, ‘stop-color’, ‘stop-opacity’, ‘stroke’, ‘stroke-dasharray’, ‘stroke-dashoffset’, ‘stroke-linecap’, ‘stroke-linejoin’, ‘stroke-miterlimit’, ‘stroke-opacity’, ‘stroke-width’, ‘text-anchor’, ‘text-decoration’, ‘text-rendering’, ‘unicode-bidi’, ‘visibility’, ‘word-spacing’, ‘writing-mode’ • filter primitive attributes —`x`, `y`, `width`, `height`, `result` • class • style
DOM Interfaces:	SVGFEMergeElement

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.

The example at the start of this chapter makes use of the `<feMerge>` filter primitive to composite two intermediate filter results together.

§ 9.16.1. Merge node `<feMergeNode>`

Name:	*`feMergeNode`*
Categories:	None.
Content model:	Any number of descriptive elements, `<animate>`, `<script>`, `<set>` elements, in any order.
Attributes:	• core attributes — id, xml:base, xml:lang, xml:space • `in`

DOM Interfaces: SVGFEMergeNodeElement

§ 9.17. Filter primitive `<feMorphology>`

Name: ***`feMorphology`***

Categories: filter primitive

Content model: Any number of descriptive elements, `<animate>`, `<script>`, `<set>` elements, in any order.

Attributes:

- core attributes — id, xml:base, xml:lang, xml:space
- presentation attributes — ‘alignment-baseline’, ‘baseline-shift’, ‘clip’, ‘clip-path’, ‘clip-rule’, ‘color’, ‘color-interpolation’, ‘color-interpolation-filters’, ‘color-rendering’, ‘cursor’, ‘direction’, ‘display’, ‘dominant-baseline’, ‘enable-background’, ‘fill’, ‘fill-opacity’, ‘fill-rule’, ‘filter’, ‘flood-color’, ‘flood-opacity’, ‘font’, ‘font-family’, ‘font-size’, ‘font-size-adjust’, ‘font-stretch’, ‘font-style’, ‘font-variant’, ‘font-weight’, ‘glyph-orientation-horizontal’, ‘glyph-orientation-vertical’, ‘image-rendering’, ‘isolation’, ‘kerning’, ‘letter-spacing’, ‘lighting-color’, ‘marker’, ‘marker-end’, ‘marker-mid’, ‘marker-start’, ‘mask’, ‘opacity’, ‘overflow’, ‘pointer-events’, ‘shape-rendering’, ‘stop-color’, ‘stop-opacity’, ‘stroke’, ‘stroke-dasharray’, ‘stroke-dashoffset’, ‘stroke-linecap’, ‘stroke-linejoin’, ‘stroke-miterlimit’, ‘stroke-opacity’, ‘stroke-width’, ‘text-anchor’, ‘text-decoration’, ‘text-rendering’, ‘unicode-bidi’, ‘visibility’, ‘word-spacing’, ‘writing-mode’
- filter primitive attributes —`x`, `y`, `width`, `height`, `result`
- class
- style
- `in`
- `operator`
- `radius`

DOM Interfaces: SVGFEMorphologyElement

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. This optimization must be omitted if it leads to privacy concerns of any matter. (See section Privacy and Security Considerations for more details.)

If the input has infinite extent and is constant (e.g FillPaint where the fill is a solid color), this operation has no effect. If the input has infinite extent and the filter result is the input to an `<feTile>`, 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.

Attribute definitions:

`operator` = **"*erode* | *dilate*"**

A keyword indicating whether to erode (i.e., thin) or dilate (fatten) the source graphic.

The initial value for `operator` is ‘erode’.

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 or zero value disables the effect of the given filter primitive (i.e., the result is the filter input image).

The initial value for `radius` is ‘0’.

Animatable: yes.

EXAMPLE 11

```
<svg width="5cm" height="7cm" viewBox="0 0 700 500"
     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 isolation="isolate" >
    <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>
```

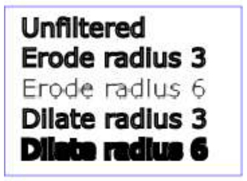


Figure 9. *Example of feMorphology*

View this example as SVG

§ 9.18. Filter primitive `<feOffset>`

Name: ***`feOffset`***

Categories: filter primitive

Content model: Any number of descriptive elements, `<animate>`, `<script>`, `<set>` elements, in any order.

Attributes:

- core attributes — id, xml:base, xml:lang, xml:space
- presentation attributes — ‘alignment-baseline’, ‘baseline-shift’, ‘clip’, ‘clip-path’, ‘clip-rule’, ‘color’, ‘color-interpolation’, ‘color-interpolation-filters’, ‘color-rendering’, ‘cursor’, ‘direction’, ‘display’, ‘dominant-baseline’, ‘enable-background’, ‘fill’, ‘fill-opacity’, ‘fill-rule’, ‘filter’, ‘flood-color’, ‘flood-opacity’, ‘font’, ‘font-family’, ‘font-size’, ‘font-size-adjust’, ‘font-stretch’, ‘font-style’, ‘font-variant’, ‘font-weight’, ‘glyph-orientation-horizontal’, ‘glyph-orientation-vertical’, ‘image-rendering’, ‘isolation’, ‘kerning’, ‘letter-spacing’, ‘lighting-color’, ‘marker’, ‘marker-end’, ‘marker-mid’, ‘marker-start’, ‘mask’, ‘opacity’, ‘overflow’, ‘pointer-events’, ‘shape-rendering’, ‘stop-color’, ‘stop-opacity’, ‘stroke’, ‘stroke-dasharray’, ‘stroke-dashoffset’, ‘stroke-linecap’, ‘stroke-linejoin’, ‘stroke-miterlimit’, ‘stroke-opacity’, ‘stroke-width’, ‘text-anchor’, ‘text-decoration’, ‘text-rendering’, ‘unicode-bidi’, ‘visibility’, ‘word-spacing’, ‘writing-mode’
- filter primitive attributes —`x`, `y`, `width`, `height`, `result`
- class
- style
- `in`
- `dx`
- `dy`

DOM Interfaces: SVGFEOffsetElement

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.

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.

The initial value for `dx` is ‘0’.

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.

The initial value for `dy` is ‘0’.

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.

§ 9.19. Filter primitive `<feSpecularLighting>`

Name:	***feSpecularLighting***
Categories:	filter primitive
Content model:	Any number of descriptive elements, `<script>` and exactly one light sources element, in any order.

Attributes:

- core attributes — id, xml:base, xml:lang, xml:space
- presentation attributes — ‘alignment-baseline’, ‘baseline-shift’, ‘clip’, ‘clip-path’, ‘clip-rule’, ‘color’, ‘color-interpolation’, ‘color-interpolation-filters’, ‘color-rendering’, ‘cursor’, ‘direction’, ‘display’, ‘dominant-baseline’, ‘enable-background’, ‘fill’, ‘fill-opacity’, ‘fill-rule’, ‘filter’, ‘flood-color’, ‘flood-opacity’, ‘font’, ‘font-family’, ‘font-size’, ‘font-size-adjust’, ‘font-stretch’, ‘font-style’, ‘font-variant’, ‘font-weight’, ‘glyph-orientation-horizontal’, ‘glyph-orientation-vertical’, ‘image-rendering’, ‘isolation’, ‘kerning’, ‘letter-spacing’, ‘lighting-color’, ‘marker’, ‘marker-end’, ‘marker-mid’, ‘marker-start’, ‘mask’, ‘opacity’, ‘overflow’, ‘pointer-events’, ‘shape-rendering’, ‘stop-color’, ‘stop-opacity’, ‘stroke’, ‘stroke-dasharray’, ‘stroke-dashoffset’, ‘stroke-linecap’, ‘stroke-linejoin’, ‘stroke-miterlimit’, ‘stroke-opacity’, ‘stroke-width’, ‘text-anchor’, ‘text-decoration’, ‘text-rendering’, ‘unicode-bidi’, ‘visibility’, ‘word-spacing’, ‘writing-mode’
- filter primitive attributes —`x`, `y`, `width`, `height`, `result`
- class
- style
- `in`
- `surfaceScale`
- `specularConstant`
- `specularExponent`
- `kernelUnitLength`

DOM Interfaces: SVGFESpecularLightingElement

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 Blinn-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).

> Note: 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 from a second filter primitive 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:

$S_r = k_s * pow(N.H, specularExponent) * L_r$
$S_g = k_s * pow(N.H, specularExponent) * L_g$
$S_b = k_s * pow(N.H, specularExponent) * L_b$
$S_a = max(S_r, S_g, S_b)$

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.

Attribute definitions:

`surfaceScale` **= "*<number>*"**
: height of surface when A_{in} = 1.

  The initial value for `surfaceScale` is '1'.

  Animatable: yes.

`specularConstant` **= "*<number>*"**
: ks in Phong lighting model. In SVG, this can be any non-negative number.

  The initial value for `specularConstant` is '1'.

  Animatable: yes.

`specularExponent` **= "*<number>*"**
: Exponent for specular term, larger is more "shiny".

  The initial value for `specularExponent` is '1'.

  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 kernelUnitLength.

  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'.

The example at the start of this chapter makes use of the `<feSpecularLighting>` filter primitive to achieve a highly reflective, 3D glowing effect.

§ 9.20. Filter primitive `<feTile>`

Name:	***`feTile`***
Categories:	filter primitive
Content model:	Any number of descriptive elements, `<animate>`, `<script>`, `<set>` elements, in any order.
Attributes:	

- core attributes — id, xml:base, xml:lang, xml:space
- presentation attributes — 'alignment-baseline', 'baseline-shift', 'clip', 'clip-path', 'clip-rule', 'color', 'color-interpolation', 'color-interpolation-filters', 'color-rendering', 'cursor', 'direction', 'display', 'dominant-baseline', 'enable-background', 'fill', 'fill-opacity', 'fill-rule', 'filter', 'flood-color', 'flood-opacity', 'font', 'font-family', 'font-size', 'font-size-adjust', 'font-stretch', 'font-style',

‘font-variant’, ‘font-weight’, ‘glyph-orientation-horizontal’, ‘glyph-orientation-vertical’, ‘image-rendering’, ‘isolation’, ‘kerning’, ‘letter-spacing’, ‘lighting-color’, ‘marker’, ‘marker-end’, ‘marker-mid’, ‘marker-start’, ‘mask’, ‘opacity’, ‘overflow’, ‘pointer-events’, ‘shape-rendering’, ‘stop-color’, ‘stop-opacity’, ‘stroke’, ‘stroke-dasharray’, ‘stroke-dashoffset’, ‘stroke-linecap’, ‘stroke-linejoin’, ‘stroke-miterlimit’, ‘stroke-opacity’, ‘stroke-width’, ‘text-anchor’, ‘text-decoration’, ‘text-rendering’, ‘unicode-bidi’, ‘visibility’, ‘word-spacing’, ‘writing-mode’

- filter primitive attributes —`x`, `y`, `width`, `height`, `result`
- class
- style
- `in`

DOM Interfaces: SVGFETileElement

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 `<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.

§ 9.21. Filter primitive `<feTurbulence>`

Name: ***`feTurbulence`***

Categories: filter primitive

Content model: Any number of descriptive elements, `<animate>`, `<script>`, `<set>` elements, in any order.

Attributes:

- core attributes — id, xml:base, xml:lang, xml:space
- presentation attributes — ‘alignment-baseline’, ‘baseline-shift’, ‘clip’, ‘clip-path’, ‘clip-rule’, ‘color’, ‘color-interpolation’, ‘color-interpolation-filters’, ‘color-rendering’, ‘cursor’, ‘direction’, ‘display’, ‘dominant-baseline’, ‘enable-background’, ‘fill’, ‘fill-opacity’, ‘fill-rule’, ‘filter’, ‘flood-color’, ‘flood-opacity’, ‘font’, ‘font-family’, ‘font-size’, ‘font-size-adjust’, ‘font-stretch’, ‘font-style’, ‘font-variant’, ‘font-weight’, ‘glyph-orientation-horizontal’, ‘glyph-orientation-vertical’, ‘image-rendering’, ‘isolation’, ‘kerning’, ‘letter-spacing’, ‘lighting-color’, ‘marker’, ‘marker-end’, ‘marker-mid’, ‘marker-start’, ‘mask’, ‘opacity’, ‘overflow’, ‘pointer-events’, ‘shape-rendering’, ‘stop-color’, ‘stop-opacity’, ‘stroke’, ‘stroke-dasharray’, ‘stroke-dashoffset’, ‘stroke-linecap’, ‘stroke-linejoin’, ‘stroke-miterlimit’, ‘stroke-opacity’, ‘stroke-width’, ‘text-anchor’, ‘text-decoration’, ‘text-rendering’, ‘unicode-bidi’, ‘visibility’, ‘word-spacing’, ‘writing-mode’
- filter primitive attributes —`x`, `y`, `width`, `height`, `result`
- class
- style
- `baseFrequency`
- `numOctaves`
- `seed`
- `stitchTiles`

- `type`

DOM Interfaces:	SVGFETurbulenceElement

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" [TaM]. 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. The filter primitive subregion is to be passed as the arguments fTileX, fTileY, fTileWidth and fTileHeight.

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 or alpha value, use the formula `colorValue = (turbFunctionResult + 1) / 2`, then clamp to the range 0 to 1.

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 or alpha value, use the formula `colorValue = turbFunctionResult`, then clamp to the range 0 to 1.

The following order is used for applying the pseudo random numbers. An initial seed value is computed based on the `seed` attribute. 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 ‘color-interpolation-filters’ property:

```
/* 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;
      do {
         for (j = 0; j < 2; j++)
           fGradient[k][i][j] = (double)(((lSeed = random(lSeed)) % (BSize + BSize)) - BSize) / BSi
      } while(fGradient[k][i][0] == 0 && fGradient[k][i][1] == 0);
      s = double(sqrt(fGradient[k][i][0] * fGradient[k][i][0] + fGradient[k][i][1] * fGradient[k][i
      if (s > 1) {
          i--; // discard the current random vector; try it again.
          continue;
      }
      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;
  }
```

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.

The initial value for `baseFrequency` is ‘0’.

Negative values are unsupported.

Animatable: yes.

`numOctaves` = "*<integer>*"

The numOctaves parameter for the noise function.

The initial value for `numOctaves` is ‘1’.

Negative values are unsupported.

Animatable: yes.

> Note: The contribution of each additional octave to the color and alpha values in the final image is one-half of the preceding octave. At some point, the contribution of additional octaves becomes smaller than the color resolution for a given color depth. UAs may clamp the specified value for numOctaves during the processing depending on the supported color depth. (For example: For a color depth of 8 bits per channel, the UA may clamp the value of numOctaves to 9.)

`seed` = "*<number>*"

The starting number for the pseudo random number generator.

The initial value for `seed` is ‘0’.

When the seed number is handed over to the algorithm above it must first be truncated, i.e. rounded to the closest integer value towards zero.

Animatable: yes.

`stitchTiles` = "***stitch* | *noStitch***"

If `stitchTiles="noStitch"`, no attempt is 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.

The initial value for `stitchTiles` is ‘noStitch’.

Animatable: yes.

`type` = "***fractalNoise* | *turbulence***"

Indicates whether the filter primitive should perform a noise or turbulence function.

The initial value for `type` is ‘turbulence’.

Animatable: yes.

EXAMPLE 12

```
<svg width="450px" height="325px" viewBox="0 0 450 325"
     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>
```

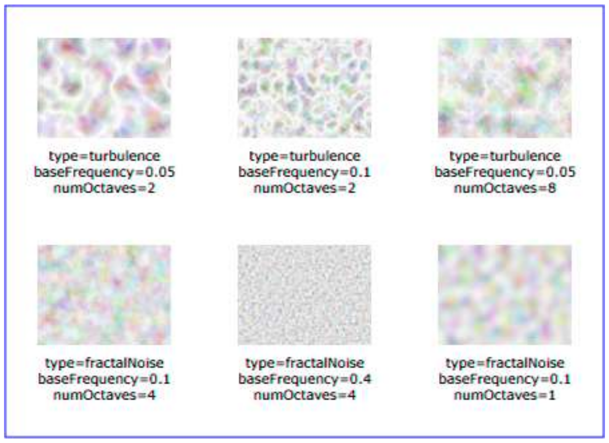


Figure 10. *Example of feTurbulence*

View this example as SVG

§ 10. The ‘color-interpolation-filters’ property

The description of the ‘color-interpolation-filters’ property is as follows:

Name:	***‘color-interpolation-filters’***
Value:	auto | sRGB | linearRGB
Initial:	linearRGB
Applies to:	All filter primitives
Inherited:	yes
Percentages:	n/a
Computed value:	as specified
Canonical order:	per grammar
Media:	visual
Animatable:	no

‘auto’
: Indicates that the user agent can choose either the ‘sRGB’ or ‘linearRGB’ spaces for filter effects color operations. This option indicates that the author doesn’t require that color operations occur in a particular color space.

‘sRGB’
: Indicates that filter effects color operations should occur in the sRGB color space.

‘linearRGB’
: Indicates that filter effects color operations should occur in the linearized RGB color space.

The ‘color-interpolation-filters’ property specifies the color space for imaging operations performed via filter effects.

Note: The ‘color-interpolation-filters’ property just has an affect on filter operations. Therefore, it has no effect on filter primitives like `<feOffset>`, `<feImage>`, `<feTile>` or `<feFlood>`.

Note: The ‘color-interpolation-filters’ has a different initial value than ‘color-interpolation’. ‘color-interpolation-filters’ has an initial value of ‘linearRGB’, where as ‘color-interpolation’ has an initial value of ‘sRGB’. Thus, in the default case, filter effects operations occur in the linearRGB color space, whereas all other color interpolations occur by default in the sRGB color space.

Note: The ‘color-interpolation-filters’ property has no affect on Filter Functions, which operate in the sRGB color space.

The ‘color-interpolation-filters’ property is a presentation attribute for SVG elements.

§ 11. Light source elements and properties

§ 11.1. 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.

§ 11.2. Light source `<feDistantLight>`

Name:	***`feDistantLight`***
Categories:	light source
Content model:	Any number of descriptive elements, `<animate>`, `<script>`, `<set>` elements, in any order.
Attributes:	• core attributes — id, xml:base, xml:lang, xml:space • `azimuth` • `elevation`
DOM Interfaces:	SVGFEDistantLightElement

Attribute definitions:

`azimuth` = "*<number>*"

Direction angle for the light source on the XY plane (clockwise), in degrees from the x axis.

The initial value for `azimuth` is ‘0’.

Animatable: yes.

`elevation` = "*<number>*"

Direction angle for the light source from the XY plane towards the Z-axis, in degrees. Note that the positive Z-axis points towards the viewer.

The initial value for `elevation` is ‘0’.

Animatable: yes.

The following diagram illustrates the angles which `azimuth` and `elevation` represent in an XYZ coordinate system.

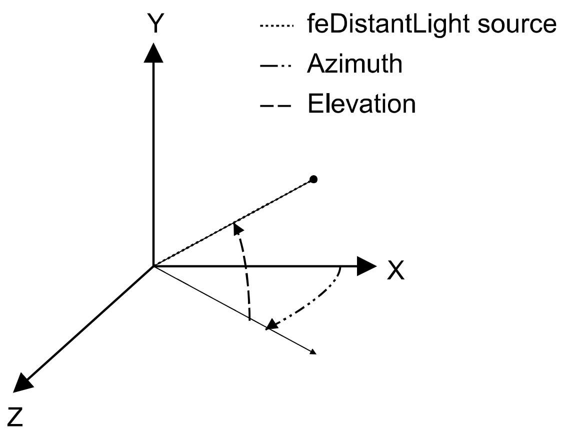


Figure 11. *Angles which azimuth and elevation represent*

§ 11.3. Light source `<fePointLight>`

Name: ***`fePointLight`***

Categories: light source

Content model: Any number of descriptive elements, `<animate>`, `<script>`, `<set>` elements, in any order.

Attributes:

- core attributes — id, xml:base, xml:lang, xml:space
- `x`
- `y`
- `z`

DOM Interfaces: SVGFEPointLightElement

Attribute definitions:

x = "***<number>***"

X location for the light source in the coordinate system established by attribute `primitiveUnits` on the `<filter>` element.

The initial value for `x` is ‘0’.

Animatable: yes.

y = "***<number>***"

Y location for the light source in the coordinate system established by attribute `primitiveUnits` on the `<filter>` element.

The initial value for `y` is ‘0’.

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 local 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 and Y.

The initial value for `z` is ‘0’.

Animatable: yes.

§ 11.4. Light source `<feSpotLight>`

Name: ***`feSpotLight`***

Categories:	light source
Content model:	Any number of descriptive elements, `<animate>`, `<script>`, `<set>` elements, in any order.
Attributes:	• core attributes — id, xml:base, xml:lang, xml:space • `x` • `y` • `z` • `pointsAtX` • `pointsAtY` • `pointsAtZ` • `specularExponent` • `limitingConeAngle`
DOM Interfaces:	SVGFESpotLightElement

Attribute definitions:

x = "***<number>***"

X location for the light source in the coordinate system established by attribute `primitiveUnits` on the `<filter>` element.

The initial value for `x` is ‘0’.

Animatable: yes.

y = "***<number>***"

Y location for the light source in the coordinate system established by attribute `primitiveUnits` on the `<filter>` element.

The initial value for `y` is ‘0’.

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 local 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 and Y.

The initial value for `z` is ‘0’.

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.

The initial value for `pointsAtX` is ‘0’.

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.

The initial value for `pointsAtY` is ‘0’.

Animatable: yes.

pointsAtZ = "***<number>***"

Z location in the coordinate system established by the attribute `primitiveUnits` on the `<filter>` element of the point at which the light source is pointing, assuming that, in the initial local 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 and Y.

The initial value for `pointsAtZ` is ‘0’.

Animatable: yes.

specularExponent = "<*number*>"

Exponent value controlling the focus for the light source.

The initial value for `specularExponent` is '1'.

See section Filter primitive <feDiffuseLighting> for how to use `specularExponent`.

> Note: `specularExponent` for `<feSpotLight>` serves a different use case than `specularExponent` for `<feSpecularLighting>`.

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 in degrees 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.

§ 11.5. The 'lighting-color' property

Name:	***'lighting-color'***
Value:	<color>
Initial:	white
Applies to:	`<feDiffuseLighting>` and `<feSpecularLighting>` elements
Inherited:	no
Percentages:	n/a
Computed value:	as specified
Canonical order:	per grammar
Media:	visual
Animatable:	as by computed value

The 'lighting-color' property defines the color of the light source for filter primitives `<feDiffuseLighting>` and `<feSpecularLighting>`.

The 'lighting-color' property is a presentation attribute for SVG elements.

§ 12. Filter CSS <image> values

CSS <image> values can be filtered with the filter functions specified for the CSS 'filter' property. This specification introduces the <image> <filter()> function with the following syntax:

```
filter() = filter( [ <image> | <string> ], <filter-value-list> )
```

The <filter()> function takes two arguments. The first argument is an <image>. The second is a filter function list as specified for the CSS 'filter' property. The function takes the <image> parameter and applies the filter rules, returning a processing image. Filter- and filter effect regions are sized according to the concrete object size of the input <image>.

Note: Since the dimension and origin of the original image must be preserved, some filter effects like <drop-shadow()> on a fully opaque image may not have any affect.

For the <blur()> function the `edgeMode` attribute on the `<feGaussianBlur>` element is set to ‘duplicate’. This produces more pleasant results on the edges of the filtered input image.

§ 12.1. Interpolating filter()

If both the starting and ending image are <filter()>s which may only differ by their used filter functions, they must be interpolated by interpolating their filter function lists as described in section Interpolation of Filters. Otherwise, they must be interpolated as generic <image>s. If the filter function interpolation can not be performed, the images must be interpolated as generic <image>s.

§ 13. Shorthands defined in terms of the `<filter>` element

§ 13.1. Filter primitive representation

Below are the equivalents for each of the filter functions expressed in terms of the 'filter element' element. The parameters from the function are labeled with brackets in the following style: [amount]. In the case of parameters that are percentage values, they are converted to real numbers.

§ 13.1.1. grayscale

```
<filter id="grayscale">
  <feColorMatrix type="matrix"
             values="
    (0.2126 + 0.7874 * [1 - amount]) (0.7152 - 0.7152  * [1 - amount]) (0.0722 - 0.0722 * [1 - amou
    (0.2126 - 0.2126 * [1 - amount]) (0.7152 + 0.2848  * [1 - amount]) (0.0722 - 0.0722 * [1 - amou
    (0.2126 - 0.2126 * [1 - amount]) (0.7152 - 0.7152  * [1 - amount]) (0.0722 + 0.9278 * [1 - amou
    0 0 0 1 0"/>
</filter>
```

§ 13.1.2. sepia

```
<filter id="sepia">
  <feColorMatrix type="matrix"
             values="
    (0.393 + 0.607 * [1 - amount]) (0.769 - 0.769 * [1 - amount]) (0.189 - 0.189 * [1 - amount]) 0 
    (0.349 - 0.349 * [1 - amount]) (0.686 + 0.314 * [1 - amount]) (0.168 - 0.168 * [1 - amount]) 0 
    (0.272 - 0.272 * [1 - amount]) (0.534 - 0.534 * [1 - amount]) (0.131 + 0.869 * [1 - amount]) 0 
    0 0 0 1 0"/>
</filter>
```

§ 13.1.3. saturate

```
<filter id="saturate">
  <feColorMatrix type="saturate" values="[amount]"/>
</filter>
```

§ 13.1.4. hue-rotate

```
<filter id="hue-rotate">
  <feColorMatrix type="hueRotate" values="[angle]"/>
</filter>
```

§ 13.1.5. invert

```
<filter id="invert">
  <feComponentTransfer>
      <feFuncR type="table" tableValues="[amount] (1 - [amount])"/>
      <feFuncG type="table" tableValues="[amount] (1 - [amount])"/>
      <feFuncB type="table" tableValues="[amount] (1 - [amount])"/>
  </feComponentTransfer>
</filter>
```

§ 13.1.6. opacity

```
<filter id="opacity">
  <feComponentTransfer>
      <feFuncA type="table" tableValues="0 [amount]"/>
  </feComponentTransfer>
</filter>
```

§ 13.1.7. brightness

```
<filter id="brightness">
  <feComponentTransfer>
      <feFuncR type="linear" slope="[amount]"/>
      <feFuncG type="linear" slope="[amount]"/>
      <feFuncB type="linear" slope="[amount]"/>
  </feComponentTransfer>
</filter>
```

§ 13.1.8. contrast

```
<filter id="contrast">
  <feComponentTransfer>
      <feFuncR type="linear" slope="[amount]" intercept="-(0.5 * [amount]) + 0.5"/>
      <feFuncG type="linear" slope="[amount]" intercept="-(0.5 * [amount]) + 0.5"/>
      <feFuncB type="linear" slope="[amount]" intercept="-(0.5 * [amount]) + 0.5"/>
  </feComponentTransfer>
</filter>
```

§ 13.1.9. blur

```
<filter id="blur">
  <feGaussianBlur stdDeviation="[radius radius]" edgeMode="[edge mode]" >
</filter>
```

Where edge mode computes to ‘none’ for the ‘filter’ property and to ‘duplicate’ for the CSS Image <filter()> function.

Note: The <blur()> function may increase the UA defined filter region. See Filter region for shorthands.

§ 13.1.10. drop-shadow

```
<filter id="drop-shadow">
  <feGaussianBlur in="[alpha-channel-of-input]" stdDeviation="[radius]"/>
  <feOffset dx="[offset-x]" dy="[offset-y]" result="offsetblur"/>
  <feFlood flood-color="[color]"/>
  <feComposite in2="offsetblur" operator="in"/>
  <feMerge>
    <feMergeNode/>
    <feMergeNode in="[input-image]"/>
  </feMerge>
</filter>
```

Note: The <drop-shadow()> function may increase the UA defined filter region. See Filter region for shorthands.

§ 13.2. Filter region for shorthands

All shorthand filters implemented with filter primitives in the previous subsection must have a UA defined filter region. The filter region must cover the visual content of an element including overflowing content, graphical control elements such as scrollbars, ‘border’/‘border-image’, ‘box-shadow’, ‘text-shadow’ and ‘outline’. Furthermore, if a shorthand filter expands this visible area like it is the case for <blur()> or <drop-shadow()> the filter region must cover this area as well.

Note: For the handling of filter sources see section Filter region.

§ 14. Animation of Filters

§ 14.1. Interpolation of Filter Function Lists

For interpolation between one filter and a second, the steps corresponding to the first matching condition in the following list must be run:

¶↪ **If both filters have a <filter-value-list> of same length without <url> and for each <filter-function> for which there is a corresponding item in each list**

Interpolate each <filter-function> pair following the rules in section Interpolation of Filter Functions.

¶↪ **If both filters have a <filter-value-list> of different length without <url> and for each <filter-function> for which there is a corresponding item in each list**

1. Append the missing equivalent <filter-function>s from the longer list to the end of the shorter list. The new added <filter-function>s must be initialized to their initial values for interpolation.
2. Interpolate each <filter-function> pair following the rules in section Interpolation of Filter Functions.

¶↪ **If one filter is ‘none’ and the other is a <filter-value-list> without <url>**

1. Replace ‘none’ with the corresponding <filter-value-list> of the other filter. The new <filter-function>s must be initialized to their initial values for interpolation.
2. Interpolate each <filter-function> pair following the rules in section Interpolation of Filter Functions.

¶↪ **Otherwise**
Use discrete interpolation.

ISSUE 4 Compute distance of filter functions. <https://github.com/w3c/csswg-drafts/issues/91>

§ 14.2. Addition

It is possible to combine independent animations of <filter-value-list>s [SVG11].

Given two filter values representing an base value (*base filter list*) and a value to add (*added filter list*), returns the concatenation of the the two lists: ‘*base filter list added filter list*’.

EXAMPLE 13
The following SVG animation has two `<animate>` elements animating ‘filter’ property of the `<rect>` element. Both specified animations are additive and have a duration of *10s*.

```
<rect width="200px" filter="none" ...>
  <animate attributeName="filter" from="blur(0px)" to="blur(10px)" dur="10s"
           additive="sum"/>
  <animate attributeName="filter" from="sepia(0)" to="sepia(1)" dur="10s"
           additive="sum"/>
</rect>
```

After *5s*, the used value of ‘filter’ is ‘blur(5px) sepia(0.5)’.

§ 14.3. Accumulation

Given two filter values V_a and V_b, returns the filter value, V_b.

§ 15. Privacy and Security Considerations

§ 15.1. Tainted Filter Primitives

It is important that the timing of any filter operation is independent of pixel values derived from the filtered content or other sources potentially containing privacy-sensitive information.

The following filter primitives may have access to pixel values that potentially contain privacy-sensitive information, either from the filtered object itself or other sources such as CSS styling. These primitives must be flagged as "tainted".

1. `<feFlood>` when the specified value of the ‘flood-color’ property computes to ‘currentColor’,
2. `<feDropShadow>` when the specified value value of the ‘flood-color’ property computes to ‘currentColor’,
3. `<feDiffuseLighting>`, when the specified value value of the ‘lighting-color’ property computes to ‘currentColor’
4. `<feSpecularLighting>` when the specified value value of the ‘lighting-color’ property computes to ‘currentColor’,
5. `<feImage>`, when the <url> reference points to an element or fetches a resource with the fetching mode *No-CORS* and
6. the filter primitives: SourceGraphic, SourceAlpha, BackgroundImage, BackgroundAlpha, FillPaint and StrokePaint.

`<feFlood>`, `<feDropShadow>`, `<feDiffuseLighting>` and `<feSpecularLighting>` are primitives with one or more CSS properties that take <color> as property value. <color> consists of (amongst others) the ‘currentColor’ keyword. The used value for ‘currentColor’ derives from the ‘color’ property. Since ‘color’ can be set by the ‘:visited’ pseudo selector, it potentially contains privacy-sensitive information and therefore these primitives must be marked as tainted.

`<feImage>` can reference cross-domain images as well as document fragments such as SVG graphics elements. These references potentially contain privacy-sensitive information and therefore the primitive must be marked as tainted.

The filter primitives SourceGraphic, SourceAlpha, BackgroundImage, BackgroundAlpha, FillPaint and StrokePaint either reference document fragments such as SVG graphics elements or style information that may derive directly or indirectly from the ‘color’ property. Therefore these primitives must be marked as tainted.

Every filter primitive that has a "tainted" flagged filter primitive as input must be flagged as "tainted" as well.

Filter operations must be implemented in such a way that they always take the same amount of time regardless of the pixel values if one of the input filter primitives is flagged as "tainted".

> Note: This specification aggravates the restrictions to filter primitives based on implementation feedback from user agents.

§ 15.2. `<feDisplacementMap>` Restrictions

If `<feDisplacementMap>` has a "tainted" flagged filter primitive as input and this input filter primitive is used as displacement map (referenced by `in2`), then feDisplacementMap must not proceed with the filter operation and acts as a pass through filter.

§ 15.3. Origin Restrictions

User agents must use the potentially CORS-enabled fetch method defined by the [HTML5] specification for the ‘filter’ property. When fetching, user agents must use "Anonymous" mode, set the referrer source to the stylesheet’s URL and set the origin to the URL of the containing document. If this results in network errors, the effect is as if the value ‘none’ had been specified.

§ 15.4. Timing Attacks

If any of the above rules are not followed, an attacker could infer information and mount a timing attack.

A timing attack is a method of obtaining information about content that is otherwise protected, based on studying the amount of time it takes for an operation to occur. If, for example, red pixels took longer to draw than green pixels, one might be able to reconstruct a rough image of the element being rendered, without ever having access to the content of the element. Security studies show that timing differences on arithmetic operations can be caused by the hardware architecture or compiler [ArTD].

§ Appendix A: The deprecated ‘enable-background’ property

SVG 1.1 introduced the ‘enable-background’ property [SVG11]. The property defined the back drop under the filter region at the time that the `<filter>` element was invoked. The concept defined by this property was identified to be incompatible with the model of stacking context in CSS at the time writing this specification. UAs can choose to implement the ‘enable-background’ property as defined in SVG 1.1 but will not be compatible to this specification or to CSS Compositing and Blending [COMPOSITING-1].

This specification does not support the ‘enable-background’ property. UAs must support the ‘isolation’ property instead [COMPOSITING-1].

§ Appendix B: DOM interfaces

§ Interface SVGFilterElement

The ***`SVGFilterElement`*** interface corresponds to the `<filter>` element.

```
interface SVGFilterElement : SVGElement {
  readonly attribute SVGAnimatedEnumeration filterUnits;
  readonly attribute SVGAnimatedEnumeration primitiveUnits;
  readonly attribute SVGAnimatedLength x;
  readonly attribute SVGAnimatedLength y;
  readonly attribute SVGAnimatedLength width;
  readonly attribute SVGAnimatedLength height;
};

SVGFilterElement includes SVGURIReference;
```

Attributes:

***`filterUnits`*, of type SVGAnimatedEnumeration, readonly**
Corresponds to attribute `filterUnits` on the given `<filter>` element. Takes one of the constants defined in SVGUnitTypes.

***`primitiveUnits`*, of type SVGAnimatedEnumeration, readonly**
Corresponds to attribute `primitiveUnits` on the given `<filter>` element. Takes one of the constants defined in SVGUnitTypes.

***`x`*, of type SVGAnimatedLength, readonly**
Corresponds to attribute `x` on the given `<filter>` element.

***`y`*, of type SVGAnimatedLength, readonly**
Corresponds to attribute `y` on the given `<filter>` element.

***`width`*, of type SVGAnimatedLength, readonly**
Corresponds to attribute `width` on the given `<filter>` element.

***`height`*, of type SVGAnimatedLength, readonly**
Corresponds to attribute `height` on the given `<filter>` element.

§ Interface SVGFilterPrimitiveStandardAttributes

```
interface mixin SVGFilterPrimitiveStandardAttributes {
  readonly attribute SVGAnimatedLength x;
  readonly attribute SVGAnimatedLength y;
  readonly attribute SVGAnimatedLength width;
  readonly attribute SVGAnimatedLength height;
  readonly attribute SVGAnimatedString result;
};
```

Attributes:

***`x`*, of type SVGAnimatedLength, readonly**
Corresponds to attribute `x` on the given element.

***`y`*, of type SVGAnimatedLength, readonly**
Corresponds to attribute `y` on the given element.

***`width`*, of type SVGAnimatedLength, readonly**
Corresponds to attribute `width` on the given element.

***`height`*, of type SVGAnimatedLength, readonly**
Corresponds to attribute `height` on the given element.

***`result`*, of type SVGAnimatedString, readonly**
Corresponds to attribute `result` on the given element.

§ Interface SVGFEBlendElement

The ***`SVGFEBlendElement`*** interface corresponds to the `<feBlend>` element.

```
interface SVGFEBlendElement : SVGElement {

  // 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;
  const unsigned short SVG_FEBLEND_MODE_OVERLAY = 6;
  const unsigned short SVG_FEBLEND_MODE_COLOR_DODGE = 7;
  const unsigned short SVG_FEBLEND_MODE_COLOR_BURN = 8;
  const unsigned short SVG_FEBLEND_MODE_HARD_LIGHT = 9;
  const unsigned short SVG_FEBLEND_MODE_SOFT_LIGHT = 10;
  const unsigned short SVG_FEBLEND_MODE_DIFFERENCE = 11;
  const unsigned short SVG_FEBLEND_MODE_EXCLUSION = 12;
  const unsigned short SVG_FEBLEND_MODE_HUE = 13;
  const unsigned short SVG_FEBLEND_MODE_SATURATION = 14;
  const unsigned short SVG_FEBLEND_MODE_COLOR = 15;
  const unsigned short SVG_FEBLEND_MODE_LUMINOSITY = 16;

  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedString in2;
  readonly attribute SVGAnimatedEnumeration mode;
};

SVGFEBlendElement includes SVGFilterPrimitiveStandardAttributes;
```

Constants in group “Blend Mode Types”:

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’.

SVG_FEBLEND_MODE_OVERLAY
: Corresponds to value ‘overlay’.

SVG_FEBLEND_MODE_COLOR_DODGE
: Corresponds to value ‘color-dodge’.

SVG_FEBLEND_MODE_COLOR_BURN
: Corresponds to value ‘color-burn’.

SVG_FEBLEND_MODE_HARD_LIGHT
: Corresponds to value ‘hard-light’.

SVG_FEBLEND_MODE_SOFT_LIGHT
: Corresponds to value ‘soft-light’.

SVG_FEBLEND_MODE_DIFFERENCE
: Corresponds to value ‘difference’.

SVG_FEBLEND_MODE_EXCLUSION
: Corresponds to value ‘exclusion’.

SVG_FEBLEND_MODE_HUE
: Corresponds to value ‘hue’.

SVG_FEBLEND_MODE_SATURATION

Corresponds to value ‘saturation’.

SVG_FEBLEND_MODE_COLOR
Corresponds to value ‘color’.

SVG_FEBLEND_MODE_LUMINOSITY
Corresponds to value ‘luminosity’.

Attributes:

***in1*, of type SVGAnimatedString, readonly**
Corresponds to attribute `in` on the given `<feBlend>` element.

***in2*, of type SVGAnimatedString, readonly**
Corresponds to attribute `in2` on the given `<feBlend>` element.

***mode*, of type SVGAnimatedEnumeration, readonly**
Corresponds to attribute `mode` on the given `<feBlend>` element. Takes one of the SVG_FEBLEND_MODE_* constants defined on this interface.

§ Interface SVGFEColorMatrixElement

The ***SVGFEColorMatrixElement*** interface corresponds to the `<feColorMatrix>` element.

```
interface SVGFEColorMatrixElement : SVGElement {

  // 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;
};

SVGFEColorMatrixElement includes SVGFilterPrimitiveStandardAttributes;
```

Constants in group “Color Matrix Types”:

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:

***in1*, of type SVGAnimatedString, readonly**
Corresponds to attribute `in` on the given `<feColorMatrix>` element.

***type*, of type SVGAnimatedEnumeration, readonly**
Corresponds to attribute `type` on the given `<feColorMatrix>` element. Takes one of the SVG_FECOLORMATRIX_TYPE_* constants defined on this interface.

***values*, of type SVGAnimatedNumberList, readonly**
Corresponds to attribute `values` on the given `<feColorMatrix>` element.

§ Interface SVGFEComponentTransferElement

The ***SVGFEComponentTransferElement*** interface corresponds to the `<feComponentTransfer>` element.

```
interface SVGFEComponentTransferElement : SVGElement {
  readonly attribute SVGAnimatedString in1;
};

SVGFEComponentTransferElement includes SVGFilterPrimitiveStandardAttributes;
```

Attributes:

***in1*, of type SVGAnimatedString, readonly**

Corresponds to attribute `in` on the given `<feComponentTransfer>` element.

§ Interface SVGComponentTransferFunctionElement

This interface defines a base interface used by the component transfer function interfaces.

```
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;
};
```

Constants in group “Component Transfer Types”:

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:

***type*, of type SVGAnimatedEnumeration, readonly**

Corresponds to attribute `in` on the given `<feComponentTransfer>` element. Takes one of the SVG_FECOMPONENTTRANSFER_TYPE_* constants defined on this interface.

***tableValues*, of type SVGAnimatedNumberList, readonly**

Corresponds to attribute `tableValues` on the given `<feComponentTransfer>` element. Takes one of the SVG_FECOLORMATRIX_TYPE_* constants defined on this interface.

***slope*, of type SVGAnimatedNumber, readonly**

Corresponds to attribute `slope` on the given `<feComponentTransfer>` element.

***`intercept`*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `intercept` on the given `<feComponentTransfer>` element.

***`amplitude`*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `amplitude` on the given `<feComponentTransfer>` element.

***`exponent`*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `exponent` on the given `<feComponentTransfer>` element.

***`offset`*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `offset` on the given `<feComponentTransfer>` element.

§ Interface SVGFEFuncRElement

The ***`SVGFEFuncRElement`*** interface corresponds to the `<feFuncR>` element.

```
interface SVGFEFuncRElement : SVGComponentTransferFunctionElement {
};
```

§ Interface SVGFEFuncGElement

The ***`SVGFEFuncGElement`*** interface corresponds to the `<feFuncG>` element.

```
interface SVGFEFuncGElement : SVGComponentTransferFunctionElement {
};
```

§ Interface SVGFEFuncBElement

The ***`SVGFEFuncBElement`*** interface corresponds to the `<feFuncB>` element.

```
interface SVGFEFuncBElement : SVGComponentTransferFunctionElement {
};
```

§ Interface SVGFEFuncAElement

The ***`SVGFEFuncAElement`*** interface corresponds to the `<feFuncA>` element.

```
interface SVGFEFuncAElement : SVGComponentTransferFunctionElement {
};
```

§ Interface SVGFECompositeElement

The ***`SVGFECompositeElement`*** interface corresponds to the `<feComposite>` element.

```
interface SVGFECompositeElement : SVGElement {

  // 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;
};

SVGFECompositeElement includes SVGFilterPrimitiveStandardAttributes;
```

Constants in group "Composite Operators":

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:

***in1*, of type SVGAnimatedString, readonly**
Corresponds to attribute `in` on the given `<feComposite>` element.

***in2*, of type SVGAnimatedString, readonly**
Corresponds to attribute `in2` on the given `<feComposite>` element.

***operator*, of type SVGAnimatedEnumeration, readonly**
Corresponds to attribute `operator` on the given `<feComposite>` element.

***k1*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `k1` on the given `<feComposite>` element.

***k2*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `k2` on the given `<feComposite>` element.

***k3*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `k3` on the given `<feComposite>` element.

***k4*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `k4` on the given `<feComposite>` element.

§ Interface SVGFEConvolveMatrixElement

The ***SVGFEConvolveMatrixElement*** interface corresponds to the `<feConvolveMatrix>` element.

```
interface SVGFEConvolveMatrixElement : SVGElement {

  // 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 SVGAnimatedString in1;
  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;
};

SVGFEConvolveMatrixElement includes SVGFilterPrimitiveStandardAttributes;
```

Constants in group “Edge Mode Values”:

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:

***in1*, of type SVGAnimatedString, readonly**

Corresponds to attribute `in` on the given `<feConvolveMatrix>` element.

***orderX*, of type SVGAnimatedInteger, readonly**

Corresponds to attribute `order` on the given `<feConvolveMatrix>` element.

***orderY*, of type SVGAnimatedInteger, readonly**

Corresponds to attribute `order` on the given `<feConvolveMatrix>` element.

***kernelMatrix*, of type SVGAnimatedNumberList, readonly**

Corresponds to attribute `kernelMatrix` on the given `<feConvolveMatrix>` element.

***divisor*, of type SVGAnimatedNumber, readonly**

Corresponds to attribute `divisor` on the given `<feConvolveMatrix>` element.

***bias*, of type SVGAnimatedNumber, readonly**

Corresponds to attribute `bias` on the given `<feConvolveMatrix>` element.

***targetX*, of type SVGAnimatedInteger, readonly**

Corresponds to attribute `targetX` on the given `<feConvolveMatrix>` element.

***targetY*, of type SVGAnimatedInteger, readonly**

Corresponds to attribute `targetY` on the given `<feConvolveMatrix>` element.

***edgeMode*, of type SVGAnimatedEnumeration, readonly**

Corresponds to attribute `edgeMode` on the given `<feConvolveMatrix>` element.

***kernelUnitLengthX*, of type SVGAnimatedNumber, readonly**

Corresponds to attribute `kernelUnitLength` on the given `<feConvolveMatrix>` element.

***kernelUnitLengthY*, of type SVGAnimatedNumber, readonly**

Corresponds to attribute `kernelUnitLength` on the given `<feConvolveMatrix>` element.

***preserveAlpha*, of type SVGAnimatedBoolean, readonly**

Corresponds to attribute `preserveAlpha` on the given `<feConvolveMatrix>` element.

§ Interface SVGFEDiffuseLightingElement

The ***`SVGFEDiffuseLightingElement`*** interface corresponds to the `<feDiffuseLighting>` element.

```
interface SVGFEDiffuseLightingElement : SVGElement {
  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber surfaceScale;
  readonly attribute SVGAnimatedNumber diffuseConstant;
  readonly attribute SVGAnimatedNumber kernelUnitLengthX;
  readonly attribute SVGAnimatedNumber kernelUnitLengthY;
};

SVGFEDiffuseLightingElement includes SVGFilterPrimitiveStandardAttributes;
```

Attributes:

***`in1`*, of type SVGAnimatedString, readonly**
Corresponds to attribute `in` on the given `<feDiffuseLighting>` element.

***`surfaceScale`*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `surfaceScale` on the given `<feDiffuseLighting>` element.

***`diffuseConstant`*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `diffuseConstant` on the given `<feDiffuseLighting>` element.

***`kernelUnitLengthX`*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `kernelUnitLength` on the given `<feDiffuseLighting>` element.

***`kernelUnitLengthY`*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `kernelUnitLength` on the given `<feDiffuseLighting>` element.

§ Interface SVGFEDistantLightElement

The ***`SVGFEDistantLightElement`*** interface corresponds to the `<feDistantLight>` element.

```
interface SVGFEDistantLightElement : SVGElement {
  readonly attribute SVGAnimatedNumber azimuth;
  readonly attribute SVGAnimatedNumber elevation;
};
```

Attributes:

***`azimuth`*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `azimuth` on the given `<feDistantLight>` element.

***`elevation`*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `elevation` on the given `<feDistantLight>` element.

§ Interface SVGFEPointLightElement

The ***`SVGFEPointLightElement`*** interface corresponds to the `<fePointLight>` element.

```
interface SVGFEPointLightElement : SVGElement {
  readonly attribute SVGAnimatedNumber x;
  readonly attribute SVGAnimatedNumber y;
  readonly attribute SVGAnimatedNumber z;
};
```

Attributes:

***`x`*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `x` on the given `<fePointLight>` element.

***`y`*, of type SVGAnimatedNumber, readonly**

Corresponds to attribute `y` on the given `<fePointLight>` element.

***`z`*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `z` on the given `<fePointLight>` element.

§ Interface SVGFESpotLightElement

The ***`SVGFESpotLightElement`*** interface corresponds to the `<feSpotLight>` element.

```
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:

***`x`*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `x` on the given `<feSpotLight>` element.

***`y`*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `y` on the given `<feSpotLight>` element.

***`z`*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `z` on the given `<feSpotLight>` element.

***`pointsAtX`*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `pointsAtX` on the given `<feSpotLight>` element.

***`pointsAtY`*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `pointsAtY` on the given `<feSpotLight>` element.

***`pointsAtZ`*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `pointsAtZ` on the given `<feSpotLight>` element.

***`specularExponent`*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `specularExponent` on the given `<feSpotLight>` element.

***`limitingConeAngle`*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `limitingConeAngle` on the given `<feSpotLight>` element.

§ Interface SVGFEDisplacementMapElement

The ***`SVGFEDisplacementMapElement`*** interface corresponds to the `<feDisplacementMap>` element.

```
interface SVGFEDisplacementMapElement : SVGElement {

  // 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;
};

SVGFEDisplacementMapElement includes SVGFilterPrimitiveStandardAttributes;
```

Constants in group "Channel Selectors":

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:

***in1*, of type SVGAnimatedString, readonly**

Corresponds to attribute `in` on the given `<feDisplacementMap>` element.

***in2*, of type SVGAnimatedString, readonly**

Corresponds to attribute `in2` on the given `<feDisplacementMap>` element.

***scale*, of type SVGAnimatedNumber, readonly**

Corresponds to attribute `scale` on the given `<feDisplacementMap>` element.

***xChannelSelector*, of type SVGAnimatedEnumeration, readonly**

Corresponds to attribute `xChannelSelector` on the given `<feDisplacementMap>` element. Takes one of the SVG_CHANNEL_* constants defined on this interface.

***yChannelSelector*, of type SVGAnimatedEnumeration, readonly**

Corresponds to attribute `yChannelSelector` on the given `<feDisplacementMap>` element. Takes one of the SVG_CHANNEL_* constants defined on this interface.

§ Interface SVGFEDropShadowElement

The ***SVGFEDropShadowElement*** interface corresponds to the `<feDropShadow>` element.

```
interface SVGFEDropShadowElement : SVGElement {
  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber dx;
  readonly attribute SVGAnimatedNumber dy;
  readonly attribute SVGAnimatedNumber stdDeviationX;
  readonly attribute SVGAnimatedNumber stdDeviationY;

  void setStdDeviation(float stdDeviationX, float stdDeviationY);
};

SVGFEDropShadowElement includes SVGFilterPrimitiveStandardAttributes;
```

Attributes:

***in1*, of type SVGAnimatedString, readonly**

Corresponds to attribute `in` on the given `<feDropShadow>` element.

***dx*, of type SVGAnimatedNumber, readonly**

Corresponds to attribute `dx` on the given `<feDropShadow>` element.

***dy*, of type SVGAnimatedNumber, readonly**

Corresponds to attribute `dy` on the given `<feDropShadow>` element.

***stdDeviationX*, of type SVGAnimatedNumber, readonly**

Corresponds to attribute `stdDeviation` on the given `<feDropShadow>` element. Contains the X component of attribute stdDeviation.

***stdDeviationY*, of type SVGAnimatedNumber, readonly**

Corresponds to attribute `stdDeviation` on the given `<feDropShadow>` element. Contains the Y component of attribute stdDeviation.

Methods:

setStdDeviation(stdDeviationX, stdDeviationY)

Sets the values for attribute `stdDeviation`.

stdDeviationX
The X component of attribute `stdDeviation`.

stdDeviationY
The Y component of attribute `stdDeviation`.

§ Interface SVGFEFloodElement

The ***SVGFEFloodElement*** interface corresponds to the `<feFlood>` element.

```
interface SVGFEFloodElement : SVGElement {
};

SVGFEFloodElement includes SVGFilterPrimitiveStandardAttributes;
```

§ Interface SVGFEGaussianBlurElement

The ***SVGFEGaussianBlurElement*** interface corresponds to the `<feGaussianBlur>` element.

```
interface SVGFEGaussianBlurElement : SVGElement {

  // 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 SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber stdDeviationX;
  readonly attribute SVGAnimatedNumber stdDeviationY;
  readonly attribute SVGAnimatedEnumeration edgeMode;

  void setStdDeviation(float stdDeviationX, float stdDeviationY);
};

SVGFEGaussianBlurElement includes SVGFilterPrimitiveStandardAttributes;
```

Constants in group “Edge Mode Values”:

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:

***in1*, of type SVGAnimatedString, readonly**
Corresponds to attribute `in` on the given `<feGaussianBlur>` element.

***stdDeviationX*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `stdDeviation` on the given `<feGaussianBlur>` element. Contains the X component of attribute stdDeviation.

***stdDeviationY*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `stdDeviation` on the given `<feGaussianBlur>` element. Contains the Y component of attribute stdDeviation.

***edgeMode*, of type SVGAnimatedEnumeration, readonly**
Corresponds to attribute `edgeMode` on the given `<feGaussianBlur>` element. Takes one of the SVG_EDGEMODE_* constants defined on this interface.

Methods:

setStdDeviation(stdDeviationX, stdDeviationY)
Sets the values for attribute `stdDeviation`.

stdDeviationX
The X component of attribute `stdDeviation`.

stdDeviationY
The Y component of attribute `stdDeviation`.

§ Interface SVGFEImageElement

The ***SVGFEImageElement*** interface corresponds to the `<feImage>` element.

```
interface SVGFEImageElement : SVGElement {
  readonly attribute SVGAnimatedPreserveAspectRatio preserveAspectRatio;
  readonly attribute SVGAnimatedString crossOrigin;
};

SVGFEImageElement includes SVGFilterPrimitiveStandardAttributes;
SVGFEImageElement includes SVGURIReference;
```

Attributes:

***preserveAspectRatio*, of type SVGAnimatedPreserveAspectRatio, readonly**
Corresponds to attribute `preserveAspectRatio` on the given `<feImage>` element.

***crossOrigin*, of type SVGAnimatedString, readonly**
The crossOrigin IDL attribute must reflect the `crossorigin` content attribute, limited to only known values.

§ Interface SVGFEMergeElement

The ***SVGFEMergeElement*** interface corresponds to the `<feMerge>` element.

```
interface SVGFEMergeElement : SVGElement {
};

SVGFEMergeElement includes SVGFilterPrimitiveStandardAttributes;
```

§ Interface SVGFEMergeNodeElement

The ***SVGFEMergeNodeElement*** interface corresponds to the `<feMergeNode>` element.

```
interface SVGFEMergeNodeElement : SVGElement {
  readonly attribute SVGAnimatedString in1;
};
```

Attributes:

***in1*, of type SVGAnimatedString, readonly**
Corresponds to attribute `in` on the given `<feMergeNode>` element.

§ Interface SVGFEMorphologyElement

The ***SVGFEMorphologyElement*** interface corresponds to the `<feMorphology>` element.

```
interface SVGFEMorphologyElement : SVGElement {

  // 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;
};

SVGFEMorphologyElement includes SVGFilterPrimitiveStandardAttributes;
```

Constants in group "Morphology Operators":

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:

***in1*, of type SVGAnimatedString, readonly**
: Corresponds to attribute `in` on the given `<feMorphology>` element.

***operator*, of type SVGAnimatedEnumeration, readonly**
: Corresponds to attribute `operator` on the given `<feMorphology>` element. Takes one of the SVG_MORPHOLOGY_OPERATOR_* constants defined on this interface.

***radiusX*, of type SVGAnimatedNumber, readonly**
: Corresponds to attribute `radius` on the given `<feMorphology>` element. Contains the X component of attribute radius.

***radiusY*, of type SVGAnimatedNumber, readonly**
: Corresponds to attribute `radius` on the given `<feMorphology>` element. Contains the Y component of attribute radius.

§ Interface SVGFEOffsetElement

The ***SVGFEOffsetElement*** interface corresponds to the `<feOffset>` element.

```
interface SVGFEOffsetElement : SVGElement {
  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber dx;
  readonly attribute SVGAnimatedNumber dy;
};

SVGFEOffsetElement includes SVGFilterPrimitiveStandardAttributes;
```

Attributes:

***in1*, of type SVGAnimatedString, readonly**
: Corresponds to attribute `in` on the given `<feOffset>` element.

***dx*, of type SVGAnimatedNumber, readonly**
: Corresponds to attribute `dx` on the given `<feOffset>` element.

***dy*, of type SVGAnimatedNumber, readonly**
: Corresponds to attribute `dy` on the given `<feOffset>` element.

§ Interface SVGFESpecularLightingElement

The ***SVGFESpecularLightingElement*** interface corresponds to the `<feSpecularLighting>` element.

```
interface SVGFESpecularLightingElement : SVGElement {
  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber surfaceScale;
  readonly attribute SVGAnimatedNumber specularConstant;
  readonly attribute SVGAnimatedNumber specularExponent;
  readonly attribute SVGAnimatedNumber kernelUnitLengthX;
  readonly attribute SVGAnimatedNumber kernelUnitLengthY;
};

SVGFESpecularLightingElement includes SVGFilterPrimitiveStandardAttributes;
```

Attributes:

***in1*, of type SVGAnimatedString, readonly**
Corresponds to attribute `in` on the given `<feSpecularLighting>` element.

***surfaceScale*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `surfaceScale` on the given `<feSpecularLighting>` element.

***specularConstant*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `specularConstant` on the given `<feSpecularLighting>` element.

***specularExponent*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `specularExponent` on the given `<feSpecularLighting>` element.

***kernelUnitLengthX*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `kernelUnitLength` on the given `<feSpecularLighting>` element.

***kernelUnitLengthY*, of type SVGAnimatedNumber, readonly**
Corresponds to attribute `kernelUnitLength` on the given `<feSpecularLighting>` element.

§ Interface SVGFETileElement

The ***SVGFETileElement*** interface corresponds to the `<feTile>` element.

```
interface SVGFETileElement : SVGElement {
  readonly attribute SVGAnimatedString in1;
};

SVGFETileElement includes SVGFilterPrimitiveStandardAttributes;
```

Attributes:

***in1*, of type SVGAnimatedString, readonly**
Corresponds to attribute `in` on the given `<feTile>` element.

§ Interface SVGFETurbulenceElement

The ***SVGFETurbulenceElement*** interface corresponds to the `<feTurbulence>` element.

```
interface SVGFETurbulenceElement : SVGElement {

  // 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;
};

SVGFETurbulenceElement includes SVGFilterPrimitiveStandardAttributes;
```

Constants in group “Turbulence Types”:

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’.

Constants in group “Stitch Options”:

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:

***baseFrequencyX*, of type SVGAnimatedNumber, readonly**

Corresponds to attribute `baseFrequency` on the given `<feTurbulence>` element. Contains the X component of the baseFrequency attribute.

***baseFrequencyY*, of type SVGAnimatedNumber, readonly**

Corresponds to attribute `baseFrequency` on the given `<feTurbulence>` element. Contains the Y component of the baseFrequency attribute.

***numOctaves*, of type SVGAnimatedInteger, readonly**

Corresponds to attribute `numOctaves` on the given `<feTurbulence>` element.

***seed*, of type SVGAnimatedNumber, readonly**

Corresponds to attribute `seed` on the given `<feTurbulence>` element.

***stitchTiles*, of type SVGAnimatedEnumeration, readonly**

Corresponds to attribute `stitchTiles` on the given `<feTurbulence>` element. Takes one of the SVG_TURBULENCE_TYPE_* constants defined on this interface.

***type*, of type SVGAnimatedEnumeration, readonly**

Corresponds to attribute `type` on the given `<feTurbulence>` element. Takes one of the SVG_STITCHTYPE_* constants defined on this interface.

§ Changes

The following significant changes were made since the 25 November 2014 Working Draft.

- Editorial changes.
- Add description elements to content model of all elements.
- Clarify that crossorigin is not animatable.
- <hue-rotate()> takes unitless zero.
- Allow author to change order of <color> and <length> values of <drop-shadow()>. Change grammar to put <color> first. Differentiate between initial value for interpolation and default values for omitted values.
- Define animation type of CSS properties.
- Make `<feColorMatrix>` and `<feConvolveMatrix>` a pass through on unfulfilled pre-conditions.
- Make `<feTurbulence>` algorithms respect uniformity.
- Apply properties that apply to all graphics elements to the use element as well.
- Corrected filter primitive representation of <saturate()>.
- Extend SVG DOM SVGFEBlendElement enumerations with new blend modes.

The following significant changes were made since the 26 November 2013 Working Draft.

- Removed Custom Filters.
- Allow the `<script>` element in the content model everywhere.
- Support all blend modes from CSS Blending specification for `<feBlend>`.
- Support all non-duplicated compositing modes from CSS Blending specification for `<feComposite>`.
- Added no-composite attribute to `<feBlend>` to avoid double compositing.
- Corrections on shorthands syntax.
- Added definition for shorthand filter regions.

The following significant changes were made since the 25 October 2012 Working Draft.

- Correction of brightness short hand filter.
- New syntax for Custom Filter function.
- Add at-function rule for Custom Filters.
- Allow Custom Filter function to be used as extension for future filter features.
- Remove unnecessary attributes and uniforms on shaders.
- Redefine origin of shader coordinate space to bottom left.
- Remove now unnecessary filter-margin properties.

See more detailed and longterm changes in the ChangeLog.

§ Acknowledgments

The editors would like to thank Robert O’Callahan, Coralie Mercier, Chris Lilley, Nikos Andronikos, Stephen Chenney, Simon Fraser, Tavmjong Bah, Robert Longson, Cameron McCormack, Brad Kemper, Tab Atkins, Brian Birtles, Michael Mullany, Rik Cabanier, Anne van Kesteren, Boris Zbarsky, Kristopher Giesing, Stephen White, Jasper van de Gronde, Kang-Hao Lu, Paul LeBeau, Debarshi Ray, Jarek Foksa, Sebastian Zartner, Yuqian Li, Amelia Bellamy-Royds and Max Vujovic for their careful reviews, comments, and corrections.

§ Conformance

§ Document conventions

Conformance requirements are expressed with a combination of descriptive assertions and RFC 2119 terminology. The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in the normative parts of this document are to be interpreted as described in RFC 2119. However, for readability, these words do not appear in all uppercase letters in this specification.

All of the text of this specification is normative except sections explicitly marked as non-normative, examples, and notes. [RFC2119]

Examples in this specification are introduced with the words “for example” or are set apart from the normative text with `class="example"`, like this:

> EXAMPLE 14
>
> This is an example of an informative example.

Informative notes begin with the word “Note” and are set apart from the normative text with `class="note"`, like this:

> Note, this is an informative note.

Advisements are normative sections styled to evoke special attention and are set apart from other normative text with `<strong class="advisement">`, like this:

UAs MUST provide an accessible alternative.

§ Conformance classes

Conformance to this specification is defined for three conformance classes:

style sheet
A CSS style sheet.

renderer
A UA that interprets the semantics of a style sheet and renders documents that use them.

authoring tool
A UA that writes a style sheet.

A style sheet is conformant to this specification if all of its statements that use syntax defined in this module are valid according to the generic CSS grammar and the individual grammars of each feature defined in this module.

A renderer is conformant to this specification if, in addition to interpreting the style sheet as defined by the appropriate specifications, it supports all the features defined by this specification by parsing them correctly and rendering the document accordingly. However, the inability of a UA to correctly render a document due to limitations of the device does not make the UA non-conformant. (For example, a UA is not required to render color on a monochrome monitor.)

An authoring tool is conformant to this specification if it writes style sheets that are syntactically correct according to the generic CSS grammar and the individual grammars of each feature in this module, and meet all other conformance requirements of style sheets as described in this module.

§ Requirements for Responsible Implementation of CSS

The following sections define several conformance requirements for implementing CSS responsibly, in a way that promotes interoperability in the present and future.

§ Partial Implementations

So that authors can exploit the forward-compatible parsing rules to assign fallback values, **CSS renderers *must* treat as invalid (and ignore as appropriate) any at-rules, properties, property values, keywords, and other syntactic constructs for which they have no usable level of support**. In particular, user agents *must not* selectively ignore unsupported property values and honor supported values in a single multi-value property declaration: if any value is considered invalid (as unsupported values must be), CSS requires that the entire declaration be ignored.

§ Implementations of Unstable and Proprietary Features

To avoid clashes with future stable CSS features, the CSSWG recommends following best practices for the implementation of unstable features and proprietary extensions to CSS.

§ Implementations of CR-level Features

Once a specification reaches the Candidate Recommendation stage, implementers should release an unprefixed implementation of any CR-level feature they can demonstrate to be correctly implemented according to spec, and should avoid exposing a prefixed variant of that feature.

To establish and maintain the interoperability of CSS across implementations, the CSS Working Group requests that non-experimental CSS renderers submit an implementation report (and, if necessary, the testcases used for that implementation report) to the W3C before releasing an unprefixed implementation of any CSS features. Testcases submitted to W3C are subject to review and correction by the CSS Working Group.

Further information on submitting testcases and implementation reports can be found from on the CSS Working Group's website at https://www.w3.org/Style/CSS/Test/. Questions should be directed to the public-css-testsuite@w3.org mailing list.

§ Index

§ Terms defined by this specification

§ Terms defined by reference

[COMPOSITING-1] defines the following terms:
- <blend-mode>
- backdrop
- color
- color-burn
- color-dodge
- darken
- difference
- exclusion
- hard-light
- hue
- isolation
- lighten
- luminosity
- multiply
- normal
- overlay
- saturation
- screen
- soft-light

[css-cascade-4] defines the following terms:
- specified value
- used value

[css-color-4] defines the following terms:
- <alpha-value>
- opacity
- transparent

[css-display-3] defines the following terms:
- containing block
- display

[css-fonts-3] defines the following terms:
- font
- font-family
- font-size
- font-stretch
- font-style
- font-variant
- font-weight

[css-fonts-4] defines the following terms:
- font-size-adjust

[css-masking-1] defines the following terms:
- clip
- clip-path
- clip-rule
- mask

[css-overflow-3] defines the following terms:
- overflow

[css-position-3] defines the following terms:
- stacking context

[css-text-3] defines the following terms:
- letter-spacing
- word-spacing

[css-text-decor-3] defines the following terms:
- text-decoration
- text-shadow

[css-transforms-1] defines the following terms:
- local coordinate system

[css-ui-3] defines the following terms:
- cursor
- outline

[css-values-4] defines the following terms:
- &&
- +
- ,
- <custom-ident>
- <length-percentage>
- <zero>
- ?
- calc()
- {a,b}
- |

[css-writing-modes-3] defines the following terms:
- direction
- unicode-bidi

[css-writing-modes-4] defines the following terms:
- writing-mode

[CSS21] defines the following terms:
- visibility

[css3-images] defines the following terms:
- <image>
- concrete object size
- image-rendering

[CSS3BG] defines the following terms:
- border
- border-image
- box-shadow

[CSS3COLOR] defines the following terms:

- <color>
- color

[CSS3VAL] defines the following terms:

- <angle>
- <integer>
- <length>
- <number-percentage>
- <number>
- <string>

[HTML] defines the following terms:

- browsing context
- img

[mediaqueries-5] defines the following terms:

- false

[selectors-4] defines the following terms:

- :visited

[SVG11] defines the following terms:

- alignment-baseline
- animate
- animatetransform
- baseline-shift
- dominant-baseline
- enable-background
- glyph-orientation-horizontal
- glyph-orientation-vertical
- kerning
- set
- simple alpha compositing

[svg12t] defines the following terms:

- unsupported

[SVG2] defines the following terms:

- SVGAnimatedBoolean
- SVGAnimatedEnumeration
- SVGAnimatedInteger
- SVGAnimatedLength
- SVGAnimatedNumber
- SVGAnimatedNumberList
- SVGAnimatedPreserveAspectRatio
- SVGAnimatedString
- SVGElement
- SVGURIReference
- SVGUnitTypes
- bounding box
- color-interpolation
- color-rendering
- container element
- defs
- desc
- descriptive element
- fill
- fill-opacity
- fill-rule
- graphics element
- image
- initial value
- marker
- marker-end
- marker-mid
- marker-start
- metadata
- object bounding box units
- pointer-events
- rect
- script
- shape-rendering
- stop-color
- stop-opacity
- stroke
- stroke-dasharray
- stroke-dashoffset
- stroke-linecap
- stroke-linejoin
- stroke-miterlimit
- stroke-opacity
- stroke-width
- text-anchor
- text-rendering
- title
- use

[WebIDL] defines the following terms:

- float
- unsigned short

§ References

§ Normative References

[COMPOSITING-1]
Rik Cabanier; Nikos Andronikos. Compositing and Blending Level 1. 13 January 2015. CR. URL: https://www.w3.org/TR/compositing-1/

[CSS-CASCADE-4]
Elika Etemad; Tab Atkins Jr.. CSS Cascading and Inheritance Level 4. 28 August 2018. CR. URL: https://www.w3.org/TR/css-cascade-4/

[CSS-COLOR-4]
Tab Atkins Jr.; Chris Lilley. CSS Color Module Level 4. 5 July 2016. WD. URL: https://www.w3.org/TR/css-color-4/

[CSS-DISPLAY-3]
Tab Atkins Jr.; Elika Etemad. CSS Display Module Level 3. 28 August 2018. CR. URL: https://www.w3.org/TR/css-display-3/

[CSS-FONTS-3]
John Daggett; Myles Maxfield; Chris Lilley. CSS Fonts Module Level 3. 20 September 2018. REC. URL: https://www.w3.org/TR/css-fonts-3/

[CSS-FONTS-4]
John Daggett; Myles Maxfield; Chris Lilley. CSS Fonts Module Level 4. 20 September 2018. WD. URL: https://www.w3.org/TR/css-fonts-4/

[CSS-MASKING-1]
Dirk Schulze; Brian Birtles; Tab Atkins Jr.. CSS Masking Module Level 1. 26 August 2014. CR. URL: https://www.w3.org/TR/css-masking-1/

[CSS-OVERFLOW-3]
David Baron; Elika Etemad; Florian Rivoal. CSS Overflow Module Level 3. 31 July 2018. WD. URL: https://www.w3.org/TR/css-overflow-3/

[CSS-POSITION-3]
Rossen Atanassov; Arron Eicholz. CSS Positioned Layout Module Level 3. 17 May 2016. WD. URL: https://www.w3.org/TR/css-position-3/

[CSS-TEXT-3]
Elika Etemad; Koji Ishii; Florian Rivoal. CSS Text Module Level 3. 12 December 2018. WD. URL: https://www.w3.org/TR/css-text-3/

[CSS-TEXT-DECOR-3]
Elika Etemad; Koji Ishii. CSS Text Decoration Module Level 3. 3 July 2018. CR. URL: https://www.w3.org/TR/css-text-decor-3/

[CSS-TRANSFORMS-1]
Simon Fraser; et al. CSS Transforms Module Level 1. 30 November 2018. WD. URL: https://www.w3.org/TR/css-transforms-1/

[CSS-UI-3]
Tantek Çelik; Florian Rivoal. CSS Basic User Interface Module Level 3 (CSS3 UI). 21 June 2018. REC. URL: https://www.w3.org/TR/css-ui-3/

[CSS-VALUES-4]
Tab Atkins Jr.; Elika Etemad. CSS Values and Units Module Level 4. 10 October 2018. WD. URL: https://www.w3.org/TR/css-values-4/

[CSS-WRITING-MODES-3]
Elika Etemad; Koji Ishii. CSS Writing Modes Level 3. 24 May 2018. CR. URL: https://www.w3.org/TR/css-writing-modes-3/

[CSS-WRITING-MODES-4]
Elika Etemad; Koji Ishii. CSS Writing Modes Level 4. 24 May 2018. CR. URL: https://www.w3.org/TR/css-writing-modes-4/

[CSS21]
Bert Bos; et al. Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. 7 June 2011. REC. URL: https://www.w3.org/TR/CSS2/

[CSS3-IMAGES]
Elika Etemad; Tab Atkins Jr.. CSS Image Values and Replaced Content Module Level 3. 17 April 2012. CR. URL: https://www.w3.org/TR/css3-images/

[CSS3BG]

Bert Bos; Elika Etemad; Brad Kemper. CSS Backgrounds and Borders Module Level 3. 17 October 2017. CR. URL: https://www.w3.org/TR/css-backgrounds-3/

[CSS3COLOR]
Tantek Çelik; Chris Lilley; David Baron. CSS Color Module Level 3. 19 June 2018. REC. URL: https://www.w3.org/TR/css-color-3/

[CSS3VAL]
Tab Atkins Jr.; Elika Etemad. CSS Values and Units Module Level 3. 14 August 2018. CR. URL: https://www.w3.org/TR/css-values-3/

[HTML]
Anne van Kesteren; et al. HTML Standard. Living Standard. URL: https://html.spec.whatwg.org/multipage/

[HTML5]
Ian Hickson; et al. HTML5. 27 March 2018. REC. URL: https://www.w3.org/TR/html5/

[MEDIAQUERIES-5]
Media Queries Level 5 URL: https://drafts.csswg.org/mediaqueries-5/

[RFC2119]
S. Bradner. Key words for use in RFCs to Indicate Requirement Levels. March 1997. Best Current Practice. URL: https://tools.ietf.org/html/rfc2119

[SELECTORS-4]
Elika Etemad; Tab Atkins Jr.. Selectors Level 4. 21 November 2018. WD. URL: https://www.w3.org/TR/selectors-4/

[SVG11]
Erik Dahlström; et al. Scalable Vector Graphics (SVG) 1.1 (Second Edition). 16 August 2011. REC. URL: https://www.w3.org/TR/SVG11/

[SVG2]
Amelia Bellamy-Royds; et al. Scalable Vector Graphics (SVG) 2. 4 October 2018. CR. URL: https://www.w3.org/TR/SVG2/

[WebIDL]
Cameron McCormack; Boris Zbarsky; Tobie Langel. Web IDL. 15 December 2016. ED. URL: https://heycam.github.io/webidl/

§ Informative References

[ArTD]
B. Jacob. Arithmetic Timing Differences. URL: https://wiki.mozilla.org/User:Bjacob/ArithmeticTimingDifferences

[Cmam]
IEC. IEC 61966-2-1:1999 Colour measurement and management - Part 2-1: Colour management - Default RGB colour space - sRGB. URL: https://webstore.iec.ch/publication/6169

[CSS3-ANIMATIONS]
Dean Jackson; et al. CSS Animations Level 1. 11 October 2018. WD. URL: https://www.w3.org/TR/css-animations-1/

[PORTERDUFF]
Thomas Porter; Tom Duff. Compositing digital images.

[TaM]
Ebert et al, AP Professional. Texturing and Modeling. 1994.

§ Property Index

Name	Value	Initial	Applies to	Inh.	%ages	Animat-able	Canonical order	Computed value	Media
‘color-interpolation-filters’	auto | sRGB | linearRGB	linearRGB	All filter primitives	yes	n/a	no	per grammar	as specified	visual
‘filter’	none | <filter-value-list>	none	All elements. In SVG, it applies to container elements without the defs element, all graphics elements and the use element.	no	n/a	See prose in Animation of Filters.	per grammar	as specified	visual

Name	Value	Initial	Applies to	Inh.	%ages	Animat-able	Canonical order	Computed value	Media
‘flood-color’	<color>	black	feFlood and feDropShadow elements	no	n/a	as color	per grammar	as specified	visual
‘flood-opacity’	<alpha-value>	1	feFlood and feDropShadow elements	no	n/a	as number or percentage	per grammar	the specified value converted to a number, clamped to the range [0,1]	visual
‘lighting-color’	<color>	white	feDiffuseLighting and feSpecularLighting elements	no	n/a	as color	per grammar	as specified	visual

§ IDL Index

```
interface SVGFilterElement : SVGElement {
  readonly attribute SVGAnimatedEnumeration filterUnits;
  readonly attribute SVGAnimatedEnumeration primitiveUnits;
  readonly attribute SVGAnimatedLength x;
  readonly attribute SVGAnimatedLength y;
  readonly attribute SVGAnimatedLength width;
  readonly attribute SVGAnimatedLength height;
};

SVGFilterElement includes SVGURIReference;

interface mixin SVGFilterPrimitiveStandardAttributes {
  readonly attribute SVGAnimatedLength x;
  readonly attribute SVGAnimatedLength y;
  readonly attribute SVGAnimatedLength width;
  readonly attribute SVGAnimatedLength height;
  readonly attribute SVGAnimatedString result;
};

interface SVGFEBlendElement : SVGElement {

  // 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;
  const unsigned short SVG_FEBLEND_MODE_OVERLAY = 6;
  const unsigned short SVG_FEBLEND_MODE_COLOR_DODGE = 7;
  const unsigned short SVG_FEBLEND_MODE_COLOR_BURN = 8;
  const unsigned short SVG_FEBLEND_MODE_HARD_LIGHT = 9;
  const unsigned short SVG_FEBLEND_MODE_SOFT_LIGHT = 10;
  const unsigned short SVG_FEBLEND_MODE_DIFFERENCE = 11;
  const unsigned short SVG_FEBLEND_MODE_EXCLUSION = 12;
  const unsigned short SVG_FEBLEND_MODE_HUE = 13;
  const unsigned short SVG_FEBLEND_MODE_SATURATION = 14;
  const unsigned short SVG_FEBLEND_MODE_COLOR = 15;
  const unsigned short SVG_FEBLEND_MODE_LUMINOSITY = 16;

  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedString in2;
  readonly attribute SVGAnimatedEnumeration mode;
};

SVGFEBlendElement includes SVGFilterPrimitiveStandardAttributes;

interface SVGFEColorMatrixElement : SVGElement {

  // 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;
};

SVGFEColorMatrixElement includes SVGFilterPrimitiveStandardAttributes;

interface SVGFEComponentTransferElement : SVGElement {
  readonly attribute SVGAnimatedString in1;
};

SVGFEComponentTransferElement includes SVGFilterPrimitiveStandardAttributes;

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;
};

interface SVGFEFuncRElement : SVGComponentTransferFunctionElement {
};

interface SVGFEFuncGElement : SVGComponentTransferFunctionElement {
};

interface SVGFEFuncBElement : SVGComponentTransferFunctionElement {
};

interface SVGFEFuncAElement : SVGComponentTransferFunctionElement {
};

interface SVGFECompositeElement : SVGElement {

  // 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;
};

SVGFECompositeElement includes SVGFilterPrimitiveStandardAttributes;

interface SVGFEConvolveMatrixElement : SVGElement {

  // 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 SVGAnimatedString in1;
  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;
};

SVGFEConvolveMatrixElement includes SVGFilterPrimitiveStandardAttributes;

interface SVGFEDiffuseLightingElement : SVGElement {
  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber surfaceScale;
  readonly attribute SVGAnimatedNumber diffuseConstant;
  readonly attribute SVGAnimatedNumber kernelUnitLengthX;
  readonly attribute SVGAnimatedNumber kernelUnitLengthY;
};

SVGFEDiffuseLightingElement includes SVGFilterPrimitiveStandardAttributes;

interface SVGFEDistantLightElement : SVGElement {
  readonly attribute SVGAnimatedNumber azimuth;
  readonly attribute SVGAnimatedNumber elevation;
};

interface SVGFEPointLightElement : SVGElement {
  readonly attribute SVGAnimatedNumber x;
  readonly attribute SVGAnimatedNumber y;
  readonly attribute SVGAnimatedNumber z;
};

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;
};

interface SVGFEDisplacementMapElement : SVGElement {

  // 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;
};
```

```
SVGFEDisplacementMapElement includes SVGFilterPrimitiveStandardAttributes;

interface SVGFEDropShadowElement : SVGElement {
  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber dx;
  readonly attribute SVGAnimatedNumber dy;
  readonly attribute SVGAnimatedNumber stdDeviationX;
  readonly attribute SVGAnimatedNumber stdDeviationY;

  void setStdDeviation(float stdDeviationX, float stdDeviationY);
};

SVGFEDropShadowElement includes SVGFilterPrimitiveStandardAttributes;

interface SVGFEFloodElement : SVGElement {
};

SVGFEFloodElement includes SVGFilterPrimitiveStandardAttributes;

interface SVGFEGaussianBlurElement : SVGElement {

  // 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 SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber stdDeviationX;
  readonly attribute SVGAnimatedNumber stdDeviationY;
  readonly attribute SVGAnimatedEnumeration edgeMode;

  void setStdDeviation(float stdDeviationX, float stdDeviationY);
};

SVGFEGaussianBlurElement includes SVGFilterPrimitiveStandardAttributes;

interface SVGFEImageElement : SVGElement {
  readonly attribute SVGAnimatedPreserveAspectRatio preserveAspectRatio;
  readonly attribute SVGAnimatedString crossOrigin;
};

SVGFEImageElement includes SVGFilterPrimitiveStandardAttributes;
SVGFEImageElement includes SVGURIReference;

interface SVGFEMergeElement : SVGElement {
};

SVGFEMergeElement includes SVGFilterPrimitiveStandardAttributes;

interface SVGFEMergeNodeElement : SVGElement {
  readonly attribute SVGAnimatedString in1;
};

interface SVGFEMorphologyElement : SVGElement {

  // 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;
};

SVGFEMorphologyElement includes SVGFilterPrimitiveStandardAttributes;

interface SVGFEOffsetElement : SVGElement {
  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber dx;
  readonly attribute SVGAnimatedNumber dy;
};

SVGFEOffsetElement includes SVGFilterPrimitiveStandardAttributes;

interface SVGFESpecularLightingElement : SVGElement {
  readonly attribute SVGAnimatedString in1;
  readonly attribute SVGAnimatedNumber surfaceScale;
  readonly attribute SVGAnimatedNumber specularConstant;
  readonly attribute SVGAnimatedNumber specularExponent;
  readonly attribute SVGAnimatedNumber kernelUnitLengthX;
  readonly attribute SVGAnimatedNumber kernelUnitLengthY;
};

SVGFESpecularLightingElement includes SVGFilterPrimitiveStandardAttributes;

interface SVGFETileElement : SVGElement {
  readonly attribute SVGAnimatedString in1;
};

SVGFETileElement includes SVGFilterPrimitiveStandardAttributes;

interface SVGFETurbulenceElement : SVGElement {

  // 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;
};

SVGFETurbulenceElement includes SVGFilterPrimitiveStandardAttributes;
```

§ Issues Index

ISSUE 1 How does filter behave on fixed background images? <https://github.com/w3c/csswg-drafts/issues/238> ↵

ISSUE 2 How to behave on invalid number of entries in the value list? <https://github.com/w3c/csswg-drafts/issues/237> ↵

ISSUE 3 Implementations do not match specification. <https://github.com/w3c/csswg-drafts/issues/113> ↵

ISSUE 4 Compute distance of filter functions. <https://github.com/w3c/csswg-drafts/issues/91> ↵

↥