# Video Image Processing Technology

# Agenda

- Key trend of “video in FPGA”
- Video image processing basics
  - Color space conversion
  - Chroma sampling
  - Scaling
  - Deinterlacing
  - Image blending
  - Filtering
  - Gamma correction
- Conclusion

# Key Trend for “Video in FPGA” – 1

***High definition (HD) video is ~4x to 6x the size of standard definition (SD) video***

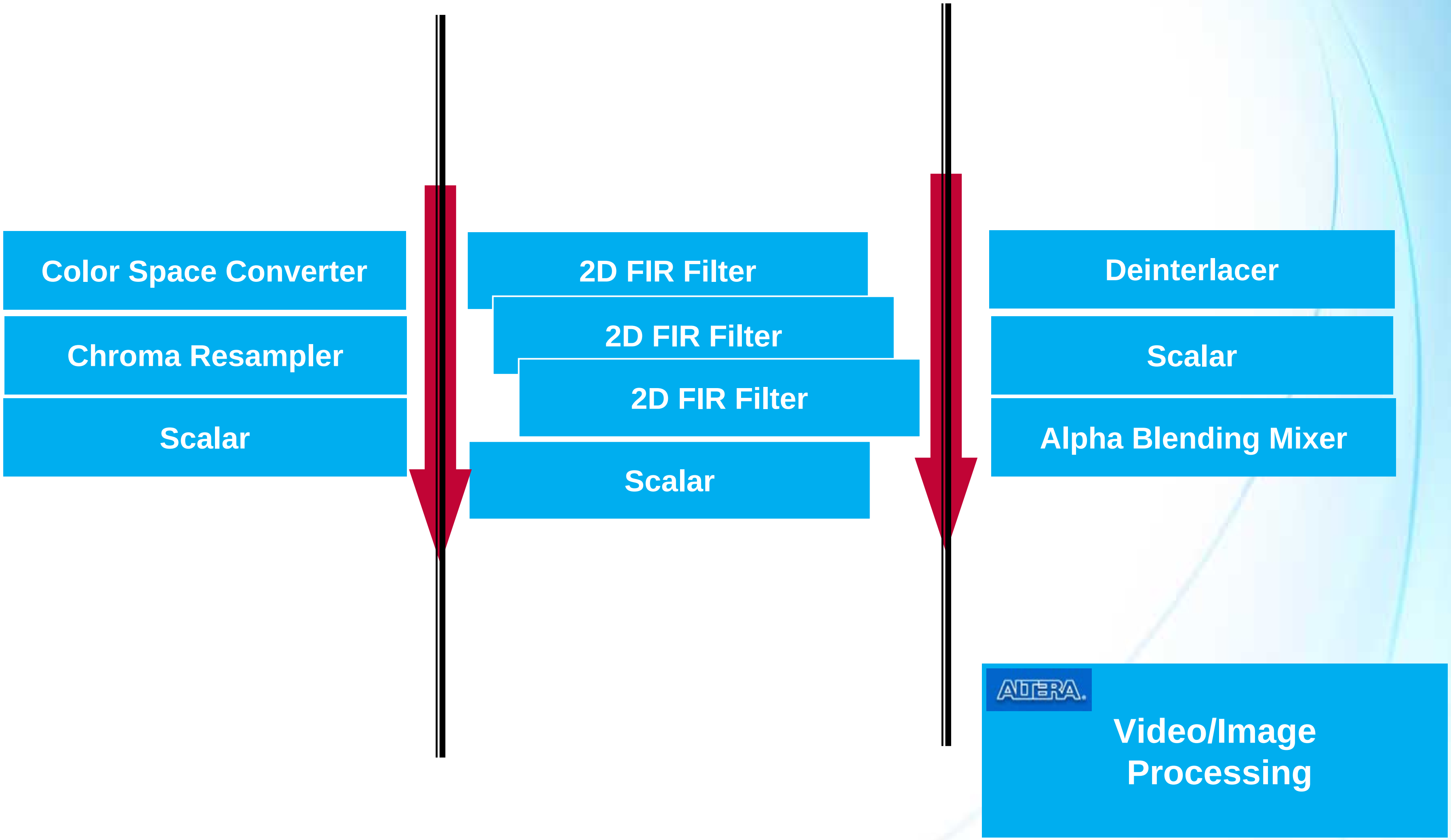
# Key Trend for “Video in FPGA” – 2

# HD → Dramatic Increase in Bits

| Image size | Frame size: (Total # of pixels) | Frame size: (Assume 10 bits per pixel) | Data rate: (Assuming 60 frames per second (FPS)) |
|---|---|---|---|
| 1920 X 1080p | 1920 x 1080 = 2.08M pixels | 62 Mbits or 7.78 Mbytes | 3,732 Mbps |
| 1920 X 1080i | 1920 x 1080 x 0.5 = 1.04M pixels | 31 Mbits or 3.89 Mbytes | 1,866 Mbps |
| 1280 X 720p | 1280 x 720 = 921K pixels | 27.7 Mbits or 3.46 Mbytes | 1,659 Mbps |
| SD 720 x 480i | 720 x 480 x 0.5 = 173K pixels | 5.2 Mbits or 0.65 Mbytes | 311 Mbps |

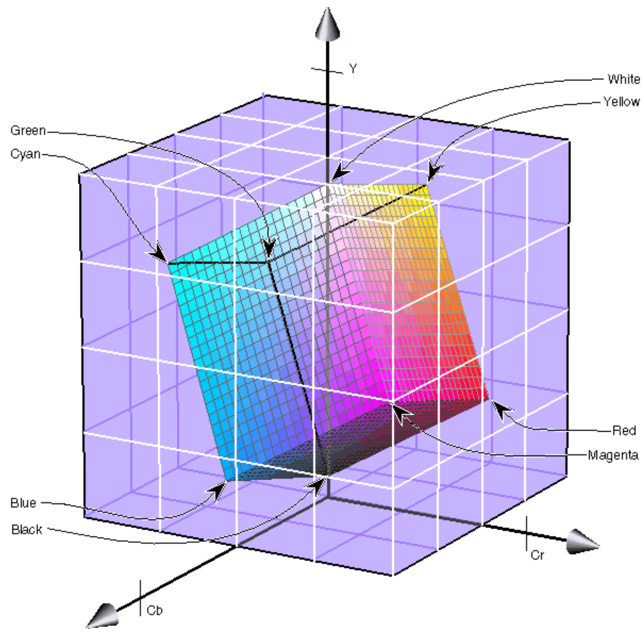
- These numbers will change when we account for HSYNC and VSYNC signals, as well as for chroma downsampling
- However, they are correct in a relative sense

# Typical Video Signal Chains



# Video Image Processing Basics

# Color Space: Basics



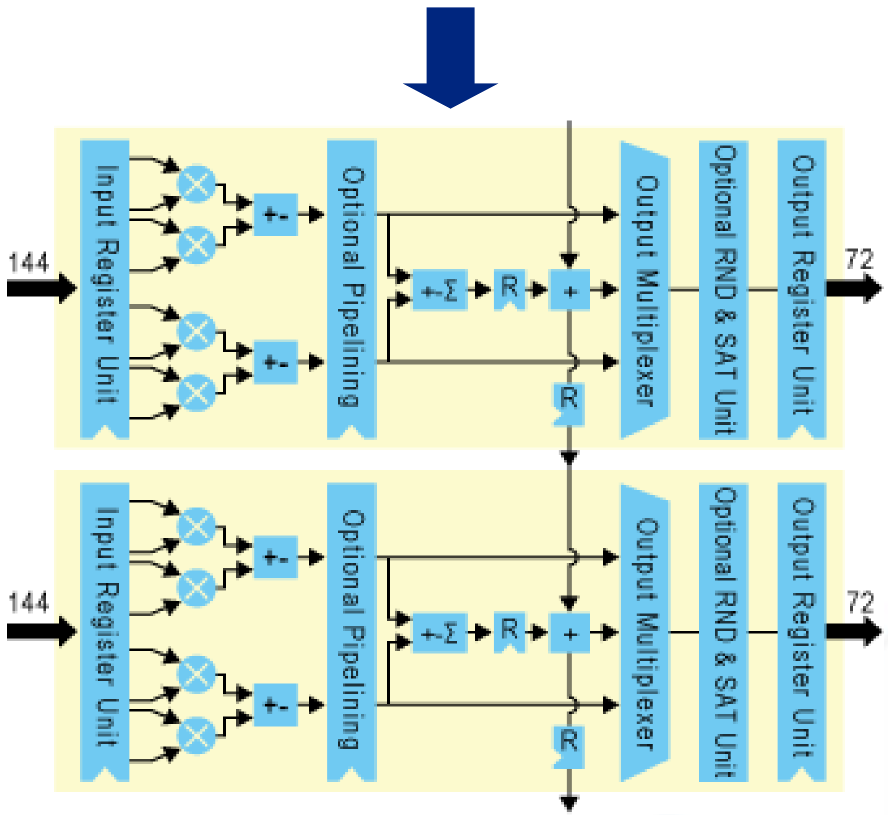

- A color space is a method by which we can specify, create, and visualize color
- Computers describe a color stimulus in terms of the excitations of red, green, and blue phosphors on the CRT faceplate
- Printers describe a color stimulus in terms of the reflectance and absorbance of cyan, magenta, yellow, and black inks on the paper

# Color Space Conversion: Basics

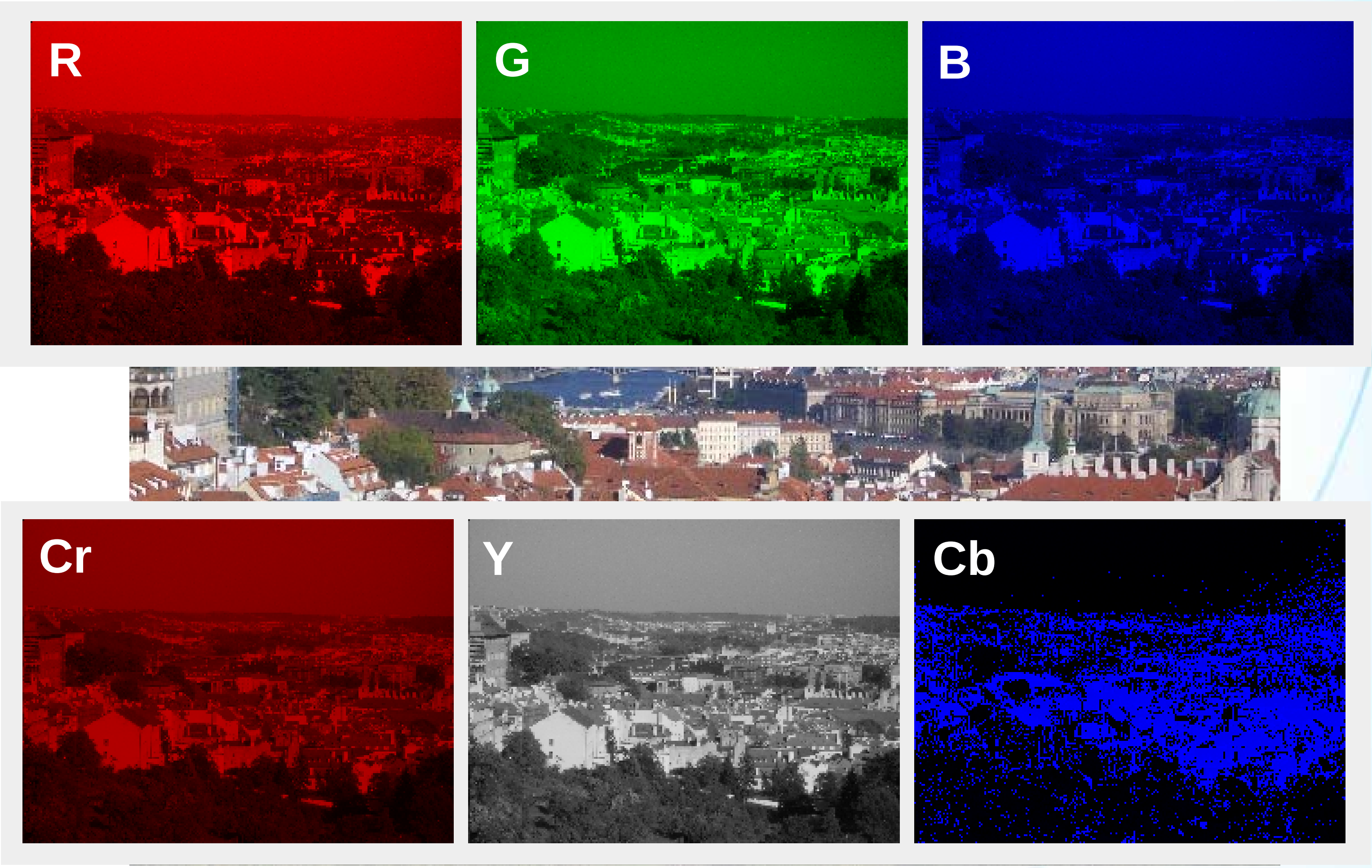
$$Y = R*0.299 + G*0.587 + B*0.114$$

$$CR = R*(-0.169) + G*(-0.332) + B*0.500 + 128$$

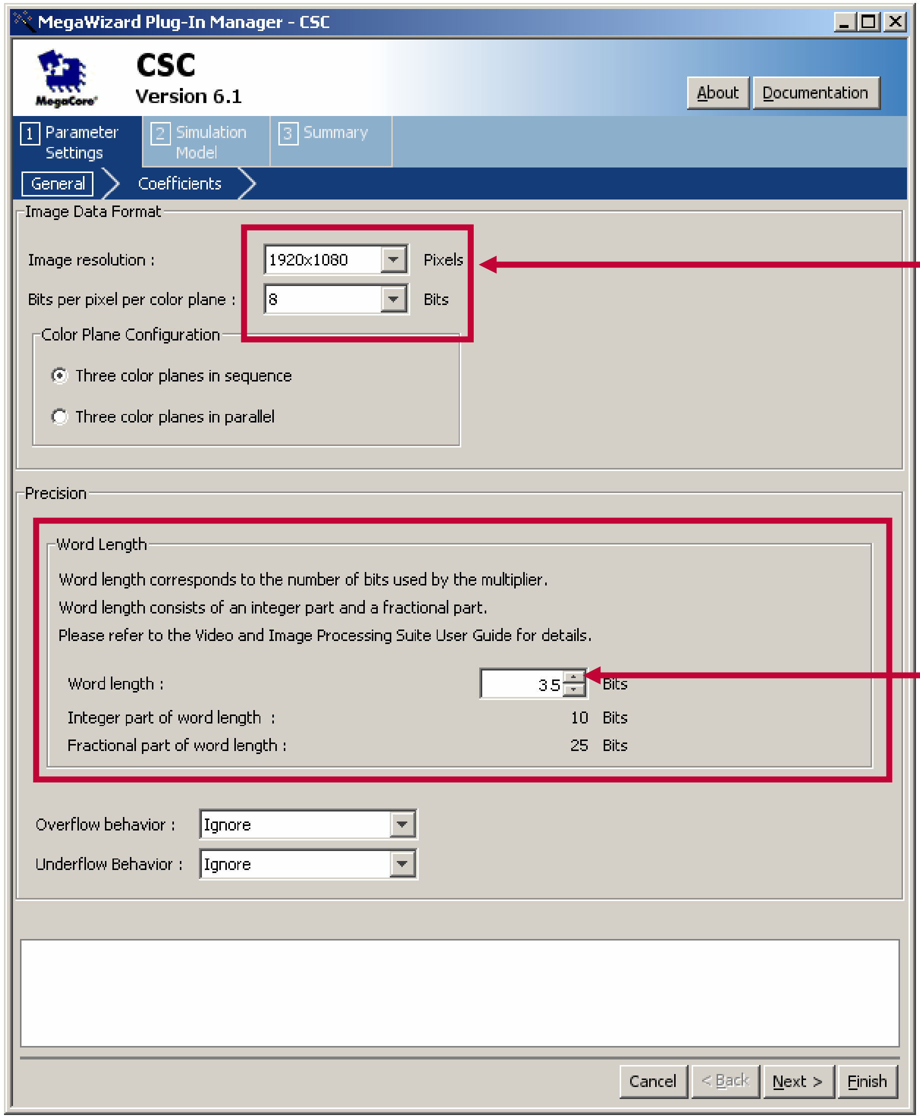
$$Cb = R*0.500 + G*(-0.419) + B*(-0.0813) + 128$$



# RGB to YCrCb



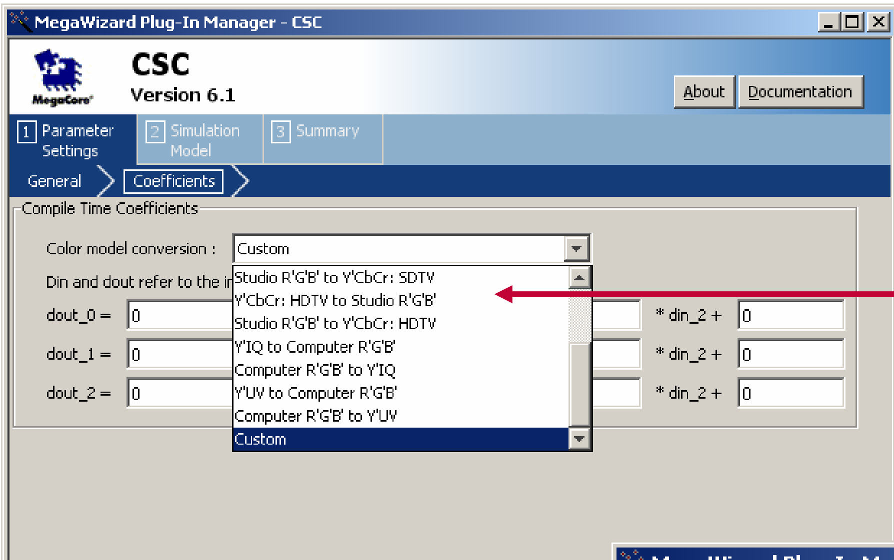
# Color Space Conversion IP



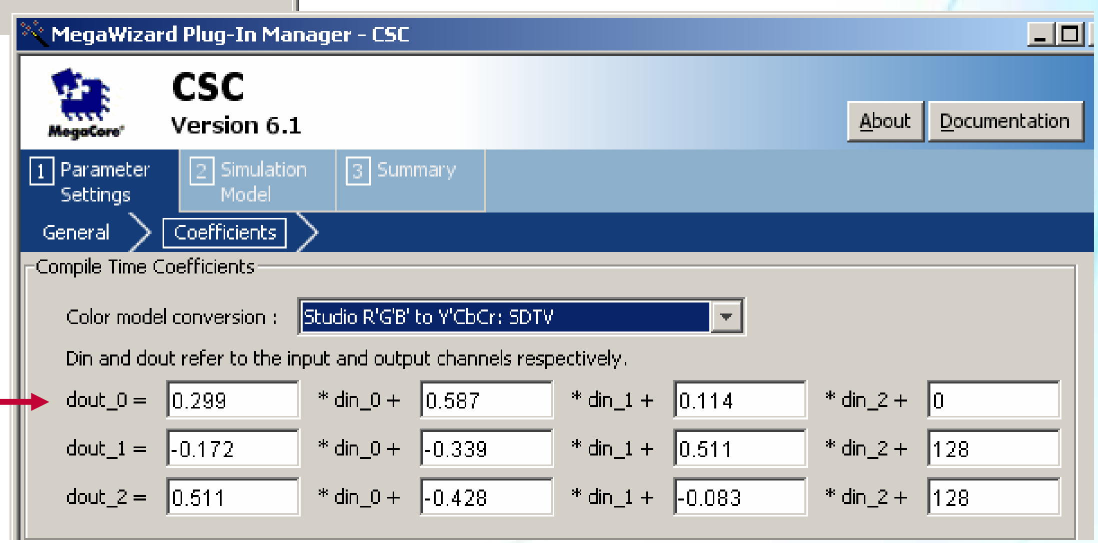

**Select the size of the image Select bits per pixel Three color planes are assumed**

**Select the precision of the multiplier**

# Color Space Conversion IP

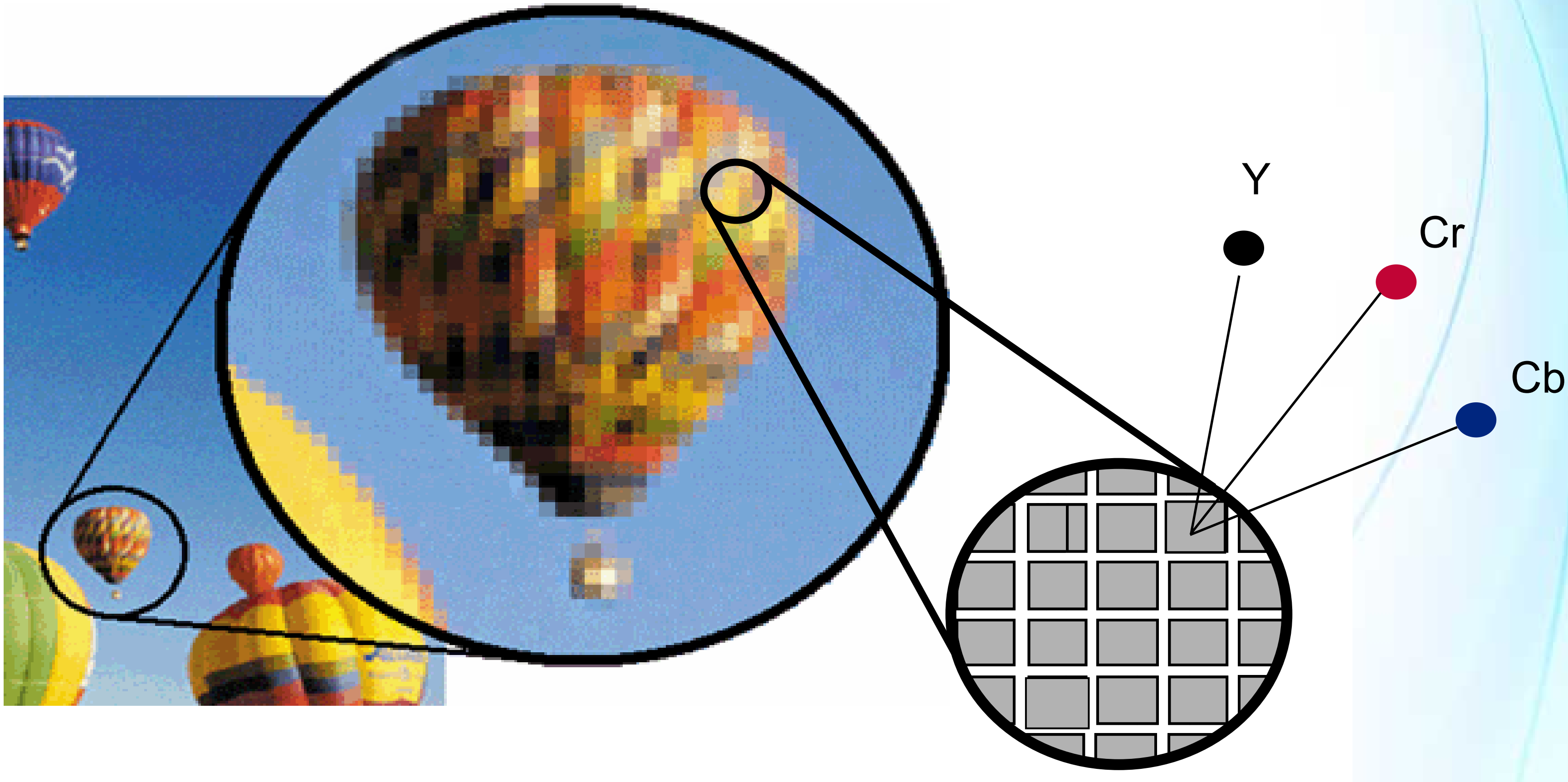



**Choose the color space conversion**

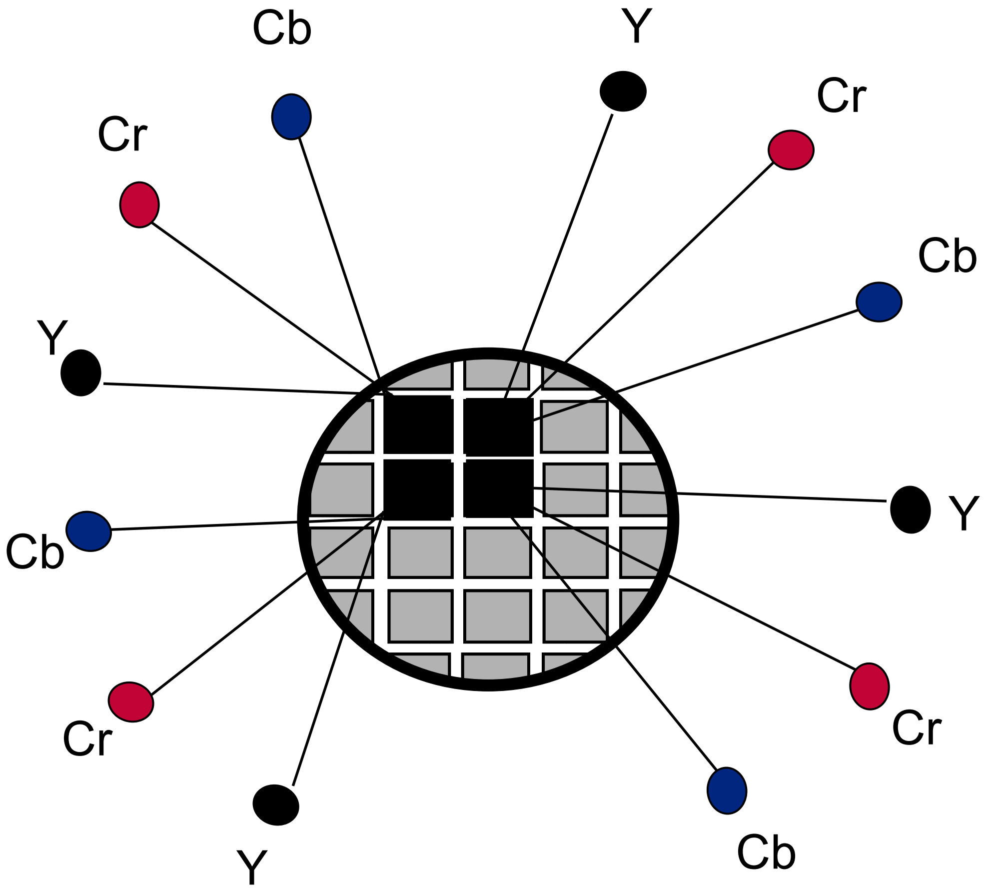



**The core can automatically select the co-efficient, or you can enter custom co-efficient**

# Chroma Downsampling: Basics

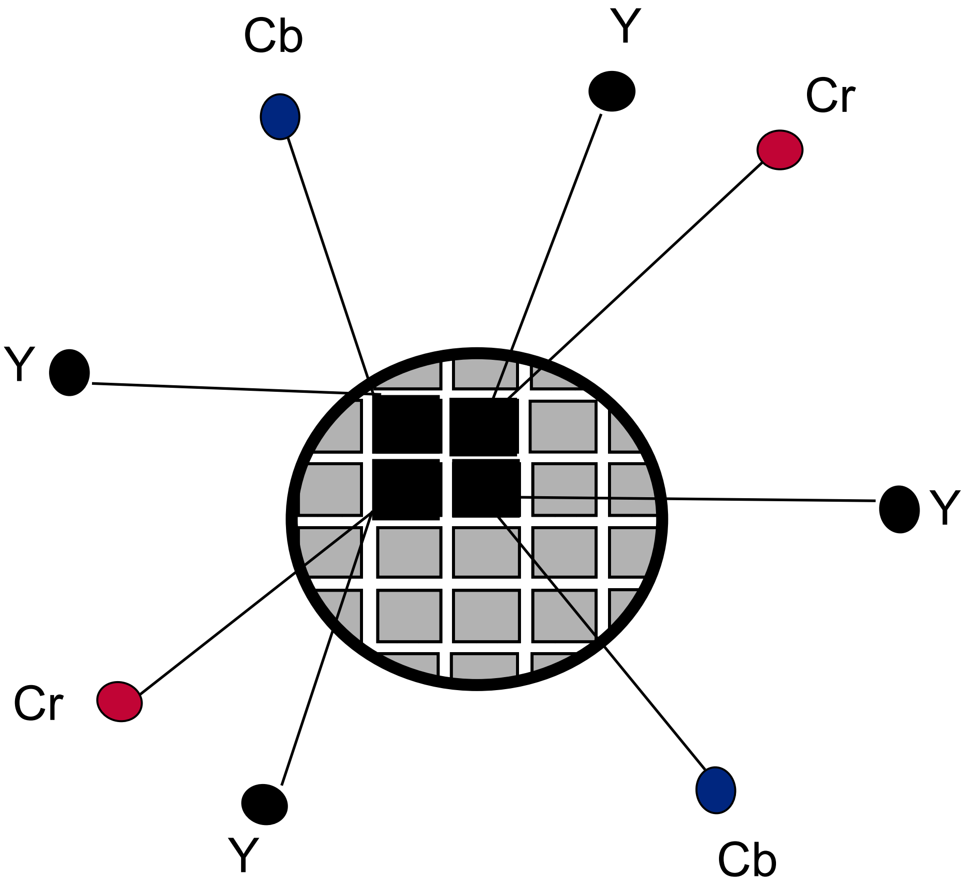


# Chroma Downsampling: Basics



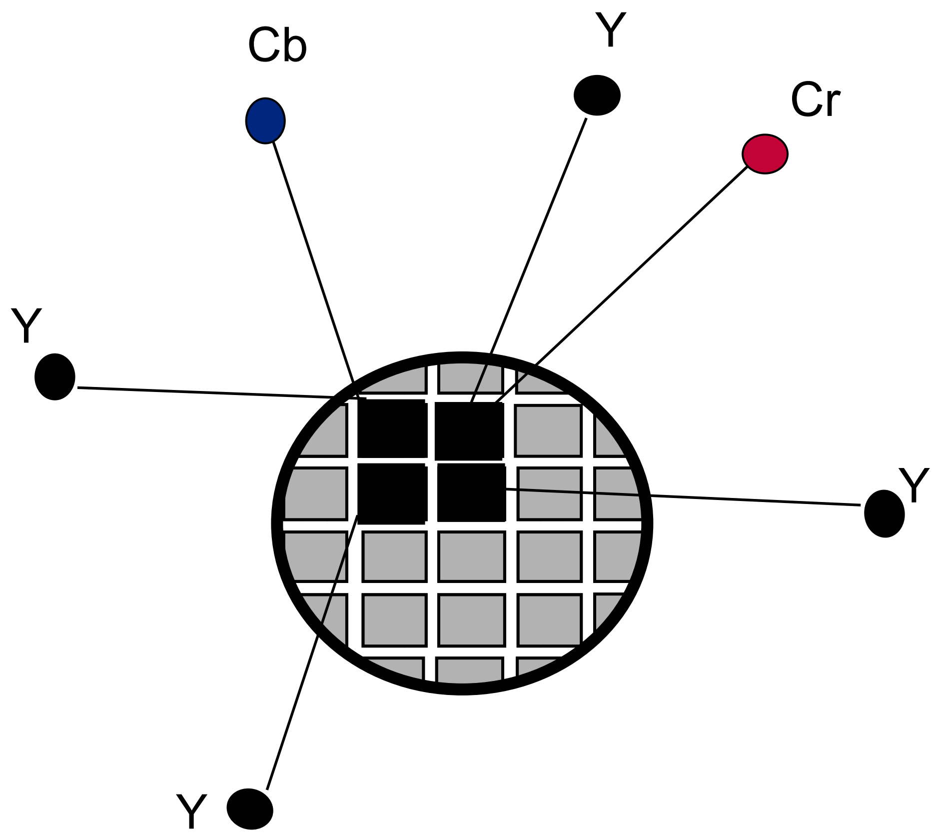

- Per pixel
  - Y (10 bits)
  - Cr (10 bits)
  - Cb (10 bits)
- Total bits
  - 40 bits for Y
  - 40 bits for Cr
  - 40 bits for Cb
- 4:4:4 chroma subsampling
- Bits for 4 pixels: 120
- Bit/pixel = 30

# Chroma Downsampling: Basics




- Per pixel
  - Y (10 bits)
  - Cr (10 bits)
  - Cb (10 bits)
- Drop Cr, Cb for alternate pixels, total bits
  - 40 bits for Y
  - 20 bits for Cr
  - 20 bits for Cb
- 4:2:2 chroma subsampling
- Bits for 4 pixels: 80
- Bit/pixel = 20

# Chroma Downsampling: Basics

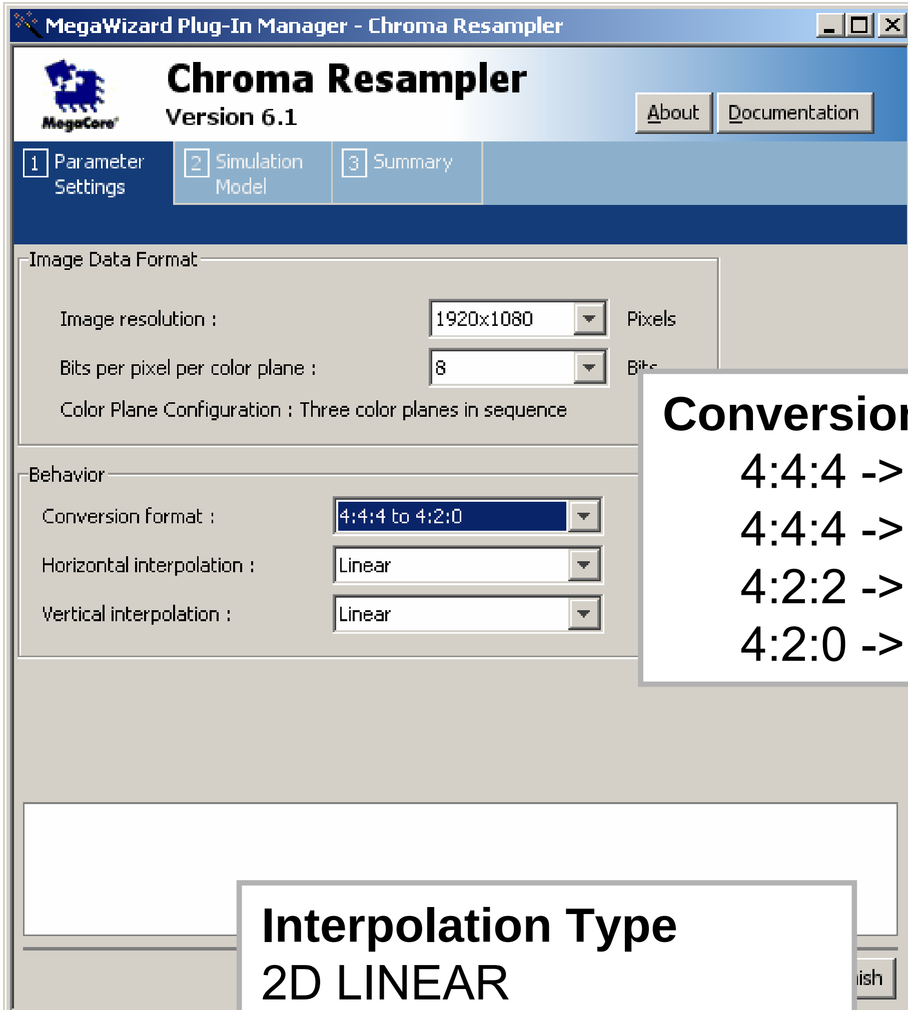



- Per pixel
  - Y (10 bits)
  - Cr (10 bits)
  - Cb (10 bits)
- Drop Cr, Cb for alternate pixels
- Drop Cr and Cb for the second line
- Total bits
  - 40 bits for Y
  - 10 bits for Cr
  - 10 bits for Cb
- 4:2:0 chroma subsampling
- Bits for 4 pixels: 60
- Bit/pixel = 15

# Why Chroma Downsampling?

| Image size | Frame size: (Total # of pixels) | Frame size: (Assume 10 bits per pixel and 4:4:4) | Frame size: (Assume 10 bits per pixel and 4:2:2) | Frame size: (Assume 10 bits per pixel and 4:2:0) |
|---|---|---|---|---|
| 1920 X 1080p | 1920 x 1080 = 2M pixels | 60 Mbits | 40 Mbits | 30 Mbits |
| 1920 X 1080i | 1920 x 1080 x 0.5 = 1M pixels | 30 Mbits | 20 Mbits | 15Mbits |
| 1280 X 720p | 1280 x 720 = 900K pixels | 27 Mbits | 18 Mbits | 13.5 Mbits |
| SD 720 x 480i | 720 x 480 x 0.5 = 173K pixels | 5.19 Mbits | 3.46 Mbits | 2.595 Mbits |

# Chroma Resampling IP

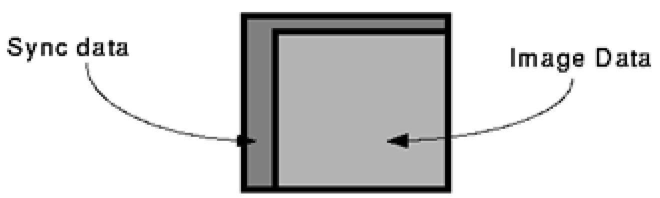



**Conversion Format**
4:4:4 -> 4:2:2
4:4:4 -> 4:2:0
4:2:2 -> 4:4:4
4:2:0 -> 4:4:4

**Interpolation Type**
2D LINEAR
2D NEAREST NEIGHBOR

# Calculating Data Rates

| Image size | Frame size | Chroma sub sample/bits per color plane/FPS | Bit/s transfer rate |
|---|---|---|---|
| 1920 x 1080p | 2200 x 1125 | 4:2:2/10/60 | 2200 x 1125 x 20 x 60 = 2.97 Gbps |
| 1920 x 1080i | 2200 x 1125 | 4:2:2/10/60 | 2200 x 1125 x 20 x 60 x 0.5 = 1.485 Gbps |
| 1280 x 720p | 1650 x 750 | 4:2:2/10/60 | 1650 x 750 x 20 x 60 = 1.485 Gbps |
| 720 x 480i | 858 x 525 | 4:2:2/10/60 | 858 x 525 x 20 x 60 x 0.5 = 270 Mbps |




1080p-SDI* rate

HD-SDI rate

HD-SDI rate

**SDI = serial digital interface**

# Scaling: Basics

D1/SDTV: 720 x 480

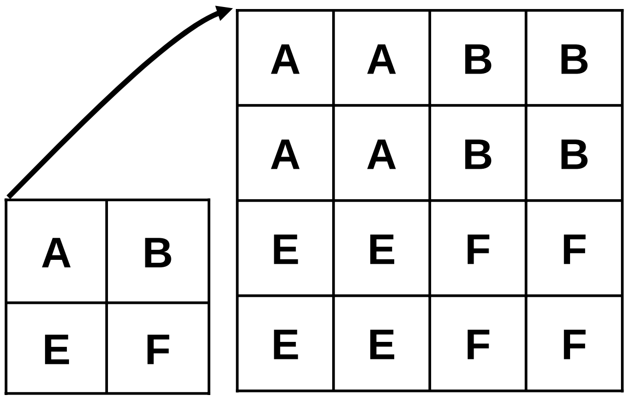
HDTV 1080p: 1920 x 1080

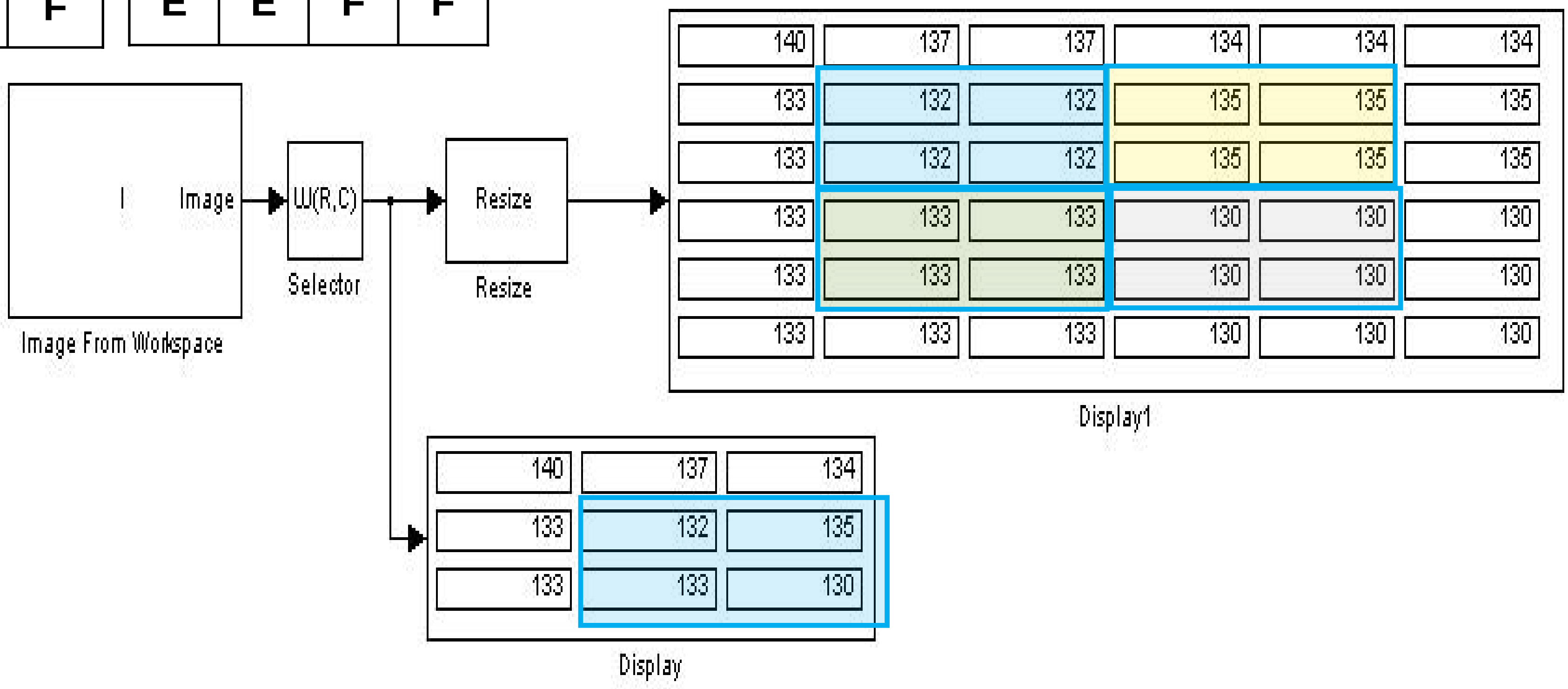
- Arbitrary input and output resolutions
- Bicubic, bilinear, and nearest neighbor
- Also with 7.1 → multi-tap (polyphase scaling)
- Real-time control of the scaling co-efficiency

# Scaling: Basics

- Nearest neighbor
  - Uses one pixel to generate the new pixel
- Bilinear
  - Uses up to 4 (2x2) pixels to generate the new pixel
- Bicubic
  - Uses up to 16 pixels (4x4) to generate the new pixel
- Multi-tap (polyphase … coming in 7.1)
  - Uses any arbitrary window size (M x N) to generate the new pixel value

# Nearest Neighbor Interpolation



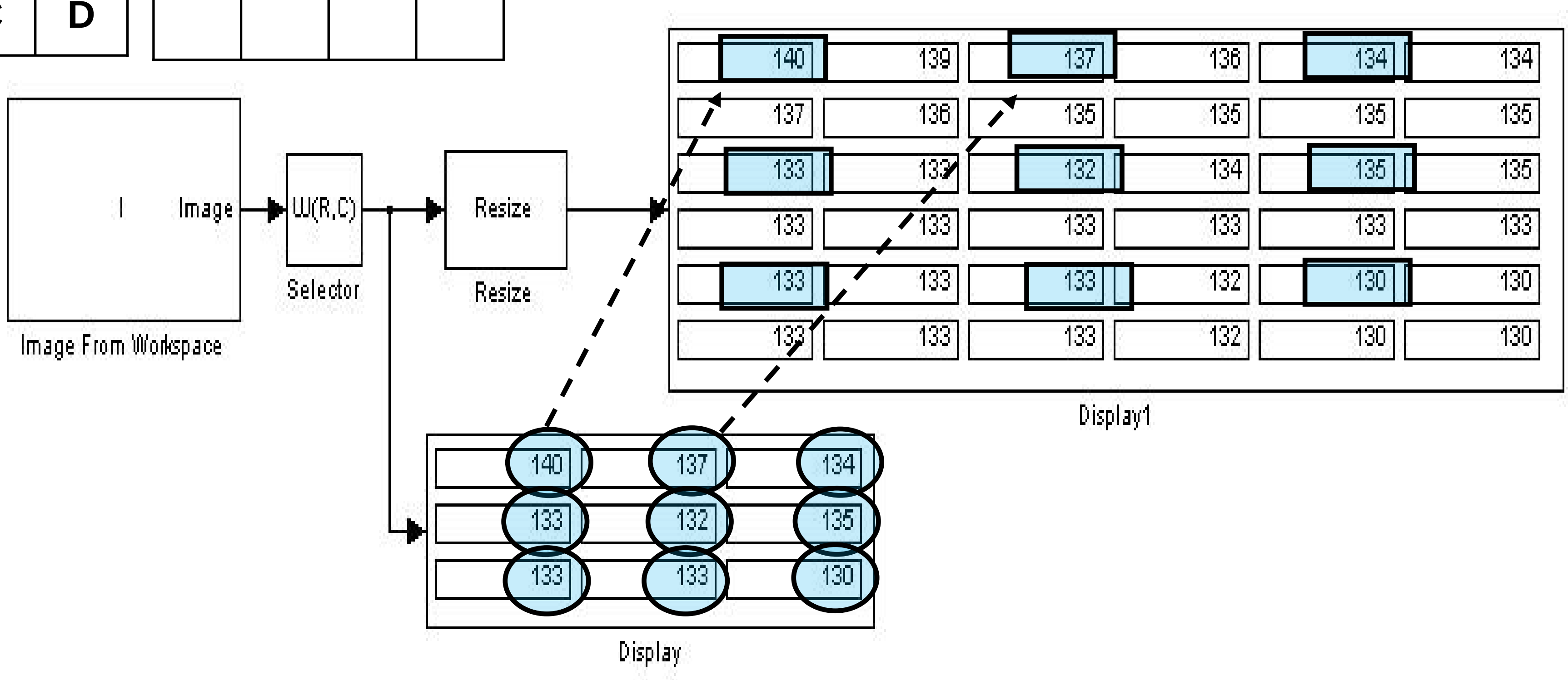
# Bilinear Interpolation

| | |
|---|---|
| A | B |
| C | D |

| | | | |
|---|---|---|---|
| A | f | B | |
| e | i | g | |
| C | h | D | |
| | | | |

f= (A+B)/2
g= (B+D)/2
h= (C+D)/2
e= (A+C)/2

i = (A+B+C+D)/4



# Scaling Comparison by Different Methods

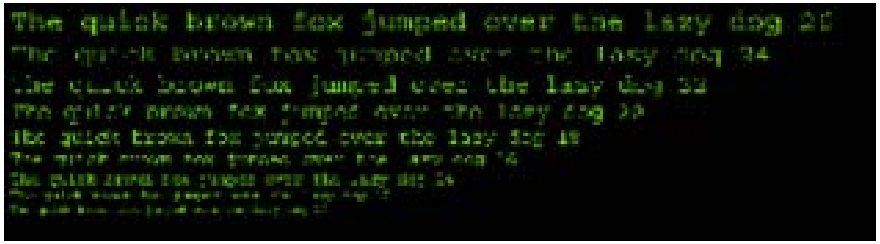


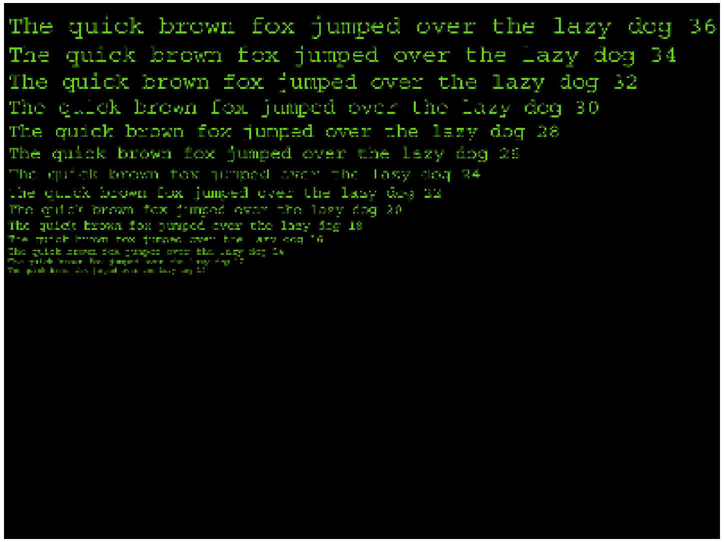
**Bicubic Scaling**

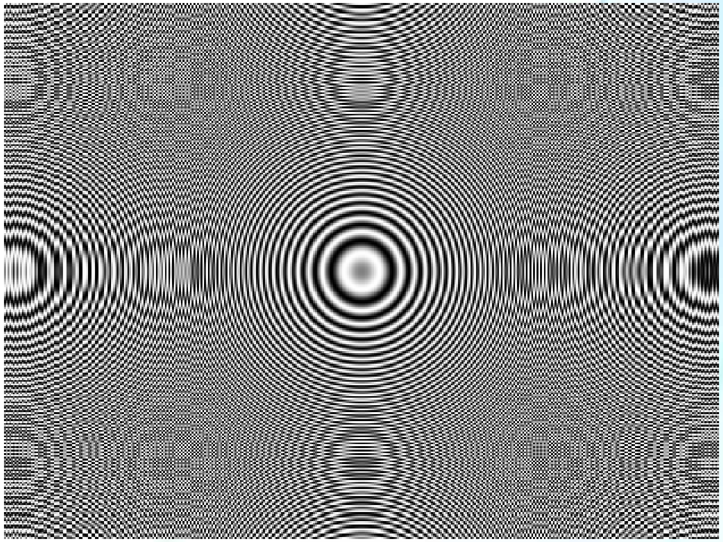
# Scaling: Basics

- Nearest neighbor
  - Uses one pixel to generate the new pixel
- Bilinear
  - Uses up to 4 (2x2) pixels to generate the new pixel
- Bicubic
  - Uses up to 16 pixels (4x4) to generate the new pixel
- Multi-tap (polyphase … coming in 7.1)
  - Uses any arbitrary window size (M x N) to generate the new pixel value
  - Very useful when downscaling

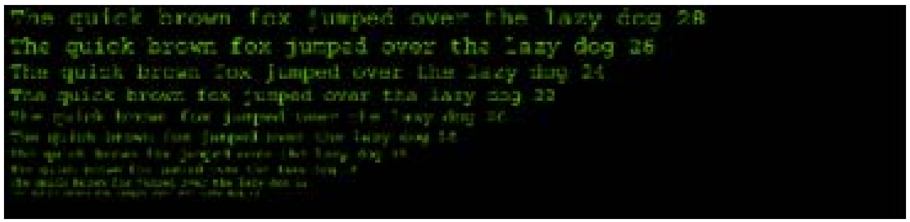
# Nearest Neighbor





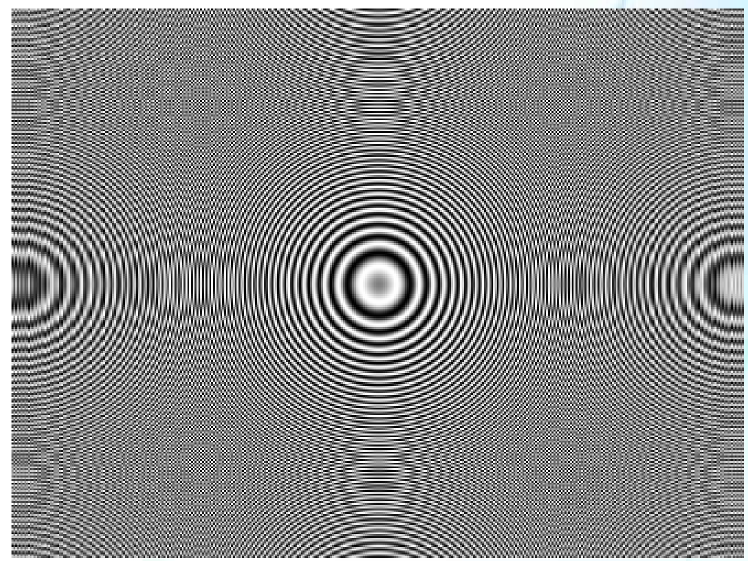



# Bilinear

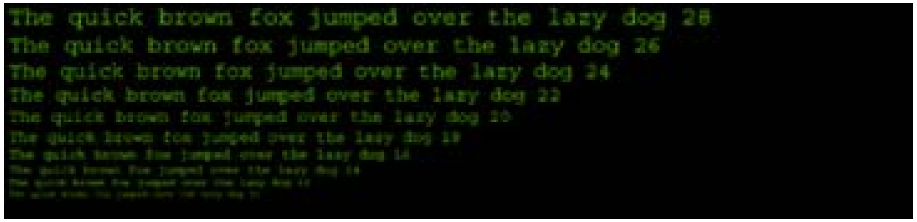


The quick brown fox jumped over the lazy dog 28
The quick brown fox jumped over the lazy dog 26
The quick brown fox jumped over the lazy dog 24
The quick brown fox jumped over the lazy dog 22

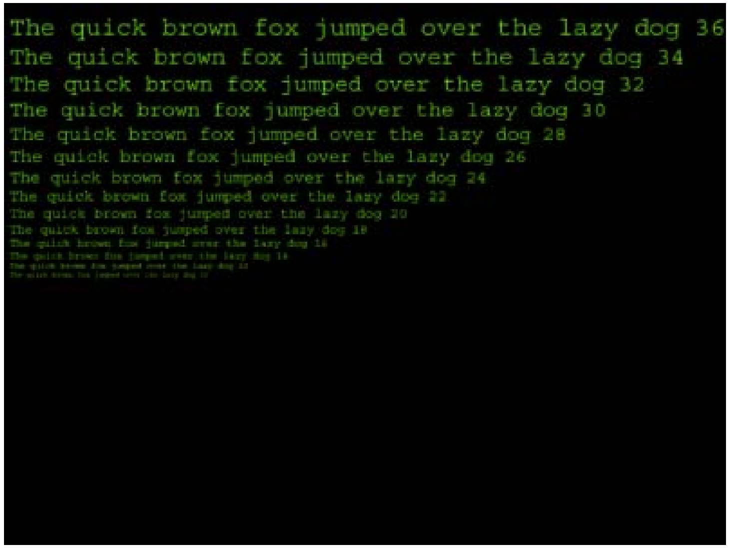

The quick brown fox jumped over the lazy dog 36
The quick brown fox jumped over the lazy dog 34
The quick brown fox jumped over the lazy dog 32
The quick brown fox jumped over the lazy dog 30
The quick brown fox jumped over the lazy dog 28
The quick brown fox jumped over the lazy dog 26
The quick brown fox jumped over the lazy dog 24
The quick brown fox jumped over the lazy dog 22

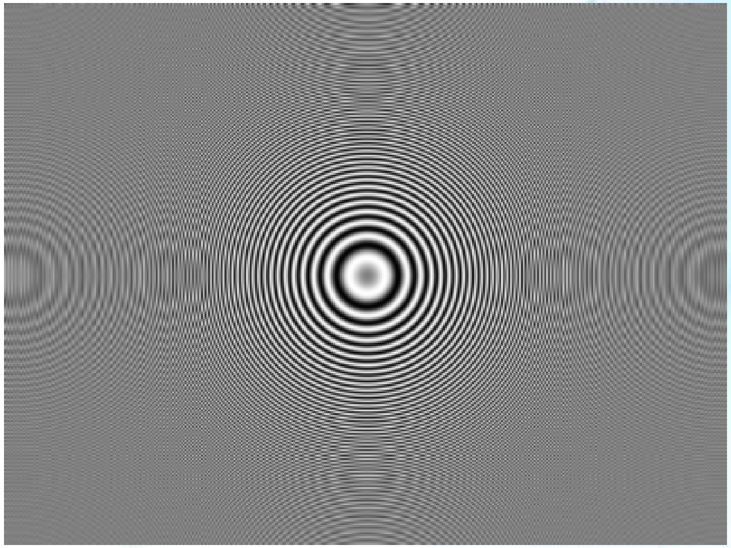
# 5-Tap (5 x 5)



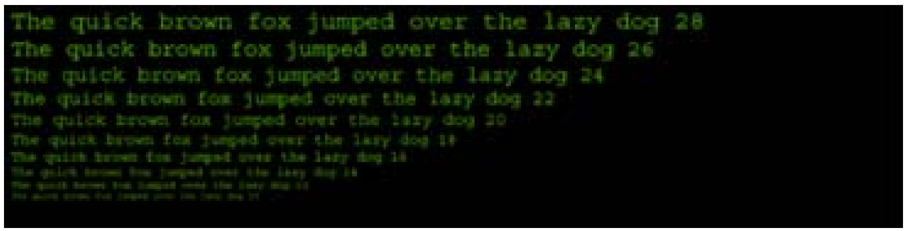
The quick brown fox jumped over the lazy dog 28
The quick brown fox jumped over the lazy dog 26
The quick brown fox jumped over the lazy dog 24
The quick brown fox jumped over the lazy dog 22
The quick brown fox jumped over the lazy dog 20

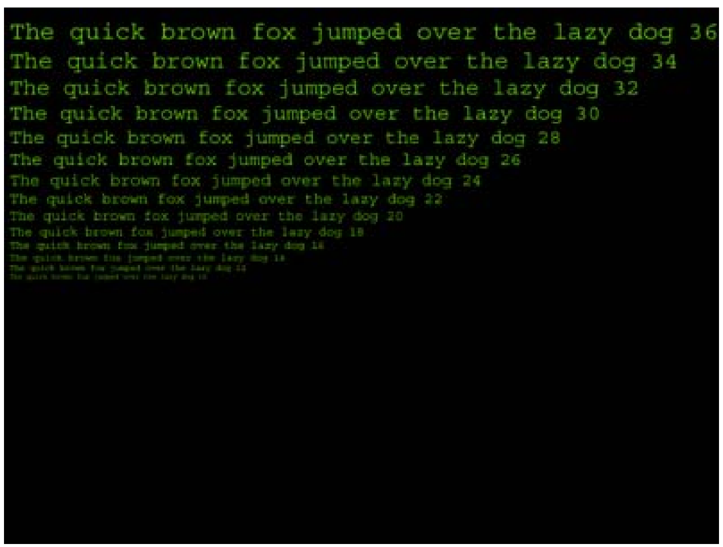
The quick brown fox jumped over the lazy dog 36
The quick brown fox jumped over the lazy dog 34
The quick brown fox jumped over the lazy dog 32
The quick brown fox jumped over the lazy dog 30
The quick brown fox jumped over the lazy dog 28
The quick brown fox jumped over the lazy dog 26
The quick brown fox jumped over the lazy dog 24
The quick brown fox jumped over the lazy dog 22
The quick brown fox jumped over the lazy dog 20
The quick brown fox jumped over the lazy dog 18

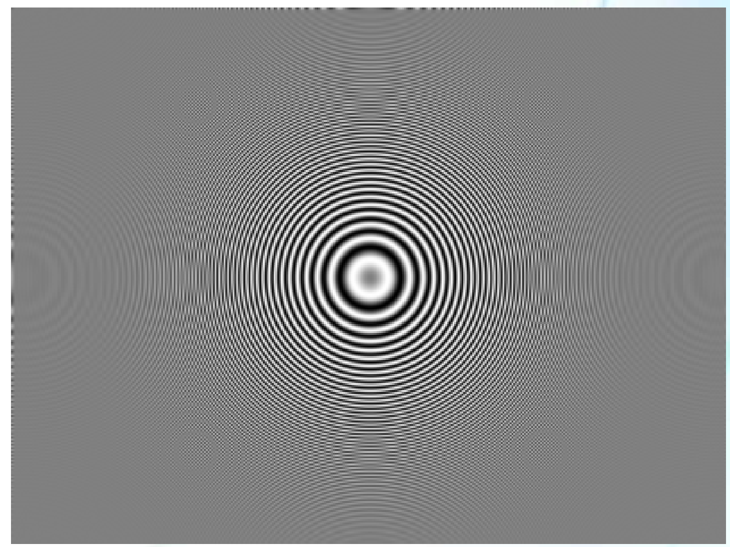
# 9-Tap (9 x 9)



The quick brown fox jumped over the lazy dog 28
The quick brown fox jumped over the lazy dog 26
The quick brown fox jumped over the lazy dog 24
The quick brown fox jumped over the lazy dog 22
The quick brown fox jumped over the lazy dog 20

The quick brown fox jumped over the lazy dog 36
The quick brown fox jumped over the lazy dog 34
The quick brown fox jumped over the lazy dog 32
The quick brown fox jumped over the lazy dog 30
The quick brown fox jumped over the lazy dog 28
The quick brown fox jumped over the lazy dog 26
The quick brown fox jumped over the lazy dog 24
The quick brown fox jumped over the lazy dog 22
The quick brown fox jumped over the lazy dog 20
The quick brown fox jumped over the lazy dog 18

# Upscaling

400 x 300 scaled to 800 x 600

# Different Upscaling Results

8-Tap (8 x 8)

4-Tap (4 x 4)

Bicubic

Bilinear

Nearest neighbor

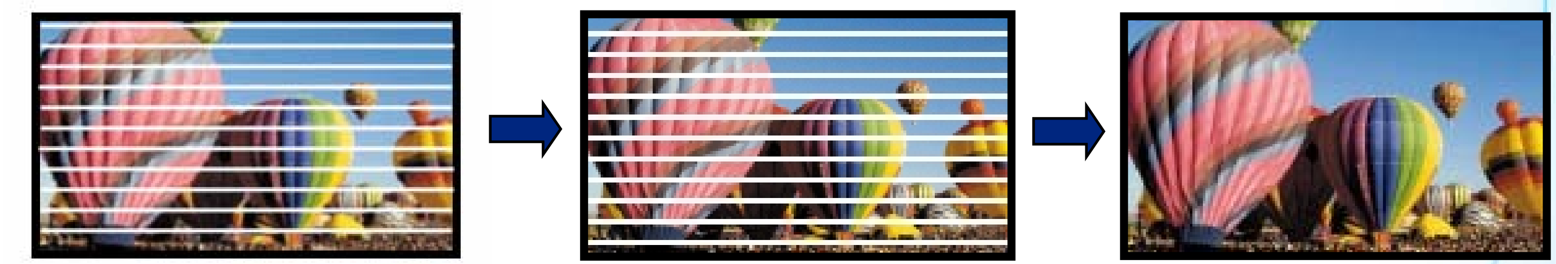
# Upscaling: Things to Remember

- Generally you can get very good results with bicubic or 4-tap scaling
- There is not much improvement beyond 4x4

# Deinterlacing: The Basics

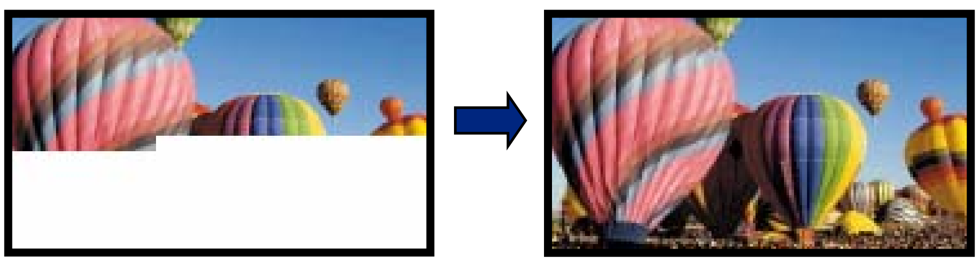
## Interlace

**First all odd lines scanned (1/60sec)** **…then all even lines (1/60sec)** **…presenting a full picture (1/30sec)**

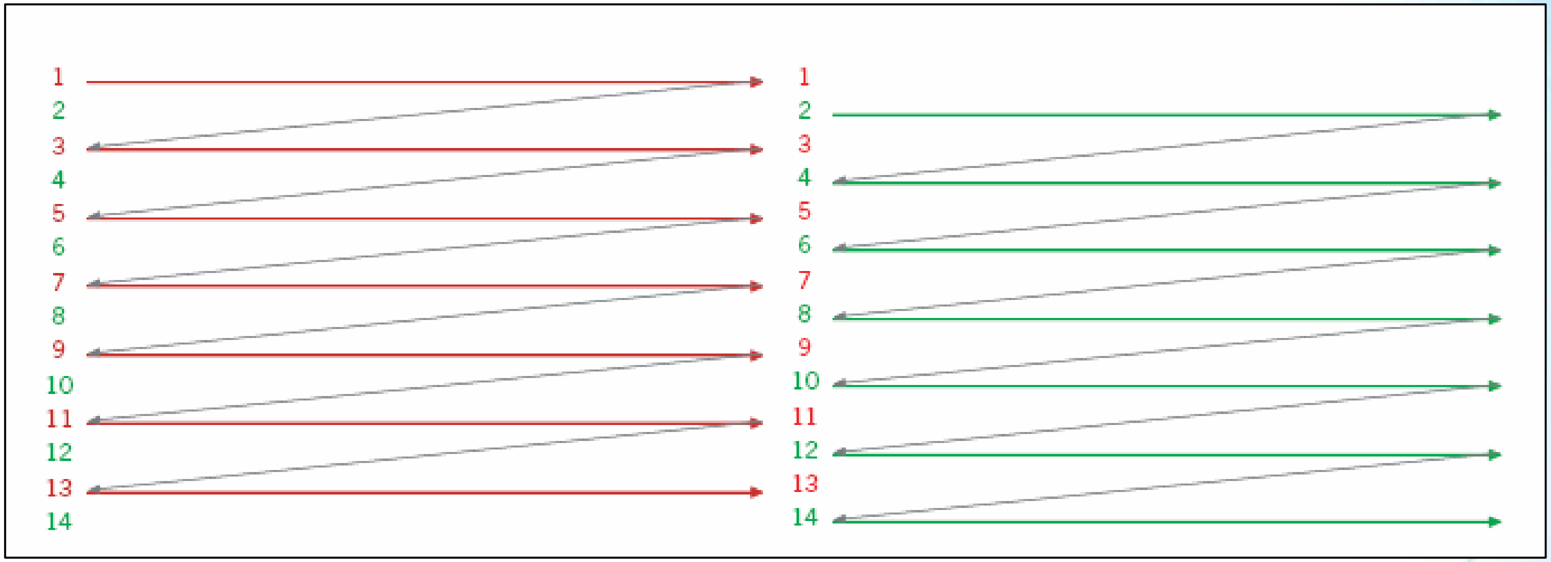


## Progressive

**All lines scanned in single pass** **…presenting a full picture (1/60sec)**



# Deinterlacing: The Basics

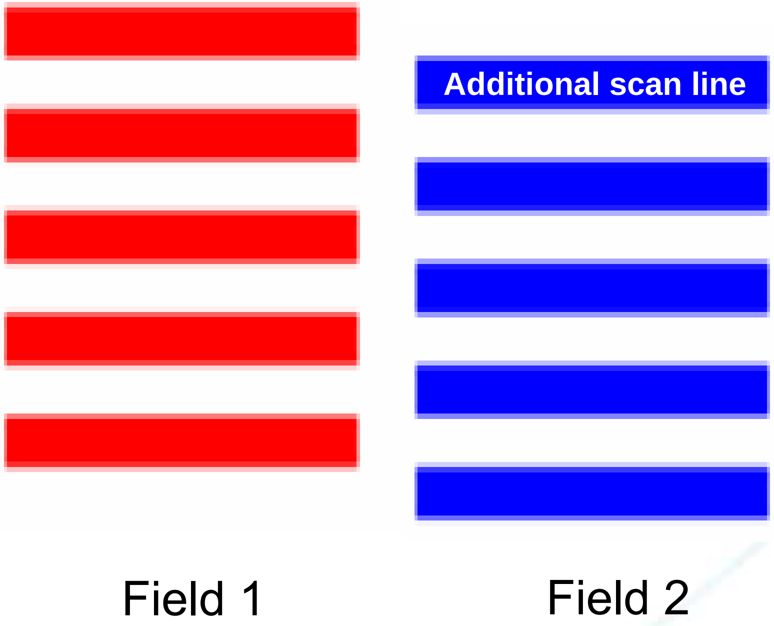



Time 't'    Time 't+1/60s'

- Because of the time intermix (1 frame = field @time 't' + field @time 't+1/60') it is impossible to:
  - Deinterlace a frame AND
  - Keep 60 frames/second AND
  - Keep the full quality (=all information for a picture)
- You will have to alter at least one of those points
  - Except, when there is no motion

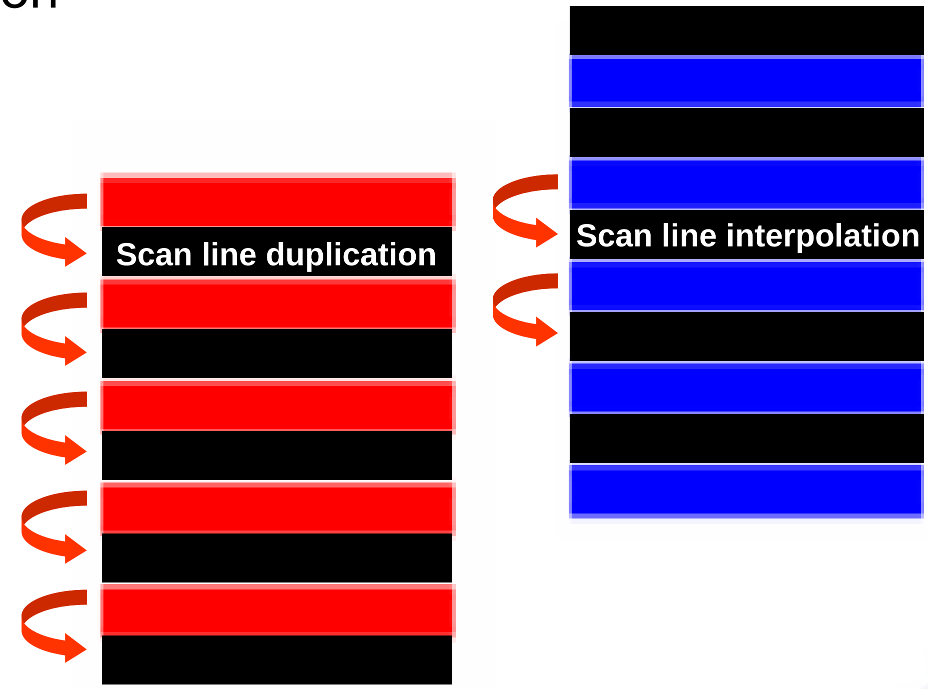
# Deinterlacing: The Basics

- How do we deinterlace video?
  - ‘Bob’ deinterlacing
  - One field of the video is made into a complete frame
  - Because each field has only half the lines of a full frame, additional scan lines have to be added to create a frame



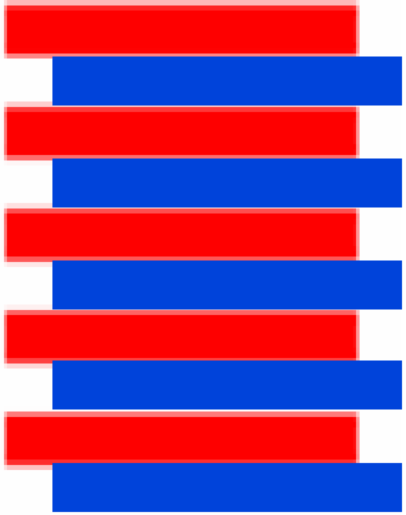
# Deinterlacing: The Basics

- Generating the additional scan line
  - Duplication
  - Interpolation



# Deinterlacing: The Basics

- How do we deinterlace video?
  - ‘Weave’ deinterlacing
  - This method simply combines the two fields into one frame
  - This methodology is good when there is not much motion between two successive fields
  - Weave leads to artifacts when there is motion

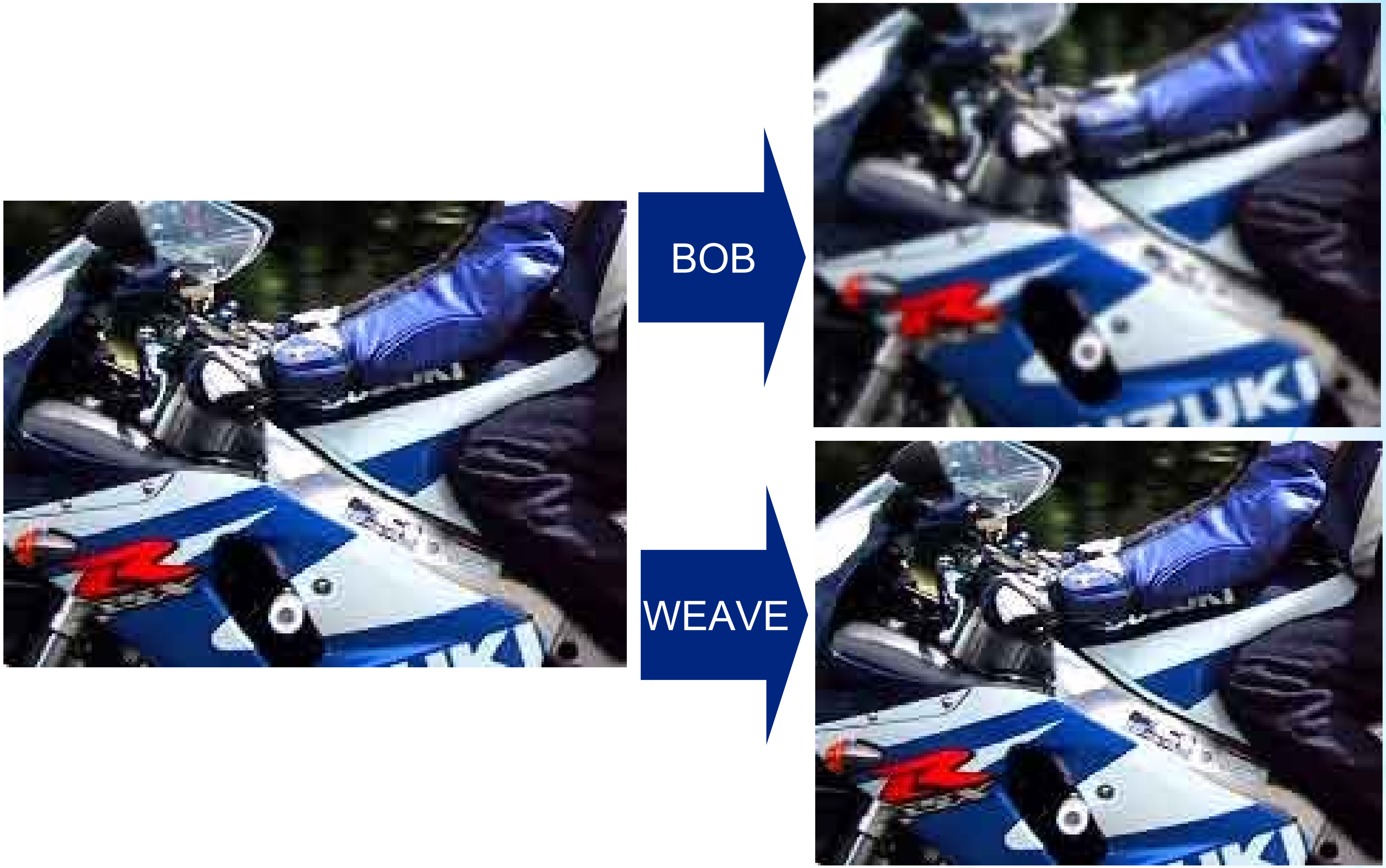


# Deinterlacing: With Motion

BOB

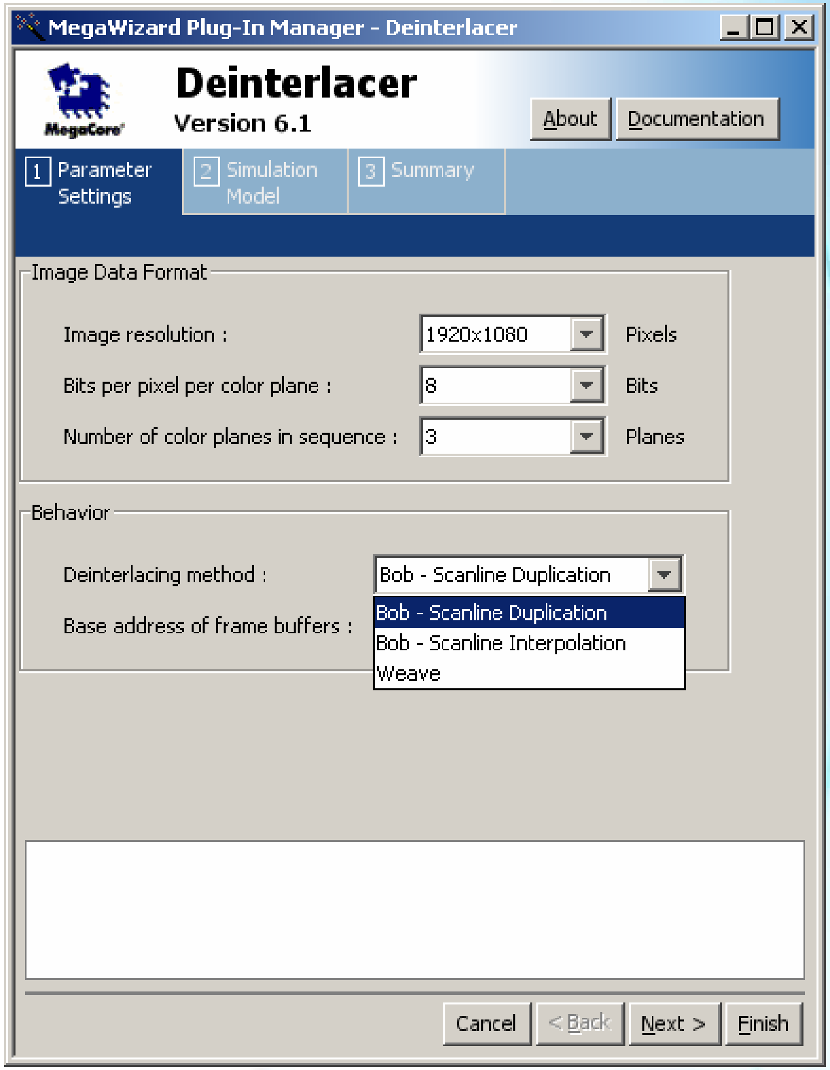
WEAVE

# Deinterlacing: Without Motion

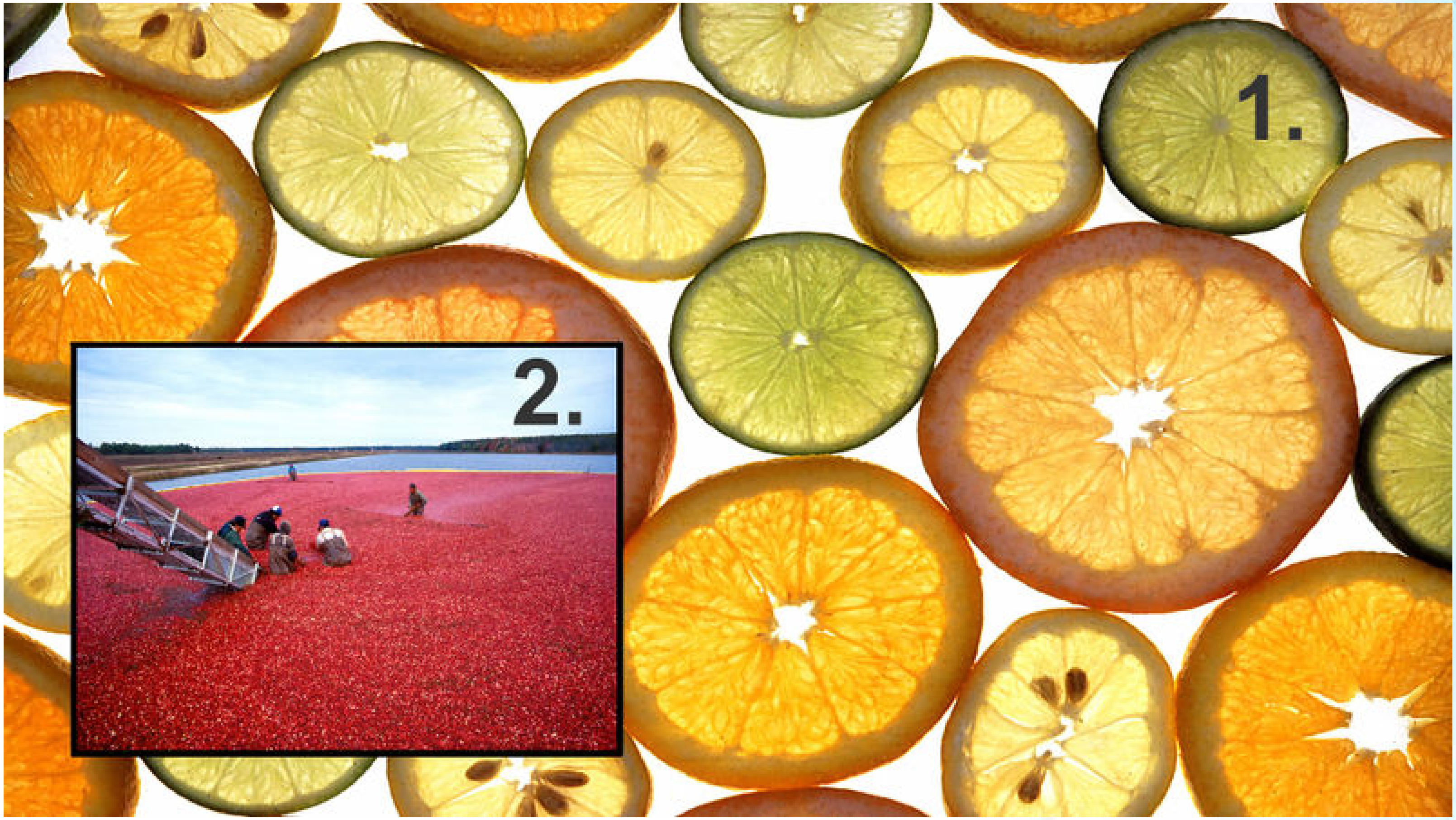


# Deinterlacing Applications

- Deinterlacing is used whenever you want to
  - Grab still image from video
  - Play video on a non-interlaced display
  - Compress video
- Applications
  - Video surveillance – before compression/storage
  - Video conferencing – to display on a non-interlaced screen
  - Broadcast – before compression and video switching



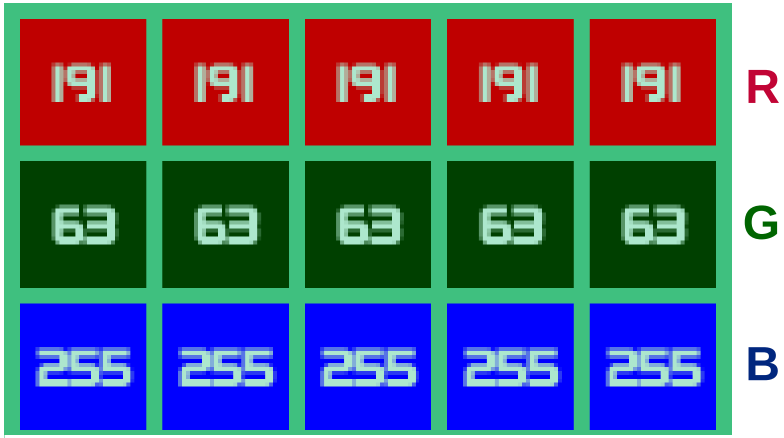
# Image Blending: For Onscreen Display (OSD) and Picture-in-Picture (PiP)



# Alpha Image Blending: Basics

- Alpha image blending is the process of digitally assembling multiple images to make a final image
- The basic operation used is known as 'alpha blending', where an opacity value, 'α', is used to control the proportions of two input pixel values that end up a single output pixel
- Consider three pixels:
  - Foreground pixel, f
  - Background pixel, b
  - Composited pixel, c
- Also alpha (α) is the opacity value of the foreground pixel
  - $\alpha=1$ for opaque foreground, $\alpha=0$ for a completely transparent foreground

# Alpha Blending: Basics




- The color is RGB (191, 63, 255)
- The alpha values go from 255 (fully opaque) to 0 (fully transparent)
- The actual resulting merged color is computed this way:

  (image color × alpha) + (background color × (100% - alpha))

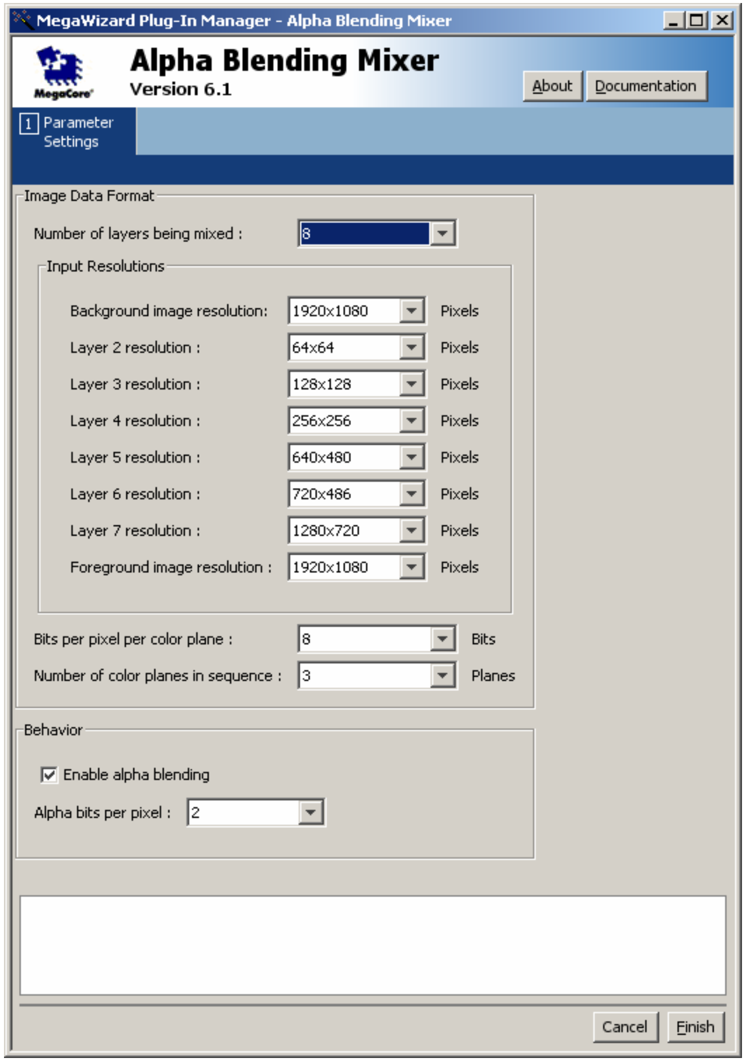
**Background**

# Alpha Image Blending: Basics

- Composite RGB image can be calculated by
  - $c_r = \alpha f_r + (1 - \alpha) b_r$
  - $c_g = \alpha f_g + (1 - \alpha) b_g$
  - $c_b = \alpha f_b + (1 - \alpha) b_b$
- This can also be written as
  - $c_r = b_r + \alpha (f_r - b_r)$
  - $c_g = b_g + \alpha (f_g - b_g)$
  - $c_b = b_b + \alpha (f_b - b_b)$

One multiply operation vs. two

# Alpha Blending IP Core




- In PIP, background video is played in the center of the screen, while smaller square video clips are played in corners of the screen
- Multi-layer mixing (2 to 8 layers)
- Every foreground layer can use a different alpha value to control its transparency, resulting in true image blending effects

# Filtering in Video Image Processing (VIP)

- Various video image processing signal chains have to filter the input signals to
  - Remove noise
  - Smooth the image
  - Sharpen the image
  - Implement custom processing
- Altera® VIP solutions provide options to implement this filtering

# 2D Filtering to Enhance Images

ORIGINAL


ORIGINAL + NOISE

# 2D Filtering to Enhance Images

Low Pass Filtered

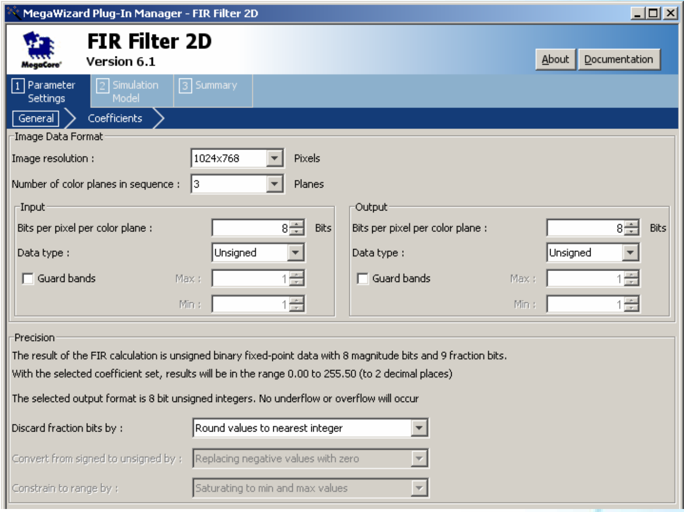

High Pass Filtered

# 2D Median Filter

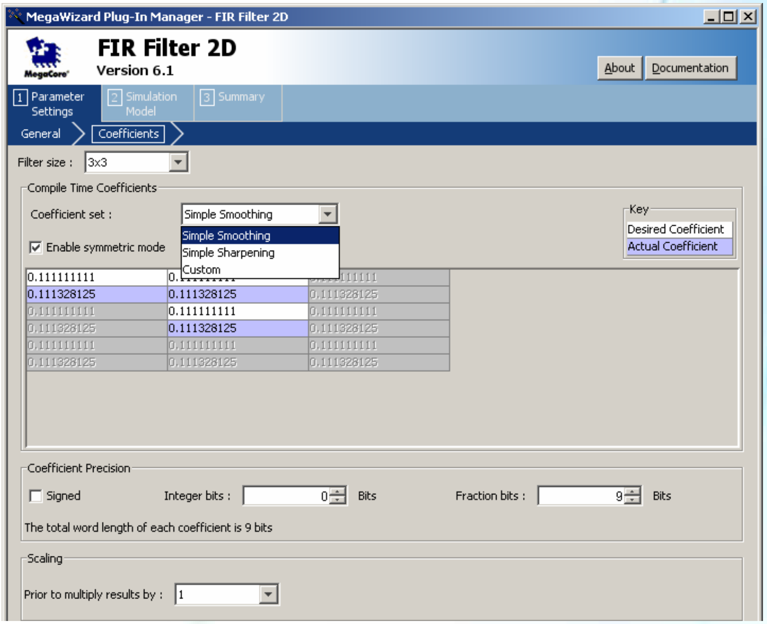
- *Noise* gets introduced into video data set via any electrical system used for storage, transmission, and/or processing
- Median filtering is a simple and very effective noise removal filtering process
- Median filtering:
  - Each pixel is determined by the median value of all pixels in a selected neighborhood (mask, template, window)
  - The median value m of a population (set of pixels in a neighborhood) is that value in which half of the population has smaller values than m, and the other half has larger values than m

# 2D Filtering

- 2D finite impulse response (FIR) filter and 2D median filter
  - 3x3, 5x5, or 7x7 filter sizes

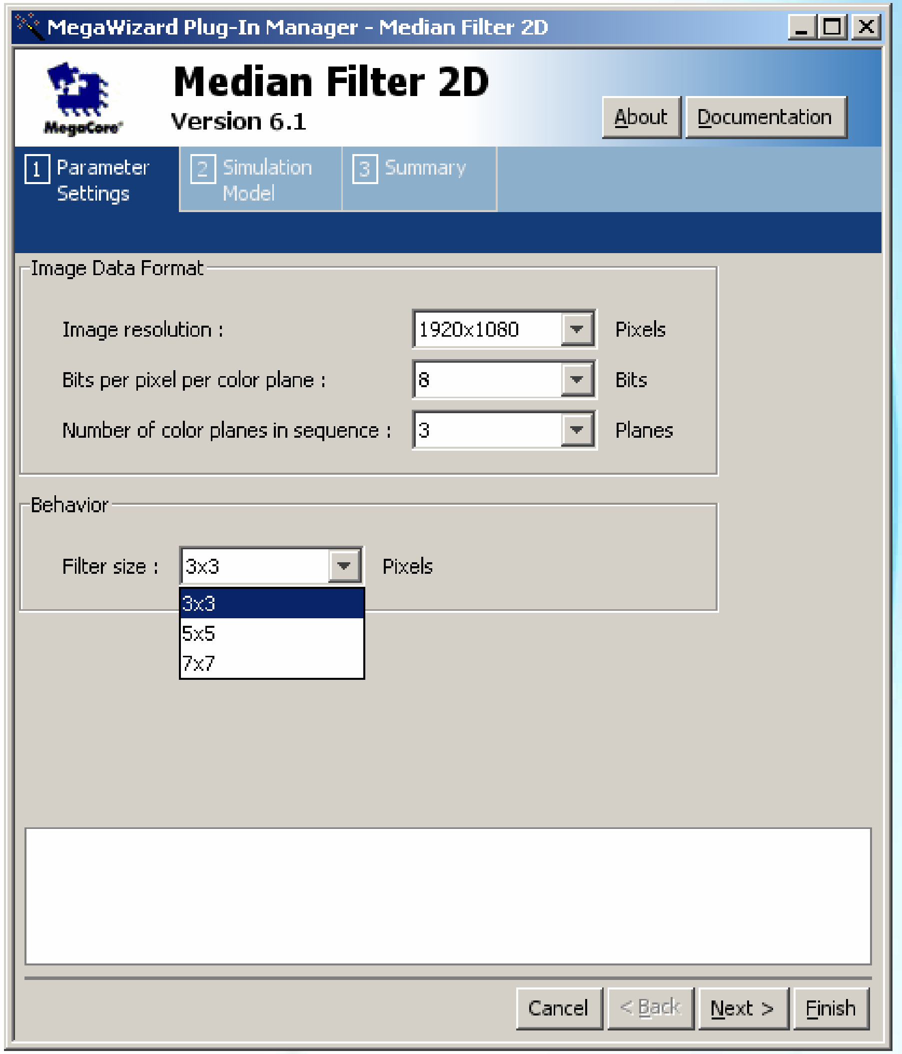



- Useful for noise reduction, smoothing, and edge enhancement



# 2D Median Filter IP

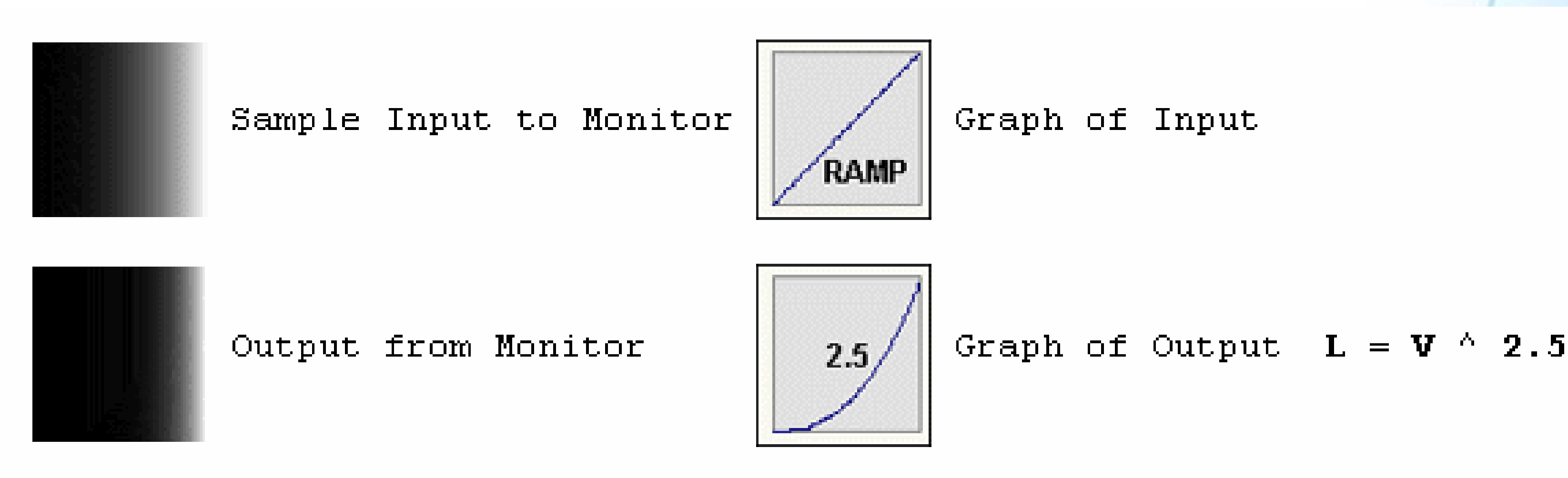
- The 2D Median Filter MegaCore® function provides a means to perform 2D median filtering operations using matrices of 3×3, 5×5, or 7×7 kernels
- Each output pixel is the median of the input pixels found in a 3x3, 5x5, or 7×7 kernel centered on the corresponding input pixel
- Where this kernel runs over the edge of the input image, zeros are filled in



# Gamma Correction: Basics

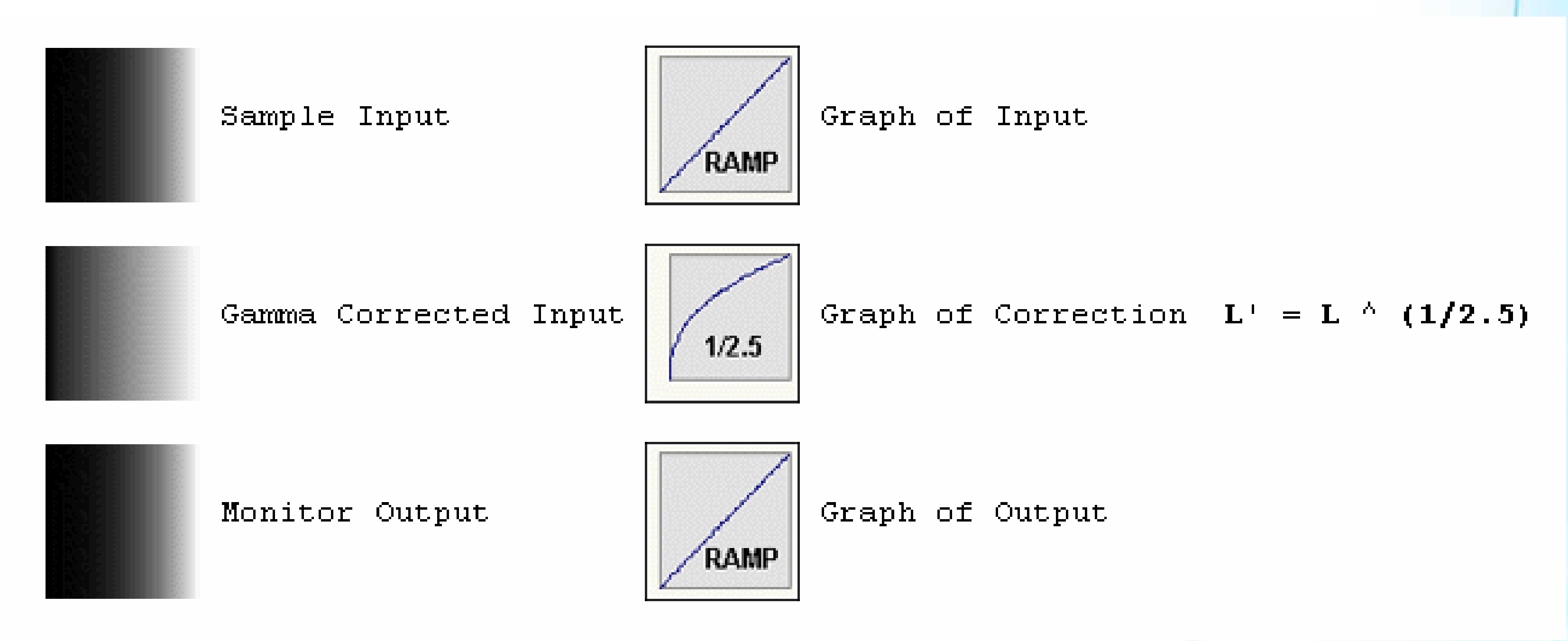
- There is a nonlinear relationship between pixel value and its displayed intensity on a monitor
- This nonlinear relationship is roughly a power function

$$displayed\_intensity\ (L) = pixel\_value\ (V)^{\wedge gamma}$$



# Gamma Correction: Basics

- To correct this annoying little problem, the input signal to the monitor (the voltage) must be "gamma corrected"



# Gamma Correction: Basics

Gamma corrected

# Conclusion

# VIP Basics – Summary

| Core | Function |
|---|---|
| Color space converter | Converts image data between a variety of different color spaces |
| Chroma resampler | Changes the sampling rate of the chroma data for image frames |
| Scalar | Resizes and clips image frames |
| Deinterlacer | Converts interlaced video formats to progressive video format |
| Alpha blending mixer | Mixes and blends multiple image streams, including PIP |
| 2D filter | Implements a 3x3, 5x5, or 7x7 FIR filter on an image data stream to smooth or sharpen images |
| Gamma corrector | Performs gamma correction on a color space |

# DSP Total Solutions

| | | | | |
|---|---|---|---|---|
| **System integration** | **SOPC Builder, VHDL/Verilog** | | | |
| **Design flow** | **VHDL/Verilog** | **MATLAB/Simulink**<br>DSP Builder<br>Synplicity<br>(Synplify DSP) | **'C' to hardware**<br>Mentor<br>Celoxica | **Processor C-code**<br>Nios® II Integrated Development Environment |
| **IP** | **Transforms Filters** | **Video and image processing** | **External digital signal processing (DSP) interfaces** | **Nios II processor** |
| **Devices** | **Cyclone® III and Stratix® III FPGAs, HardCopy® structured ASICs** | | | |

# Summary

- Key trend of “video in FPGA”
  - SD transitions to HD
  - MPEG4-2 moves to MPEG4-10
- Video image processing technology consists of:
  - Color space conversion
  - Chroma sampling
  - Scaling
  - Deinterlacing
  - Image blending
  - Filtering
  - Gamma correction
- Altera provides total solution for video image processing technology