A
Techniques

Standard and Private Techniques

For some XIE operations, several recognized algorithms or *techniques* offer varying tradeoffs between quality of results and performance. Also, in some cases, different techniques are required due to an image's class or content. In general, techniques that are specified for import and export elements identify image compression schemes, and techniques that are specified for process elements identify various image processing or enhancement algorithms.

Several *standard* techniques are defined in this appendix. Standard techniques are those that are well known to the image processing community. They are further categorized in this document as *required*, *optional*, or *excluded* (see Table B-2). While required techniques must be implemented in all servers, it is expected that most optional techniques will also be widely available. Some techniques are excluded from certain subsets.

Vendors may choose to extend XIE with their own *private* techniques to provide for their particular needs. Therefore, private techniques are likely to be available only on a particular vendor's platforms.

Technique numbers

Standard and private techniques can be differentiated by their technique numbers. The most significant bit of the technique number is zero (0) for standard techniques and one (1) for private techniques. Standard technique numbers are defined in this document (see Appendix C). Private technique numbers are assigned dynamically (the method for generating private technique numbers is chosen by the server implementor). Because private technique numbers are not statically defined, the number to name-string mapping must be obtained on a per-server basis via the **QueryTechniques** protocol request.

Technique names

The name string for standard techniques contains only the name of the algorithm or compression scheme (for example, "ERROR-DIFFUSION" or "CCITT-G32D"). Private technique name strings also include the name of the vendor that owns the rights to the technique (for example, "_PHOTOCO_SQUASH-BITS"). Because the organization name is encompassed by "_" (underscore) characters, the leading underscore provides another means by which standard techniques can be can differentiated from private techniques.

Default Techniques

In some cases, it is appropriate to assign a *default* technique. A default technique is a synonym that must be bound to a standard or private technique. Where there is no required standard technique, a default technique must be defined that is bound to an optional or a private technique. No default is defined in cases where it would be inappropriate (for example, decode techniques describing client data).

Technique parameters

For each occurrence of a technique within an element definition, allowance is made to pass parameters that are specific to the technique. Some techniques have defined parameters, and others have none. For techniques that do have defined parameters, the **QueryTechniques** request can be invoked to determine if the parameters must be supplied. If the parameters are optional and they are omitted by the client, the server will supply implementation specific default values in their place.

If the default technique is requested, no additional parameters are accepted (implementation specific default values will be provided for parameters defined for the technique to which ***Default*** is bound).

Technique information

The **QueryTechniques** request returns information about ***All*** techniques, all ***Default*** techniques, or a group of techniques that are functionally similar (for example, ***Dither***, ***Encode***, ***Geometry***, and so on). The following information is returned about the selected techniques:

needs-parameters	Indicates that the technique requires additional parameters; if false, the technique takes no parameters, or its parameters are optional. If the parameters are optional, they can be totally omitted, otherwise they must all be supplied.
group	Identifies which group the technique belongs to.
number	Specifies the numeric identifier assigned to the technique (MS bit is zero for *standard* techniques, or one for *private* techniques).
speed	Specifies the server's assessment of the speed of this technique relative to other techniques in the same group (where 0 is the slowest, and 255 is the fastest).
name	Identifies the technique name string in the form: <STANDARD-TECHNIQUE-NAME> or _<ORGANIZATION-NAME>_<PRIVATE-TECHNIQUE-NAME>

The remainder of this appendix provides a complete description of each technique and its parameters.

Color Allocation Techniques

Color allocation techniques allocate or match colors or gray shades from a COLORMAP. Cells allocated from a dynamic COLORMAP are recorded in a **ColorList**. If the COLORMAP is static, image pixels can only be matched to existing COLORMAP entries; no allocations are made and no **ColorList** needs to be specified.

AllocAll

Parameters

fill: CARD32

AllocAll allocates a read-only COLORMAP cell for each new pixel found. If the COLORMAP runs out of cells, the remaining new pixels are mapped to *fill*. A **ColorAlloc** event can be sent if it is necessary to use *fill*. ***AllocAll*** is only appropriate for dynamic COLORMAPs and requires that the number of discrete image pixels *fit* within the size of the COLORMAP to avoid running out of cells.

Match

Parameters

match-limit: XieTypFloat
gray-limit: XieTypFloat

Match allows a trade-off between image fidelity and COLORMAP usage via a pair of granularity parameters [[1]]. The highest priority is given to allocating read-only cells in a sequence that provides an even distribution of pixels throughout the colorspace. Secondary priority is given to the frequency of usage of image pixels. Any image pixel that is a close enough match to an existing read-only cell will share that cell (where “close” is determined by the granularity controls). For other image pixels, new read-only allocations are made. When no more cells are available, each remaining image pixel is matched to the closest read-only cell. ***Match*** is appropriate for both static and dynamic COLORMAPs. For the sake of computational efficiency, the number of discrete image pixels should not exceed the size of the COLORMAP.

Match-limit and *gray-match* control the allocation of colors and gray shades, respectively. The minimum value (0.0) specifies exact matches (within the limits of the COLORMAP). The maximum value (1.0) encompasses the entire colorspace within which no new cells are allocated. A **ColorAlloc** event can be sent if the COLORMAP runs out of cells.

Requantize

Parameters

max-cells: CARD32

Requantize first reduces the total number of discrete pixels values in the image to be no more than a specified number and, then, allocates the resulting pixels values as read-only cells from the COLORMAP. One method of accomplishing this reduction process can be found in [[2]].

Max-cells specifies the maximum number of COLORMAP allocations to allow. If *max-cells* is zero or greater than the number of unallocated COLORMAP cells, the reduction algorithm will restrict its output to the number of free cells. A **ColorAlloc** event can be sent if the number of pixels had to be restricted to a lesser number than *max-cells* due to a lack of free COLORMAP cells. ***Requantize*** is only appropriate for dynamic COLORMAPs.

1 A full description of the color matching algorithm is available upon request from author Shelley.
2 "Color image quantization for frame buffer display", by P. SHeckbert, *Computer Graphics* (ACM SIGGRAPH ’82Conf. Proc.), vol. 16, no. 3, page 297.

Constrain Techniques

ClipScale

Parameters

input-low: XieTypConstant
input-high: XieTypConstant
output-low: XieTypLevels
output-high: XieTypLevels

For each band, output pixels will be clipped to the range [*output-low, output-high*].

If *input-low* is less than *input-high*, then all pixels less than or equal to *input-low* will map to *output-low*, and all pixels that are greater than or equal to *input-high* will map to *output-high*. All intermediate pixel values are scaled proportionately to the output range. Nonintegral output values are rounded to the nearest integer.

If *input-low* is greater than *input-high* then all pixels that are greater than or equal to *input-low* will map to *output-low*, and all pixels that are less than or equal to *input-high* will map to *output-high*. All intermediate pixel values will be linearly mapped to the output range such that *input-low* maps to *output-low*, and *input-high* maps to *output-high*. Nonintegral output values are rounded to the nearest integer.

If *input-low* equals *input-high*, or if *output-low* or *output-high* exceeds *levels*-1, a ***FloTechnique*** error is generated.

HardClip

Parameters

< none >

HardClip constrains the data such that all negative pixels are set to zero, and all pixels greater than *levels*-1 are set to *levels*-1. All other pixels are rounded to the nearest integer value.

Convert From RGB

CIELab

Parameters

matrix: XieTypMatrix
white-adjust: XieTypWhiteAdjustTechnique
white-params: <technique-dependent>

Matrix is a 3x3 RGB to ***CIEXYZ*** conversion matrix (the source white point is also encoded in *matrix*). *White-adjust* is the **WhiteAdjustTechnique** that can be used to shift the white point of the output data. *White-params* is the list of parameters required by *white-adjust*.

The input **DataType** can be ***Constrained*** or ***Unconstrained***; the output **DataType** is always ***Unconstrained***. When the input is ***Constrained***, the data are normalized to the range [0, 1] (that is, scaled by 1/(*levels* **-** 1) prior to the conversion.

CIEXYZ

Parameters

matrix: XieTypMatrix
white-adjust: XieTypWhiteAdjustTechnique
white-params:<technique-dependent>

Matrix is a 3x3 RGB to ***CIEXYZ*** conversion matrix (the source white point is also encoded in *matrix*). *White-adjust* is the **WhiteAdjustTechnique** that can be used to shift the white point of the output data. *White-params* is the list of parameters required by *white-adjust*.

The input **DataType** can be ***Constrained*** or ***Unconstrained***; the output **DataType** is always ***Unconstrained***. When the input is ***Constrained***, the data are normalized to the range [0, 1] (that is, scaled by 1/(*levels* **-** 1) prior to the conversion.

YCbCr

Parameters

luma: XieTypConstant
levels: XieTypLevels
bias: XieTypConstant

Luma represents the proportions of red, green, and blue in the luminance band, ***Y***. *Levels* determines the output *levels* if *src* is ***Constrained*** (otherwise, *levels* is ignored). *Bias* is used to add an offset to the output pixels values.

Source data may be ***Constrained*** or ***Unconstrained***; the output *type* will match.

YCC

Parameters

luma: XieTypConstant
levels: XieTypLevels
scale: XieTypFloat

Luma represents the proportions of red, green, and blue in the luminance band, ***Y***. *Levels* determines the output *levels* if *src* is ***Constrained*** (otherwise, *levels* is ignored). *Scale* is used to compress the output pixels (a typical value is about 1.35 to 1.4 in the literature).

Convert To RGB

CIELab

Parameters

matrix: XieTypMatrix
white-adjust: XieTypWhiteAdjustTechnique
white-params: <technique-dependent>
gamut-compress: XieTypGamutTechnique
gamut-params: <technique-dependent>

Matrix is a 3x3 ***CIEXYZ*** to RGB conversion matrix (the target white point is also encoded in *matrix*). *White-adjust* is the **WhiteAdjustTechnique** that can be used to shift the white point of the source data prior to conversion. *White-params* is the list of parameters required by *white-adjust*. *Gamut-compress* is the **GamutTechnique** that can be used to keep the output pixels within the bounds of the RGB colorspace. *Gamut-params* is the list of parameters required by *gamut-compress*.

The input **DataType** must be ***Unconstrained***; the output **DataType** is also ***Unconstrained***.

CIEXYZ

Parameters

matrix: XieTypMatrix
white-adjust: XieTypWhiteAdjustTechnique
white-params: <technique-dependent>
gamut-compress: XieTypGamutTechnique
gamut-params: <technique-dependent>

Matrix is a 3x3 ***CIEXYZ*** to RGB conversion matrix (the target white point is also encoded in *matrix*). *White-adjust* is the **WhiteAdjustTechnique** that can be used to shift the white point of the source data prior to conversion. *White-params* is the list of parameters required by *white-adjust*. *Gamut-compress* is the **GamutTechnique** that can be used to keep the output pixels within the bounds of the RGB colorspace. *Gamut-params* is the list of parameters required by *gamut-compress*.

The input **DataType** must be ***Unconstrained***; the output **DataType** is also ***Unconstrained***.

YCbCr

Parameters

luma: XieTypConstant
levels: XieTypLevels
bias: XieTypConstant
gamut-compress: XieTypGamutTechnique
gamut-params: <technique-dependent>

Luma represents the proportions of red, green, and blue in the luminance band, ***Y***. *Levels* determines the output *levels* if *src* is ***Constrained*** (otherwise, *levels* is ignored). *Bias* is used to remove any offset from the source pixels values. *Gamut-compress* is the **GamutTechnique** that can be used to keep the output pixels within the bounds of the RGB colorspace. *Gamut-params* is the list of parameters required by *gamut-compress*.

Source data may be ***Constrained*** or ***Unconstrained***; the output *type* will match.

YCC

Parameters

luma: XieTypConstant
levels: XieTypLevels
scale: XieTypFloat
gamut-compress: XieTypGamutTechnique
gamut-params: <technique-dependent>

Luma represents the proportions of red, green, and blue in luminance ***Y***. *Levels* determines the output if *src* is ***Constrained*** (otherwise, *levels* is ignored). *Scale* is used to expand the source pixels (a typical value is about 1.35 to 1.4 in the literature). *Gamut-compress* is the **GamutTechnique** that can be used to keep the output pixels within the bounds of the RGB colorspace. *Gamut-params* is the list of parameters required by *gamut-compress*.

Source data may be ***Constrained*** or ***Unconstrained***; the output *type* will match.

Convolution Edge Techniques

Various methods of handling edge conditions are provided for **Convolve**. These techniques determine what pixel values are used when **Convolve** requires data beyond the image bounds. **ConvolveTechnique**s only come into play when the kernel is positioned partially off the edge of the image. Data around the edges of a **ProcessDomain** are convolved with adjacent image pixels wherever possible.

Constant

Parameters

constant: XieTypConstant

Constant uses the value specified by *constant* if pixels are required from beyond the edge of the image.

Replicate

Parameters

< none >

Replicate uses the nearest edge pixel value if pixels are required from beyond the edge of the image.

Decode Techniques

UncompressedSingle

Parameters

fill-order: XieTypOrientation
pixel-order: XieTypOrientation
pixel-stride: CARD8
left-pad: CARD8
scanline-pad: CARD8

UncompressedTriple

Parameters

fill-order: XieTypOrientation
pixel-order: XieTypOrientation
band-order: XieTypOrientation
interleave: XieTypInterleave
pixel-stride: XieTypTripletofCARD8
left-pad: XieTypTripletofCARD8
scanline-pad: XieTypTripletofCARD8

These techniques are used when no compression scheme has been applied to the image data. They are used for ***SingleBand*** and ***TripleBand*** data, respectively. Their parameters define the format of the **DataStream** of uncompressed data (the server may reformat the data as it chooses prior to processing or storage).

The following parameters are common to both techniques:

- When multiple pixels are put in the same byte or a pixel spans multiple bytes, *fill-order* specifies whether the pixels (or parts of pixels) are packed into the most or least significant bits of a byte first.
- For pixels that span a byte boundary, *pixel-order* defines whether the most or least significant bits of the pixel are transported first within the **DataStream**.
- *Pixel-stride* is the number of bits between the start of consecutive pixels within a scanline. It must be at least enough bits to contain the number of input *levels*.
- *Left-pad* is the number of pad bits preceding the first image pixel in each scanline. If the server's **Alignment** attribute is ***Alignable*** or *pixel-stride* fits the definition of ***Alignable***, the value of *left-pad* must be a multiple of *pixel-stride* or a multiple of 8; otherwise, *left-pad* may be any arbitrary value.
- *Scanline-pad* defines a multiple of bytes to which each scanline is padded; valid values are: 0 (not aligned), 1, 2, 4, 8, and 16. The total number of bits-per-scanline in the **DataStream** includes: *left-pad*, the image data (*width* x *pixel-stride*), and sufficient additional bits to satisfy *scanline-pad*.

The following parameters are only used by ***UncompressedTriple:***

- *Band-order* is the order of the image bands or image planes as they are transmitted through the protocol stream.
- *Interleave* describes how the image bands are interleaved (per pixel within a single plane, or sent as three separate planes). If *interleave* is ***BandByPixel***, inter-band dimensions must match (that is, the widths and the heights of all bands must match).

CCITT-G31D

Parameters

encoded-order: XieTypOrientation
radiometric: BOOL
normal: BOOL

CCITT-G31D is the CCITT group 3 one dimensional encoding technique as defined by [[1]].

Encoded-order specifies the *bit-order* of the encoded data. *Radiometric* specifies that "white runs" in the encoded data should be represented as image ones upon decode (maximum intensity), or, conversely, they will be decoded as image zeros if *radiometric* is false. *Normal* specifies that the data was processed according to its normal *fill-order* when it was originally encoded [2].

CCITT-G32D

Parameters

encoded-order: XieTypOrientation
radiometric: BOOL
normal: BOOL

CCITT-G32D is the CCITT group 3 two dimensional encoding technique as defined by [[1]].

Encoded-order specifies the *bit-order* of the encoded data. *Radiometric* specifies that "white runs" in the encoded data should be represented as image ones upon decode (maximum intensity), or, conversely, they will be decoded as image zeros if *radiometric* is false. *Normal* specifies that the data was processed according to its normal *fill-order* when it was originally encoded [2].

CCITT-G42D

Parameters

encoded-order: XieTypOrientation
radiometric: BOOL
normal: BOOL

CCITT-G42D is the CCITT group 4 two dimensional encoding technique as defined by [[3]].

Encoded-order specifies the *bit-order* of the encoded data. *Radiometric* specifies that "white runs" in the encoded data should be represented as image ones upon decode (maximum intensity), or, conversely, they will be decoded as image zeros if *radiometric* is false. *Normal* specifies that the data was processed according to its normal *fill-order* when it was originally encoded [2].

1 CCITT T.4, "Standardization of Group 3 Facsimile Apparatus for Document Transmission"

2 Image compression generally starts with the least significant pixel in the image (at coordinate 0,0) and proceeds in raster order to the end of the image. This would be the *normal* order. Some encoders read pixels in reversed order within each byte (that is, start reading at bit 7 and proceed down through 0, then 15 through 8, and so on). Specifying *normal* as false can correct for this situation upon decode (that is, the bit order within each byte of decoded data is reversed before the data are forwarded to downstream elements).

3 CCITT T.6, "Facsimile Coding Schemes and Coding Control Functions for Group 4 Facsimile Apparatus"

JPEG-Baseline

Parameters

interleave: XieTypInterleave
band-order: XieTypOrientation
up-sample: `BOOL`

JPEG-Baseline is the baseline Huffman DCT encoding technique that is defined in [1]. Only JPEG Interchange Format (JIF) is supported (that is, all tables, compressed data, and so on are embedded in the data stream, all delineated by `markers`).

Interleave determines if all bands of a ***TripleBand*** image will be interleaved within a single encoded stream or if three separate encoded streams are expected. *Band-order* specifies the order in which the image bands were originally encoded. *Up-sample* specifies that, if any bands in an interleaved encoded data stream are down-sampled, they should be up-sampled by the JPEG decoder.

Interleave, *band-order*, and *up-sample* are ignored for ***SingleBand*** images, and *up-sample* is always ignored if *interleave* is ***BandByPlane***. If *up-sample* is false and some of the encoded bands of an interleaved image were down-sampled, an alternative method for up-sampling the image would be to use a **Geometry** element with appropriate *band-mask* and *sample* technique parameters.

JPEG-Lossless

Parameters

interleave: XieTypInterleave
band-order: XieTypOrientation

JPEG-Lossless is the Huffman predictive sequential lossless encoding technique that is defined in [1].

Interleave describes how the bands of ***TripleBand*** data are interleaved; either all bands are interleaved within a single encoded stream, or three separate encoded streams are expected. *Band-order* specifies the order in which the image bands were originally encoded.

Interleave and *band-order* are ignored for ***SingleBand*** images.

[1] ISO DIS 10918-1 "Digital Compression and Coding of Continuous-tone Still Images"

TIFF-2

Parameters

encoded-order: XieTypOrientation
radiometric: BOOL
normal: BOOL

TIFF-2 is modified Huffman encoding as described in [[1]]. (TIFF compression scheme 2).

Encoded-order specifies the *bit-order* of the encoded data. *Radiometric* specifies that “white runs” in the encoded data should be represented as image ones upon decode (maximum intensity), or, conversely, they will be decoded as image zeros if *radiometric* is false. *Normal* specifies that the data was processed according to its normal *fill-order* when it was originally encoded [2].

TIFF-PackBits

Parameters

encoded-order: XieTypOrientation
normal: BOOL

TIFF-PackBits is byte-oriented run-length encoding as described in [[1]] (TIFF compression scheme 32773).

Encoded-order specifies the *bit-order* of the encoded data. *Normal* specifies that the data was processed according to its normal *fill-order* when it was originally encoded [2].

1 "TIFF Tag Image File Format", revision 6.0, draft 2, byAldus Corporation

2 Image compression generally starts with the least significant pixel in the image (at coordinate 0,0) and proceeds in raster order to the end of the image. This would be the *normal* order. Some encoders read pixels in reversed order within each byte (that is, start reading at bit 7 and proceed down through 0, then 15 through 8, and so on). Specifying *normal* as false can correct for this situation upon decode (that is, the bit order within each byte of decoded data is reversed before the data are forwarded to downstream elements).

Dithering Techniques

Dithering reduces the *z-resolution* or number of quantization levels in a continuous tone image. It simulates the visual appearance of the original image by substituting the original continuous tone data with patterns of lower z-resolution data. In ordered dither, fixed patterns are used. In error-diffusion dither, the patterns are somewhat random and generally less distracting to the eye. A thorough treatment of dithering techniques can be found in [1].

ErrorDiffusion

Parameters

< none >

ErrorDiffusion dither is an image *level* reduction process where the difference (*error*) between output pixel values and input pixel values is fed back through a filter to be applied to future input pixels. Thus, past quantization errors are negatively distributed or *diffused* into the yet to be requantized image.

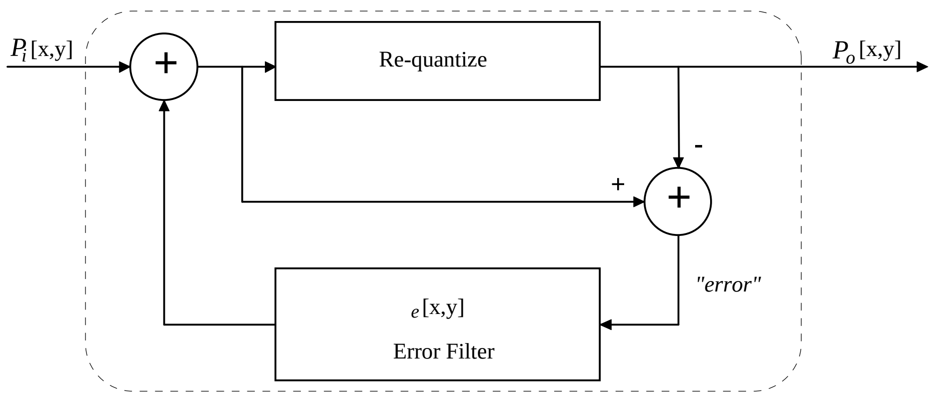


Figure A-1 Diagram of the *ErrorDiffusion* algorithm

An analysis of various error filters is presented in chapter 8 of [1]. One popular error filter is described in [2].

Ordered

Parameters

threshold-order: `CARD8`

Dispersed-dot ***Ordered*** dither replaces a matrix, or block, of pixels with a patterned matrix of pixels. This patterned matrix of pixels is applied across the entire image. Because these patterns may introduce artifacts that are distracting to the eye, the *threshold-order* parameter is available to determine the size of the dither matrix, and, therefore, the number of *levels* that can be simulated. If the value of threshold-order is **M**, the threshold matrix can simulate 2^M+1 levels.

For more information refer to [3], and chapter 6 of [4].

[1] "Digital Halftoning", by R. A. Ulichney, Cambridge, MA: The MIT Press
[2] "Adaptive algorithm for spatialgrey scale", by R. W. Floyd and L.Steinberg, *SID Int. Sym. Digest of Tech. Papers*, page 36
[3] "An optimum method for two level rendition of continuous-tone pictures", by B. E. Bayer, *Proc. IEEE Int. Conf. Commun., Conference Record*, page 26-11
[4] "Digital Halftoning", by R. A. Ulichney, Cambridge, MA: The MIT Press

Encode Techniques

ServerChoice (only valid for **ExportPhotomap**)

Parameters

preference: CARD8 { *PreferDefault*, *PreferSpace*, *PreferTime* }

ServerChoice allows the server to choose an encode technique. An optional hint can be provided to help the server make its choice, but the server is not obligated to obey the hint. *PreferTime* hints that retrieval performance is the desired metric; *PreferSpace* indicates that frugal use of storage space is more important. Normally, ***ServerChoice*** must choose a lossless encode technique, but when data is received from an adjacent upstream import element, it may choose to pass the import element's input data directly to the **Photomap**.

UncompressedSingle

Parameters

fill-order: XieTypOrientation
pixel-order: XieTypOrientation
pixel-stride: CARD8
scanline-pad: CARD8

UncompressedTriple

Parameters

fill-order: XieTypOrientation
pixel-order: XieTypOrientation
band-order: XieTypOrientation
interleave: XieTypInterleave
pixel-stride: XieTypTripletofCARD8
scanline-pad: XieTypTripletofCARD8

UncompressedSingle and ***UncompressedTriple*** specify that no compression scheme is to be applied to the image data. They are used for ***SingleBand*** and ***TripleBand*** data, respectively. Their parameters define the format of the **DataStream** of uncompressed data that is made available for client retrieval via **GetClientData**.

The following parameters are common to both techniques:

- When multiple pixels are put in the same byte or a pixel spans multiple bytes, *fill-order* specifies whether the pixels (or parts of pixels) are packed into the most or least significant bits of a byte first.
- For pixels that span a byte boundary, *pixel-order* defines whether the most or least significant bits of the pixel are put first within the **DataStream**.
- *Pixel-stride* is the number of bits between the start of consecutive pixels within a scanline. It must be at least enough bits to contain the number of *src levels*.
- *Scanline-pad* defines a multiple of bytes to which each scanline is padded; valid values are: 0 (not aligned), 1, 2, 4, 8, and 16.

The following parameters are only used by ***UncompressedTriple:***

- *Band-order* is the order of the image bands or image planes as they are transmitted through the protocol stream.
- *Interleave* describes how the bands are interleaved (per pixel within a single plane, or sent as three separate planes). Export of down-sampled ***BandByPixel*** data is not supported (that is, all bands must have equal widths and equal heights).

CCITT-G31D

Parameters

align-eol: `BOOL`
radiometric: `BOOL`
encoded-order: XieTypOrientation

CCITT-G31D is the CCITT group 3 one dimensional encoding technique as defined by [1].

Align-eol, if true, specifies that sufficient fill bits must precede EOL codes to guarantee that each EOL will end on a byte boundary (thus EOL will be a nibble of 0 followed by a byte of 1: $xxxx,0000_2$ $0000,0001_2$). *Radiometric* specifies that image ones will be encoded as "white runs", or, conversely, image zeros will be encoded as "white runs" if *radiometric* is false. *Encoded-order* specifies the *bit-order* for the encoded data.

CCITT-G32D

Parameters

uncompressed: `BOOL`
align-eol: `BOOL`
radiometric: `BOOL`
encoded-order: XieTypOrientation
k-factor: `CARD32`

CCITT-G32D is the CCITT group 3 two dimensional encoding technique as defined by [1].

Uncompressed, if true, will enable the use of the `uncompressed-mode` CCITT extension. If true, *align-eol* specifies that sufficient fill bits must precede EOL codes to guarantee that each EOL will end on a byte boundary (thus EOL will be a nibble of 0 followed by a byte of 1: $xxxx,0000_2$ $0000,0001_2$). *Radiometric* specifies that image ones will be encoded as "white runs", or, conversely, image zeros will be encoded as "white runs" if *radiometric* is false. *Encoded-order* specifies the *bit-order* for the encoded data. *K-factor* specifies the number of 2D scanlines to produce for each 1D scanline.

CCITT-G42D

Parameters

uncompressed: `BOOL`
radiometric: `BOOL`
encoded-order: XieTypOrientation

CCITT-G42D is the CCITT group 4 two dimensional encoding technique as defined by [2].

If true, *uncompressed* will enable the use of the `uncompressed-mode` CCITT extension. *Radiometric* specifies that image ones will be encoded as "white runs", or, conversely, image zeros will be encoded as "white runs" if *radiometric* is false. *Encoded-order* specifies the *bit-order* for the encoded data.

1 CCITT T.4, "Standardization of Group 3 Facsimile Apparatus for Document Transmission"
2 CCITT T.6 "Facsimile Coding Schemes and Coding Control Functions for Group 4 Facsimile Apparatus"

JPEG-Baseline

Parameters

interleave: XieTypInterleave
band-order: XieTypOrientation
horizontal-samples: XieTypTripletofCARD8
vertical-samples: XieTypTripletofCARD8
q-table: LISTofCARD8
ac-table: LISTofCARD8
dc-table: LISTofCARD8

JPEG-Baseline is baseline Huffman DCT encoding as defined by [1]. A stream of JPEG Interchange Format (JIF) data is produced (that is, all tables, compressed data, and so on, delineated by markers).

Interleave determines if all bands of a ***TripleBand*** image will be interleaved within a single encoded stream or if three separate encoded streams will be produced. *Band-order* specifies the order in which the bands are encoded. *Horizontal-samples* and *vertical-samples* are the horizontal and vertical sampling factors. *Q-table* is the quantization table. *Ac-table* specifies the AC Huffman table and *dc-table* specifies the DC Huffman table.

One *q-table* per band or a single *q-table* may be shared between all bands. Generally, there is a single AC/DC pair of Huffman tables, but, for ***TripleBand BandByPixel*** data, there may be two pairs of tables (one for the luminance band and the other for the chromanance bands). If any table is specified with zero length, it indicates that the server implementor is to supply that table (for example, the example tables defined in [1]).

Interleave, *band-order*, *horizontal-samples,* and *vertical-samples* are ignored for ***SingleBand*** images, and *horizontal-samples* and *vertical-samples* are always ignored if *interleave* is ***BandByPlane***. *Horizontal-samples* and *vertical-samples* share the definitions and restrictions stipulated for parameters H_i and V_i, respectively, that are specified in annexes A and B of [1].

JPEG-Lossless

Parameters

interleave: XieTypInterleave
band-order: XieTypOrientation

predictor: { PredictorNone	None
PredictorA	A
PredictorB	B
PredictorC	C
PredictorABC	A+B-C
PredictorABC2	A+((B-C)/2)
PredictorBAC2	B+((A-C)/2)
PredictorAB2 }	(A+B)/2

table: LISTofCARD8

JPEG-Lossless, corresponds to frames encoded using Huffman, predictive sequential lossless encoding as defined by [1]. A **DataStream** of JPEG Interchange Format (JIF) data is returned (that is, all tables, compressed data, and so on, delineated by markers).

Interleave determines if all bands of a ***TripleBand*** image will be interleaved within a single encoded stream or if three separate encoded streams will be produced. *Band-order* specifies the order in which the bands are encoded. *Interleave* and *band-order* are ignored for ***SingleBand*** images.

Predictor is the predictor selection value (one per band). *Table* is the lossless entropy encoding table (up to one per band). Specifying a *table* of length zero indicates that the server implementor should supply a table (for example, the example tables defined in [1]).

1 ISO DIS 10918-1 "Digital Compression and Coding of Continuous-tone Still Images"

TIFF-2

Parameters

radiometric: BOOL
encoded-order: XieTypOrientation

TIFF-2 is modified Huffman compression as described in [1] (TIFF compression scheme 2).

Radiometric specifies that image ones will be encoded as “white runs”, or, conversely, image zeros will be encoded as “white runs” if *radiometric* is false. *Encoded-order* specifies the *bit-order* for the encoded data.

TIFF-PackBits

Parameters

encoded-order: XieTypOrientation

TIFF-PackBits is byte-oriented run-length encoding as described in [1] (TIFF compression scheme 32773).

Encoded-order specifies the *bit-order* of the encoded data.

1 "TIFF Tag Image File Format", revision 6.0, draft 2, byAldus Corporation

Gamut Compression Techniques

GamutTechnique defines the gamut compression techniques used by the **ConvertToRGB** element. It deals with converted colors that fall outside the gamut of the RGB colorspace.

None

Parameters

< none >

None specifies that no gamut compression is required.

ClipRGB

Parameters

< none >

ClipRGB ensures that ***Unconstrained*** output pixel values remain within the implicit range [0.0, 1.0].

Note that ***Constrained*** data are clipped to the range [0, *levels*-1] by definition.

Geometry Techniques

Antialias

Parameters

< none >

Some **Geometry** techniques produce artifacts when replicating pixels or sampling the input image at distinct intervals. One example are the “jaggies” that one often sees along diagonal lines and curved objects in an image. Another indication is that some techniques, such as, nearest neighbor, seem to drop image detail. For example, one could scale down an image by four simply by selecting every fourth row and column in the input image. Given an image with a set of randomly placed vertical lines, each line will have a 75% chance of being dropped in the output image. As a result, the output image in the worst case may have little resemblance to the input image.

Antialiasing techniques incorporate information from an `area` of pixels in the input image in order to produce each output pixel. This implies that line dropouts and other artifacts will occur less often, and the output image may have markedly better resemblance to the input image.

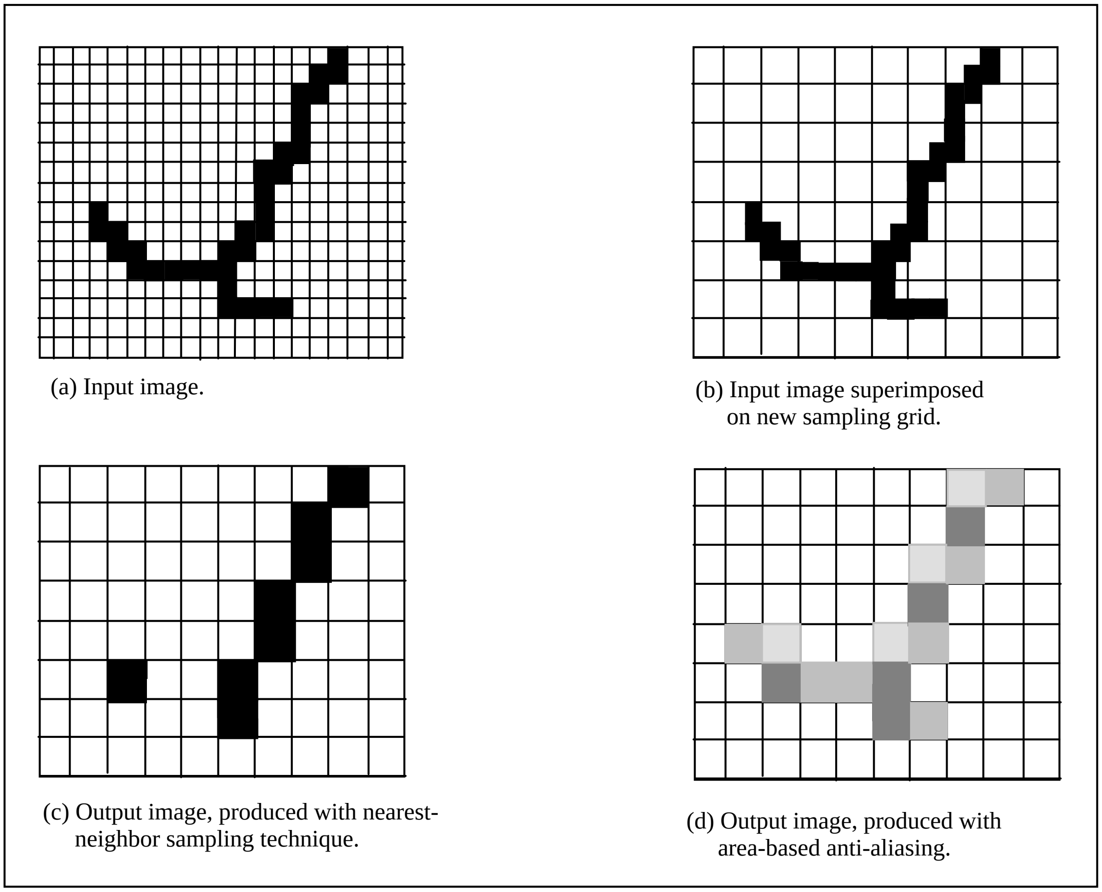
The techniques ***AntialiasByArea*** and ***AntialiasByLowpass*** are two examples of specific antialiasing algorithms. ***Antialias*** may be mapped to either of them or to a vendor-private algorithm. All parameters for the specific algorithm employed are defaulted by the server.

AntialiasByArea

Parameters

simple: INT16

Geometry may be used to scale an image down from a larger size to a smaller size. This will decrease the density of pixels per unit area in the image. In Figure A-2(a), an input image at the original resolution is displayed. It is to be reduced to the grid shown in Figure A-2(b), where the original image is superimposed on the new, lower density grid. Figure A-2(c) shows a possible result of using nearest-neighbor sampling to fill the output grid. Specifically, the output pixel's upper-left corner is matched against the nearest upper-left corner of pixels in the input image. If the nearest input pixel is black, the output pixel is filled with black. Otherwise, it is filled with white. While the pixel intensities (in this case, pure black and pure white) are preserved by this operation, the overall image shape is changed.


(a) Input image.

(b) Input image superimposed on new sampling grid.

(c) Output image, produced with nearest-neighbor sampling technique.

(d) Output image, produced with area-based anti-aliasing.

Figure A-2 Effect of sampling technique when scaling to a lower pixel density

In Figure A-2(d), the output image is computed by assigning to each output pixel the weighted average of the intensity values of input pixels that fall within its area. That is, the four corners of the output pixel are projected back onto the input image. The input pixels that lie within this area contribute according to the following:

$$I(P_o) = \sum_{P_i} \frac{A\left(\hat{P}_o \bigcap P_i\right)}{A(\hat{P}_o)} \times I(P_i) \qquad (1)$$

$I(P_o)$ is the intensity of output pixel P_o. $\hat{P}_o$ stands for the projection of P_o into the input image space. $A(\hat{P}_o)$ is the area of $\hat{P}_o$ in the input image space. $A(\hat{P}_o \bigcap P_i)$ is the area of the intersection of $\hat{P}_o$ with the rectangular locus of P_i. The sum progresses over all pixels P_i in the input image (note that for most such pixels, the area of intersection will be zero).

Figure A-3 illustrates the calculation for a single output pixel, **P**, whose projection intersects nine input pixels to various degrees. Note that the number of pixels intersected depends on the degree of scaling involved. A greater reduction in producing the output image will cause more input pixels to be intersected per output pixel. Also, in case of nonorthogonal rotation of the image, the areas of intersection will not be rectangular in general.

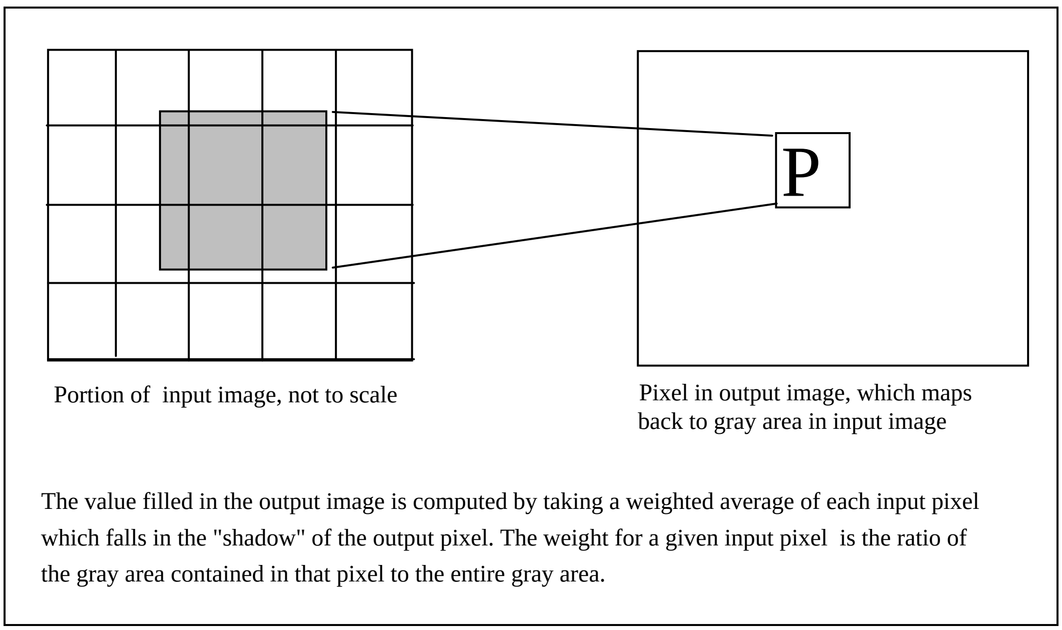


Figure A-3 Illustration of an output pixel mapping back to the input image

Because of the computational complexity of this method, two approximations are supported. If *simple* is nonzero, the pixels covered by the projected area will be averaged without regard to the relative amount of gray area that they contain: if they are touched by the area, they are included in a simple average. If *simple* is set to *N*, with *N* odd and greater than one (3,5,7, ...), then only the center point of the output pixel is projected, and a simple average is taken of an *N* by *N* window centered on the projection. For best results, *N* should correspond roughly to the amount of scaling that will be done.

AntialiasByLowpass

Parameters

kernel-size: `INT16`

Geometry may be used to scale an image down from a larger size to a smaller size. This will decrease the density of pixels per unit area in the image. Figure A-2 demonstrates the difference between scaling an image down using nearest-neighbor vs. antialias techniques. ***AntialiasByArea*** preserves shape, as in Figure A-2 (d) but can be very slow computationally. The method presented here represents an approximation to ***AntialiasByArea*** , which can be faster, yet provides similar results.

Figure A-4 demonstrates how antialiasing can be produced using a low-pass filter. A 3x3 boxcar kernel is passed over the original input image, in Figure A-4 (b). This image is shown superimposed on the lower density grid in Figure A-4 (c). Nearest neighbor sampling by selecting the value of the input pixel in the upper-left corner of the output pixel area results in the image in Figure A-4(d). As was the case with ***AntialiasByArea***, the overall structure of the input image is maintained.

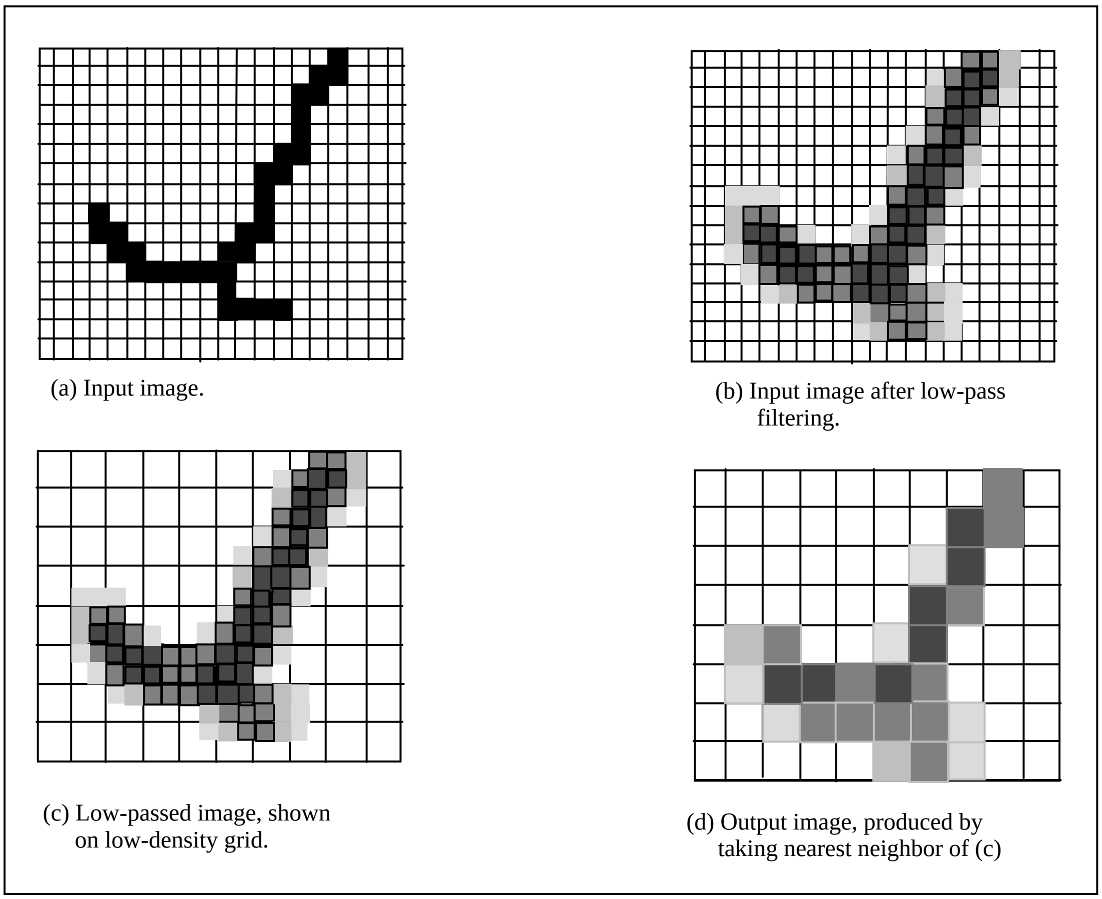


Figure A-4 Sequence producing an antialiased image using a low-pass filter

The user is allowed to select the size of the image kernel via the *kernel-size* parameter. For best results, *kernel-size* should be chosen to correspond roughly to the amount of scaling that will be done. Note that the server chooses the best kernel for the appropriate size; the values used in the kernel are not alterable by the client application. Clients wishing to specify the kernel in more detail should use **Convolve** directly.

BilinearInterpolation

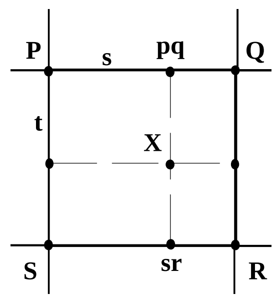
Parameters

< none >

Geometry maps each pixel (x',y') in the *output* image to the coordinate location (x,y) in the *input* image by:

$$\begin{pmatrix}\mathbf{x}\\ \mathbf{y}\end{pmatrix}=\begin{pmatrix}\mathbf{a} & \mathbf{b}\\ \mathbf{c} & \mathbf{d}\end{pmatrix}\begin{pmatrix}\mathbf{x}'\\ \mathbf{y}'\end{pmatrix}+\begin{pmatrix}\mathbf{t_x}\\ \mathbf{t_y}\end{pmatrix} \quad (1)$$

It is not unusual that the input location (x,y), so derived will be nonintegral and will not correspond exactly to a single pixel in the input image. In this case, we can obtain an interpolated value by first interpolating its neighbors in the X direction followed by an interpolation in the Y direction, as depicted below.



Pixel grid locations **P**, **Q**, **R**, and **S** are integral. Pixel location **X** = (x,y)^T, obtained from the mapping equation above, differs from **P** by fractional amounts **s** in the x direction and **t** in the y direction. Point **pq** is the projection of **X** on line **PQ**, and point **sr** is the projection of **X** on line **SR**.

Let ***I*(P)** be the value of the input image at coordinate **P**, if **P** is within the image extent. Otherwise, let ***I*(P)** be *constant*, where *constant* is the pixel value passed to the Geometry element. A value of ***I*(X)** must be estimated from ***I*(P)**, ***I*(Q)**, ***I*(R)**, and ***I*(S)**. We may obtain estimates of ***I*(pq)** and ***I*(sr)** by:

$$I(\mathbf{pq}) = (1 - \mathbf{s}) * I(\mathbf{P}) + \mathbf{s} * I(\mathbf{Q})$$

$$I(\mathbf{sr}) = (1 - \mathbf{s}) * I(\mathbf{S}) + \mathbf{s} * I(\mathbf{R})$$

Given values for ***I*(pq)** and ***I*(sr)**, ***I*(X)** is obtained by interpolating in the **y** direction:

$$I(\mathbf{X}) = (1 - \mathbf{t}) * I(\mathbf{pq}) + \mathbf{t} * I(\mathbf{sr})$$

Thus

$$I(\mathbf{X}) = (1 - \mathbf{s}) * (1 - \mathbf{t}) * I(\mathbf{P}) + (1 - \mathbf{s}) * \mathbf{t} * I(\mathbf{S}) + \mathbf{s} * (1 - \mathbf{t}) * I(\mathbf{Q}) + \mathbf{s} * \mathbf{t} * I(\mathbf{R})$$

Note that ***I*(X)** provides sensible values for the extreme values of **s** and **t** (*0* and *1*) and for **s = t =** $\frac{1}{2}$.

Gaussian

Parameters

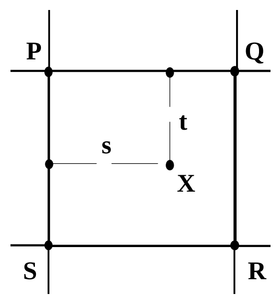
sigma: XieTypFloat
normalize: XieTypFloat
radius: CARD8
simple: BOOL

Sigma is the drop-off rate; *normalize* is a normalization constant; *radius* defines the extent of computation; and, *simple* allows the use of an approximate form.

Geometry maps each pixel (x',y') in the *output* image to the coordinate location (x,y) in the *input* image by:

$$\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{a} & \mathbf{b} \\ \mathbf{c} & \mathbf{d} \end{pmatrix} \begin{pmatrix} \mathbf{x}' \\ \mathbf{y}' \end{pmatrix} + \begin{pmatrix} \mathbf{t}_{\mathbf{x}} \\ \mathbf{t}_{\mathbf{y}} \end{pmatrix} \tag{1}$$

It is not unusual that the input location (x,y), so derived will be nonintegral and will not correspond exactly to a single pixel in the input image. The situation is depicted below:



Pixel grid locations **P**, **Q**, **R**, and **S** are integral. Pixel location $\mathbf{X} = (x,y)^{\mathbf{T}}$, obtained from the mapping equation above, differs from **P** by fractional amounts **s** (horizontally) and **t** (vertically).

Let $I(\mathbf{P})$ be the value of the input image at coordinate **P**, if **P** is within the image extent. Otherwise, let $I(\mathbf{P})$ be *constant*, where *constant* is the pixel value passed to the Geometry element. A value of $I(\mathbf{X})$ must be estimated from $I(\mathbf{P})$, $I(\mathbf{Q})$, $I(\mathbf{R})$, and $I(\mathbf{S})$. From sampling theory, a band width limited continuous input image can be recovered perfectly (under certain conditions) from its sampled output by computing the convolution:

$$I(x,y) = \sum_m \sum_n i(m,n)h(x-m, y-n) \tag{2}$$

$I(x,y)$ is the continuous image, $i(m,n)$ the discrete sampled image, and $h(u,v)$ the impulse response function for an appropriate low pass filter. Note that both nearest-neighbor and bilinear interpolation resampling techniques may be cast in this form [1].

A Gaussian impulse response function has the quality that its Fourier transform has the same bell curve shape. For certain kinds of images, this provides a better approximation to an ideal low-pass filter than is represented by either bilinear or nearest-neighbor interpolation. The specific form of $h(u,v)$ is given by:

1 A. Jain. Fundamentals of Digital Image Processing. Englewood Cliffs, J.J. Prentice-Hall, 1989

$$h(u,v) = \frac{1}{2\pi\sigma^2} \times \exp\left(-\frac{u^2+v^2}{2\sigma^2}\right) \qquad (3a)$$

The $1/2\pi\sigma^2$ term is called the *normalization constant* and may be altered using the *normalize* parameter. The suggested value for σ is 1. Note that all technique parameters must be chosen in concert.

As an example of Gaussian interpolation, assume σ = 1 and that point **P** in the above figure has coordinates (***m'***,***n'***). Then ***x*** = ***m'***+ **s** and ***y*** = ***n'***+ **t**, and the basic interpolation equation at (***x***,***y***) becomes:

$$I(x,y) = I(m'+s,n'+t) = \sum_m \sum_n i(m,n)h(m'-m+s,n'-n+t) \qquad (4)$$

The contribution from point **P** in this equation is represented by the summation value for **m = m'**, **n = n'**. Identifying this term in Eq (4), the contribution is:

$$\Delta I(x,y;m',n') = i(m',n')h(s,t) = i(m',n')\tfrac{1}{2\pi}e^{-\frac{1}{2}\left(s^2+t^2\right)} \qquad (5)$$

Or:

$$\Delta I(x,y;P) = \tfrac{1}{2\pi}I(P)e^{-\frac{1}{2}\left(s^2+t^2\right)} \qquad (6)$$

Similar calculations reveal:

$$\Delta I(x,y;Q) = \tfrac{1}{2\pi}I(Q)e^{-\frac{1}{2}\left((s-1)^2+t^2\right)} \qquad (7)$$

$$\Delta I(x,y;R) = \tfrac{1}{2\pi}I(R)e^{-\frac{1}{2}\left((s-1)^2+(t-1)^2\right)} \qquad (8)$$

$$\Delta I(x,y;S) = \tfrac{1}{2\pi}I(S)e^{-\frac{1}{2}\left(s^2+(t-1)^2\right)} \qquad (9)$$

A suggested value for *radius* is one, that is, only pixels within a distance of one in either the ***x*** or ***y*** direction are involved in the calculation. Thus, ***I***(***x***,***y***) is given by the sum of Eqs. 6-9. Choosing a larger *radius* would bring in more terms.

For computational convenience, a simplified form of Gaussian interpolation is provided. If *simple* is true, the impulse-response function of Equation 3a is replaced by:

$$h(u,v) = N \times 2^{-\frac{u^2+v^2}{\sigma^2}} \qquad (3b)$$

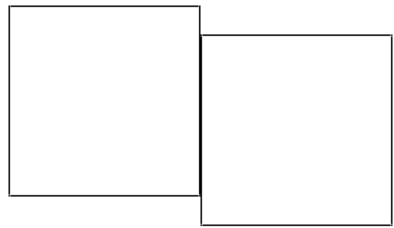
The normalization factor ***N*** is defined by *normalize*. As with true Gaussian interpolation, the *radius* parameter is used to determine the number of pixels involved in the computation.

NearestNeighbor

Parameters

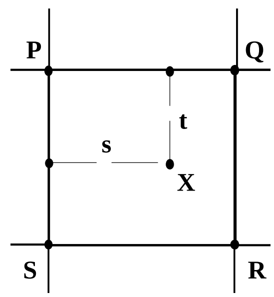
modify: CARD8	{ *FavorDown*, *FavorUp*, *RoundNW*, *RoundNE*, *RoundSW*, *RoundSE* }

Geometry maps each pixel (***x′***,***y′***) in the *output* image to the coordinate location (***x***,***y***) in the *input* image by:



(1)

It is not unusual that the input location (***x***,***y***), so derived will be nonintegral and will not correspond exactly to a single pixel in the input image. The situation is depicted below:



Pixel grid locations **P**, **Q**, **R**, and **S** are integral. Pixel location **X** = (***x***,***y***)$^{\mathbf{T}}$, obtained from the mapping equation above, differs from **P** by fractional amounts **s** in the ***x*** direction and **t** in the ***y*** direction.

Let ***I*(P)** be the value of the input image at coordinate **P** if **P** is within the image extent. Otherwise, let ***I*(P)** be *constant*, where *constant* is the pixel value passed to the Geometry element. A value of ***I*(X)** must be estimated from ***I*(P)**, ***I*(Q)**, ***I*(R)**, and ***I*(S)**. In nearest-neighbor sampling, we simply choose the image value from the discrete location closest to **X**. Thus,

if $\mathbf{s} < 1/2$, $\mathbf{t} < 1/2$, set $I(\mathbf{X}) = I(\mathbf{P})$,

if $\mathbf{s} > 1/2$, $\mathbf{t} < 1/2$, set $I(\mathbf{X}) = I(\mathbf{Q})$,

if $\mathbf{s} > 1/2$, $\mathbf{t} > 1/2$, set $I(\mathbf{X}) = I(\mathbf{R})$,

if $\mathbf{s} < 1/2$, $\mathbf{t} > 1/2$, set $I(\mathbf{X}) = I(\mathbf{S})$.

The behavior on even boundaries ($\mathbf{s} = 1/2$ or $\mathbf{t} = 1/2$) is determined by the *modify* parameter. If *modify* is *FavorDown*, all less than signs in the above are replaced with less than or equal signs. Thus, **P** would win all ties; **S** and **Q** would lose to **P** but win over **R**; and, **R** would lose all ties. If modify is *FavorUp* , then all greater than signs would be replaced with greater than or equals, and the opposite behavior would occur.

Four additional options are provided. The *RoundNW* option will always choose **P**; *RoundNE* will always choose **Q**; *RoundSE* will always choose **R**; and, *RoundSW* will always choose **S**. These are not strictly nearest neighbor algorithms but are available for computational/filtering convenience.

Histogram Shapes

MatchHistogram matches the histogram of a source image to a specific shape represented by a probability density function. **HistogramShape** defines various shapes that can be requested.

Flat

Parameters

< none >

Flat specifies that the output image is to have a histogram that approximates a uniform density (histogram equalization). No additional parameters are required.

Gaussian

Parameters

mean: XieTypFloat
sigma: XieTypFloat

Gaussian specifies that the output image is to have histogram that approximates a Gaussian probability density. The supplied parameters are used to generate a Gaussian probability density function centered around the *mean* level with a spread specified by *sigma*:

$$\boldsymbol{P}_{Gaussian}(x) = \frac{1}{\sqrt{2\pi \times \mathtt{sigma}^2}} \times \exp\left(\frac{-(x - \mathtt{mean})^2}{2 \times \mathtt{sigma}^2}\right)$$

Hyperbolic

Parameters

constant: XieTypFloat
shape-factor: BOOL

Hyperbolic specifies that the output image is to have histogram that approximates a hyperbolic probability density.

Constant is used to generate a hyperbolic probability density function:

$$\boldsymbol{P}_{Hyperbolic}(x) = \frac{1}{(x + \mathtt{constant}) \times \log\left(1 + \frac{1}{\mathtt{constant}}\right)}$$

Shape-factor should be specified as false if the histogram shape represents decreasing values for higher *levels* or true if the shape represents increasing values for higher *levels*.

White Point Adjustment Techniques

WhiteAdjustTechnique defines the white point adjustment techniques that can be used when converting to or from the RGB colorspace. White point correction can be used to ensure that white *looks* white, or it can be used to change the overall tint of an image.

None

Parameters

< none >

None specifies that no white point correction is desired.

CIELabShift

Parameters

white-point: XieTypConstant

CIELabShift specifies that white point correction is to be accomplished by adding the white point displacement to the ***a*b**** plane in the ***CIELab*** colorspace. The *white-point* is specified using ***CIEXYZ*** encodings.

If **WhiteAdjustTechnique** is used with **ConvertFromRGB**, *white-point* specifies the desired white point of the output data. If **WhiteAdjustTechnique** is used with **ConvertToRGB**, *white-point* specifies the white point of the source data.