Delta Media Server

DeltaRealTime User Guide

User Guide

M360-1

7THSENSE

DeltaRealTime User Guide : User Guide

Printed: December 2020

This edition is for software version Delta 2.6 or later
Document ref.: M360-1

E: info@7thsense.one
W: 7thsense.one

7thSense Design Ltd	7thSense Design LLC, Michigan	7thSense Design LLC, Orlando
2 The Courtyard, Shoreham Road Upper Beeding Steyning West Sussex BN44 3TN UK	332 E Lincoln Ave Suite 100 Royal Oak, MI 48067 USA	4207 Vineland Rd Suite M1 Orlando, FL 32811 USA
T: +44 (0) 1903 812299	T: +1 248 599 2717	T: +1 407 505 5200

Overview

DeltaRealTime merges the high-fidelity playback of Delta Media server with real-time interactive assets (e.g. from Unity, or Unreal Engine) to create real-time high fidelity, interactive 3D scenes, whether for a game, exhibit, simulation environment or themed attraction.

DeltaRealTime achieves this by matching camera definitions for pre-rendered playback media and real-time frames, and then compositing pre-rendered and real-time content based on depth and mathematical definition (stencil) information, on a per pixel basis.

3D Depth

In order to merge playback and real-time sequences, we compare the depth buffer for each pixel of the real-time asset with that of the playback frame, to determine which is in front. In the basic mode of DeltaRealTime, any pixel of a real-time asset that is not in front, is not rendered, by using the alpha channel. In this way, real-time assets moving about in the playback frame will move in and out, and disappear behind features in the pre-rendered background. Depending on the range of depth values required, the depth buffers can be 8, 16 or 24-bit (see Playback and Real-time Media Formats 17). The greater the range, the better the effect.

Depth and alpha alone cannot blend the two layers and the visual result depends entirely on the properties and quality of the RGB playback and real-time imagery (see Basic (No Stencil) Mode 15).

Stencil feed

In the full implementation mode 6 of DeltaRealTime, the Delta compositor also takes account of a stencil feed, which provides further data in RGB channels to modify each pixel. If depth buffer comparison shows that the real-time pixel is in front, then the stencil is used to re-evaluate the real-time pixel: is it opaque, semi-transparent, or does it shade or brighten the playback pixel behind it? It could be any combination, depending on the R, G and B values in the stencil pixel. This gives the means of truly blending hi-fidelity interactive displays, using the power of Delta Media Server and the Delta compositing engine (see How the Stencil Works 20).

Terminology

The following terms are used in this guide:

Playback sequence
    Pre-rendered colour image media.

Real-time feed

Interactive gaming assets, to be introduced into the playback media (normally input into Delta through a capture card or Spout shared texture).

Depth sequence

A separate pixel-matched media sequence, carrying depth values for each pixel of the corresponding colour media. In this document, illustrations are portrayed as 8-bit depth, in greyscale from white (nearest) to black (deepest).

Stencil feed

Matched to the pixels of the real-time feed, per-pixel data values are carried in the three colour channels: alpha (in blue), semi-transparency (in green) and illumination (in red).

Culling

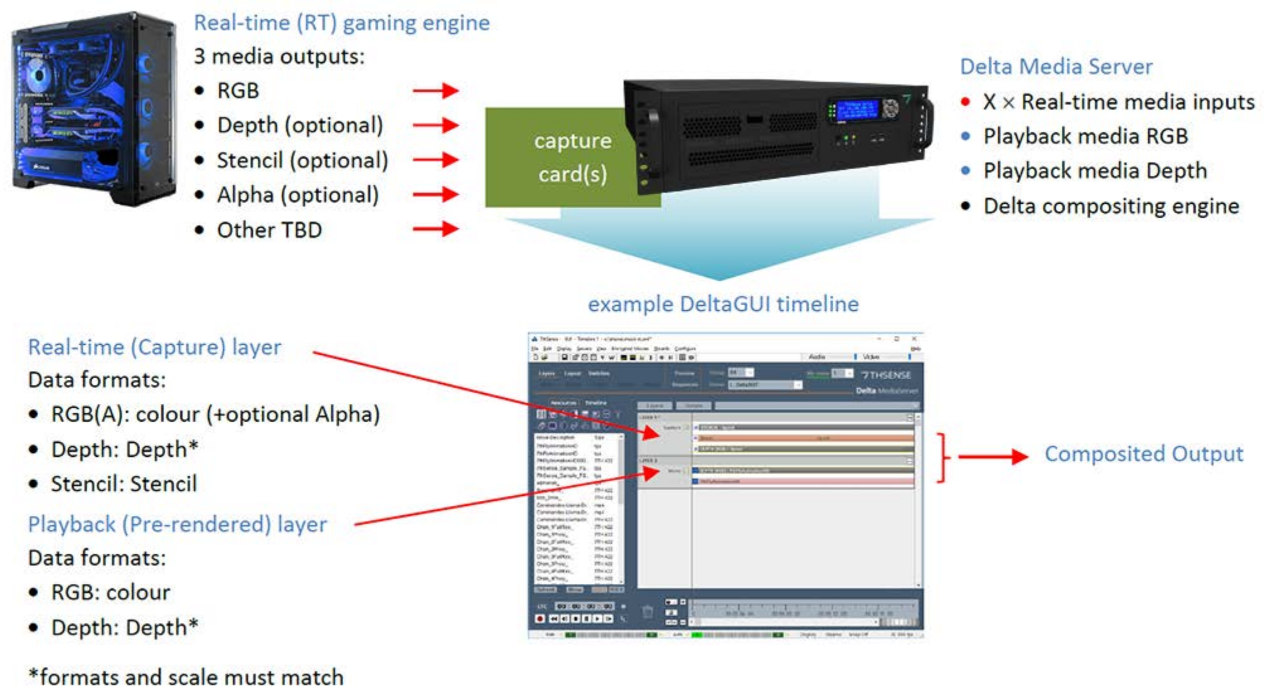
Where a stencil is *not* used, and a pixel in a real-time asset is (by depth comparison) to be occluded by the background, the real-time image pixel does not need to be rendered, so is simply 'culled', thus reducing processing load.

Modes of Use

A separate real-time gaming engine PC takes the load for rendering the interactive assets. In basic mode, it is possible to host the real-time rendering engine on the same Delta Media Server, but only for the simplest real-time scenes. A capture card, or cards, makes the real-time sequences and feeds available to the DeltaGUI timeline, where they are layered with the pre-rendered playback media.

Mode	External gaming engine	Delta Media Server	RGB image stream		Depth buffer feed		Effects stencil
			Playback	Real-time	Playback	Real-time	Real-time
Full capability	●	●	●	●	●	●	●
Basic mode	●	●	●	●	●	●	○
Delta server only	○	●	●	●	●	●	○

Full Mode With Depth and Stencil

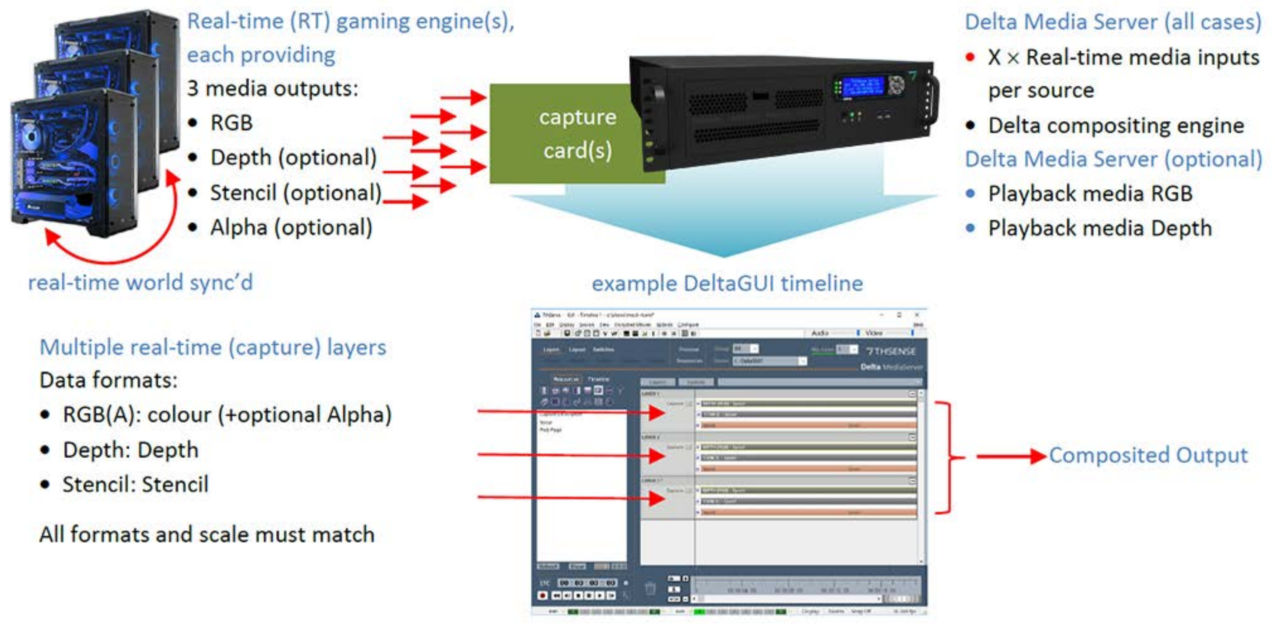


In this mode, the application of a stencil feed provides

- data values for illumination (e.g. blooming, fire, light sources) in the red channel
- data values for semi-transparency (e.g. mist, smoke, cloud) in the green channel, and
- data values for opacity (solid objects) in the blue channel.

For clarity, examples are given for each of these in isolation. However, an explosion might simultaneously cast a shadow on the ground from the sun, illuminate an object beside it, but be semi-transparent to a structure behind it.

The Delta compositing engine can equally be used to combine any combination: multiple gaming inputs only (including background features), and/or with multiple playback resources. If multiple gaming engines are used, these virtual cameras must be synced within the shared real-time world:


Real-time (RT) gaming engine(s), each providing
3 media outputs:
• RGB
• Depth (optional)
• Stencil (optional)
• Alpha (optional)
real-time world sync'd
capture card(s)
Delta Media Server (all cases)
• X × Real-time media inputs per source
• Delta compositing engine
Delta Media Server (optional)
• Playback media RGB
• Playback media Depth
example DeltaGUI timeline
Multiple real-time (capture) layers
Data formats:
• RGB(A): colour (+optional Alpha)
• Depth: Depth
• Stencil: Stencil
All formats and scale must match
Composited Output

Use this mode, combining all three feeds: media, depth and stencil, for the most realistic blending.

Here are three examples of how depth and each colour channel of the stencil are used together. For the explanation of how the stencil is used to calculate the final composition of each pixel, see How the Stencil Works [20].

- A. Solid object layering using depth, and blue in the stencil [7]
- B. Semi-transparent object layering, using depth, and green in the stencil [9]
- C. Illumination object layering, using depth, and red in the stencil [11]
- D. Putting it together: the RGB stencil [12]

A. Solid object layering using depth, and blue in the stencil

In these examples, if we were to composite multiple real-time feeds instead of with playback media, then a stencil would be required for each real-time feed. The illustrations here are deliberately flat and simple for clarity, but they are followed by a real-life worked example [12].

Playback: a street scene

Image sequence RGB layer

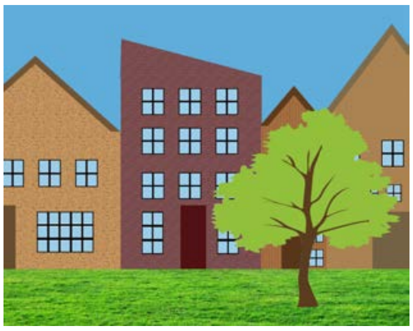
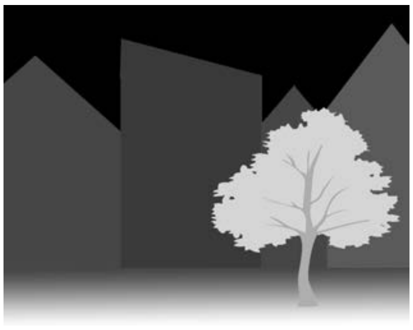
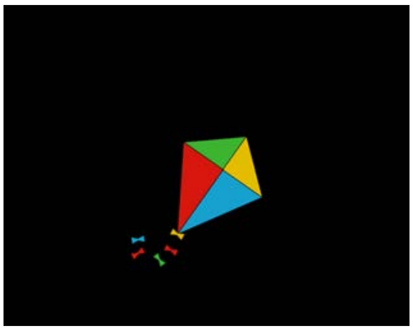


Image sequence depth layer

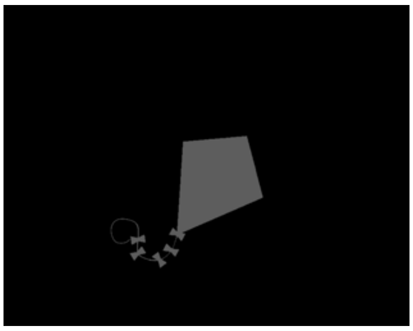


Real-time asset: an opaque kite

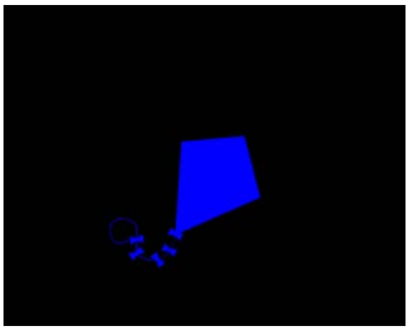
Video feed RGB layer



Video feed depth layer

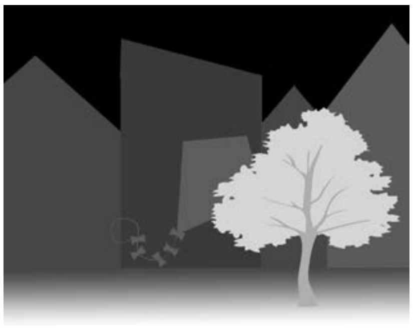


Video feed stencil layer
(uses blue for alpha)

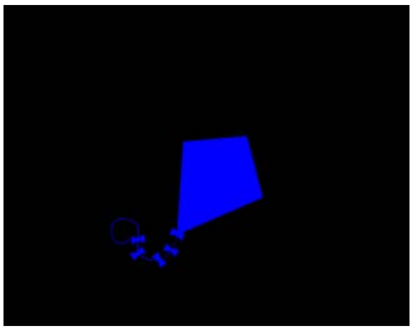


Output: in the composited scene, the kite flies between tree and houses

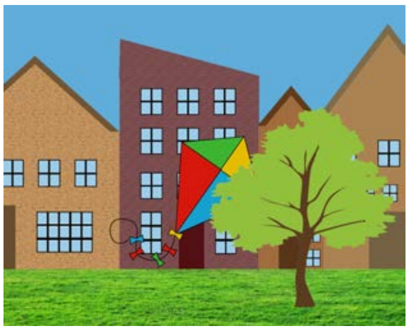
Compares depth:
which pixels are in front?



applies stencil (blue for alpha)



composites playback and real-time assets for output



B. Semi-transparent object layering, using depth, and green in the stencil

Playback: a street scene

Image sequence RGB layer

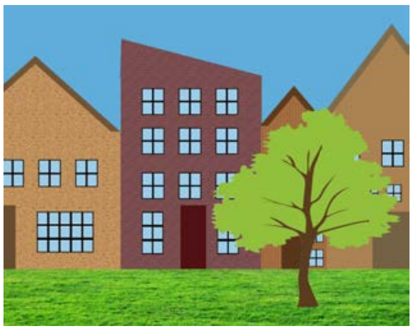
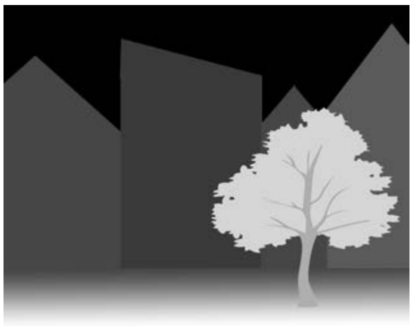
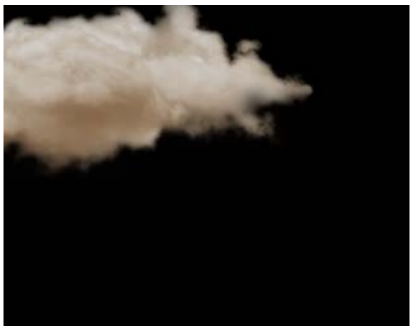


Image sequence depth layer



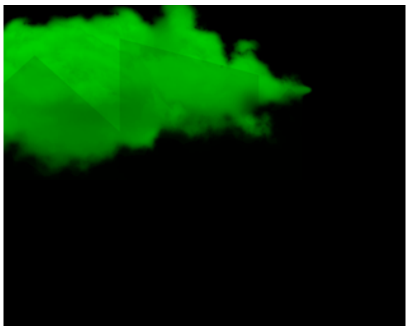
Real-time asset: a cloud of smoke and shadow

Video feed RGB layer



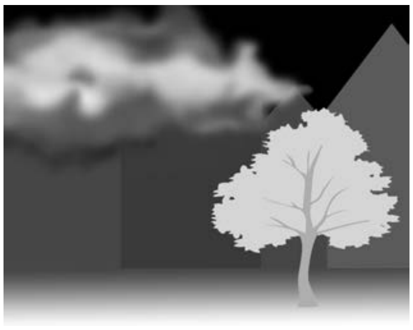
Video feed depth layer

Video feed stencil layer
(uses green for semi-transparency)

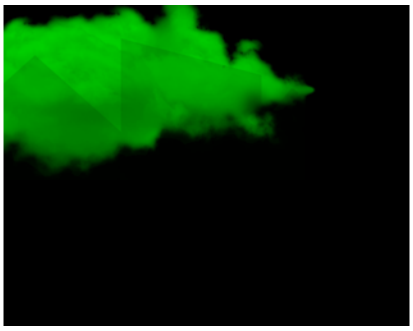


Output: in the composited scene the cloud is between tree and houses and casts a slight shadow on the facades

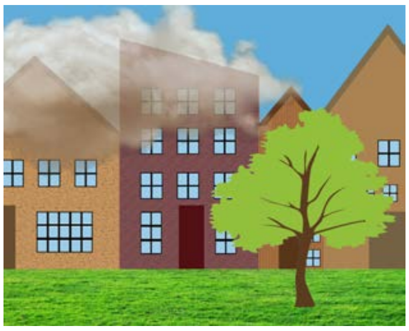
Compares depth:
which pixels are in front?



applies stencil (green for semi-transparency and shadow)



composites playback and real-time assets for output



C. Illumination object layering, using depth, and red in the stencil

Playback: a street scene

Image sequence RGB layer

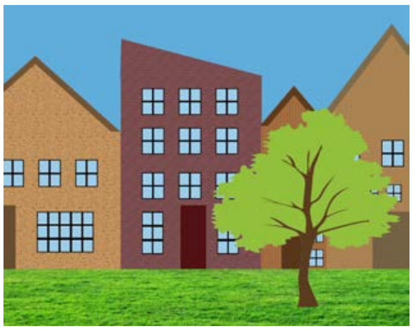
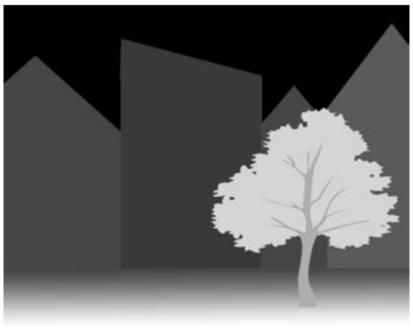
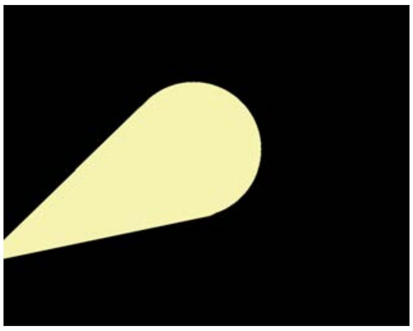


Image sequence depth layer

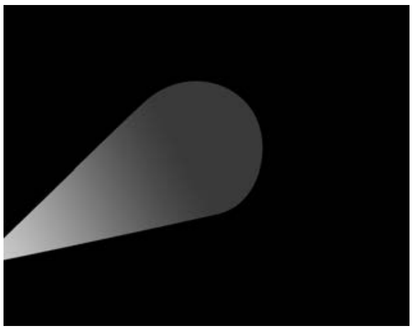


Real-time asset: a spotlight beam

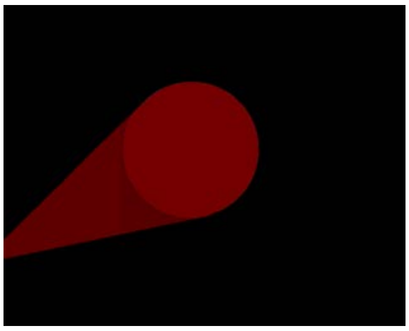
Video feed RGB layer



Video feed depth layer

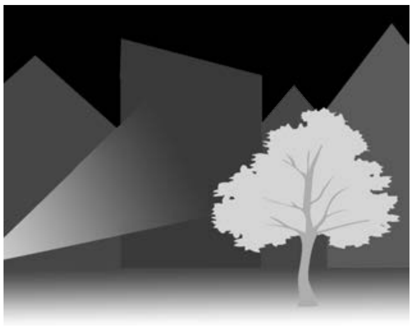


Video feed stencil layer
(uses red to modify lightness)

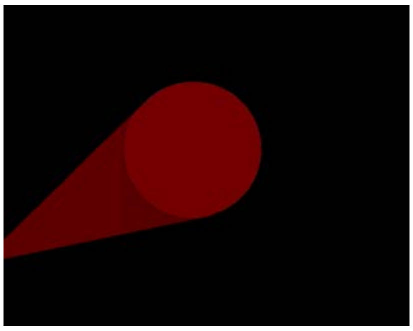


Output: in the composited scene a beam of light illuminates the façade

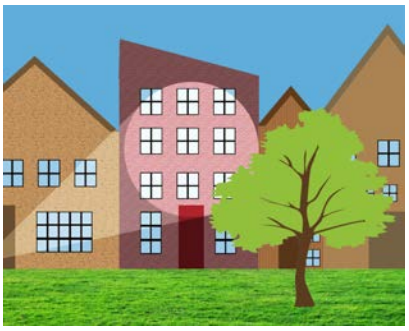
Compares depth:
which pixels are in front?



applies stencil (red for illumination)



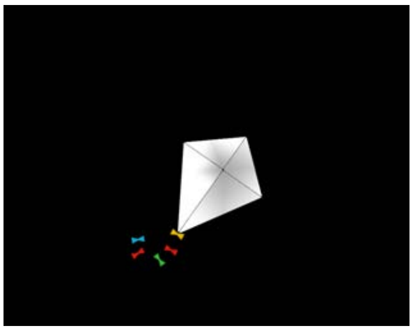
composites playback and real-time assets for output



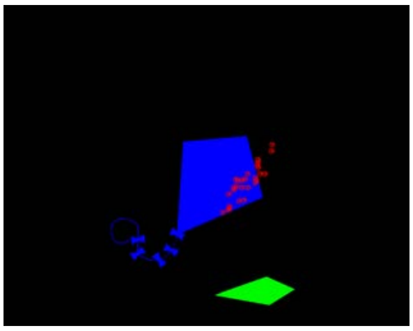
D. Putting it together: the RGB stencil

Imagine the kite in the first example were silver. Its opacity is held in the intensity of the blue channel. If sunlight were reflected off the kite onto the tree, this illumination would be rendered in the red channel. At the same time, it could cast a shadow on the ground; this shadow-cast would be rendered in the green channel.

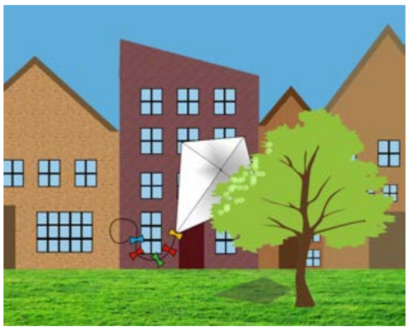
Real-time kite is reflective but opaque and casts shadow



stencil carries illumination (in red), shadow (in green), and alpha (in blue)



composites playback and real-time assets, adding light and shadow



Note: red and blue are shown separately for clarity. In reality, if viewed as RGB media, the colours would appear mixed.

Real life worked example

In this worked example, an undersea scene is populated with swimming creatures from a single gaming source. Multiple sources can be used; the principle is the same: all depths are compared, stencils are applied and the final composited result is displayed.

What the illustrations show

- Playback and real-time asset frames are shown correctly in colour as they will be seen. They are the only colour frames you see.
- Depth frames are greyscale (white is nearest). Areas with all-zero values for RGB are shown here as black.
- Stencil frames have RGB values per pixel only as required. Here they are shown in a black frame for clarity and contrast. These frames are never seen in colour, but are used in Delta as the stencil data format.

Playback

The playback media (as we move around the scene) comprises a (colour) media layer and a (greyscale) depth layer:

Image sequence RGB layer

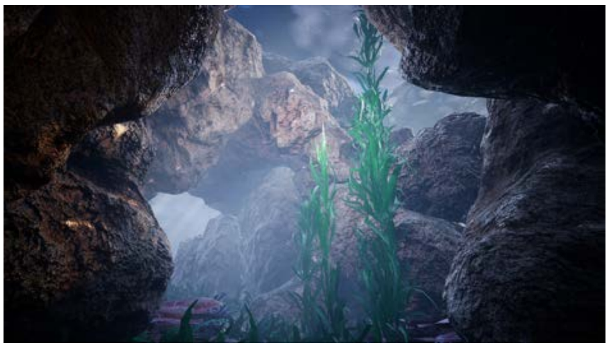
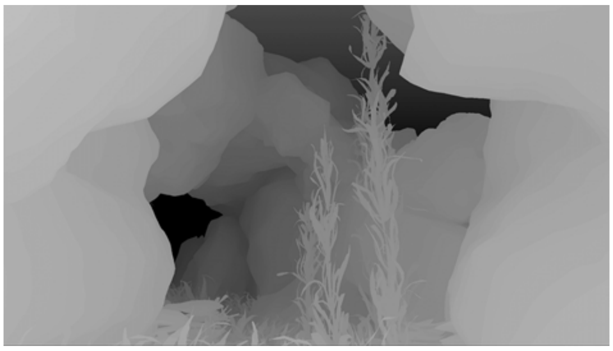


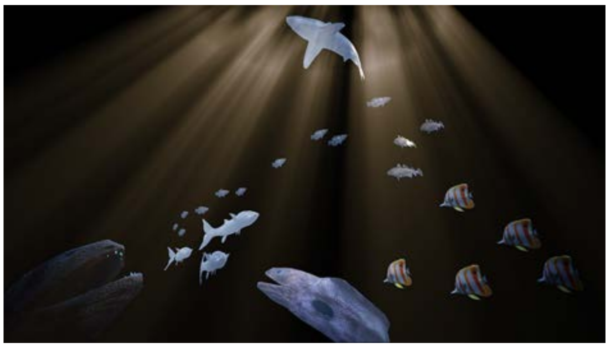
Image sequence depth layer



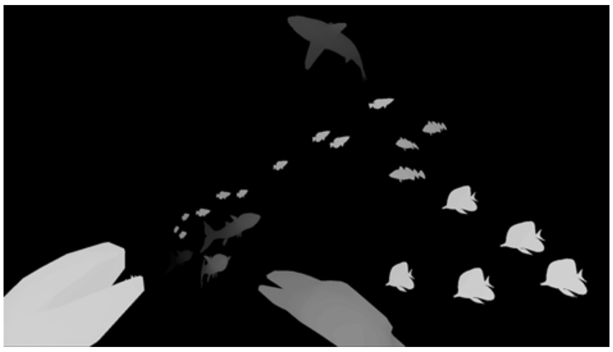
Real-time captured assets

Over this, the captured real-time assets are (colour) creatures media also has a (greyscale) depth layer:

Real-time sequence RGB layer

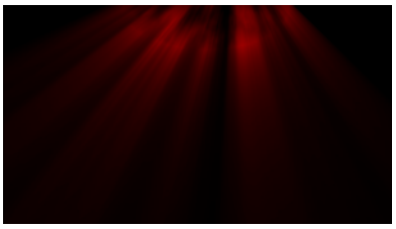


Real-time sequence depth layer



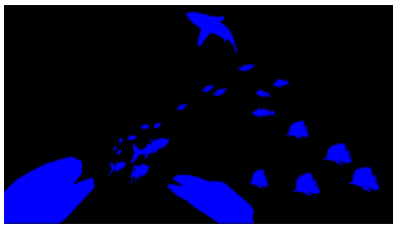
Finally, a stencil (data) layer comprising: highlights (red channel), shadow (green channel), and alpha transparency (blue channel) is added. First, here are the separate channels:

Highlights

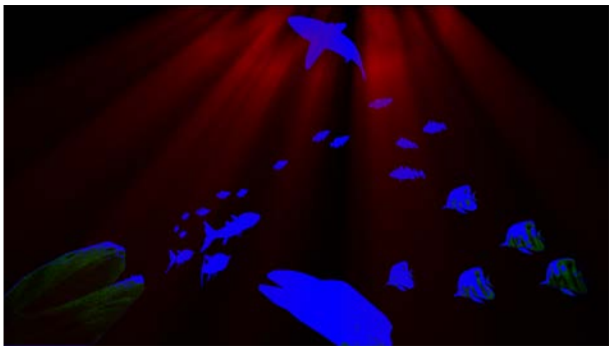


Shadows

Alpha



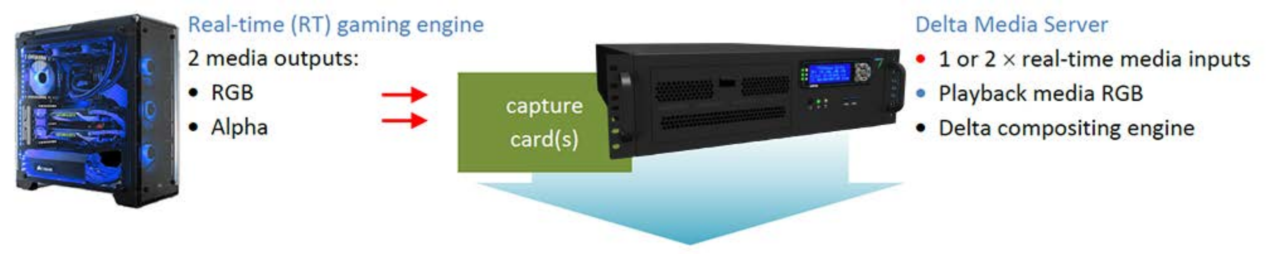
In practice, these are combined in a single RGB stencil image sequence, which is rendered in Delta with the data format: Stencil R(+) G(×) B(Alpha):



Composited show

Putting it together, the display becomes an interactive show, combining the playback environment with the real-time sea creatures:

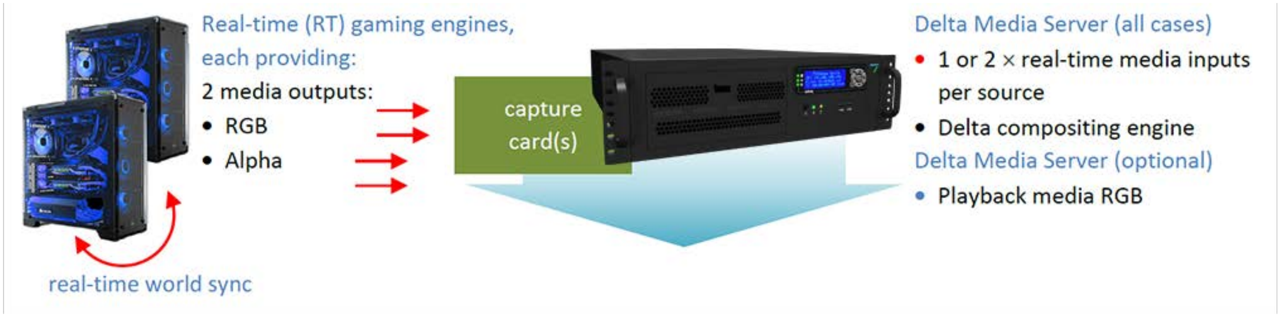
Basic (Depth, Alpha, No Stencil) Mode


Real-time (RT) gaming engine
2 media outputs:
• RGB
• Alpha
capture card(s)
Delta Media Server
• 1 or 2 × real-time media inputs
• Playback media RGB
• Delta compositing engine

In this mode, the real-time engine must use the 3D scene model when rendering. The alpha channel output takes depth into consideration to pre-cull real-time pixels where playback pixels are in front.

Chroma Key options are available if simple compositing is all that is required, to avoid the need for an alpha channel.

This could, as in full mode, composite any combination of multiple gaming inputs only, and/or with multiple playback resources, though without the full depth and stencil feeds, offers none of the stencil blending effects.


Real-time (RT) gaming engines, each providing:
2 media outputs:
• RGB
• Alpha
real-time world sync
capture card(s)
Delta Media Server (all cases)
• 1 or 2 × real-time media inputs per source
• Delta compositing engine
Delta Media Server (optional)
• Playback media RGB

Playback: a street scene

Image sequence RGB layer

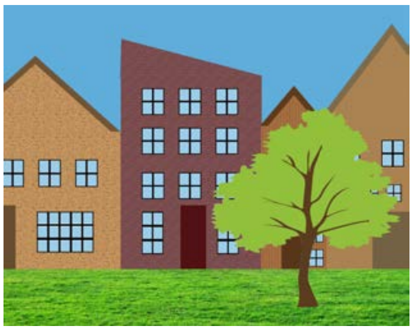
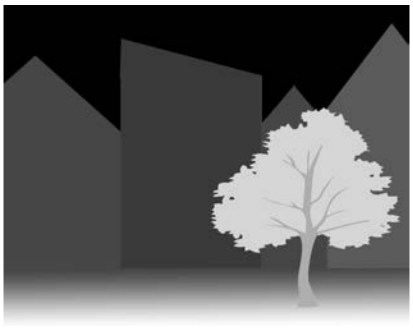
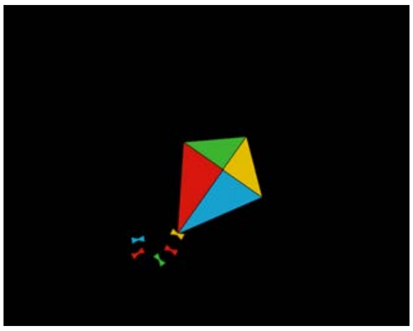


Image sequence depth layer

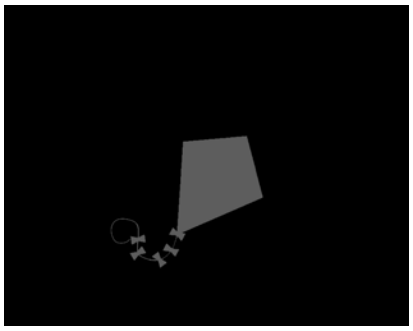


Real-time asset: an opaque kite

Video feed RGB layer

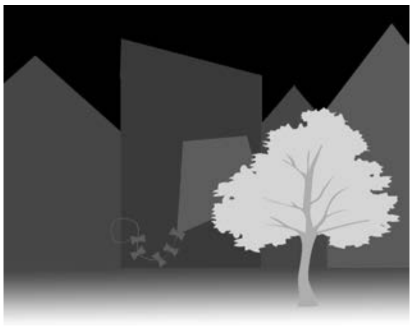


Video feed depth layer

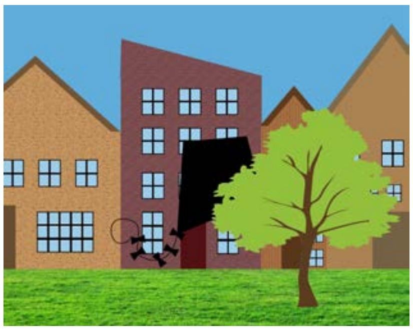


Output: in the composited scene, the kite flies between tree and houses

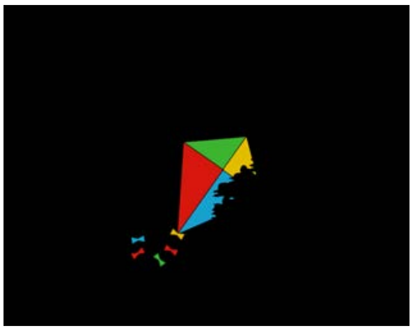
Compares depth:
which pixels are in front?



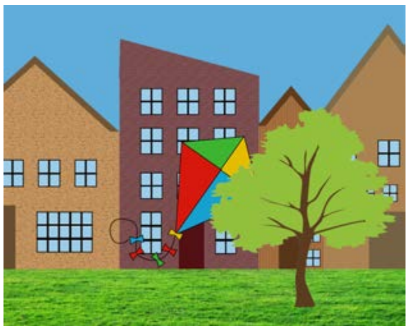
where real-time pixels are in front, culls pixels from playback assets, using alpha channel:



where playback pixels are in front, culls pixels from real-time assets, using alpha channel:



composites playback and real-time assets for output

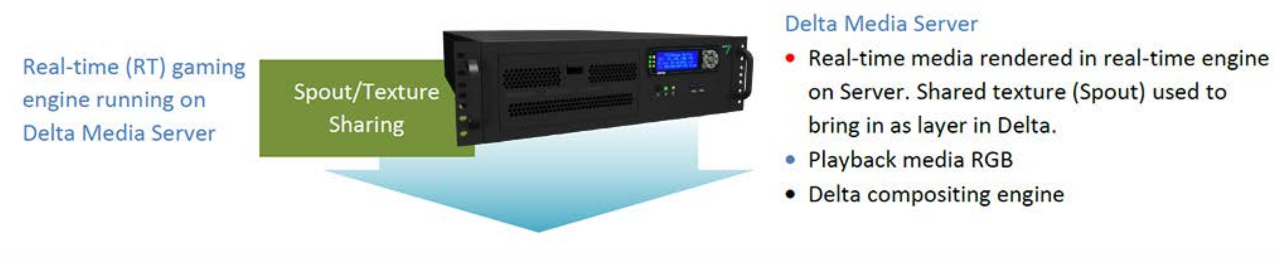


Minimal-demand mode: Delta Media Server only

(Onboard real-time rendering engine)

In this mode, the real-time engine must use the scene model when rendering. The alpha channel output takes depth into consideration to pre-cull real-time pixels where playback pixels are in front.

The real-time engine must not interfere with Delta Media Server loading. This mode is probably only suitable for simple / lower load real-time scenes.


Real-time (RT) gaming engine running on Delta Media Server
Spout/Texture Sharing
Delta Media Server
• Real-time media rendered in real-time engine on Server. Shared texture (Spout) used to bring in as layer in Delta.
• Playback media RGB
• Delta compositing engine

As in Basic Mode, no stencil is used. Instead, the real-time assets are culled against the playback media. Effects of semi-transparency and illumination or blooming are not available to the compositor engine. The result depends entirely on the properties of the RGB playback and real-time imagery.

Playback and Real-time Media Formats

Although not essential, is it recommended that all playback (pre-rendered) media have the same resolution and format.

Playback RGB(A) / colour frame sequence

Delta supports the following formats, depending on the performance specification of your server:

DPX 10 and 12-bit RGB 4:4:4 or 10-bit YUV 4:4:4

TGA 24-bit RGB or TGA 32-bit RGBA

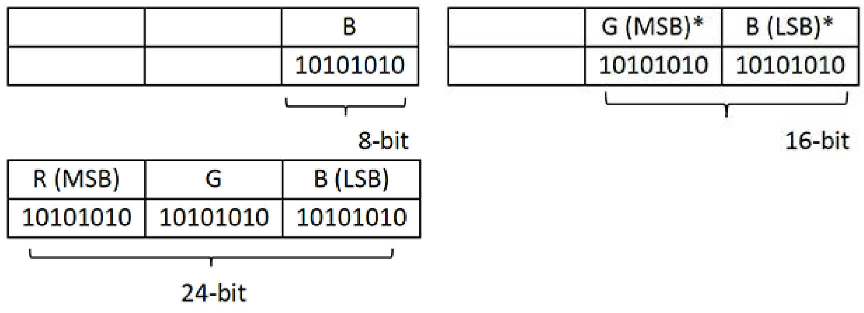
.7th 10-bit 4:4:4 or .7th 10-bit 4:4:4:4 or 10-bit 4:2:2 or 8-bit 4:4:4 or 8-bit 4:2:2

Playback and real-time depth frame sequence format

Depth information is required as a TGA 24-bit RGB, or .7th 444 image sequence for 16-bit depth. For 8-bit depth information an 8-bit TGA should be used.

DeltaRealTime currently supports 8, 16 or 24-bit depth data per pixel. 8-bit depth data should be delivered as an 8-bit greyscale TGA sequence, giving 256 levels of depth. 16-bit depth data should be stored in the Blue and Green channels of a 24-bit TGA, giving 2562 (65,536) levels of depth. 24-bit uses R, G and B, giving 2563 (16,777,216) levels of depth. When RGB and Depth media sequences are played together, we have a depth value and RGB colour value for every pixel in the playback media frame.

Real-time Depth values are packed in R, G and B



* Most/Least Significant Bit

The packing format and scale of the real-time and playback depth video feeds must match that of the playback RGB image media. See Playback depth frame sequence format for depth definitions.

Real-time video formats

All real-time inputs are captured into Delta Media server from a remote real-time rendering cluster. Delta Media Server can support Display Port, HDMI, 12-G,3-G SDI inputs at 8 or 10 bits per pixel.

Real-time RGB feed

The real-time colour feed should contain all real-time generated assets only. The remainder of the pixels in the frame should be black (0,0,0).

Real-time RGB(A) feed can also be from a shared texture (Spout) resource when real time engine is running on Delta Media Server.

Synchronisation

The per-frame camera definitions for playback and real-time views must **always** match. The pre-rendered playback media is the master, as its path is fixed.

A common method to ensure paths are matched is to export a spline of the pre-rendered eyepoints movement into the real-time engine. DeltaServer will periodically broadcast the current frame position over UDP (e.g. every 60 frames). The real-time engine maintains its synchronisation along the spline, based on the positional data from Delta Media server.

It is also recommended that the real-time graphics output and Delta Media Server graphics output be genlocked at all times. It is essential to have frame-accurate positional synchronisation when using DeltaRealTime.

Example Packets

An example of the protocol sent from Delta Media Server to the real-time source for synchronisation is shown below:

drt_start id=1

// id is the instance of DeltaRealTime to be started. The default id=1

drt_pos id=1 frame=50

//the prerendered scene has a number of frames as part of its movement spline. The frame variable indicates the frame position along the movement spline.

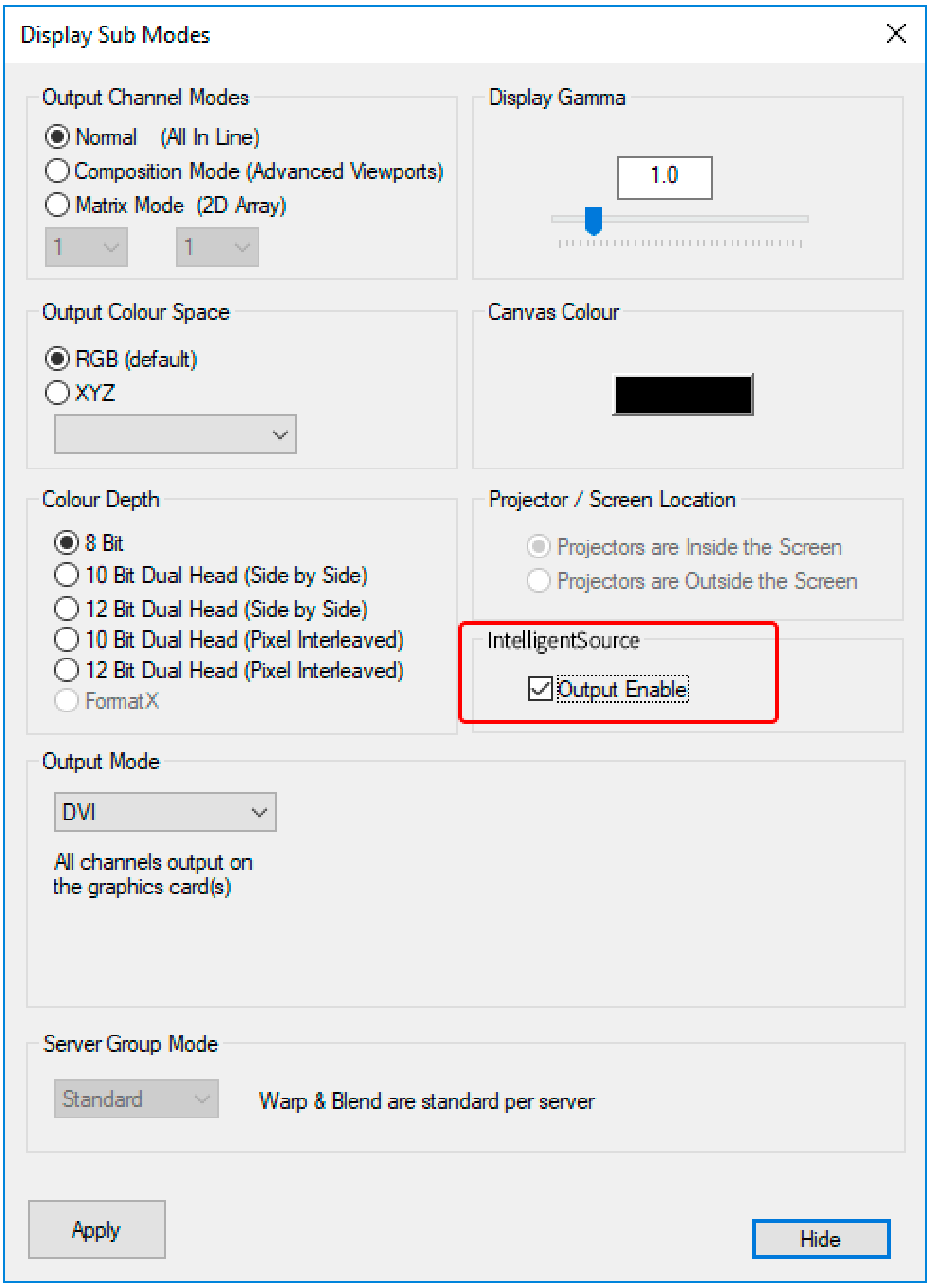
Notes

- All UDP commands will be terminated by carriage return.
- UDP unicast or broadcast packets can be used.

Real-time Fallback

In the event of a failure in the real-time feed, the show can be made to switch into a default playback mode using the IntelligentSource™ feature of Delta (from version 2.7). This will detect from the video signal if the real-time feed is no longer live and switch in a replacement feed so that a show flow is maintained.

IntelligentSource must be enabled in DeltaGUI, *Display > Output Setup*:


Display Sub Modes

Output Channel Modes
- [✓ Normal (All In Line)]
- Composition Mode (Advanced Viewports)
- Matrix Mode (2D Array)
- 1 1

Display Gamma
1.0

Output Colour Space
- [✓ RGB (default)]
- XYZ

Canvas Colour

Colour Depth
- [✓ 8 Bit]
- 10 Bit Dual Head (Side by Side)
- 12 Bit Dual Head (Side by Side)
- 10 Bit Dual Head (Pixel Interleaved)
- 12 Bit Dual Head (Pixel Interleaved)
- FormatX

Projector / Screen Location
- [✓ Projectors are Inside the Screen]
- Projectors are Outside the Screen

IntelligentSource
- [✓ Output Enable]

Output Mode
DVI
All channels output on the graphics card(s)

Server Group Mode
Standard Warp & Blend are standard per server

Apply Hide

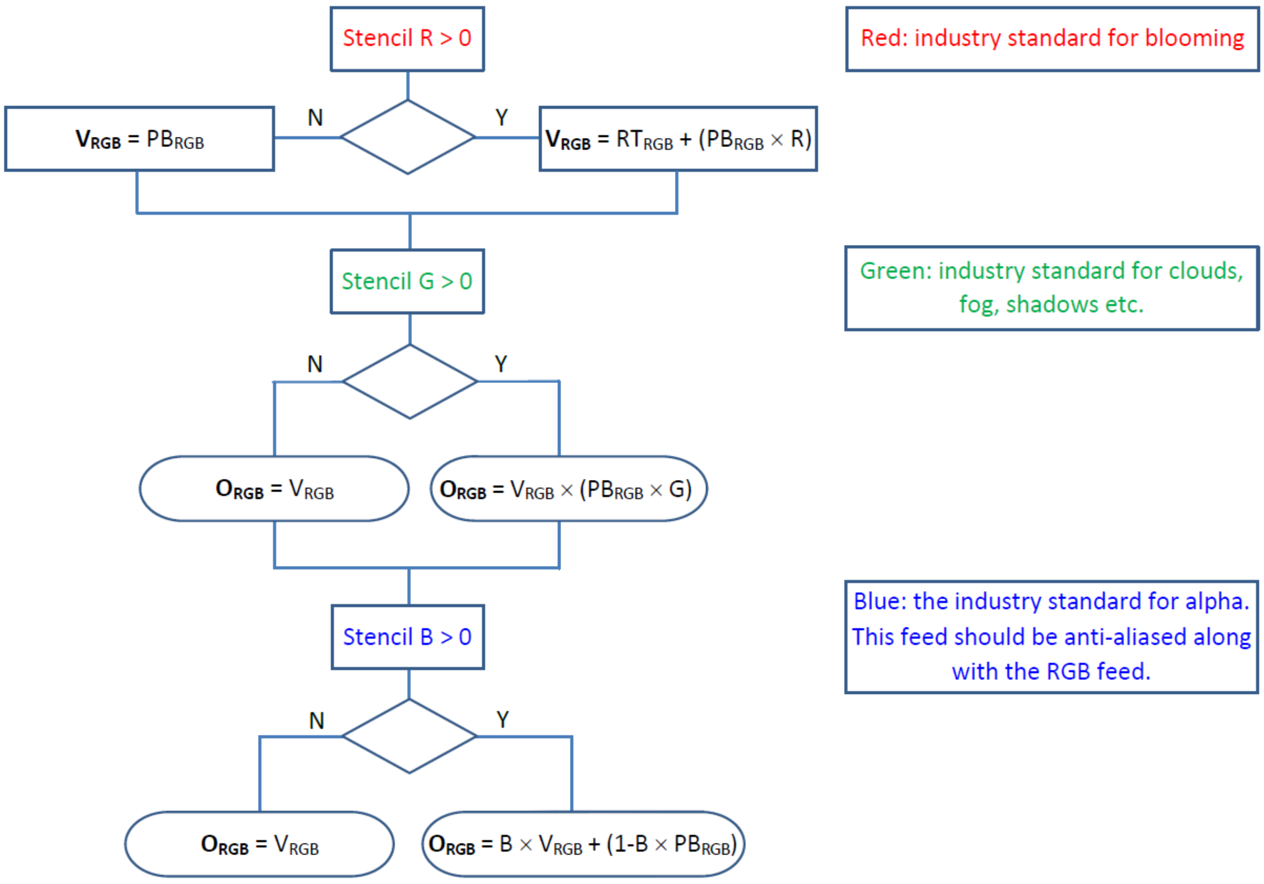
It can then be enabled in any capture resource, in its Resource Editor, Timeline tab.

How the Stencil Works

The application of a real-time stencil feed provides values for illumination (e.g. blooming, fire, light sources), for semi-transparency (e.g. mist, smoke, cloud), and for opacity (solid objects). The stencil defines the mathematical operations required, per pixel, to composite the real-time colour pixel and playback colour pixel.

After the depth comparison between playback and real-time pixels, the stencil feed is applied if required. As above, the following formulae are applied per pixel where:

PB = Playback image value
RT = Realtime input value
V = temporary calculated value
O = Output value


Stencil R > 0
N
Y
$V_{RGB} = PB_{RGB}$
$V_{RGB} = RT_{RGB} + (PB_{RGB} \times R)$
Red: industry standard for blooming
Stencil G > 0
Green: industry standard for clouds, fog, shadows etc.
N
Y
$O_{RGB} = V_{RGB}$
$O_{RGB} = V_{RGB} \times (PB_{RGB} \times G)$
Stencil B > 0
Blue: the industry standard for alpha. This feed should be anti-aliased along with the RGB feed.
N
Y
$O_{RGB} = V_{RGB}$
$O_{RGB} = B \times V_{RGB} + (1-B \times PB_{RGB})$

Real-time assets (e.g. from Unity or Unreal gaming engines) have their own colour media (RGB), depth buffer values and a stencil (per pixel mask) overlay. Together, we have depth value and colour value for every pixel in the playback and real-time input frames.

Considerations

- Depth information is mainly useful if playback (pre-rendered) assets are opaque.
- For semi-transparent pre-rendered assets, the stencil plane generation needs to take depth into account. This requires the real-time engine to have loaded all scene models / geometry.

Assembling a Show in DeltaGUI

DeltaRealTime operates in DeltaGUI 2.6 and above, which allows different data formats to be interpreted in movie and capture resources.

Layers

Capture assets must be above the playback media, so are placed in a higher layer.

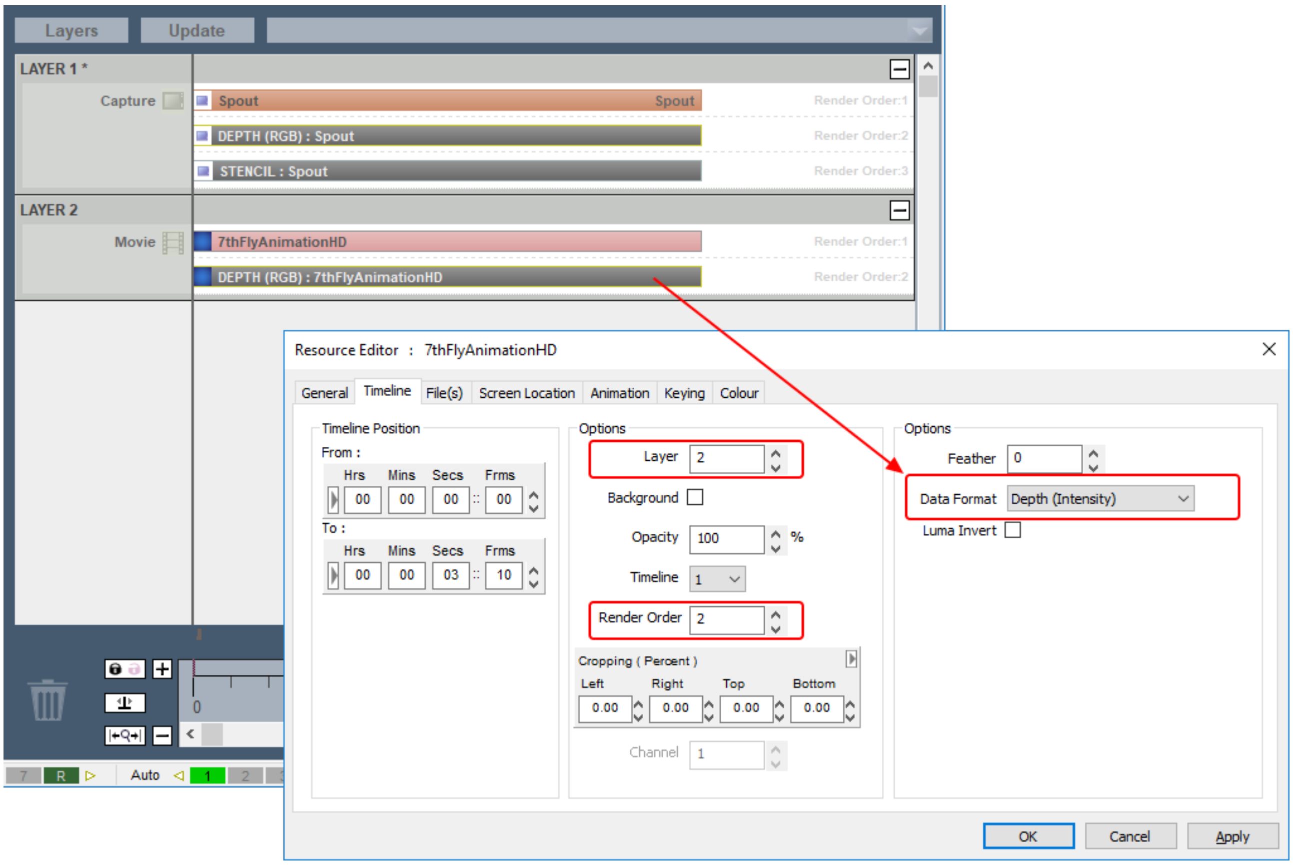
In this example, we have a movie playing in Layer 2, and alongside it a depth resource, providing per pixel data of the movie for comparison.

Render order

In order to process the data formatted resources within a layer in the correct order, apply this render order to each: Media, then Depth, then Stencil.

Data format: Depth

In this illustration, the depth media have been rendered in greyscale. In the properties of the depth resource, Timeline tab, we assign the data format property of Depth (Intensity):



Layers | Update

LAYER 1 *
Capture
- Spout — Spout — Render Order:1
- DEPTH (RGB) : Spout — Render Order:2
- STENCIL : Spout — Render Order:3

LAYER 2
Movie
- 7thFlyAnimationHD — Render Order:1
- DEPTH (RGB) : 7thFlyAnimationHD — Render Order:2

Resource Editor : 7thFlyAnimationHD

General | Timeline | File(s) | Screen Location | Animation | Keying | Colour

Timeline Position
From : Hrs 00, Mins 00, Secs 00, Frms 00
To : Hrs 00, Mins 00, Secs 03, Frms 10

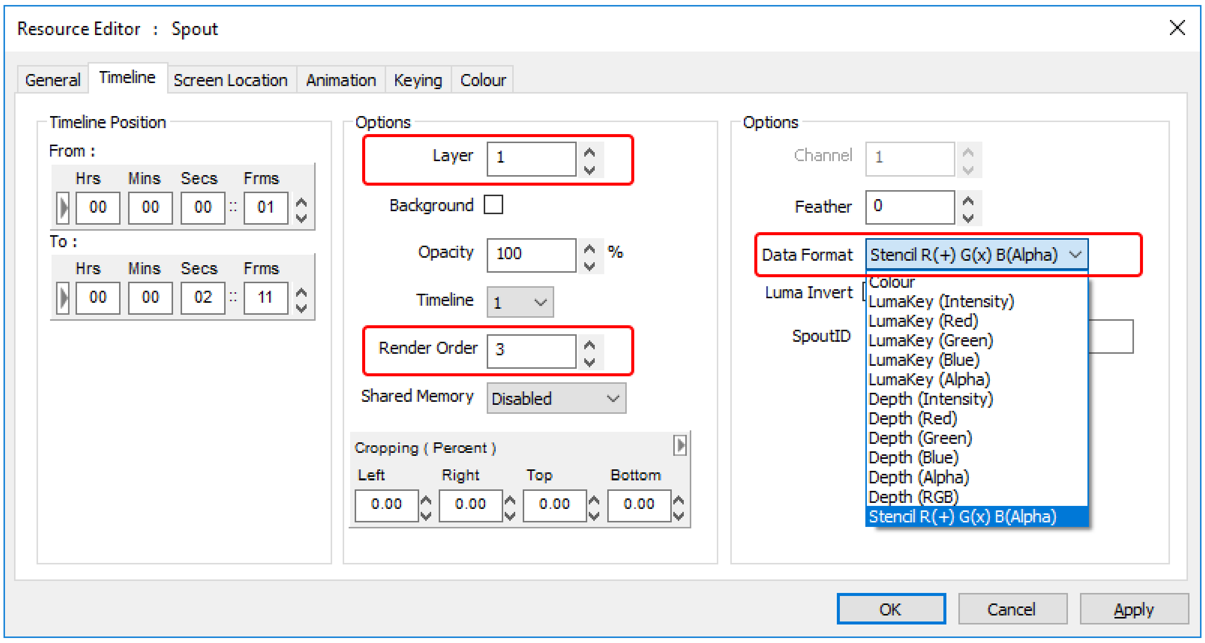
Options
Layer: 2
Background: ☐
Opacity: 100 %
Timeline: 1
Render Order: 2
Cropping (Percent): Left 0.00, Right 0.00, Top 0.00, Bottom 0.00
Channel: 1

Options
Feather: 0
Data Format: Depth (Intensity)
Luma Invert: ☐

OK | Cancel | Apply

Data format: Stencil

In the layer above this, we have placed three Spout capture resources – all from the gaming engine/PC: one for the real-time colour display, another depth layer for comparing the capture with the playback movie, and the stencil, which is used to recalculate each pixel for illumination, shadow and alpha as described in this guide. The stencil resource data format is ‘Stencil R(+) G(x) B(alpha)’:


Resource Editor : Spout

General | Timeline | Screen Location | Animation | Keying | Colour

Timeline Position
From : Hrs 00 Mins 00 Secs 00 Frms 01
To : Hrs 00 Mins 00 Secs 02 Frms 11

Options
Layer 1
Background
Opacity 100 %
Timeline 1
Render Order 3
Shared Memory Disabled

Cropping (Percent)
Left 0.00 Right 0.00 Top 0.00 Bottom 0.00

Options
Channel 1
Feather 0
Data Format Stencil R(+) G(x) B(Alpha)
Luma Invert
SpoutID

Colour
LumaKey (Intensity)
LumaKey (Red)
LumaKey (Green)
LumaKey (Blue)
LumaKey (Alpha)
Depth (Intensity)
Depth (Red)
Depth (Green)
Depth (Blue)
Depth (Alpha)
Depth (RGB)
Stencil R(+) G(x) B(Alpha)

OK Cancel Apply

Document Information

Date	Document edition	Software version	Revision Details	Author/Editor
September 2018	1	Delta 2.6	New Release	Andie Davidson

E: info@7thsense.one
W: 7thsense.one

7thSense Design Ltd
2 The Courtyard, Shoreham Road
Upper Beeding
Steyning
West Sussex
BN44 3TN
UK

T: +44 (0) 1903 812299

7thSense Design LLC, Michigan
332 E Lincoln Ave
Suite 100
Royal Oak, MI 48067
USA

T: +1 248 599 2717

7thSense Design LLC, Orlando
4207 Vineland Rd
Suite M1
Orlando, FL 32811
USA

T: +1 407 505 5200