

***frComp* Specification**

Frame Composer

Revision 2.0

27 July 2025

Copyright 2012 ArchiTek All Rights Reserved

Confidential and Proprietary

1.	Orverview	7	
1.1.	Introduction	7	
1.2.	Main Parameters	9	
1.3.	Implementation Parameters	10	
1.4.	Glossary	10	
1.5.	Others	11	
2.	Signal Lines	12	
2.1.	Control Bus Interface	12	
2.2.	PSS Interface	12	
2.3.	Memory Interface (Remapper Read Use)	13	
2.4.	Memory Interface (Pixel Cache Read Use)	13	
2.5.	Memory Interface (Blender Read Use)	14	
2.6.	Memory Interface (Blender Write Use)	14	
2.7.	Memory Interface (Steal Write Use)	15	
2.8.	Memory Interface (Histogram Write Use)	15	
2.9.	Memory Interface (Parameter Read Use)	16	
2.10.	Utility	16	
3.	Configuration and Operation	17	
3.1.	System Overview	17	
3.2.	Input and Output Data	21	
3.3.	Drive Interface (Initiator)	24	
3.4.	Notes on Fragmentation	26	
3.5.	Coordinate Generation (Polygon Shapes and Scanning)	28	
3.6.	Remapping (Remapper)	33	
3.7.	Matrix Transformation (Affine/Homography Transform)	35	
3.8.	Pixel Cache	37	
3.9.	Filter Data and Coefficient Selection	40	
3.10.	Preprocessing for Bayer Images	46	
3.11.	Filter	47	
3.11.1.	2D/2F/SAD/SSD Filter (SrcIn)	49	
3.11.2.	None-linear (SrcIn)	54	
3.11.3.	Mask Filter (SrcIn)	57	
3.11.4.	Hamming Filter (SrcIn)	61	
3.11.5.	Extrema Filter (SrcIn)	62	

3.11.6. Bitmap Filter (SrcIn)	64
3.11.7. Pattern Filter (SrcOut)	65
3.12. Envelope Processing	68
3.13. 3D CLUT (Color Space Conversion)	70
3.13.1. 1D Mode (Standard)	70
3.13.2. 1D Mode (Binary)	71
3.13.3. 2D Mode	71
3.13.4. 3D Mode	71
3.14. Pixel Processing (Extractor and Blender)	72
3.14.1. Extractor	73
3.14.2. Blender	74
3.15. Coordinate Extraction (Steal)	77
3.16. Histogram (Ver.BC)	78
3.17. Use of Blut	79
3.18. Address Masking	80
3.19. Input/Output Format	82
3.20. Internal Computation	87
3.21. Connection with pss	88
3.22. Performance	89
4. Register Description	90
4.1. Overview	90
4.2. Definition	90
4.3. Details	90
4.3.1.1. Reset Register	90
4.3.1.2. System Register	91
4.3.1.3. DitherHigh/Low Register	91
4.3.1.4. BayerMask0-3 Register	91
4.3.1.5. Utility Register	92
5. Command List Description	94
5.1. Overview	94
5.2. Definition	94
5.3. Details	96
5.3.1.1. MasterCntl Command	96
5.3.1.2. Vertex0-2 Command	104
5.3.1.3. PixelCntlB,G,R,A Command	105

5.3.1.4.	PixelKeyCRC Command	112
5.3.1.5.	PixelKeyMRC Command	114
5.3.1.6.	PixelKeyLow Command	115
5.3.1.7.	PixelKeyHigh Command	115
5.3.1.8.	PixelOrg Command	116
5.3.1.9.	PixelMod Command	118
5.3.1.10.	PixelDefault Command	119
5.3.1.11.	PixelConst Command	119
5.3.1.12.	SrcInInfo Command	120
5.3.1.13.	SrcInBase Command	123
5.3.1.14.	SrcOutInfo Command	123
5.3.1.15.	SrcOutBase Command	124
5.3.1.16.	SrcMapInfo Command	124
5.3.1.17.	SrcMapBase Command	128
5.3.1.18.	SrcSize Command	128
5.3.1.19.	SrcOffset Command	128
5.3.1.20.	DstInInfo Command	130
5.3.1.21.	DstInBase Command	131
5.3.1.22.	DstOutInfo Command	131
5.3.1.23.	DstOutBase Command	133
5.3.1.24.	DstMapInfo Command	134
5.3.1.25.	DstMapBase Command	135
5.3.1.26.	DstSize Command	135
5.3.1.27.	DstOffset Command	136
5.3.1.28.	CICntl Command	137
5.3.1.29.	COCntl Command	141
5.3.1.30.	HistCntl0 Command	143
5.3.1.31.	HistCntl1 Command	144
5.3.1.32.	ClutCntl Command	145
5.3.1.33.	BlutCntl Command	148
5.3.1.34.	StealCntl Command	149
5.3.1.35.	AffineCoef0-8 Command	151
5.3.1.36.	FilterCntlIn/Out Command	151
5.3.1.37.	FilterCntlOp Command	155
5.3.1.38.	FilterCoef00 Command (Coefficient Filter Mode)	158
5.3.1.39.	FilterCoef10-27 Command (Coefficient Filter Mode)	160

5.3.1.40.	FilterTable Command (Mask Filter Mode)	161
5.3.1.41.	FilterCenter Command (Mask Filter Mode)	161
5.3.1.42.	FilterAround Command (Mask Filter Mode)	161
5.3.1.43.	FilterReplace Command (Mask Filter Mode)	162
6.	Application Notes	163
6.1.	Overall Control	163
6.1.1.	Processing Unit	163
6.1.2.	Functional Orthogonality	164
6.1.3.	Processing Symmetry	164
6.1.4.	Polygon Rendering	165
6.1.5.	Scan Modufications	168
6.2.	Coordinate Operations	169
6.2.1.	Mapping Data	169
6.2.2.	Polar Coordinate Transformation	171
6.2.3.	Spherical Transformation	172
6.2.4.	Free-form Deformation	173
6.2.4.1.	Abstracion	174
6.2.5.	Affine Transformation	175
6.2.5.1.	Parameter Settings	175
6.2.5.2.	Translation	176
6.2.5.3.	Mirroring (Flip)	176
6.2.5.4.	Scaling	177
6.2.5.5.	Rotation	177
6.3.	Image Attributes	178
6.3.1.	Input Format	178
6.3.2.	Output Format	180
6.3.3.	Width and Address	180
6.3.4.	Attribute Conversion	181
6.4.	Filter Settings	181
6.4.1.	Filter Selection	181
6.4.1.1.	2D/2F Filter	182
6.4.1.2.	Arbitrary Coefficients and Interpolation	183
6.4.1.3.	Sobel Filter	184
6.4.1.4.	Canny Filter	186
6.4.1.5.	Bilateral Filter	188
6.4.1.6.	Cross-Correlation	190

6.4.1.7.	Thinning	191
6.4.1.8.	Scratch Correction	194
6.4.1.9.	Morphological Operations	196
6.4.1.10.	Feature Point Extracion	197
6.5.	Clut Confuguration	198
6.5.1.	Effects of Transformation	198
6.5.2.	3D Mode	199
6.5.3.	2D Mode	201
6.5.4.	1D Mode	202
6.5.5.	Input Value Range	202
6.5.6.	Specific Color Extracion	203
6.5.7.	Coordinate Transformation	205
6.6.	Extractor Configuration	205
6.6.1.	Binarization	205
6.7.	Blender Configuration	208
6.7.1.	Alpha Blending Configuration	208
6.7.2.	Handling Pixel Values Beyond 8 Bits	209

1. Overview

1.1. Introduction

- **Frame Composer (hereinafter referred to as frComp)** is a compact image processing engine that processes and transfers data from a Source image to a Destination image. Most functions are orthogonal and operate independently, allowing flexible combinations. Combined functions can be processed in a single pass, delivering performance of up to 4 elements per cycle multiplied by the number of combined functions.
- **Supported pixel formats** include 8-bit \times 4-element (32bpp), 8-bit \times 3-element (24bpp), RGB565/YUYV (16bpp), and half-precision floating point (Ver.C). The final accumulation stage supports 16-bit \times 2-pixel or 32-bit \times 1-pixel operations. Internally, data is processed using signed 9-bit \times 4-element format, except for filters using half-precision floating point. In filtering and other calculations, at least 4-bit fractional precision is maintained.
- For **Bayer images**, a 4×4 user-defined pattern enables arbitrary element extraction and interpolation using filtering (Ver.C).
- **Source and Destination coordinates** are independently derived from a reference coordinate, enabling flexible image transfers. Images are processed in fragments, divided into multiple lines. There is no performance degradation even when applying different processing contexts per fragment. By time-division processing of multiple tasks, simultaneous multi-tasking can be virtually achieved. The maximum coordinate size supported is 65536 (approximately $32 \times$ full HD width).
- By specifying **polygon shapes**, processing can be performed per triangle or parallelogram region rather than per fragment (Ver.C), reducing the burden of coordinate calculation and scanning on the host circuit.
- **Coordinate mapping** from Source to Destination is possible using mapping data in memory. Useful for feature point processing, lens distortion correction, etc. Mapping data can be compressed by $1/2^n$ ($n = 0-7$) and reconstructed using bi-linear interpolation, ensuring accuracy while saving data size.
- **Affine and Homography transformations** with floating-point precision using 3×3 matrices are supported. Operations include scaling, rotation, and deformation. When the polygon shape is a triangle, texture mapping based on memory references becomes possible. No size limitations are imposed. Rotation matrices can also be generated from angle information.
- Equipped with a **5×5 full-color Pixel Cache** and a **9×9 grayscale image cache**. Efficient memory access is performed even with irregular Source

coordinate movement. Arbitrary values can be assigned to out-of-boundary data.

- Supports **Point, Bi-cubic, Bi-linear, Non-linear filters**, and arbitrary coefficient 2D filters up to 5×5 . For grayscale, 9×9 2D filter processing is possible. Some arbitrary coefficient filters can apply Bi-linear interpolation simultaneously, and coefficients can incorporate table values based on the difference between center and surrounding pixels (Bilateral filter).
- Using another image's values as coefficients in a 5×5 2D filter enables **cross-correlation or auto-correlation**. For filters larger than 5×5 , multi-step processing with accumulation and correction is required. Using mapping functions, **SAD (Sum of Absolute Difference)** and **SSD (Sum of Squared Difference)** up to 5×5 are also supported (Ver.C).
- Supports **Non-linear filters** for selecting pixel values based on max, min, or median of specific elements. Median supports up to 3×3 kernel size, while max/min support up to 9×9 (5×5 for Ver.A).
- Equipped with a **Mask filter** for non-linear processing. It uses the state of the 8 surrounding pixels and the center pixel as an index to reference a table, enabling center pixel operations like dilation, erosion, thinning, and blending within kernels.
- Supports filters that select maximum/minimum values from **up to 8 layers of 3×3 kernels**, useful for feature point extraction.
- Includes a **distance filter** for extracting the nearest true point from the center in 1-bit Bitmap data.
- Allows **binary pattern generation** by comparing each pixel in a 9×9 kernel with arbitrary values (e.g., kernel center, specific coordinates, constants). These patterns can be evaluated using a downstream 3D Clut.
- Filter results can be evaluated under specified conditions to **write Source coordinates to memory**. Coordinates are written serially to reduce data volume.
- Enables **arbitrary color space transformations or function conversions** using memory-based 3D Clut (3D Color Look-Up Table). Any 3 input elements can be transformed into any 4 output elements, supporting RGB, YUV, HSV formats and more. Applications include HOG preprocessing, gamma correction, and pattern recognition (e.g., FAST).
- Combines pre- and post-filter pixel values to generate **masks or perform binarization**, including adaptive binarization using pixel thresholds.
- Supports **various binary operations** (e.g., α blending, squared sum, division) between Destination and Source images. Final results can undergo table-based conversion.
- For pixel computation results, **8-bit elements can be concatenated** to perform accumulation at 16/24/32-bit precision, supporting high-precision grayscale

image processing including negative values. For grayscale, half-precision floating point format is also supported (Ver.C).

- **Histograms** of the final image can be acquired per pixel element (Ver.B/C), with support for cumulative updates. Only the necessary number of results can be automatically written to memory.

1.2. Main Parameters

- **Memory Bus**
 - Remapper Read: 32-bit \times 3
 - Cache Read: 32-bit \times 13
 - Blender Read: 32-bit \times 1, Blender Write: 32-bit \times 1
 - Histogram Write: 32-bit \times 1
 - Steal Write: 32-bit \times 1
 - Command List Read: 64-bit \times 1
- **Throughput**
 - Up to 1 pixel / 4 elements / cycle, or 4 pixels / 1 element / cycle
- **Pixel Formats**
 - 8-bit components (Grayscale, Bayer)
 - 16-bit components (RGB565, ARGB1555, YUV422, half-precision floating point format)
 - 24-bit components (RGB888, YUV, etc.)
 - 32-bit components (ARGB8888, AYUV, etc.)
- **Mapping Data**
 - 16-bit integer format (two's complement; fractional position specified separately)
- **Coordinate Matrix**
 - Single-precision floating point format (32-bit)
- **Filter Coefficients**
 - Half-precision floating point format (16-bit)
- **Envelope Coefficients**
 - Half-precision floating point format (16-bit)
- **Histogram**
 - 32-bit (lower 24 bits valid) \times 256 entries per element
- **Extracted Coordinates**
 - 32-bit (upper bits: Y-coordinate, lower bits: X-coordinate) \times variable length
- **Clock**
 - Undefined (depends on implementation process)

1.3. Implementation Parameters

- The following section explains the parameters used in the hardware description.

Parameter Name	Description	Default Value
BLR	<ul style="list-style-type: none">• Radix of burst length for Command List reading• Configures the burst unit for 64-bit memory access	1 (4 and under)
BSR	<ul style="list-style-type: none">• Radix of burst length for data read/write operations• Configures the burst unit for 64-bit memory access	2 (4 and under)
BWLR	<ul style="list-style-type: none">• Burst length of memory used for external cache flush• Configured by summing the word length (e.g., 4 bytes per word); for a burst length of 4, the total corresponds to Radix 4	4

1.4. Glossary

- The following section explains the terminology used in this specification.

Term	Detail
Original	Refers to unprocessed data. The corresponding data path is referred to as SecOrg.
Modify	Refers to data processed by filters or similar operations. The corresponding data path is referred to as SecMod.
Clut	Abbreviation for Color Look-up Table. A table referenced using ARGB elements as keys. Includes 1D conversion for transforming each element individually,

	2D conversion for combinations of two elements, and 3D conversion for combinations of three elements.
Interpolation	Uses Linear conversion, which performs linear interpolation based on the distance between two discrete values, and Cubic conversion, which uses four discrete values for interpolation.
Bi-linear Interpolation	2D version of linear interpolation. Used in Remap, Filter, and Clut.
Bi-cubi Interpolation	2D version of cubic interpolation. Used in Filter.
Tri-linear Interpolation	3D version of linear interpolation. Used in Clut.
Source	Represents the transfer source, with "Src" as the modifier. There are three types: SrcIn, SrcOut, and SrcMap.
Destination	Represents the transfer destination, with "Dst" as the modifier. There are three types: DstIn, DstOut, and DstMap. DstIn refers to the read path of the destination. DstMap also serves as the configuration for SrcOutMap, which is linked to SrcOut.
Float	Floating-point representation and its operations. Implements functionality excluding IEEE 754 features such as NaN, Inf, and rounding. Uses single precision for Affine and half precision for Filter.

1.5. Others

- The **ItalicBold** font indicates a core.
- The **Thoma** font indicates a signal.
- The **Command.Field** font indicates a Command List name and field name. The field name may be omitted in some cases.

2. Signal Lines

2.1. Control Bus Interface

Signal Name	IO	Pol	Source	Description
cntlReq	I	+	clk	<ul style="list-style-type: none">Request signalEvaluate cntlGnt
cntlGnt	O	+	clk	<ul style="list-style-type: none">Grant signal
cntlRwx	I	+	clk	<ul style="list-style-type: none">R/W signalEvaluate cntlReq & cntlGnt0: Write1: Read
cntlAddr[31:0]	I	+	clk	<ul style="list-style-type: none">Address signalEvaluate cntlReq & cntlGnt
cntlWrAck	O	+	clk	<ul style="list-style-type: none">Write acknowledge signal
cntlWrData[31:0]	I	+	clk	<ul style="list-style-type: none">Write data signalEvaluate cntlWrAck
cntlRdAck	O	+	clk	<ul style="list-style-type: none">Read acknowledge signal
cntlRdData[31:0]	O	+	clk	<ul style="list-style-type: none">Read data signalSync cntlRdAck
cntlIrq	O	+	clk	<ul style="list-style-type: none">Interrupt signalLevel hold type(Fix'0')

2.2. PSS Interface

Signal Name	IO	Pol	Source	Description
iVld	I	+	clk	<ul style="list-style-type: none">Pipeline start valid signal
iStall	O	+	clk	<ul style="list-style-type: none">Pipeline start stall signal
iEnd[3:0]	I	+	clk	<ul style="list-style-type: none">Information of end of indexes
iAddr[31:0]	I	+	clk	<ul style="list-style-type: none">Address to fetch context dataEvaluate iVld & !iStalliAddr[5:4] indicate the parameter fetch timing<ul style="list-style-type: none">➤ 0: At iIndex[63:0] = 0➤ 1: At iIndex[47:0] = 0➤ 2: At iIndex[31:0] = 0➤ 3: At iIndex[15:0] = 0

iDelta[15:0]	I	+	clk	<ul style="list-style-type: none"> • Transfer volume • Evaluate iVld & !iStall
iIndex[64:0]	I	+	clk	<ul style="list-style-type: none"> • Five coordinates to specify the processing • Evaluate iVld & !iStall
oVld	O	+	clk	<ul style="list-style-type: none"> • Pipeline end valid signal
oStall	I	+	clk	<ul style="list-style-type: none"> • Pipeline end stall signal

2.3. Memory Interface (Remapper Read Use)

Signal Name	IO	Pol	Source	Description
mr _n Req	O	+	clk	<ul style="list-style-type: none"> • Request signal
mr _n Gnt	I	+	clk	<ul style="list-style-type: none"> • Grant signal
mr _n Rxw	I	+	clk	<ul style="list-style-type: none"> • R/W signal • Write indicates cache flush
mr _n Bank[1:0]	O	+	clk	<ul style="list-style-type: none"> • Bank signal • Indicates a hint of buffer location for outside cache
mr _n Addr[31:0]	O	+	clk	<ul style="list-style-type: none"> • Address signal • LSB2bit indicates bank hit(usually subscript n)
mr _n RdStrb	O	+	clk	<ul style="list-style-type: none"> • Read strobe
mr _n RdAck	I	+	clk	<ul style="list-style-type: none"> • Read acknowledge signal
mr _n RdData[31:0]	I	+	clk	<ul style="list-style-type: none"> • Read data signal

Signal name subscript n is channel number from 0 to 2

2.4. Memory Interface (Pixel Cache Read Use)

Signal Name	IO	Pol	Source	Description
mc _n Req	O	+	clk	<ul style="list-style-type: none"> • Request signal
mc _n Gnt	I	+	clk	<ul style="list-style-type: none"> • Grant signal
mc _n Rxw	I	+	clk	<ul style="list-style-type: none"> • R/W signal • Write indicates cache flush
mc _n Bank[2:0]	O	+	clk	<ul style="list-style-type: none"> • Bank signal • Indicates a hint of buffer location for outside cache
mc _n Addr[31:0]	O	+	clk	<ul style="list-style-type: none"> • Address signal

mc _n RdStrb	O	+	clk	• Read strobe
mc _n RdAck	I	+	clk	• Read acknowledged signal
mc _n RdData[31:0]	I	+	clk	• Read data signal

Signal name subscript n is channel number from 0 to 12

2.5. Memory Interface (Blender Read Use)

Signal Name	IO	Pol	Source	Description
mbRdReq	O	+	clk	• Request signal
mbRdGnt	I	+	clk	• Grant signal
mbRdNew	O	+	clk	• Transaction start signal
mbRdEnd	O	+	clk	• Transaction end signal
mbRdType	O	+	clk	• Type signal (Fixed '0') • Indicates access direction (0:increment, 1: decrement)
mbRdBE [BSR-1:0]	O	+	clk	• Burst end signal • Indicates terminal lsb address in burst length
mbRdAddr[31:0]	O	+	clk	• Address signal
mbRdStrb	O	+	clk	• Read strobe
mbRdAck	I	+	clk	• Read acknowledged signal
mbRdData[31:0]	I	+	clk	• Read data signal

BSR is given as burst length radix parameter

2.6. Memory Interface (Blender Write Use)

Signal Name	IO	Pol	Source	Description
mbWrReq	O	+	clk	• Request signal
mbWrGnt	I	+	clk	• Grant signal
mbWrNew	O	+	clk	• Transaction start signal
mbWrEnd	O	+	clk	• Transaction end signal
mbWrType	O	+	clk	• Type signal (Fixed '0') • Indicates access direction (0:increment, 1: decrement)
mbWrBE [BSR-1:0]	O	+	clk	• Burst end signal • Indicates terminal lsb address in burst

				length
mbWrAddr[31:0]	O	+	clk	• Address signal
mbWrStrb	O	+	clk	• Write strobe
mbWrAck	I	+	clk	• Write acknowledge signal
mbWrData[31:0]	O	+	clk	• Write data signal
mbWrMask[3:0]	O	+	clk	• Write mask signal
mtReq	I	+	clk	<ul style="list-style-type: none"> • Request terminal signal • Propagated mbWrReq signal to coherency port • If no bridge and using mc2, connect mbWrReq signal directly
mtGnt	I	+	clk	• Grant terminal signal same as mtReq signal

BSR is given as burst length radix parameter

2.7. Memory Interface (Steal Write Use)

Signal Name	IO	Pol	Source	Description
msReq	O	+	clk	• Request signal
msGnt	I	+	clk	• Grant signal
msNew	O	+	clk	• Transaction start signal
msEnd	O	+	clk	• Transaction end signal
msBE [BSR-1:0]	O	+	clk	<ul style="list-style-type: none"> • Burst end signal • Indicates terminal lsb address in burst length
msAddr[31:0]	O	+	clk	• Address signal
msStrb	O	+	clk	• Write strobe
msAck	I	+	clk	• Write acknowledge signal
msData[31:0]	O	+	clk	• Write data signal
msMask[3:0]	O	+	clk	• Write mask signal

BSR is given as burst length radix parameter

2.8. Memory Interface (Histogram Write Use)

Signal Name	IO	Pol	Source	Description
mhReq	O	+	clk	• Request signal

mhGnt	I	+	clk	• Grant signal
mhNew	O	+	clk	• Transaction start signal
mhEnd	O	+	clk	• Transaction end signal
mhBE [BSR-1:0]	O	+	clk	• Burst end signal • Indicates terminal lsb address in burst length
mhAddr[31:0]	O	+	clk	• Address signal
mhStrb	O	+	clk	• Write strobe
mhAck	I	+	clk	• Write acknowledge signal
mhData[31:0]	O	+	clk	• Write data signal
mhMask[3:0]	O	+	clk	• Write mask signal

BSR is given as burst length radix parameter

2.9. Memory Interface (Parameter Read Use)

Signal Name	IO	Pol	Source	Description
meReq	O	+	clk	• Request signal
meGnt	I	+	clk	• Grant signal
meAddr[31:0]	O	+	clk	• Address signal
meStrb	O	+	clk	• Read strobe signal
meAck	I	+	clk	• Read acknowledge signal
meFlush	O	+	clk	• Read flush signal
meData[63:0]	I	+	clk	• Read data signal

2.10. Utility

Signal Name	IO	Pol	Source	Description
rstReq	O	+	clk	• Internal reset signal to reset the external system
rstAck	I	+	clk	• Acknowledge of rstReq
fReq	I	+	clk	• 1 clock early request against the miReq signal • Use to generate gate signal (for mc2)
pReq	O	+	clk	• 1 clock early request against the meReq signal • Use to generate gate signal (for mc2)

gate	O	+	clk	• Gated clock control signal signifying condition of each internal block
Gclk	I	+	clk	• Gated clock
Clk	I	+	clk	• Clock
Reset	I	+	clk	• Synchronous reset signal

3. Configuration and Operation

3.1. System Overview

- The **Pipeline Slice Scheduler** (hereinafter referred to as *pss*) retrieves the necessary context from memory, fragments the information, generates coordinate data, and activates *frComp*. Refer to the separate specification for *pss* for further details.
- The connection interface only requires the input of coordinate and Command List base addresses. Since it uses simple Valid/Stall control, the use of *pss* is not mandatory. If *pss* is not used, it can be replaced with a custom circuit.
- *frComp* follows the pipeline structure shown in **Figure 1**, processing in the order: **Initiator**, **Polygon Generator**, **Remapper**, **Affine Transform**, **Pixel Cache**, **Filter**, **Envelope**, **3D Clut**, **Extractor**, and **Blender**. This processing order cannot be changed.
- Pixel data is first accumulated in a local cache (L1), enabling efficient processing, especially for localized accesses. However, large filter sizes (kernels) may result in high-load memory access. For improved performance, an external unified cache system (L2) that integrates all memory access types is effective.
- The engine generally processes **four elements at the same coordinate simultaneously**. For convenience, elements are referred to as ARGB in the following explanation, though the color space itself is not restricted. Element roles differ only in the 3D Clut (e.g., only RGB elements are referenced in 3D transformations); otherwise, element processing is equivalent. When handling grayscale data only, up to four pixels can be processed simultaneously. Additionally, binary Bitmap data can be handled for specific operations.

- As shown in **Figure 1**, processing is categorized into three domains:
Coordinate, **Pixel**, and **Information**.
-

• **Coordinate Processing:**

Performs transformations on reference coordinates, known as *parametric coordinates* (a 4D coordinate system incremented from 0 up to a configured range). These coordinates and transformation units are input externally. Processing generates new (X, Y) coordinates, supplying them to both the Source and Destination systems. The operations include:

- Generation of a polygon shape from a single transformation unit
 - Recombination of 4D coordinates (X, Y, Z, W) into new coordinates
 - Replacement with new coordinates fetched from memory
 - Matrix transformation (applied to Source side only)
-

• **Pixel Processing:**

Refers to pixel acquisition from memory using coordinates, pixel processing, and writing results back to memory. Each of the four elements is processed independently. For input formats with fewer than 4 elements (e.g., 8bpp), a single element is duplicated to form four. The processing includes:

- **SrcIn** supports up to 7 filter types; **SrcOut** supports 1 optional filter (only SrcIn filters generate flags)
 - Arbitrary elements from SrcIn and SrcOut can be combined with add or multiply operations (separated into *Origin* and *Modified*)
 - Independent color space conversion for *Origin* and *Modified*
 - After selecting either *Origin* or *Modified*, **Extractor** performs pixel value judgment, and **Blender** executes blending; Extractor may also generate flags
 - If filter and Extractor flags are active, the corresponding coordinates are written to memory
 - Histogram data is written to memory
-

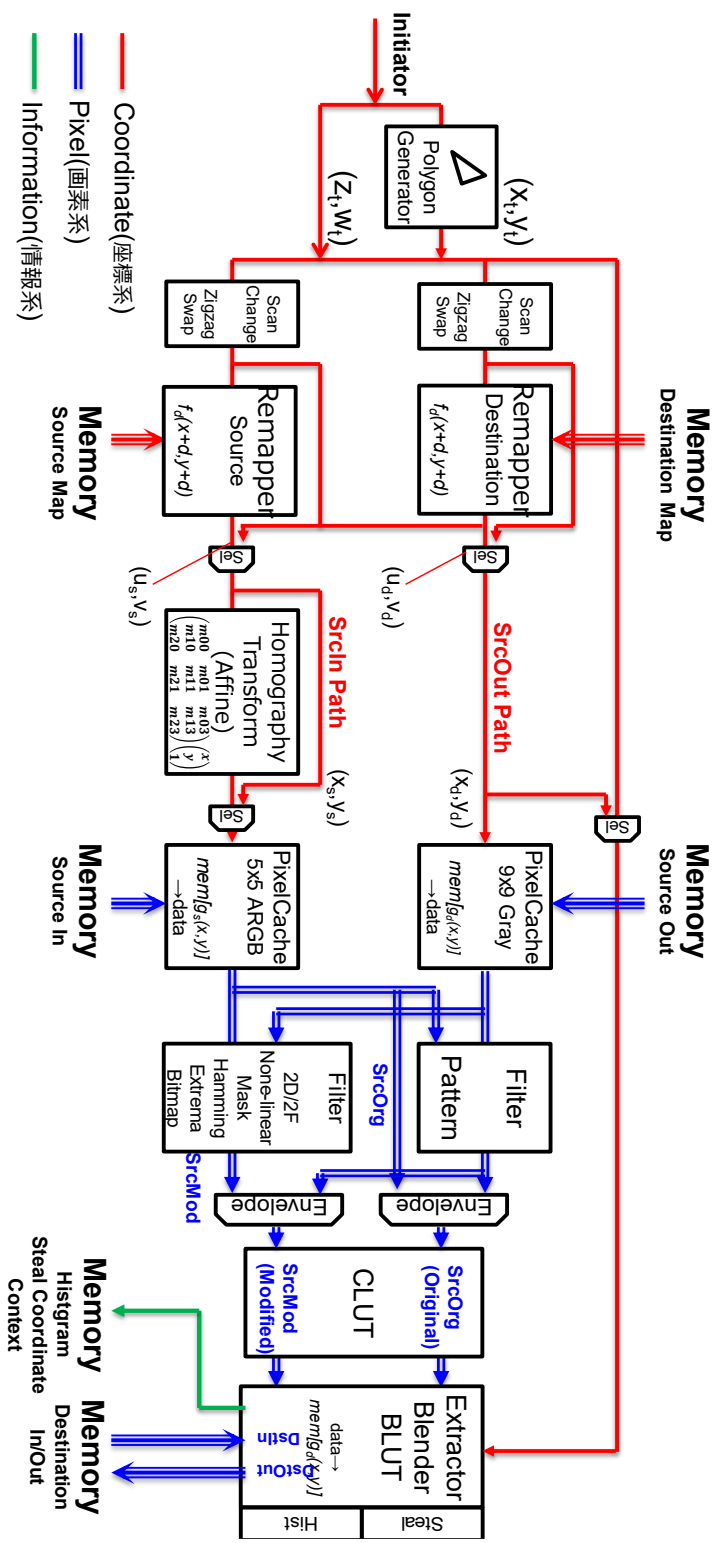
• **Information Processing (Statistics):**

Outputs coordinates of pixels meeting specific filter conditions and the results of histogram aggregation to memory.

• **Context Data I/O:**

Some context data is exchanged, used for parameter sharing between instances of the same engine or between different engines:

- Polygon shape length (used in 1D/2D context reference modes)
- Constants for the Envelope stage
- Constants for the Blender stage



3.2. Input and Output Data

- Multiple input and output data types are handled. Each can be individually enabled or disabled. Data of the same type can generally be reused.

Classification	I/O	Description
Context (In)	Input	<ul style="list-style-type: none">● At the beginning of the fragmentation process, context data is retrieved and later restored.● The size is 32 bytes.
Context (Out)	Output	<ul style="list-style-type: none">● At the end of the fragmentation process, context data is saved and backed up.● The size is 32 bytes.
Map (Source)	Input	<ul style="list-style-type: none">● Mapping data used for the input frame● Referenced based on the corresponding parametric coordinates● The size is proportional to the frame dimensions (reduction ratio is specified in the Command List)
Map (Destination)	Input	<ul style="list-style-type: none">● Mapping data used for the output frame● Referenced based on the corresponding parametric coordinates● The size is proportional to the frame dimensions (reduction ratio is specified in the Command List)
Map (Steal)	Output	<ul style="list-style-type: none">● Mapping data that outputs the corresponding coordinates based on per-pixel evaluation● Written based on the address specified in the Command List● Size is variable● Supports 1D access only

Frame Buffer (Source In)	Input	<ul style="list-style-type: none"> ● Input frame (Primary) ● Referenced based on either the parametric coordinates or coordinates transformed via Map (Source) ● Size is specified in the Command List ● Normally accessed in 2D, but serialized 1D access is also supported
Frame Buffer (Source Out)	Input	<ul style="list-style-type: none"> ● Input frame (Secondary) ● Referenced based on either the parametric coordinates or coordinates transformed via Map (Destination) ● Size is specified in the Command List ● Normally accessed in 2D, but serialized 1D access is also supported
Frame Buffer (Destination In)	Input	<ul style="list-style-type: none"> ● Target frame input (for modification) ● Referenced based on either the parametric coordinates or coordinates transformed via Map (Destination) ● Size is specified in the Command List ● Normally accessed in 2D, but serialized 1D access is also supported
Frame Buffer (Destination Out)	Output	<ul style="list-style-type: none"> ● Target frame output ● Written based on either the parametric coordinates or coordinates transformed via Map (Destination) ● Size is specified in the Command List ● Normally accessed in 2D, but serialized 1D access is also supported
Color Lookup Table	Input	<ul style="list-style-type: none"> ● Color conversion table ● Referenced based on the address specified in the Command List ● Size is 16 KBytes ● Supports selection of 1D, 2D, 3D, or Binary formats

Blend Lookup Table	Input	<ul style="list-style-type: none"> Common color conversion table for all color components Referenced based on the address specified in the Command List Size is 256 bytes Used for per-element pixel conversion and as auxiliary data for filter parameters
Histogram	Output	<ul style="list-style-type: none"> Histogram data The frame can be divided into blocks, allowing multiple outputs Maximum size per unit is $256 \times 4 \times 4$ bytes (4 KBytes)

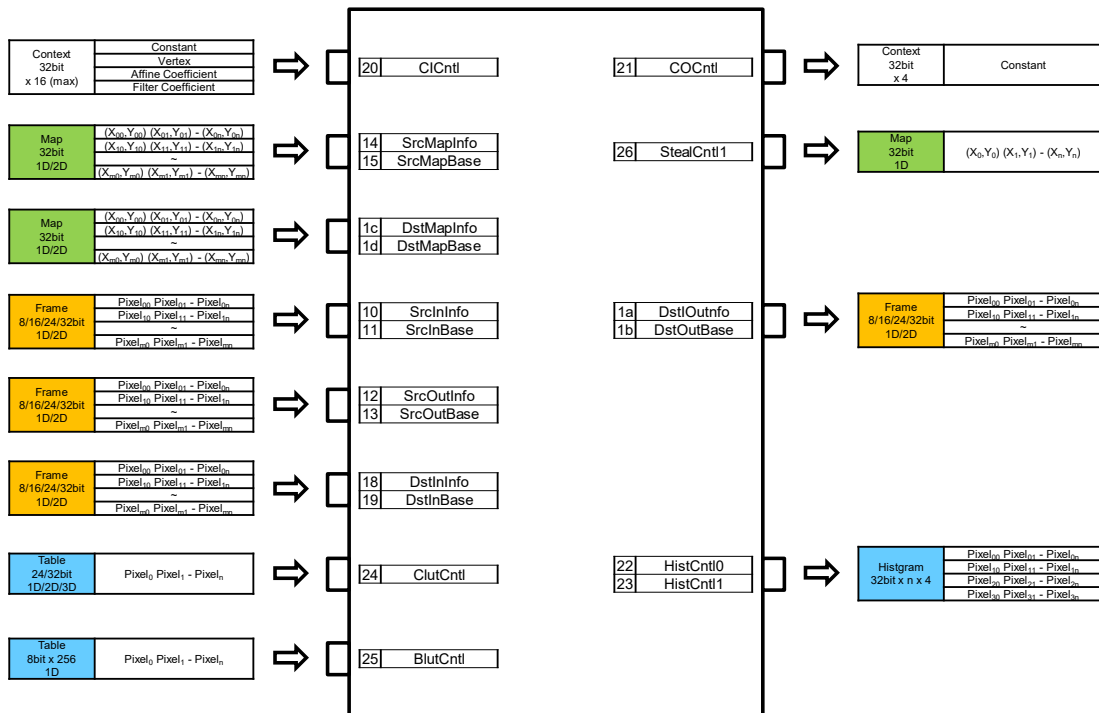


Figure 1 Inputs and Outputs

- The Source In (SrcIn) and Source Out (SrcOut) systems in the frame buffer are used to process input images.

In *frComp*, SrcIn serves as the main input path, while SrcOut provides complementary processing capabilities.

Extended SrcIn functions and special operations that cannot be handled by SrcIn

alone can also be executed.

Some of the functions supported by SrcOut include:

- **Extended kernels for 2D filters:**

Uses four ARGB elements to process 7×7 and 9×9 kernels. Applicable to grayscale images only.

- **Pattern filter:**

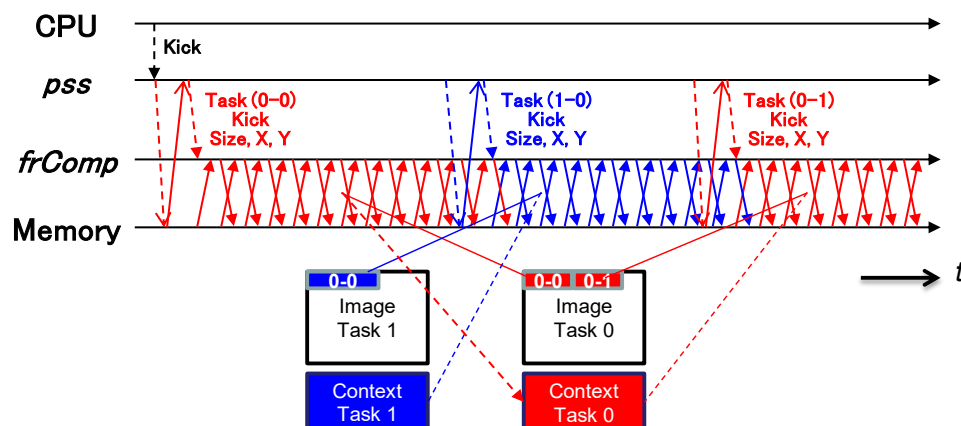
Evaluates the relationship between the center pixel and surrounding pixels using kernels up to 9×9 in size.

- **Hamming filter:**

Calculates Hamming distance between two binary images. Input images are provided through both SrcIn and SrcOut.

3.3. Drive Interface (Initiator)

- The **pss** scans the parametric coordinates—used as reference indices for positioning—along the X-axis and sends them to the **Initiator** of *frComp*. Settings for the *pss* (such as image information and processing units) must be preloaded into memory.
pss manages multiple configurations—**N instances** (depending on implementation)—using time-division control, and drives *frComp* according to the scheduling.



- The **Initiator** reads the Command List from memory based on the image information provided by *pss* and sets up the pipeline accordingly.
The parameters extracted from the Command List are managed using **triple-buffer control**, so performance degradation does not occur unless the specified processing units are extremely short. Even in such cases, if the same context continues across units, **chained processing** is applied, maintaining performance.
- The context (handover information) required for fragmentation is exchanged via memory.
Context is applicable only to a limited set of functions and holds **8 words (32 bytes)** of information.

-
- If *pss* is not used, the basic method to drive *frComp* is as follows.
It operates on a **line-by-line basis**.
While this appears similar to rectangular rendering in polygon generation (described later), the key difference is that **different tasks can be inserted between lines**.
 - Use a counter to increment Y-direction values from 0 to (height – 1).
Assert the **iVld** signal when active; if **iStall** is ‘0’, increment the counter.
 - Assert the starting address of the 256-byte Command List on the **iAddr** signal.
 - Assert (width – 1) in the X-direction on the **iDelta** signal.
 - Assert the counter value to **iIndex[31:16]**; all other bits in **iIndex** are set to ‘0’.

-
- **iAddr[5:4]** specifies the timing for parameter reset.
The actual parameter address is determined by **iAddr[31:6]**, aligned to 64-byte units.
Forcing a parameter reset—even when parameters remain unchanged—may cause performance degradation due to mandatory reloading and setup.

iAddr[5:4]	Description
0	Parameters are loaded when $X = Y = Z = W = 0$.

1	Parameters are loaded when $X = Y = Z = 0$.
2	Parameters are loaded when $X = Y = 0$.
3	Parameters are loaded when $X = 0$.

3.4. Notes on Fragmentation

- In the fragmentation of processing, alternating between different parameter sets generally does **not** cause inconsistencies.
However, additional configuration may be required if intermediate results (**context**) need to be carried over.
- Context is passed via memory. In principle, it is sufficient to specify the memory address.

Groups that require context can be designated arbitrarily.

The content of the context starting from the specified address is as follows:

Address	Description
+0	Constant Group (Input/Output) Used for processing with a specified length, such as a 32-bit one-dimensional length or a pair of 16-bit two-dimensional lengths.
+4	Constant Group (Input/Output) The input is optional and may be referenced by Envelope or Blender . The output represents the accumulated value of a specific pixel element .
+8	Constant Group (Input/Output) The input is optional and may be referenced by Envelope or Blender . The output represents the minimum value of a specific pixel element .
+c	Constant Group (Input/Output) The input is optional and may be referenced by Envelope or Blender . The output represents the maximum value of a specific pixel element .

+10 ~ +18	Vertex Group (Input Only) Replaces Vertex0 to Vertex2 .
+1c ~ +3c	Affine Group (Input Only) Replaces AffineCoef0 to AffineCoef8 .
+40 ~ +44	Reserved
+48 ~ +7c	Filter Group (Input Only) Replaces FilterCoef0 to FilterCoef13 .

- **Context is not read or written unless explicitly configured in the Command List.** For operations that utilize context—such as coordinate generation with context reference (1D/2D operations), **Envelope**, and **Blender constants**—you must ensure context is properly read.
- The following outlines features that require special attention during fragmentation, along with corresponding handling methods:

Function	Description
3D Clut	<ul style="list-style-type: none"> ▪The referenced table is reloaded for each Command List (however, reloading is skipped if the Command Lists are consecutive and the number of table references is two or fewer). ▪Do not make the fragmentation length excessively short (e.g., less than 1K words), to ensure that table reload time is less than the fragment processing time, avoiding performance degradation. ▪Alternatively, limit table references across the entire system to two sets or fewer.
Envelope /Blender	Some constants reference context data ▪Enable the context feature as needed.
Steal	Performs serial addressing ▪Enable the context feature as needed.

Histogram	Statistical data is written to memory <ul style="list-style-type: none"> • Enable the histogram feature • Avoid excessively short fragmentation lengths (e.g., less than 1K words) to prevent performance degradation
-----------	---

3.5. Coordinate Generation (Polygon Shapes and Scanning)

- When a shape other than **Normal** is specified, the engine uses **vertices defined in the Command List** instead of parametric coordinates, and processes the shape formed by connecting those vertices. This type of processing is executed in a **single launch without fragmentation**.
- When using **pss**, the process operates per launch unit. By setting the fragmentation unit to 1, processing is triggered each time the lowest index parametric coordinate **X increments by 1**. This is particularly useful for operations like SAD/SSD search sequences.
- There are four supported shape types, including **Normal**. The fragmentation unit is defined by the shape, and **no other processing can be inserted into frComp until the current shape's processing is complete**. In some cases, **iDelta** is used as a **Size** parameter.

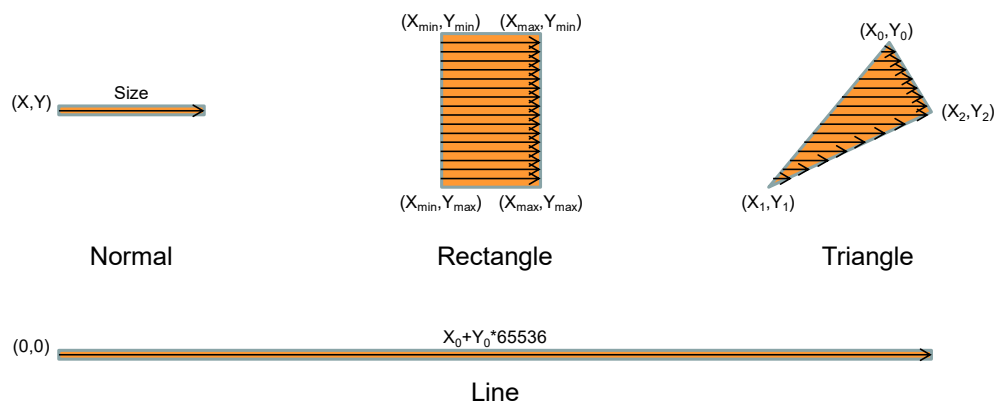


Figure 2 Polygon Drawing

For the **Normal** shape, only the **iIndex signal** (representing parametric coordinates) and the **iDelta signal** (representing size) are used.

For all other shapes, only the **coordinates specified in the Command List** are used.
The **order in which coordinates are specified does not matter**.
Regardless of shape, processing is always scanned **from smaller to larger values in the Y direction**, and **from left to right in the X direction**.

Shape	Description
Normal	Specifies the parametric coordinates and size using the iIndex and iDelta signals.
Line	Scanning starts from (0,0) for a 32-bit length specified in the first vertex. Assign the upper 16 bits to Y and the lower 16 bits to X.
Rectangle	Scans a rectangle defined by the minimum XY and maximum XY values of two vertices, using them as diagonal corners. Scanning proceeds from the smaller Y-axis value.
Triangle (Ver.C)	Scans a triangle formed by three vertices (order is arbitrary), starting from the smallest Y-axis value. Diagonal lines are rendered by combining two triangles.
Context Reference Normal	Same as Normal , however, the length is not taken from the Command List parameters , but instead from word 0 (32-bit) of the context . <i>(Word 0 of the context must be preloaded into memory or written using the Steal function.)</i> Y is determined by iIndex[31:16] , which differs from the Line shape.

Context Reference Line	<p>Same as Line, however, the length is not taken from the Command List parameters, but instead from word 0 (32-bit) of the context. <i>(Word 0 of the context must be preloaded into memory or written using the Steal function.)</i></p>
Context Reference Rectangle	<p>Same as Rectangle, however, the width and height are not taken from the Command List parameters, but instead from word 0 of the context. The lower 16 bits specify the X size, and the upper 16 bits specify the Y size. <i>(Word 0 of the context must be preloaded into memory or written using the Steal function.)</i></p>
Context Reference Triangle (Ver.C)	<p>Same as Triangle, however, references word 0 of the context. Processing is executed only if word0[15:0] is non-zero. If word0[17] is 1: Draw only clockwise triangles when word0[16] is 0 Draw only counterclockwise triangles when word0[16] is 1.</p>

- **Scanning within a shape follows the fill rules below** to prevent edge overdraw and gaps. These rules can be disabled if needed.
However, they do **not** apply to **Normal**, **Line**, or **Rectangle** shapes (in Ver.C).
 - Only draw when the center of a grid point lies **inside the shape**
 - If the **left edge** lies on the ideal line, draw it; if the **right edge** lies on the ideal line, do **not** draw it
 - If the **left and right edges** are the same, do **not** draw
 - Do **not** draw the final scan line

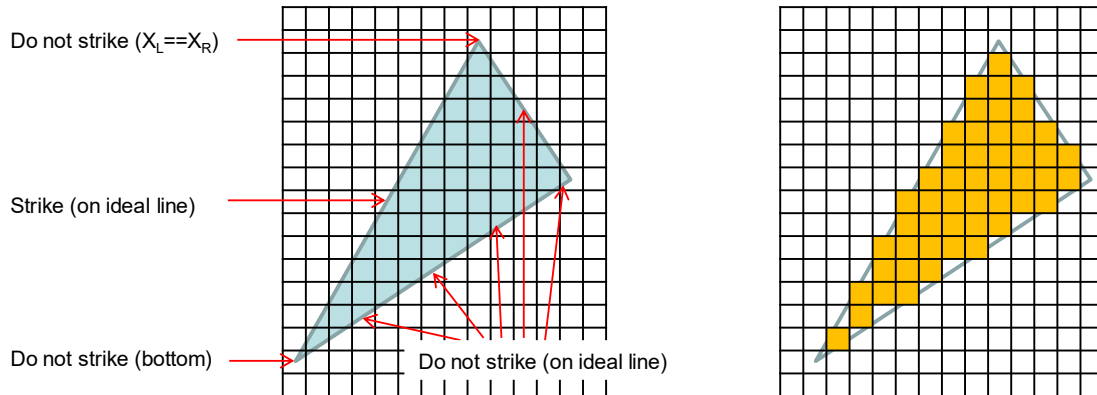


Figure 3 Polygon Fill Rule

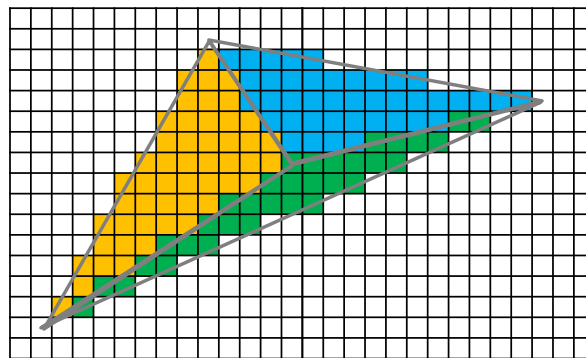


Figure 4 Three Triangles Drawing

- For the **Normal** and **Line** shapes, a mode is available that places points only at **(0, y)**.
This can be used for operations such as scanning the X-axis internally and performing accumulation processing before writing results.
- **Texture coordinates are not directly specified for given positions.**
Instead, matrix transformations are used to derive texture coordinates from parametric coordinates.
For details, refer to the application note *"Polygon Rendering."*
- The **Line shape** is intended to support coordinates that exceed 16 bits.
This is particularly useful for the **Remapper** (described later), where 1D arrays may exceed 16-bit (65536) lengths.
Since values **65537 and above cannot be represented using 2D coordinates**, this mode should be used.

Additionally, the **coordinate count** extracted using the *Steal* function (described later) can be used via the context.

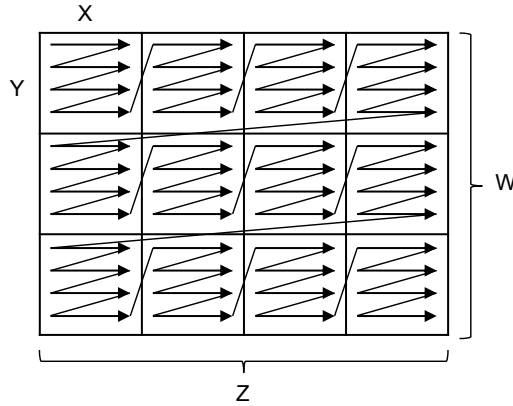
- Final parametric coordinates are generated by combining **XY coordinates obtained by scanning the specified shape (Polygon function)** and **ZW coordinates supplied via the *iIndex* signal (Zigzag function)**.

This coordinate recombination can be independently configured for both **Source** and **Destination** coordinate systems.

(Refer to the Command List for configuration details.)

$$\begin{aligned}(x', y') &= \text{Polygon}(x, y) \\ (x_{srcIn}, y_{srcIn}) &= \text{Zigzag}(x', y', z, w, \text{mask}, \text{box}) \\ (x_{srcOut}, y_{srcOut}) &= \text{Zigzag}(x', y', z, w, \text{mask}, \text{box}) \\ (x_{dst}, y_{dst}) &= \text{Zigzag}(x', y', z, w, 0, 0)\end{aligned}$$

Figure 5 Zigzag Scan



Source and Destination coordinates are independent and can be processed differently.

For example, the **Source coordinates** can be set in polar coordinates, while the **Destination coordinates** can be individually configured with distortion.

The highest performance is achieved when the **Destination coordinates are directly mapped from parametric coordinates**.

However, note that **matrix transformations cannot be applied to the Destination coordinate system**.

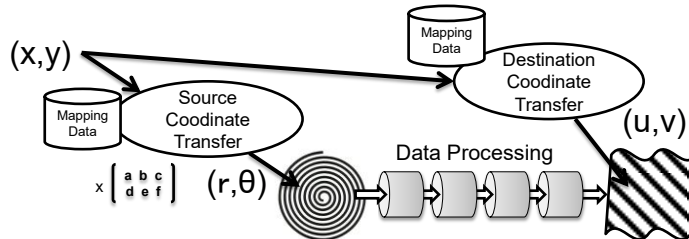


Figure 6 Coordinate and Data Processing

3.6. Remapping (Remapper)

- The **Remapper**, as shown in *Figure 9*, performs memory access based on the input **parametric coordinates (X, Y)** plus an offset, and reads new coordinates from a table to perform coordinate remapping.
Since preparing mapping data for each pixel would be excessive, it is possible to **sample at every 2ⁿ samples** and then use **bi-linear interpolation** to reconstruct the final coordinates.
- The Remapper can be configured to either:
 - **Directly output the mapped coordinates**, or
 - **Add them to the parametric coordinates** before output.
This allows specifying **relative displacement** from the parametric coordinates.
- If an **escape code (0x8000)** is read:
 - In the **Source Remapper**, the previously read value is reused.
 - In the **Destination Remapper**, a **mask flag** is set, which suppresses memory access for that coordinate (i.e., no rendering).
Note: This has **no effect during texture transformation**.
Optionally, the escape code behavior can be **disabled via configuration**.

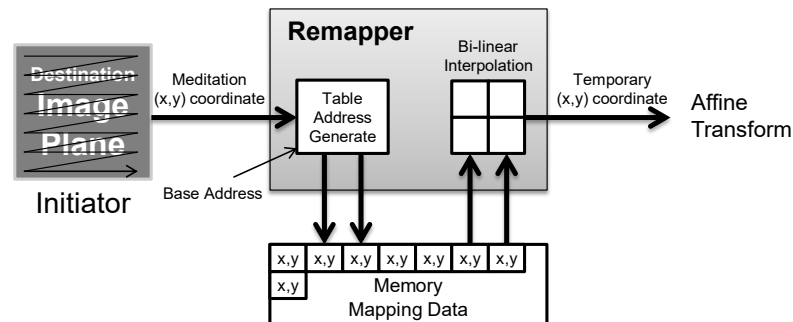


Figure 7 Remapper

- **Texture transformation** is supported as a function mutually exclusive with coordinate mapping.
In texture transformation, the **lower bits of the parametric coordinates XY** are enabled and used to set a new **X coordinate**.
Then, based on the parametric coordinates, a **luminance value** is read from memory and used as the new **Y coordinate**.

By performing another memory access using this new XY coordinate, an image with **scattered luminance** can be obtained.

For more details, refer to the application note *"Abstraction."*

- The **matrix transformation (Affine/Homography Transform)** described later requires **single-precision floating-point input**.

Therefore, in this **Remapper** section, you can select conversion from:

- **Integer to single-precision float**
- **Half-precision float to single-precision float**
- **No conversion**

In these cases, you must specify **32-bit × 2 words** instead of packed **16-bit × 2 words**.

Note that when a **floating-point format** is selected, **bi-linear interpolation cannot be used**.

MapInfoFormat	Description
Nearest	Coordinate data packed as 16-bit words in memory (Xo, Yo) = mem(Xi, Yi)
Bi-linear	
2x1	Coordinate data packed as 32-bit words in memory Xo = mem(2 Xi, Yi) Yo = mem(2 Xi + 1, Yi)
1x2	Coordinate data striped in 32-bit words in memory Xo = mem(Xi, 2 Yi) Yo = mem(Xi, 2 Yi + 1)

- (Xi, Yi) are the input coordinates, and (Xo, Yo) are the output coordinates.
- **mem()** refers to a memory array consisting of **32-bit words**.

When mapping data is **sampled at every 2ⁿ intervals**, the area reduces to **¼ with each increase in N**, allowing significant **compression of the mapping data**.

This also contributes to a **reduction in memory access**.

However, this method is only applicable when the mapping data exhibits **relatively linear variation**, rather than discrete jumps.

Additionally, **floating-point format data cannot be used** for 2ⁿ-sampled mapping data.

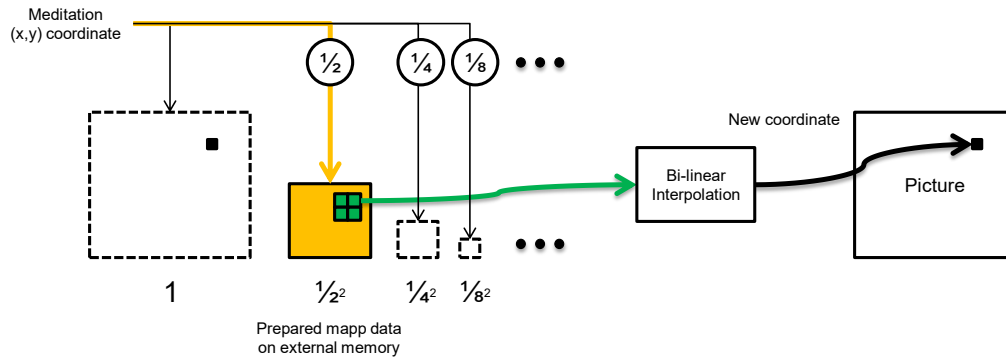


Figure 8 Compressed Map Data

3.7. Matrix Transformation (Affine/Homography Transform)

Homography Mode 1:

$$\text{Remapper} \xrightarrow{(x_s, y_s)} \begin{bmatrix} m00 & m01 & m02 \\ m10 & m11 & m12 \\ m20 & m21 & m22 \end{bmatrix} \begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix} \xrightarrow{(u,v)} \begin{bmatrix} \text{Source} \\ \text{Integer} \\ \text{Plane} \end{bmatrix} \xrightarrow{\text{Pixel Cache (Memory Access)}}$$

Homography Mode 2:

$$\text{Remapper} \xrightarrow{\begin{pmatrix} x_d, y_d \\ x_s, y_s \end{pmatrix}} \begin{bmatrix} m00 & m01 & x_d \\ m10 & m11 & y_d \\ m20 & m21 & m22 \end{bmatrix} \begin{bmatrix} x_s + m02 \\ y_s + m12 \\ 1 \end{bmatrix} \xrightarrow{(u,v)} \begin{bmatrix} \text{Source} \\ \text{Integer} \\ \text{Plane} \end{bmatrix}$$

Rotate Mode 1:

$$\text{Remapper} \xrightarrow{(x_s, y_s)} \begin{bmatrix} \cos y_s & -\sin y_s & m02 \\ \sin y_s & \cos y_s & m12 \end{bmatrix} \begin{bmatrix} m20 \cdot x_s \\ m21 \cdot x_s \\ m22 \end{bmatrix} \xrightarrow{(u,v)} \begin{bmatrix} \text{Source} \\ \text{Integer} \\ \text{Plane} \end{bmatrix}$$

Rotate Mode 2:

$$\text{Remapper} \xrightarrow{\begin{pmatrix} x_d, y_d \\ x_s, y_s \end{pmatrix}} \begin{bmatrix} \cos(y_s + m12) & -\sin(y_s + m12) & x_d \\ \sin(y_s + m12) & \cos(y_s + m12) & y_d \end{bmatrix} \begin{bmatrix} m20(x_s + m02) \\ m21(x_s + m02) \\ m22 \end{bmatrix} \xrightarrow{(u,v)} \begin{bmatrix} \text{Source} \\ \text{Integer} \\ \text{Plane} \end{bmatrix}$$

- The matrix transformation converts coordinates generated by the **Remapper** (in floating-point format) into **Source coordinates** (fixed-point format). Based on the Source coordinates, **Destination coordinates** may optionally be used.

As shown in *Figure 11*, there are **four basic transformation modes**, with all coefficients in **floating-point format**.

Figure 9 Affine/Homography Transform

- In **Homography mode**, image operations such as **translation, scaling, rotation, and distortion** based on raster scanning can be performed.

Since this mode includes a **division function**, **perspective correction** is also supported.

- In **Rotate mode**, **angle and magnitude** are specified for each pixel to perform rotational coordinate operations.

This mode requires **floating-point coordinate input** for both angle and magnitude.

Therefore, coordinates must be **converted to floating-point format using the Remapper**—integer coordinates cannot be used directly.

- Operations are performed in the order: **Remapping → Matrix Transformation**.

Conversely, this means that with respect to Source coordinates, the operations are applied in the order: **Matrix Transformation → Remapping**.

For example, when applying a polar coordinate transformation centered on a shifted point in the image, you would first perform **translation using matrix transformation**, then apply the coordinate transformation.

For further details, refer to the application note *“Coordinate Operations.”*

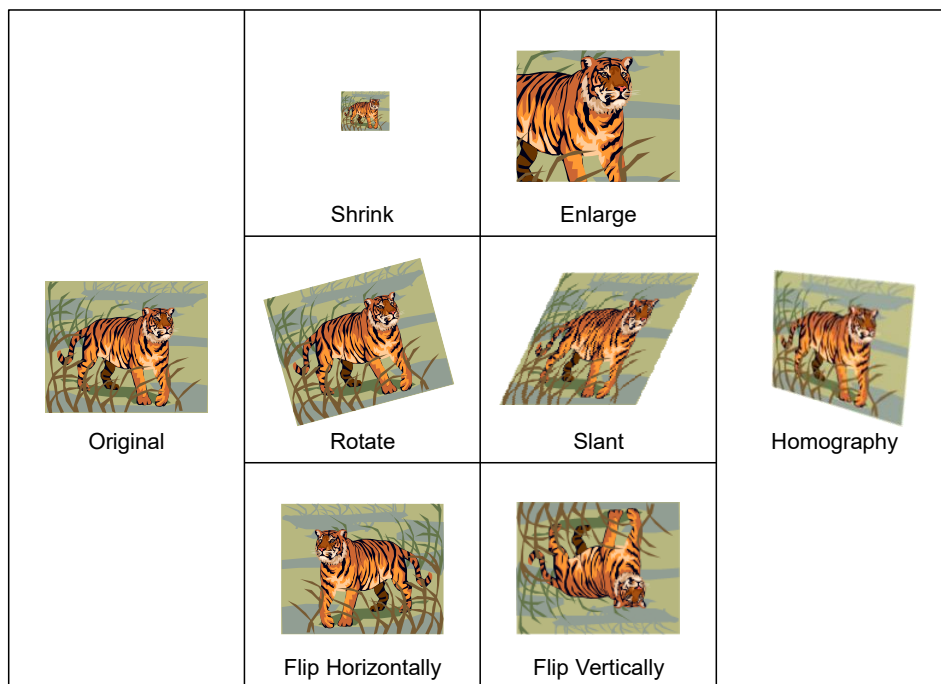


Figure 10 Example of Homography Transformation

3.8. Pixel Cache

- The pixel cache retrieves data within a certain range centered around a specified coordinate (O) from memory and loads it into cache memory.
- The range of the cache is automatically determined based on the specified filter mode.
- When the source coordinate is updated, only the differential data is refilled from memory.

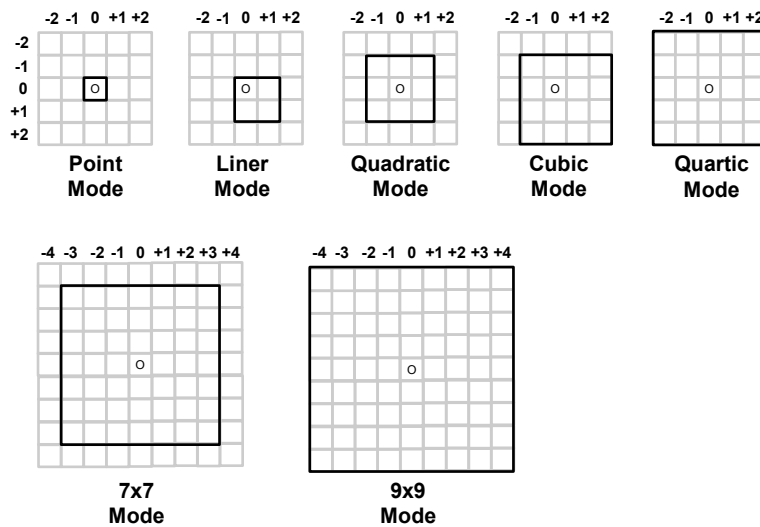


Figure 11 Kernel Size

- The pixel cache consists of two sets: the Source In type and the Source Out type. The Source In type supports ARGB with a 5×5 kernel, while the Source Out type supports only grayscale with a 9×9 kernel. However, for kernels up to 3×3 , ARGB is also supported.
- Pixels outside the image area are replaced with an arbitrary default value. Additionally, several edge options can be configured for the cache output. It is disabled when the image width is set to 0.

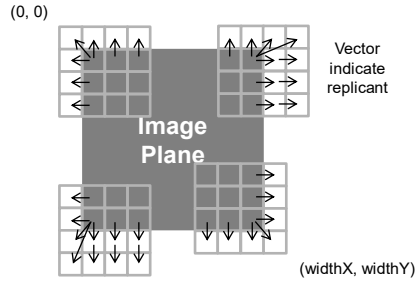
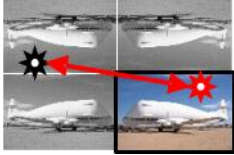
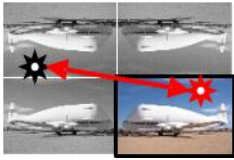


Figure 12 Replicate Edge Pixels

Option	Description
0 CV: Exclusive	If the center coordinate is out of bounds, replace the entire pixel cache with the default value.
1 CV: -	Do not perform out-of-bound checking. Equivalent to setting Width = 0.
8 CV: Constant	Replace out-of-bound pixels with a default value.
9 CV: Replica (Ver.AB は不可)	Replace out-of-bound pixels with the value of the nearest valid pixel.
10 CV: Warp (Ver.AB は不可)	Replace out-of-bound pixels with the corresponding values from a wrapped-around image.
11 CV: Reflect_101 (Ver.AB は不可)	<p>Mirror Read mirror position Write mirror position</p>  <p>Replace out-of-bound pixels with the corresponding values from a mirrored image (pixels beyond the edge are reflected at the edge point).</p>

<p>15</p> <p>CV:Reflect (Ver.AB は不可)</p>	<p>Mirror Read mirror position Write mirror position</p>  <p>Replace out-of-bound pixels with the corresponding values from a mirrored image (pixels beyond the edge are treated as the same edge value).</p>
--	---

- As the kernel size increases, the load on memory also increases proportionally. The Source In type performs up to 5 simultaneous memory accesses, while the Source Out type performs up to 9. The external memory system should be parallelized as much as possible to support simultaneous access.
- Kernel size settings are separated for the Source In and Source Out types. Both SrcIn and SrcOut types support special addressing schemes, as shown in Figure 15, to enable processing along the time axis rather than the spatial axis (Ver.AB). Memory is accessed by cumulatively adding the offset ($2 \times \text{FilterCntl.Stride}$) from the kernel center coordinate (O).

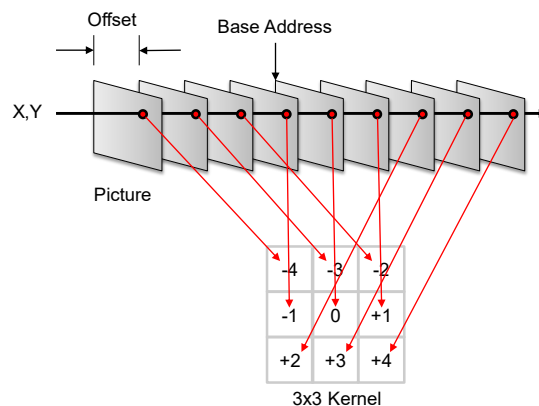


Figure 13 Kernel Filling in Time Domain

In the SrcIn type, specifying a 5×5 kernel size with a 1bpp format allows handling of a 32×25 bitmap. However, this requires the use of a subsequent bitmap filter. The bitmap filter operates on a 32×25 bitmap, utilizing the central 25×25 pixels. The default value is set to 0x00000000 for “0” and 0xffffffff for “1”. Bitmap alignment can be either left-aligned or right-aligned (only left alignment is supported in Ver.A).

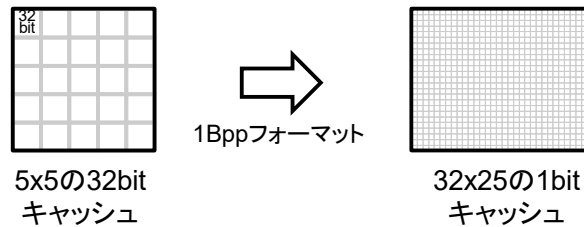


Figure 14 Bitmap Kernel

In both the SrcIn and SrcOut types, memory access for horizontal kernel lines can be masked. When there are unused horizontal lines, they can be deliberately disabled to reduce memory access load. Coefficient adjustment corresponding to the unused lines (e.g., setting coefficients to 0) is required.

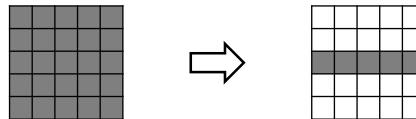


Figure 15 Kernel Load Reduction

3.9. Filter Data and Coefficient Selection

- Data and coefficients for the filter are selected and input from the Command List data, pixel cache output, or tables (see Figure 19). The basic configuration uses 4 elements from a 5×5 kernel, and the input selection options are as follows.
- Only the Pattern Filter directly uses a 9×9 kernel. Details are described below.

Input	Description
Command List	Parameters within the Command List: Coef000–003, Coef100–107, Coef200–215
SrcIn	5 × 5 pixel cache output (Source In type) Supports the 4 ARGB components.
SrcOut	9 × 9 pixel cache output (Source Out type) Supports only a single Gray component.
SrcOut'	Rearranged 9 × 9 (Gray) or 1 × 1 (Color) pixel cache output (Source Out type) into 5 × 5 blocks Placement varies for each element.
Blut	Lookup Table used in Blender.

- The SrcOut type supports grayscale and allows up to a 9 × 9 kernel, but in the 2D Filter, only up to 5 × 5 elements are supported per filter pass. Therefore, a 9 × 9 kernel is split into 4 elements, and a 7 × 7 kernel is split into 2 elements for processing.

- “SrcOut' ” is generated by rearranging the pixel cache output of the SrcOut type into a 5 × 5 kernel. Placement Type 0 corresponds to 5 × 5; Types 0 and 1 together cover 7 × 7; and Types 0 through 3 cover 9 × 9. These placement types are exclusive, with no overlapping regions.

By selecting the appropriate data and coefficients, arbitrary filtering from 5 × 5 to 9 × 9 can be performed. Figure 18 shows the relative positions when a 9 × 9 kernel is rearranged into 5 × 5 blocks. The numbers indicate the order from the top-left to bottom-right of the original 9 × 9 kernel.

FilterCntlOp.InSel[5:4]	Description
0	All ARGB elements use 9 × 9 (Gray) Placement Type 0.
1	R and B elements use 9 × 9 (Gray) Placement Type 0 A and G elements use 9 × 9 (Gray) Placement Type 1

2	B element uses 9×9 (Gray) Placement Type 0 G element uses 9×9 (Gray) Placement Type 1 R element uses 9×9 (Gray) Placement Type 2 A element uses 9×9 (Gray) Placement Type 3
3	B element uses the B component of the 1×1 (Color) G element uses the G component of the 1×1 (Color) R element uses the R component of the 1×1 (Color) A element uses the A component of the 1×1 (Color)

0	1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16	17
18	19	20	21	22	23	24	25	26
27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44
45	46	47	48	49	50	51	52	53
54	55	56	57	58	59	60	61	62
63	64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79	80

Type	0	1	2	3																																																																																																				
Relocation	<table> <tr><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td></tr> <tr><td>29</td><td>30</td><td>31</td><td>32</td><td>33</td></tr> <tr><td>38</td><td>39</td><td>40</td><td>41</td><td>42</td></tr> <tr><td>47</td><td>48</td><td>49</td><td>50</td><td>51</td></tr> <tr><td>56</td><td>57</td><td>58</td><td>59</td><td>60</td></tr> </table>	20	21	22	23	24	29	30	31	32	33	38	39	40	41	42	47	48	49	50	51	56	57	58	59	60	<table> <tr><td>34</td><td>43</td><td>52</td><td>61</td><td>70</td></tr> <tr><td>19</td><td>10</td><td>11</td><td>12</td><td>69</td></tr> <tr><td>28</td><td>25</td><td></td><td>13</td><td>68</td></tr> <tr><td>37</td><td>16</td><td>15</td><td>14</td><td>67</td></tr> <tr><td>46</td><td>55</td><td>64</td><td>65</td><td>66</td></tr> </table>	34	43	52	61	70	19	10	11	12	69	28	25		13	68	37	16	15	14	67	46	55	64	65	66	<table> <tr><td>8</td><td>17</td><td>26</td><td>35</td><td>44</td></tr> <tr><td>73</td><td>0</td><td>1</td><td>2</td><td>53</td></tr> <tr><td>74</td><td>7</td><td></td><td>3</td><td>62</td></tr> <tr><td>75</td><td>6</td><td>5</td><td>4</td><td>71</td></tr> <tr><td>76</td><td>77</td><td>78</td><td>79</td><td>80</td></tr> </table>	8	17	26	35	44	73	0	1	2	53	74	7		3	62	75	6	5	4	71	76	77	78	79	80	<table> <tr><td></td><td></td><td></td><td></td><td></td></tr> <tr><td></td><td>72</td><td>63</td><td>54</td><td></td></tr> <tr><td></td><td>9</td><td></td><td>45</td><td></td></tr> <tr><td></td><td>18</td><td>27</td><td>36</td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td></td></tr> </table>							72	63	54			9		45			18	27	36						
20	21	22	23	24																																																																																																				
29	30	31	32	33																																																																																																				
38	39	40	41	42																																																																																																				
47	48	49	50	51																																																																																																				
56	57	58	59	60																																																																																																				
34	43	52	61	70																																																																																																				
19	10	11	12	69																																																																																																				
28	25		13	68																																																																																																				
37	16	15	14	67																																																																																																				
46	55	64	65	66																																																																																																				
8	17	26	35	44																																																																																																				
73	0	1	2	53																																																																																																				
74	7		3	62																																																																																																				
75	6	5	4	71																																																																																																				
76	77	78	79	80																																																																																																				
	72	63	54																																																																																																					
	9		45																																																																																																					
	18	27	36																																																																																																					

Blank indicate zero value

Figure 16 Relocation of 9x9 position

- Data selection can be made from SrcIn, SrcOut, or a constant value of 1.0, and is selectable per element.

- For SrcOut, each element is assigned either 9×9 Gray or 1×1 Color based on FilterCntlOp.InSel[5:4].
- Ultimately, 8-bit data for four elements with a 5×5 kernel size is output to the filter.

FilterCntlIn.Force[2n+1:2n]	Description
0	SrcIn
1	SrcOut (Gray)
2	The central 3×3 region is from SrcIn, while the surrounding area is from SrcOut (Ver.C).
3	All pixels are set to 1.0.

- Coefficients can be selected from Command List data, the outputs of the two pixel caches, or user-defined tables.
- Ultimately, 16-bit data for four elements with a 5×5 kernel size is output to the filter.

FilterCntlOp.InSel[1:0]	Description
0	Command List
1	Blut
2	SrcIn
3	SrcOut'

- When coefficients are set in the Command List, the output behavior differs between the 2D/2F (Ver.C) Filters and other filters.
- **2D/2F Filters** allow flexible arrangement of 5×5 coefficients for each element.
- **Other filters** use the Command List coefficients in a straightforward sequential layout.

- The numbers indicate the index of the Coef data in the Command List.
 - White cells: Coef000
 - Green cells: Coef100–107
 - Orange cells: Coef200–215
 - White-outlined cells: fixed values 0.0 / 1.0
 - Gray cells: undefined values

Filter Type	Description				
2D/2F/ SAD/SSD	Selected[1:0] by element from FilterCntlOp.InOp	0	1	2	3
	FilterCntlIn Mode[3]=0				
	FilterCntlIn Mode[3]=1				
Bitmap	–	Reserved			
Non-linear	Component	B	G	R	A
	Mask 25bit	Coef000[0] – Coef215[0]	Coef000[1] – Coef215[1]	Coef000[2] – Coef215[2]	Coef000[3] – Coef215[3]
Mask	Lut 256bit	Coef207–200[15:0], Coef107–100[15:0]			
	InColor 8bit	Coef208[7:0]			
	InDelta 16bit	Coef209[15:0]			
	OutColor 8bit	Coef210[7:0]			
	OutDelta 16bit	Coef211[15:0]			
	ReplaceColor 32bit	Coef213[15:0], Coef212[15:0]			
Hamming	N 32bit	Coef101[15:0], Coef100[15:0]			
Extrema	–	Reserved			

- When coefficients are set using Blut, the coefficient output is applied uniformly as a 5 × 5 matrix across all filters.

- The numbers represent the Blut indices.
- For example, the coefficient for the central element B is formed by concatenating the 8-bit values of Blut0 and Blut1 into a 16-bit value: {Blut1, Blut0}.

Filter Type	Description																				
All	Component	B					G					R					A				
		73 72	81 80	89 88	97 96	105 104	75 74	83 82	91 90	99 98	107 106	77 76	85 84	93 92	101 100	109 108	79 78	87 86	95 94	103 102	111 110
		193	9	17	25	113	195	11	19	27	115	197	13	21	29	117	199	15	23	31	119
		192	8	16	24	112	194	10	18	26	114	196	12	20	28	116	198	14	22	30	118
		185	65	1	33	121	187	67	3	35	123	189	69	5	37	125	191	71	7	39	127
		184	64	0	32	120	186	66	2	34	122	188	67	4	36	124	190	70	6	38	126
		177	57	49	41	129	179	59	51	43	131	181	61	53	45	133	183	63	55	47	135
		176	56	48	40	128	178	58	50	42	130	180	60	52	44	132	182	62	54	46	134
		169	161	153	145	137	171	163	155	147	139	173	165	157	149	141	175	167	159	151	143
		168	160	152	144	136	170	162	154	146	138	172	164	156	148	140	174	166	158	150	142

- The positional relationship with the Command List coefficients is as follows. When replacing Command List settings with Blut, use the corresponding mapping shown below.
- This mapping also applies to coefficient values that are defined redundantly in the Coef fields.

Blut Number	Coef Number	Blut Number	Coef Number
1, 0	Coef000	25, 24	Coef200
3, 2	Coef001	27, 26	Coef201
5, 4	Coef002	29, 28	Coef202
7, 6	Coef003	31, 30	Coef203
9, 8	Coef100	33, 32	Coef204
11, 10	Coef101	35, 34	Coef205
13, 12	Coef102	37, 36	Coef206
15, 14	Coef103	39, 38	Coef207
17, 16	Coef104	41, 40	Coef208
19, 18	Coef105	43, 42	Coef209
21, 20	Coef106	45, 44	Coef210
23, 22	Coef107	47, 46	Coef211
		49, 48	Coef212
		51, 50	Coef213
		53, 52	Coef214
		55, 54	Coef215

- When coefficients are set to **SrcIn**, the values from SrcIn are converted to float and directly used as the 5×5 coefficient matrix.
- When coefficients are set to **SrcOut'**, the values from SrcOut' are converted to float and directly used as the 5×5 coefficient matrix.

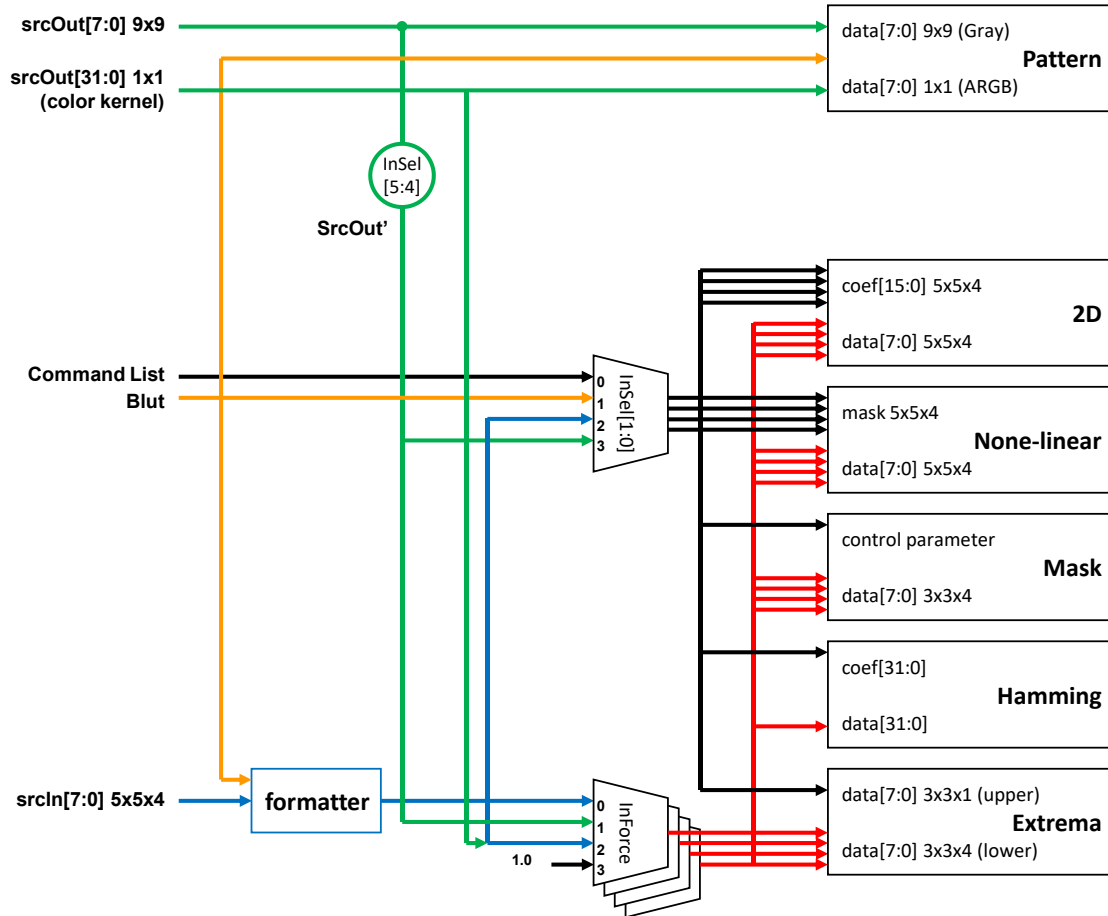


Figure 17 Inputs of Filter

3.10. Preprocessing for Bayer Images

- Input data to the filter can be selectively extracted using a 4×4 mask pattern for pixels and elements. Pixels and elements that do not match the pattern are replaced with a value of 0. This function is useful for extracting only the assigned elements from a Bayer image.

- There are two approaches for applying the mask pattern:
 1. Treat the Bayer image as a grayscale image and set non-target elements to 0.
 2. Convert the Bayer image to full color in a single pass, assigning different patterns to each ARGB component.

Figure 20 shows examples of a grayscale image with only the Green element extracted, and a full-color image where each component is extracted without overlap. The image can then be restored using filters such as a Gaussian filter.
- When extracting a single element, the $4 \times$ speed Gray image processing mode can be used.
For extracting two or more elements simultaneously, use the standard Gray image (Replica) setting.
- The mask pattern values are set via the **BayerMask0–3** registers.
Whether the Bayer mask is applied is controlled by the **FilterCntlOut.Force** setting in the Command List.

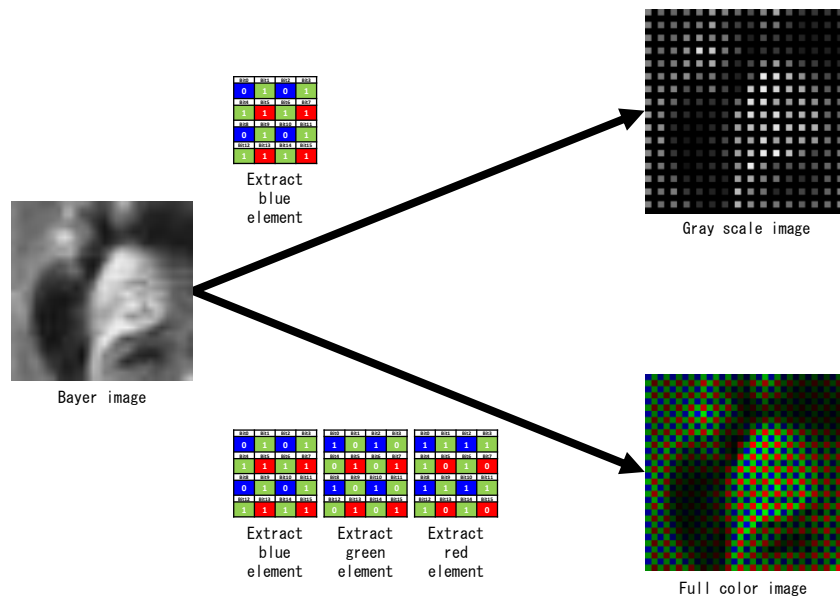


Figure 18 Bayer Image Transfer

3.11. Filter

- Eight types of filters with different purposes and kernel sizes are implemented. Seven filters are for SrcIn-type images, and one is for SrcOut-type images.

Only one filter from the same type group can be selected at a time, but filters from different groups can be processed simultaneously.

Filter Types:

- **2D Filter (SrcIn):** Performs convolution using arbitrary coefficients and pixel values with fixed-point precision, supporting kernel sizes from 5×5 to 9×9 .
- **2F Filter (SrcIn):** Performs convolution using arbitrary coefficients and pixel values with floating-point precision, supporting up to a 5×5 kernel.
- **Bitmap Filter (SrcIn):** Based on a 32×25 binary pattern, returns the nearest valid label from the center.
- **None-linear Filter (SrcIn):** Selects the median in a 3×3 region or the minimum/maximum value up to a 9×9 kernel.
- **Mask Filter (SrcIn):** Operates on the center pixel based on the median and surrounding values within a 3×3 region.
- **Extrema Filter (SrcIn):** Performs extremum detection across up to 8 layers of grayscale images within a 3×3 region.
- **Pattern Filter (SrcOut):** Converts the relationship between the median or fixed value and surrounding pixels in a 9×9 region into a binary pattern.
- For SrcIn-type filters, if the result meets certain conditions, the coordinates of the corresponding pixels can be written to memory.

The coordinates are stored as 32-bit fixed-point (X, Y), and packed sequentially in order of output.

The conditions for writing vary by filter type and are described below.

Filter Mode	Description
2D	When the result of the selected element exceeds ± 1.0 (<i>excluding</i> 1.0 itself)
2F	When the result is negative.
None-linear	Returns true if the center pixel matches the specified type (median, maximum, or minimum). If multiple candidates exist, the center pixel takes priority.
Mask	When the evaluation result is true.
Hamming	When outputting the total (sum) value.
Extrema	When a local maximum or minimum is detected.

Bitmap	None.
Pattern	None.

• Coefficients are generally provided as parameters, but they can also be replaced with data from the SrcIn or SrcOut types.

3.11.1. 2D/2F/SAD/SSD Filter (SrcIn)

- The **2D/2F Filter** performs convolution on pixel data arranged in a kernel generated by the Pixel Cache, using arbitrary (float) coefficients along with interpolation coefficients automatically derived from pixel positions. The required kernel size is automatically communicated to the Pixel Cache. Filtering can be enabled or disabled per element.
- The **2F Filter (Ver.C)** processes each pixel using half-precision floating-point arithmetic. Both input and output formats can be selected as either half-precision floating point or fixed point.
- The **2F Filter (Ver.C)** also supports **SAD (Sum of Absolute Difference)** and **SSD (Sum of Squared Difference)** operations. By specifying data and coefficients from SrcIn and SrcOut, and applying an offset to SrcOut coordinates through remapping, it identifies both the sequence of minimum values and the corresponding sequence numbers.
- **Data input** consists of 8-bit data for four elements using a 5×5 kernel. **Coefficient input** consists of 16-bit data for four elements using a 5×5 kernel.
- The following interpolation modes are available:
 - **Ver.AB:** Fixed-point truncation
 - **Ver.C:** Selectable between rounding and truncation

Interpolation Mode	Description
Nearest	Nearest-neighbor interpolation.
Bi-linear	Linear interpolation.
Bi-cubic	Cubic interpolation.

Interpolation coefficients are automatically calculated based on the fractional distances to neighboring pixels.

For each fractional distance ΔX and ΔY , the corresponding coefficients **CXn** (for the

X-axis) and **CY_n** (for the Y-axis) are computed.

These are then multiplied in matrix form to obtain the coefficients for each grid point.

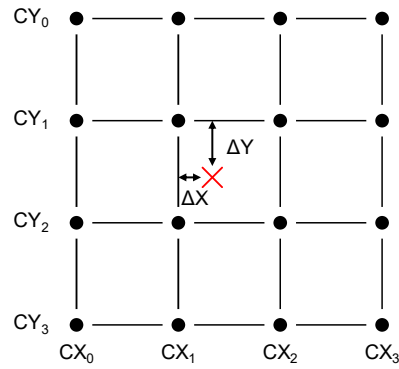


Figure 19 Interpolation Image

- In **Bi-linear mode**, the coefficients along the X-axis or Y-axis are calculated using the following formulas.
A 2×2 kernel is used.

$$C_0 = 0$$

$$C_1 = 1 - \Delta$$

$$C_2 = \Delta$$

$$C_3 = 0$$

• In **Bi-cubic mode**, the coefficients along the X-axis or Y-axis are calculated using the following formulas.

A 4×4 kernel is used.

$$C_0 = -\Delta^3 + 2\Delta^2 - \Delta$$

$$C_1 = \Delta^3 - 2\Delta^2 + 1$$

$$C_2 = -\Delta^3 + \Delta^2 + \Delta$$

$$C_3 = \Delta^3 - \Delta^2$$

- The transformation is performed by applying an affine transformation to the input coordinates based on the output coordinates, with both sharing the same origin at (0,0).

For example, doubling the size and then scaling down by half will return the image to its original state.

In contrast, a 1/2 reduction directly results in decimated pixel values from the original image.

To perform downscaling without aliasing, configure a parallel shift via affine transformation at (x, y).

For instance, in a 1/2 downscale, set m00 = 2.0, m11 = 2.0.

For more details on coordinate transformations, refer to the application note *"Coordinate Operations"*.

- The following five filter sizes are available.

Arbitrary coefficient filters and interpolation can generally be processed simultaneously, except in certain cases.

The target elements for processing can be freely selected.

Note: Bi-cubic interpolation cannot be used with arbitrary coefficients.

Filter Mode	Performance	Specifiable interpolation types
1x1	4 elements per pixel / per cycle (2F processes 1 element only)	Nearest
2x2		Nearest Bi-linear
3x3		Nearest
4x4		Nearest Bi-linear Bi-cubic
5x5		Nearest
7x7	2 elements per pixel / per cycle (2F processes 1 element only)	Nearest
9x9 (2D のみ)	1 element per pixel / per cycle	
1x1	1 element per 4 pixels	Nearest
2x2		

3x3	/ per cycle (2D only)	
4x4		
5x5		
5x5 FilterCntl Op.InSel[3]]= '1'	4 elements per 0.5 pixel / per cycle	Nearest

- Filter output may include negative values depending on the coefficients used. While downstream modules can handle negative values, it is also possible to forcibly convert the output to positive values at this stage.

Option	Description
Absolute	If the 9-bit output value is negative, convert it to a positive value.

• The SrcOut type can provide grayscale data up to a 9×9 kernel, but the 2D Filter supports only up to 5×5 per element.

Therefore, for a 9×9 kernel, the data is divided into 4 elements; for a 7×7 kernel, into 2 elements.

These are processed individually and then integrated (added) in the final stage.

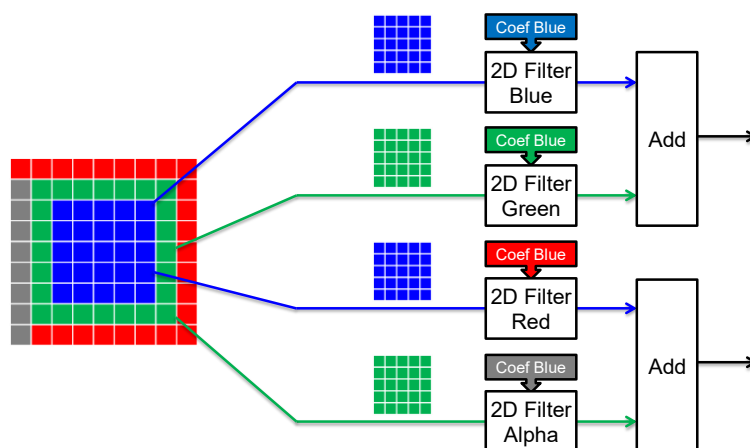


Figure 20 In case of two 7x7 Filter

- In **Ver.C**, it is possible to mix SrcIn and SrcOut pixel data within a single kernel. The central 3×3 region uses SrcIn, while the surrounding 12 pixels are sourced from SrcOut. Applying a 2D filter with arbitrary coefficients on this combined kernel allows simultaneous addition or subtraction of SrcIn and SrcOut data.
- After filtering, each element is represented in 9-bit format within the range of -1.0 to $+1.0$.
The value 0x100 corresponds to the maximum (1.0), and 0x101 corresponds to the minimum ($-255/256$).
- Under specific grayscale processing conditions, $4 \times$ speed filter operation is supported. This replaces 4-element parallel processing with 4-pixel parallel processing. Although the image is handled as if it were full-color ARGB, it is functionally grayscale.
To enable this, set the processing size and stride to one-fourth, and change the SrcIn format (SrcInInfo.Rot).

Conditions for $4 \times$ speed filtering:

- Grayscale filters with kernel size up to 5×5
- Image input is grayscale (supplied from SrcIn)
- Image width is a multiple of 4
- No element-wise operations in 3D Clut or Blender
- Statistics are gathered for each X-coordinate modulo 4 (0, 1, 2, 3)
- The **steal** flag is the sum of overflows per element (values exceeding $+1.0$ or below -1.0).

To ignore certain elements, set the corresponding FilterCntlIn.En to '0'.

- Coefficients can be weighted using a Gaussian distribution based on the difference between the center pixel and surrounding pixels (FilterCntlOp.InSel[3:2] = '2').

The variance σ of the Gaussian is specified via FilterCoef00.Thresh[3:0].

When applying a smoothing filter, using a coefficient table where large pixel differences (likely edges) result in smaller weights helps preserve edges—this is equivalent to a **Bilateral filter**.

However, the processing speed is reduced to **2 elements per cycle**, and only kernel sizes up to 5×5 are supported.

The elements and data source types (SrcIn/SrcOut) used for evaluation and filtering can be freely selected (Ver.C).

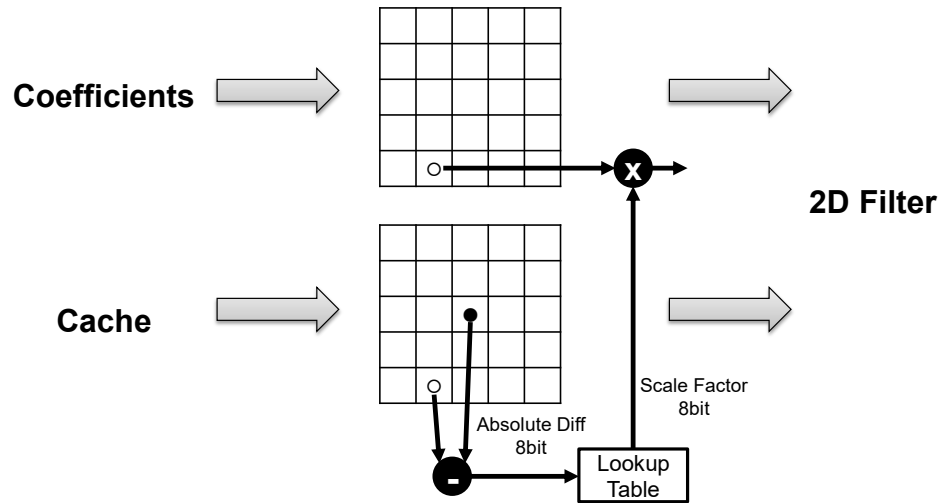


Figure 21 Coefficient Operation

3.11.2. None-linear Filter (SrcIn)

- The None-linear filter sorts pixel values within a specified region and selects the **median**, **maximum**, or **minimum** value. The median is supported up to a **3 × 3** kernel, while maximum and minimum values are supported up to **9 × 9**.
- Specific kernel positions can be masked (excluded from sorting) using coefficient settings.

Filter Mode	Performance	Combinable Interpolation Types
3x3 Min/Max/ Median	4 elements per pixel / per cycle	Nearest
5x5 Min/Max		
7x7 Min/Max	2 elements per pixel / per cycle	
9x9 Min/Max	1 element per pixel / per cycle	
3x3 Min/Max/	1 element per 4 pixels	

Median	/ per cycle	
5x5 Min/Max		

- **Data input** consists of 8-bit data for four elements using a 5×5 kernel. If the kernel size is smaller than 5×5 (as specified by the Filter Mode), the unused portions are filled with 0.
- **Coefficient input** is a 25-bit mask per element (representing the 5×5 kernel). This mask must be set regardless of the image cache size specified in FilterCntl0.InMode.

When coefficients are referenced from the Command List, specific mappings apply—refer to the corresponding bit positions for correct configuration.

- Even if the image cache size is 3×3 , the data is treated as a 5×5 array with zero-padding.

For example, in **Min filtering**, if the mask is not applied and zero-padding is included, the result will always be 0, since the 0 values will be considered in the evaluation.

Component	Parameter	Bit Location
Blue	Coef000[15:0]	Bit0–15
	Coef001[8:0]	Bit16–24
Green	Coef100[15:0]	Bit0–15
	Coef101[8:0]	Bit16–24
Red	Coef104[15:0]	Bit0–15
	Coef105[8:0]	Bit16–24
Alpha	Coef200[15:0]	Bit0–15
	Coef201[8:0]	Bit16–24

Bit9	Bit10	Bit11	Bit12	Bit13
Bit24	Bit1	Bit2	Bit3	Bit14
Bit23	Bit8	Bit0	Bit4	Bit15
Bit22	Bit7	Bit6	Bit5	Bit16
Bit21	Bit20	Bit19	Bit18	Bit17

- Each element can independently select any of the **Min**, **Max**, or **Median** filters. Additionally, the result from a specified evaluation element can be used to apply the same selected position to all other elements. For example, if element **A** is used for evaluation, the position selected (e.g., top-left) will be applied to elements **R**, **G**, and **B** as well.
- If multiple pixels within the kernel share the same evaluation value (e.g., maximum), the **center position** is prioritized. For instance, if four maximum values are found in a 5×5 kernel including the center, the center is selected as the result.
- The **SrcOut** type can supply grayscale data up to a 9×9 kernel, but the Non-linear Filter supports only up to 5×5 per element. Therefore, for 9×9 , the data is divided into 4 elements; for 7×7 , into 2 elements. These are processed individually and then integrated (via Min/Max) in the final stage. This follows the same approach as in the 2D Filter.
- After filtering, each element is represented using a **9-bit value** in the range **0.0 to 1.0**. 0x100 represents the maximum value (1.0), and 0x0 the minimum (0.0).
- The **signed flag** (SrcInInfo.Signed) for the SrcIn format is effective. Internally, a temporary offset of 0x80 is applied before sorting, and the original value is restored before output.
- The **steal flag** is asserted as **true** when the **center pixel value** (of the selected element) matches the **selected pixel value** (of the same element) resulting from the evaluation.

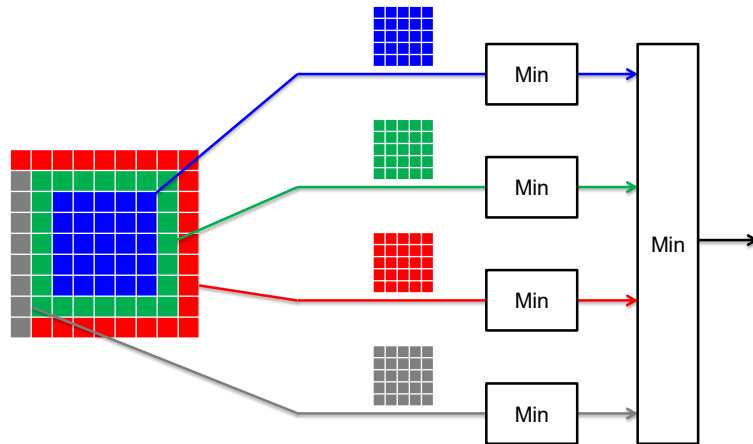


Figure 22 In case of 9x9 Minimum Filter

- You can either select one of the four flags or create a new flag by combining multiple flags.

3.11.3. Mask Filter(SrcIn)

- The Mask filter evaluates the relationship between the center value and surrounding values in a 3x3 matrix, determines the result based on the truth conditions, and outputs a specific value according to the combination.

Filter Mode	Performance	Combinable Interpolation Types
3x3	1 pixel, 1 component / cycle	Nearest

After generating the final flag as described below, data generation is performed. The flag is determined using a user-defined table based on the result of a truth evaluation.

The data output can be selected from two methods:

One method outputs the original data, replaces the data, or directly outputs the result of the truth evaluation based on the flag result.

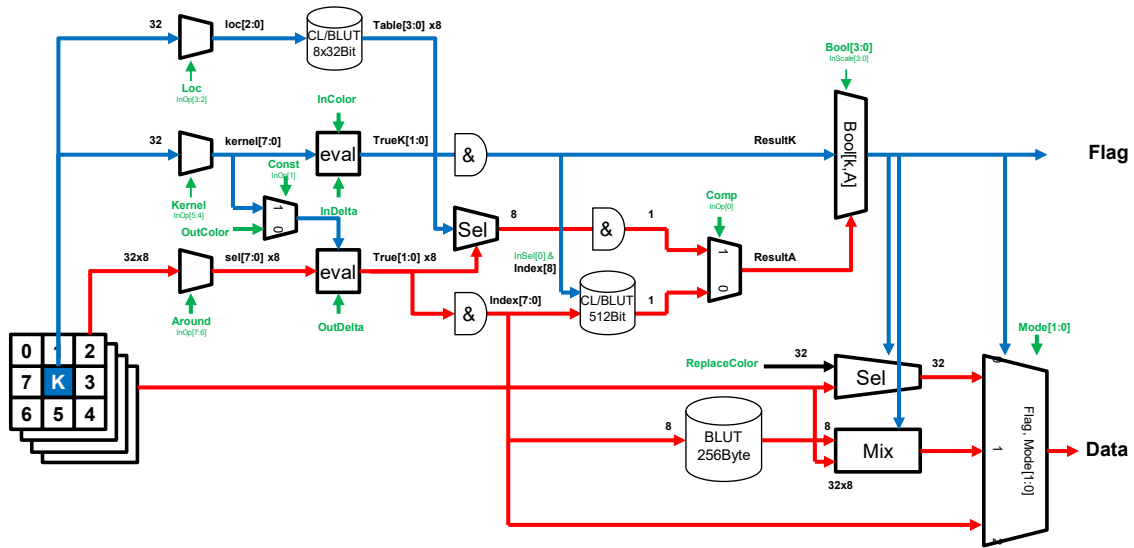


Figure 23 Mask Filter

- The truth value of the center is represented by 2 bits based on its comparison with a specified value. One component is selected arbitrarily as the center value. Note that if `FilterCenter.DeltaUpper = '0'`, the comparison is performed using `0x100`.

$$\text{TrueK}_0 = \text{KernelValue} \geq (\text{FilterCenter.Value} - \text{FilterCenter.DeltaLower})$$

$$\text{TrueK}_1 = \text{KernelValue} < (\text{FilterCenter.Value} + \text{FilterCenter.DeltaUpper})$$

- The truth value of surrounding values is represented by 2 bits based on their comparison with either a specified value or the center value. One component is arbitrarily selected as the surrounding value. Unlike coefficients, the surrounding pixels are numbered sequentially in a clockwise order. Note that if `FilterAround.DeltaUpper = '0'`, the comparison is performed using `0x100`.

$$\text{True}[i]_0 = \text{AroundValue}[i]$$

$$\geq (\text{FilterAround.Value or Kernel} - \text{FilterAround.DeltaLower})$$

$$\text{True}[i]_0 = \text{AroundValue}[i]$$

$$< (\text{FilterAround.Value or Kernel} + \text{FilterAround.DeltaUpper})$$

- The final flag is generated based on the evaluation result of the center value's truth value and the surrounding values' truth values (using a user-defined table). The evaluation of the center value's truth is performed as follows:

$$\text{ResultK} = \text{TrueK}_0 \& \text{TrueK}_1$$

The evaluation of the truth values for the surrounding pixels is performed within a 3x3 area and can be selected from the following two methods:

- Multiply the 2-bit truth value of each pixel to generate a total of 8 bits corresponding to the surrounding pixels. This 8-bit value is used as an index to a 1-bit \times 256 table (FilterTable) to obtain a 1-bit result.
- Alternatively, include the center value's result to form a 9-bit index, and use it to look up a 1-bit \times 512 table (Blut) to obtain a 1-bit result.

$$\begin{aligned} \text{Index}[i] &= \text{True}[i]_0 \& \text{True}[i]_1 \quad (i = 0 - 7) \\ \text{Index}[8] &= \text{TrueK}_0 \& \text{TrueK}_1 \end{aligned}$$

$$\begin{aligned} \text{ResultA} &= \text{FilterTable}(\text{as 1bit table})[\text{Index}[7:0]] \text{ or} \\ \text{ResultA} &= \text{Blut}(\text{as 1bit table})[\text{Index}[8:0]] \end{aligned}$$

- The lower 3 bits of the center value of the element selected by FilterCntlOp.InOp[3:2] are used as an index to a 32-bit \times 8 table (FilterTable) to obtain a 32-bit value. This value is then divided into 4-bit segments for each surrounding pixel. After evaluating the 2-bit truth value of each pixel against the corresponding 4-bit segment, the results are multiplied to obtain a final 1-bit result.

$$\begin{aligned} \text{Table} &= \text{FilterTable}(\text{as 32bit table})[\text{KernelValue}[2:0]] \\ \text{Temporary}[i] &= \text{Table}(\text{as 4bit table})[i] \quad (i = 0 - 7) \\ \text{ResultA} &= \bigcap_i \text{Temporary}[i][2 \cdot \text{True}[i]_1 + \text{True}[i]_0] \quad (i = 0 - 7) \end{aligned}$$

The final flag is obtained through the following Boolean algebra computation.

$$\text{Flag} = \text{FilterCoef00.Scale}[2 \cdot \text{ResultK} + \text{ResultA}]$$

- For example, in a Canny filter, suppose the absolute luminance is assigned to element B, and the luminance gradient (in 8 directions) is assigned to element G. The second method described above is used for evaluating the truth values of surrounding pixels. The operations for the 8 surrounding pixels are retrieved from the FilterTable using element G (luminance gradient). Each operation is expressed in 4 bits; by setting only the bit corresponding to FilterCenter.DeltaUpper to '1' and the others to '0', the first bit represents " \geq ", the second " \leq ", and the third " $=$ " in the comparison between the center and surrounding pixels. The condition is set for each surrounding pixel, and the flag is set to '1' only if all surrounding pixels satisfy the condition (operations in FilterCntlOp.InOp are also required).
- In addition to the above Mask processing, a Mix process can be performed where a new pixel value is generated based on the pattern of the truth evaluation results True[i] for the surrounding pixels by referencing a Lookup Table (Blut). For example, if the center value is determined to be inappropriate relative to the surrounding values, a new center value can be created using those surrounding pixel values. The Blut is accessed using the truth result, and only the surrounding pixels corresponding to '1' bits in the obtained 8-bit value are considered valid. If multiple bits are '1', the average of the corresponding reference pixel values is used as the replacement value.

Blut Value	Replace Value
Blut[0] = 1	Pixel 0 of the 3x3 region (see Figure 25)
Blut[1] = 1	Pixel 1 of the 3x3 region (see Figure 25)
Blut[2] = 1	Pixel 2 of the 3x3 region (see Figure 25)
Blut[3] = 1	Pixel 3 of the 3x3 region (see Figure 25)
Blut[4] = 1	Pixel 4 of the 3x3 region (see Figure 25)
Blut[5] = 1	Pixel 5 of the 3x3 region (see Figure 25)

Blut[6] = 1	Pixel 6 of the 3x3 region (see Figure 25)
Blut[7] = 1	Pixel 7 of the 3x3 region (see Figure 25)
Blut[7:0] = All 0	FilterReplace

- When configuring the FilterTable as a Blut, parameters such as FilterCenter.DeltaUpper, which are redundantly defined in the Coef section of the Command List, must also be configured within the Blut.

3.11.4. Hamming Filter(SrcIn/SrcOut) (Ver.C)

- The Hamming filter calculates the Hamming distance between two binary patterns and outputs either the smallest or largest value within a single fragmentation process (the former when FilterCntIn.Mode = '0', the latter when FilterCntIn.Mode = '1'). It also outputs the corresponding position.
- The binary data unit must be a multiple of 32 bits. The binary pattern is represented as a two-dimensional array of concatenated patterns. The size of the two-dimensional array is specified by the polygon-shaped rectangle (XMax, YMax), and the index given from pss to frComp is based on the origin at (X, Y) = 0.
- When processing two-dimensional binary patterns consecutively, it is necessary to specify whether the operation is cross-correlation or auto-correlation (auto-correlation must be disabled when using identical patterns; otherwise, the distance will always be zero). Also, the index Y from pss to frComp must be set to YMax. This enables exhaustive processing over $XMax \times 32$ bits units repeated $YMax^2$ times.
- Care must be taken with various settings. Please refer to the example configuration for performing auto-correlation between N entries of 256-bit units (32-bit \times 8). (The configuration values are in decimal format.)
- The flag used in Steal becomes true at the end of the polygon-shaped rectangle.

Parameter	Description
Delta from <i>pss</i>	Set the number of processing iterations in the X direction to 0, and configure it so that the number of processing iterations in the Y direction becomes M.
<u>MasterCntl.Shape='10'</u>	Specify the polygon-shaped rectangle, and write only the final data of the X coordinate to memory.
<u>MasterCntl.BoxX='10'</u>	Replace the destination X coordinate with the Y coordinate obtained after the shape processing.
<u>MasterCntl.BoxY='13'</u>	Replace the destination Y coordinate with the Y coordinate received from pss.
<u>MasterCntl.OutRead='1'</u>	Enable SrcOut reading.
<u>MasterCntl.OutScan='2'</u>	Modify the scan direction of the SrcOut Y coordinate using SrcOffset.
<u>SrcInInfo.Exp/Format='3'</u>	Set to 32bpp (the stride is generally based on XMax when a polygon shape is specified).
<u>SrcOutInfo.Exp/Format='3'</u>	Set to 32bpp (stride is generally determined by XMax when a polygon shape is specified).
<u>DstOutInfo.Exp/Format='3'</u>	Set to 32bpp (stride is generally determined by YMax when a polygon shape is specified).
<u>DstOffset.CoorY0='3'</u>	In SrcOut, replace the source Y coordinate with the Y coordinate received from pss.
<u>FilterCntlIn.Class='6'</u> <u>FilterCntlIn.Mode='2'</u> <u>FilterCntlOp.InSel[1:0]='3'</u>	Specify the Hamming filter (Self/Min) and select SrcOut as the coefficient.
<u>FilterCntlOp.InOp</u>	Set the output format. In the case of 0xE4. Output Data = {Eval, Comp} Output Origin = {PosY, PosX}
<u>FilterCoef00.CoefRef</u>	Set YMax for the polygon shape specification.

3.11.5. Extrema Filter(SrcIn)

- The Extrema filter treats a 3x3 region as a single layer and determines whether the center pixel is a local minimum or maximum across multiple layers, as shown in the table below. In Ver.AB, up to 5 layers can be used; in Ver.C, up to 8 layers are supported. The filter scans through the layers by shifting them three at a time, checking for extrema (minimum or maximum) at the center pixel across three consecutive layers.
- The scan starts from the lowest layer, and the evaluation stops at the first layer where the condition is met. The number of layers is specified by FilterCntl0.InScale.

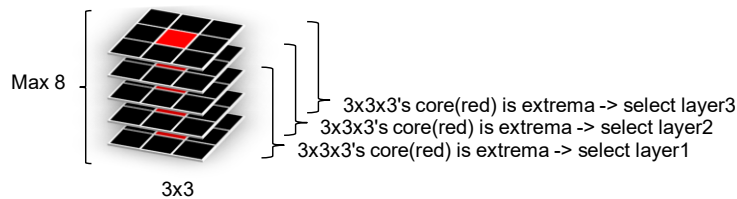


Figure 24 Extrema Select

- It is also possible to evaluate the differences between each layer (when FilterCntlIn.Mode[1] = '1'). If difference mode is selected, new differential layers are generated by subtracting adjacent input layers, resulting in (number of input layers – 1) differential layers. The total number of input layers must be specified in advance. Invalid layers are ignored.
- When FilterCntlIn.Mode[0] = '0', the original data is output. When FilterCntlIn.Mode[0] = '1', the evaluated result is output instead.
- Since the top and bottom layers do not lie at the center of any three-layer group, their evaluation result is not always true. However, virtual upper and lower layers can be defined to allow evaluation at the top and bottom layers.
- The scan starts from the lowest layer, and evaluation stops once a maximum or minimum is detected. However, it is also possible to extract both minimum and maximum values simultaneously if they exist in different layers.
- Layers 0 through 3 correspond to SrcIn pixels, and Layers 4 through 7 correspond to SrcOut pixels. In both SrcIn and SrcOut, each ARGB component corresponds to a separate layer. If the number of layers is 4 or fewer, configuration for SrcOut is not required.

Layer Number	Description
0(bottom layer)	SrcIn ElementB
1	SrcIn ElementG
2	SrcIn ElementR
3	SrcIn ElementA
4	SrcOut ElementB
5	SrcOut ElementG (Ver.C)
6	SrcOut ElementR (Ver.C)
7	SrcOut ElementA (Ver.C)

- When performing difference operations, the results are obtained from a maximum of (number of layers — 1) differential layers. If difference operations are not performed, the input data itself is used instead.
- Using the “**Coordinate Extraction (Steal)**” feature described later, only the coordinates where a minimum or maximum is detected can be written to memory. Additionally, when writing the corresponding XY coordinates, the layer number in which the minimum or maximum was found can be embedded into the upper 4 bits of each coordinate.
- The flag used in **Steal** becomes true when a local minimum or maximum is detected.

3.11.6. Bitmap Filter(SrcIn)

- The Distance mode of the Bitmap filter is a distance filter that operates on a 25x25 1-bit bitmap. It encodes the nearest position where the value is true ('1') from the center pixel into an 8-bit label. The search begins from the origin (0) and proceeds sequentially. If no '1' is found, the output is set to 0xFF.
- In nearest-neighbor searches, positions with the same distance that are line-symmetric or point-symmetric are considered equivalent. Therefore, positions categorized into 8 directional quadrants are first unified into a single representative quadrant and then assigned a label number. For example, pixels at relative coordinates (7,3), (3,7), (−3,7), (−7,3), (−7,−3), (−3,−7), (3,−7), and (7,−3) are all OR-ed and treated as having the same value as the pixel at (7,3).

- The center value's truth evaluation is expressed in 2 bits, similar to the Mask filter, and is based on a comparison with a specified value. The center (Center) can be selected from the SrcOut center value, SrcIn component centers (A, R, G, B), or a fixed value (Kernel). Kernel and Delta values are defined using entries in the Blut:

Kernel = Blut[208]

Delta0 = Blut[210]

Delta1 = Blut[211]

Center = {SrcOut_C, SrcIn[A]_C, SrcIn[R]_C, SrcIn[G]_C, SrcIn[B]_C, Kernel}

Truth conditions for each surrounding pixel *i* are evaluated as:

True[i]_0 = AroundValue \geq (Center $-$ Delta0)

True[i]_1 = AroundValue $<$ (Center $+$ Delta1)

- Up to 81 (9x9) truth flags of 2 bits each—including the center—are rearranged and mapped to any of 32 output positions (64 bits in total). The layout is defined using Blut. For example, the first of the 32 positions references Blut[0], which indicates which truth flag to place.
- Finally, either the logical AND of the 2-bit truth flags is taken and compressed into a 32-bit value, or the lower 16 truth flag sets are selected and packed into 32 bits.
- Note: the order of positions differs from the clockwise ordering used in the Mask filter. Here, pixels are scanned sequentially starting from the top-left, including the center point.

Blut Index	Description
208	Specify position #0, 0–80
209	Specify position #1, 0–80
...	
239	Specify position #31, 0–80
240	Kernel Value
241	Boolean Operator

242	Delta0 Value
243	Delta1 Value
244–255	Don't care

Bool[3:0] (Blut[241])	Description
0	0
1	$\sim \text{True}_0[i] \ \& \ \sim \text{True}_1[i]$
2	$\text{True}_0[i] \ \& \ \sim \text{True}_1[i]$
3	$\sim \text{True}_1[i]$
4	$\sim \text{True}_0[i] \ \& \ \text{True}_1[i]$
5	$\sim \text{True}_0[i]$
6	$\text{True}_0[i] \ \wedge \ \text{True}_1[i]$
7	$\sim \text{True}_0[i] \ \mid \ \sim \text{True}_1[i]$
8	$\text{True}_0[i] \ \& \ \text{True}_1[i]$
9	$\text{True}_0[i] \ \sim \sim \text{True}_1[i]$
10	$\text{True}_0[i]$
11	$\text{True}_0[i] \ \mid \ \sim \text{True}_1[i]$
12	$\text{True}_1[i]$
13	$\sim \text{True}_0[i] \ \mid \ \text{True}_1[i]$
14	$\text{True}_0[i] \ \mid \ \text{True}_1[i]$
15	1

The flag used in **Steal** is not generated directly. Instead, it is generated as needed by performing element value evaluations in subsequent stages using the **Extractor** and **Blender**.

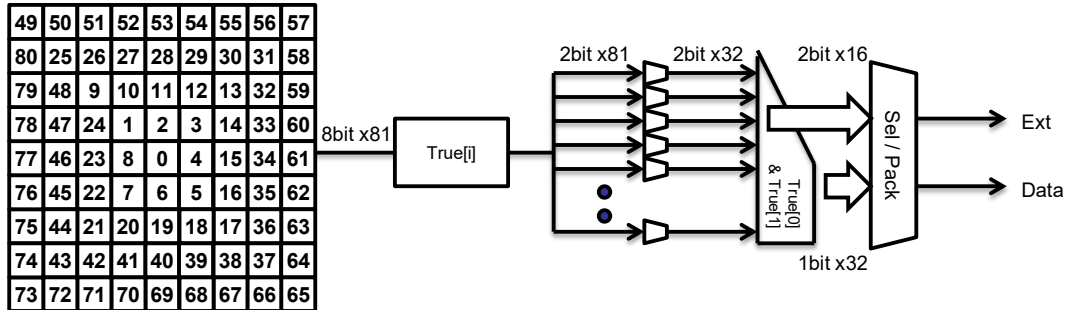


Figure 27 Pattern Filter

3.12. Envelope Processing

- Pixel values from SrcOut can be retrieved from memory and combined with either SrcMod (the filtered pixel values of SrcIn) or SrcOrg (the unfiltered pixel values of SrcIn). For any given component, replacement, addition/subtraction, or multiplication operations can be performed. This function is used for element insertion and composition.
- The newly generated pixel is passed to the 3D CLUT (color space conversion) after the following calculation. For each element, the operation (denoted as * in the formula) can be selected from four types: no operation, replacement, addition/subtraction, or multiplication. The two operands on the right-hand side can be selected from several options, including SrcOut and constants (e.g., PixelConst.A, R, G, B). Additionally, internally stored register values (such as the histogram's most frequent index, minimum value, and maximum value) can also be referenced.
- Each component of the operands can be selected freely. However, in operations involving SrcMod, the components of SrcMod cannot be freely chosen. The same restriction applies to operations involving SrcOrg.
- After envelope processing, each element is represented in a 9-bit format ranging from 0.0 to +1.0. A value of 0x100 corresponds to the maximum (1.0), and 0x0 to the minimum (0.0).

$$\text{new SrcMod}_{\text{ARGB}} = \text{Clamp} \left[\text{Abs} \left(\begin{Bmatrix} \text{SrcOrg}_{\text{ARGB}} \\ \text{SrcOut}_{\text{ARGB}} \end{Bmatrix} * \begin{Bmatrix} 1.0 \\ \text{Context}(\text{UserSet}) \\ \text{ConstA}, R, G, B \\ \text{SrcOrgA}, R, G, B \\ \text{SrcOutA}, R, G, B \end{Bmatrix} \right) \right]$$

$$\text{new SrcOrg}_{\text{ARGB}} = \text{Clamp} \left[\text{Abs} \left(\begin{Bmatrix} \text{SrcMod}_{\text{ARGB}} \\ \text{SrcOut}_{\text{ARGB}} \end{Bmatrix} * \begin{Bmatrix} 1.0 \\ \text{Context}(\text{UserSet}) \\ \text{ConstA}, R, G, B \\ \text{SrcModA}, R, G, B \\ \text{SrcOutA}, R, G, B \end{Bmatrix} \right) \right]$$

- All elements can also be concatenated into a single 32-bit value, enabling 32-bit accumulated addition. This addition starts with the first pixel value at the beginning of the line and is incrementally summed. When this result is combined with the vertical accumulation from the Blender, an integrated image can be obtained.
- SrcOut refers to the output that bypasses the Pattern Filter. In contrast, if SrcExt is defined as the output after applying the Pattern Filter, the mutual relationships among the data paths of SrcMod and SrcOrg are as illustrated in the diagram.

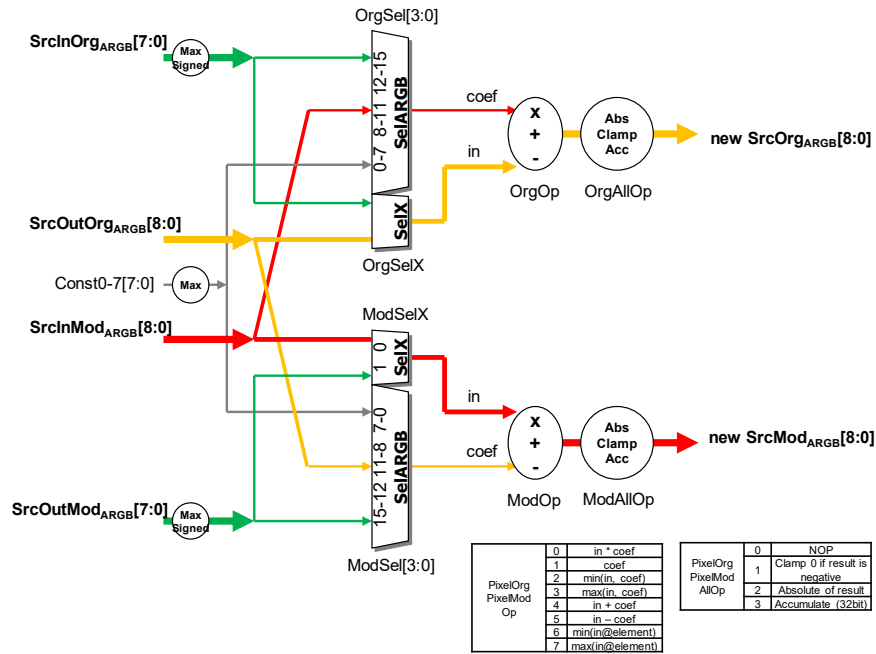


Figure 28 Envelope Path

3.13. 3D CLUT (Color Space Conversion)

- The **3D CLUT (Color Lookup Table)** is a mechanism that generates new pixel values by referencing a lookup table using three pixel components as indices. Depending on the number of reference indices used, it supports 1D, 2D, 3D, and Binary modes. As the transformation is table-based, it can perform not only color space conversions but also arbitrary functions such as L2 norms or trigonometric functions. Precision decreases as the number of reference indices increases (1D > 2D > 3D).
- Both outputs from **Envelope Processing**, SrcMod and SrcOrg, can individually enable or disable color space conversion. However, if both conversions are enabled, performance decreases from 1 pixel/cycle to 0.5 pixels/cycle.
- When frComp is activated, the reference tables are automatically loaded into internal SRAM by the **Initiator**. Two sets of caches are maintained for the reference tables. On a cache miss, 16KB of data is loaded per miss. If more than two commands use the 3D CLUT, frequent SRAM reloading will occur, potentially degrading performance.
- In 2D and 3D modes, input components are limited to 8 bits. Therefore, if a 9th bit exists to represent negative values, a folding process is required to convert 9-bit values into 8-bit format (PixelCntl*.Inword). This step is not necessary in 1D or Binary modes.
- Specific components can be selectively converted, as specified by ClutCntl.En.
- It is possible to combine two components and output them as a single 16-bit value with higher precision.

3.13.1. 1D Mode (Standard)

- Each ARGB component independently references a table. The indices are 9-bit values ranging from the minimum -1.0 (0x101) to the maximum 1.0 (0x100), requiring a configuration of 512 entries × 4 components. No interpolation is performed.
- This mode is used for per-component conversions such as gamma correction. Unlike other modes, the A component is also referenced during conversion.
- Two table entries can be referenced in 1D mode, and the selection is determined by filter flags.

3.13.2. 1D Mode (Binary)

- When ClutCntl.Sel = '1' in 1D mode, a binary lookup is performed using n bits from RGB components (R uses only its LSB). This references a 128Kbit table ($4K \times 32\text{-bit}$), and outputs a result of size $128K \div 2^n$ bits. (In Ver.AB, $n = 16$ or 17 ; in Ver.C, $n = 12\text{--}17$.)
- If the result is 1 bit, it is expanded to 9 bits and copied to ARGB. If the result is 2 bits, the LSB is copied to RGB, and the MSB to A. When expanding to 9 bits, the MasterCtrl.Max value of each component determines the expansion: if '1', it becomes 0x100; if '0', it becomes 0xff.

3.13.3. 2D Mode

- A 4K-word table is referenced using the upper 6 bits each of the R and B components (total 12 bits). Four neighboring values are retrieved from the table, and bi-linear interpolation is performed using the lower 2 bits of R and B as weights. Interpolation can be enabled or disabled via ClutCntl.Sel.
- Negative input values are rounded to zero. To convert negative values, use the folding option to reduce 9-bit values into 8-bit (i.e., discard LSB after shifting). The value 0x100 (representing 1.0) is automatically converted to 0xff, so no additional handling is required.
- Although only R and B are referenced, the result is output to all components.

3.13.4. 3D Mode

- A 4K-word table is referenced using the upper 4 bits each of the RGB components (total 12 bits). Eight neighboring values are retrieved from the table, and tri-linear interpolation is performed using the lower 4 bits of RGB as weights. Interpolation can be enabled or disabled via ClutCntl.Sel.
- Handling of negative inputs is the same as in 2D mode (see **Handling of Negative Input in 2D Mode**).
- Only RGB components are referenced, but the result is output to all components. For linear transformations such as RGB to YUV conversion, the precision is within about 0.5 LSB error compared to dedicated hardware.

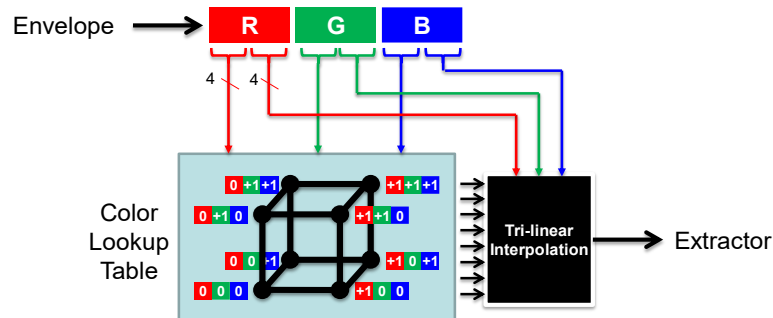
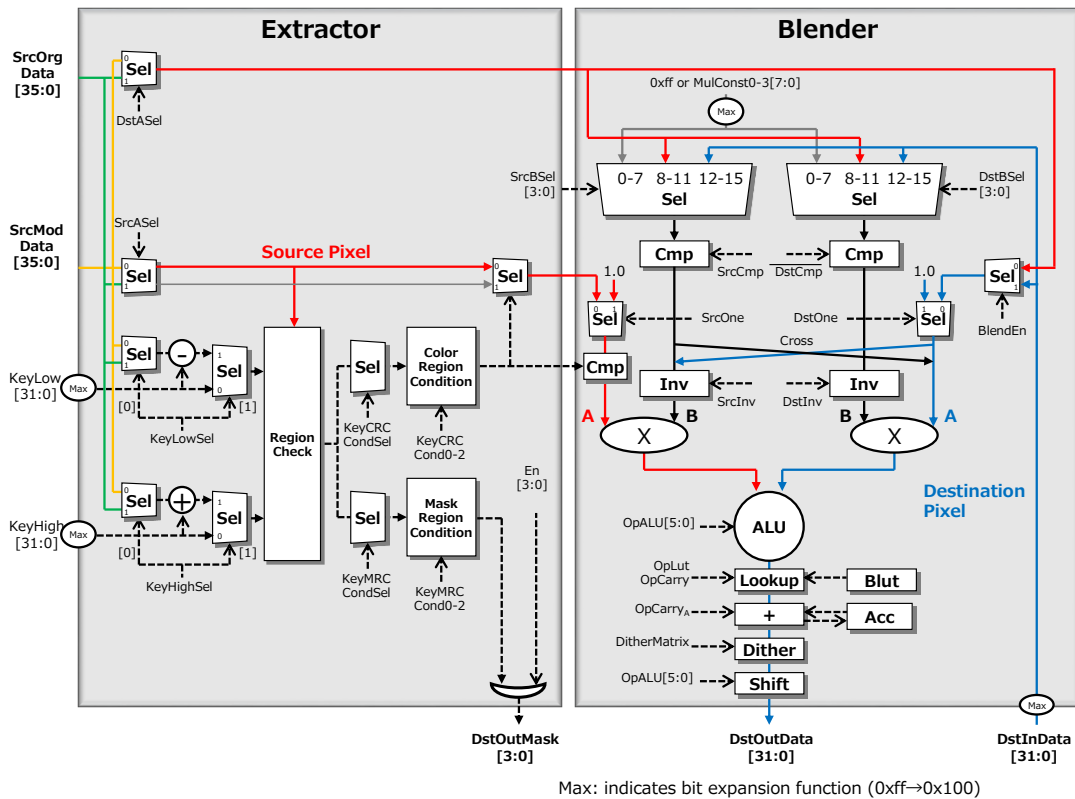


Figure 29 3D Clut Data Path

3.14. Pixel Processing (Extractor and Blender)

- Pixel processing is divided into two stages: the **Extractor** for preprocessing and the **Blender** for postprocessing. The Extractor mainly performs binarization, while the Blender handles operations between pixels.
- SrcModData and SrcOrgData, which are the results of color space conversion, are used as source data. Additionally, a new memory access is performed to retrieve destination data (DstInData). A total of three data sources are used in the composition process.
- The DstOutMask generated by the Extractor can be used to mask memory writes. When Steal is enabled, this function can be disabled via StealCntl.Mask.

Figure 30 Extractor and Blender



3.14.1. Extractor

- As shown in *Figure 33*, values less than or equal to PixelKeyLow are converted to the original pixel value, values greater than or equal to PixelKeyHigh are converted to 0xFF (0x100), and values in between are converted to 0. The reference pixel can be selected from either SrcMod or SrcOrg. Additionally, the inverse of the selected source (i.e., if SrcMod is selected, SrcOrg is used, and vice versa) can also be selected.
- Operations for masking memory writes are also supported.

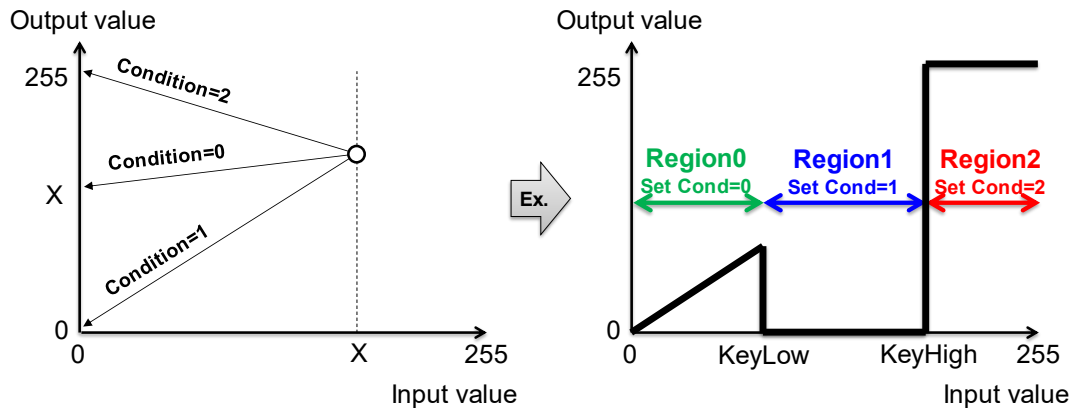


Figure 31 Pixel Extraction Example

- The lower threshold PixelKeyLow and the upper threshold PixelKeyHigh can be set not only as fixed values but also using either SrcMod or SrcOrg. This allows for applications such as adaptive binarization.

Mode	KeyHigh	KeyLow
Fixed	Specified Value	Specified Value
SrcMod	SrcMod Data + Specified Value	SrcMod Data – Specified Value
SrcOrg	SrcOrg Data + Specified Value	SrcOrg Data – Specified Value

- The combination of the element used for evaluation and the element to be modified is freely configurable. For example, you can define a region based on the lower and upper thresholds of element A, and apply the resulting operation to the RGB elements.

3.14.2. Blender

- The Blender processes data in the following sequence: **pixel and element selection, division and complement, multiplication, ALU operations** (e.g., alpha blending), **function conversion**, and **error diffusion**. The element used for alpha blending can be freely selected.

- Pixel and element selection, along with division and complement operations, are used to generate operands for multiplication. Using the Source data generated by the Extractor and the Destination data retrieved from memory, Operand A and Operand B are determined, and two multiplication operations are performed.

Operand (Source)	Source	Selection
A	Extractor : SrcMod	The output is as described on the left.
	Extractor : SrcOrg	
B	Initiator	A constant, its complement, or its reciprocal
	Context	UserSet or Min/Max, or their complement or reciprocal
	Extractor : SrcMod	Any element of the above-mentioned output, or its complement or reciprocal
	Extractor : SrcOrg	
	Memory	

Operand (Destination)	Source	Selection
A	Memory	The output is as described on the left.
	Extractor : SrcMod	
	Extractor : SrcOrg	

B	Initiator	A constant, its complement, or its reciprocal
	Context	UserSet or Min/Max, or their complement or reciprocal
	Extractor : SrcMod	Any element of the above-mentioned output, or its complement or reciprocal
	Extractor : SrcOrg	
	Memory	

- In the ALU, element-wise binary operations are performed using the two results generated by the multipliers.

ALU Operation	Description
Add	Addition
Sub	Subtraction
Absolute	Absolute difference
Boolean	Bitwise operations (16 types)
Mul	Multiplication
Min	Minimum value
Max	Maximum value
Flag	Comparison result (==, !=, >, <, <=, >=)

- Up to the ALU input stage, operations are performed using 9-bit signed values per element. After the ALU output, each element is reduced to 8 bits. The lower 8 bits of the ALU result are directly output. If clamping is required—where negative values are clamped to 0, and values equal to or greater than 0x100 are clamped to 0xFF—then clamping-enabled addition or subtraction is used in the ALU.
- The carry-out signal generated by the ALU can be propagated in the order of BGRA elements. For example, to implement a 16-bit accumulator, elements B and G, and elements R and A can be grouped together, allowing the carry signal to propagate within these pairs. If only the upper 8 bits (elements G and

- A) of a 16-bit frame input are set to 0, accumulated addition or subtraction using elements B and R becomes possible.
- A 32-bit accumulator is available for each line processing cycle. It is reset to 0 at the beginning of the line, and accumulates and outputs 32-bit ARGB values until the end of the line.
- The output of the adder/subtractor can be passed through a user-defined **Blender Lookup Table (Blut)** to perform custom transformations such as logarithmic or square root functions. Although the Blut is also used by various filters, it cannot be shared across multiple functions simultaneously.
- BG elements can be processed as **half-precision floating-point** data for accumulation and multiplication (Ver.C).
- Simple **error diffusion** is applied to the final output pixel using a **Dither Matrix** (DitherLow, DitherHigh). The Dither Matrix is a user-definable 4×4 matrix of signed 4-bit values. Dither coefficients are automatically selected using the lower 2 bits of the destination X and Y coordinates (4 bits total). The values are configured via the Dither0 and Dither1 registers.

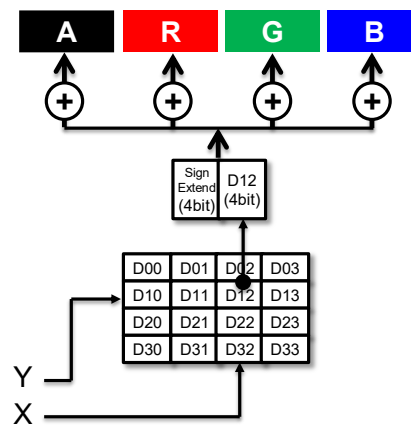


Figure 32 Dithering

3.15. Coordinate Extraction (Steal)

- Whether to write a given coordinate to memory is determined based on the filter processing result (flags) and the Extractor result (DstOutMask). Note that the Extractor mask is also used by the Blender for memory write masking. To enable the Extractor mask only for Steal and disable it for the Blender, set StealCntl.Mask.

- Finally, based on the evaluation result, the corresponding coordinates are sequentially stored in memory. Since serial address management is required, context read/write must occur with each fragmentation process (COCntl.Base and COCntl.En).
- The total number of extracted coordinates is written to word 0 of the context. If the context is not cleared beforehand (COCntl.Clr), counting will continue from the previous value, so care must be taken.
- Similarly, the evaluation result is written to word 1 of the context. If the result is true, the value 0xFFFFFFFF is written; if false, 0 is written. Once a pixel is evaluated as true, it will not become false again until the context is cleared.
- By using the polygon context reference in one dimension, the total number of extracted coordinates can be set from the context, and the extracted coordinates can be transformed via Remap for further processing.

3.16. Histogram (Ver.BC)

- The results of the Blender can be accumulated into histograms for each element. The accumulated values are expressed as 24-bit integers. Due to saturation logic, the count will never exceed the maximum representable value ($2^{24} - 1$). The accumulation can either be cleared at the start of processing or continued from the previous state without resetting.

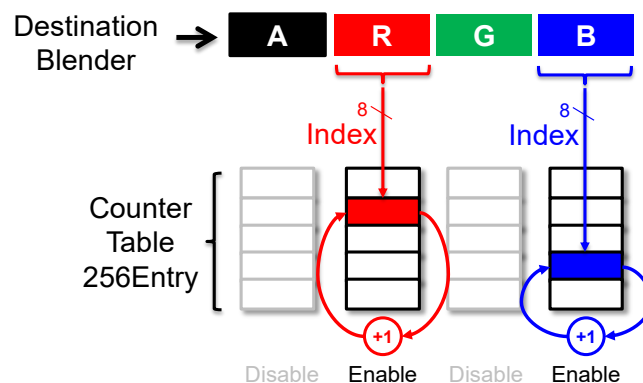


Figure 33 Histogram Counter and Table

- When processing of a single frame is completed, the histogram results can be written to memory. The selection of components and the number of

components to be written can be configured freely. Additionally, results can be written sequentially by region using a zigzag scan of the indices. Note that each polygon is counted as one frame.

- Similarly, at the end of processing for a frame, the **minimum and maximum values** for each component can also be written to memory as context data. These context values can be referenced in subsequent processes such as Polygon, Envelope, or Blender.
- In memory, the results are basically stored sequentially by component in the order of processing. For example, when acquiring histograms of only components B and A for each frame, the histogram of component B is written first, followed by that of component A. The number of index entries (2HistCntl1.Num) is configurable.
- Histogram results are aggregated per unit (HistCntl0.Unit). For example, to aggregate per frame (XY), set $\text{HistCntl0.Unit} = '1'$; to aggregate across Z -indexed frames ($XY \times Z$), set $\text{HistCntl0.Unit} = '2'$.
- The following is an example of acquiring histograms using a zigzag scan. A small area (XY) is processed for $Z \times W$ regions. For each small area, a histogram is generated for RGB components only, and written to memory according to the specified number of indices. Each small area's histogram is written in the order of component B to R, and this is repeated Z times. After writing histograms for all Z regions, the histogram stride for Z updates (HistCntl0.Stride) is applied to begin the next set of Z histograms. This process is repeated W times. If the results are to be tightly packed, the stride should be set to Z (or technically $Z - 1$).

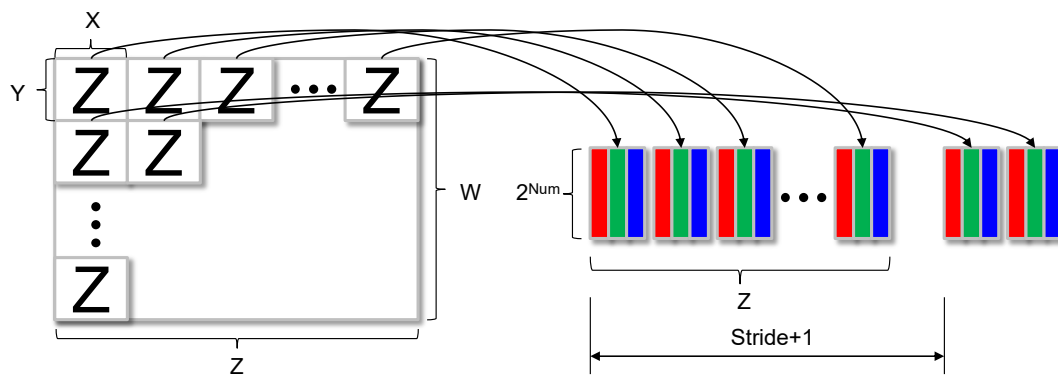


Figure 34 Histogram of Zigzag Scan

3.17. Use of Blut

- The **Blut (Blender Lookup Table)** is referenced by both the Blender and various Filters. Since overlapping data cannot be configured when accessed simultaneously, mutual exclusivity is required. In principle, the Blender and Filters cannot be configured to use the Blut at the same time. However, the Pattern Filter and other Filters can be configured simultaneously.
- The approximate usage range of the Blut is outlined below. For detailed information, please refer to the specifications of each respective function.

Byte Address	Blender	2D	NL	Mask	Hamming	Extrema			
0	Lut	Coefficient	Mask	Lut	Value	Top Layer 3x3			
8									
16									
24									
32					Color				
40									
48					Color				
56									
64				Color					
72									
80				Color					
88									
96		Color							
104									
112		Color							
120									
128		Color							
136									
144		Color							
152									
160		Color							
168									
176		Color							
184									
192	Color								
200									
208	Color								
216									
224	Color								
232									
240	Color								
248									
Pattern Filter Value									
Bayer Mask									

3.18. Address Masking

- In frame memory addressing, it is possible to fix the upper bits of the address while allowing only the lower bits to vary (**Base.Wrap). This enables data

exchange between engines to fit within the capacity of a ring buffer rather than the full spatial capacity of a frame buffer.

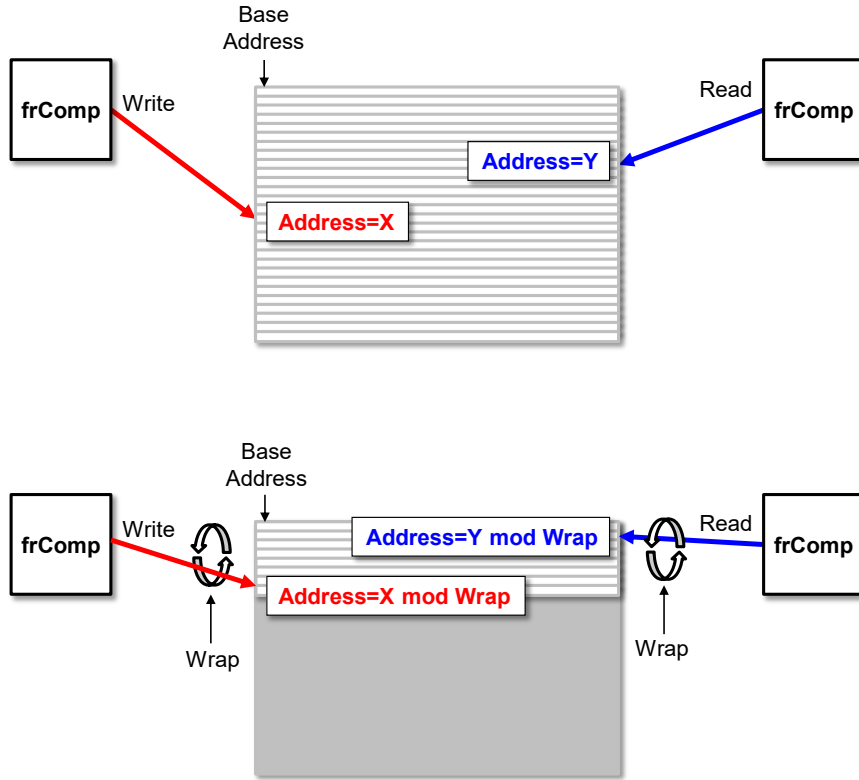


Figure 35 Ring Buffer Control by Address Mask

- The address mask enables variation of the address up to a specified lower bit position, while the upper bits retain the value set by `***Base.Base`. Therefore, to make effective use of this feature, the product of the image start address, the line update stride, and the number of valid lines must be a power of two.
- Ring buffer management is controlled using indices provided to **frComp**. It monitors the producer and consumer indices and performs conflict-free ring buffer control (similar to FIFO pointer management). When using **pss**, ring buffer control between engines can be automatically managed through link-based coordination.

3.19. Input/Output Format

- The pixel data in memory supports the following formats. Bits and words are packed starting from the MSB in memory.

Bit/Word	Component	Description
8	A or R or G or B or Gray	<ul style="list-style-type: none">On read, values are assigned to all elements of the ARGB pipeline; on write, only a selected element is written.The maximum value is treated as a fixed-point number less than 1.0. For the value 0xFF, it is selectable whether to interpret it as 1.0 or 255/256.
16	ARGB or RGB	<ul style="list-style-type: none">For RGB, the 5, 6, and 5-bit values are expanded to 8 bits respectively for read/write operations. The A (alpha) component is assigned a grayscale value via a simplified calculation.If there are insufficient bits in the LSB direction, the MSB-side bits are duplicated to fill the gap.
	A or R or G or B or Gray (Ver.C)	<ul style="list-style-type: none">Half-precision floating-point inputA fixed-point format using the lower 10 bits with the 5-bit exponent set to 0 (it is also possible to specify 8 bits per word and assign the lower 8 bits).
24	RGB (Ver.C)	<ul style="list-style-type: none">RGB is assigned 8, 8, and 8 bits respectively for read/write operations. The A (alpha) component is assigned a grayscale value via a simplified calculation.
32	ARGB	<ul style="list-style-type: none">ARGB is assigned 8, 8, 8, and 8 bits respectively for read/write operations.

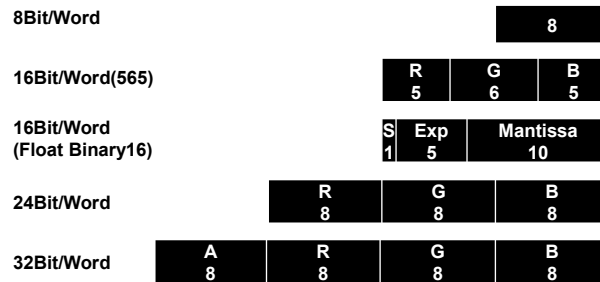


Figure 36 Pixel Format(ARGB)

The coordinate data in memory read by the Remapper supports the following formats. The same formats apply to coordinate data written out from the filter stage.

Bit/Word	Component	Description
32	X, Y	<ul style="list-style-type: none"> • X and Y coordinates packed into a 32-bit value • Two's complement representation, with X and Y each ranging from -32,767 to 32,767 • 0x8000 is treated as an escape value (see relevant section for details) • To set subpixel precision, the fractional bit position can be specified via the Command List

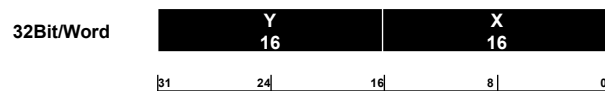


Figure 37 Coordinate Format (X, Y)

When the coordinate data read by the Remapper is an escape value, the following operations are performed (excluding texture conversion). Depending on the settings, escape values can also be ignored.

	X ≠ Escape	X = Escape
Y ≠ Escape	<p>[Normal Mode]</p> <ul style="list-style-type: none"> • (X,Y) Operate as Coordinates 	<p>[Copy Mode]</p> <ul style="list-style-type: none"> • For the first coordinate in a line: <ul style="list-style-type: none"> If absolute mode is set (MasterCntl.DstRemap / SrcRemap = '0'), the fallback coordinate is (0, Y). If relative mode is set (MasterCntl.DstRemap / SrcRemap = '1'), the fallback coordinate is (0, 0). • For coordinates in the middle of a line, the most recent valid result is used.
Y = Escape	<p>[Zero Mode]</p> <ul style="list-style-type: none"> • Absolute Value Setting • MasterCntl.DstRemap • / When SrcRemap = '0', intermediate coordinates (X, Y) are set as relative values. • MasterCntl.DstRemap / SrcRemap = '1'で (0,0) 	<p>[Default Mode]</p> <ul style="list-style-type: none"> • Treated as the coordinate (0x8000, 0x8000) during operation • For the source path, the default value in the cache is used (note that escape values may be altered by matrix transformation) • For the destination path, a pixel mask is applied (no write operation is performed)

- The transformation matrix referenced by Affine/Homography transformations supports the following formats. The floating-point format (Float) used is a subset of IEEE 754 representation. NaN and Infinity are not supported.

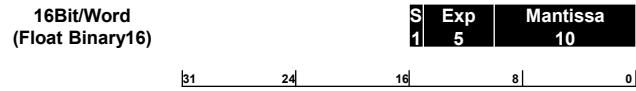
Bit/Word	Component	Description
32	Matrix Element	<ul style="list-style-type: none"> Each Element of the 3×3 Matrix $\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} m00 & m01 & m03 \\ m10 & m11 & m13 \\ m20 & m21 & m23 \end{pmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}$ $\begin{pmatrix} X' \\ Y' \end{pmatrix} = \begin{pmatrix} u/w \\ v/w \end{pmatrix}$



Figure 38 Matrix Format (M)

- The coefficients referenced by the 2D Filter, and the inputs referenced by the 2F Filter, support the following formats. The floating-point format (Float) used is a subset of IEEE 754 representation. NaN and Infinity are not supported.
- In the 2D Filter, values are converted to 10-bit fixed-point format before multiplication, so the lower 6 bits (beyond the valid range) are set to 0. Please round the mantissa or otherwise adjust the floating-point values in advance according to the expected significant digits.

Bit/Word	Component	Description
16	Coefficient	<ul style="list-style-type: none"> The 2D Filter consists of the following 5×5 coefficients. $\begin{pmatrix} c0 & c1 & c2 & c3 & c4 \\ c5 & c6 & c7 & c8 & c9 \\ c10 & c11 & c12 & c13 & c14 \\ c15 & c16 & c17 & c18 & c19 \\ c20 & c21 & c22 & c23 & c24 \end{pmatrix}$ <ul style="list-style-type: none"> Only values in the range from -2.0 to 2.0 are supported (excluding -2.0 and 2.0). If the exponent (Exp) is 0, bits [9:2] of the mantissa represent the fractional part of the fixed-point value.



- Color Lookup Table data, whether accessed via registers or from memory, supports the following formats.

Bit/Word	Component	Description
32	ARGB	<ul style="list-style-type: none"> Referenced as ARGB with 8, 8, 8, and 8 bits assigned respectively.

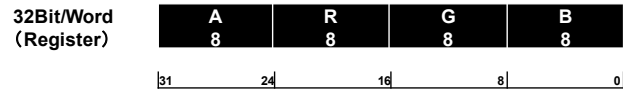


Figure 39 Color Lookup Table Format (RGB)

Vertices in the Command List support the following formats.

Bit/Word	Component	Description
32	X, Y	<ul style="list-style-type: none"> X and Y coordinates packed into a 32-bit value Unsigned representation, with X and Y each ranging from 0 to 65,535
32	Size	<ul style="list-style-type: none"> 32-bit size representation

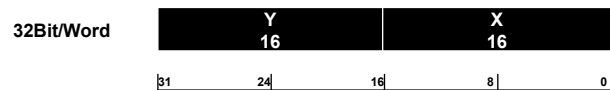


Figure 40 Vertex Format (X, Y)

The Dither register supports the following formats.

Bit/Word	Component	Description
4	Matrix Element	<ul style="list-style-type: none"> Each element of the following 4×4 matrix is represented using 4 bits, and elements from d00 to d13 and from d20 to d33 are packed into 32 bits. Select either the destination coordinate X or Y, and add Δ in two's complement form to the lower bits of the pixel. $\Delta = sel \begin{pmatrix} d00 & d01 & d02 & d03 \\ d10 & d11 & d12 & d13 \\ d20 & d21 & d22 & d23 \\ d30 & d31 & d32 & d33 \end{pmatrix}$

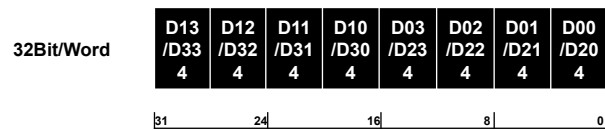


Figure 41 Dither Matrix Format (D)

3.20. Internal Computation

- The **internal coordinate representation** (between remapping and matrix transformation) uses unsigned 8-bit integers plus 6-bit fractional fixed-point format.
- Computation precision in the 2D Filter** is fixed-point with a maximum of signed 15-bit integer plus 4-bit fraction.
Computation precision in the 2F Filter is half-precision floating-point.
- Computation precision in the 3D CLUT** is fixed-point with a maximum of unsigned 8-bit integer plus 8-bit fraction.
- The **precision of pixel data** in the Extractor, Blender, and other stages is typically fixed-point signed 9-bit per component. Two's complement is used, but 0x100 represents a positive value of 1.0.
Pixels output from the Blender are forcibly clamped to **unsigned 8-bit**.
For BG elements, **half-precision floating-point** can be used for accumulation and multiplication (Ver.C).

3.21. Connection with pss

- The **iAddr signal** output from pss is used to fetch the **Command List** from memory. See the section on the Command List for further details.
If pss is not present, access the pss interface directly.
- The **iIndex signal** output from pss, combined with parameters in the Command List, is used to compute the starting addresses of the input and output image data.

The initial address is calculated using the following formula, where:

- X, Y, Z are 0th, 1st, and 2nd-order coordinates,
- StrideX is the step size for Y-axis changes,
- Len is the size in bytes per pixel,
- Buf is a switch indicating the front/back buffer as specified by the address-setting command,
- The **Plus term** is used for **double-buffering** (Ver.C): the LSB of the Z coordinate is used to offset by one screen's worth ($\text{StrideX} \times \text{WidthY}$), enabling double-buffer control.

$$\text{StartAddress} = (\text{BaseAddress} + X + \text{StrideX} \times Y + \text{Plus}) \times \text{Len}$$

$$\text{Plus} = \text{StrideX} \times \text{WidthY} \times (Z[0] \otimes \text{Buf})$$

- **Scanline processing** is performed starting from coordinates X and Y for the transfer length specified by the iDelta signal output from pss. The iDelta signal generally defines the unit of fragmentation. The end of a line may result in a partial segment. It is also possible to process an entire line at once.
- The **least significant bit of bit 4 in the address (iAddr) signal** output by pss is not used as part of the actual address value. Therefore, the Command List should be stored in memory in **32-byte aligned units**.
- Conversely, this LSB can be set to '1' to provide **initialization hints** to frComp. When such a hint is given, a cache clear signal (iR_{xw}) is deasserted during memory read access. External memory systems should monitor this signal and clear any read-only caches accordingly. This is not required for read/write caches. Note that the **Flush flag** in the Command List (present in each memory access register) must also be set.
- The iCID signal serves as a **tag to specify registers maintained within the same frame**. A fixed value may be used, but doing so may result in fragmentation limitations for some functions. Refer to the precautions for fragmentation with ID 0. If fragmentation is not required, fixed values present no issue.

3.22. Performance

- The system operates at **1 pixel per cycle**, regardless of the number of components per pixel. However, in the **2D Filter**, cache access—due to simultaneous access to multiple pixels—can become the **primary memory performance bottleneck**.
- **Memory access wait states** can degrade performance.
 - Waits may occur due to the ratio of required memory bandwidth to the available memory bus bandwidth.
 - Waits may also result from the system's inability to absorb **read latency** (i.e., the time from address request to data acknowledge).
- A **high-bandwidth memory bus** is required. However, since the same data is often read multiple times, using a **centralized cache system** can significantly improve performance. To maintain **cache coherence**, the system outputs external flush signals (mrRxw / mcRxw).

4. Register Description

4.1. Overview

- All registers are accessed via the **control bus**.
- Some registers may affect pipeline behavior or performance, so the **timing of configuration** must be handled carefully.
- The following access types are used in the register descriptions:
 - **R** — Read Only (writes have no effect)
 - **R/W** — Read / Write
 - **R/WC** — Read / Write, auto-cleared after read
- Do not access **Reserved registers**, and always write '0' to **Reserved fields**.
- In address and data notations, 'x' indicates a **don't care** value.

4.2. Definition

Address	Register Name	Description
0000_0000	Reset	Reset Control
0000_0004	System	System Control
0000_0100	DitherLow	Dither Control (Lower)
0000_0104	DitherHigh	Dither Control (Upper)

4.3. Details

4.3.1.1. Reset Register

[Address: 0x0000_0000]

31		28			24			20			16			12			8			4			0
																							Reset

Name	Type	Default	Description
Reset	R/W0		Synchronous reset: When set to '1', the internal reset state is activated and will automatically be cleared. Unlike the reset_n signal, the contents of other registers are preserved.

Upon setting '1', the rstReq signal is immediately asserted. This signal notifies external systems

that frComp has entered a reset state and requests appropriate handling. Once the external handling is complete, the rstAck signal must be asserted (if no action is required, rstAck should always be held at '1'). After these procedures are completed, the Reset automatically returns to '0'.

4.3.1.2. System Register

[Address: 0x0000_0004]

31				28				24				20				16				12				8				4				0
																																GateOff

Name	Type	Default	Description
GateOff	R/W	0	Gated Clock Off Mode: When set to '1', all bits of the gate signal are fixed to '1'.

4.3.1.3. DitherHigh/Low Register

[Address: 0x0000_0100 – 0x0000_0104]

31			28			24			20			16			12			8			4			0
D13[3:0]			D12[3:0]			D11[3:0]			D10[3:0]			D03[3:0]			D02[3:0]			D01[3:0]			D00[3:0]			
D33[3:0]			D32[3:0]			D31[3:0]			D30[3:0]			D23[3:0]			D22[3:0]			D21[3:0]			D20[3:0]			

Name	Type	Default	Description
D _{YX}	R/W	0	Sets the dithering matrix used by the Blender. In DYX, Y represents the row and X the column, both determined by the lower 2 bits of the destination coordinate. DYX is expressed in 4-bit two's complement format. It is sign-extended to 4 bits (MSB) and added to each component. This is used when dithering is applied in the ALU stage of the Command List.

4.3.1.4. BayerMask0-3 Register

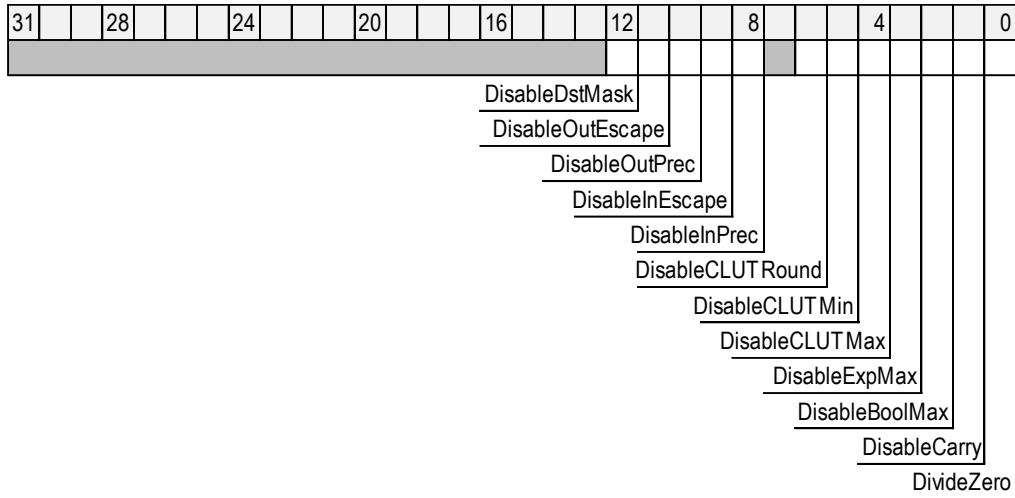
[Address: 0x0000_0140 – 0x0000_014c]

31				28				24				20				16				12				8				4				0
Mask1																Mask0																
Mask3																Mask2																
Mask5																Mask4																
Mask7																Mask6																

Name	Type	Default	Description
Mask _n	R/W	0	Sets a 4 × 4 Bayer mask pattern in 16-bit units before Filter processing.

4.3.1.5. Utility Register

[Address: 0x0000_0200]



Name	Type	Default	Description
DisableDstMask	R/W	0	In the Destination path, setting to '1' disables masking.
DisableOutEscape	R/W	0	In OutRemap, setting to '1' ignores escape codes.
• DisableOutPrec ^(注)	R/W	0	In OutRemap, setting to '1' disables precision correction for 1 × 2 and 2 × 1 formats.
DisableInEscape	R/W	0	In InRemap, setting to '1' ignores escape codes.
DisableInPrec ^(注)	R/W	0	In Remap, setting to '1' disables precision correction for 1 × 2 and 2 × 1 formats.
DisableCLUTRound	R/W	0	In CLUT, setting to '1' disables rounding in each result.
DisableCLUTMin	R/W	0	In CLUT, if the binary computation result is

0xFF, it is forcibly set to 0x100.

DisableCLUTMax	R/W	0	In CLUT, even if the normal computation result is 0xFF, it is not forcibly set to 0x100.
DisableExpMax	R/W	0	In Blender : 0: The maximum value of the fp16 exponent is set to 0x1F. 1: The maximum value is limited to 0x1E.
DisableBoolMax	R/W	0	In Blender, setting to '1' disables forced conversion of Boolean algebra input 0x100 to 0xFF.
DisableCarry	R/W	0	In Blender, setting to '1' clears the carry between operations in the order $A \leftarrow R \leftarrow G \leftarrow B$.
DivideZero	R/W	0	In Blender, when division by zero occurs: 0 is treated as 0.0, and 1 is treated as 1.0.

(Note): In SrcRemap and DstMap, if the mapping data format is 32-bit single-precision floating-point, the setting must be '1' to **disable precision correction**.

5. Command List Description

5.1. Overview

- The **Command List** is stored in memory in **256-byte units**. The starting address of the Command List is indicated by the iAddr signal output from pss. After startup, frComp fetches the Command List and loads it into internal registers.
- Each stage in the pipeline independently manages the necessary parameters in alignment with its timing. This allows seamless execution of different Command Lists without requiring synchronization or monitoring of completion status from pss.
- **Reserved commands and fields must always be set to '0'.**
- The listed addresses are **relative to the address output by pss**. They must be **aligned to 16 bytes**.

5.2. Definition

Address	Command Name	Description
00	MasterCntl	Master Control
04	Vertex0	Coordinates of Vertex 0
08	Vertex1	Coordinates of Vertex 1
0c	Vertex2	Coordinates of Vertex 2
10	PixelCntlB	Pixel Control (B Component)
14	PixelCntlG	Pixel Control (G Component)
18	PixelCntlR	Pixel Control (R Component)
1c	PixelCntlA	Pixel Control (A Component)
20	PixelKeyCRC	Pixel Key Color Range Control
24	PixelKeyMRC	Pixel Key Color Mask Range Control
28	PixelKeyLow	Pixel Key Color Reference (Lower)
2c	PixelKeyHigh	Pixel Key Color Reference (Higher)
30	PixelOrg	Pixel Control (Post Non-Filter Processing Path)
34	PixelMod	Pixel Control (Post-Filter Processing Path)

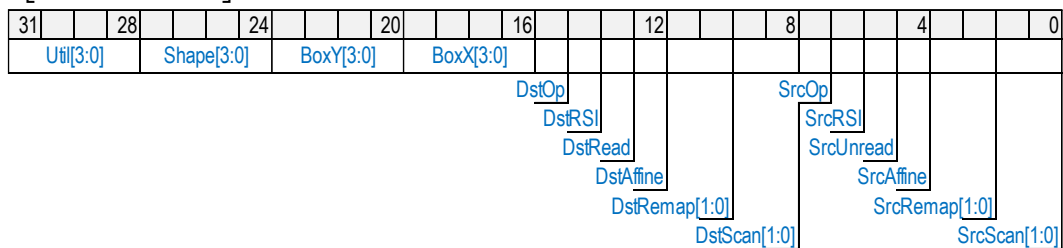
38	PixelDefault	Pixel Default Value (ARGB)
3c	PixelConst	Pixel Constants (C0, C1, C2, C3)
40	SrcInInfo	Source Input Information
44	SrcInBase	Source Input Base Address
48	SrcOutInfo	Source Output Information
78	SrcOutBase	Source Output Base Address
50	SrcMapInfo	Source Mapping Information
54	SrcMapBase	Source Mapping Base Address
58	SrcSize	Source Region Size
5c	SrcOffset	Source Coordinate Offset
60	DstInInfo	Destination Input Information
64	DstInBase	Destination Input Address
68	DstOutInfo	Destination Output Information
6c	DstOutBase	Destination Output Address
70	DstMapInfo	Destination Mapping Information
74	DstMapBase	Destination Mapping Base Address
78	DstSize	Destination Region Size
7c	DstOffset	Destination Coordinate Offset
80	CICntl	Context Input Control
84	COCntl	Context Output Control
88	HistCntl0	Histogram Control 0
8c	HistCntl1	Histogram Control 1
90	ClutCntl	Clut (Color Lookup Table) Control
94	BlutCntl	Blut (Blend Lookup Table) Control
98	StealCntl	Coordinate Extraction Control
9c	AffineCoef0	Matrix Transformation Coefficient(0 行 0 列)
a0	AffineCoef1	Matrix Transformation Coefficient(0 行 1 列)
a4	AffineCoef2	Matrix Transformation Coefficient(0 行 2 列)
a8	AffineCoef3	Matrix Transformation Coefficient(1 行 0 列)
ac	AffineCoef4	Matrix Transformation Coefficient(1 行 1 列)
b0	AffineCoef5	Matrix Transformation Coefficient(1 行 2 列)
b4	AffineCoef6	Matrix Transformation Coefficient(2 行 0 列)

b8	AffineCoef7	Matrix Transformation Coefficient(2 行 1 列)	
bc	AffineCoef8	Matrix Transformation Coefficient(2 行 2 列)	
c0	FilterCntl0	Filter Control 0	
c4	FilterCntl1	Filter Control 1	
c8	FiltCoef00/FilterTable	Filter Coefficients 00	Option
cc	FiltCoef01/FilterTable	Filter Coefficients 01	Option
d0	FiltCoef10/FilterTable	Filter Coefficients 10	Option
d4	FiltCoef11/FilterTable	Filter Coefficients 11	Option
d8	FiltCoef12/FilterTable	Filter Coefficients 12	Option
dc	FiltCoef13/FilterTable	Filter Coefficients 13	Option
e0	FiltCoef20/FilterTable	Filter Coefficients 20	Option
e4	FiltCoef21/FilterTable	Filter Coefficients 21	Option
e8	FiltCoef22/FilterCenter	Filter Coefficients 22	Option
ec	FiltCoef23/FilterAround	Filter Coefficients 23	Option
f0	FiltCoef24/FilterReplace	Filter Coefficients 24	Option
f4	FiltCoef25	Filter Coefficients 25	
f8	FiltCoef26	Filter Coefficients 26	
fc	FiltCoef27	Filter Coefficients 27	

5.3. Details

5.3.1.1. MasterCntl Command

[Address: 0x00]



Name	Description
Util[3:0]	Specify special processing.

Util[3]	Description
0	The intermediate coordinates transformed by Shape and DstScan become the Destination coordinates.
1	The coordinates generated by the Destination Remapper are used as the Destination coordinates.

Util[2]	Description
0	Normal Output Mode.
1	<p>Planar Output Mode</p> <p>Since this mode uses <i>DstInInfo</i>, setting <i>PixelCntl.BlendEn</i> is prohibited.</p> <p>Each bit of <i>DstInInfo[3:0]</i> enables output for an arbitrary plane among the four available planes:</p> <ul style="list-style-type: none"> • Bit 0: Enables an 8-bit element plane using <i>DstInBase.Addr[31:6]</i> as the base address. • Bit 1: Enables an 8-bit element plane by adding <i>DstInInfo[23:4]</i> to the upper 20 bits of the base address. • Bit 2: Enables an 8-bit element plane by combining <i>{DstInBase[11:0], DstInInfo[31:24]}</i> as the upper 20 bits of the base address. • Bit 3: Enables an 8-bit element plane by using <i>DstInBase[31:12]</i> as the upper 20 bits of the base address.

Util[1]	Description
0	—
1	<p>Double Buffer Mode</p> <p>SrcIn and SrcOut:</p> <p>If <i>Src*.SZ</i> is '1', the buffer capacity, calculated as $(Src*Info.Stride + 1) \times SrcSize.WidthY$, is added based on the parity of the Z-coordinate. <i>Src*.Rdc[2]</i> determines the parity: '0' for odd, '1' for even.</p>

	<p><i>Src*.SZ</i> and <i>Src*.Rdc[2]</i>, which affect other functions, are forcibly set to 0.</p> <p>DstIn and DstOut: If <i>Dst*.SZ</i> is '1', the buffer capacity, calculated as $(Dst*Info.Stride + 1) \times DstSize.WidthY$, is added based on the parity of the Z-coordinate. <i>Dst*.Rdc[2]</i> determines the parity: '0' for odd, '1' for even. <i>Dst*.SZ</i> and <i>Dst*.Rdc[2]</i>, which affect other functions, are forcibly set to 0.</p>
--	--

Util[0]	Description
0	<u> </u>
1	<p>Performs unsigned data maximum value extension. Values are set in the order of ARGB from the MSB. When set to '1', data with a value of 0xFF is treated as 0x100 (i.e., 1.0). This applies to:</p> <ul style="list-style-type: none"> • Blender inputs • ARGB values of <i>PixelKeyHigh</i>, <i>PixelKeyLow</i>, and <i>PixelConst</i> • Inputs to the 2D Filter and Non-linear Filter • Reference data for the 3D Clut <p>Note: In Boolean operations, since 0xFF becomes 0x100, the target bit may be inverted—caution is required.</p>

Shape[3:0]

Specify the polygon shape.

Shape[3]	Description
0	<p>When Shape[2:0] is <i>Normal</i>, <i>Line</i>, or <i>Rect</i>:</p> <ul style="list-style-type: none"> • No special handling. <p>When Shape[2:0] is <i>Triangle</i>:</p> <ul style="list-style-type: none"> • The Fill Rule is ignored, and double edge hits are allowed.
1	<p>When Shape[2:0] is <i>Normal</i>, <i>Line</i>, or <i>Rect</i>:</p> <ul style="list-style-type: none"> • Only the start point of the line is

	<p>rasterized.</p> <p>When Shape[2:0] is <i>Triangle</i>:</p> <ul style="list-style-type: none"> Follows the Fill Rule.
--	---

Shape[2:0]	Description
0	<p>Normal (Fragmented Line): Uses the <i>iIndex</i> signal as the intermediate coordinate and the <i>iDelta</i> signal as the width.</p>
1	<p>Line: Scans a line with a 32-bit length, using <i>Vertex0.Y</i> as the upper 16 bits and <i>Vertex0.X</i> as the lower 16 bits. The length must be explicitly set (note that it is not specified as "length - 1"; if the value is 0, the line is skipped).</p>
2	<p>Rectangle: Uses the line connecting (<i>Vertex0.X</i>, <i>Vertex0.Y</i>) and (<i>Vertex1.X</i>, <i>Vertex1.Y</i>) as the diagonal of the rectangle, and the <i>iDelta</i> signal as the width.</p>
3	<p>Triangle: Uses (<i>Vertex0.X</i>, <i>Vertex0.Y</i>), (<i>Vertex1.X</i>, <i>Vertex1.Y</i>), and (<i>Vertex2.X</i>, <i>Vertex2.Y</i>) as the vertices of the triangle.</p>
4	<p>Context Reference – Normal (Fragmented Line): Uses the <i>iIndex</i> signal as the intermediate coordinate and <i>Context Data 0</i> as the width.</p>
5	<p>Context Reference – Line: Scans a line using <i>Context Data 0</i> as a 32-bit length. The length must be explicitly set (note that it is not specified as "length - 1"; if the value is 0, the line is skipped).</p>
6	<p>Context Reference – Rectangle: Performs rectangular scanning using <i>Context Data 0</i>.</p> <ul style="list-style-type: none"> Upper 16 bits specify the height (Y-direction) Lower 16 bits specify the width (X-direction)

	direction) Note: Values are not specified as "length – 1". If either value is 0, the scan is skipped.
7	Reserved

BoxY

Modifies the final Destination Y-coordinate.

BoxY	Description
0	Normal (Y-coordinate after polygon shaping)
1–9	Add the W-coordinate as an offset to the Y-coordinate after polygon shaping: $Y + 2 \times \text{BoxY} \times W$
10	Replace with the X-coordinate after polygon shaping.
11	Replace with 0.
12	Replace with the X-coordinate.
13	Replace with the Y-coordinate.
14	Replace with the Z-coordinate.
15	Replace with the W-coordinate.

BoxX

Modifies the final Destination X-coordinate.

BoxX	Description
0	Normal (X-coordinate after polygon shaping)
1–9	Add the Z-coordinate as an offset to the X-coordinate after polygon shaping: $X + 2 \times \text{BoxX} \times Z$
10	Replace with the Y-coordinate after polygon shaping.
11	Replace with 0.
12	Replace with the X-coordinate.
13	Replace with the Y-coordinate.
14	Replace with the Z-coordinate.
15	Replace with the W-coordinate.

DstOp	Setting it to '1' changes the matrix transformation to Rotate mode.
DstRSI	<p>Setting it to '1' disables the coordinate size setting <i>DstSize</i> of the Destination Remapper (edge processing is not performed in the Remapper).</p> <p>Additionally, when performing bi-linear or higher-order interpolation, the reference map must be made one size larger.</p>
DstAffine	<p>Setting it to '1' supplies matrix parameters (translation components) to the coordinate output of the Source Remapper.</p> <p>When set to '0', the default values <i>AffineCoef2</i> and <i>AffineCoef5</i> are used.</p>
DstRead	<p>Setting it to '1' enables memory access using the coordinates generated by the Destination Remapper to prepare data for the SrcOut system.</p> <p>In this case, <i>SrcOutInfo</i> and <i>SrcOutBase</i> must be configured.</p> <p>When set to '0', the generated coordinates themselves are treated as pixel values.</p>
DstRemap	In the Destination Remapper, the mapping settings for the Destination are configured. Reference addresses and related parameters are defined in <i>DstMapInfo</i> and <i>DstMapBase</i> .

DstRemap	Description
0	NOP
1	The read mapping data is used as the new X and Y coordinates.
2	The read mapping data is added to the X and Y coordinates.
3	The read mapping data is subtracted from the X and Y coordinates.

DstScan

Performs transformation of the intermediate coordinates for the Destination.

This operation takes place after the polygon shaping transformation.

DstScan	Description
[0]	Setting it to '1' allows modification of the X-coordinate using the <i>DstOffset</i> command.
[1]	Setting it to '1' allows modification of the Y-coordinate using the <i>DstOffset</i> command.

SrcOp

Setting it to '1' enables Rotate mode for the matrix transformation.

Setting it to '0' enables Homography mode.

Src Op	Dst Op	Src Affine	Dst Affine	Description
0	0	0	0	$\begin{pmatrix} u_s \\ v_s \end{pmatrix}$
0	0	0	1	$\begin{pmatrix} u_s + u_d + m_{02} \\ v_s + v_d + m_{12} \end{pmatrix}$
0	0	1	0	$\frac{\begin{pmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \end{pmatrix} \begin{pmatrix} u_s \\ v_s \\ 1 \end{pmatrix}}{\begin{pmatrix} m_{20} & m_{21} & m_{22} \end{pmatrix} \begin{pmatrix} u_s \\ v_s \\ 1 \end{pmatrix}}$
0	0	1	1	$\frac{\begin{pmatrix} m_{00} & m_{01} & u_d \\ m_{10} & m_{11} & v_d \end{pmatrix} \begin{pmatrix} u_s + m_{02} \\ v_s + m_{12} \\ 1 \end{pmatrix}}{\begin{pmatrix} m_{20} & m_{21} & m_{22} \end{pmatrix} \begin{pmatrix} u_s + m_{02} \\ v_s + m_{12} \\ 1 \end{pmatrix}}$
0	1	0	0	Reserved
0	1	0	1	Reserved
0	1	1	0	Reserved
0	1	1	1	Reserved
1	0	0	0	$\begin{pmatrix} 0 \\ 0 \end{pmatrix}$

1	0	0	1	$\begin{pmatrix} u_d \\ v_d \end{pmatrix}$
1	0	1	0	$\begin{pmatrix} \cos v_s & -\sin v_s & m_{02} \\ \sin v_s & \cos v_s & m_{12} \end{pmatrix} \begin{pmatrix} m_{20}u_s \\ m_{21}u_s \\ m_{22} \end{pmatrix}$
1	0	1	1	$\begin{pmatrix} \cos(v_s + m_{12}) & -\sin(v_s + m_{12}) & u_d \\ \sin(v_s + m_{12}) & \cos(v_s + m_{12}) & v_d \end{pmatrix} \begin{pmatrix} u_s \\ v_s \end{pmatrix}$
1	1	0	0	$\begin{pmatrix} ul_s + u_d \\ uu_s + v_d \end{pmatrix}$
1	1	0	1	Reserved
1	1	1	0	$\begin{pmatrix} \cos v_s & -\sin v_s & ul_s \\ \sin v_s & \cos v_s & uu_s \end{pmatrix} \begin{pmatrix} u_d \\ v_d \\ m_{22} \end{pmatrix}$
1	1	1	1	$\begin{pmatrix} \cos(v_s + m_{12}) & -\sin(v_s + m_{12}) & ul_s \\ \sin(v_s + m_{12}) & \cos(v_s + m_{12}) & uu_s \end{pmatrix} \begin{pmatrix} u_s \\ v_s \\ m_{22} \end{pmatrix}$

u_s, v_s : Source remap out

ul_s, uu_s : Source remap out's lower and upper

u_d, v_d : Destination remap out

m00=AffineCoef0, m01=AffineCoef1, m02=AffineCoef2

m10=AffineCoef3, m11=AffineCoef4, m12=AffineCoef5

m20=AffineCoef6, m21=AffineCoef7, m22=AffineCoef8

SrcRSI

Setting it to '1' disables the coordinate size setting *SrcSize* of the Source Remapper (edge processing is not performed in the Remapper).

Additionally, when performing bi-linear or higher-order interpolation, the reference map must be made one size larger.

SrcUnread

Setting it to '1' disables memory access based on the coordinates generated by the Destination Remapper.

SrcOutInfo and *SrcOutBase* do not need to be configured.

Note that this is the inverse logic of *DstRead*.

Use this setting in cases where the source image is not needed, such as screen clearing, to reduce memory load. When set to '1', the generated coordinates themselves are treated as pixel values.

- SrcAffine

Setting it to '1' applies a matrix transformation to the coordinate output of the Source Remapper.

The transformation matrix is defined in *AffineCoef*.
- SrcRemap

In the Source Remapper, the mapping settings for the Source are configured.

Reference addresses and related parameters are defined in *SrcMapInfo* and *SrcMapBase*.

For more details, refer to *DstRemap*.
- SrcScan

Performs transformation of the intermediate coordinates for the Source.

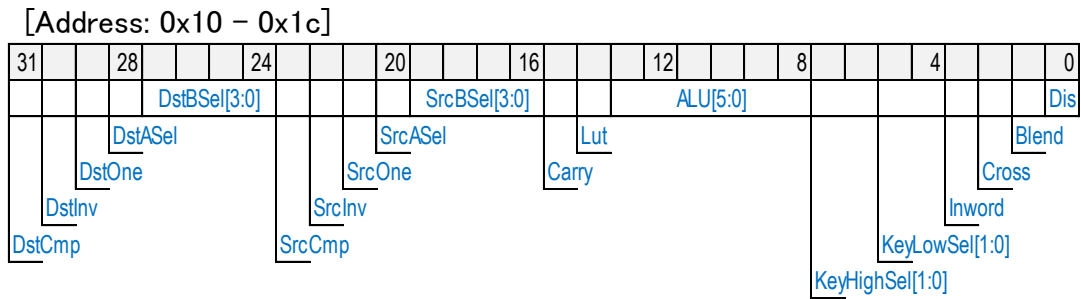
This operation is applied after the polygon shaping transformation.

For details, refer to *DstScan*.

5.3.1.2. Vertex0-2 Command

[Address: 0x04 – 0x0c]																																								
31				28					24					20					16						12					8					4					0
Y[15:0]																X[15:0]																								
Name		Description																																						
Y		Specify a positive Y-coordinate. Negative values are not supported.																																						
X		Specify a positive X-coordinate. Negative values are not supported.																																						

5.3.1.3. PixelCntlB,G,R,A Command



Name	Description
DstCmp	In the Blender, set the one's complement or two's complement of the Destination pixel. Note that this is the inverse of <i>SrcCmp</i> .

DstCmp	Description
0	Take the complement of the Destination pixel.
1	Use the Destination pixel as-is.

DstInv

Setting it to '1' applies the reciprocal of the pixel value after *DstCmp* processing in the Blender. (Refer to Figure 32.)

DstOne

In the Blender, sets the multiplicand of the Destination pixel to 1.0.

DstASel

Configures the selection of the Destination pixel in the Blender:

- '0': Use *SrcModData**
 - '1': Use *SrcOrgData**
- If *Blend* is '1', this setting is ignored and the input data referenced by the Blender is used instead.

DstBSel

Sets the selection of the Destination pixel **before** *DstCmp* processing in the Blender.

DstBSel	Description
0	Use 0xFF (1.0) as the value.
1	Specified element of Context Data 1.
2	Specified element of Context Data 2.
3	Specified element of Context Data 3.
4	PixelConst.B
5	PixelConst.G
6	PixelConst.R
7	PixelConst.A
8	Element B selected by <i>DstASel</i> .
9	Element G selected by <i>DstASel</i> .
10	Element R selected by <i>DstASel</i> .
11	Element A selected by <i>DstASel</i> .
12	Element B of <i>DstIn</i> .
13	Element G of <i>DstIn</i> .
14	Element R of <i>DstIn</i> .
15	Element A of <i>DstIn</i> .

SrcCmp

In the Blender, sets the one's complement or two's complement of the Source pixel. Note that this is the inverse of *DstCmp*.

SrcCmp	Description
0	Use the Source pixel as-is.
1	Take the complement of the Source pixel.

SrcInv

Setting it to '1' applies the reciprocal of the pixel value after *SrcCmp* processing in the Blender. (Refer to Figure 32.)

SrcOne

In the Blender, sets the multiplicand of the Source pixel to 1.0.

Note: Unlike the Destination side, this may be followed by one's or two's complement settings in the Extractor stage.

SrcASel

In the Extractor, sets the selection of the Source pixel:

- '0': Use *SrcModData*
 - '1': Use *SrcOrgData*
- The final Source pixel may be replaced with 0x00, 0xFF, or inverted based on the region control of the Extractor.

SrcBSel

Sets the selection of the Source pixel **before** *SrcCmp* processing in the Blender. The behavior is the same as *DstBSel*.

Carry

Setting it to '1' enables carry propagation (carry-over) between adjacent elements during ALU operations in the Blender.

Only the A, R, and G elements of ARGB can have carry set:

- A adds the carry from R
- R adds the carry from G
- G adds the carry from B

If only A and G are enabled, AR and RG are treated as 16-bit elements for multiplication.

If A, R, and G are all enabled, ARGB is treated as a 32-bit element for multiplication. When all ARGB carry flags are '1', carry generation is disabled, and 32-bit accumulation is performed.

The accumulation value is reset to 0 at the start of each line. The output will be the accumulated value (the first value of the line is the initial ARGB value).

Lut

In the Blender, configures the use of *Blut* and *Dither*.

- '0': Not used
- '1': Used

ALU

Selects the type of ALU operation in the Blender. Operations are performed per element.

- **S**: Source pixel input to the ALU
- **D**: Destination pixel

ALU[5:4] = '0': Addition/Subtraction mode (*).

ALU[3:0]	Description	ALU[3:0]	Description
0	$S + D$	8	$S + D$ w/o clamp
1	$D + S$	9	$D + S$ w/o clamp
2	$S - D$	10	$S - D$ w/o clamp
3	$D - S$	11	$D - S$ w/o clamp
4	$ S + D $	12	$ S + D $ w/o clamp
5	$ D + S $	13	$ D + S $ w/o clamp
6	$ S - D $	14	$ S - D $ w/o clamp
7	$ D - S $	15	$ D - S $ w/o clamp

*In standard calculations, results greater than 0xFF are clamped to 0xFF, and results less than or equal to 0 are clamped to 0.

However, if '**w/o clamp**' is enabled, clamping is not performed, and the lower 8 bits of the result are used.

ALU[5:4]=1 の場合 (Boolean*)

ALU[3:0]	Description	ALU[3:0]	Description
0	0	8	$S \& D$
1	$\sim S \& \sim D$	9	$S \sim \sim D$
2	$S \& \sim D$	10	S
3	$\sim D$	11	$S \sim D$
4	$\sim S \& D$	12	D

5	$\sim S$	13	$\sim S \mid D$
6	$S \wedge D$	14	$S \mid D$
7	$\sim S \mid \sim D$	15	1

*Operations are performed on a per-bit basis, and no carry is generated.

ALU[5:4]=2 の場合
(Mul/Hamming/Min/Max/Sum/Dither/Float/Shift)

ALU[3:0]	Description
0	S * D Unsigned
1	S * D Signed
2	256 * S * D Unsigned
3	256 * S * D Signed
4	Min(S, D)
5	Max(S, D)
6	Sum(S ^ D) (Ver.C)
7	Dither(S) (Ver.C)
8	S + D /float16 (Ver.C)
9	Reserved
10	Reserved
11	Reserved
12	Shift[2n+1:2n] = ALU[1:0] (Ver.C) if Component A then n=0 if Component R then n=1
13	
14	
15	

- **Sum()** counts the number of '1' bits (Hamming distance).
- For **Mul** results in **Unsigned** mode:
 - Values $\geq 0x100$ are clamped to 0xFF
 - Values ≤ 0 are clamped to 0x00
- For **Mul** results in **Signed** mode:
 - Values $\geq 0x80$ are clamped to 0x7F
 - Values $\leq -0x80$ are clamped to 0x80
- **Shift[3:0]** is treated as a two's complement value; output is left-shifted accordingly (right-shifted if negative).

When ALU[5:4] = '3' (Comparison results):

ALU[3:0]	Description	ALU[3:0]	Description
0	Flag (注)	8*	0
1	Reserved	9*	1
2	Reserved	10*	S == D
3	Reserved	11*	S != D
4	Reserved	12*	S > D
5	Reserved	13*	S < D
6	Reserved	14*	S >= D
7	Reserved	15*	S <= D

- When **ALU[3:0]** is 8 or higher, the 1-bit comparison result is replicated across 8 bits.

The comparison targets are the following flags (8-bit):

{S ≤ D, S ≥ D, S < D, S > D, S ≠ D, S = D, 1, 0}

KeyHighSel

In the Extractor, configures the selection for the upper bound in region-based comparison.

KeyHighSel	Description
0	Selects control based on the filter flag. (Note: If KeyLowSel is 0, this option is forcibly selected.)
1	Select the value of PixelKeyHigh .
2	Select the value of SrcModData + PixelKeyHigh .
3	Select the value of SrcOrgData + PixelKeyHigh .

- The combination of **KeyLowSel** / **KeyHighSel** values '10' and '01' is **not allowed**.

KeyLowSel

In the Extractor, configures the selection for the lower bound in region-based comparison.

KeyLowSel	Description
0	Select control based on the filter flag (<i>Forces this selection if KeyHighSel is 0</i>)
1	Select the value of PixelKeyLow
2	Select the value of SrcModData – PixelKeyLow
3	Select the value of SrcOrgData – PixelKeyLow

Inword

In the 3D Clut, folds 9-bit input element data into 8-bit. Has no effect on elements not using the 3D Clut.

When set to '1', adds 1.0 (0x100), right-shifts the result to fit into 8 bits, allowing direct conversion (MSB becomes '0').

- Maximum negative value –1.0 (0x101) becomes 0
- Maximum positive value 1.0 (0x100) becomes 0xFF

Cross

In the Blender, swaps the data paths of the Source and Destination.

- The complemented result of *SrcCmp* on the Source side is routed to multiplicand B on the Destination side
- Multiplicand A from the Destination side is routed into the reciprocal unit on the Source side
(Refer to **Figure 32** for details.)

Blend

In the Blender, configures whether Destination data is used:

- Set to '1': Reads and uses data from *DstInInfo* and *DstInBase*
- Set to '0': Substitutes with *SrcModData** or *SrcOrgData** as selected by *DstASel*

Dis

Controls whether Blender output is enabled ('0' enables output).

The masking behavior of each element depends on *DstOutInfo.Format*.

DstOutInfo.Format	Description
0	Element ARGB values are output using the following Wired-OR logic: $\text{`Out} = A \ \& \ \sim\text{DisA} \quad R \ \& \ \sim\text{DisR}$
1	Don't care
2	Functions as Byte Disable (Byte Mask)
3	

5.3.1.4. PixelKeyCRC Command

[Address: 0x20]

31		28		24		20		16		12		8		4		0
ACond[5:0]				RCond[5:0]				GCond[5:0]				BCond[5:0]				
ASel[1:0]				RSel[1:0]				GSel[1:0]				BSel[1:0]				

Name	Description
A,R,G,BSel	In the Extractor, selects the evaluation result (belonging range) to be referenced for each element operation. The <i>Cond</i> setting is applied to the selected evaluation result.

Sel	Description
0	Select the evaluation result of element B
1	Select the evaluation result of element G
2	Select the evaluation result of element R
3	Select the evaluation result of element A

Name: A, R, G, BSel

Description:.

Sel	Description
0	Select the evaluation result of element B
1	Select the evaluation result of element G
2	Select the evaluation result of element R
3	Select the evaluation result of element A

A, R, G, BCond

Defines the control region for each ARGB element.

If **PixelCntl*.KeyHighSel = '0'** or **PixelCntl*.KeyLowSel = '0'**, the control is based on the filter's Boolean flag (**Flag**)

(Refer to **Figure 32**).

Cond	Description										
[5:4]	Each 2-bit field in Cond[5:0] defines the behavior under specific comparison conditions.										
	<table><tr><th>Value</th><th>Description</th></tr><tr><td>0</td><td>Select element of Src*; if <i>PixelCntl*.SrcOne = '1'</i>, select 1 instead</td></tr><tr><td>1</td><td>Select element of Src' ; if <i>PixelCntl*.SrcOne = '1'</i>, select 0</td></tr><tr><td>2</td><td>Select complement of element from Src*; if <i>PixelCntl*.SrcOne = '1'</i>, select 0</td></tr><tr><td>3</td><td>Select complement of element from Src' ; if <i>PixelCntl*.SrcOne = '1'</i>, select 1</td></tr></table>	Value	Description	0	Select element of Src* ; if <i>PixelCntl*.SrcOne = '1'</i> , select 1 instead	1	Select element of Src' ; if <i>PixelCntl*.SrcOne = '1'</i> , select 0	2	Select complement of element from Src* ; if <i>PixelCntl*.SrcOne = '1'</i> , select 0	3	Select complement of element from Src' ; if <i>PixelCntl*.SrcOne = '1'</i> , select 1
	Value	Description									
	0	Select element of Src* ; if <i>PixelCntl*.SrcOne = '1'</i> , select 1 instead									
	1	Select element of Src' ; if <i>PixelCntl*.SrcOne = '1'</i> , select 0									
	2	Select complement of element from Src* ; if <i>PixelCntl*.SrcOne = '1'</i> , select 0									
3	Select complement of element from Src' ; if <i>PixelCntl*.SrcOne = '1'</i> , select 1										
[3:2]	Selected when : <ul style="list-style-type: none">• Src > Low and Src <= High, or• Src > High and Src <= Low (Use the same value definitions as Cond[5:4]) When using flag control , this condition applies if Flag = '1'										
	[1:0]	Selected when : <ul style="list-style-type: none">• Src <= Low and Src <= High (Use the same value definitions as Cond[5:4]) When using flag control , this condition applies if Flag = '0'									

Notes

- **Src**: Source element selected by PixelCntl¥*.SrcASel

- **Src'** : Inverse selection of the element chosen by SrcASel
- **Low**: Lower bound value selected by PixelCntl¥*.KeyLowSel
- **High**: Upper bound value selected by PixelCntl¥*.KeyHighSel

5.3.1.5. PixelKeyMRC Command

[Address: 0x24]

31			28				24					20				16					12					8					4				0
ACond[5:0]								RCond[5:0]								GCond[5:0]								BCond[5:0]											
ASel[1:0]								RSel[1:0]								GSel[1:0]								BSel[1:0]											

Name	Description
------	-------------

A, R, G, BSel

In the Extractor, selects the evaluation result (range classification) used for element masking.
The *Cond* settings are applied to the selected evaluation result.
Masking is only valid for 8-bit elements.
The final mask result is a logical OR with **PixelCntl¥*.Blend** (see **Figure 32**).

Sel

Description

- | | |
|---|---------------------------------------|
| 0 | Select evaluation result of element B |
| 1 | Select evaluation result of element G |
| 2 | Select evaluation result of element R |
| 3 | Select evaluation result of element A |

A, R, G, BCond

In the Extractor, defines the control region for masking of each element. Even if another element satisfies a masking condition, this setting can **override and cancel the mask**.

Cond	Description										
[5:4]	Condition: Src > Low and Src > High										
	<table><tr><th>Value</th><th>Description</th></tr><tr><td>0</td><td>Do not Mask</td></tr><tr><td>1</td><td>Mask</td></tr><tr><td>2</td><td>Do not mask; also unmask other elements</td></tr><tr><td>3</td><td>Mask; also unmask other elements</td></tr></table>	Value	Description	0	Do not Mask	1	Mask	2	Do not mask; also unmask other elements	3	Mask; also unmask other elements
	Value	Description									
	0	Do not Mask									
	1	Mask									
	2	Do not mask; also unmask other elements									
3	Mask; also unmask other elements										
[3:2]	Condition: <ul style="list-style-type: none">• Src > Low and Src ≤ High• Src > High and Src ≤ Low <i>(Uses the same value definitions as Cond[5:4])</i>										
[1:0]	Condition: Src ≤ Low and Src ≤ High <i>(Uses the same value definitions as Cond[5:4])</i>										

Notes:

- **Src** is the Source element selected by PixelCntl¥*.SrcASel
- **Low** is the lower comparison value selected by PixelCntl¥*.KeyLowSel
- **High** is the upper comparison value selected by PixelCntl¥*.KeyHighSel

5.3.1.6. PixelKeyLow Command

[Address: 0x28]

31			28				24				20				16				12				8					4				0
A[7:0]								R[7:0]								G[7:0]								B[7:0]								

Name	Description
A,R,G,B	Sets the lower fixed value for region control in the Extractor.

5.3.1.7. PixelKeyHigh Command

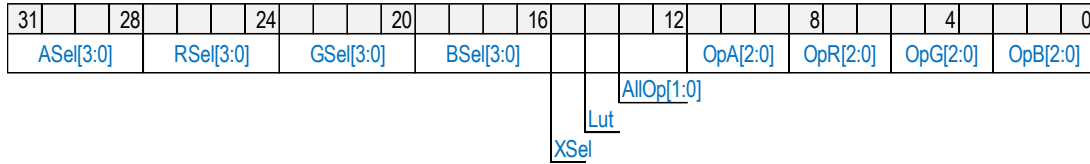
[Address: 0x2c]

31		28				24				20				16					12					8					4				0
A[7:0]						R[7:0]						G[7:0]						B[7:0]															

Name	Description
A,R,G,B	Sets the upper fixed value for region control in the Extractor.

5.3.1.8. PixelOrg Command

[Address: 0x30]



Name	Description
A,R,G,BSel	In the Envelope processing, selects the OrgSel for each element individually (refer to Figure 24).

Sel	Description
0	0xff(1.0)
1	Specified element of Context Data 1
2	Specified element of Context Data 2
3	Specified element of Context Data 3
4	PixelConst.B
5	PixelConst.G
6	PixelConst.R
7	PixelConst.A
8	Element B of SrcMod
9	Element G of SrcMod
10	Element R of SrcMod
11	Element A of SrcMod
12	Element B of SrcExt
13	Element G of SrcExt
14	Element R of SrcExt
15	Element A of SrcExt

XSel

In Envelope processing, selects the source for **OrgSelX**:

- '0': Non-filter processing data

- '1': *SrcOut*
Applies to all elements.

Lut

Applies the **3D Clut** to the *SrcOrg* system after Envelope processing.
Requires enabling via *ClutCntl.En*.

AllOp

Specifies the operation to be applied after the result of Envelope processing.

AllOp	Description
0	NOP
1	Performs absolute value conversion. Applies to all elements.
2	Clamps values ≤ 0 to 0. Applies to all elements.
3	Performs cumulative addition for each fragment in the X direction.

OpA,R,G,B

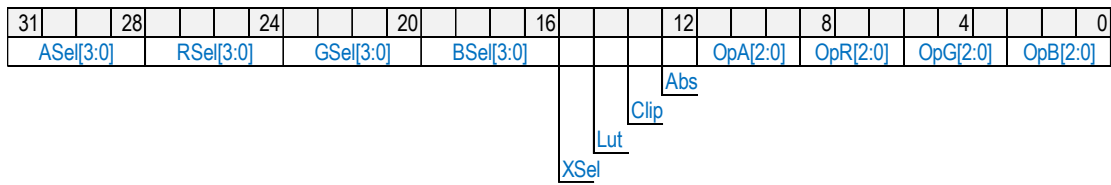
Selects the per-element operation between **OrgSel** and **OrgSelX**.

When **Op** = '6' or '7', the minimum or maximum value is selected for each element of **OrgSelX**, respectively.

Op	Description
0	$\text{OrgSelX} * \text{OrgSel}$
1	OrgSel
2	$\min(\text{OrgSelX}, \text{OrgSel})$
3	$\max(\text{OrgSelX}, \text{OrgSel})$
4	$\text{OrgSelX} + \text{OrgSel}$
5	$\text{OrgSelX} - \text{OrgSel}$
6	$\min(\text{OrgSelX}'\text{'s elements})$
7	$\max(\text{OrgSelX}'\text{'s elements})$

5.3.1.9. PixelMod Command

[Address: 0x34]



Name	Description
A,R,G,BSel	In the Envelope processing, selects the ModSel for each element individually (refer to Figure 24).

Sel	Description
0	0xff(1.0)
1	Specified element of Context Data 1
2	Specified element of Context Data 2
3	Specified element of Context Data 3
4	PixelConst.B
5	PixelConst.G
6	PixelConst.R
7	PixelConst.A
8	Element B of SrcOrg
9	Element G of SrcOrg
10	Element R of SrcOrg
11	Element A of SrcOrg
12	Element B of SrcOut
13	Element G of SrcOut
14	Element R of SrcOut
15	Element A of SrcOut

XSel

In Envelope processing, selects **ModSelX**:

- '0': Non-filter processing data
- '1': *SrcOut*
Applies to all elements.

Lut

Uses **3D Clut** on the *SrcOrg* system after Envelope processing.
ClutCntl.En must also be enabled.

Clamp

Clamps Envelope processing results ≤ 0 to 0. Applies to all elements.

This is applied **after** the **Abs** (absolute value) operation.

If a negative value becomes positive through Abs, that positive value is used.

Abs

Applies absolute value transformation to the result of Envelope processing.

Applies to all elements.

OpA, R, G, B

Selects the operation between **ModSel** and **ModSelX** for each element.

When **Op** = '6' or '7', selects the **minimum** or **maximum** value from *ModSelX* for each element, respectively.

Op	Description
0	ModSelX * ModSel
1	ModSel
2	min(ModSelX , ModSel)
3	max(ModSelX , ModSel)
4	ModSelX + ModSel
5	ModSelX - ModSel
6	min(ModSelX ' s elements)
7	max(ModSelX ' s elements)

5.3.1.10. PixelDefault Command

[Address: 0x38]

31		28			24				20				16				12				8				4				0
A[7:0]					R[7:0]					G[7:0]					B[7:0]														
Name					Description																								
A,R,G,B					Sets the value for pixels that exceed various boundaries.																								

5.3.1.11. PixelConst Command

[Address: 0x3c]

31			28			24			20			16			12			8			4			0
A[7:0]						R[7:0]						G[7:0]						B[7:0]						
Name						Description																		

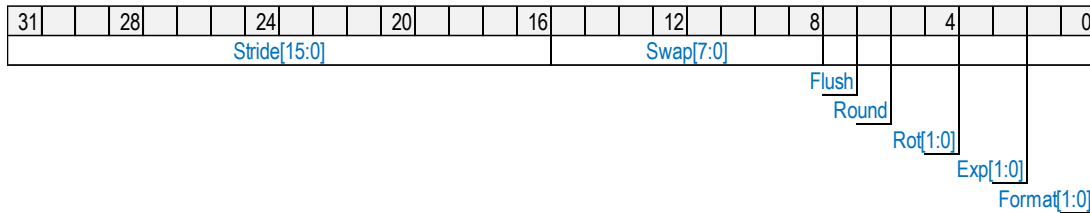
A, R, G, B

Referenced by both the **Blender** and **Envelope** blocks.

In the **Blender**, it is used as the multiplicand in multiplication when selected by **PixelCntl*.DstBSel** or **PixelCntl*.SrcBSel**.

5.3.1.12. SrcInInfo Command

[Address: 0x40]



Name	Description
Stride	<p>In the Pixel Cache, set the address update width for the referenced Source input data as “update width – 1”.</p> <p>The unit depends on the Format.</p> <p>For 1 Bpp, the setting must be specified in 32 Bpp units.</p> <p>For example, if updating in the Y-direction requires a 64-bit move, the setting value should be '1' (i.e., 64 / 32 – 1).</p>

Swap

In the **Pixel Cache**, configure the **Byte Swap** for the referenced Source input data.

This defines the byte-level mapping from input data **In[31:0]** to internal data **Pipe[31:0]**.

⚠ Ensure a **1-to-1 mapping** is used.
Incorrect settings can result in **undefined values** or **overlapping (aliasing)**, so caution is required.

Value	Swap[7:6]	Swap[5:4]	Swap[3:2]	Swap[1:0]
	Pipe[31:24]	Pipe[23:16]	Pipe[15:8]	Pipe[7:0]
0	In[31:24]	In[23:16]	In[15:8]	In[7:0]
1	In[7:0]	In[31:24]	In[23:16]	In[15:8]
2	In[15:8]	In[7:0]	In[31:24]	In[23:16]
3	In[23:16]	In[15:8]	In[7:0]	In[31:24]

Flush

In the **Pixel Cache**, this setting initializes the cache for the referenced Source input data.

It is used to **clear the cache** when there is a possibility that the cache data may be updated.


As an external signal operation, it performs an **empty write access** on the memory bus (*mrR_{xw}* signal = '0').

 **Do not use** this setting if:

- External cache is not used
- External cache does not support a flush function
- You are using **Version A, B, or C**, where this function is **disabled**.

Round

Performs rounding of the fractional part of the X and Y coordinates input to the **Pixel Cache**.

 **Do not enable this setting** when using **Bi-linear** or **Bi-cubic** interpolation.

Rot

In the **Pixel Cache**, configure the detailed **pixel format** of the referenced Source input data.

(For more information, refer to the **Format** specification.)

Exp

In the **Pixel Cache**, configure the detailed **pixel format** for the referenced Source input data.

Refer to the **Format** documentation for specific details on format definitions and settings.

Format

In the **Pixel Cache**, configure the **Bits per Pixel (Bpp)** for the referenced Source input data.

If using **1 Bpp**, additional configuration for the **Bitmap Filter** is required.

Format	Exp	Pipe [31:24]	Pipe [23:16]	Pipe [15:8]	Pipe [7:0]	Note
0 8Bpp	0	0	0	0	In[7:0]	
	1	In[7:0]	In[7:0]	In[7:0]	In[7:0]	8bit Replica
0 1Bpp	2	Internal Special (MSB First)				MSB First Only at Ver.A
	3	Internal Special (LSB First)				
1 16Bpp	0	Gray	In [15:11] [15:13]	In [10:5] [10:9]	In [4:0] [4:2]	RGB565 Lower Replica
	1	In [15:8]	In[7:0]	In[7:0]	In [7:0]	Rot=' 0'
			In[7:0]	In[15:8]		Rot=' 1'
			In[15:8]	In[7:0]		Rot=' 2'
			In[15:8]	In[15:8]		Rot=' 3'
2	0xff	In [31:24] /In [15:8]	In [23:16]	In [7:0]	Alpha=1.0 YUYV	
3	In[31:16] >> X		In[15:0] >> X		X={Signed, Rot} force Signed to '1'	
2 24Bpp	0	In [23:16]	In [23:16]	In [15:8]	In [7:0]	
	1	Gray	In [23:16]	In [15:8]	In [7:0]	
	2	0xff	In [23:16]	In [15:8]	In [7:0]	Alpha=1.0
	3	Gray	Gray	Gray	Gray	All Gray
3 32Bpp	0	In [31:24]	In [23:16]	In [15:8]	In [7:0]	Rot[0]=' 0'
		8Bpp X[1:0]= 0	8Bpp X[1:0]= 1	8Bpp X[1:0]= 2	8Bpp X[1:0]= 3	Rot[0]=' 1'
	1	Gray	In [23:16]	In [15:8]	In [7:0]	
	2	0xff	In [23:16]	In [15:8]	In [7:0]	Alpha=1.0
	3	Gray	Gray	Gray	Gray	All Gray

In: Memory side

Pipe: Blender side (=ARGB)

Gray: $(2 \text{ In}[23:16] + 5 \text{ In}[15:8] + \text{In}[7:0]) / 8$

FI: $\text{ftoi}(\text{In}[15:0])$

5.3.1.13. SrcInBase Command

[Address: 0x44]

31			28				24				20				16				12				8				4			0					
Base[31:6]																								Wrap[5:0]											

Name	Description
Base	<p>In the Pixel Cache, configure the base address for the referenced Source input data.</p> <p>The address must be aligned to a 64-byte boundary.</p>
Wrap	<p>By setting the MSB to '1', a mask is applied using Wrap[4:1] (4 bits).</p> <p>The mask value applied to the 32-bit address is: 0x007FFFFFFF >> ~Wrap[4:1]</p> <p>Additionally, Wrap[0] (1 bit) is sent as the LSB of the address to the memory system.</p> <p>If the MSB is '0', no masking is applied, and Wrap[1:0] (2 bits) are sent as the LSB of the address to the memory system as informational bits.</p>

5.3.1.14. SrcOutInfo Command

[Address: 0x48]

31			28				24				20				16				12				8				4			0	
Stride[15:0]																Swap[7:0]								Rot[1:0]							
																				Flush				Exp[1:0]							
																				Round				Format[1:0]							

Name	Description
Stride	Sets the update interval (minus 1) for the rectangular Envelope pattern applied to the Source data or the address of the Texture data. (See SrcInInfo.Stride for details.)
Swap	Sets the byte swap for the rectangular Envelope pattern data applied to the Source data or the Texture data. (See SrcInInfo.Swap for details.)
Flush	Initializes the cache for the rectangular Envelope pattern data applied to the Source data or the Texture data. (See SrcInInfo.Flush for details.)
Round	Rounds the fractional part of the X and Y coordinates input to

the Pixel Cache. Do not enable this when using Bi-linear or Bi-cubic interpolation.

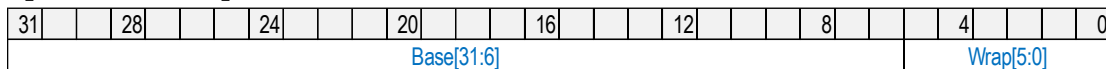
Rot Specifies the detailed pixel format of the Source input data referenced by the Pixel Cache. (See Format for details.)

Exp Specifies the detailed pixel format of the rectangular region or Texture data applied to the Source data. (See SrcInInfo.Exp for details.)

Format Specifies the bits per pixel (Bpp) format for the rectangular region or Texture data applied to the Source data. (See SrcInInfo.Format for details.)

5.3.1.15. SrcOutBase Command

[Address: 0x4c]

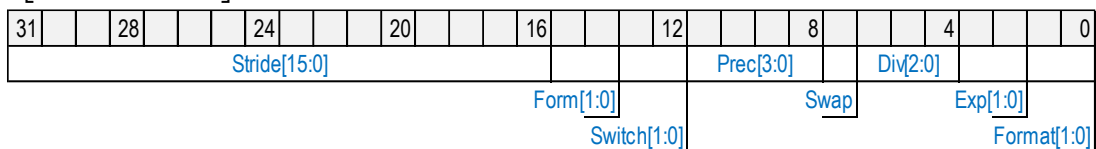


Name	Description
Base	Sets the base address of the rectangular Envelope pattern applied to the Source data or the Texture data. Must be set in 64-byte boundary units. (See SrcInBase.Base for details.)

Wrap See SrcInBase.Wrap for details.

5.3.1.16. SrcMapInfo Command

[Address: 0x50]



Name	Description
Stride	In the Source Remapper, sets the address update interval (minus 1) for the referenced source map data. (See SrcInInfo.Stride for details.)

Form In the Source Remapper, defines the data transformation format. Input data must match this format. Internally, data is temporarily converted to single-precision floating-point for

processing. If using floating-point format, set Div = 0.

From	Description
0	ItoF (Integer Input)
1	Reserved
2	HFtoF (Half-Precision Floating-Point Input)
3	FtoF (Single-Precision Floating-Point Input)

Switch

Selects the reference coordinates and base coordinates in the Source Remapper.

Switch[0]	Description
0	The reference coordinates are those generated by MasterCntl.Shape.
1	The reference coordinates are those input to frComp.

Switch[1]	Description
0	The base coordinates are those generated by MasterCntl.Shape.
1	The base coordinates are those input to frComp.

Prec

In the Source Remapper, if MasterCntl.SrcRemap is not '0', the fractional position of the referenced coordinates is specified as a two's complement value relative to the LSB. If MasterCntl.SrcRemap is '0', processing is performed on the parametric coordinates. Since only 4 bits of the fractional part are valid for use in subsequent matrix transformations, any excess bits will be truncated.

For non-Texture cases (Exp[1] = '0'):

Prec	Description
0	Multiplies the 16-bit output coordinates by 1 (x1).
1	Divides the 16-bit output coordinates by 2 (x1/2).
2	Divides the 16-bit output coordinates by 4 (x1/4).
3	Divides the 16-bit output coordinates by 8 (x1/8).

4	Divides the 16-bit output coordinates by 16 (x1/16).
5	Divides the 16-bit output coordinates by 32 (x1/32).
6	Divides the 16-bit output coordinates by 64 (x1/64).
7	Divides the 16-bit output coordinates by 128 (x1/128).
8	Disables the escape value (0x8000).
9	Multiplies the 16-bit output coordinates by 128 (x128)
10	Multiplies the 16-bit output coordinates by 64 (x64)
11	Multiplies the 16-bit output coordinates by 32 (x32)
12	Multiplies the 16-bit output coordinates by 16 (x16)
13	Multiplies the 16-bit output coordinates by 8 (x8)
14	Multiplies the 16-bit output coordinates by 4 (x4)
15	Multiplies the 16-bit output coordinates by 2 (x2)

Note: From Ver.C onward, rounding is applied when Format[0] = '0' (Nearest Neighbor).

For Texture mode (Exp[1] = '1'):

Prec[2:0]	Description
0	Uses 1-bit input coordinates Y000, X000 as XY
1	Uses 2-bit input coordinates Y1:01:01:0, X1:01:01:0 as XY
2	Uses 3-bit input coordinates Y2:02:02:0, X2:02:02:0 as XY
3	Uses 4-bit input coordinates Y3:03:03:0, X3:03:03:0 as XY
4	Uses 5-bit input coordinates Y4:04:04:0, X4:04:04:0 as XY
5	Uses 6-bit input coordinates Y5:05:05:0, X5:05:05:0 as XY
6	Uses 7-bit input coordinates Y6:06:06:0, X6:06:06:0 as XY
7	Uses 8-bit input coordinates Y7:07:07:0, X7:07:07:0 as XY

Note: If Prec[3] = '1', the escape value (0x8000) is disabled.

Swap

In the Source Remapper, sets the Half Word Swap for the referenced data.

Swap	Pipe[31:16]	Pipe[15:0]
0	In[31:16]	In[15:0]
1	In[15:0]	In[31:16]

Div

In the Source Remapper, specifies the coordinate sampling interval (2^{Div}) .

If set to 0, the input and output coordinates correspond one-to-one.

If set to a non-zero value, 2^{Div} output coordinates are generated for each input coordinate. When selecting $\text{Div} \neq 0$, Form must be set to 0.

Exp

In the Source Remapper, specifies the detailed pixel format of the referenced source map data. (See Format for details.)

Format

In the Source Remapper, specifies the bits per pixel (Bpp) format of the referenced source map data.

Format	Exp	Xo	Yo	Note
0	0	mem ₃₂ (Xi, Yi) Nearest		SrcOffset Offset
	1			SrcOffset Mask
	2	Xi, Yi	mem ₈ (Xi, Yi) Nearest	SrcOffset Offset
	3			SrcOffset Mask
1	0	mem ₃₂ (Xi, Yi) Bi-linear		SrcOffset Offset
	1			SrcOffset Mask
	2	Xi, Yi	mem ₁₆ (Xi, Yi) Nearest	SrcOffset Offset
	3			SrcOffset Mask
2*	0	mem ₃₂ (2Xi, Yi)	mem ₃₂ (2Xi+1, Yi)	32bit Deta Packing
	1			
	2	Xi, Yi	mem ₃₂ (Xi, Yi) [23:16][7:0] Nearest	SrcOffset Offset
	3			SrcOffset Mask
3*	0	mem ₃₂ (Xi, 2Yi)	mem ₃₂ (Xi, 2Yi+1)	32bit Deta Planar (Interleaved by Stride)
	1			
	2	Xi, Yi	mem ₃₂ (Xi, Yi) Gray 変換 Nearest	SrcOffset Offset
	3			SrcOffset Mask

Xi, Yi are input coordinates.

Xo, Yo are output coordinates, with the fractional bit

position selected by **Prec**.

memn() indicates memory data access with an n -bit word length.

When **Form** = 3, precision correction must be disabled. Set both *DisableInPrec* and *DisableOutPrec* in the Utility register to 1.

5.3.1.17. SrcMapBase Command

[Address: 0x54]

31			28				24				20				16				12				8				4			0					
Base[31:6]																								Wrap[5:0]											

Name	Description
Base	In the Source Remapper, sets the base address of the referenced source map data. (See SrcInBase.Base for details.)
Wrap	See SrcInBase.Wrap for details.

5.3.1.18. SrcSize Command

[Address: 0x58]

31		28			24			20			16			12			8			4			0
WidthY[15:0]												WidthX[15:0]											

Name	Description
WidthY,X	<p>In the Pixel Cache (Source In), the reference range is specified in pixel units.</p> <p>Used for edge copying and color replacement judgment when crossing boundaries.</p> <p>Specifies the image size (e.g., 640 × 480 for VGA).</p> <p>A value of 0 represents infinity (no boundary check is performed).</p>

5.3.1.19. SrcOffset Command

[Address: 0x5c]

31			28				24				20				16				12				8				4			0	
OffsetY[15:0]																OffsetX[15:0]															
BoxY[3:0]				MaskY[3:0]				CoorY1[3:0]				CoorY0[3:0]				BoxX[3:0]				MaskX[3:0]				CoorX1[3:0]				CoorX0[3:0]			

Name	Description
OffsetY	If MasterCntl.SrcScan[1] is '0', sets an offset or mask on the Y coordinate of the referenced SrcIn in the Source Remapper. Offsets are expressed in two's complement.

OffsetX If MasterCntl.SrcScan[0] is '0', sets an offset or mask on the X coordinate of the referenced SrcIn in the Source Remapper. Offsets are expressed in two's complement.

Box*, Mask*, Coord* If MasterCntl.SrcScan is '1' (with MSB as Y', LSB as X'), modifies the XY coordinates of the referenced SrcIn in the Source Remapper using the following methods.

Coord*[2:0]	CoordX0	CoordY0	CoordX1	CoordY1
	U0	V0	U1	V1
0	X	Y	0	0
1	0			
2				
3				
4	X			
5	Y			
6	Z			
7	W			

CoordX1*[3],	CoordX0*[3]	X'
0	0	$U0 \% 2^{16-MaskX} + U1 * 2^{BoxX}$
0	1	$U0 / 2^{MaskX} + U1 * 2^{BoxX}$
1	0	$U0 \% 2^{16-MaskX} + U1 / 2^{BoxX}$
1	1	$U0 \% 2^{BoxX} + U1 / 2^{BoxY} * 2^{BoxX}$

CoordY1*[3],	CoordY0*[3]	Y'
0	0	$V0 \% 2^{16-MaskY} + V1 * 2^{BoxY}$
0	1	$V0 / 2^{MaskY} + V1 * 2^{BoxY}$
1	0	$V0 \% 2^{16-MaskY} + V1 / 2^{BoxY}$
1	1	$V0 / 2^{BoxX} \% 2^{16-MaskX} + V1 / 2^{BoxY} \% 2^{16-MaskY} * 2^{MaskX}$

5.3.1.20. DstInInfo Command

[Address: 0x60]

31		28		24		20		16		12		8		4		0
Stride[15:0]								Swap[7:0]				Rdc[2:0]		Exp[1:0]		Format[1:0]

Name	Description
Stride	In the Blender, sets the address update interval (update width – 1) for the referenced Destination input data. (See SrcInInfo.Stride for details.) Note: 0xffff is treated as 0.
Swap	In the Blender, sets the byte swap for the referenced Destination input data. (See SrcInInfo.Swap for details.)
Rdc	In the Blender, sets the pixel value shift amount ($\times 2^{-n}$) for the referenced Destination input data.
Exp	In the Blender, specifies the detailed pixel format of the referenced Destination input data. (See Format for details.)
Format	In the Blender, sets the bits per pixel (Bpp) format for the referenced Destination input data. Must be the same format as DstOut.

Format	Exp	Pipe [31:24]	Pipe [23:16]	Pipe [15:8]	Pipe [7:0]	Note
0 8Bpp	0	In[7:0]				8bit Replica
	1	Unkown				Reserved
	2	In[7:0]				8bit Replica Sign Extention 2's complement
	3	Unkown				Reserved
1 16Bpp	0	Gray	In [15:11] [15:13]	In [10:5] [10:9]	In [4:0] [4:2]	RGB565 Lower Replica
	1	In [15:8]	In[7:0]	In[7:0]	In[7:0]	Rdc[1:0]=' 0'
			In[7:0]	In[15:8]	In[7:0]	Rdc[1:0]=' 1'
			In[15:8]	In[7:0]	In[7:0]	Rdc[1:0]=' 2'
			In[15:8]	In[15:8]	In[7:0]	Rdc[1:0]=' 3'
	2	0xff	In [31:24] /In [15:8]	In [23:16]	In [7:0]	Alpha=1.0 YUYV
	3	In[15:0] >>Rdc if [15] then 0		In[15:0] >>Rdc if [15] then 0		Reduce Rdc ≠ 0
		In [15:8]	In [7:0]	In [7:0]	In [7:0]	Pass Rdc=0

2 24Bpp	0	In [23:16]	In [23:16]	In [15:8]	In [7:0]	
	1	Gray	In [23:16]	In [15:8]	In [7:0]	
	2	0xff	In [23:16]	In [15:8]	In [7:0]	Alpha=1.0
	3	Gray	Gray	Gray	Gray	All Gray
3 32Bpp	0	In [31:24]	In [23:16]	In [15:8]	In [7:0]	Pass
	1	Gray	In [23:16]	In [15:8]	In [7:0]	
	2	In [31:24]	In [23:16]	In [15:8]	In [7:0]	Sign Extension 2's complement
	3	Gray	Gray	Gray	Gray	All Gray

In: Memory side
Pipe: Blender side (=ARGB)

5.3.1.21. DstInBase Command

[Address: 0x64]

31		28			24				20				16				12				8				4			0
Base[31:6]																							Wrap[5:0]					

Name	Description
Base	In the Blender, sets the base address of the referenced Destination input data. (See SrcInBase.Base for details.)
Wrap	<p>If the MSB is set to '1', the 4-bit field Wrap[4:1] specifies a mask. The mask value applied to the 32-bit address is calculated as $0x007FFFFF \gg \sim \text{Wrap}[4:1]$. Additionally, the LSB of the address sends Wrap[0] (1 bit) as information to the memory system.</p> <p>If the MSB is '0', the Y coordinate is masked within the Blender to implement ring buffer processing. The mask range is set by Wrap[4:2]. If this range is '0', no masking is applied. In this case, Wrap[1:0] (2 bits) are sent as LSB information to the memory system.</p>

5.3.1.22. DstOutInfo Command

[Address: 0x68]

31			28				24				20				16				12				8				4			0		
Stride[15:0]																Swap[7:0]									Rdc[2:0]							
																										Exp[1:0]						Format[1:0]

Name	Description
Stride	In the Blender, sets the address update interval (update width – 1) for the Destination output data. (See SrcInInfo.Stride for details.) Note: 0xffff is treated as 0.
Swap	In the Blender, sets the byte swap for the Destination output data. This defines the byte-wise mapping from internal data Pipe[31:0] to output data Out[31:0]. If not configured as a one-to-one mapping, it may result in unknown values or data overlap.

Value	Swap[7:6]	Swap[5:4]	Swap[3:2]	Swap[1:0]
	Out[31:24]	Out[23:16]	Out[15:8]	Out[7:0]
0	Pipe[31:24]	Pipe[23:16]	Pipe[15:8]	Pipe[7:0]
1	Pipe[23:16]	Pipe[15:8]	Pipe[7:0]	Pipe[31:24]
2	Pipe[15:8]	Pipe[7:0]	Pipe[31:24]	Pipe[23:16]
3	Pipe[7:0]	Pipe[31:24]	Pipe[23:16]	Pipe[15:8]

Rdc	In the Blender, sets the reduction ratio of the pixel format for the referenced Destination output data.
Exp	In the Blender, specifies the detailed pixel format of the referenced Destination output data. (See Format for details.)
Format	In the Blender, sets the bits per pixel (Bpp) format of the referenced Destination output data (partially differs from DstInInfo.Format). For 16 Bpp YUYV format: <ul style="list-style-type: none"> The R component is output in the upper 8 bits. The G and B components are alternately output to the lower 8 bits depending on whether the output horizontal pixel position is even or odd.

Format	Exp	Out [31:24]	Out [23:16]	Out [15:8]	Out [7:0]	Note
0 8Bpp	0	Pipe[7:0]				A[7:0] & ~Dis _A R[7:0] & ~Dis _R G[7:0] & ~Dis _G B[7:0] & ~Dis _B
	1	Reserved				

	2	Pipe[7:0]		A[7:0] & ~Dis _A R[7:0] & ~Dis _R G[7:0] & ~Dis _G B[7:0] & ~Dis _B Sign Extension 2's complement
	3	Unknown		Pipe [15:0] >> Rdc Pass & Reduce Clip if negative and overflow
1 16Bpp	0	Unknown	Pipe [23:19] [15:10] [7:3]	RGB565 Lower Cut
	1	Unknown	Pipe[15:0]	
	2	Unknown	Pipe [23:16] /Pipe [7:0]	YUYV Upper is even Lower is odd
	3	Unknown	Pipe [31:16] >> Rdc	Reduce Rdc ≠ 0 Clip if negative and overflow
			Pipe[15:0]	Pass Rdc=0
2 24Bpp	0	Unknown	Pipe[23:0]	Pass
	1	Unknown		Reserved
	2			
	3	Unknown	Pipe[31:0] >> Rdc	Pass & Reduce
3 32Bpp	0	Pipe[31:0]		Pass
	1	0xff	Pipe[23:0]	A=0xff
	2	Pipe[31:0]		Sign Extension 2's complement
	3	Pipe[31:0] >> Rdc		Pass & Reduce Sign is expanded

Out: Memory side
Pipe: Blender side (=ARGB)

5.3.1.23. DstOutBase Command

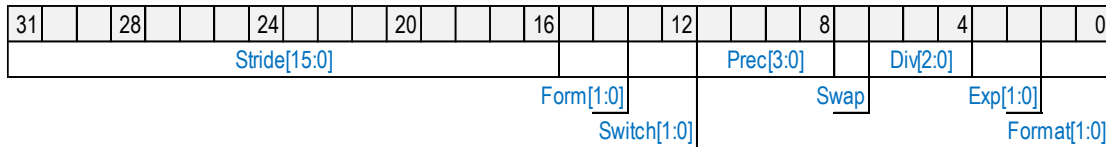
[Address: 0x6c]

31		28		24		20		16		12		8		4		0
Base[31:6]														Wrap[5:0]		
Name		Description														

- Base** In the Blender, sets the base address of the referenced Destination output data. Must be set in 64-byte boundary units. (See SrcInBase.Base for details.)
- Wrap** See DstInBase.Wrap for details.

5.3.1.24. DstMapInfo Command

[Address: 0x70]



Name	Description
Stride	In the Destination Remapper, sets the address update interval (update width – 1) for the referenced Destination map data. (See SrcMapInfo.Stride for details.)
Form	In the Destination Remapper, defines the data transformation format. Input data must conform to this format. Internally, data is temporarily converted to single-precision floating-point for processing. (See SrcMapInfo.Form for details.)
Switch	In the Source Remapper, selects the reference and base coordinates. (See SrcMapInfo.Switch for details.)
Prec	In the Destination Remapper, if MasterCntl.DstRemap is '1', the fractional position of the referenced coordinates is specified as a two's complement value relative to the LSB. (See SrcMapInfo.Prec for details.)
Swap	In the Destination Remapper, sets the Half Word Swap for the referenced data. (See SrcMapInfo.Swap for details.)
Div	In the Destination Remapper, specifies the coordinate sampling interval (2^{Div}). (See SrcMapInfo.Div for details.)

Exp	In the Destination Remapper, sets how the referenced Destination map data is handled. (See SrcMapInfo.Exp for details.)
Format	In the Destination Remapper, sets the bits per pixel (Bpp) format of the referenced Source map data. (See SrcMapInfo.Format for details.)

5.3.1.25. DstMapBase Command

[Address: 0x74]

31			28				24				20				16				12				8				4			0	
Base[31:6]																										Wrap[5:0]					

Name	Description
------	-------------

Base	In the Destination Remapper, sets the base address of the referenced Destination map data. (See SrcMapBase.Base for details.)
-------------	---

Wrap	See SrcMapBase.Wrap for details.
-------------	----------------------------------

5.3.1.26. DstSize Command

[Address: 0x78]

31			28				24				20				16				12				8				4			0	
WidthY[15:0]																WidthX[15:0]															

Name	Description
------	-------------

WidthY, WidthX	<p>Specifies the reference range in Source Out or the write range in the Blender in pixel units.</p> <ul style="list-style-type: none"> When MasterCntl.DstOp = '0' (Source Out read), it is used to determine edge copying and color replacement when crossing boundaries. Specifies the image size (e.g., 640 × 480 for VGA). When MasterCntl.DstOp = '1' (Blender write), pixels that go beyond the boundary are masked. A value of 0 represents infinity (no boundary check is performed).
-----------------------	---

5.3.1.27. DstOffset Command

[Address: 0x7c]

31			28			24			20			16			12			8			4			0
OffsetY[15:0]												OffsetX[15:0]												
BoxY[3:0]			MaskY[3:0]			CoorY1[3:0]			CoorY0[3:0]			BoxX[3:0]			MaskX[3:0]			CoorX1[3:0]			CoorX0[3:0]			

Name	Description
------	-------------

OffsetY	If MasterCntl.SrcScan[1] is '0', sets an offset on the Y coordinate of the referenced Destination in the Destination Remapper. The value is expressed in two's complement.
----------------	--

OffsetX	If MasterCntl.SrcScan[0] is '0', sets an offset on the X coordinate of the referenced Destination in the Destination Remapper. The value is expressed in two's complement.
----------------	--

Box*, Mask*, Coor*	If MasterCntl.DstScan is '1' (MSB = Y', LSB = X'), modifies the XY coordinates of the referenced SrcOut in the Destination Remapper using the following methods.
---------------------------	--

Coor*[2:0]	CoorX0	CoorY0	CoorX1	CoorY1
	U0	V0	U1	V1
0	X	Y	0	0
1	0			
2				
3				
4	X			
5	Y			
6	Z			
7	W			

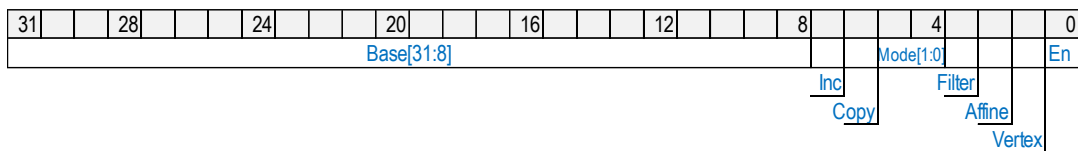
Coor X1* [3],	Coor X0* [3]	X'
0	0	$U0 \% 2^{16-MaskX} + U1 * 2^{BoxX}$
0	1	$U0 / 2^{MaskX} + U1 * 2^{BoxX}$

1	0	$U0 \% 2^{16-MaskX} + U1 / 2^{BoxX}$
1	1	$U0 \% 2^{BoxX} + U1 / 2^{BoxY} * 2^{BoxX}$

Coor Y1* [3],	Coor Y0* [3]	Y'
0	0	$V0 \% 2^{16-MaskY} + V1 * 2^{BoxY}$
0	1	$V0 / 2^{MaskY} + V1 * 2^{BoxY}$
1	0	$V0 \% 2^{16-MaskY} + V1 / 2^{BoxY}$
1	1	$V0 / 2^{BoxX} \% 2^{16-MaskX} + V1 / 2^{BoxY} \% 2^{16-MaskY} * 2^{MaskX}$

5.3.1.28. CICntl Command

[Address: 0x80]



Name	Description
Base	Sets the start address for storing the 32-byte input context. Must be set in 256-byte boundary units.

Address + n	Group	Description
0	Header	Number of operation (when エラー! 参照元が見つかりません。_is_enabled)
1		Total value (when エラー! 参照元が見つかりません。_is_enabled)
2		Minimum value (when エラー! 参照元が見つかりません。_is_enabled)

3		Maximum value (when エラー! 参照元が見つかりません。 is enabled)
4	Vertex	Vertex0 (when Vertex is enabled)
5		Vertex1 (when Vertex is enabled)
6		Vertex2 (when Vertex is enabled)
7	Affine	Affin Coef0 (when Affine is enabled)
8		Affin Coef1 (when Affine is enabled)
9		Affin Coef2 (when Affine is enabled)
10		Affin Coef3 (when Affine is enabled)
11		Affin Coef4 (when Affine is enabled)
12		Affin Coef5 (when Affine is enabled)
13		Affin Coef6 (when Affine is enabled)
14		Affin Coef7 (when Affine is enabled)
15		Affin Coef8 (when Affine is enabled)

16	Filter	Reserved
17		Reserved
18		FilterCoef00 (when Filter is enabled)
19		FilterCoef01 (when Filter is enabled)
20		FilterCoef10 (when Filter is enabled)
21		FilterCoef11 (when Filter is enabled)
22		FilterCoef12 (when Filter is enabled)
23		FilterCoef13 (when Filter is enabled)
24		FilterCoef20 (when Filter is enabled)
25		FilterCoef21 (when Filter is enabled)
26		FilterCoef22 (when Filter is enabled)
27		FilterCoef23 (when Filter is enabled)
28		FilterCoef24 (when Filter is enabled)
29		FilterCoef25 (when Filter is enabled)

30		FilterCoef26 (when Filter is enabled)
31		FilterCoef27 (when Filter is enabled)

Inc When set to '1', adds the product of the iIndex[31:16] signal (Y index) and the number of entries per flag (see table below) to the Base. This increments the context per Y.
Even if the increment value is large, Groups whose flags are not set will not be read.
(For example, if only the Filter flag is set, the increment amount is 32, but the Header, Vertex, and Affine contexts will use the original Command values.)

Filter	Affine	Vertex	En	Group	Description
0	0	0	0	–	–
0	0	0	1	Header	4
0	0	1	0	Header +Vertex	8
0	0	1	1		
0	1	0	0	Header +Vertex +Affine	16
0	1	0	1		
0	1	1	0		
0	1	1	1		
1	0	0	0	Header +Vertex +Affine +Filter	32
1	0	0	1		
1	0	1	0		
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		

Copy When set to '1', copies the number of operations, total value, minimum value, and maximum value from the input context to the output context.

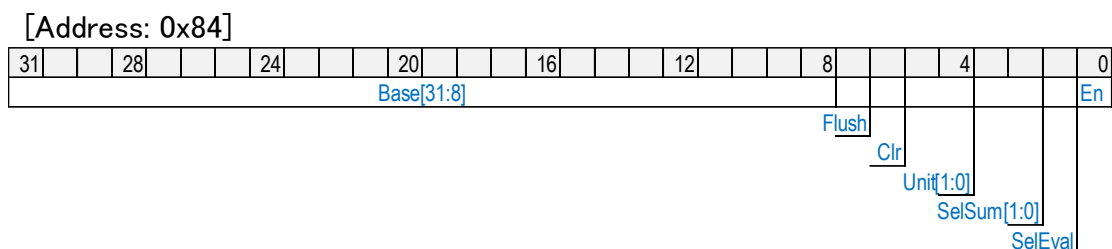
Mode Specifies the loading of a specific context.
If Mode[1] is '1', only the context specified by CiCntl[4:0] is targeted.
CiCntl[4:0] specifies the context number only; the following functions

do not apply.

In this mode, the offset applied by Inc is fixed at 4.

Filter	When set to '1', loads the 2D filter coefficients. This takes precedence over the Command List settings.
Affine	When set to '1', loads the matrix transformation coefficients. This takes precedence over the Command List settings.
Vertex	When set to '1', loads the vertex coordinates. This takes precedence over the Command List settings.
En	When set to '1', loads the number of operations, total value, minimum value, and maximum value from the input context.

5.3.1.29. COCntl Command



Name	Description
Base	Sets the start address for storing the 32-byte output context. Must be set in 256 byte boundary units.
Flus	In the output context, clears the cache information and reloads the data.
Clr	In the output context, clears the content to zero at the timing of the first fragmentation based on the unit specified by Unit. This takes precedence even if COCntl.Flush is '1'.
Unit	In the output context, sets the control unit. At the timing of the first fragmentation for the specified unit, the context contents can be cleared.

Unit	Description
0	Reserved
1	2D (starts when index X=0, Y=0)
2	3D (starts when index X=0, Y=0, Z=0)
3	4D (starts when index X=0, Y=0, Z=0, W=0)

SelSum

Specifies the element to be used for the total value stored in the output context.

SelSum	Description
0	Selects element B
1	Selects element G
2	Selects element R
3	Selects element A

SelEval

Specifies the data to be used for the total, maximum, and minimum values stored in the output context.

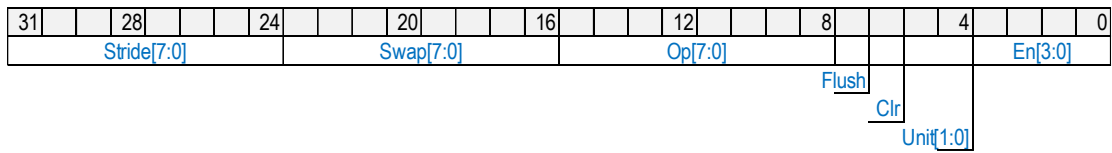
SelEval	Description
0	Selects DstOutData from Blender output
1	Selects DstOrgData from Blender output

En

When set to '1', enables writing to the output context.

5.3.1.30. HistCntl0 Command

[Address: 0x88]



Name	Description
------	-------------

Stride In the Histogram, when stacking aggregation units (Unit) in 2D, this sets the interval minus 1 between them.

For example, when aggregating a frame (indexed by X and Y) and arranging the results by index Z (horizontal axis) and index W (vertical axis), this sets the interval for when index W is incremented.

The actual address interval is calculated as:

(Number of enabled elements indicated by En) × (Index count indicated by HistCntl1.Num = 2^Num) × 4.

Swap In the Histogram, sets the index swap for each element. Selects which element of DstOutData (referred to as **Data** in the table below) from the Blender output to use.

	Swap[7:6]	Swap[5:4]	Swap[3:2]	Swap[1:0]
Value	Index[31:24]	Index[23:16]	Index[15:8]	Index[7:0]
0	Data[31:24]	Data[23:16]	Data[15:8]	Data[7:0]
1	Data[7:0]	Data[31:24]	Data[23:16]	Data[15:8]
2	Data[15:8]	Data[7:0]	Data[31:24]	Data[23:16]
3	Data[23:16]	Data[15:8]	Data[7:0]	Data[31:24]

Op In the Histogram, the histogram is updated based on the following calculation formula. Every 2 bits from the upper side correspond to the computation for each ARGB element.

Op[2n-1:2n]	Description
0	Hist[Index] = Hist[Index] + 1
1	Hist[Index] = Hist[Index] + DstOrg
2	Hist[DstOrg] = Hist[DstOrg] + 1
3	Hist[DstOrg] = Hist[DstOrg] + Index

Flush In the Histogram, clears the cache information of the referenced

histogram and reloads the data.

This is unnecessary if each process uses a unique address.

Clr

In the Histogram, clears the content to zero at the timing of the first fragmentation, as specified by the Unit.

Unit

n the Histogram, sets the aggregation unit.

Unit	Description
0	Reserved
1	2D (starts when index X = 0, Y = 0)
2	3D (starts when index X = 0, Y = 0, Z = 0)
3	4D (starts when index X = 0, Y = 0, Z = 0, W = 0)

En

In the Histogram, specifies whether to enable aggregation for each element. Set to '1' to enable.

This flag also determines which elements are written to memory.

If all are set to '0', no data will be written.

En	Description
[0]	Performs Histogram aggregation for element B.
[1]	Performs Histogram aggregation for element G.
[2]	Performs Histogram aggregation for element R.
[3]	Performs Histogram aggregation for element A.

5.3.1.31. HistCntl1 Command

[Address: 0x8c]

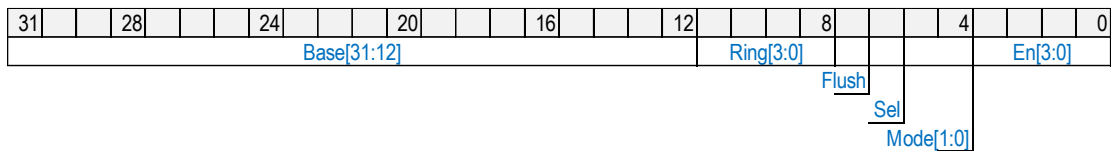
31		28		24		20		16		12		8		4		0
Base[31:8]												Num[3:0]			Integral[3:0]	

Name	Description
Base	In the Histogram, sets the start address for storing the result. Must be set in 256-byte boundary units.
Num	In the Histogram, sets the number of indices to be written (2^{Num}). Values of 8 or higher are treated as 8.
Integral	In the Histogram, specifies the elements to be output as cumulative (integrated) values.

Integral	Description
[0]	Accumulate Histogram for element B
[1]	Accumulate Histogram for element G
[2]	Accumulate Histogram for element R
[3]	Accumulate Histogram for element A

5.3.1.32. ClutCntl Command

[Address: 0x90]



Name	Description
Base	In the 3D CLUT, sets the base address for loading data into the referenced SRAM. Must be set in 256-byte boundary units. The referenced data is packed either as 24-bit per word (from LSB) or 32-bit per word, depending on the Mode. If En is not '0', setting this to '0' is prohibited.

Ring In the 3D CLUT (excluding 1D mode Binary), sets the interpolation type for each ARGB element result.

- If set to '0', extrapolation is performed at the final edge.
- If set to '1', ring interpolation is performed (connecting the coordinate following the final one back to coordinate 0).

In 1D mode Binary, this specifies a 2ⁿ-bit lookup:

- Ver.AB supports n = 0, 1
- Ver.C supports n = 0–5

Flush In the 3D CLUT, clears the cache information of the referenced SRAM and reloads the data.

Sel In the 3D CLUT, configures various options. (See Mode for details.)

Mode Sets the reference mode in the 3D CLUT.

Mode	Description
0	<p>1D CLUT (Standard, Sel = '0'):</p> <p>Each element indexes a 32-bit table using values from -1.0 (0x10100000) to 1.0 (0x10100000).</p> <p>1D CLUT (Binary, Sel = '1'):</p> <p>If Ring[0] = '0', a 17-bit table index is formed from R000, G7:07:07:0, B7:07:07:0, A7:07:07:0 elements output either 0 (0x00) or 1 (0xFF/0x100).</p> <p>If Ring[0] = '1', a 16-bit index from G7:07:07:0, B7:07:07:0 is used for the lookup.</p> <ul style="list-style-type: none"> • MSB result applies to A • LSB result applies to RGB

1	2D CLUT: Uses a combination of R7:27:27:2 and B7:27:27:2 to index a 32-bit table. If Sel = '1', performs bi-linear interpolation using R1:01:01:0, B1:01:01:0.
2	D CLUT: Uses R7:47:47:4, G7:47:47:4, B7:47:47:4 to index a 32-bit table. If Sel = '1', performs tri-linear interpolation using R3:03:03:0, G3:03:03:0, B3:03:03:0.
3	Reserved

Blut.Ext	Description
0	2D/3D CLUT outputs 9 bits per element.
1	2D/3D CLUT treats the B element as 8.8 fixed-point and concatenates it with the G and B elements for output. The MSB of the 9-bit output per element is set to 0.

Blut.Sel	Description
0	No dynamic range expansion.
1	Performs dynamic range expansion. The table references A2:02:02:0 and adjusts low-value RGB settings. Pre-configure the RGB table with values multiplied by $2^{A[2:0]}$. Each of R, G, and B can have separate A2:02:02:0 values.

En

In the 3D CLUT, specifies whether to enable processing for each element.

If all bits are set to '0', the 3D CLUT is bypassed.

En	Description
[0]	Enables processing for Blue element
[1]	Enables processing for Green element
[2]	Enables processing for Red element
[3]	Enables processing for Alpha element

5.3.1.33. BlutCntl Command

[Address: 0x94]

31		28			24			20			16			12			8			4			0
Base[31:8]																			Sel	OptT[1:0]	OptC[1:0]	Ext	En
																		Flush					

Name	Description
Base	<p>In Blender or Filter processing, sets the base address for loading data into the referenced SRAM. Must be set in 256-byte boundary units.</p> <p>The referenced data is packed as 8 bits per word, starting from the LSB.</p> <p>If En is not '0', setting this to '0' is prohibited.</p>
Flush	<p>In Blender or Filter processing, clears the cache information of the referenced SRAM and reloads the data.</p>
Sel	<p>In the 3D CLUT, configures various options. (See ClutCntl Command for details.)</p>
OptT	<p>Information sent as the LSB of the address to the memory system.</p> <p>Common address information for Clut and Blut.</p>
OptC	<p>Information sent as the LSB of the address to the memory system.</p> <p>Common address information for Output Context, Histogram,</p>

- Ext

and Steal.

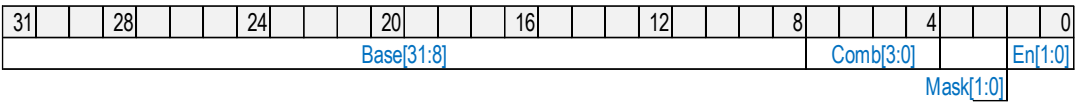
In the 3D CLUT, configures various options. (See ClutCntl Command for details.)
- En

In Blender or Filter processing, specifies whether the reference is enabled.

In general, set this to '1' if any of PixelCntl*.Lut is '1', or if the Filter references Blut.

5.3.1.34. StealCntl Command

[Address: 0x98]



Name	Description
Base	Sets the start address used by Steal. Must be set in 256-byte boundary units.

- Comb

Sets the final evaluation logic for Steal.

S represents flags generated by the Filter, and D represents flags (D) generated by the Extractor.

If set to 0, the Steal function is disabled.

Op[3:0]	Description	Op[3:0]	Description
0	0	8	S & D
1	$\sim S \ \& \ \sim D$	9	$S \ \sim \sim D$
2	$S \ \& \ \sim D$	10	S
3	$\sim D$	11	$S \ \ \sim D$
4	$\sim S \ \& \ D$	12	D

5	$\sim S$	13	$\sim S \mid D$
6	$S \wedge D$	14	$S \mid D$
7	$\sim S \mid \sim D$	15	1

Mask

Configures how the Steal flag inversion mask (StealMask) and the Extractor-generated mask (ExtractorMask) affect the final mask (BlenderMask).

Mask[1:0]	Description
0	BlenderMask = ExtractorMask
1	BlenderMask = 0
2	BlenderMask = ExtractorMask \mid StealMask
3	BlenderMask = StealMask

En

Selects and executes the Steal method.

En[1:0]	Description
0	NOP
1	Performs Steal at the current coordinates (SrcX, SrcY) (available from Ver.C onward).
2	Performs Steal on Blender data (Mod) (available from Ver.C onward).
3	Performs Steal on Blender data (Org) (available from Ver.C onward).

5.3.1.35. AffineCoef0-8 Command

[Address: 0x9c – 0xbc]

31			28				24					20					16					12					8					4					0
S	Exp[7:0]							Mantissa[22:0]																													

Name	Description
S	Sets the Sign bit of the IEEE 754 Binary32 (single-precision) format.
Exp	Sets the Exponent field of the IEEE 754 Binary32 (single-precision) format. Values 0x00 and 0xFF are not supported.
Mantissa	Sets the Mantissa field of the IEEE 754 Binary32 (single-precision) format.

5.3.1.36. FilterCntIn/Out Command

[Address: 0xc0/0xc4] (Ver.AB)

31		28			24			20			16			12			8			4			0				
		Stride[5:0]				Force[7:0]						VMask[3:0]				Mode[3:0]				Class[3:0]				En[3:0]			
Edge[1:0]																											

[Address: 0xc0/0xc4] (Ver.C)

31			28			24			20			16			12			8			4			0
Edge[3:0]			Signed[3:0]			Force[7:0]						VMask[3:0]			Mode[3:0]			Class[3:0]			En[3:0]			

Name	Description
Edge	Sets the endpoint option when retrieving the filter kernel. SrcSize must be non-zero.

Edge	Description
0	If the center coordinate of the Pixel Cache (rounded up in the signed direction) is not within the image, the entire Pixel Cache is replaced with PixelDefault. If the above condition is not met, only the out-of-bounds parts are replaced with PixelDefault.
1	No boundary checking is performed.
2–7	Reserved

8	Out-of-bounds parts are replaced with PixelDefault.
9*	Out-of-bounds parts are replaced with the nearest pixel value.
10*	Out-of-bounds pixels are replaced with corresponding values from a wrapped image.
11*	Out-of-bounds pixels are replaced with corresponding values from a mirrored image (no duplication at fold points).
12-14	Reserved
15	Out-of-bounds pixels are replaced with corresponding values from a mirrored image (with duplication at fold points).

* Options 1, 9-11, and 15 are available from Ver.C onward.

Signed

Configures how to interpret each input ARGB element:

- '1': Two's complement
- '0': Unsigned

Stride

Sets the address interval between adjacent pixels in the filter kernel.

For Stride[5], the address interval is $2 \times \text{Stride}[4:0]$ bytes (Ver.AB).

Force

In FilterIn, selects the data to enter the filter on a per-element basis.

Each 2 bits from the upper side corresponds to an ARGB element.

In FilterOut, selects the Bayer mask pattern.

- When using the BayerMask register:

Grayscale selects 1 of 8 patterns

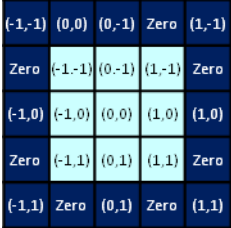
Full color selects 1 set out of 4 patterns

- When using Blut:

Grayscale selects 1 of 4

Full color selects from a set of 4

InForce[2n-1:2n]	Description
0	SrcIn

1	SrcOut (Gray)
2	<p>Center pixel from SrcIn, surrounding pixels from SrcOut. See diagram below. (Coordinates in parentheses are for SrcIn and SrcOut 3x3 kernels centered at 0.)</p> 
3	All pixels set to 1.0

VMask

Masks access to horizontal lines in the kernel.

VMask	Description
[0]	Masks access to lines at ± 1 from the kernel center
[1]	Masks access to lines at ± 2 from the kernel center
[2]	Masks access to lines at ± 3 from the kernel center
[3]	Masks access to lines at ± 4 from the kernel center

Mode

Sets the kernel size. Must be configured according to each filter type.

Class

Specifies the type of filter. (See Mode for details.) Differs between SrcIn and SrcOut types. For SrcOut, the setting is fixed to Pattern filter.

Mode[2:0] Class[3:0]		0	1	2	3	4	5	6	7
0	2D Point	1 x 1	2 x 2	3 x 3	4 x 4	5 x 5	Reserved		
1	2D Bi-linear		2 x 2*		4 x 4**	Reserved			
2	2D Bi-cubic	Reserved			4 x 4***	Reserved			
3	Reserved	Reserved							
4	Non-linear	1 x 1	2 x 2	3 x 3	4 x 4	5 x 5	Reserved		
5	Mask	Mask	Mix	Reserved					
6	Hamming	Min	Max	Reserved					
7	Extrema	In Check In Out	In Check Diff Out	Diff Check In Out	Diff Check Diff Out	Reserved			
8	2F (Ver.C) Point	1 x 1	2 x 2	3 x 3	4 x 4	5 x 5	Reserved		
9	2F (Ver.C) Bi-linear		2 x 2*		4 x 4**	Reserved			
10	2F (Ver.C) Bi-cubic	Reserved			4 x 4***	Reserved			
11	SAD/SSD (Ver.C)	1 x 1	2 x 2	3 x 3	4 x 4	5 x 5	Reserved		
12	Bitmap	32 x 25 (Ver.C*)	Reserved						
13~15	Reserved	Reserved							

SrcIn types:

* 1x1 filter coefficient Coef000 is valid (1x1 is bilinearly interpolated to 2x2)

** 3x3 filter coefficients Coef000, Coef100–107 are valid (3x3 is bilinearly interpolated to 4x4)

*** Filter coefficients are invalid

SrcOut types:

Always Pattern filter (fixed)

Class[3:0] \ Mode[3:0]	0	1	2	3	4	5	6	7
0-15 Pattern	1 x 1	Reserved	3 x 3	Reserved	5 x 5	Reserved	7 x 7	Reserved

Class[3:0] \ Mode[3:0]	8	9	10	11	12	13	14	15
0-15 Pattern	9 x 9	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved

En

For SrcIn type, enables or disables filtering for each element. Setting all bits to '0' bypasses filtering. For Extrema Filter only, this acts as a flag to enable difference calculation for Layers 0–3.

For SrcOut type, kernel size is fixed to 1x1 only when En =

'1111' (not required except for Ver.AB).

En	Description
[0]	Enable for Blue element
[1]	Enable for Green element
[2]	Enable for Red element
[3]	Enable for Alpha element

5.3.1.37. FilterCntlOp Command

[Address: 0xc8]

31		28		24			20			16			12			8			4			0
OutOp[7:0]				OutSel[7:0]				InOp[7:0]				InSel[7:0]										

Name Description

OutSel In the Pattern Filter, the input reference value can be switched, and the output data can be selected.

OutSel[2:0]	Description
0	NOP
1	Reserved
2	SrcOut (Center reference value)
3	Blut[240]
4	SrcIn[7:0] (Center reference value)
5	SrcIn[15:8] (Center reference value)
6	SrcIn[23:16] (Center reference value)
7	SrcIn[31:24] (Center reference value)

OutSel[3]	Description
0	Outputs the lower 32 bits of the 64-bit boolean flag result (2 bits × 32).
1	Outputs a 32-bit result obtained by performing a bitwise AND operation on each 2-bit pair of the 64-bit boolean flag result (2 bits × 32).

OutSel[5:4]	Description
0	NOP
1	Bayer Performs mask processing. (BayerMask Register Use)

2	Reserved
3	Bayer Performs mask processing.(Blut Use)

InOp

Configures the filter mode for the SrcIn system.

InOp	2D/2F	2F	None – linear	Mask	Hamm ing	Extre ma	Bitma p
[0]	Coe f	Coef	Type	Comp	OutputData[Min Ignore	Defaul t
[1]	Set B	Set B	Set B	Const	15:0]*	Max Ignore	Reser ved
[2]	Coe f	Reser ved	Type	Loc	OutputData[Lower Ignore	
[3]	Set G		Set G		31:16] *	Upper Ignore	
[4]	Coe f		Type	Kernel	OutputOri n[15:0	Equal Ignore	
[5]	Set R		Set R		*]	Alt Ignore	
[6]	Coe f	0: fp out	Type	Aroun d	OutputOri n[15:0	Zero Ignore	
[7]	Set A	2: ftoi	Set A		*]	Swap	

- 0: Comp (Total comparison result)
- 1: Eval (32-bit evaluation result)
- 2: PosX (X-coordinate for minimum/maximum value)
- 3: PosY (Y-coordinate for minimum/maximum value)

For 2D/2F Filters:

Configures the coefficient placement for each element. The black numbers indicate coefficient indices (among the 28 coefficients stored in FilterCoef000–215). The background color denotes the index range: white for 0xx, light gray for 1xx, dark gray for 2xx, and white–outline boxes represent fixed values (0.0 / 1.0).

For Non-linear Filters:

Configures the filter mode for each element. The composite result of the four Outputs the value corresponding to the result of the element specified by InSel[7:6]. (i.e., the output follows the result of the selected element) elements can be reselected and output using InSel[5:4].

InOp[2n+1:2n]	Description
0	Minimum 5x5
1	Maximum 5x5
2	Median 3x3
3	Outputs the value corresponding to the result of the element specified by InSel[7:6]. (i.e., the output follows the result of the selected element)

For Mask Fiter: Configures various modes (see Figure 25).

For Hamming Fiter: Sets the number of inspection bits in multiples of 32 bits (value is specified as N-1).。

For Extrema Fiter:

InOp	Description
[0]	When set to '1', minimum values are not evaluated.
[1]	When set to '1', maximum values are not evaluated.
[2]	When set to '1', the lowest level (0) is also evaluated as a minimum value (a virtual level below is treated as the minimum).
[3]	When set to '1', the highest level (InScale[2:0] – 1) is also evaluated as a maximum value (a virtual level above is treated as the maximum).
[4]	When set to '1', equal comparison results are also treated as extrema (max/min).
[5]	When set to '1', both maximum and minimum information are output simultaneously (normally exclusive, with lower hierarchy taking priority).

[6]	When set to '1', pixels with difference value or pixel value of 0 are excluded from extrema evaluation candidates.
[7]	When set to '1', swaps the outputs of SrcMod and SrcOrg.

For Bitmap Fiter: Sets the out-of-range bit value to InOp[0].

InSel Selects the input and output data for the 2D, 2F, Non-linear, Mask, Hamming, and Extrema Filters.

Coefficient Input:

InSel[1:0]	Description
0	Command List
1	Blut
2	SrcIn
3	SrcOut'

Coefficient Operation: (Valid only for 2D/2F Filters)

InSel[3:2]	Description
0	Normal
1	Reserved
2	Bilateral coefficient calculation using the element specified by InSel[7:6]
3	Epsilon coefficient calculation using the element specified by InSel[7:6]

Plane Output: (From 9x9 Source Out)

InSel[5:4]	Description
0	1-plane: Each filter output element is output as-is.

1	2-plane: <ul style="list-style-type: none"> – 2D Filter: Output results for AR and GB are summed and output to – Non-linear Filter: Output ARGB results as: $B' = \min(B, G), G' = \max(B, G), R' = \min(R, A), A' = \max(R, A)$ – Other filters: Reserved.
2	4-plane: <ul style="list-style-type: none"> – 2D Filter: Output results for ARGB are summed and output to – Non-linear Filter: Output ARGB results as: $B' = \min(B, G, R, A), R' = \max(B, G, R, A)$ – Other filters: Reserved.
3	Special plane: <ul style="list-style-type: none"> – Non-linear Filter: Uses the element specified by InSel[7:6] for c – Other filters: Reserved.

Plane Evaluation:

InSel[7:6]	Description
0	Evaluate element B
1	Evaluate element G
2	Evaluate element R
3	Evaluate element A

5.3.1.38. FilterCoef00 Command (Coefficient Filter Mode)

[Address: 0xcc]

31		28			24			20			16			12			8			4			0
S	Exp[4:0]			Mantissa[9:0]								Ref[15:0]											

Name	Description
S	Sets the Sign bit in IEEE754 Binary16 (half-precision) format.
Exp	Sets the Exponent in IEEE754 Binary16 (half-precision) format. 0x00 and 0x1f are not supported.
Mantissa	Sets the Mantissa in IEEE754 Binary16 (half-precision) format.
Ref	Specifies the reference value for the filter.

Filter Class	Description
2D	Multiplies the result by $2^{\text{Ref}[11:8]}$ (two's complement). Specifies the bilateral variance via Ref[3:0] when InSel[3:2] = 2. Specifies the epsilon threshold via Ref[7:0] when InSel[3:2] = 3.

2F	Multiplies the result by $2^{\text{Ref}[11:8]}$ (two's complement). Specifies the number of invalid MSBs using Ref[15:12] (e.g., 0 means no invalid MSBs, 6 means lower 10 bits are valid).
SAD/SSD	Ref[15:0] sets the minimum value.
NL	Not used
Mask	Ref[11:8] specifies the Boolean algebra table.
Hamming	Ref[15:0] specifies the YMax of the polygon shape "Rectangle."
Extrema	Ref[10:8] specifies the evaluation level.
Bitmap	Not used

Bilateral Variance Table	Description
0	Gaussian distribution with $\sigma = 0.375$
1	Gaussian distribution with $\sigma = 0.75$
2	Gaussian distribution with $\sigma = 1.5$
3	Gaussian distribution with $\sigma = 3$
4	Gaussian distribution with $\sigma = 6$
5	Gaussian distribution with $\sigma = 12$
6	Gaussian distribution with $\sigma = 24$
7	Gaussian distribution with $\sigma = 48$
8–15	Reserved

5.3.1.39. FilterCoef10-27 Command (Coefficient Filter Mode)

[Address: 0xd0 + 2n (n = 0–24)]

31		28				24				20				16				12					8					4				0
Expn+1[4:0]				Mantissan+1[9:0]										S_n	Expn[4:0]				Mantissan[9:0]													

S_{n+1}

Name	Description
S	Sets the Sign bit in IEEE754 Binary16 (half-precision) format.
Exp	Sets the Exponent in IEEE754 Binary16 (half-precision) format. 0x00 and 0x1f are not supported.
Mantissa	Sets the Mantissa in IEEE754 Binary16 (half-precision) format.

5.3.1.40. FilterTable Command (Mask Filter Mode)

[Address: 0xc8 + 4n (n = 0–7)]

31			28				24				20				16				12				8				4			0	
T ₃₁	T ₃₀	T ₂₉	T ₂₈	T ₂₇	T ₂₆	T ₂₅	T ₂₄	T ₂₃	T ₂₂	T ₂₁	T ₂₀	T ₁₉	T ₁₈	T ₁₇	T ₁₆	T ₁₅	T ₁₄	T ₁₃	T ₁₂	T ₁₁	T ₁₀	T ₉	T ₈	T ₇	T ₆	T ₅	T ₄	T ₃	T ₂	T ₁	T ₀

Name	Description
T _x	The eight registers are concatenated to form a 256-bit table. The index for each entry is extended as x + 32n. This table is ultimately referenced as T255–T0 by the Mask Filter.

5.3.1.41. FilterCenter Command (Mask Filter Mode)

[Address: 0xe8]

31			28				24				20				16				12				8				4			0
DeltaUpper[7:0]							DeltaLower[7:0]														Val[7:0]									

Name	Description
DeltaUpper	Used to determine the upper threshold for center pixel evaluation by adding it to the value of Value.
DeltaLower	Used to determine the lower threshold for center pixel evaluation by subtracting it from the value of Value.
Value	Specifies the reference value used for comparing the center pixel in Mask Filter mode.

5.3.1.42. FilterAround Command (Mask Filter Mode)

[Address: 0xec]

31			28				24				20				16				12				8				4			0
DeltaUpper[7:0]							DeltaLower[7:0]														Val[7:0]									

Name	Description
DeltaUpper	Used to determine the upper threshold for neighboring pixel evaluation by adding it to either Value or the center pixel value.
DeltaLower	Used to determine the lower threshold for neighboring pixel evaluation by subtracting it from either Value or the center pixel value.
Value	Specifies the reference value used for comparing surrounding pixels in Mask Filter mode. Used when FilterContl.Op[5] is '0'. When set to '1', the center pixel value is selected.

5.3.1.43. FilterReplace Command (Mask Filter Mode)

[Address: 0xf0]

31			28			24				20			16				12			8				4			0
A[7:0]							R[7:0]							G[7:0]							B[7:0]						

Name	Description
A,R,G,B	Specifies the pixel value to be used for replacement in Mask Filter mode when the target condition is met.

6. Application Notes

6.1. Overall Control

6.1.1. Processing Unit

- frComp converts the intermediate coordinates into physical coordinates by processing units and executes the operations. Shorter processing units allow faster switching between different tasks, but may disrupt continuous access to external memory, resulting in wasted cycles during access transitions.
- The minimum time required to load a Command List is 32 cycles. Therefore, for frComp, which can process one pixel per cycle, it is preferable to set the processing unit to at least 32 pixels. However, if the same Command List continues, the loading process can be skipped. As such, even if the processing unit is smaller than 32 pixels, it poses no issue unless frequent switching between different Command Lists occurs.
- When using *pss*, it is necessary to consider the minimum task switching time for *pss* (number of pipeline stages \times 2 cycles). In most cases, setting the processing unit to the total number of horizontal pixels in the image is sufficient.
- One important consideration in Command List loading time is whether Clut data needs to be loaded. If loading is required, a minimum of 2K cycles is consumed for each task switch. However, since two Clut banks are available, reloading is unnecessary if the same Clut is referenced across tasks.
- Both Clut and Blut are loaded from memory, and any change in memory content requires cache clearing and reloading. By setting ClutCntl.Flush and BlutCntl.Flush to '1', the cache can be cleared and reloaded at the beginning of a frame.
- The choice of pixel format is important for memory bandwidth considerations. For example, full-color images at 32 Bpp use four times the bandwidth of grayscale images at 8 Bpp. This difference becomes more significant in filter processing with a 5×5 kernel, where the bandwidth usage increases in proportion to the kernel area or fill size. If external caches are used, the format selection also affects whether memory access exceeds the cache capacity.
- Contexts, Cluts, and Bluts can be defined simultaneously in any number (up to 2 in Ver.A). Since fragmentation processing overlaps with load/save operations, it generally has no impact on performance. However, because loading and saving occur repeatedly with each fragmentation, the amount of

fragmentation and number of simultaneous operations should be managed carefully.

6.1.2. Functional Orthogonality

- Ideally, the configuration of one function should not affect others; however, frComp has certain limitations:
 - Only one of the six filters can be selected for the source coordinate system.
 - Blut is shared between the Mask Filter and the Blender and cannot be used by both simultaneously.
 - Available interpolation types are limited depending on the kernel size used in the 2D filter.
 - In Steal operations, writing to the context is required. Reading min/max values from the context may overwrite the currently processed values, so care must be taken.

6.1.3. Processing Symmetry

- Although each pixel consists of four independent elements, certain functions require element-specific considerations:
 - When the input format has unique characteristics (e.g., RGB565 or YUV)
 - When specific elements are fixed in calculations in Median/Mask Filters
 - When 3D Clut access is fixed to particular elements in 2D/3D modes
- If element-wise processing is fully independent and no neighboring pixels are accessed via filters, four pixels can be processed simultaneously by treating each as one element per pixel. However, the I/O format must be treated as 32 Bpp, imposing a restriction that the pixel width must be a multiple of four. For example, a simple transfer of a grayscale image should use a 32 Bpp I/O format rather than 8 Bpp.
- The source coordinate system serves as the Main path and has greater functionality than the Destination coordinate system (Sub path), resulting in asymmetry.

Function	Source Path	Destination Path
Remapper Interpolation	Nearest Bi-linear	Nearest Bi-linear
Affine Transform	Yes	No

Pixel Cache	1x1 – 5x5	1x1 – 9x9 3x3 (Extrema Filter Use) 5x5 (Correlation Use) 1x1 (Other Use)
Filter	2D 2F Non-linear Mask Extrema Bitmap SSD/SAD	Patern

The coordinate mapping circuitry used for the Source output image (SrcOut path) and the Destination coordinates is shared. Therefore, both mappings cannot be used simultaneously unless they are identical. Typically, these paths are treated exclusively.

6.1.4. Polygon Rendering

- When ShapeCntl.Type is set to '0', only fragments with the same Y coordinate and differing X coordinates are processed (normal mode). For other values, polygon rendering is performed based on the configured vertex values (polygon mode). The polygon is decomposed into scanlines in the positive Y direction, and intermediate coordinates are automatically generated. All other operations remain unchanged.
- Since vertex values are defined using 16-bit positive integers, it is not possible to define polygons that span negative screen coordinates or lie outside the screen area. Such shapes must be clipped at the configuration stage. Texture mapping is performed by combining matrix transformation and filtering (this differs from Texture conversion). Intermediate coordinates are directly provided as Destination coordinates (X, Y), and matrix transformation is used to generate Texture coordinates (U, V).
- When the texture coordinates (U, V) corresponding to three vertices (X, Y) are given, the transformation matrix **M** can be computed as a set of constants. However, if the determinant of matrix **V**, which represents the polygon's area, is zero (i.e., the area is zero), a valid configuration cannot be obtained. Any filter mode can be used in the 2D Filter, but bi-linear interpolation is commonly applied.

• Matrix Derivation

Given three vertices $(x_0, y_0), (x_1, y_1), (x_2, y_2)$ $(x_0, y_0), (x_1, y_1), (x_2, y_2)$, define matrix **V** as:

$$V = \begin{pmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{pmatrix}$$

The coefficients of the desired transformation matrix are computed as:

$$\begin{pmatrix} m_{00} & m_{01} & m_{03} \\ m_{10} & m_{11} & m_{13} \end{pmatrix}^T = V^{-1} \begin{pmatrix} u_0 & v_0 \\ u_1 & v_1 \\ u_2 & v_2 \end{pmatrix}$$

Here, the inverse of **V** can be derived from the Destination coordinates (X,Y)(X, Y) as:

$$V^{-1} = \frac{1}{\det(V)} \begin{pmatrix} y_1 - y_2 & y_2 - y_0 & y_0 - y_1 \\ x_2 - x_1 & x_0 - x_2 & x_1 - x_0 \\ x_1y_2 - y_1x_2 & y_0x_2 - x_0y_2 & x_0y_1 - y_0x_1 \end{pmatrix}$$

$$\det(V) = x_0y_1 + x_1y_2 + x_2y_0 - x_0y_2 - x_2y_1 - x_1y_0$$

We provide specific configuration examples and diagrams illustrating cases with and without matrix transformation during texture mapping. In the diagrams, parentheses indicate texture coordinates (U, V).
The matrix **M** is supplied to *frComp* in floating-point precision.

Vertex No	x	y	u	v
0	13	18	1	1
1	63	3	0	63
2	32	63	63	32

det(V)	2535
--------	------

V^{-1}	-0.023668639	0.017751479	0.00591716
	-0.012228797	-0.007495069	0.019723866
	1.527810651	-0.095857988	-0.431952663

M	0.349112426	1.230374753	-25.6852071
	1.284023669	0.146745562	-18.33372781

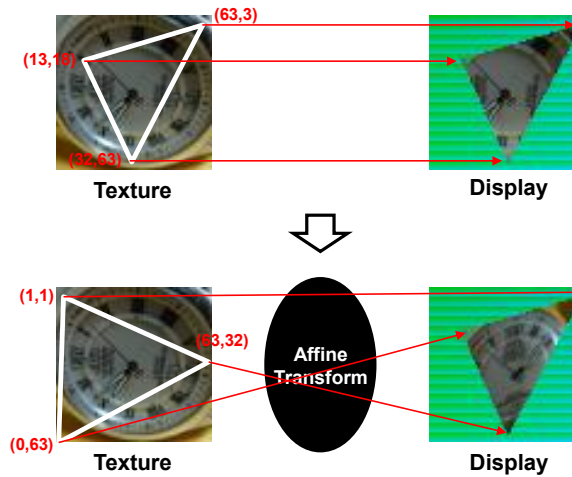


Figure 42 Texture Mapping

When using matrix transformation for texture mapping, there may be references beyond the edges of the texture image due to rounding or truncation errors. To handle this, set an appropriate range in SrcSize and configure FilteCntl.Edge so that edge pixels are referenced even when access goes beyond the image boundary. Alternatively, expand the texture image size to include a guard region that safely absorbs out-of-bound accesses.

When processing pixels of arbitrary length defined by registers, a polygon shape (Line mode) must still be specified. While coordinate values are generally limited to 16-bit in non-Line modes, Line mode supports 32-bit linear coordinates.

When input coordinates to frComp are specified as X_{in} , Y_{in} , Z_{in} , and W_{in} , the post-polygon-rendering coordinates X_p and Y_p are generated as described below. Note that when $MasterCntl.Shape \neq 0$, all shape-related coordinates X_p , Y_p are generated in a single invocation of frComp.

$$\begin{pmatrix} X_p \\ Y_p \end{pmatrix} = \begin{pmatrix} X_{in} \\ Y_{in} \end{pmatrix} \quad \text{When}$$

$MasterCntl.Shape=0$

$$\begin{pmatrix} X_p \\ Y_p \end{pmatrix} = \text{Parallelogram} \begin{pmatrix} x0 & y0 \\ x1 & y1 \end{pmatrix} \quad \text{When}$$

MasterCntl.Shape=1

$$\begin{pmatrix} Xp \\ Yp \end{pmatrix} = \text{Rectangle} \begin{pmatrix} x0 & y0 \\ x1 & y1 \\ x2 & y2 \\ x3 & y3 \end{pmatrix}$$

When

MasterCntl.Shape=2

$$\begin{pmatrix} Xp \\ Yp \end{pmatrix} = \text{Triangle} \begin{pmatrix} x0 & y0 \\ x1 & y1 \\ x2 & y2 \end{pmatrix}$$

When

MasterCntl.Shape=3

$$\begin{pmatrix} Xp \\ Yp \end{pmatrix} = \text{Line}(x0 + y0 \times 65536)$$

When

MasterCntl.Shape=4

$$\begin{pmatrix} Xp \\ Yp \end{pmatrix} = \text{Line}(\text{context data})$$

When

MasterCntl.Shape=5

$$\begin{pmatrix} Zp \\ Wp \end{pmatrix} = \begin{pmatrix} Zin \\ Win \end{pmatrix}$$

When always

6.1.5. Scan Modifications

- The coordinates processed by polygon rendering can be further modified by incorporating Z and W coordinates, regardless of whether polygon processing is enabled. Various forms of four-dimensional scanning are supported.
- The Source pixel input system and Destination pixel input system can each be configured independently using the settings described below. Primarily, Types 0 to 2 involve one-dimensional coordinate remapping, while Type 3 involves two-dimensional coordinate remapping. These can be used to improve memory access efficiency through mosaic-style addressing or to facilitate data structure transformations.

$$\begin{pmatrix} U0 \\ V0 \end{pmatrix} = \text{Sel0} \begin{pmatrix} Xo \\ Yo \\ Xp \\ Yp \\ Zp \\ Wp \end{pmatrix}, \begin{pmatrix} U1 \\ V1 \end{pmatrix} = \text{Sel1} \begin{pmatrix} Xo \\ Yo \\ Xp \\ Yp \\ Zp \\ Wp \end{pmatrix}$$

*o: original, *p:

polygon

$$\begin{pmatrix} Xt \\ Yt \end{pmatrix} = \begin{pmatrix} U0 \% 2^{16-\text{Mask}X} + U1 * 2^{\text{Box}X} \\ V0 \% 2^{16-\text{Mask}Y} + V1 * 2^{\text{Box}Y} \end{pmatrix}$$

Type0

$$\begin{aligned}
\begin{pmatrix} X_t \\ Y_t \end{pmatrix} &= \begin{pmatrix} U0/2^{MaskX} + U1 * 2^{BoxX} \\ V0/2^{MaskY} + V1 * 2^{BoxY} \end{pmatrix} && \text{Type1} \\
\begin{pmatrix} X_t \\ Y_t \end{pmatrix} &= \begin{pmatrix} U0\%2^{16-MaskX} + U1/2^{BoxX} \\ V0\%2^{16-MaskY} + V1/2^{BoxY} \end{pmatrix} && \text{Type2} \\
\begin{pmatrix} X_t \\ Y_t \end{pmatrix} &= \begin{pmatrix} U0\%2^{BoxX} + U1/2^{BoxY} * 2^{BoxX} \\ V0/2^{BoxX}\%2^{16-MaskX} + V1/2^{BoxY}\%2^{15-MaskY} * 2^{MaskX} \end{pmatrix} && \text{Type3}
\end{aligned}$$

Figure 43 Index Map of Type3

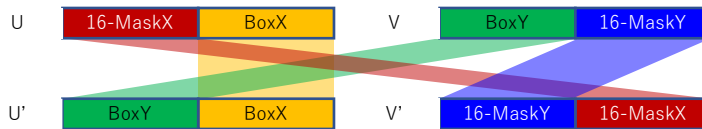
Although scan modification is typically unnecessary for the Destination pixel input system, it may be used when connecting the Destination pixel input system to the SrcOut path for operations such as cross-correlation.

6.2. Coordinate Operations

6.2.1. Mapping Data

The *Remapper* generates Source and Destination coordinates from intermediate coordinates. In general, the intermediate coordinates are set equal to the Destination coordinates. In the following explanation, intermediate coordinates are denoted as $(X_t, Y_t)(X_t, Y_t)$. Note that the coordinates generated by polygon scan traversal are also considered intermediate coordinates.

The Source coordinates $(X_s, Y_s)(X_s, Y_s)$ and Destination coordinates $(X_d, Y_d)(X_d, Y_d)$ corresponding to the intermediate coordinates $(X_t, Y_t)(X_t, Y_t)$ are obtained using



mapping data. Mapping data can be expressed as either absolute or relative values. The mapping data $(X_m, Y_m)(X_m, Y_m)$ uses two's complement representation, which differs from the representation format of the intermediate coordinates $(X_t, Y_t)(X_t, Y_t)$ that are directly input to *frComp*. Be aware that in memory, coordinate data is stored with Y in the upper 16 bits (MSB) and X in the lower 16 bits (LSB).

$$(X_m, Y_m) \xleftarrow{\text{access}} \text{map}(\text{BaseAddr} + 4 \cdot Y_t/2^{Div} \cdot \text{Stride} + 4 \cdot X_t/2^{Div})$$

Here, BaseAddress indicates the starting address of the mapping data; Stride defines the memory increment when Y_t changes; and Div specifies the subdivision factor.

- The mapping data (X_m,Y_m)(X_m, Y_m)(X_m,Y_m) is processed into the mapped coordinates (X_r,Y_r)(X_r, Y_r)(X_r,Y_r) using the following transformation. An arbitrary fractional precision Prec can be specified. Setting the fractional precision improves the accuracy of interpolation performed by the subsequent filter stage.

$$(X_r, Y_r) = \frac{\text{BiLiner} \begin{bmatrix} (X_{m_{00}}, Y_{m_{00}}) & (X_{m_{01}}, Y_{m_{01}}) \\ (X_{m_{10}}, Y_{m_{10}}) & (X_{m_{11}}, Y_{m_{11}}) \end{bmatrix}}{2^{\text{Prec}}} + (X_a, Y_a)$$

Here, BiLiner[] refers to the bi-linear interpolation function, where the indices of (X_m,Y_m)(X_m, Y_m) represent the positions of the four neighboring samples. Prec denotes the fractional precision setting. (X_a,Y_a)(X_a, Y_a) represents an offset: it is set to (0,0)(0, 0) for absolute values, or to (X_t,Y_t)(X_t, Y_t) for relative values.

- The transformation from intermediate coordinates (X_t,Y_t)(X_t, Y_t) to Source and Destination coordinates can be configured independently. These are denoted as (X_{rs},Y_{rs})(X_{rs}, Y_{rs}) for Source coordinates and (X_{rd},Y_{rd})(X_{rd}, Y_{rd}) for Destination coordinates. Finally, the Source coordinates are computed by applying a matrix transformation to the mapped values.

$$\begin{pmatrix} X_s \\ Y_s \end{pmatrix} = \begin{pmatrix} X_t/Z_t \\ Y_t/Z_t \end{pmatrix}, \begin{pmatrix} X_t \\ Y_t \\ Z_t \end{pmatrix} = \begin{pmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{pmatrix} \begin{pmatrix} X_{rs} \\ Y_{rs} \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} X_d \\ Y_d \end{pmatrix} = \begin{pmatrix} X_{rd} \\ Y_{rd} \end{pmatrix}$$

- The above description implies that matrix transformation precedes Source coordinate calculation. Conceptually, this means that the Source image is first inverse-transformed—e.g., translated or rotated—before being mapped. Note that this is the reverse of the actual hardware processing order and must be handled with care.

- The mapping data (X_m, Y_m) can be consolidated into square regions of size $2^{Div} \times 2^{Div}$, depending on the subdivision level Div . For example, if $Div = 4$, the region from Destination coordinate $(0, 0)$ to $(15, 15)$ can be represented using a single (X_m, Y_m) value. Although this discrete sampling introduces approximation errors, the Interp mechanism reduces this error by applying bi-linear interpolation using the four adjacent mapping data points, based on the remainder after dividing the intermediate coordinate by 2^{Div} . The value of Div should be selected as a trade-off between compressing the mapping data (larger Div) and tolerable approximation error.
- In the mapping data, the value $0x8000$ functions as an escape code. Its behavior differs depending on whether it is used in the Source or Destination Remapper:
 - In the Source Remapper, it holds the previously accessed value (data retention).
 - In the Destination Remapper, it acts as a drawing mask.

This escape code is particularly useful when processing arbitrary shapes. For instance, in mappings derived from motion vectors, escape codes can be embedded in regions that should not be processed. Note that the escape code feature can also be disabled if needed.

6.2.2. Polar Coordinate Transformation

- In polar coordinate transformation, consider the case where mapping from Destination coordinates to Source coordinates is specified one-to-one for each pixel. When the Destination coordinates (X_d, Y_d) represent polar coordinates—with X_d as the angle and Y_d as the radius—the corresponding Source coordinates (X_s, Y_s) can be calculated using the following formulas:

$$X_s = \frac{Y_d}{2} \cos\left(\frac{2\pi}{W_x} X_d\right) + \frac{W_y}{2} \quad Y_s = \frac{Y_d}{2} \sin\left(\frac{2\pi}{W_x} X_d\right) + \frac{W_y}{2}$$

- In this case, the Destination coordinates are directly assigned from the intermediate coordinates, while the Source coordinates are derived using the Source Remapper. By scanning through the Destination coordinates (X_d, Y_d) , the corresponding Source coordinates (X_s, Y_s) can be obtained. These values are then sampled and stored as mapping data. To ensure that the resulting coordinates (X_s, Y_s)

do not exceed 0xFFFF, they are scaled by a factor of 2^{2n} , where n is a common multiplier. The translational component in the above formula may alternatively be handled through matrix transformation.

- The resulting coordinates are packed into 32-bit words in the format $\{Y_s, X_s\} \times \{Y_s, X_s\}$ and stored in memory (note that the order is Y_s, X_s, Y_s, X_s). The memory layout generally follows the same arrangement as the Destination image, though the stride can be configured independently.
- The value of n , which represents the precision, is specified via `SrcMapInfo.Prec`. Since a maximum value of 7 can be set, n must also be 7 or less. A value of 0 is also valid.
 n indicates the number of fractional bits used to represent subpixel precision; higher values of n improve the accuracy of subsequent bi-linear interpolation and are advantageous for image quality.

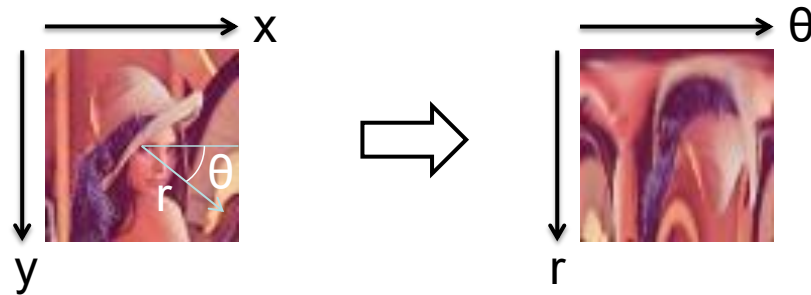


Figure 44 Polar Transfer

6.2.3. Spherical Transformation

- When projecting 3D information, it is necessary to convert the data to screen coordinates, similar to 3D graphics rendering. Corresponding mapping data must be prepared in advance.
 For example, to create an image projected onto a hemispherical surface with radius L , the mapping data should be configured such that the sampling intervals are wider near the center of the hemisphere and become denser toward the boundary.
- The variation in sampling interval corresponds to the relative distance from the defined intermediate coordinates. This relative coordinate $(\Delta X, \Delta Y)$ can be derived from the distance (dX, dY) between a given point and the center of the hemisphere, using the following expressions.
 If the condition $dX^2 + dY^2 > L^2$ is met, the mapping is not applied, and the mapping data is set to zero ('0').

As described in the section on polar coordinate transformation, it is recommended to ensure at least 4 bits of fractional precision to improve accuracy.

$$\Delta X = dX \cdot \cos^{-1} \frac{\sqrt{dX^2 + dY^2}}{L} \div 2\pi \quad \Delta Y = dY \cdot \cos^{-1} \frac{\sqrt{dX^2 + dY^2}}{L} \div 2\pi$$

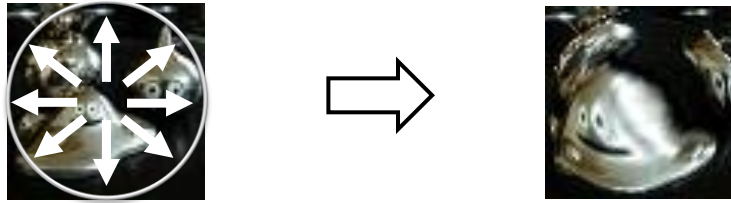


Figure 45 Crystal Ball Transfer

6.2.4. Free-form Deformation

- By generating coordinate mapping data for each screen pixel, a wide range of free-form deformations can be achieved.
When applying free-form deformation, special attention must be paid to how unmapped regions are handled, as well as to mitigating visual artifacts that may arise from overly sparse or dense sampling.
- Unmapped areas should be handled by intentionally modifying the mapping data so that they result in a specific color. If no special handling is applied, the default behavior is to use the PixelDefault value.

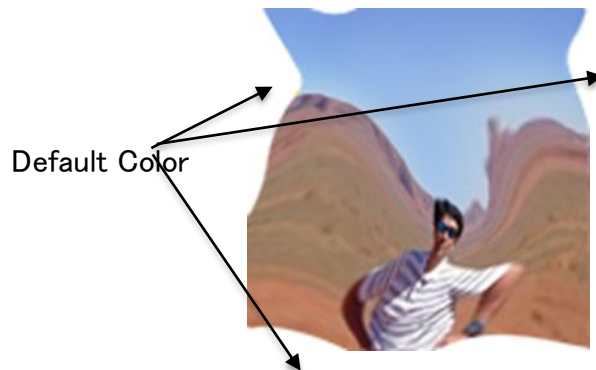


Figure 46 Free Deformation

- Sampling density can be determined by examining the distance between adjacent coordinate

In the mapping data used during coordinate transformation. Larger distances indicate sparser sampling, which increases the likelihood of aliasing artifacts.

6.2.4.1. Abstraction

- Image abstraction (Texture conversion) can be achieved by reading from a texture based on the luminance and position of the image. This processing can be performed on either the Source or Destination side. Since it is handled by the *Remapper*, coordinate mapping cannot be used in this case.
- The example shown below demonstrates a one-pass process in which the original image is abstracted in the Destination path, edges are extracted in the Source path, and text is composited using the Blender.



Figure 47 Cartooned Picture

- Unlike standard texture mapping described in the polygon rendering section, this technique selects tile patterns based on luminance values and samples them using intermediate coordinates. This enables expression of luminance with reduced data—similar to dithering. For example, it is equivalent to binary halftoning used in newspaper photographs.
- The presence of texture processing is controlled using `SrcMapInfo.Div` or `DstMapInfo.Div`, while the texture size N is specified using `SrcMapInfo.Prec` or `DstMapInfo.Prec`.
The textures are stored in memory, and N^2 -sized texture tiles are prepared for 256 possible luminance levels I (up to a maximum of 64K textures).
Texture formats are freely selectable, ranging from grayscale to full color.
- Prior to texture access, the Source Remapper or Destination Remapper generates new coordinates. The original image is read, a grayscale element is selected and used as the luminance value I , which becomes the new Y

coordinate. Meanwhile, the intermediate coordinates (X,Y)(X, Y) are packed to form the new X coordinate.

The bit width of luminance **I** (either 8-bit or 16-bit) is selectable via SrcMapInfo.Exp or DstMapInfo.Exp.

- To ensure the packed intermediate coordinates (X,Y)(X, Y) do not exceed the texture size N^2 , a coordinate mask width **N** (ranging from 1 to 256) is specified using SrcMapInfo.Prec or DstMapInfo.Prec.
- Finally, for the SrcIn system, texture-related information must be set in SrcInInfo and SrcInBase.

For the SrcOut system, the corresponding settings are made in SrcOutInfo and SrcOutBase.

Using the newly computed coordinates, the texture data stored in memory is accessed and read accordingly.

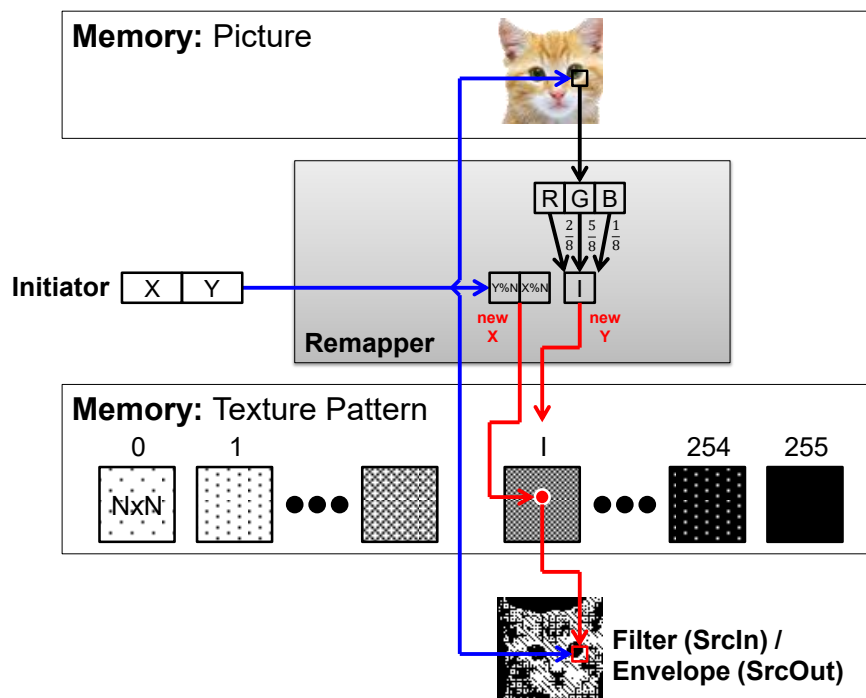


Figure 48 Texture Value Selection

6.2.5. Affine Transformation

6.2.5.1. Parameter Settings

- Matrix transformation is used for 2D image transformations. Operations such as translation, scaling, rotation, and other coordinate transformations are applied to the Source image.

These transformations are achieved by modifying the values in the transformation matrix and can also be combined with coordinate mapping.

$$\begin{pmatrix} X_s \\ Y_s \end{pmatrix} = \begin{pmatrix} X_t/Z_t \\ Y_t/Z_t \end{pmatrix}, \begin{pmatrix} X_t \\ Y_t \\ Z_t \end{pmatrix} = \begin{pmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{pmatrix} \begin{pmatrix} X_{rs} \\ Y_{rs} \\ 1 \end{pmatrix}$$

- All parameters must be specified in single-precision floating-point format. Negative values are allowed; however, special values such as NaN (Not-a-Number) and Infinity (∞) are not permitted.
The final computation result is converted to a fixed-point format with 4 bits of fractional precision and passed to the 2D Filter.
- The following section provides simple examples of matrix transformations. As previously mentioned, these operations can be combined by appropriately configuring the matrix values.

6.2.5.2. Translation

- To apply a translation to the Source image, use the following matrix in the transformation process.

Here, dx represents the translation distance in the X-axis direction, and dy in the Y-axis direction.

From the perspective of the Destination coordinates, positive values cause the Source image to shift to the right (X-axis) or downward (Y-axis), assuming the monitor origin is located at the top-left corner.

$$\begin{pmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{pmatrix}$$

6.2.5.3. Mirroring (Flip)

- To horizontally flip the Source image, apply the following matrix in the transformation process.

Wx denotes the width of the image.

$$\begin{pmatrix} -1 & 0 & W_x - 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- Similarly, to vertically flip the Source image, apply the following matrix in the transformation process.

W_y denotes the height of the image.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & W_y - 1 \\ 0 & 0 & 1 \end{pmatrix}$$

6.2.5.4. Scaling

- To scale the Source image, apply the following matrix in the transformation process.

M_x and M_y represent the scaling factors in the X and Y directions, respectively.

A value of 1.0 indicates no scaling (1:1), values greater than 1.0 perform downscaling, and values less than 1.0 perform upscaling.

The transformation is applied with the image origin at coordinate (0, 0).

$$\begin{pmatrix} M_x & 0 & 0 \\ 0 & M_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

6.2.5.5. Rotation

- To rotate the Source image around its center ($W_x/2, W_y/2$), apply the following matrix in the transformation process.
 θ represents the rotation angle; a positive value indicates counterclockwise rotation, while a negative value indicates clockwise rotation.

$$\begin{pmatrix} \cos\theta & -\sin\theta & dx \\ \sin\theta & \cos\theta & dy \\ 0 & 0 & 1 \end{pmatrix}$$

At this point,

$$dx = \frac{W_x}{2}(1 - \cos\theta) + \frac{W_y}{2}\sin\theta$$

$$dy = -\frac{W_x}{2}\sin\theta + \frac{W_y}{2}(1 - \cos\theta)$$

•When rotation is combined with scaling, the transformation is applied as shown below.

Since the transformation is based on the Destination coordinates, note that the matrix computation is inverted relative to the usual Source-based transformation.

$$\begin{pmatrix} \cos\theta & -\sin\theta & dx \\ \sin\theta & \cos\theta & dy \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} Mx & 0 & 0 \\ 0 & My & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} Mx \cdot \cos\theta & -My \cdot \sin\theta & dx \\ Mx \cdot \sin\theta & My \cdot \cos\theta & dy \\ 0 & 0 & 1 \end{pmatrix}$$

At this point,

$$\begin{aligned} dx &= \frac{Wx}{2} (1 - Mx \cdot \cos\theta) + \frac{Wy}{2} My \cdot \sin\theta \\ dy &= -\frac{Wx}{2} Mx \cdot \sin\theta + \frac{Wy}{2} (1 - My \cdot \cos\theta) \end{aligned}$$

•When rotation results in regions that have no corresponding Source image pixels, the generated pixels are determined according to the specified edge handling method. Depending on the configuration, the system may assign the PixelDefault value, use the nearest neighboring pixel, or apply a mirrored (wrapped) pixel. These edge handling choices have no impact on performance.

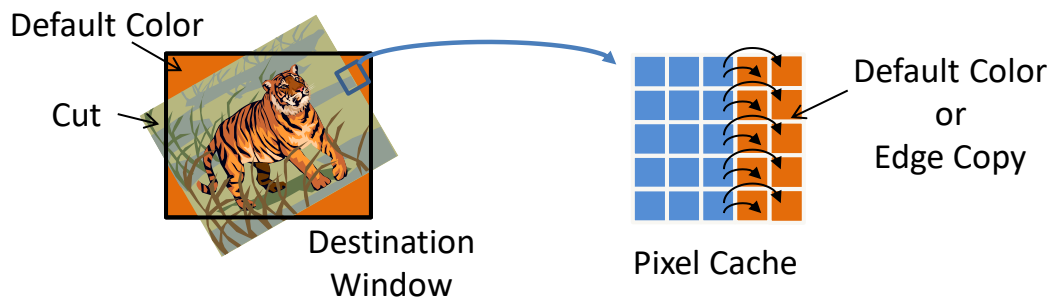


Figure 49 Rotation and Edge Correction

6.3. Image Attributes

6.3.1. Input Format

- *frComp* provides three pixel value input interfaces: SrcIn, SrcOut, and DstIn. Each interface supports freely selectable pixel formats. Additionally, both SrcIn and SrcOut include circuits that approximate grayscale

values from RGB inputs.

Based on the selected format, the grayscale result can be assigned either to element A alone or to all elements.

- To handle memory endianness adjustments as well as element reordering and duplication, swap settings are available via SrcInInfo.Swap, SrcOutInfo.Swap, and DstInInfo.Swap.

The swap circuitry allows arbitrary 8-bit segments to be remapped to any 8-bit storage positions.

After swapping, data is converted into an internal 4-element representation according to the configured format.

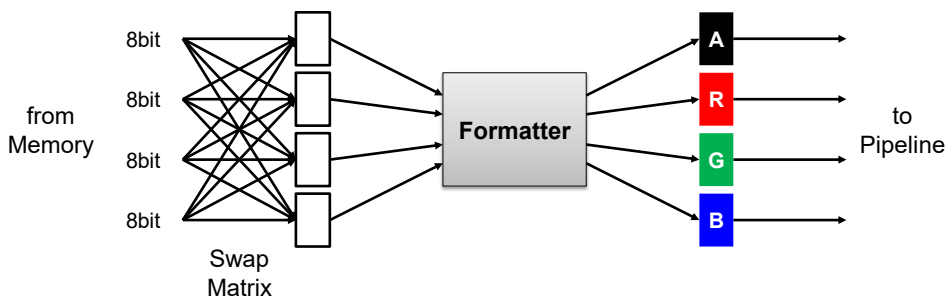


Figure 50 Source Swap and Formatter

- The grayscale values generated in the SrcIn and SrcOut paths can be copied to either element A or all elements, depending on the format settings (SrcInInfo.Format/Swap, SrcOutInfo.Format/Exp, DstInInfo.Format/Swap). Grayscale generation assumes the input pixel is in the RGB color space; therefore, it is not applicable to pixels in other color spaces. For YUV formats, the luminance component Y is used instead of a grayscale value and can be copied to element A or all elements (configurable via format or swap settings).
- Grayscale values are approximately generated using the formula below. If precise grayscale values are required rather than approximations, a grayscale image must first be generated using a 3D CLUT.

$$\text{Gray} = \frac{2\text{Red} + 5\text{Green} + \text{Blue}}{8}$$

- If a grayscale image is already available, it should be read using an **8 Bpp** format. In this case, the grayscale value is assigned to all elements. For images where the grayscale value is embedded in element A, use a **32 Bpp** format. In this case, by default, only element A will be assigned the grayscale

value.

To assign the grayscale value to all elements, use the Swap setting to replicate it across elements.

6.3.2. Output Format

- *frComp* provides a pixel value output interface via the DstOut path. Unlike input processing, Swap and format settings (DstOutInfo.Format/Swap) are applied in the reverse direction. There is no built-in grayscale generation function in the output stage.

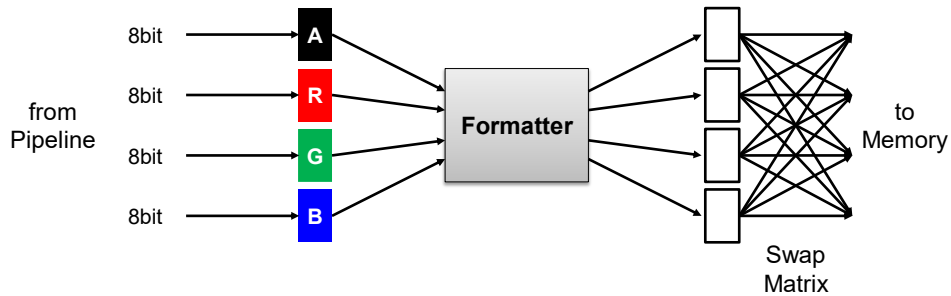


Figure 51 Destination Swap and Formatter

6.3.3. Width and Address

- The input image width and base address must be configured separately for the SrcIn, SrcOut, and DstIn paths. Image attributes are fully defined by specifying the pixel-stride (address update interval in pixels) using SrcInInfo.Stride, SrcOutInfo.Stride, DstInInfo.Stride, and DstOutInfo.Stride; the starting address using SrcInBase.Addr, SrcOutBase.Addr, DstInBase.Addr, and DstOutBase.Addr; and the image format as previously described.
- The upper bounds of the input image are specified for edge handling using SrcSize and DstSize, while the lower bounds are fixed at zero. Refer to the matrix transformation examples for more details on edge handling behavior.

6.3.4. Attribute Conversion

- Image attribute conversion—such as format, width, and address—can be executed by configuring the DstIn (SrcIn) and DstOut paths and performing a BitBlt (Bit Block Transfer) operation.

Image offsets for BitBlt are configured via SrcOffset and DstOffset.

- Avoiding use of the SrcIn path reduces consumption of external resources such as caches, and is therefore advantageous for executing multiple tasks in parallel.

On the other hand, using the SrcIn path enables richer image processing features such as scaling and alpha blending.

Select the appropriate configuration based on the application requirements.

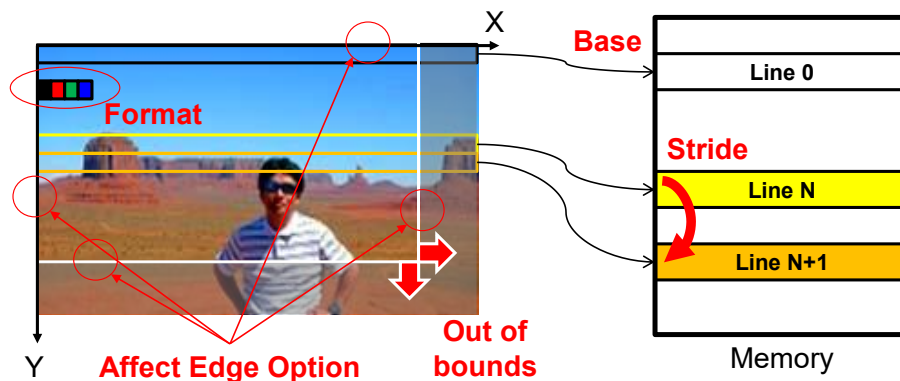


Figure 52 Property of Picture

6.4. Filter Settings

6.4.1. Filter Selection

- For processing the Source image, only one filter can be selected from the following seven types: **2D/2F**, **Non-linear**, **Mask**, **Hamming**, **Extrema**, **SAD/SSD**, and **Bitmap**.

To disable filtering entirely, set all bits of FilterCntlIn.En to '0'.

- For processing the Destination image, only the **Pattern** filter is available. To disable filtering, set all bits of FilterCntlOut.En to '0'.
- The size of the pixel cache must be configured based on the selected filter and kernel size.

Some filters impose restrictions on usable kernel sizes. Larger kernels increase memory access frequency, potentially degrading performance.

- Depending on the kernel size, it may not be possible to assign arbitrary coefficients to all elements.

For example, a 5×5 filter requires 100 coefficients to support 4 elements, but only 27 coefficients can be supplied through the Command List. If the filter exhibits symmetry, such as in a Gaussian filter, 24 coefficients may be sufficient to cover all 4 elements, allowing per-element coefficient assignment.

- When using **Blut**, it is possible to provide filter coefficients as follows:
 5×5 kernel \rightarrow 4 elements,
 7×7 kernel \rightarrow 2 elements,
 9×9 kernel \rightarrow 1 element.

6.4.1.1. 2D / 2F Filter

- In addition to low-pass and high-pass filters, general-purpose filters such as Sobel and Laplacian can be used.
Coefficients are supplied in half-precision floating-point format.
2D Filter performs fixed-point operations on all 4 elements simultaneously, while **2F Filter** performs half-precision floating-point operations on a single element.
- In the **2D Filter**, coefficients are scaled using a factor of 2^n to maximize dynamic range and improve precision, ideally bringing values close to ± 2.0 . Final scaling is applied using 2^{-n} (specified in `FilterCoef13.Scale`).
A pixel value of 256 is internally normalized to 1.0.
Output values are clipped to the range $(-1.0, +1.0]$.
Value handling options, such as absolute value or negative suppression, can be configured via `FilterCntl.Op`.
- In the **2F Filter**, the results remain in half-precision floating-point format. Since downstream blocks like **Envelope** and **Clut** operate on 8-bit paths, conversion to 8-bit output must be specified when using these functions (`MasterCntl.Inword`).
While Blender's standard 8-bit features cannot be used, its floating-point accumulator remains available.
- If negative coefficients are used, the resulting output may be negative. Negative values are retained up to the Blender stage; however, when using the 2D/3D mode of the 3D Clut, adjustment is required (`MasterCntl.Inword`) because this mode does not support negative values.
In such cases, subtraction of a constant in the Blender stage is necessary.
- For kernel sizes of 2×2 or smaller, four coefficient sets can be selected per element.
For 3×3 kernels, three sets can be selected per element.
For 4×4 and larger kernels, only one coefficient set is supported, but options

such as transposition or constant values (All 1.0 or All 0.0) can be specified per element.

- For example, in morphological operations, using a 3×3 kernel: dilation coefficients can be assigned to element B, and erosion coefficients to element G, enabling simultaneous dilation and erosion.

6.4.1.2. Arbitrary Coefficients and Interpolation

- When combining arbitrary coefficients with interpolation, the required kernel size must be one level larger.
For example, a 3×3 coefficient set requires a 4×4 kernel.
This adjustment is handled automatically by *frComp*, but limitations on coefficient selection should be noted in advance.
- Only combinations with kernel sizes of 1×1 or 3×3 are supported, and only **bi-linear interpolation** is available (bi-cubic is not supported).
Note that when interpolating 3×3 coefficients, the internal kernel becomes 4×4 , reducing the selectable coefficient sets from 3 to 1.
- Higher-order interpolation yields smoother images.
The figure below shows results of enlarging a 4×4 repeated image to 64×64 using different modes.
Interpolation is anchored at the origin (0,0) of the original coordinate space.
For example, if an image is enlarged by an integer factor N and then downsampled by $1/N$, it will be restored to the original regardless of interpolation mode.
If the interpolation grid needs to be centered between pixels, a translation must be applied via matrix transformation.
- The sampling relationship between the input and output image is governed by the affine transformation matrix.
If the scale factor is a ratio of integers, the same source pixel may map to multiple output pixels according to the greatest common divisor.
For instance, scaling by $1/2$ will map 2 input pixels to 1 output pixel, potentially causing aliasing.
To avoid this, apply a fractional offset (e.g., ± 0.5) in the matrix to shift sampling points.

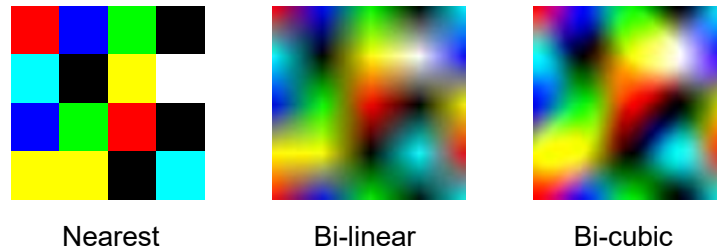


Figure 53 Comparison of Interpolation Mode

6.4.1.3. Sobel Filter

- The Sobel filter, commonly used for edge detection, is typically applied to grayscale images (refer to relevant literature for detailed background). It is assumed that the grayscale values are stored in element A.
- A horizontal filter is applied to element B, and a vertical filter is applied to element R.
Representative 3×3 coefficients are shown below.
Note that the actual coefficients are normalized by multiplying by $1/2$ so that their values remain below 2.

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

- When applying a pre-processing filter such as an averaging or Gaussian filter, and the filter size is 3×3 or smaller, its coefficients can be pre-multiplied (superimposed) onto the Sobel filter coefficients.
The example below shows a new set of 5×5 coefficients created by superimposing a 3×3 Gaussian filter onto the Sobel filter.
- Note that for 5×5 filters, arbitrary coefficients cannot be assigned per element; however, **transpose (diagonal symmetry)** can be specified individually for each element.
In this case, the configuration sets **no transpose for element B** and **transpose enabled for element R**.

$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \circ \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} = \frac{1}{16} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 2 & 8 & 12 & 8 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ -2 & -8 & -12 & -8 & -2 \\ -1 & -4 & -6 & -4 & -1 \end{bmatrix}$$

$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \circ \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = \frac{1}{16} \begin{bmatrix} 1 & 4 & 0 & -4 & -1 \\ 4 & 8 & 0 & -8 & -4 \\ 6 & 12 & 0 & -12 & -6 \\ 4 & 8 & 0 & -8 & -4 \\ 1 & 2 & 0 & -4 & -1 \end{bmatrix}$$

▪ As a result of the 2D Filter, element **B** contains the horizontally differentiated values, and element **R** contains the vertically differentiated values.

These are 9-bit signed values. To enable 2D transformation in the subsequent **3D CLUT**, they are converted to 8-bit format (**MasterCntl.Inword**).

▪ In the **3D CLUT**, a 2D transformation is performed using elements **R** and **B**, enabling non-linear processing.

The 2D transformation uses the upper 6 bits of each element (**R** and **B**), forming a 12-bit index to a 4K-word lookup table.

When preparing the table, take care to handle the signed nature of the 8-bit values by subtracting an offset of 32, and use it to compute the corresponding luminance values.

$$\text{Intensity} = \sqrt{R^2 + B^2}$$

```
for (r = 0; r < 64; r++)
  for (b = 0; b < 64; b++) {
    x = b - 32;
    y = r - 32;
    put(r, b, sqrt(x * x + y * y));
  }
```

- The **3D CLUT** can be configured by assigning the same value to all table entries and then performing post-processing in the **Blender**. Alternatively, the **Extractor** can be used to apply a threshold and binarize the result.

In such cases, a thresholded table may be pre-defined within the **3D CLUT** to simplify processing and generate the final image accordingly.

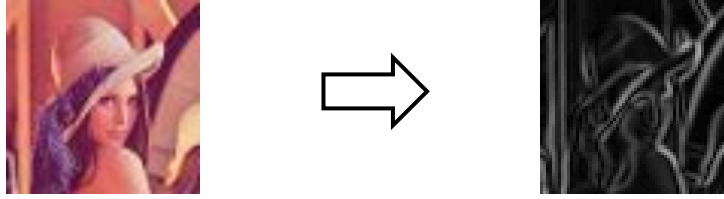


Figure 54 Sobel Filter Result

6.4.1.4. Canny Filter

- Another commonly used edge detection method is the **Canny filter** (refer to standard literature for detailed algorithms).
The Canny filter builds upon the output of the Sobel filter, applying further processing steps.
Due to the sequential nature of these operations, intermediate results must be written to memory, and the final result cannot be generated in a single pass like the Sobel filter.
Additionally, **hysteresis thresholding**, which handles edge continuity and intensity refinement, must be performed by a separate engine or the CPU.
- As a prerequisite, the **3D CLUT** used in conjunction with the Sobel filter must be extended to include **gradient regions** in addition to luminance values. These gradient regions are generated simultaneously with intensity values, using the **RB** elements resulting from the Sobel filter.

$$\tan \theta = \frac{R}{B}$$

if $-0.4142 < \tan \theta \leq 0.4142$ then 0

if $-2.4142 < \tan \theta \leq -0.4142$ then 1

if $2.4142 < |\tan \theta|$ then 2

if $0.4142 < \tan \theta < 2.4142$ then 3

- With the above settings, *frComp* is activated (Pass 1), and the results are temporarily written to memory.
The contents are essentially the same as those from the Sobel filter, but with gradient region information (ranging from 0 to 3) embedded into element **G**. Additionally, a Gaussian filter is applied concurrently with the Sobel filter.
- Next, **non-maximum suppression** is performed using the gradient region data. This thinning process is implemented using the **Mask Filter in Con mode**. Based on the gradient region information, comparisons are made between the

center pixel and its neighbors in a 3×3 kernel.

For each gradient region, a corresponding mask pattern is used:

- Gradient region 0 (horizontal): 0x000CE000 (compare left and right)
- Gradient region 1 (diagonal / \searrow): 0x00C00E00
- Gradient region 2 (vertical): 0x0C0000E0
- Gradient region 3 (diagonal / \swarrow): 0xC000000E
- Among the per-pixel operations, 0xC and 0xE evaluate to true when the center pixel is greater than its neighbor.
0x0 always evaluates to true.
0xE includes equality to avoid suppressing pixels with equal gradient values.
Note that the mask group index shown in the figure excludes the center pixel; the mask pattern is generated based on this index.
To ensure the center pixel value is not a factor in evaluation, FilterInMask should be set to 0xFFFFFFFF.



Figure 55 Mask Pattern

- When *frComp* is executed with the above settings (**Pass 2**), a grayscale output is obtained.

In this pass, approximate binarization can be achieved by configuring a threshold in the **Extractor** block.

The examples below illustrate simple thresholding (Threshold = 40) and hysteresis thresholding (Low = 32, High = 48).

- The latter (hysteresis thresholding) provides better results, with reduced noise and improved edge continuity.

However, the former also provides a reasonably accurate approximation suitable for many use cases.

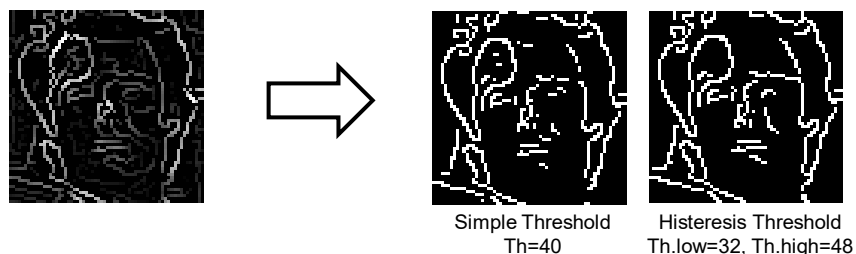


Figure 56 Canny Filter Result

6.4.1.5. Bilateral Filter

- The **Bilateral Filter** is used when edge-preserving smoothing is required. It is a type of non-linear filter that, during Gaussian-based smoothing, modulates the influence of neighboring pixels based on their difference from the center pixel—thereby preventing edge blurring. The influence (weight) on the filter coefficients is generally defined according to a **Gaussian distribution**.
- The Gaussian filter coefficients used within the **2D Filter** are not described here.
The coefficient weights based on pixel differences (following a Gaussian distribution) are determined using the sharpness parameter σ and are configured via the **Blut** table.
Since **Blut** is also used by the **Blender**, it must be managed exclusively. Additionally, accessing the Blut table is circuit-intensive and may not be implemented in certain cases.
In such cases, predefined Gaussian patterns for various σ values can be selected via FilterCntl1.Value.
- To enable bilateral filtering, set FilterCntl1.Op[1:0] = 2 in the filter control register.
The pixel component used to evaluate differences (for weight computation) is specified by FilterCntl1.Op[3:2].
Typically, SrcInInfo.Format and Exp are configured so that grayscale values (automatically computed from RGB) are stored in element **A**, and '3' is set to select that component.
- While the 2D Filter's Gaussian coefficients remain constant, the **scaling coefficients (weights)** vary per pixel depending on the pixel differences. Therefore, **per-pixel normalization** must be applied.
Normalization involves the following two operations:
 - Set FilterCntl0.InForce[7:6] = 1 to assign a fixed value of 0xFF (1.0) to element **A**.

After filtering, the sum of the weights becomes the new value of element A (i.e., the filter energy).

- In the **Blender**, divide elements RGB by element A.

This operation serves as the normalization step.

- Below is an example of Gaussian coefficients for a 5×5 **2D Filter**.

The effective coefficients are automatically modulated based on the configured σ value and the difference between the center and surrounding pixels. The available σ values range from 0.75 up to 48 in powers of two.

Figure 60 shows the relationship between pixel intensity difference and weight scaling for σ values of 6, 12, 24, 48, and 96.

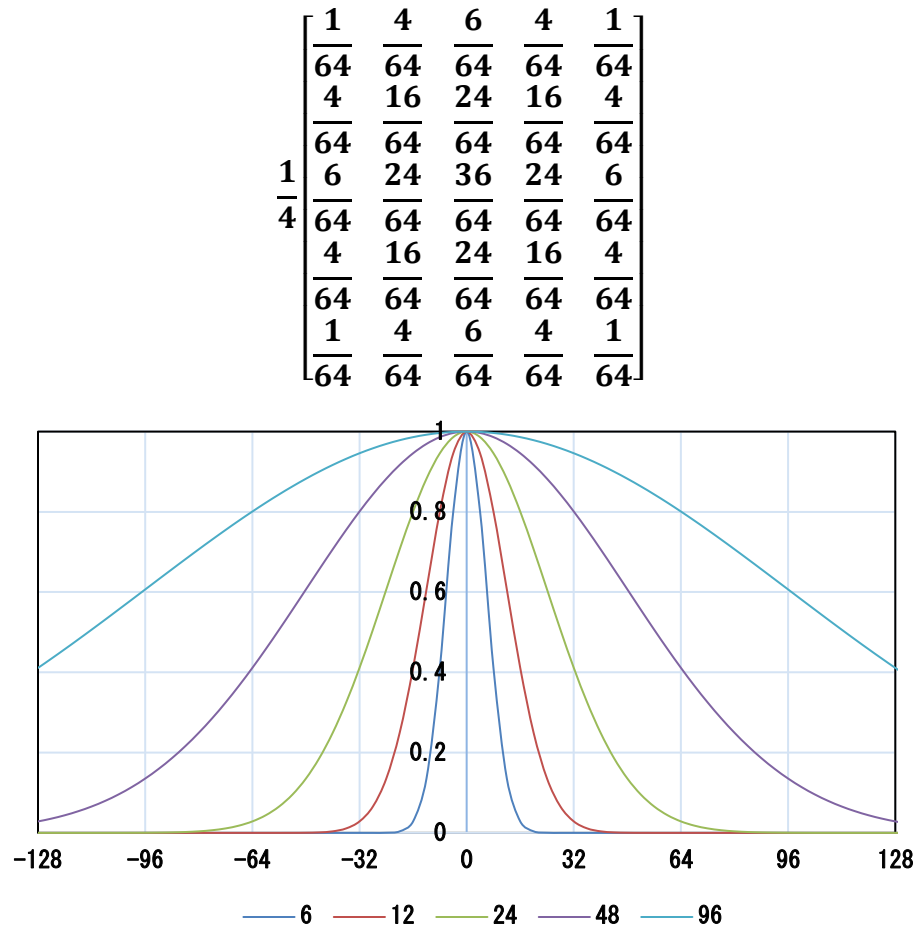


Figure 57 Modeified Normal Distribution

The following images are shown for comparison: the original image, results with FilterCntl1.Value set to 4 ($\sigma = 6$) and 6 ($\sigma = 24$), and an image processed using only the 2D Filter.

While the results also depend on the specific coefficients used in the 2D Filter, it can

be observed that fine patterns are suppressed when $\sigma = 24$ —in contrast to a standard Gaussian filter, which causes the entire image to appear blurred.

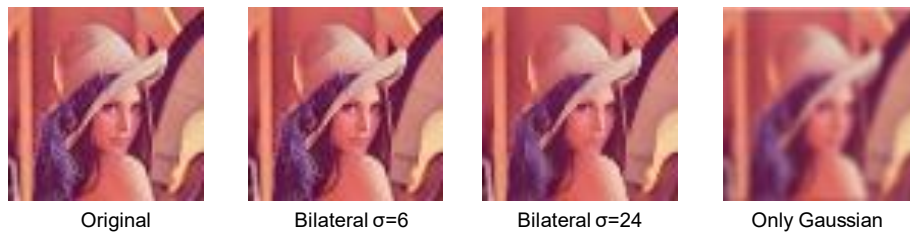


Figure 58 Bilateral Filter Results

6.4.1.6. Cross-Correlation

- Cross-correlation between images can be performed by dynamically assigning another image as the coefficient set. This uses a **5 × 5 2D Filter**.

The correlation result is written to memory as pixel data.

By storing results with 16-bit precision and accumulating across multiple passes, correlations larger than 5×5 can be computed.

- Set the **SrcIn** path to the base image and the **SrcOut** path to the reference image.

While various formats can be used for the reference image on the SrcOut path, only element **B** can be used as a coefficient.

For example, if using a 32 Bpp format image (SrcOutInfo.Format = 3), setting SrcOutInfo.Exp = 3 allows grayscale values to be mapped to all elements, which can then be used.

To reference specific elements, modify SrcOutInfo.Swap (only valid for 32 Bpp formats).

- The reference image on the SrcOut path can be addressed either **relatively** to the SrcIn image or **absolutely** using coordinates set in the **Destination Remapper**.

In relative mode, for each pixel in the base image, a 5×5 region in the reference image is convolved.

By shifting this relative position and accumulating the results, local correlations with surrounding pixels can be computed.

The relative position is set using DstOffset.

- Additionally, the indexing method for the SrcOut image can be assigned to separate coordinates from those used for the SrcIn image.

The **SrcIn** image is scanned using coordinates (X, Y), while the **SrcOut** image is scanned using (Z, W).

This is useful when evaluating correlation against a specific pattern indicated by (Z, W).

To shift patterns every 5×5 block (or simplified 4×4), use the **Destination Remapper**.

- The processing time for full cross-correlation over large images is proportional to the **square of the number of pixels**.

For example, if both source coordinates (X, Y) and reference coordinates (Z, W) are scanned using pss—at 4×4 granularity ($1/16$), the minimum required cycles would be $(X_{\max} \times Y_{\max} \times Z_{\max} \times W_{\max}) / 16$.

Therefore, caution is advised when handling large images.

- For **normalization**, the energy of the filter region must be computed for both the base and reference images.

To compute the energy of the SrcIn image, set all values of the SrcOut image to 1.0.

Conversely, to compute the energy of the SrcOut image, set all values of the SrcIn image to 1.0.

The following is an example of grayscale image correlation using a 5×5 filter:

- Configure both SrcIn and SrcOut paths to reference grayscale images. (For SrcIn, use a format and exponent setting so that all ARGB elements are mapped to grayscale: SrcInInfo.Format, SrcInInfo.Exp)
- Set FilterCntl1.Sel[5:4] = 3 so that element **R** uses coefficients all set to 1.0, and computes the 5×5 energy of the SrcIn image.
- Set FilterCntl0.DataSel[1:0] = 3 so that element **B** uses values all set to 1.0, and computes the 5×5 energy of the SrcOut image.
- The result of the cross-correlation is written only to element **A** (element **G** is unused).
- Use **2D mode** of the **CLUT** (details described later) to combine the energy values from elements **R** and **B**.
Load the precomputed combination values into the CLUT so that the total energy is output to element **B**.
- As in bilateral filtering (though the element roles differ), the **Blender** divides the value in element **A** (correlation result) by the value in element **B** (total energy), yielding the final normalized result.

6.4.1.7. Thinning

- **Thinning** is an iterative process in which center pixels are removed based on the states of surrounding pixels. The process continues until no further deletions are possible.

The **Mask Filter** is used to convert the surrounding pixel states into an index, which is then used to look up whether or not the center pixel should be deleted.

The following describes the behavior in conjunction with *frComp*.

- Only non-zero pixel values are eligible for deletion. These target pixels are selected using FilterInColor and FilterInMask.

First, FilterInMask is used to extract the relevant bits to be referenced.

Then, FilterInColor defines the pixel value that qualifies for evaluation.

For example, to target pixels where element B equals 0xFF, set the following:

- FilterInColor = 0x000000FF
- FilterInMask = 0xFFFFFFFF00

The target condition (**Hit**) is determined using the expression:

$$\text{Hit} = ((\text{ARGB} \& \sim 0xFFFFFFFF00) == (0x000000FF \& \sim 0xFFFFFFFF00))$$

- The surrounding pixels in the 3×3 region around the center pixel are numbered from the top-left to the bottom-right, excluding the center pixel itself.

As a result, pixel indices range from 0 to 7.

- Similar to the target pixel condition described above, the surrounding pixels are evaluated using FilterOutColor and FilterOutMask.

For each surrounding pixel that matches the target value, the corresponding bit (by index) is set to 1.

These bits are then packed into an 8-bit index, which is used to access the

FilterTable and retrieve a 1-bit evaluation result (Eval), as shown below:

$$\text{Judge}[i] = ((\text{ARGB}[i] \& \sim \text{FilterOutMask}) == (\text{FilterOutColor} \& \sim \text{FilterOutMask})), \text{ where } i = 0-7$$

$$\text{Eval} = \text{FilterTable}[\text{Judge}]$$

- Based on the four possible combinations of Hit and Eval, the behavior for the center pixel—whether to delete it or not—can be configured.

In this case, the center pixel is deleted **only when both Hit = 1 and Eval = 1**.

Deletion is performed by replacing the pixel value using FilterReplaceColor.

If FilterReplaceColor = 0, the pixel is rendered black in RGB representation.

- The configuration of the **FilterTable** should be based on standard thinning algorithms (refer to relevant literature).

For example, to thin in a diagonal direction from the top-left to the bottom-right, construct the table to satisfy specific deletion patterns.

(Note: This is just one of many possible implementations.)

Judge[0] & !Judge[2] & Judge[3] & !Judge[4] & Judge[5] & !Judge[7]

!Judge[1] & !Judge[2] & Judge[3] & !Judge[4] & Judge[5] & Judge[6]

!Judge[0] & !Judge[1] & !Judge[2] & Judge[5] & Judge[6] & Judge[7]

0x01110101_00510000_00000000_00500000_00000000_00000000_00000000_00000000

- Additional deletion patterns should also be prepared for the following directions: bottom-right to top-left, top-right to bottom-left, and bottom-left to top-right.
These directional thinning passes are executed in sequence, forming a cycle. When no further pixels are deleted in any pass, the thinning process is considered complete.
- The FFD bit in the **Info register** is cleared to '0' at the start of *frComp* execution.
If any pixels are deleted by the Mask Filter during processing, FFD is set to '1'. Therefore, thinning is considered complete when the FFD bit remains at '0'.
- Alternatively, you can check **Word 1 of the context** for the thinning status.
If the value is 0xFFFFFFFF, it indicates that at least one pixel was deleted in the previous cycle.
- Even if processing continues beyond the point where the termination condition is met, the result will remain unchanged.
The accompanying figure shows an illustrative example in which the outermost pixels are retained in red for explanatory purposes.

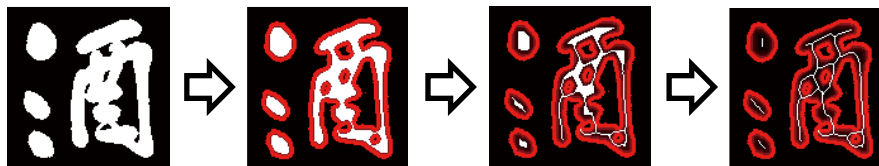


Figure 59 Thinning Process

- The figure below shows examples of thinning results using different methods.
In the first example, based on the initially described table, the process converges in **16 iterations**.
In contrast, the alternative table example converges in **10 iterations** and also produces fewer spurious branches (“hairs”).

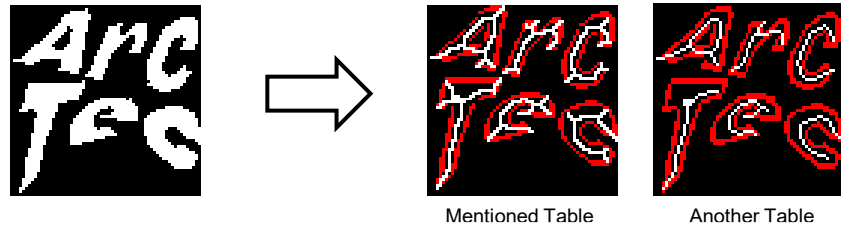


Figure 60 Difference of Thinning Table

6.4.1.8. Scratch Correction

This section describes a simple method for performing scratch correction using frComp, without requiring advanced processing. Pixels identified as scratches are complemented using surrounding pixels via the Mask Filter. For example, as shown in the figure below, scratch-like red lines can be removed.

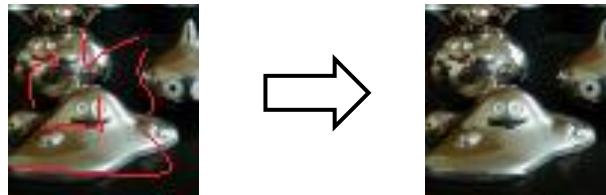


Figure 61 Scratch Correction

- It is assumed that scratch detection has already been performed using methods such as morphological operations or specific color extraction, which are described later. The detection result should be embedded into the target image by setting 0xFF to Element A as a marker for detected pixels. Configure FilterInColor and FilterInMask so that processing is applied only when Element A is 0xFF. Refer to the thinning example for further details.
- The Mask Filter should use the MaskB mode, and the Lookup Table (Blut) should be configured with substitution pixel reference indices corresponding to all possible 3x3 scratch patterns. For example, in the 3x3 grid shown below, if the red regions represent scratches, excluding the center, the binary reference index would be 01000001. This value is used as the address, and the correction reference index 00010000 is set at Bit 4 of the corresponding Blut entry. This configuration must be applied for all 256 possible patterns.

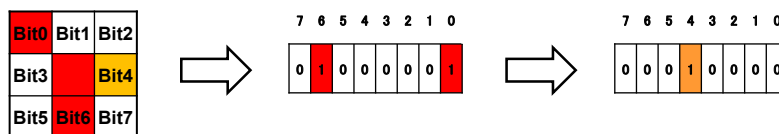


Figure 62 Pattern and Selection Map

- Since the reference index is expressed as a vector, multiple references can be specified. When multiple references are used, the resulting pixel value is the average obtained by dividing the sum of the referenced pixel values by the number of references. As the number of references increases, the behavior approaches that of a low-pass filter, resulting in a more blurred image. For scratch correction, it is recommended to minimize the number of references to preserve image sharpness.
- The FilterTable is used to configure exception handling. When all pixels in the 3×3 grid are either scratches or non-scratches, only reference indices 0 and 255 should be set to '1' so that the original image is referenced. Additionally, set FilterCntl1.Op to '2' so that pixel replacement occurs only when the center pixel is marked as a scratch and the evaluation value (used as an index to the FilterTable) is '0'. The default replacement value FilterReplaceColor is not used. The overall process is illustrated in the diagram below.

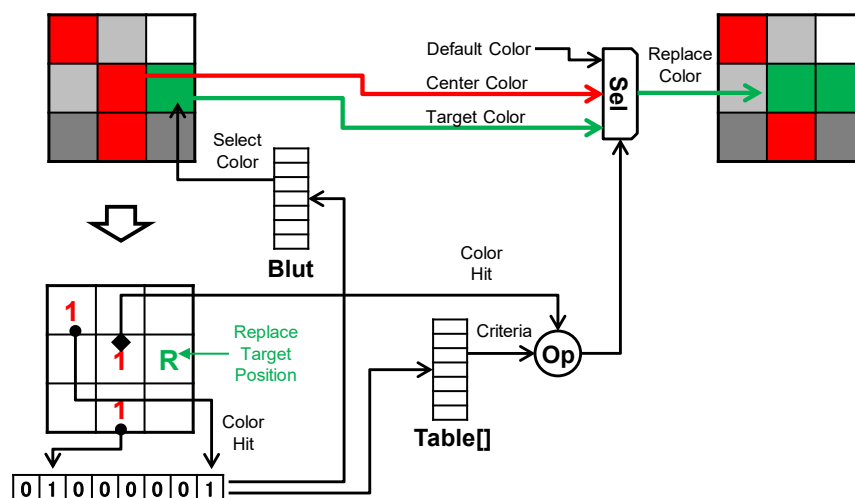


Figure 63 Replace Process

- If a scratch has a width of more than one pixel, multiple passes—similar to thinning—are required. The termination condition is determined by checking either the Info register's FFD flag or Word 1 of the context. However, since the correction is only an approximation using neighboring pixels, the wider the scratch, the more distortion will appear in the output image.

6.4.1.9. Morphological Operations

- Dilation and erosion are performed using operators defined by coefficient matrices of arbitrary sizes up to 5×5 . The filter result is controlled using thresholding via the Extractor (this can also be done using a 3D Clut).
- The operator coefficients must be evenly distributed such that their total does not exceed 1.0 (0x100 in fixed-point representation). In dilation, the result is activated (i.e., a dot is set) if any non-zero result is obtained. In erosion, the result is activated only when the sum equals the total of the operator coefficients (assuming pixel values are either 0 or 1.0).

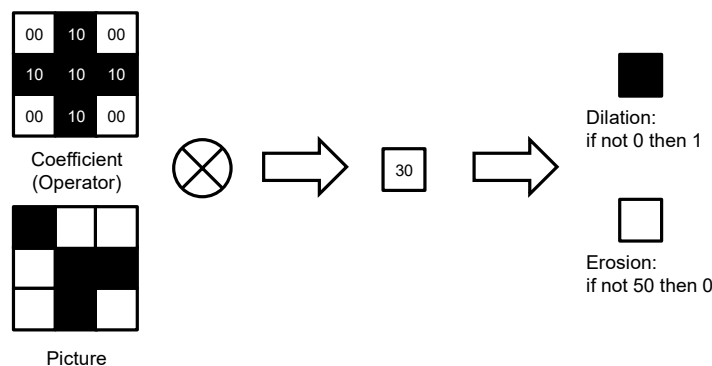


Figure 64 Morphological Operation

- For kernels up to 3×3 , different filter coefficients can be set per element, allowing dilation and erosion to be performed simultaneously for each element (coefficients are often the same). For example, when generating the difference between Closing (dilation followed by erosion) and Opening (erosion followed by dilation), you can first apply both dilation and erosion simultaneously to Element GB and write the result to memory. Then, apply erosion and dilation in the reverse order and compute the absolute difference using the Blender to obtain the final result.

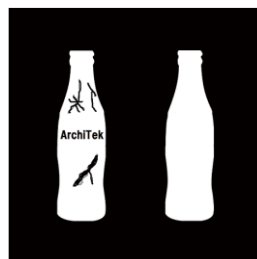


Figure 65 Dilation Result

6.4.1.10. Feature Point Extraction

- Multiple grayscale images are generated using Gaussian filters with different σ values (image pyramid), and candidate feature points can be extracted by detecting local maxima and minima using the Extrema Filter. The coordinates of these candidate points are then written to memory via the Steal function, enabling more precise analysis to identify true feature points.
- The Extrema Filter operates only within a single scale level of the image pyramid. Four grayscale images (corresponding to four levels) must be packed into one word, with Element B representing the lowest level and Element A the highest. When constructing the image pyramid, Gaussian filters with different σ values can be applied to each element, enabling simultaneous processing for improved performance when possible.
- If the image pyramid includes more than four levels, the data must be split into two sets. Levels 0 through 3 should be loaded via the SrcIn path, while levels 4 through 7 should be loaded via the SrcOut path.

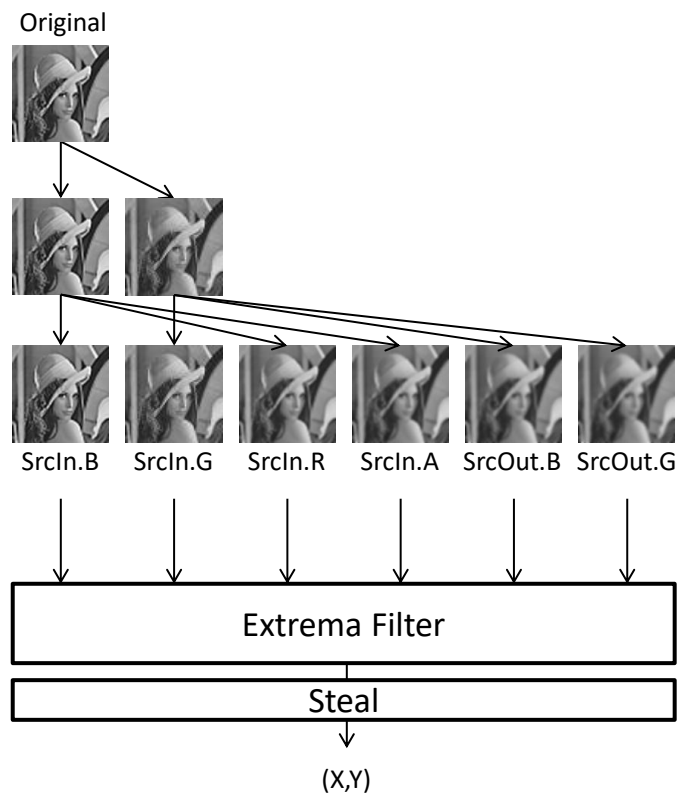


Figure 66 Feature Points Extraction

6.5. Clut Configuration

6.5.1. Effects of Transformation

- The primary purpose is to perform color space conversion. Depending on the number of elements used, the Clut operates in 1D, 2D, 3D, or binary mode.
 - In **1D mode**, each element is converted individually.
 - In **2D mode**, a transformation is applied based on Elements R and B.
 - In **3D mode**, a transformation is applied using Elements R, G, and B to output four elements.
 - In **binary mode**, a boolean result is generated from Elements R, G, and B.
- By selecting the appropriate mode according to the use case, Clut settings can consolidate color space-related processing. Additionally, Clut can be utilized not only for color space conversions but also for various types of computation.

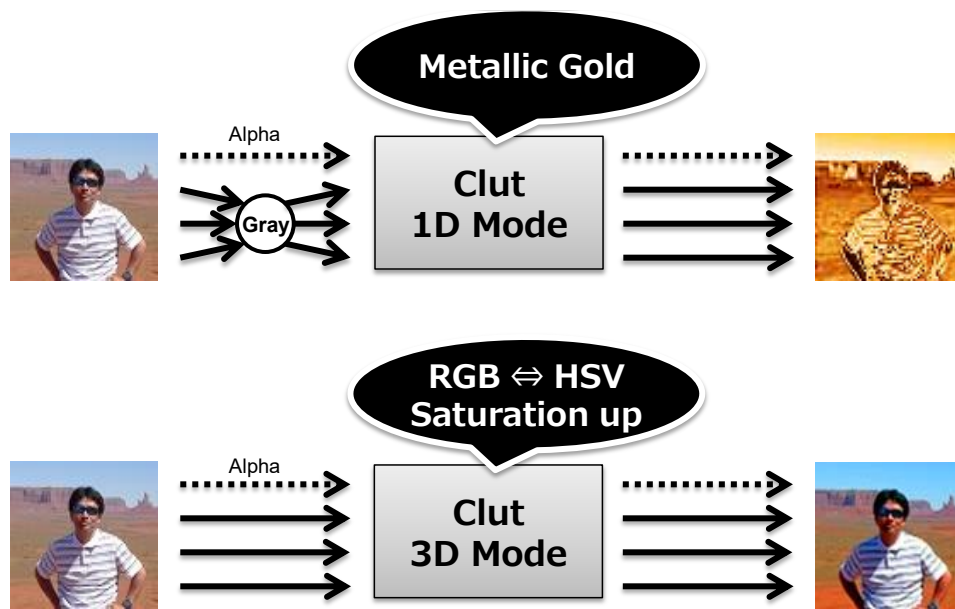


Figure 67 Clut Application

6.5.2. 3D Mode

- In this mode, new ARGB elements are generated from input elements RGB (not limited to RGB; other color spaces such as YUV or HSV may also be used). The corresponding Color Lookup Table comprehensively maps the combination results. For example, a table can be defined to convert RGB to HSV, manipulate a specific hue, and then convert back to RGB. This type of transformation is not achievable using conventional element-wise Clut processing.
- Since generating a full 2^{24} -entry table (8 bits \times 3 elements) is impractical, the inputs are uniformly sampled every 16 values per element. This reduces the table size to 2^{12} entries (4 bits \times 3 elements). Each entry in the table stores a 24-bit RGB value.
- In *frComp*, final values are generated from coarsely sampled data using tri-linear interpolation. This method enables highly accurate results for linear transformations (e.g., RGB to YUV conversion with an error margin of approximately 0.5 bits in the LSB). However, for nonlinear transformations, interpolation errors may increase.
- The table index is capped at 240 due to the 16-level sampling per element. Since index 256 does not exist, indices 241–255 cannot be directly interpolated. These are approximated by extrapolating from existing values: the value at index 240 is doubled, and the value at index 224 is subtracted to obtain the estimated result. This extrapolation is performed using linear interpolation.
- Table generation is carried out by incrementing each RGB element in steps of 16, and extracting the corresponding transformed value. The following is a conceptual example in C language to generate an RGB to YUV conversion table. The `put()` function represents the act of storing values in the table, and clamping or rounding is not accounted for in this example.

```
for (r = 0; r < 16; r++)
    for (g = 0; g < 16; g++)
        for (b = 0; b < 16; b++) {
            y = ( 77 * r + 150 * g + 29 * b) / 16;
            u = (-43 * r - 85 * g + 128 * b) / 16;
            v = (128 * r - 107 * g - 21 * b) / 16;
            put(r, g, b, y, u, v);
        }
```

- The table data is transferred from the start address specified in the Command List to internal SRAM. In tri-linear transformation, eight index values are referenced simultaneously, which requires a specialized addressing scheme optimized for SRAM access. Assuming the `unpack[]` external array holds the pre-packed one-dimensional table values, the following C code illustrates how this is organized in the `put()` function:

```

/* separate bit location */
r0 = r    & 1;
r1 = (r >> 1) & 1;
r2 = (r >> 2) & 1;
r3 = (r >> 3) & 1;
g0 = g    & 1;
g1 = (g >> 1) & 1;
g2 = (g >> 2) & 1;
g3 = (g >> 3) & 1;
b0 = b    & 1;
b1 = (b >> 1) & 1;
b2 = (b >> 2) & 1;
b3 = (b >> 3) & 1;

/* location and assignment */
ua = b0    | (g0 << 1) | (r0 << 2)
    | (b1 << 3) | (g1 << 4) | (r1 << 5)
    | (b2 << 6) | (g2 << 7) | (r2 << 8)
    | (b3 << 9) | (g3 << 10) | (r3 << 11);

value = (a << 24) | (y << 16) | (u << 8) | v;
unpack[ua] = value;

```

- The 32-bit data is ultimately stored in memory as-is. If Element A is not used, the upper 8 bits may contain unknown values; however, `ClutCntl.En[3]` must be set to '0'. The same rule applies in 2D mode.
- To improve precision, a simplified floating-point configuration is supported (from Ver.C onward). In the table entry for each element, the LSB 3 bits of Element A specify the fractional bit position for the RGB elements. For example, if Element A is set to 2, the fractional point for the RGB elements is placed between LSB2 and LSB3. This configuration is effective when defining tables with limited dynamic range.

6.5.3. 2D Mode

- This mode uses only Elements R and B. Because the upper 6 bits are used for indexing and the lower 2 bits are applied for bi-linear interpolation, this mode offers higher precision than 3D mode. As described in the section on the Canny filter, this mode is suitable for applying arbitrary (linear or non-linear) transformations based on two elements.
- While 3D mode uses all input elements, 2D mode discards Element G. Element A remains unchanged.

```
/* separate bit location */
```

```
r0 = r    & 1;
```

```
r1 = (r >> 1) & 1;
```

```
r2 = (r >> 2) & 1;
```

```
r3 = (r >> 3) & 1;
```

```
r4 = (r >> 4) & 1;
```

```
r5 = (r >> 5) & 1;
```

```
b0 = b    & 1;
```

```
b1 = (b >> 1) & 1;
```

```
b2 = (b >> 2) & 1;
```

```
b3 = (b >> 3) & 1;
```

```
b4 = (b >> 4) & 1;
```

```
b5 = (b >> 5) & 1;
```

```
/* location and assignment */
```

```
ua = b0    | (r0 << 1)
```

```
    | (b1 << 2) | (r1 << 3)
```

```
    | (b2 << 4) | (r2 << 5)
```

```

| (b3 << 6) | (r3 << 7)

| (b3 << 8) | (r4 << 9)

| (b3 << 10) | (r5 << 11);

```

```
value = f(r, b);
```

```
unpack[ua] = value;
```

6.5.4. 1D Mode

- In this mode, each of the ARGB elements is indexed individually. Unlike 2D/3D transformations, Element A is also subject to transformation. The table values range from 0x101 (−255/256), 0x102 (−254/256), ..., to 0x0 (0.0), 0x1 (1/256), ..., 0xFF (255/256), and up to 0x100 (1.0). Index values span from 0 to 0x1FF, with 0x100 corresponding to a value of 0.0.
- The memory layout differs from 2D/3D modes. Each element's table consists of 512 entries, each occupying one 32-bit word. Only the lower 9 bits of each 32-bit entry are used. The table starts with Element B, indexed from 0x00 upward. Positive indices from 0x00 to 0x100 are followed by negative indices from 0x101 onward (totaling 512 entries). The same arrangement applies sequentially to Elements G, R, and A.
- 1D mode allows two entries to be defined simultaneously. After the first set of values, a second set can be stored consecutively. The second entry is selected when both the preceding filter flag and ClutCntl.Sel are set to '1'.

6.5.5. Input Value Range

- Filter operations may produce negative values. These are expressed using the most significant bit of a 9-bit word. However, the 3D Clut only accepts 8-bit inputs. In such cases, set the corresponding bit in MasterCntl.Inword to '1' for the affected element to convert the value range to a minimum of 0 and a maximum of 0xFF. If this is not done and a negative value is input, it will be treated as '0' by default.

- If it is known that the input will never be negative and only the value 1.0 (0x100) is used, setting MasterCntl.Inword is unnecessary. The value 1.0 is automatically mapped to 0xFF.
- Tables can support pseudo-floating-point representation (Ver.C). By sacrificing the LSB 3 bits of Element A, a common decimal point position for the RGB elements can be specified (positive direction only). This setting can be applied per table. This method increases the dynamic range for small values, enabling more precise results.

6.5.6. Specific Color Extraction

- This section describes a method for extracting specific colors using 3D mode. Here, skin tone extraction is used as an example. Extraction is typically performed in the HSV color space, where hue selection is possible. Hue is represented from 0° to 360° , with skin tones generally falling around 0° to 30° . Saturation and brightness vary depending on lighting conditions. For simplicity, the saturation threshold is set to $1/4$ (normalized) to exclude dark regions, while brightness is used as-is.

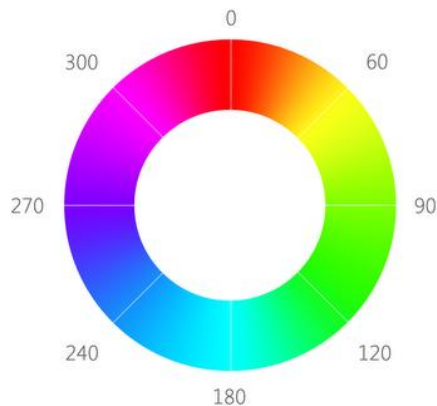


Figure 68 Color Space Hue

- In the lookup table, entries corresponding to skin tones are marked as true with a value of 1.0, while all others are set to 0.0. Since the table must be represented in RGB format, each RGB component (8-bit) is varied in steps of 16 to generate the table. This results in a total of 4,096 samples.
- Each RGB sample is then converted to HSV format, and samples where $H > 30^\circ$ or $S < 0.25$ are excluded. The remaining valid samples are considered to

represent skin tones, and their corresponding RGB entries in the table are all set to 0xFF.

- The diagram below is shown in a 64×64 format for illustrative purposes; however, the actual table is a 3-dimensional grid of size $16 \times 16 \times 16$.

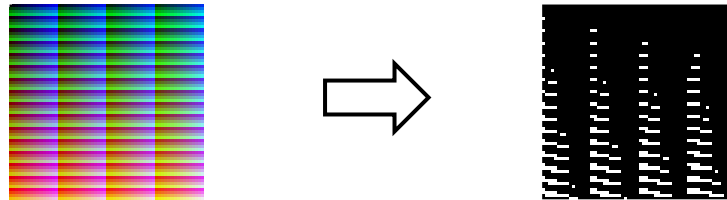
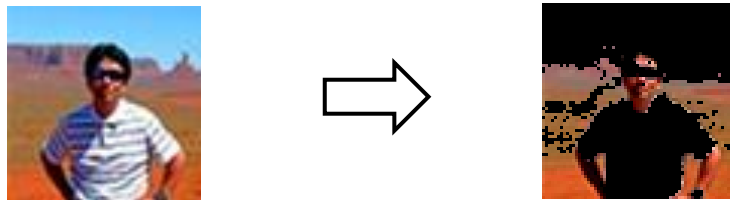


Figure 69 RGB Incremental and Converted Table

- The generated table, composed of 24-bit RGB values, is packed into 32-bit words and stored in memory. The frComp engine is then started to load the table into the 3D Clut. After skin tone extraction, the RGB components of the corresponding pixels are set to 0xFF.
- To make the result easier to interpret, the extracted pixel values are multiplied by the original input pixel values. Specifically, the **SrcOrg** path applies the 3D Clut transformation (used as α), while the **SrcMod** path bypasses the Clut and provides the original pixel value. The multiplication of these two yields a result where only skin tone regions remain visible, and non-skin areas appear black.
- As shown in the figure, non-skin regions are rendered black, while the

Figure 70 Detect Skin Color




6.5.7. Coordinate Transformation







- When performing coordinate transformation using the **SrcOut** path, set **MasterCntl.ReplaceEn** to '0'. This causes the coordinate values **X** and **Y** to be assigned to Elements **AR** and **GB**, respectively. If Elements **A** and **G** are both '0', then Elements **R** and **B** contain sufficient information, and a standard 2D mode transformation can be applied directly.
- If Elements **A** and **G** are not '0', use **DstMapInfo.Prec** to right-shift the coordinate values and fold the necessary information into Elements **R** and **B**. If coordinates can be negative, the lookup table must be prepared in advance to handle negative values appropriately.


6.6. Extractor Configuration

6.6.1. Binarization

- Binarization is performed by defining three regions using two threshold values: **Low** and **High**. Specific processing can be assigned to each of these regions. The following is an example of the region definitions based on thresholds Low and High.
 - Regions I to III represent the values to be assigned as pixel values in each range.
 - A dash (–) indicates that the input pixel value is passed through unchanged.

Type	Low High	I	II	III	Result
Original	0 0	0	0	0	

Simple Binarization	0 0x80	0	0	0xff	
Simple Binarization (Inverted)	0 0x80	0xff	0xff	0	
Partial Binarization (Maximize in Range)	0x40 0xc0	0xff	–	0xff	
Partial Binarization (Minimize in Range)	0x40 0xc0	0	–	0	
Adaptive Binarization Threshold = Mean +8	8 0xff	0	0xff	Don't care	
Threshold = Mean	0 0xff	0	0xff	Don't care	

Threshold =Mean -8	0 8	Don't care	0	0xff	
-----------------------	--------	------------	---	------	---

The sample image size is 128×128 . Adaptive binarization uses a 5×5 mean filter.

- **Adaptive binarization** utilizes data from both the **SrcOrg** and **SrcMod** paths. A 5×5 mean filter, Gaussian filter, or median filter is applied to one of the inputs to determine the threshold value. The other threshold is set to either 0x00 or 0xFF. Finally, binarization is performed by comparing the threshold with the data from the **SrcOrg** path.
- In adaptive binarization, a constant offset may be added to or subtracted from the threshold value. For addition, set the offset value to **PixelKeyHigh**; for subtraction, set it to **PixelKeyLow**.
 - In the **addition** case, only Regions I and II are referenced, so **PixelKeyLow** should be set to 0.
 - In the **subtraction** case, only Regions II and III are referenced, so **PixelKeyHigh** should be set to 0xFFFFFFFF.
- Binarization can also be applied to color images. In this case, each element is evaluated individually. Alternatively, binarization can be determined using a shared evaluation value. For example, each of the ARGB elements can be binarized independently, or the binarization result of Element A can be applied to Elements R, G, and B.
- When using a wide-area average filter (larger than 5×5), the filtering should first be performed using **frComp**, with the result written to memory. Then, use **Envelope** to insert the filtered result from the **SrcOut** path into Element A, and apply it as the threshold for binarization.
For handling wide-area filters that may result in pixel values exceeding 8 bits, refer to the section on processing pixel values beyond 8 bits.

6.7. Blender Configuration

6.7.1. Alpha Blending Configuration

- The alpha (α) value can be selected from any element or from a fixed constant value specified by PixelConst. For example, to blend a generated image into a destination image using a fixed alpha value, configure the fields in PixelCntl as shown below. The α value should be set in PixelConst.C0. This configuration is applied uniformly to the RGB elements (Element A can be set arbitrarily).

Field	Value	Description
DstCmp	0	No inversion of the destination alpha value
DstInv	0	No reciprocal of the destination value
DstOne	0	Do not use fixed 1.0 input for destination
DstASel	0	Use post-filtered value for destination
DstBSEL	4	Use fixed value from PixelConst.C0 as destination alpha
SrcCmp	0	No re-inversion of the source alpha (default inversion is applied)
SrcInv	0	No reciprocal of the source value
SrcOne	0	Do not use fixed 1.0 input for source
SrcASel	0	Not referenced; can be set to Unknown (since Blend = 1)
SrcBSEL	4	Use fixed value from PixelConst.C0 as source alpha (other constants also allowed); with SrcCmp = 0, this is automatically inverted to $1 - \alpha$
OpCarry	0	Treat as 8-bit pixel values
OpLut	0	Do not reference lookup table for blend result
OpALU	0	Use clamped addition (values below 0 become 0; values above 0xFF are saturated to 0xFF)
KeyHighSel KeyLowSel	0	Operation settings for Extractor
Max	0/1	Whether to treat pixel value 0xFF as 1.0 (optional)
Cross	0	Do not swap source and destination in calculation
Blend	1	Enable blending operation

En	1	Enable write-out
----	---	------------------

- By configuring the blend source (**DstIn*** settings) and blend destination (**DstOut*** settings) identically, the source image can be overlaid directly onto the destination image. Conversely, if different settings are used, alpha blending can be applied between the source and destination images to generate a new destination image. In the former case, DstInInfo, DstInBase, DstOutInfo, and DstOutBase must be the same, but this is not required in the latter.
- The alpha value (α) does not have to be fixed; it can be assigned from a varying element, or different alpha values can be used for the source and destination images, depending on requirements.
- Normally, the source image is selected from either the **SrcIn** or **SrcOut** path, but both can be routed to the A and B paths inside the **Blender**. This allows for advanced operations, such as blending an original image and an abstract image using threshold-controlled alpha values.

6.7.2. Handling Pixel Values Beyond 8 Bits

- The Blender's ALU supports 16-, 24-, or 32-bit operations by concatenating adjacent elements during addition or subtraction. Concatenation is enabled via the PixelCntl.OpCarry setting. When set to '1', the carry from the lower element is included in the computation. Elements are ordered as ARGB from high to low. For example, if only PixelCntlG.OpCarry is '1', Element G's computation incorporates the carry from Element B, effectively forming a 16-bit value with G as the upper 8 bits and B as the lower 8 bits.
- Source pixels are typically 9-bit signed values and cannot be directly used for 16/24/32-bit operations. To handle this, the upper elements are set to zero. For example, to extend Element B, set Elements A, R, and G to zero (via PixelCntlA, R, G settings), allowing the ALU to receive the 9-bit value extended to 32 bits with proper sign extension.
- Destination pixels are fed directly into the ALU, so their format must match the intended bit length for the operation. As with source handling, element extension is allowed. When using results iteratively, they should be saved in the same bit length as the operation.
- PixelCntlG.OpALU enables addition/subtraction without clamping. Clamping would zero out negative results per element. Without clamping, results are expressed in 2's complement across 16/24/32 bits. The result is written to memory using the desired bit length specified in DstOutInfo.Format. Even if

PixelCntl.Max is set to '1' to allow per-element representation of 1.0, no issues arise.

- When applying filters larger than 5×5 , processing is split and accumulated across multiple passes. Since 8-bit memory cannot store negative values, computations must use at least 16-bit operations. In such cases, only two elements can be processed simultaneously (not all four). The process for handling single-element accumulation is as follows:
 - **Reading Accumulated Data:**
Set DstInInfo.Format = 1, DstInInfo.Exp = 3, and DstInInfo.Rdc = 0 to load the data as 16 Bpp. For the first accumulation pass, do not reference the destination pixel.
 - **Writing Accumulated Data:**
Set DstOutInfo.Format = 1, DstOutInfo.Exp = 3, and DstOutInfo.Rdc = 0 to write data as 16 Bpp.
 - **Converting Accumulated Result to 8 Bpp:**
For the final write, change DstOutInfo.Format = 0 to output as 8 Bpp. Set DstInInfo.Exp = 2 to clamp negative values to zero, and use DstInInfo.Rdc to adjust gain. This is useful when the accumulated result is expected to exceed 1.0.

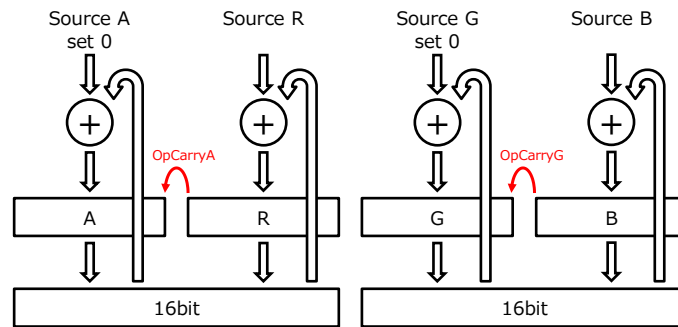


Figure 71 Accumulated 16bit Result

- For example, when performing 45×45 pattern matching, a 5×5 2D filter is applied 9×9 times, with the coefficients adjusted appropriately for each iteration. The accumulated results include both the filtered output using arbitrary coefficients and the averaged output, calculated simultaneously. These are stored as 16-bit values each, for a total of 32 bits.
- Finally, using the known energy of the reference pattern and the per-pixel results (from both the weighted and averaged outputs), normalization is performed via a 3D Clut. As shown in the diagram below, a normalized image is generated, with matched regions separately color-coded.

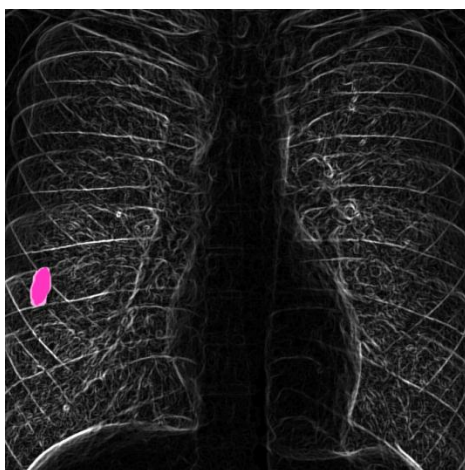


Figure 72 Pattern Matching Result (45x45 Normalized Cross-correlation)