2D Effects

Overview

General demands on Effects

Effects represent the algorithms that transform input data into visible output in the destination framebuffer. Input data can either be images, or data that describes how an image is generated (e.g. Gradient parameters). OpenGL effects are identifiable by the prefix “GL” in their name (e.g. GlLinearGradientBrushBlend). Naturally, these effects are only available on platforms that support OpenGL.

Types of Effects

CGI Studio offers four types of effects. The *Brush Effect* defines its input based on its parameters alone. The *In-place Effect* requires an input (source) and produces an output (destination). The *Blend Effect* requires input (source) and is capable of reading back the target it renders. The *Combined Effect* is a combination of an arbitrary number of other specific effects in a specific sequence in order to have them processed in a more optimized way like a single pass.

Brush Effect

A *Brush Effect* provides means of generating graphical content out of data provided by the user. An example would be an effect which generates a rectangle of solid color based on red, green and blue values provided by an input parameter. A brush effect generates pixel data as output. E.g. SolidColorBrush

In-place Effect

An *In-place Effect* requires source data provided. Source data is always given in form of pixel data and is supposed to be provided by either a Brush Effect or another In-place Effect. This kind of effect can utilize any additional parameters to transform the source pixel data to destination data. The destination data is again pixel data.

Blend Effect

A *Blend Effect* is very similar to an In-place Effect. To be specific, every In-place Effect resembles a specialization of a Blend Effect, namely that they do blend with a fixed functionality (ignoring the target data). What makes Blend Effects special is only their purpose of being the effect which also takes the already computed destination data into account as an input to calculate the new destination data. This allows Blend Effects to transform the source data and the destination data to any kind of combination of both. Destination data is again pixel data. The blend effect is responsible for setting up the alpha blending parameters, how the computed image gets “blended” into the destination surface (especially the framebuffer).

Combined Effect

A *Combined Effect* is a single effect which represents a whole sequence of other effects chained together. For example, if you want to render bitmap data with a blending function, there usually is a combined effect which represents the Brush Effect and the Blend Effect in one single object. This single object can take the advantage to perform an operation in a single step as the external interfaces (pixel data) can be replaced internally by setting options in the native rendering API. A *Combined Effect* for *Brush* and *Blend Effect* for example is the Candera::BitmapBrushBlend.

Basic Effect Properties

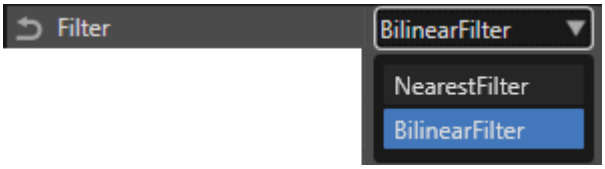
CGI-Studio provides various effects, each of which have certain properties, which they can be configured with. Properties that apply to most of the CGI-Studio Blend Effects are described below.

Filter

When the content of a node (e.g. a texture) should be displayed in a different size than its original size (this means if a magnification or a minification of the content is desired), filtering is applied.

- If each source pixel maps onto more than one destination pixel, this is known as *minification*.
- If each source pixel maps exactly onto one destination pixel, filtering is not applied.
- If each source pixel maps onto less than one destination pixel this is known as *magnification*.

Values



NearestFilter

Nearest-neighbor interpolation is the fastest and crudest filtering method — it simply uses the color of the source pixel closest to the center of the destination pixel for the pixel color. While fast, this results in a large number of artifacts - texture 'blockiness' during magnification, and aliasing and shimmering during minification.

BilinearFilter

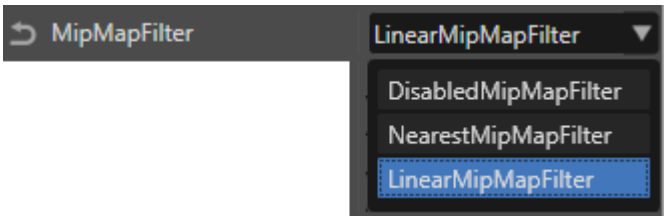
Bilinear filtering is the next step up. In this method the four nearest source pixels to the center of the destination pixel are sampled (at the closest mipmap level), and their colors are combined by weighted average according to distance. This removes the 'blockiness' seen during magnification, as there is now a smooth gradient of color change from one source pixel to the next, instead of an abrupt jump as the center of the destination pixel crosses the source pixel boundary. Bilinear filtering is almost invariably used with mipmapping; though it can be used without, it would suffer the same aliasing and shimmering problems as nearest-neighbor filtering.

Mip Map Filter

This setting is available for all effects using images and controls filtering when the image has a mipmap chain. This Mip Map Filter option is available for all platforms. However, it is applied on

OpenGL based 2D implementations only.

Values



DisabledMipMapFilter

Mipmap filtering is disabled and not applied.

NearestMipMapFilter

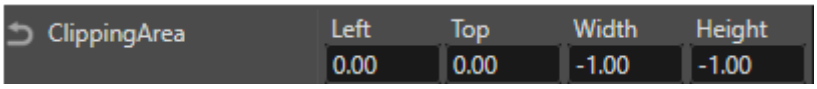
This method still uses nearest neighbor interpolation, but adds mipmapping — first the nearest mipmap level is chosen according to distance, then the nearest texel center is sampled to get the pixel color. This reduces the aliasing and shimmering significantly, but does not help with blockiness.

LinearMipMapFilter

For OpenGL devices LinearMipMapFiltering is a common standard filter mechanism. It is a required standard for all OpenGL versions. It uses nearest-neighbor sampling from individual mipmaps whilst linearly interpolating the two nearest mipmaps relevant to the sample.

ClippingArea

The Clipping Area can be configured for effects that use images. It specifies the area of the image to be rendered.



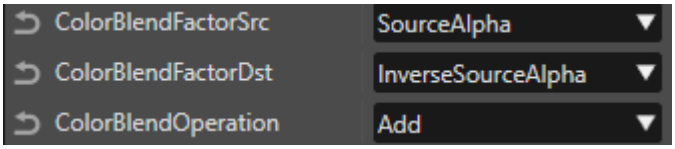
This area is defined by its top-left position (Left, Top) and its dimensions (Width, Height). The default configuration, as shown in the image above, renders the entire image without applying any clipping.

Blend Factor

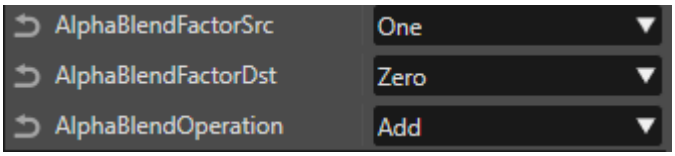
Blending is the process of mixing color values of the pixels that are already drawn with those, that are about to be drawn. Blending parameters allow the source and destination colors for each output to be combined in various ways. The output for a combined color value is computed separately for each color component (red, green, blue). A combined transparency value is calculated using the source and destination alpha values.

In CGI Studio you can define a Color Blend Factor for Source and for Destination and a Color Blend Operation. The same applies for the Alpha value, where you can define Alpha Blend Factors for Source and Destination as well as an Alpha Blend Operation.

Color Blend Settings (with default values)



Alpha Blend Settings (with default values)



Computation of a Combined Color Value

Formula:

$$value_{DST} = value_{SRC} * \langle src\_blend\_factor \rangle \langle op \rangle value_{DST} * \langle dst\_blend\_factor \rangle$$

valueDST	color value of destination (background) pixel to blend
valueSRC	color value of new scene graph element (source) pixel to blend
src_blend_factor	factor to multiply source pixel color value with
dst_blend_factor	factor to multiply destination pixel color value with
op	operation to combine multiplied source and destination values

Computation of a Combined Alpha Value

Formula:

$$\alpha_{DST} = \alpha_{SRC} * \langle src\_blend\_factor \rangle \langle op \rangle \alpha_{DST} * \langle dst\_blend\_factor \rangle$$

alphaDST	alpha value of destination (background) pixel to blend
alphaSRC	alpha value of new scene graph element (source) pixel to blend
src_blend_factor	factor to multiply source pixel alpha value with
dst_blend_factor	factor to multiply destination pixel alpha value with
op	operation to combine multiplied source and destination alpha values

Values

Value		Meaning
Zero	0	keep value as it is
One	1	ignore value
SourceColor	valueSRC	source color value
InverseSourceColor	(1 - valueSRC)	1 - source color value
SourceAlpha	alphaSRC	source alpha value
InverseSourceAlpha	(1 - alphaSRC)	1 - source alpha value
DestColor	valueDST	destination color value
InverseDestColor	(1 - valueDST)	1 - destination color value
DestAlpha	alphaDST	destination alpha value
InverseDestAlpha	(1 - alphaDST)	1 - destination alpha value

As shown in the equations in Computation of a Combined Color Value and Computation of a Combined Alpha Value the color and alpha blend factors determine the value by which each pixel is multiplied. Note that color and alpha values are normalized between 0.0 and 1.0 rather than using the 0 to 255 range.

Blend Operation

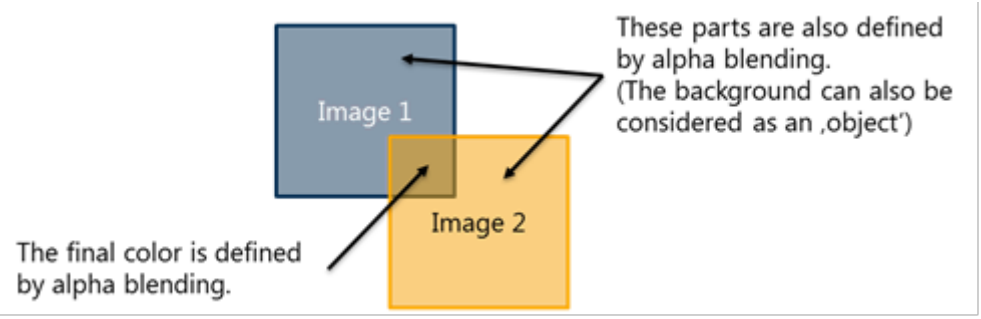
The Blend Operation specifies how the destination and source pixel values are combined. In CGI Studio, the Blend Operation can be configured for the color (Color Blend Operation) and for the alpha value (Alpha Blend Operation).

Values

Value	Operation
Add	<term1> + <term2>
Subtract	<term1> - <term2>
ReverseSubtract	<term2> - <term1>
BitwiseAnd	

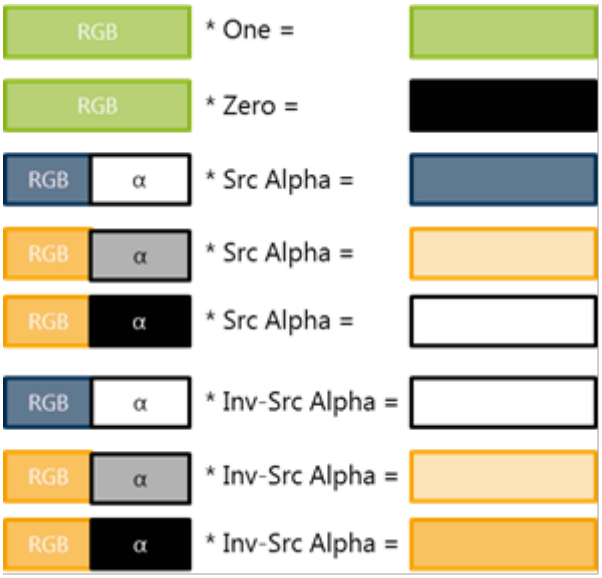
Blending

Alpha Blending is the process of combining the pixel data of multiple overlapping objects to a single pixel with a final color and alpha value.



Blend Factor

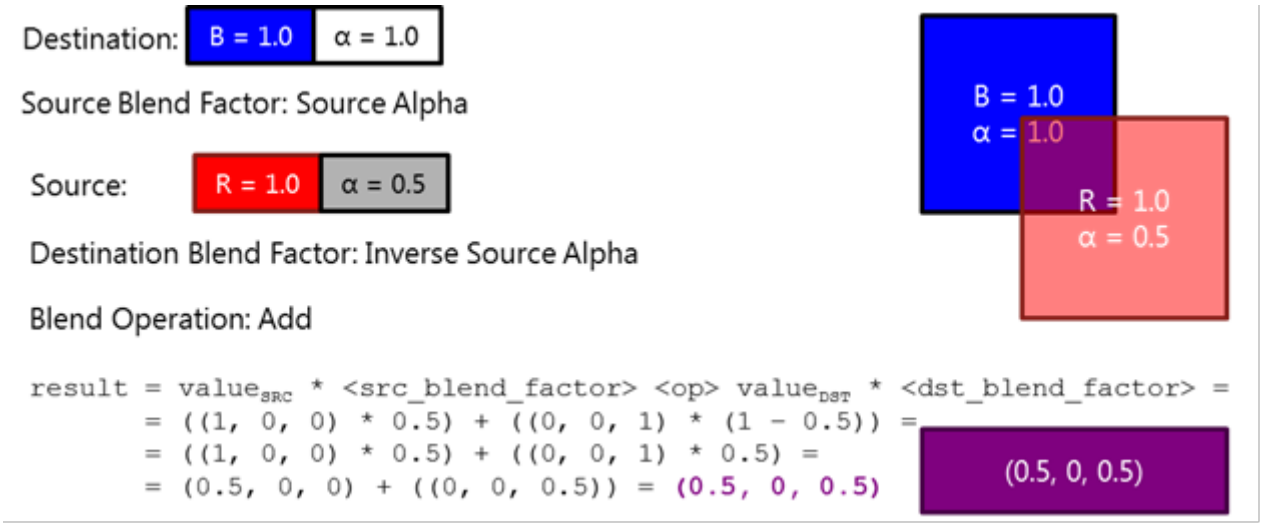
Here is an example on the different blend factors using alpha blending:



> Alpha is 1.0 (white) for non-transparent pixels and 0.0 (black) for transparent pixels.

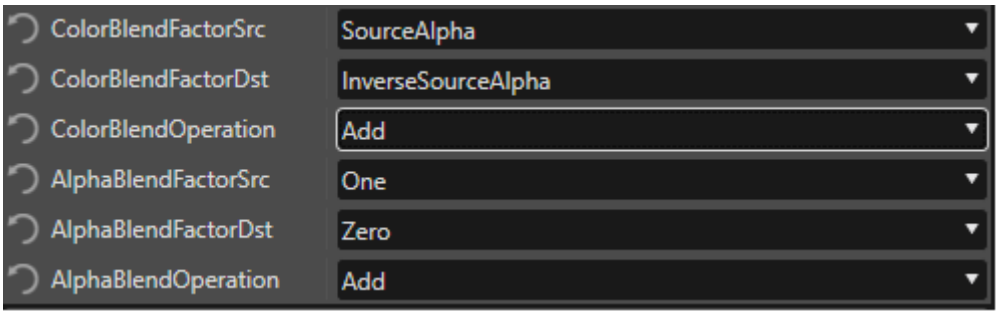
Blend Operation

The blend operation specifies how the destination and source pixel values are combined. Here is an example blending 50% red on opaque blue:



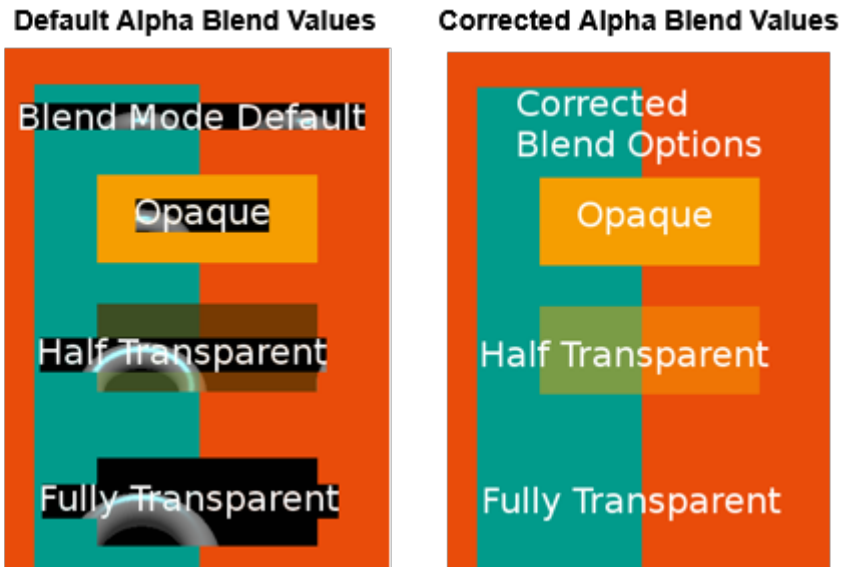
Blend Problems

The default values used for blending are set to the following configuration:



The default values ensure maximum performance when no layer blending is needed or no objects with transparency on the same layer are overlapping. The reason is that the Destination Alpha is multiplied with 0 (ignored) and the Source Alpha is used instead. This means the transparency of the last drawn item is used. The default Color Blend Values are fine for all common cases.

In the following example, a partially transparent 2D scene on one layer is rendered over a 3D scene of another layer.



	Default Alpha Blend Values	Corrected Alpha Blend Values
Configuration		
Alpha Blend Factor Src	One	InverseDestAlpha
Alpha Blend Factor Dst	Zero	One
Alpha Blend Operation	Addd	Add
Problem caused by	- Scene using layers - Overlapping objects with transparency	
Reason	Destination Alpha is multiplied with 0 (ignored), and the source Alpha is used instead. Therefore, the transparency of the last drawn item is used and the background layer with the 3D scene is partially visible.	With the corrected Alpha Blend vales, the transparency of the SolidColorNodes is applied as intended and the background layer is not visible.

> Alpha blending is disabled by default. Blending should not be enabled unless really necessary. Unnecessarily enabling blending will lead to a performance loss. When using images with alpha information, blending can be enabled in the Solutions Options dialog: “File – Solution Options... - Default Render Mode – Enable Blending”

Bitmap Brush Blend Effects

BitmapBrushBlend

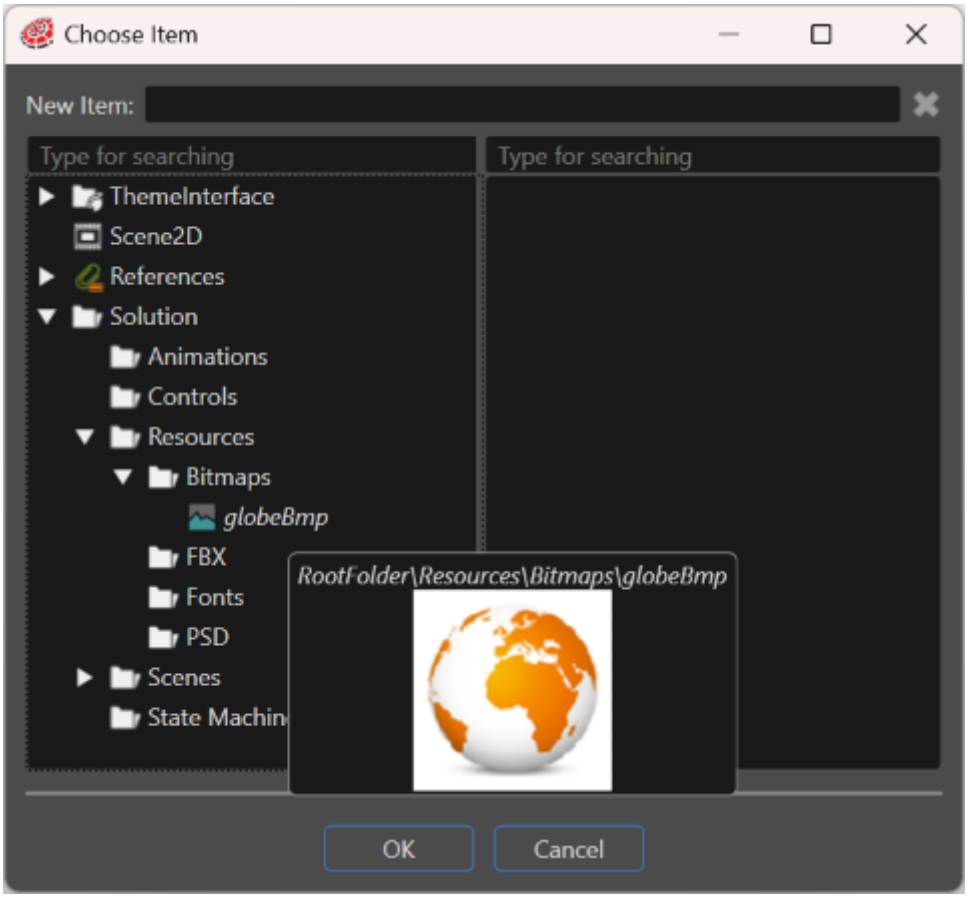
With this effect you can configure a RenderNode to display an image.

Image

This is the place where you configure the image to be displayed by this RenderNode.

When clicking on the "magnifying glass"-icon you can choose your desired image from the “Choose Item” dialog, which lists the already imported images.

A preview of the image is displayed in a tooltip when moving the mouse over the image’s name.

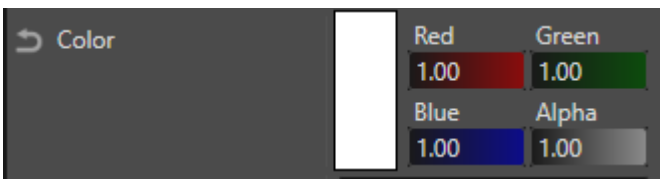


BitmapBrushColorBlend

BitmapBrushColorBlend is similar to BitmapBrushBlend but it additionally allows to define a color with which the configured image should be blended.

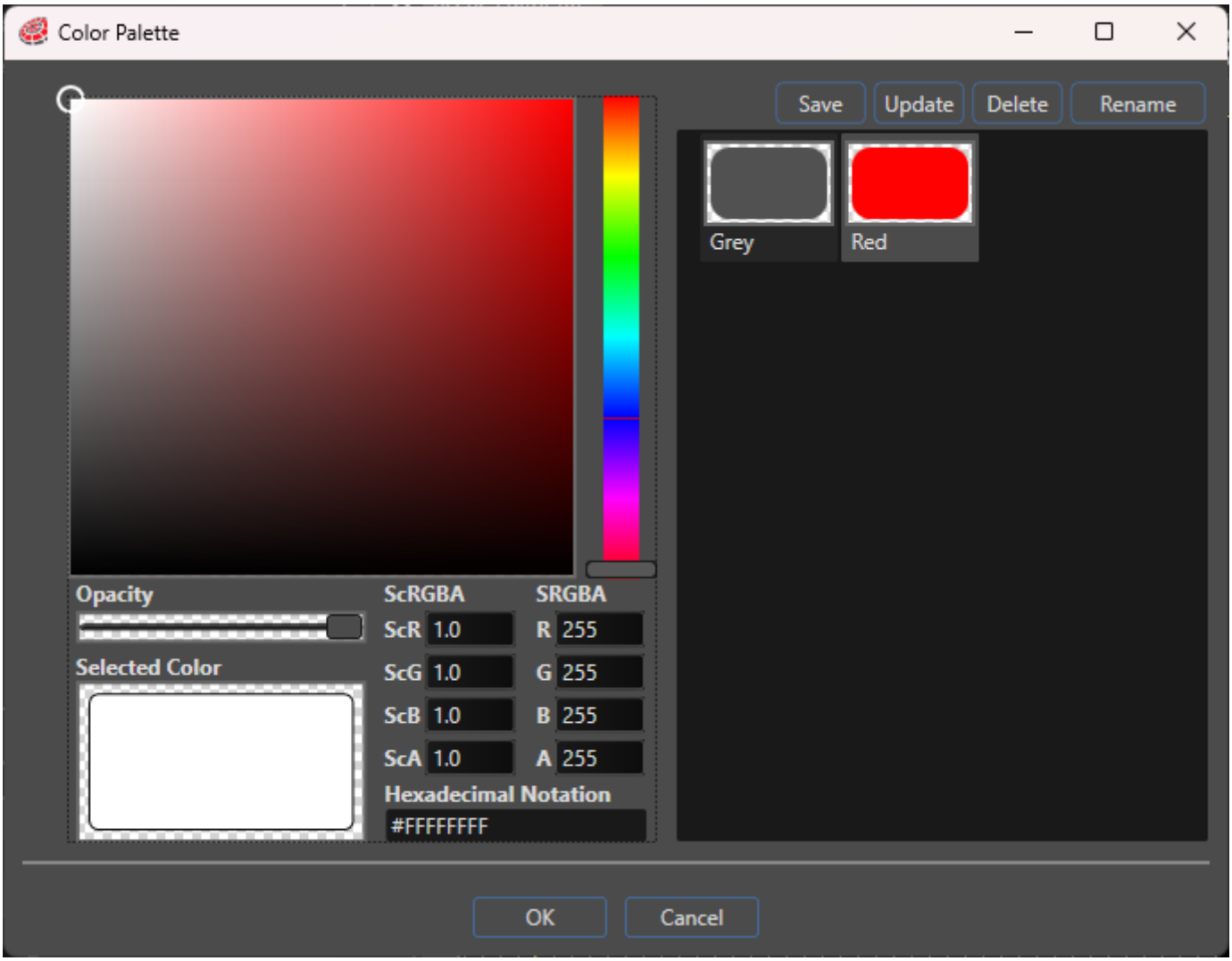
Color

The color can be chosen in the “Color Palette” dialog which opens when clicking on the icon on the right.



This dialog also offers the possibility to define and save custom colors you need for your project. This can be achieved by configuring a color and clicking the “Save” button on the top right, which creates a new rectangle in your configured color and allows you to name it.

The default blending color after applying this effect is white.



BitmapBrushHslBlend

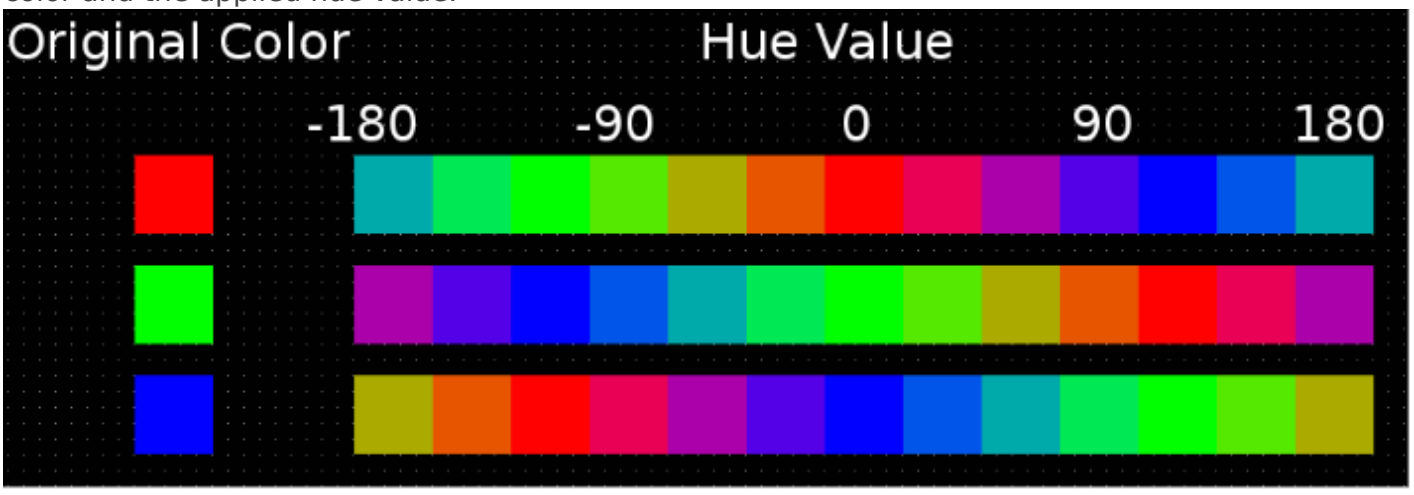
The BitmapBrushHslBlend effect is similar to the BitmapBrushBlend effect, additionally a hue, saturation and lightness correction can be applied according to the following settings. This makes it possible to change defined colors of the image (e.g. change the blue color of the working man to orange as in the picture).

Hue

With this property you can alter the color of the image. Increasing the value moves from blue through green, yellow orange, red purple and back to blue. In CGI-Studio the values for hue range from -180 to 180. The value is periodical outside this range.

Hue Value	Meaning
-180	Color is inverted (the complementary color of the RGB color system is used)
0	Color is unchanged
180	Color is inverted (the complementary color of the RGB color system is used)

Here is an example on how the colors are altered in Scene Composer depending on the original color and the applied hue value.



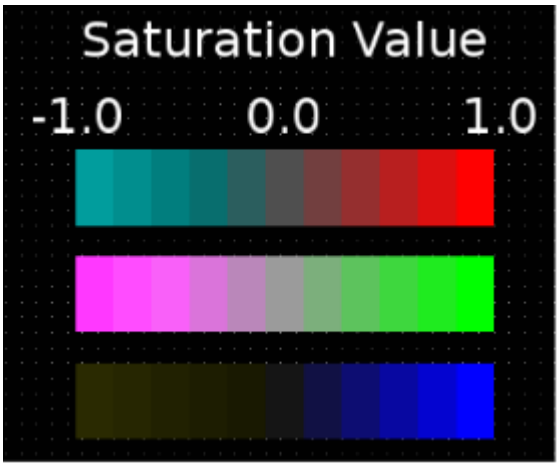
Saturation

Saturation is the colorfulness of a color relative to its own brightness. This value is used to saturate the color. The values range from -1.0 to 1.0.

Saturation Value	Meaning

-1	Colors of the image are displayed inverted (with the complementary colors of the RGB color system).
0	Image is displayed in greyscale.
1	Image is displayed unchanged.

Here is an example on various saturation values for red, green and blue in CGI-Studio.

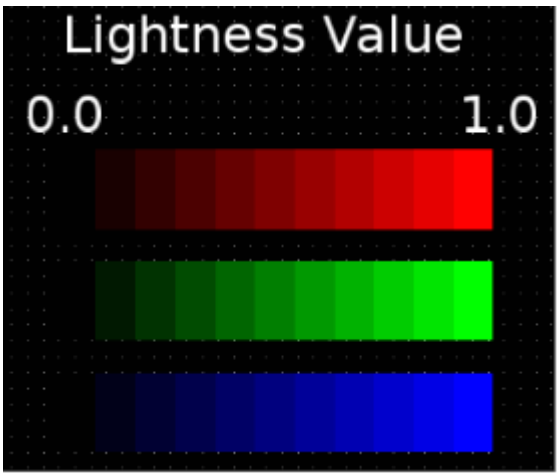


Lightness

This setting is used to change the brightness of the image’s colors. Its values range from 0.0 to 1.0.

Lightness Value	Meaning
0.0	Image is displayed black.
1.0	Image is displayed unchanged

Here is an example on various lightness configurations in CGI-Studio for red, green and blue.

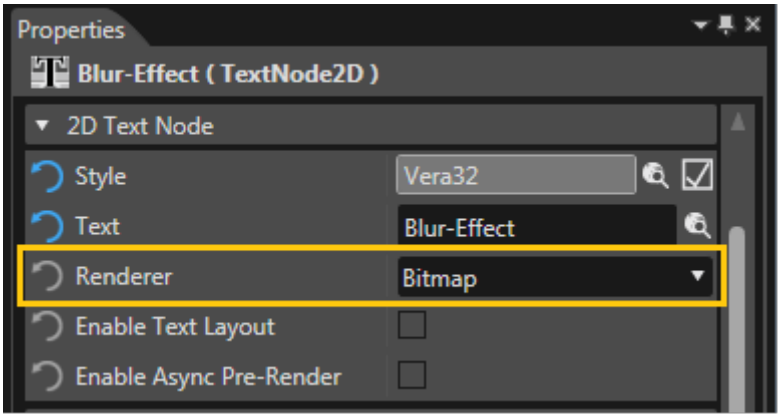


BlurBitmapBrushBlend

This effect is similar to BitmpaBrushBlend, where you can display an image with a RenderNode. Additionally, this image can be distorted by a gaussian blur using the filter size setting.

This blurring effect can be used to reduce image noise and reduce detail. The visual effect of this blurring technique is a smooth blur resembling that of viewing the image through a translucent screen.

The default renderer of TextNodes is the Bitmap renderer.



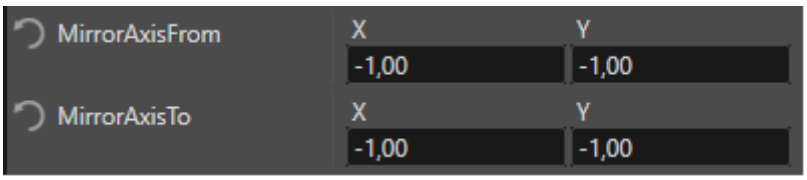
If a TextNode is rendered with the Bitmap renderer, the BlurBitmapBrushBlend effect can also be applied.

Filter Size

The filter size defines the quadratic size of the filter kernel. The bigger the configured filter size, the more the image is blurred. But a bigger filter size also means more complex calculations and hence less performance.

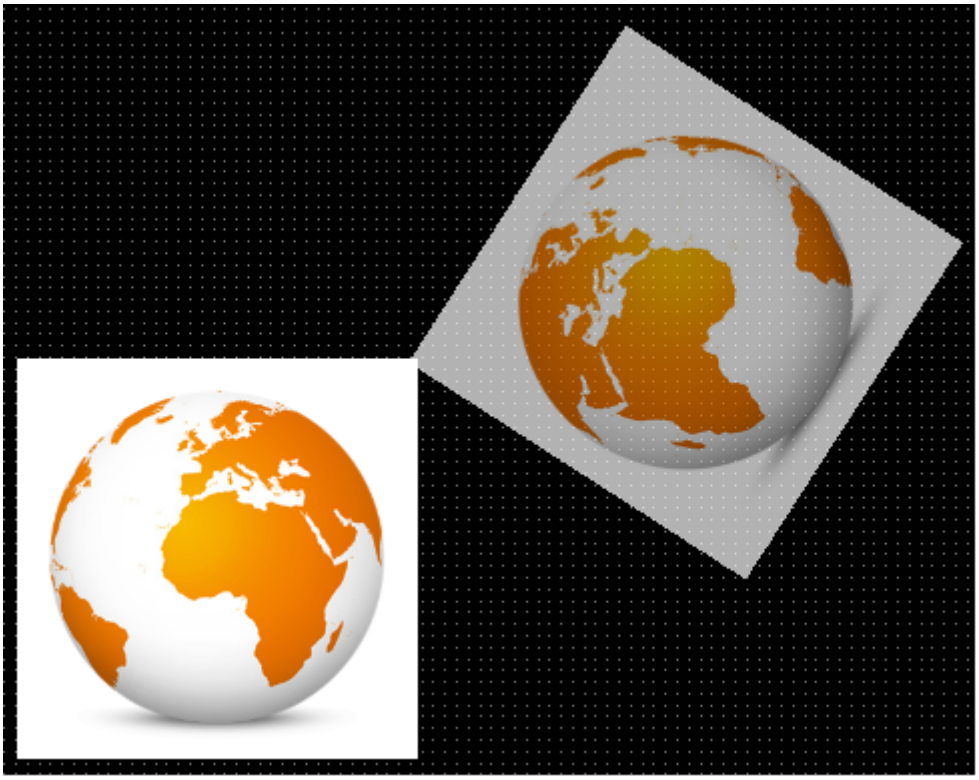
MirrorBitmapBrushBlend

You can define an image to be rendered with a RenderNode. Additionally, with this effect, the RenderNode is enhanced with a mirror image – a reflection that is mirrored across a defined central line, the mirror axis. Every point has the same distance from this mirror axis and the reflection has the same size as the original image. The effect MirrorBitmapBrushBlend has additional properties to define a mirror axis (Mirror Axis From and Mirror Axis To) and another property to configure the alpha value of the mirror image.



Mirror Axis From, Mirror Axis To

The mirror axis is the central line, where every point has the same distance from. It defines, where exactly this effect mirrors the image to. It is configured by these two points: “Mirror Axis From” and “Mirror Axis To”, each of which has to be configured with an X and a Y coordinate.



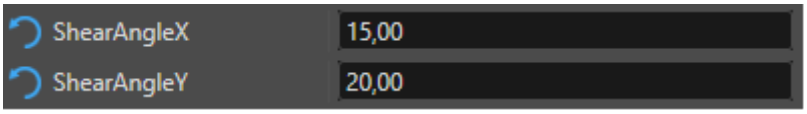
Alpha

This property defines the opacity of the mirrored image. A value of 1 means the mirror image is opaque, a value of 0 means the mirror image is totally transparent.

If a TextNode is rendered with the Bitmap renderer, which is the default, the MirrorBitmapBrushBlend effect can also be applied.

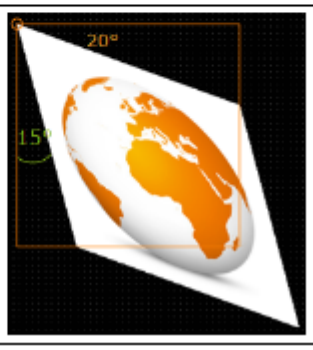
ShearBitmapBrushBlend

Similar to the BitmapBrushBlend effect, this effect offers the possibility to display an image with a RenderNode. With the ShearBitmapBrushBlend effect you can enter two additional properties Shear Angle X and Shear Angle Y to add a shearing effect.



Shear Angle X, Shear Angle Y

These two properties define the shearing of the image. Shear Angle X defines the horizontal shearing angle, Shear Angle Y defines the vertical shearing angle.

Shear Angle Configuration		Shear Angle Y	
		0	20
Shear Angle X	0		
	15		

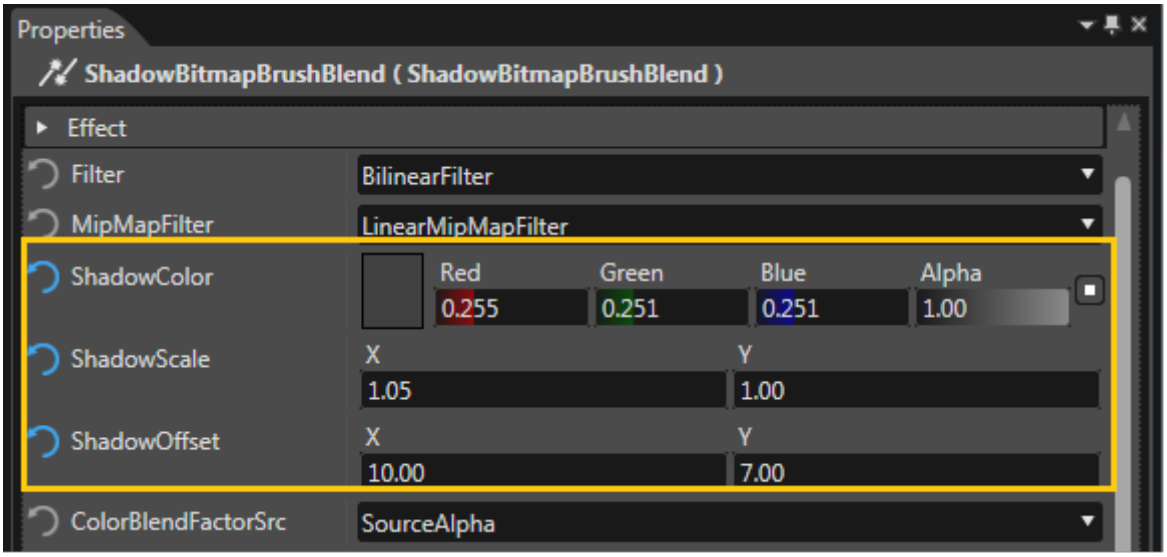
(Note that the angle is displayed here for explanation reasons and is not displayed in SceneComposer.)

If a TextNode is rendered with the Bitmap renderer, which is the default, the ShearBitmapBrushBlend effect can also be applied.

Shear-Effect

ShadowBitmapBrushBlend

This effect allows to display a Bitmap on a RenderNode and additionally add a shadow in the background of this Bitmap. For proper configuration, 3 parameters for Shadow Color, Shadow Scale and Shadow Offset can be altered.



Shadow Color

With this parameter the color of the shadow can be configured. The color can be chosen in the “Color Palette” dialog, which opens when clicking on the rectangular color preview. The default Shadow Color is white.

Shadow Scale

The property “Shadow Scale” defines the shadow’s size and proportions. With a Shadow Scale of “1;1” the shadow is as big as the displayed Bitmap and is not visible, if the Shadow Offset is set to “0;0”. Setting the X coordinate of the Shadow Scale to 2, makes the shadow twice as wide. When enlarging the Shadow Scale, the center of the shadow remains at the center of the Bitmap.

Shadow Offset

With the Shadow Offset property the horizontal and vertical offset of the shadow’s center to the center of the displayed Bitmap can be configured. The values are given in pixels.

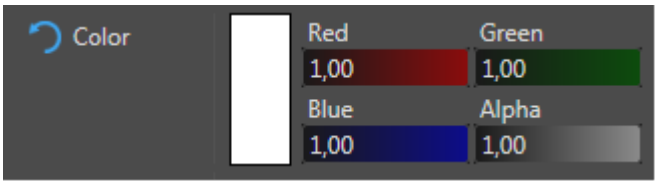
If a TextNode is rendered with the BitmapRenderer, which is the default, the ShadowBitmapBrushBlend effect can also be applied.

Solid Color Brush Blend Effects

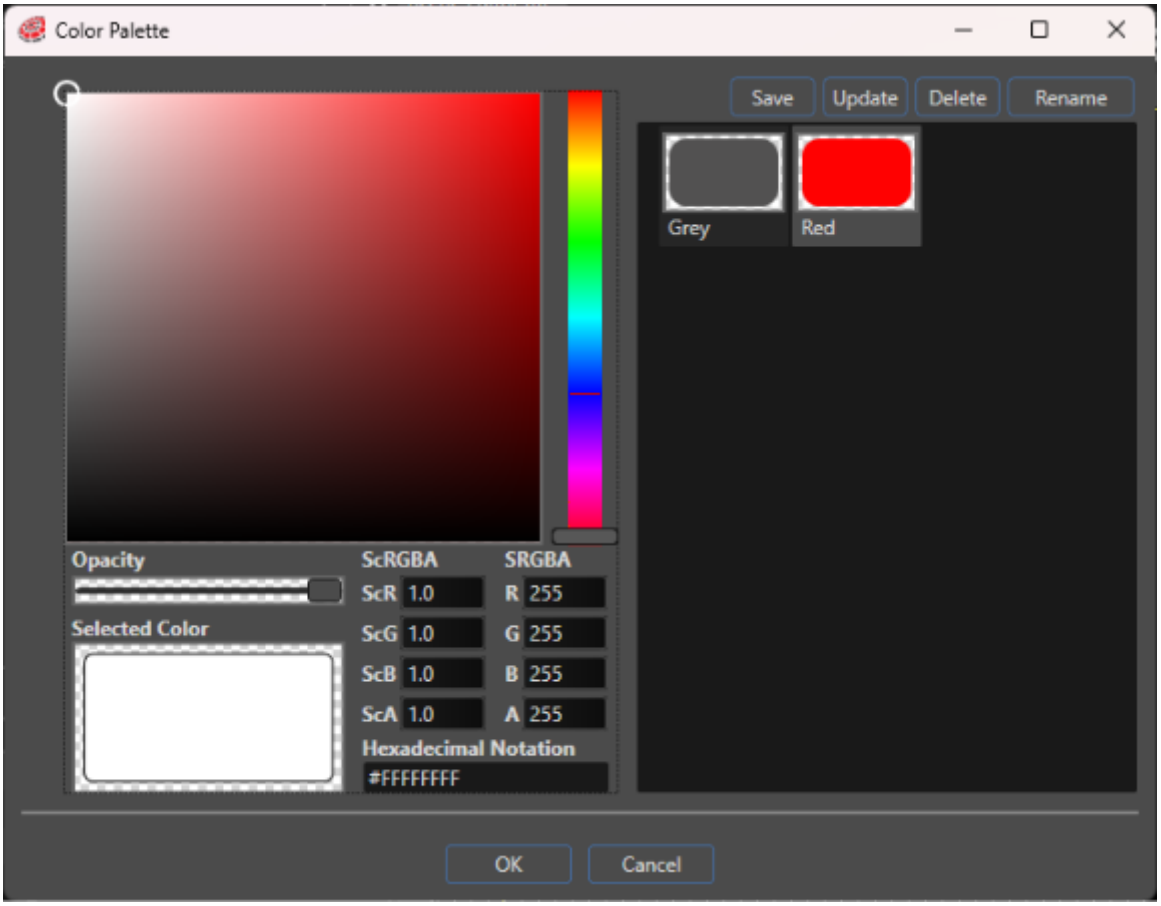
SolidColorBrushBlend

A Solid Color Node is used to display a colored rectangle. When creating a SolidColorNode by clicking the "magnifyingglass"-icon, this is a Node2D with a SolidColorBrushBlend effect on it.

Color



The color can be chosen in the “Color Palette” dialog where you can also define and save the custom colors you need for your project. The default blending color after applying this effect is white.



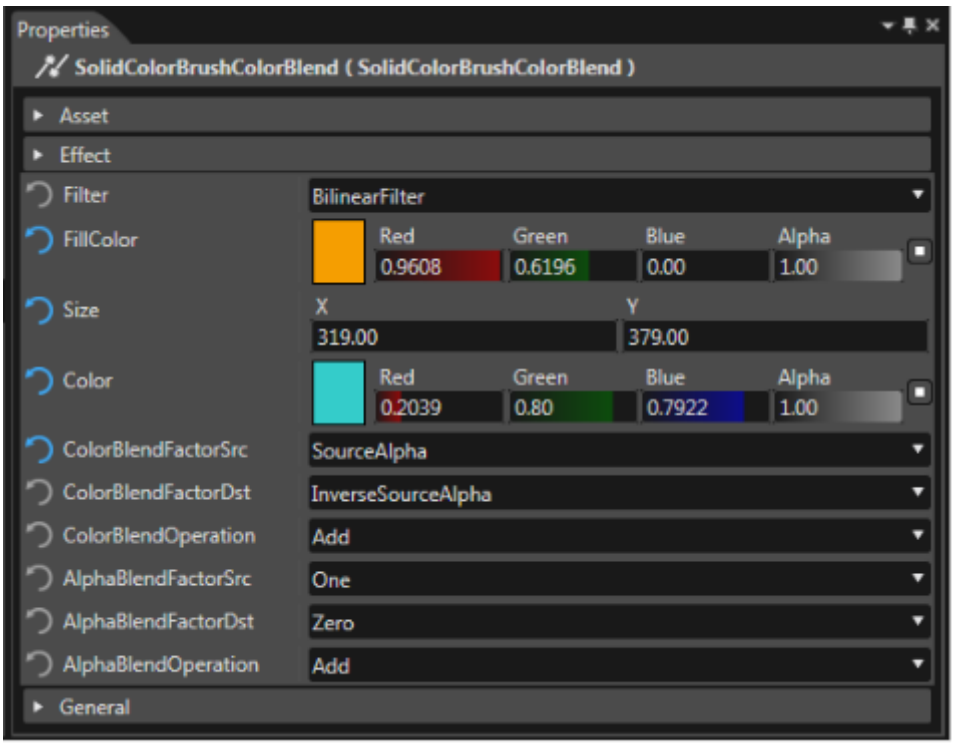
Size

This property defines the horizontal and vertical size of the SolidColorNode in pixels.

SolidColorBrushColorBlend

A SolidColorBrushColorBlend effect adds a color effect to the existing SolidColorBrushBlend effect.

It multiplies the FillColor with the Color. This effect can be useful when it comes to reusing ColorEffects out of an effect, to multiply a color with the actual color to blit; especially in cases where the original color should be preserved, but should be changed for certain effects.



Filter

For more details please see chapter Filter of the "Basic Effect Properties" page.

FillColor

The FillColor is the solid color for this effect.

Size

This property defines the horizontal and vertical size of the Node in pixels.

Color

The SolidColorBrushColorBlend's Color property defines the constant color for modulating all input color channels (RGBA).

Blending

There are several parameters that can be edited to adjust the color blending:

• ColorBlendFactorSrc • ColorBlendFactorDst • ColorBlendOperation	• AlphaBlendFactorSrc • AlphaBlendFactorDst • AlphaBlendOperation

BlendFactor
The BlendFactor properties may be configured to have one of the following values:

- Zero
- One
- SourceColor
- InverseSourceColor
- SourceAlpha
- InverseSourceAlpha
- DestColor
- InverseDestColor
- DestAlpha
- InverseDestAlpha

BlendOperation
The BlendOperation properties may be configured to have one of the following values:

- Add
- Subtract
- ReverseSubtract
- BitwiseAnd

For more details about blending please refer to chapter Blending.

SolidColorBrushHslBlend

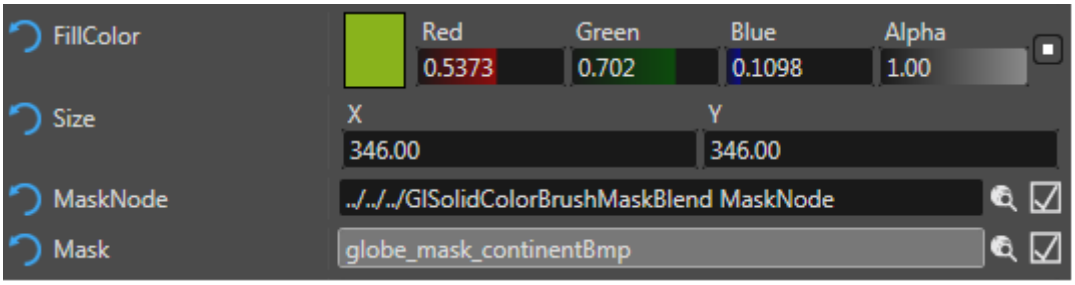
This effect is similar to the SolidColorBrushBlend, but it has additional properties to be defined. Using the properties hue, saturation and lightness, a color correction can be applied. Additional information about hue, saturation and lightness properties can be found in chapter

BitmapBrushHslBlend .

OpenGL Solid Color Brush Mask Effect

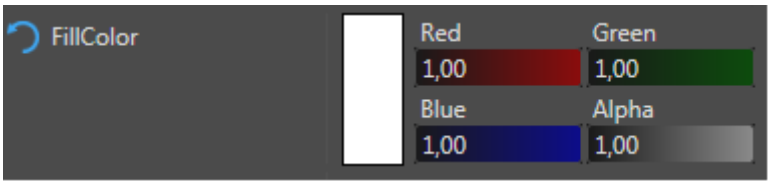
GlSolidColorBrushMaskBlend

With this effect, you can define a rectangle with a any solid color, similar to the SolidColorNode. Additionally, the configuration of a mask image and a mask node is possible in order to specify transparent areas in this colored rectangle.



Fill Color

The color can be chosen in the “Color Palette” dialog where you can also define and save the custom colors you need for your project. The default blending color after applying this effect is white.



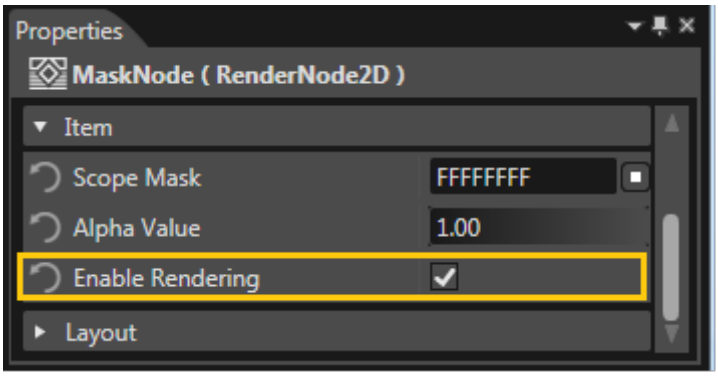
Size

This property defines the horizontal and vertical size of the SolidColorNode in pixels.

Mask Node

The “Mask Node” provides transformation information (that means position, orientation and scaling factor) for the mask image. If no Mask Node is configured, the mask is set to the same position as the displayed image with no rotation and scaling factor 1.

One way to position the mask properly is to create a BitmapNode with the mask image and apply the necessary transformations. With the visible mask image, the correct position, rotation and scaling factor can easily be found. After the desired position is set to this BitmapNode, the property “Enable Rendering” can be unchecked to make this node disappear.



Now, this BitmapNode can be set as Mask Node by choosing it from the “Choose Item” dialog.

Mask

An image with alpha information can be configured as mask for this solidly colored rectangle. This way, the mask image can add alpha information to a plain rectangle without alpha information. The color information of the mask image is not taken into account. Also, blend options are not applied between the displayed solid rectangle and the mask image.

OpenGL Bitmap Brush Mask Effects

GlBitmapBrushMaskBlend

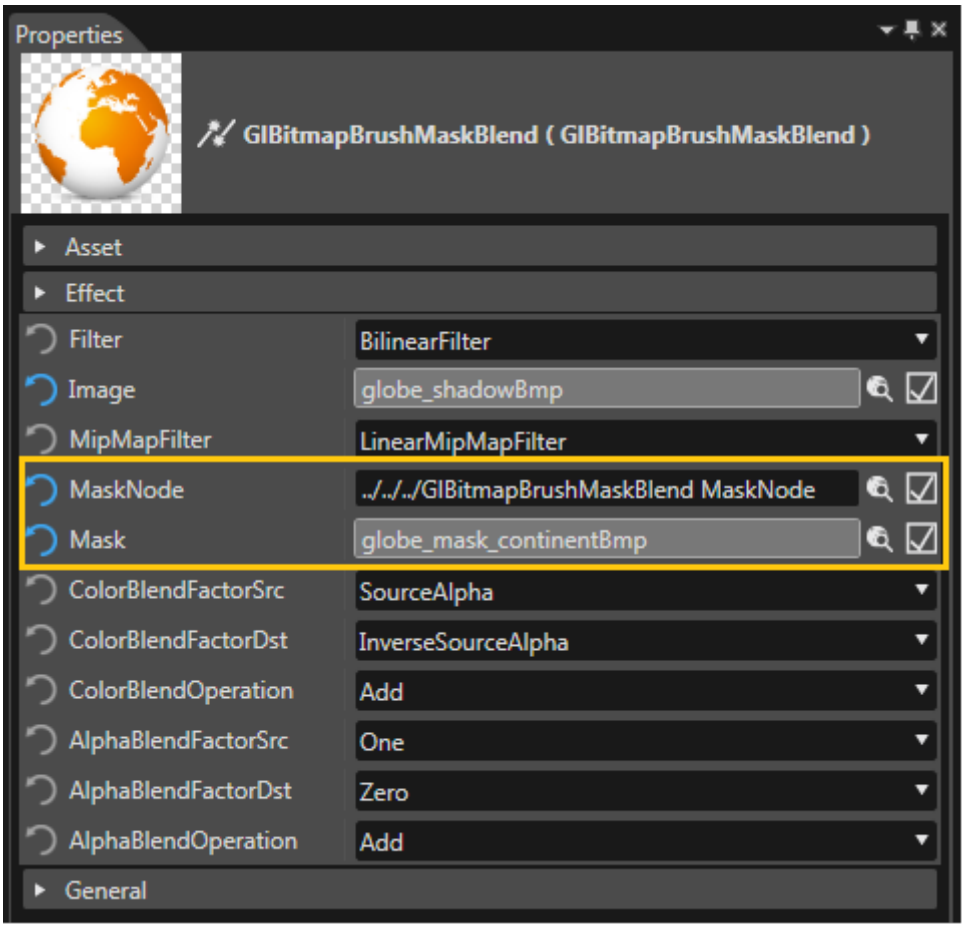
This effect allows to add a mask (i.e. to add additional alpha information) to an image by defining a mask image. The position, orientation and scaling of this mask image is defined by a Mask Node.

Image

Here you can define the image that should be displayed. The image can be selected from the list of imported images in the “Choose Item” dialog. The mask itself is configured with the two additional properties “Mask Node” and “Mask”.

Mask

An image with alpha information can be configured as mask for another image. This way, the mask image can add alpha information to an image without alpha information. The color information of the mask image is not taken into account. Also, blend options are not applied between the displayed bitmap and the mask image.



Mask Node

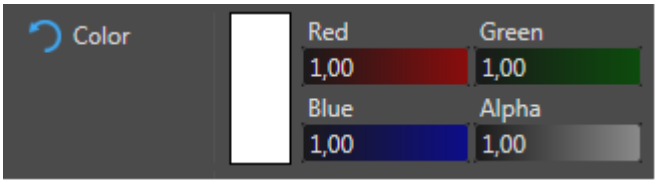
The “Mask Node” provides transformation information (that means position, orientation and scaling factor) for the mask image. If no Mask Node is configured, the mask is set to the same position as the displayed image with no rotation and scaling factor 1. The Mask Node can be selected in the “Choose Item” dialog that opens after clicking the icon to the right of the property’s value.

GlBitmapBrushColorMaskBlend

The effect OpenGlBitmapBrushColorMaskBlend is similar to OpenGlBitmapBrushMaskBlend but has an additional color Property, which the configured bitmap is blended with.

Color

The color can be chosen in the color palette dialog where you can also define and save the custom colors you need for your project. The default blending color after applying this effect is white.



OpenGL Gradient Brush Effects

The CGI-Studio gradient brush paints an area with two colors that blend into each other along an axis. You can use a gradient brush to create impressions of light and shadow, giving your controls a feeling of 3D. You can also use them to simulate smooth surfaces like glass, chrome, or water. CGI-Studio provides three types of gradient brushes: GlLinearGradientBrushBlend, GlLinearGradientBitmapBrushBlend and GlRadialGradientBrushBlend.

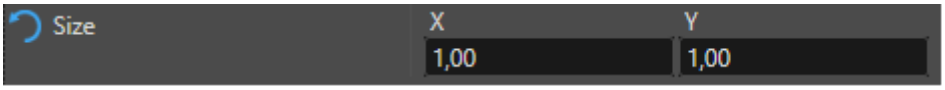
> Note that the Gradient Brush Effects are OpenGL effects that are only available on platforms supporting OpenGL.

GlLinearGradientBrushBlend

The GlLinearGradientBrushBlend paints an area with a gradient defined along a line, the Gradient Direction. Additionally to common blend effect properties, you can define the size of the painted area, the gradient's two colors as well as the center, the direction and the magnitude of the gradient.

Size

The filled size (width and height) in pixel.

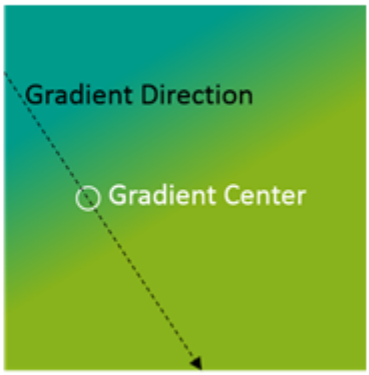
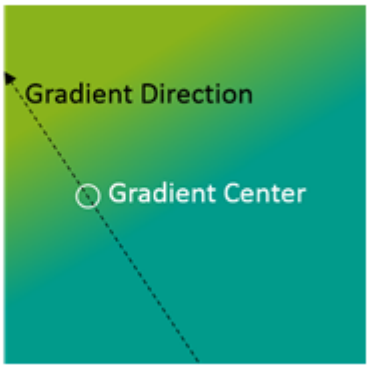


Gradient Center

This 2D coordinate value defines the center of the gradient effect. This is the position where both colors are half opaque. The gradient center coordinate is relative to a bounding box: 0 indicates 0% of the bounding box and 1 indicates 100% of the bounding box.

Gradient Direction

This 2D vector value defines the direction of the gradient effect. The colors of the gradient can be switched by changing the direction of this vector.

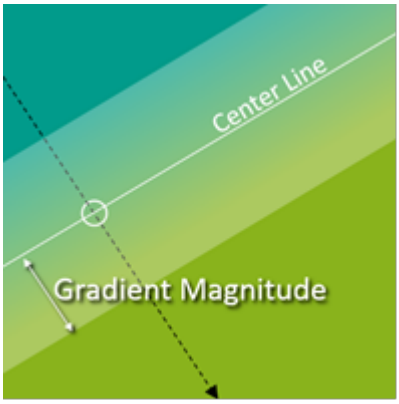
Gradient Direction value	3; 5	-3; -5
Visual result		

There are three special gradient directions that need to be addressed:

Gradient Direction value	1; 0	0; 1	0; 0
Description	vertical gradient	horizontal gradient	radial gradient
Visual result			

Gradient Magnitude

Defines the magnitude of the gradient effect. The magnitude is the distance from the center line to the closest parallel line where the ’positive’ color is fully opaque in the Gradient Direction and the ‘negative’ color is fully opaque in the opposite direction.



Two special values should be noted:

Gradient Magnitude value	0,0	0,5
Description	No gradient, the color changes without blending	One configured color is displayed in the top left corner, the other defined color is displayed in the bottom right corner, blending of the two colors is done all across the area between
Visual result		

Positive Color

Defines the positive color. The opacity of the positive color increases in the Gradient Direction.

Negative Color

Defines the negative color. The opacity of the negative color decreases in the Gradient Direction.

GlLinearGradientBitmapBrushBlend

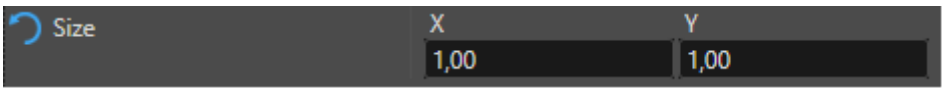
The GlLinearGradientBitmapBrushBlend effect blends a specified bitmap image with a linear gradient to create a combined visual output. The gradient's configuration is identical to the GlLinearGradientBrushBlend effect. The image configuration is the same as for the BitmapBrushBlend effect.

GlRadialGradientBrushBlend

Like the GlLinearGradientBrushBlend, the GlRadialGradientBrushBlend paints an area of a defined size with a generated radial gradient between a Center Color and a Radial Color. The gradient layout can be defined by the Gradient Center and the Gradient Magnitude.

Size

The filled size (width and height) in pixel.

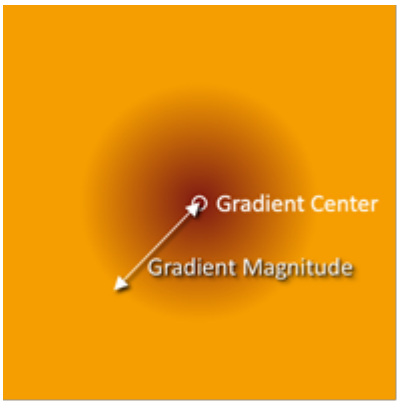


Gradient Center

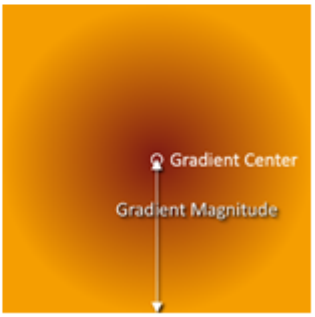
This 2D coordinate value defines the center of the gradient effect. This is the position where the 'center' color is fully opaque and can also be configured to be outside of the node (e.g. -0,25; -0,25). The gradient center coordinate is relative to a bounding box: 0 indicates 0% of the bounding box and 1 indicates 100% of the bounding box.

Gradient Magnitude

Defines the magnitude of the gradient effect. The magnitude is the distance from the center point to the closest point where the radial color is fully opaque and the center color is fully transparent.



There are two special values for the Gradient Magnitude:

Gradient Magnitude Value	0	0,5
Description	no gradient, radial color only	The center color starts opaque at the center point and the radial color is opaque only on a very thin radial area in the outskirts. The colors are blended in a radial gradient in between.
Visual result		

Center Color

Defines the center color. The opacity of the center color decreases from the center point to the outlying area.

Radial Color

Defines the radial color. The opacity of the radial color increases from the center point to the outlying area.

OpenGL Flip Effects

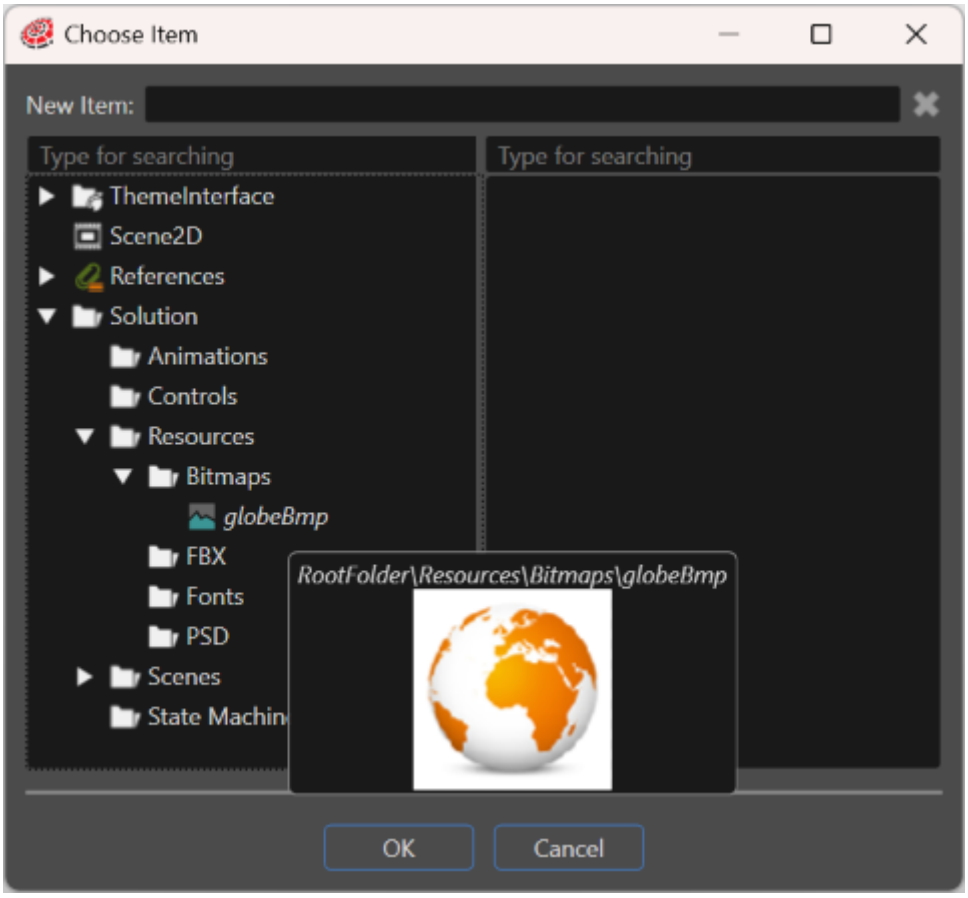
> Note that the OpenGL Flip Bitmap Brush Effect is an OpenGL effect and only available on platforms supporting OpenGL.

GlFlipBitmapBrushBlend

With this effect a bitmap can be defined that can be flipped horizontally or vertically.

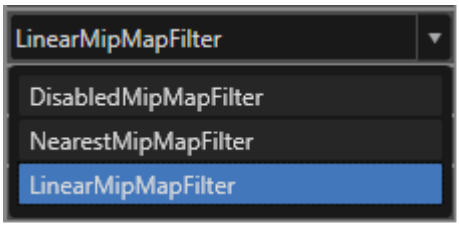
Image

An image can be defined for this effect that can be flipped in the desired direction using the flip properties. The image can be selected from the list of imported images in a dialog as follows:



MipMapFilter

MipMapping is used to save filtering work during texture minification by prefiltering the texture and storing it in smaller sizes. The applied MipMapFilter algorithm can be selected here. You can choose from no filtering at all, nearest neighbor filtering or linear filtering.



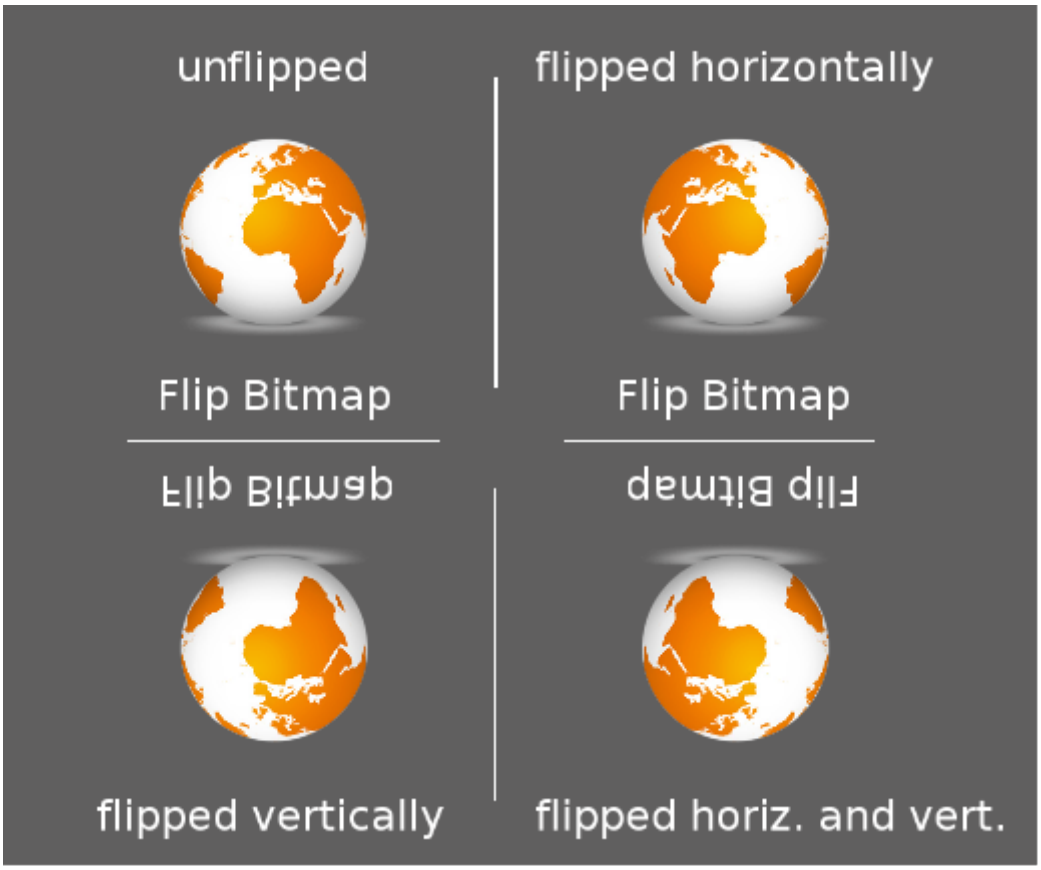
Flip

With the properties Flip H (flip horizontally) and Flip V (flip vertically) you can configure in which direction the image should be flipped. Flipping in both directions is also possible, which makes the image appear upside down.

Flip H	Flip V
	
Flip around the vertical axis	Flip around the horizontal axis

Example

Here is an example on horizontal and vertical flipping using the GlFlipBitmapBrushBlend effect.

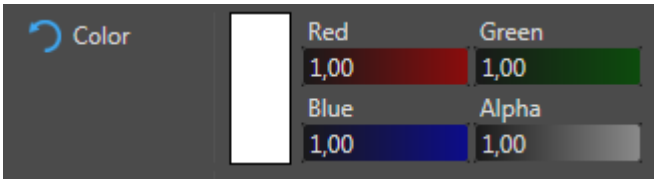


GlFlipBitmapBrushColorBlend

The GlFlipBitmapBrushColorBlend effect is similar to the GlFlipBitmpaBrushBlend effect with the additional possibility to define a color to be blended with.

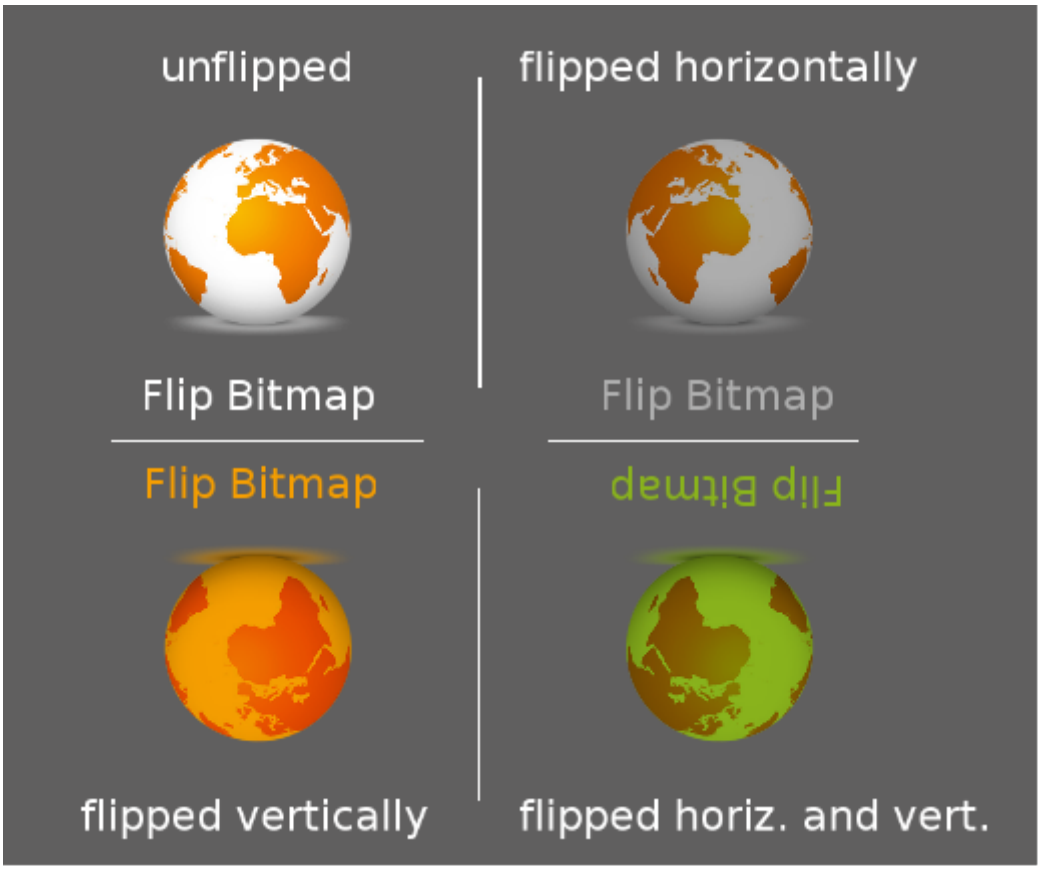
Color

The color can be chosen in the color palette dialog where you can also define and save the custom colors you need for your project. The default blending color after applying this effect is white.



Example

Vertical and horizontal flipping using the GlFlipBitmapColorBrushBlend effect can look as follows:


unflipped
flipped horizontally
Flip Bitmap
Flip Bitmap
Flip Bitmap
Flip Bitmap
flipped vertically
flipped horiz. and vert.

OpenGL Drop Shadow Effects

> Note that the OpenGL Drop Shadow Effects are OpenGL effects and only available on platforms supporting OpenGL.

> Be advised that the OpenGL Drop Shadow Effects are rendered twice and therefore consume performance.

GlDropShadowBitmapBrushBlend

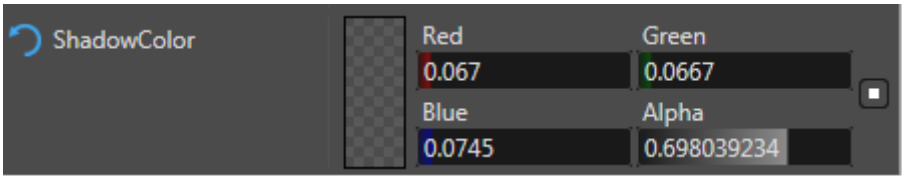
With this effect an image can drop a shadow that is defined by the alpha channel of the image, the shadow color, the light angle, distance and shadow scale.

Image

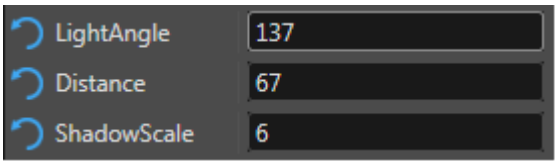
Here you define the image that should be displayed. The image can be selected from the list of imported images in the "Choose Item" dialog.

Shadow Color

The color of the shadow can be entered using the input fields. Alternatively, you can click on the color field on the left, which opens the "Color Palette" Window where you can pick the desired color from a color palette.



More Shadow Properties

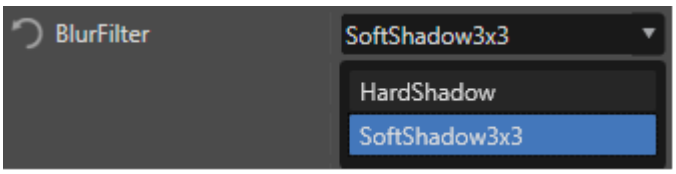


Light Angle:	Angle of virtual light that creates the shadow effect.
Distance:	Distance in pixels in which the shadow is cast at the given angle.
Shadow Scale:	The size of the shadow in pixels.

Blur Filter

With this property the blurriness of the shadow can be defined. There are two options available:

Hardshadow	no blurring of the shadow at all
SoftShadow3x3	creates a soft blurry shadow



GlDropShadowBitmapBrushColorBlend

The OpenGL Drop Shadow Bitmap Brush Color Blend effect adds color blending to the GlDropShadowBitmapBrushBlend effect. This means, the visible image drops a configurable shadow and additionally, a color can be defined to blend the visible image with.

> This effect is designed for the use with the text. Do not use with any text cache type other than BitmapCache!

Color

This property defines the constant color for modulating all input color channels (RGBA). This means, the displayed image is blended with the color configured with this property according to the Blending options configured below.

Blending Options

To configure the color blending, several color blend options are available.
More details regarding color blending properties in CGI Studio can be found at the SolidColorBrushColorBlend effect.

Further information about blending in general are given in chapter Blending.

Drop Shadow Properties

The configurable properties for the dropped shadow are the same as for OpenGl Drop Shadow Bitmap Brush Blend effect:

- Shadow Color
- Light Angle
- Distance
- Shadow Scale
- Blur Filter

OpenGL Outline Effects

> Note that the OpenGL Outline Effects are OpenGL effects and only available on platforms supporting OpenGL.

GlOutlineBitmapBrushColorBlend

The OpenGl Outline Bitmap Brush Color Blend effect draws an outline to the configured image according to the performed configurations and additionally blends the image with the configured color.

> This effect is specially designed for the use with text. Do not apply it to any text cache type other than BitmapCache!

Image

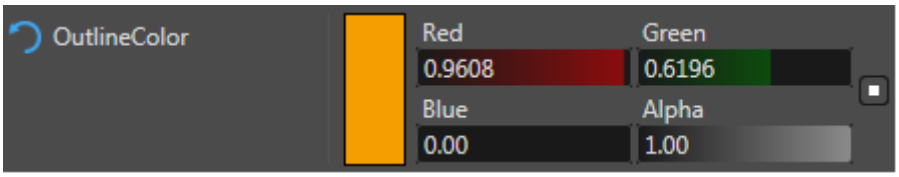
Here you define the image that should be displayed. The image can be selected from the list of imported images in the "Choose Item" dialog.

Outline Width

This property configures the width of the outline.
Please note that this width shall not exceed the width of the thinnest lines that are outlined.

Outline Color

The color of the outline can be configured with this color property. The outline's color can be entered using the input fields. Alternatively, you can click on the color field on the left can, which opens the "Color Palette" Window where you can pick the desired color from a color palette.



Color

This property defines the constant color for modulating all input color channels (RGBA). This means, the displayed image is blended with the color specified by this property, based on the Blending options configured below.

Blending Options

To configure the color blending, several color blend options are available.
More details regarding color blending properties in CGI Studio can be found at the SolidColorBrushColorBlend effect.

Further information about blending in general are given in chapter Blending.

OpenGL Multi Texture Brush Blend

GlMultiTextureBrushBlend

This effect allows to define two textures. These two textures are blended using a blend factor.

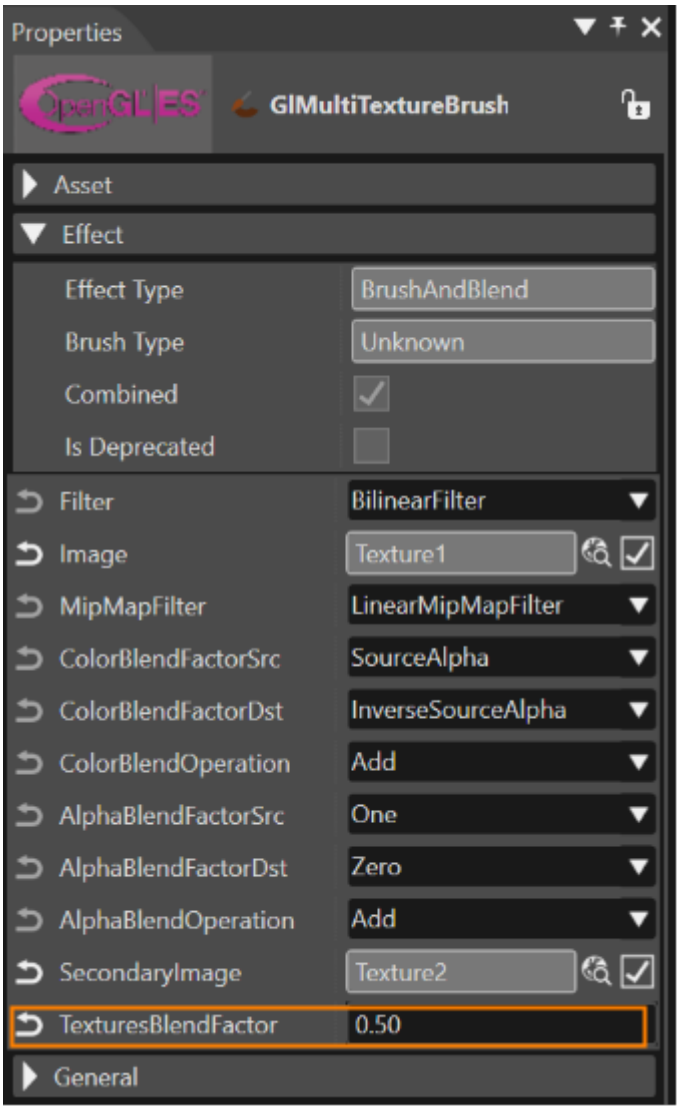
Image and Secondary Image

Here you can define the first image.

The second image can be configured with the property "SecondaryImage".

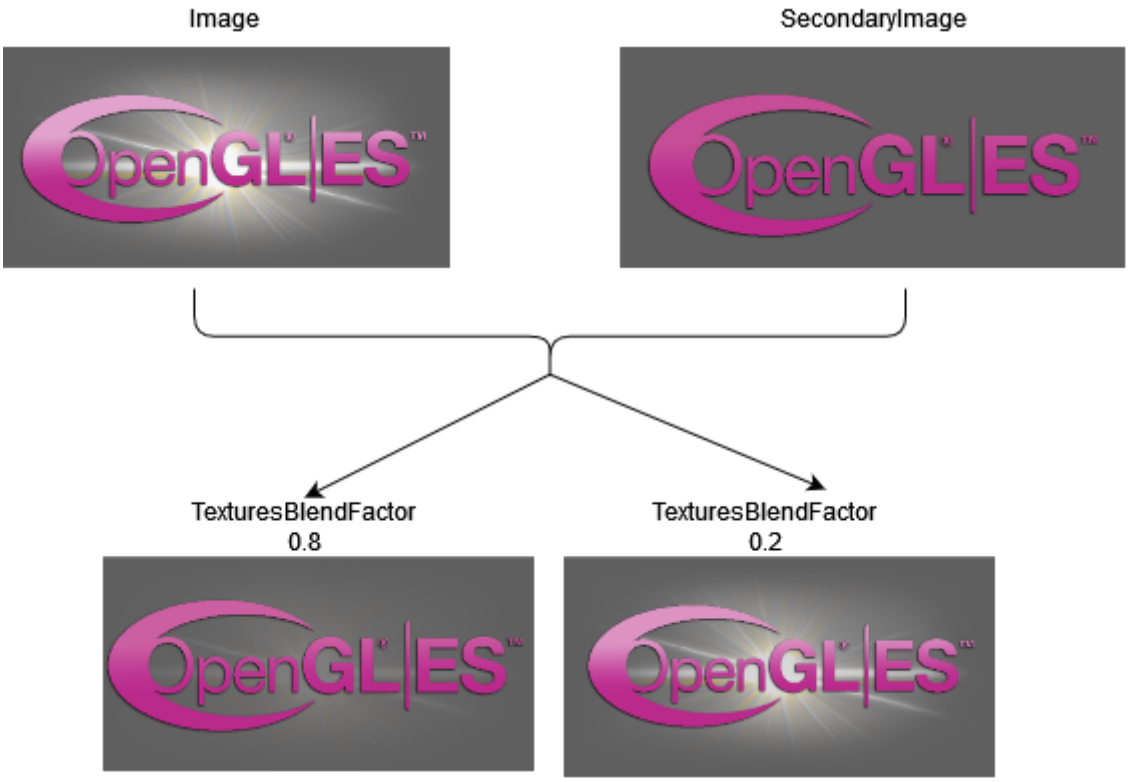
Blend Factor

The given images will be blended according to the TexturesBlendFactor.



The value of the TexturesBlendFactor ranges between [0...1].

E.g. 0.2 would be 20% of Image and 80% of Secondary Image.



This OpenGL effect demonstrates the usage of the API for 2D Effects on 3D Devices.

Detailed basic information on this effect can be found here.

MVG Effects

MVG effects are used to render MVG Nodes. The MVG effects basic effect properties are LineWidth and StrokeColor.

When an MVG node is created, its corresponding MVG effect is automatically applied by default.

MVGArcBrush

This is the default effect applied to an MVG Arc Node, and it is used to render the node's visual representation. The most important properties of this effect are Size as well as AngleStart and AngleEnd.

MVGEllipseBrush

This is the default effect applied to an MVG Ellipse Node, and it is used to render the node's visual representation. Its most important properties are Size, AngleStart and AngleEnd, FillColor and PaintMode.

MVGPathBrush

This is the default effect applied to an MVG Path Node, and it is used to render the node's visual representation. The appearance of the path is defined by properties such as FillColor and PaintMode. The path's shape is determined by the Segments and Data properties.

MVGRectBrush

This is the default effect applied to an MVG Rect Node, and it is used to render the node's visual representation. Properties such as Size, FillColor and PaintMode define the MVG Node's appearance. Additionally, each of the rectangle's four corners can be individually configured with a specific radius.

Custom Shader for 2D Effects on 3D Devices

Purpose

2D Effects on 3D devices are realized by usage of shader in the RenderDevice2D for implementation of 2D over 3D.
Several times, available effects for such devices have been enhanced. Please refer to Open GL related Effects below for more information.

- OpenGL Solid Color Brush Mask Effect
- OpenGL Bitmap Brush Mask Effects
- OpenGL Gradient Brush Effects
- OpenGL Flip Effects
- OpenGL Drop Shadow Effects
- OpenGL Outline Effects

For more flexibility and the option, to write custom 2D effects with a custom shader, the API has been enhanced:

- It is possible to provide a custom shader source from an effect implementation
- It is also possible to transfer uniforms (definition and value) from an effect implementation

The API

The API is in class `Candera::Internal::Context2DOver3DDevicePool`:

```
/**
 *  Set a custom program.
 *  @param vertexShader  The vertex shader code
 *  @param pixelShader  The pixel shader code
 *  @param uniforms  An array with uniform definitions (@note count is limited)
 *  @param uniformCount  Number of uniform definitions
 *  @param guid  A guid to distinguish custom programs
 */
void SetCustomProgram(const void* vertexShader, const void* pixelShader, CustomUniform
uniforms[],
□□□□□  UInt8 uniformCount, FeatStd::Internal::Guid const& guid);
```

```
/**
 *  Reset usage of a custom program.
 */
void ResetCustomProgram();

/**
 *  Check if a custom program is set.
 *  @return  True, if a custom program is set. False, otherwise.
 */
bool HasCustomProgram() const;
```

> The API is internal and shall only be used by experienced users!

Furthermore, the RenderDevice2D API has been enhanced to set a second surface (texture).

```
/**
 * List of surface types.
 */
enum SurfaceType{
DestinationSurface = 0,   ///< Destination surface.
SourceSurface = 1,        ///< Source surface.
MaskSurface = 2,          ///< Mask surface.
SecondSourceSurface = 3   ///< Further Source surface.
};
```

GlMultiTextureBrushBlend

A new effect, `GlMultiTextureBrushBlend`, has been added with the purpose to demonstrate the usage of the new API.

> The appearance of this effect in Scene Composer can be viewed on page Open GL Multi Texture Brush Blend

This effect uses the following shader code in its implementation:

```
static const Char* s_vertexShader = {
    "#ifndef GL_ES\n"
    "#define lowp\n"
    "#define mediump\n"
    "#define highp\n"
```

```
    "#define precision\n"
    "#endif\n"
    "precision highp float;\n"

    "attribute vec4 a_Position;\n"
    "attribute vec4 a_TextureCoordinate;\n"

    "uniform mat4 u_Transform;\n"
    "uniform mat4 u_TexTransform;\n"

    "varying vec2 v_TexCoord;\n"

    "void main(void)                                                \n"
    "{\n"
    "   gl_Position = a_Position * u_Transform;                     \n"
    "   v_TexCoord.xy = (a_TextureCoordinate * u_TexTransform).xy; \n"
    "}\n"
};

static const Char* s_fragmentShader = {
    "#ifndef GL_ES\n"
    "#define lowp\n"
    "#define mediump\n"
    "#define highp\n"
    "#define precision\n"
    "#endif\n"
    "precision highp float;\n"

    "uniform sampler2D u_Texture;\n"
    "uniform sampler2D u_Texture2;\n"
    "uniform vec4 u_ConstTexColor;\n"
    "uniform float u_TexturesBlendFactor;\n"

    "varying vec2 v_TexCoord;\n"

    "void main(void)\n"
    "{
\n"
    "    vec4 color = u_ConstTexColor;
\n"
    "    vec4 texCol = texture2D(u_Texture, v_TexCoord) * u_TexturesBlendFactor;
```

```
\n"
    "    vec4 texCol2 = texture2D(u_Texture2, v_TexCoord) * (1.0F - u_TexturesBlendFactor);
\n"
    "    gl_FragColor = color * (texCol + texCol2);
\n"
    "}\n"
};
```

> The shader code here is written for OpenGL ES 2.0 on purpose! The sample effect shall run also on devices supporting only this version.

The core functionality inside the `GlMultiTextureBrushBlend::Render` method:

```
static_cast<void>(ActivateSecondaryImage(output));

Internal::Context2DOver3DDevicePool& pool =
Internal::Context2DOver3DDevicePool::GetInstance();
FeatStd::Internal::Guid guid("8D86313D-47B2-49BD-89C3-DBD24E23A5A1");

Float texturesBlendFactor = m_texturesBlendFactor.Get();
texturesBlendFactor = (texturesBlendFactor < 0.0F) ? 0.0F : texturesBlendFactor;
texturesBlendFactor = (texturesBlendFactor > 1.0F) ? 1.0F : texturesBlendFactor;

Internal::Context2DOver3DDevicePool::CustomUniform customUniforms[1] = {
{ Internal::Context2DOver3DDeviceProgram::FloatUniform, "u_TexturesBlendFactor",
texturesBlendFactor }
};

pool.SetCustomProgram(s_vertexShader, s_fragmentShader, customUniforms, 1, guid);

m_bitmapBrush.Render(input, inputArea, transform, node, output, outputArea);
if (pool.HasCustomProgram()) {
pool.ResetCustomProgram();
}

success = DeactivateSecondaryImage(output) && success;
```

As seen above, the value from the effect property `m_texturesBlendFactor` is normalized and then provided as uniform `u_TexturesBlendFactor`, which is used by shader code defined with `s_vertexShader` and `s_fragmentShader`.

> Parameters for the shader are provided as uniforms. For the user, the parameters are exposed as Effect properties!

Custom Effect Set

It is now possible to configure the list of Effects required by the application. A new project will be created in this regard, which will include the generated EffectSet and the necessary dependent source files.

CMake Flag

A new flag has been introduced to specify that a custom Effect Set shall be used: **CGIAPP_ENABLE_CUSTOM_EFFECT_SET**. This feature is disabled by default.

CMakeLists Template File

A CMakeLists.txt template file can be used as reference on how to configure the custom effect set and integrate it in the application. This is located in *cgi_studio_candera/templates/EffectLists.template.txt*

- CGIEFFECTS_LIST - refers to a list of Candera Effects that should be added to the generated EffectSet
- CGIEFFECTS_PROJECTNAME - specifies the project name

Integrate Custom Effect Set into application

- Create a new folder in the application source directory (eg: Effects) and add a *CMakeLists.txt* file inside, based on the template file mentioned above.
- Populate the list of needed effects and specify the project name.
- In the CMake configuration file of the application add the following line: CgiAddProject(Effects).
- Configure and build the application as usual.

Example CMakeLists file

> Do not include the "Candera" namespace, this will be automatically added when the EffectSet will be generated.

```
# Effect List
set(PRV_CGIAPP_EFFECT_LIBRARY_LIST
    GlRadialGradientBrushBlend
    BitmapBrushBlend
```

```
    BitmapBrushColorBlend
    ShadowBitmapBrushBlend
)

set(PRV_CGIAPP_EFFECTSET_SUBDIRS
    .
)

CgiGetCurrentDir(PRV_CUR_DIR)

# Enable Custom EffectSet flag
set(CGIAPP_ENABLE_CUSTOM_EFFECT_SET TRUE CACHE BOOL "Enable Custom Effect Set")

# Include EffectSet Generator
set(CGISTUDIO_EFFECTSET_CMAKE_INCLUDE "${FeatStd_DIR}/cmake/cgistudio/cgistudio_effectset_gene
if(EXISTS ${CGISTUDIO_EFFECTSET_CMAKE_INCLUDE})
    include(${CGISTUDIO_EFFECTSET_CMAKE_INCLUDE})
    # Create Custom EffectSet
    if(CGIAPP_ENABLE_CUSTOM_EFFECT_SET)
        CgiCreateEffectSet("EffectSet" "${PRV_CGIAPP_EFFECT_LIBRARY_LIST}")
    endif(CGIAPP_ENABLE_CUSTOM_EFFECT_SET)
else()
    message(FATAL_ERROR "EffectSet Generator '${FeatStd_DIR}/cmake/cgistudio/cgistudio_effects
endif()
```

> Is is not possible to generate an empty EffectSet. An appropriate error message will be shown during project configuration and further processing will be halted.

> The user takes responsibility of assuring that all effects which are needed by the used CGI Studio products are included. (eg.: Candera::BitmapBrushColorBlend is required by Candera::TextNode2D).