Adobe Premiere® 5.0 for Windows® 95/Windows NT® and Mac OS

Expressions for Creating Transitions and Filters

The Adobe Premiere program lets you create your own transitions and filters. This file explains how to apply a custom transition or filter effect and how to set up the arithmetic expressions that describe what the transition or filter will do.

Creating custom transitions and filters

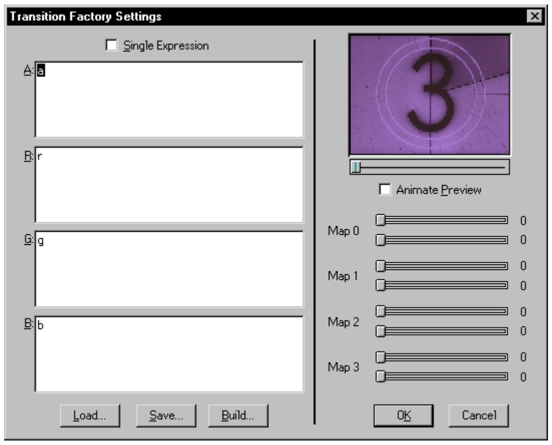
The following procedures describe how to apply a custom transition or filter in Premiere. For more information on how transitions and filters work, refer to the *Adobe Premiere 5.0 User Guide*, chapters 5 and 10.

To create a custom transition or filter:

1 Do one of the following:

• To create a transition, drag the Transition Factory transition from the Transition palette to the Transition track in the Timeline. The Transition Factory Settings dialog box appears.

• To create a filter, select the clip you want to modify, choose Clip > Filters, and double-click Filter Factory in the Available column. The Filter Factory Settings dialog box appears.



2 Specify the expressions as follows:

• To specify an expression in the alpha channel, select Single Expression and type the expression in the A field. The evaluation of the alpha channel expression will be applied to each of the other three channels: R, G, and B. When the same value is applied to each channel of the pixels in an image, the image will be a grayscale image.

• To specify separate expressions for the R, G, and B channels, make sure that the Single Expression option is deselected and type the expressions in the R, G, and B fields. Even if you specify the same expression in all three channels, their evaluations will probably be different.

As you type an expression, a small yellow caution sign appears. It will remain visible until you have typed a legal expression. If the caution sign does not disappear, it means that there is an error in the expression. To see which part of the expression is in error, click the caution sign to select the incorrect portion.

3 If the expressions include user-supplied slider information, drag the appropriate Map sliders to preview the effects. The Map 0 sliders correspond to sliders 0 and 1; the Map 1 sliders correspond to sliders 2 and 3; and so on.

4 When you have correctly set up the effect you want, click Save to save the expressions in a text file. Saving the expression allows you to load and edit the transition or filter in the future. You should give the text file the same name that you plan to give the transition or filter, but save it in a folder other than the Adobe Premiere Plug-Ins folder.

5 If you want to use this one instance of the transition or filter only, click OK to apply the effect. If you want to use the effect more than once, see the next procedure.

To save a custom transition for additional use:

1 Follow steps 1 through 4 of the previous procedure.

2 Click Build.

3 (Windows only) Specify a unique module name with 8 characters or less in the DLL Module Name field.

4 Specify a name for the effect in the Title field. The title will appear in the Transition palette or in the Filters dialog box after you restart Adobe Premiere.

5 Use the Author field to include credits or copyright information, or a description of the effect.

6 If the expressions include user-supplied slider information, select the appropriate number of Slider or Map options and specify labels for the sliders in the corresponding text boxes. The labels will appear with the sliders in the Transition Setting or Filter Settings dialog box.

To display the sliders individually in the Settings dialog box, use the Slider options. To display the sliders in pairs, use the Map options. Whether you should use individual or paired sliders depends on the type of effect you are defining.

7 Click OK. A standard Save dialog box appears. Save the effect in the Adobe Premiere Plug-Ins folder. The custom module is saved in the Adobe Premiere *plugins* directory with the module name you specified and (on Windows only) with the *.pff* file extension.

8 To make the effect available to users, restart the Adobe Premiere program.

To edit a custom transition or filter:

1 Do one of the following:

• Double-click the Transition Factory transition in the Transition palette, and click Custom.

• Choose Clip > Filters, double-click the Filter Factory filter, and then click Custom.

2 Click Load. Use the Open dialog box to load the text file containing the effect's expressions. You must have saved the expressions in a text file when you created the transition or filter to be able to edit it.

Follow the steps in the previous two procedures to edit and rebuild the transition

About digital images

A digital video image is a conglomeration of tiny picture elements, called *pixels*. Pixels project the color and brightness of the image. Each pixel in an image is uniquely identified by its coordinates. The first coordinate is the horizontal position of the pixel, and the second coordinate is the vertical position of the pixel. The horizontal coordinates start counting at the left edge of the image and increase as you move to the right. The vertical coordinates start counting at the top of the image and increase as you move down. Therefore, the top left corner of the image has the coordinates (0,0). The range of coordinates for an image depends on its resolution.

In RGB format, the color of a pixel is stored as three numbers: the amount of red, the amount of green, and the amount of blue. The three color values are called *channels*. Channel values can range from 0 to 255.

• If a channel value is set to 0, none of its color is present in the pixel.

• If a channel value is set to 255, the maximum amount of that color is present in the pixel.
For example, if a pixel has the channel values (255,0,0), the pixel is entirely red: 255 red, 0 green, 0 blue.

• If all three channels have the same value, the pixel is a shade of gray. For example, (80,80,80) is a dark gray, (128,128,128) is a medium gray, and (200,200,200) is a light gray.

• If all three channels are 0, the pixel is black. If all three channels are 255, the pixel is white.

The transitions and filters you create affect the channel values of the pixels in an image. You specify an expression for each channel, and each operation is performed on the appropriate channel for every pixel in the image. Expressions can include specific pixel coordinates whose channel values are evaluated and used in the calculation.

***Note:** If an expression evaluates to a number greater than 255, the channel is set to 255. Likewise, if an expression evaluates to a number less than 0, the channel is set to 0.*

Components of expressions

A transition or filter performs an operation on the channels of each pixel in an image. These channel operations are described by arithmetic expressions. Expressions are made up of combinations of four types of components: constants, variables, functions, and operators. The following sections describe these four components. For a quick reference of all allowable variables, functions, and operators, see "Expression Reference" on page 11.

The Transition Factory and Filter Factory allow only integer numbers in expressions—no fractions or decimal numbers are allowed. Variables and functions will always evaluate to integers.

Constants

A *constant* is a number that is supplied directly in the expression. Constants can be used to construct simple expressions such as 10+5, and these expressions can always be replaced by another constant; in this case, 15.

Constants can also be written in hexadecimal form. To use hexadecimal values, prefix the number by 0x, as in 0xaf10.

Variables

A *variable* is a short name, such as *x*, that can be evaluated. The value of a variable depends on the current image, the current pixel, or the current channel for which an expression is being evaluated. For example, the variable *x* always evaluates to the horizontal coordinate of the current pixel, and the variable *y* always evaluates to the vertical coordinate. The variables for the current pixel's channel values are r (red), g (green), and b (blue).

If you are creating a transition, you specify whether you are affecting the first image or the second image. To do so, include a 0 or a 1 with the variable. For example, *r*0 is the red channel value for the current pixel in the first image, whereas *r*1 is the red channel value for the current pixel in the second image.

You can combine variables and constants to form expressions. For example, the expression *r*+*g* retrieves the red and green channel values for the current pixel and adds them together.

Functions

Functions are short names that can be evaluated, and they require one or more arguments. For example, the rnd function requires two arguments. It evaluates to a number that is greater than or equal to the first argument and less than or equal to the second argument. The expression rnd(1,10) evaluates to a number between 1 and 10, inclusive. (The rnd function is called a random number generator, and it is useful for adding noise or texture to an image.)

Arguments are written within parenthesis and are separated by commas. The arguments can be expressions. For example, the expression rnd(*r*–10,*r*+10) evaluates to the red channel of the current pixel, plus or minus 10.

Another function, src (source), retrieves channel values for a particular pixel. It requires three arguments: the horizontal coordinate of the pixel, the vertical coordinate, and the channel index. (The index for the red channel is 0; the green channel is 1; the blue channel is 2.) For example, the expression src(10,20,0) retrieves the red channel value for the pixel at coordinates (10,20). The expression src(*x*,*y*,0) retrieves the red channel value for the current pixel. The expression src(*x*+1,*y*,0) retrieves the red channel for the pixel to the right of the current pixel.

The remaining available functions are described in the section "Expression Reference" later in this file.

Operators

The *operators* include all of the arithmetic that can be used in an expression. There are five types of operators: basic, relational, logical, conditional, and bitwise.

• The basic operators are +, –, *, /, and %. The % (modulo) operator calculates the remainder of a division. For example, the expression 11%3 evaluates to 2.

• Relational operators compare two expressions and evaluate to 0 (false) or 1 (true). For example, the < operator evaluates to 1 if the expression on the left is less than the expression on the right. The expression r<g evaluates to 1 when the red channel of the current pixel has a lower value than the green channel. Otherwise, it evaluates to 0. The set of relational operators includes <, <=, >, >=, ==, and !=.

The == operator evaluates to 1 when the two expressions surrounding it evaluate to the same thing. The != operator evaluates to 1 when the two expressions surrounding it evaluate to something different. For example, the expression 1==1 evaluates to 1, and the expression 2==1 evaluates to 0. The expression 1!=2 evaluates to 1, and the expression 1!=1 evaluates to 0.

• Logical operators let you combine several relational expressions. For example, you could evaluate whether the horizontal coordinate of a pixel is between 10 and 30, inclusive. The appropriate relational expressions are *x*>=10 and *x*<=30. You can use the logical && operator to combine them into the single expression (*x*>=10)&&(*x*<=30). The && operator evaluates the expressions on both sides. If neither expression evaluates to 0, the && operator evaluates to 1. If either expression evaluates to 0, the && operator evaluates to 0.

The logical operator || is similar to the operator &&, but it performs a slightly different logical operation. The || operator is also placed between two relational expressions. If either of the expressions evaluate to anything but 0, the || operator evaluates to 1. If both expressions evaluate to 0, the || operator evaluates to 0. For example, the expression (*x*>10||(*y*>10) evaluates to 1 when the horizontal coordinate is greater than 10. The only time this expression evaluates to 0 is when the horizontal coordinate is <=10 and the vertical coordinate is <=10. The following chart shows the difference between the && and || operators:

LEFT EXPRESSION	RIGHT EXPRESSION	LEFT&&RIGHT	LEFT||RIGHT
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Finally, you place the ! operator before an expression to invert the expression’s evaluation. If the expression evaluates to 0, the ! operator evaluates to 1. If the expression evaluates to anything but 0, the ! operator evaluates to 0.

• The single conditional operator ? lets you make a choice between two alternatives. A conditional expression includes a condition expression and two alternative expressions. The conditional operator evaluates the condition expression and uses the result to decide which of the two alternatives it should evaluate. If the conditional expression evaluates to anything but 0, the first alternative is evaluated. If the conditional expression evaluates to 0, the second alternative is evaluated.

For example, in the expression (*x*%2)?r:g, the ? conditional operator separates the condition expression (*x*%2) from the two alternative expressions *r* and *g*. The alternative expressions are separated by a colon (:). The condition expression divides the horizontal coordinate of the current pixel by 2 and returns the remainder of the division. If the horizontal coordinate is an odd number, the result is something other than 0, and if the horizontal coordinate is an even number, the result is 0. Therefore, if the pixel has an odd horizontal coordinate, the conditional operator returns the value of the red channel. If the current pixel has an even horizontal coordinate, the conditional operator returns the value of the green channel.

• Bitwise operators directly manipulate the bits in a value. The bitwise operators include &, |, ^, ~, <<, and >>. You place the &, |, and ^ operators between two expressions. The & operator performs a logical-and operation on the corresponding bits of the evaluated expressions; the | operator performs a logical-or; and the ^ operator performs a logical-exclusive-or. The ~ operator takes only one expression, and it performs a logical-not on each bit of the evaluated expression.

The << and >> expressions are placed between two expressions. Both operators shift the bits in the left expression's evaluation by some number, which is specified by the right expression's evaluation. The << operator shifts bits to the left. The >> operator shifts bits to the right.

Providing user-controlled sliders

When you create a transition or filter, you can provide up to eight slider controls for the user to adjust when applying the effect. Slider values can range from 0 to 255. You set up the effect's sliders by using the ctl (control), val (value), and map (mapping) functions in your expressions.

If you use the these functions to retrieve slider information, you should set up the Slider or Map options in the Transition Factory's or Filter Factory's Build Custom dialog box. For information on using the Build Custom Transition dialog box, see "To create a custom transition or filter:" on page 1.

• The ctl function retrieves the specified sliders current value. This function requires one argument: the slider control index, which is a number between 0 and 7. For example, the expression ctl(0) evaluates the current value of the first slider.

• The val function converts the range of possible slider values (always 0 to 255) into a range that you specify. For example, to get a value between 1 and 100 from a slider, you would use the expression val(0,1,100). If slider 0 is set to 0, the slider value evaluates to 0. If slider 0 is set to 255, the slider value evaluates to 100. Slider values between 0 and 255 are converted into values between 1 and 100.

• The map function groups the sliders into pairs. Each even/odd slider pair sets the values in a table, which is accessed by the map function. There are four mapping tables—one for each slider pair. Sliders 0 and 1 set the values for mapping table 0; sliders 2 and 3 set the values for mapping table 1, and so on. Each table contains 256 entries, which are calculated each time the slider values change.

The map function takes two arguments: the table index and the item index. For example, the expression map(1,20) returns item 20 from table 1. The table index must be between 0 and 3. The item index must be between 0 and 255, inclusive.

Examples

This section provides several examples of using expressions to achieve a result. The examples are presented in order of complexity. The Adobe Premiere program also provides some sample transition and filter expressions. These samples have been saved as text files and are located in a folder in your Adobe Premiere folder. You can use the Transition Factory or Filter Factory to load a sample file and observe its effect.

Affecting a single channel (filter)

To make an image more red, you could use the following expressions:

A: 0
R: r+100
G: g
B: b

The first expression evaluates the red channel of each pixel and adds 100 to each one. The next two expressions evaluate the other two channels and leave them unchanged.

Affecting channels using sliders (filter)

To add a user-controlled slider value to the current channel values, you could use the following expressions:

A: 0
R: r+ctl(0)
G: g+ctl(1)
B: b+ctl(2)

The first expression evaluates the red channel of each pixel and adds the value of slider 0 to each one. The next two expressions do the same thing to the green and blue channels, using the values of slider 1 and slider 2, respectively.

Adding noise to channels using slider and random values (filter)

To use slider values to determine the range of random numbers, you could use the following expressions:

A: 0
R: r+rnd(–ctl(0),ctl(0))
G: g+rnd(–ctl(1),ctl(1))
B: b+rnd(–ctl(2),ctl(2))

The filter defined by these expressions adds noise to all three channels. The amount of noise in each channel is determined by the slider controls. The first expression evaluates the slider setting from slider 0. This value is used as the argument for the rnd function. If the slider setting is 0, the rnd function evaluates to 0. If the slider setting is 100, the rnd function can return any number between –100 and 100, inclusive. The result of the rnd function is then added to the current value of the red channel. As the slider setting is raised from 0 to 255, the random numbers are selected from a wider and wider range, resulting in more and more noise being added to the red channel.

The next two expressions perform the same operation on the green and blue channel, using sliders 1 and 2 for input, respectively.

Amplifying or toning down channels (filter)

To amplify or tone down a channel based on the values of a different channel, you could use the following expressions:

A: 0
R: (b>100)?r+50:r–50
G: g
B: b

The first expression evaluates the blue channel to determine if it is greater than 100. If it is greater than 100, the entire expression evaluates to the red channel value plus 50. If it not greater than 100, the expression evaluates to the red channel value minus 50. The other two expressions do nothing. Therefore, this filter amplifies the red channel if there is more blue than you want, or it tones down the red channel if there is less blue than you want.

You could also use slider values in a similar type of filter, as follows:

A: 0
R: (b>ctl(0))?r+ctl(1):r–ctl(1)
G: g
B: (b>ctl(0))?b–ctl(1):b+ctl(1)

The first expression uses the setting from slider 0 as a "cutoff" value. If the blue channel is greater than the cutoff value, the red channel is amplified by the value of slider 1. If the blue channel is less than the cutoff value, the red channel is toned down by the value of slider 1. The third expression is the opposite of the first one, but it works on the blue channel instead of the red channel. The effect is that anything that is added to the red channel is subtracted from the blue channel, and vice versa.

Dissolving between two images (transition)

To create a simple cross dissolve between the first clip and the second clip, you could use the following expressions:

A: mix(c1,c0,t,total)
R: mix(c1,c0,t,total)
G: mix(c1,c0,t,total)
B: mix(c1,c0,t,total)

The first expression combines the current value of the alpha channel in the first clip with the current value of the alpha channel in the second clip by using the fraction time/total time. The remaining three expressions do the same thing for the current values of the R, G, and B channels.

Averaging the channel values of neighboring pixels (filter)

You can use the src (source) function to retrieve the channel values from neighboring pixels and average them, as follows:

A: 0
R: (src(x–1,y,0)+src(x,y,0)+src(x+1,y,0))/3
G: (src(x–1,y,1)+src(x,y,1)+src(x+1,y,1))/3
B: (src(x–1,y,2)+src(x,y,2)+src(x+1,y,2))/3

The first expression uses the src function to retrieve the red channel value for three different pixels: the pixel to the left of the current pixel, the current pixel, and the pixel to the right of the current pixel. These three values are added together and then divided by 3. The next two expressions do the same thing using the green and blue channels, respectively.

Pushing the second image into the first image (transition)

To create a transition that would "push" the second image into the first image from left to right, you could use the following expressions:

A: 0
R: x*1024/xmax<t*1024/total?src0(xmax+x–t*xmax/total,y,p):src1(x–t*xmax/total,y,p)
G: x*1024/xmax<t*1024/total?src0(xmax+x–t*xmax/total,y,p):src1(x–t*xmax/total,y,p)
B: x*1024/xmax<t*1024/total?src0(xmax+x–t*xmax/total,y,p):src1(x–t*xmax/total,y,p)

In each of these expressions, the comparison before the question mark determines from which image to sample pixels. The next two expressions within the expression select the correct pixel for each image based on the percentage of the time that has passed for the clip.

Expression reference

This section provides a summary of all operators, variables, and functions that you can use in Transition Factory and Filter Factory expressions.

Operators

You can use the following operators in your expressions. The operators are presented in their order of precedence. Precedence determines which operators are evaluated first within an expression when the order of evaluation is ambiguous. For example, in the expression 2+3*4, the * operator is evaluated first because it has higher precedence than the + operator.

OPERATORS	DEFINITIONS
!, ~	Logical not, bitwise not
+, –	Add, subtract
*, /, %	Multiply, divide, modulo
<<, >>	Shift left, shift right
<, <=, >, >=	Less than, less than or equal to, greater than, greater than or equal to
==, !=	Equal to, not equal to
&, ^, |	Bitwise and, bitwise exclusive or, bitwise or
&&, ||	Logical and, logical or
?:	Conditional

• The arithmetic operators +, –, *, /, and % operate on signed, long integers that are 32 bits wide.

• The logical operators &&, ||, and ! treat all expressions as either true or false. Any value other than 0 is considered true, and only a 0 value is considered false.

• The shifting operators (<< and >>) perform logical — not arithmetic — shifts, so the sign of the shifted operand is not preserved.

Variables

You can use the following variables in your expressions. When creating a transition, append a 0 to variables describing values for the clip that plays first (such as *x*0 for the horizontal coordinate of the current pixel in the first image). Append a 1 to variables describing values for the clip that plays second (such as *x*1 for the horizontal coordinate of the current pixel in the second image).

VARIABLES	DEFINITIONS
r, g, b	Red, green, and blue channel values for the current pixel
a	Alpha channel value for the current pixel
c	Value of the current channel, whichever channel the expression is defining
i, u, v	Calculated channel values for the current pixel in YUV space
x, y	Coordinates of the current pixel
p	Channel index for the current expression
d	Direction (angle) of the current pixel from the center of the image, where d is an integer between 0 and 1024, inclusive
m	Distance (magnitude) from the center of the image to the current pixel
t	Current time
total	Maximum time
xmin, xmax	Range of horizontal coordinates over the width of the image
ymin, ymax	Range of vertical coordinates over the height of the image
pmin, pmax	Range of channel indexes within one pixel
dmin, dmax	Range of angles within the image, where dmin is always 0 and dmax is always 1024
mmin, mmax	Range of magnitudes with the image, where mmin is always 0 and mmax is always one half the diagonal size of the image

• The *i*, *u*, and *v* variables do not exist in an RGB image, so they are calculated from the RGB channels. Because this calculation takes some time, using these variables is slower than using the *r*, *g*, and *b* variables. The following formulas are used to convert from RGB to YUV:

i=((76*r)+(150*g)+(29*b))/256
u=((–19*r)+(–37*g)+(56*b))/256
v=((78*r)+(–65*g)+(–13*b))/256

• The *xmin* and *xmax* variables return the range of possible values for the *x* variable. The *ymin* and *ymax* variables return the range of possible values for the *y* variable. The *pmin* and *pmax* variables return the range of possible values for the *p* variable. These ranges are closed on the minimum and open on the maximum: *xmin*<=*x*<*xmax*, *ymin*<=y<*ymax*, and *pmin*<=p<*pmax*.

Functions

You can use the following functions in your expressions. Many functions place restrictions on the possible values of their arguments. If an argument is out of range, the expression will return a 0. For example, the expression ctl(8) evaluates to 0 because the ctl function requires an argument between 0 and 7.

When creating a transition, append a 0 to source functions (src, cnv, and rad) describing values for the clip that plays first. (For example, the expression src0(10,20,0) evaluates to the red channel value for the pixel at coordinates (10,20) in the first image.) Append a 1 to source function describing values for the clip that plays second. (For example, the expression src1(10,20,0) evaluates to the red channel value for the pixel at coordinates (10,20) in the second image.)

FUNCTIONS	DEFINITIONS
src*(x,y,p)*	Channel p for the pixel at coordinates x,y
cnv*(...)*	Convolution of neighboring channel values
rad*(d,m,p)*	Channel value of pixel p in the source image, which is m units away, at an angle of d, from the center of the image
ctl*(i)*	Value of slider i, where i is an integer between 0 and 7, inclusive
val*(i,a,b)*	Value of slider i, mapped onto the range a to b
map*(i,n)*	Item n from mapping table i, where i is an integer between 0 and 3, inclusive, and n is and integer between 0 and 255, inclusive
min*(a,b)*	Lesser of a and b
max*(a,b)*	Greater of a and b
abs*(a)*	Absolute value of a
add*(a,b,c)*	Sum of a and b, or c, whichever is greater
sub*(a,b,c)*	Difference of a and b, or c, whichever is greater
dif(a,b)	Absolute value of the difference of a and b
rnd*(a,b)*	Random number between a and b, inclusive
mix*(a,b,n,d)*	Mixture of a and b by fraction n/d
scl*(a,il,ih,ol,oh)*	Scale a from input range (il to ih) to output range (ol to oh)
sqr*(x)*	Square root of x
sin*(x)*	Sine of x, where x is an integer between 0 and 1024, inclusive
cos*(x)*	Cosine of x, where x is an integer between 0 and 1024, inclusive
tan*(x)*	Tangent of x, where x is an integer between 0 and 1024, inclusive
r2x*(d,m)*	x displacement of the pixel m units away, at an angle of d, from an arbitrary center
r2y*(d,m)*	y displacement of the pixel m units away, at an angle of d, from an arbitrary center
c2d*(x,y)*	Angle displacement of the pixel at coordinates x,y
c2m*(x,y)*	Magnitude displacement of the pixel at coordinates x,y

• The src (source) function is slow compared to the other operators and functions. Several evaluations of the src function in one expression can noticeably slow down the processing of an image. The coordinates passed to the src function should be within the ranges specified by the *xmin*,*ymin* and *xmax*,*ymax* variables; otherwise, the coordinates will be pinned.

• The cnv (convolution) function retrieves the channel values of the pixel adjacent to the current pixel and scales them by the arguments. The adjacent pixels are determined by a 3-pixel by 3-pixel grid with the current pixel in the center. The first nine arguments to the cnv function correspond to the channel values of these nine pixels.

The cnv function multiplies each channel value by its corresponding argument, adds the products together, and divides the result by the tenth argument in the list. For example, the expression cnv(–1,–2,–1,–2,16,–2,–1,–2,–1,4) scales the channel values as follows:

-1	**-2**	**-1**
-2	**16**	**-2**
-1	**-2**	**-1**

The channel values of the current pixel, in the center of the grid, are scaled by 16; the channel values of the pixels directly above, below, to the left, and to the right of the current pixel are scaled by –2; and so on. All of these products are then added together, and the result is divided by 4. The effect of this expression is to sharpen the image's channel values.

• The val (value) function converts the requested slider setting into a value in the requested range. The slider setting is multiplied by the size of the range (b–a) and offset by the start of the range (a). This function is useful when the range returned by the sliders (always 0 to 255, inclusive) does not match the range of values you want to use. For example, if the requested range is 1 to 10, a slider setting of 0 returns a value of 1, a setting of 255 returns a value of 10, and a setting of 127 returns a value of 5.

The start of the requested range does not have to be less than the end of the range. For example, the expression val(0,10,–10) returns values between 10 and –10.

• The map (mapping) function uses tables that are constructed according to the slider settings. Each table uses a pair of sliders: table n uses sliders 2*n* and 2*n*+1 for the high and low values, respectively. The table is constructed as follows, where *L* is the value of the low slider, *H* is the value of the high slider, and I is an entry: if *I*<=*L*, use 0; if *I*>=*H*, use 255; if *L*<*I*<*H*, use (*I*–*L*)*255/(*H*–*L*).

• The rnd (random) function returns a different random number each time it is called, but the entire function resets each time an image is processed. As a result, a transition or filter that uses the rnd function will have the same effect each time it is used on the same image.

• The mix (mixture) function combines the two input values using the specified fraction. A fraction of 1/2 returns the average of the two input values. A fraction close to 1 returns the first input value, and a fraction close to 0 returns the second input value. The mix function is defined as *a***n*/*d*+*b**(*d*–*n*)/*d*.

• The scl (scale) function maps a value from an input range onto an output range. For example, an input range of 0 to 255 could be mapped onto an output range of –100 to 100 by the expression scl(c,0,255,–100,100). In this example, channel values close to 0 are mapped starting at –100, and channel values close to 255 are mapped up to 100.

• The r2x and r2y functions convert radial expressions to Cartesian expressions. The c2d and c2m functions convert Cartesian expressions to radial expressions.