PLAY WITH PIXELS

Bitmap manipulation with Flash CS3

Koen De Weggheleire

Lecturer Flash Platform Solutions
at Technical University Kortrijk, Belgium

HOWEST – newmovieclip.com

Adobe

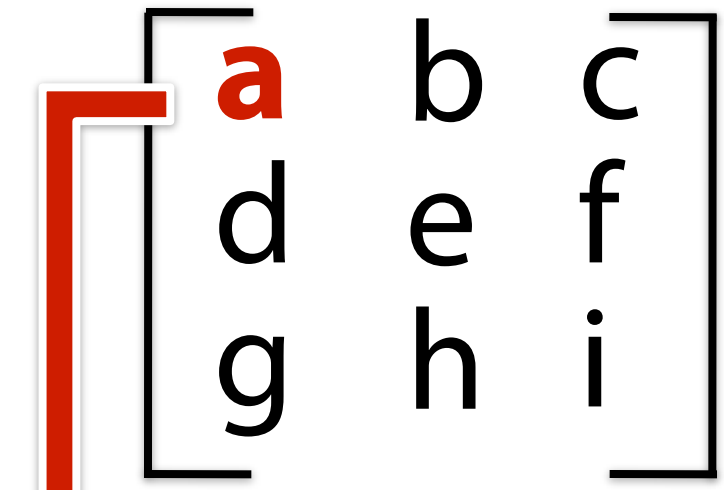
Session overview

- Already done:
  - Introduction
  - Talk about myself
- Coming soon:
  - Matrices and image manipulation
  - BitmapData class and filters
  - Demos
  - Q and hopefully A ☺

MATH

ooooh nooo !

Defining a MATRIX



- MATRIX
  = rectangular table of numbers !
- 3 x 3 matrix
  = three ROWS x three COLUMNS
- Dimensions of a matrix
  = number of rows by number of columns

- Each value stores a piece of data that can be used in a calculation !
  - flash.geom.Matrix
  - flash.filters.ColorMatrixFilter
  - flash.filters.ConvolutionFilter
- **Representation in Flash : [a , b , c , d , e , f , g , h , i]**

Defining a MATRIX

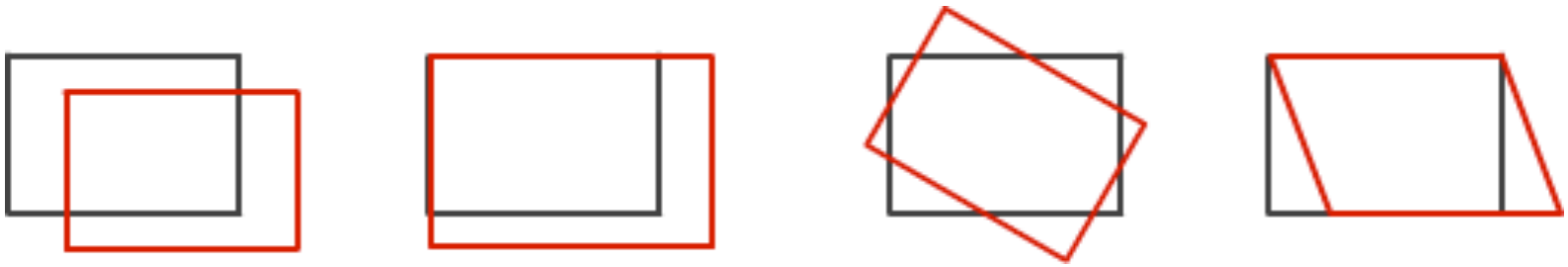
- **Useful TIP :** Flash allows arrays to be declared over multiple lines:
  - [a , b , c
    d , e , f
    g , h , i]

- **Identity Matrix Concept :** Matrix that **causes no effect** when it is concatenated with another matrix of the same dimensions (MATH) (=null transformation)
  - HOW IS AN IDENTITY MATRIX REPRESENTED ?

$$\begin{bmatrix} \mathbf{1} & 0 & 0 \\ 0 & \mathbf{1} & 0 \\ 0 & 0 & \mathbf{1} \end{bmatrix} = [\mathbf{1}, 0, 0, 0, \mathbf{1}, 0, 0, 0, \mathbf{1}]$$

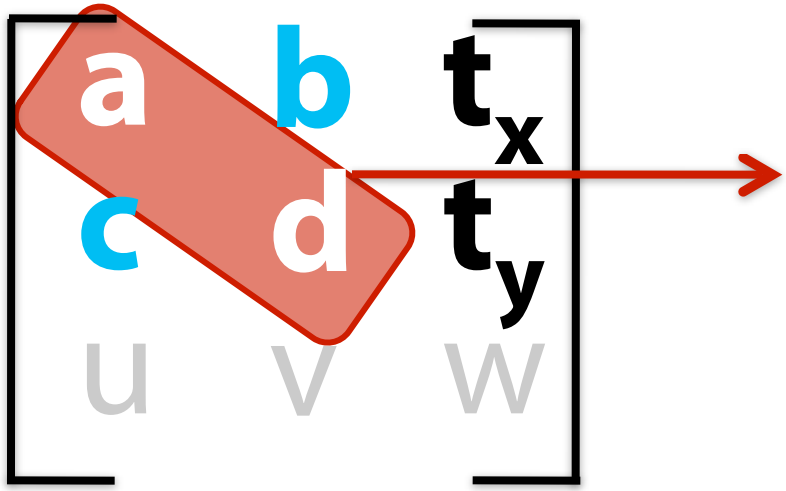
Matrix Transformations

- **HOW IT WORKS IN FLASH : 3 steps !**
  - Setting the properties of a **Matrix Object**
  - Apply the Matrix Object to the **matrix property** of a **Transform Object**
  - Apply that Transform object as the **transform property** of the **Display Object**
- **What you can do : translation, rotation, scaling and skewing**



Matrix Transformations

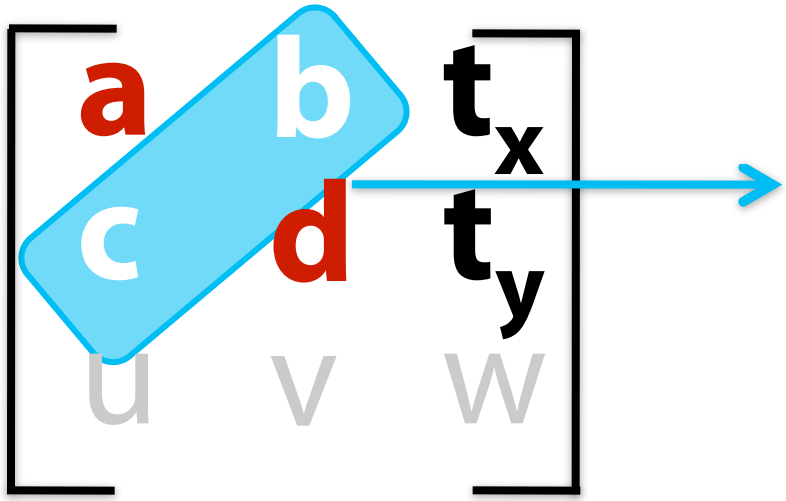
- **Transformation matrix**



X and Y scale values
1 = 100% scale

Matrix Transformations

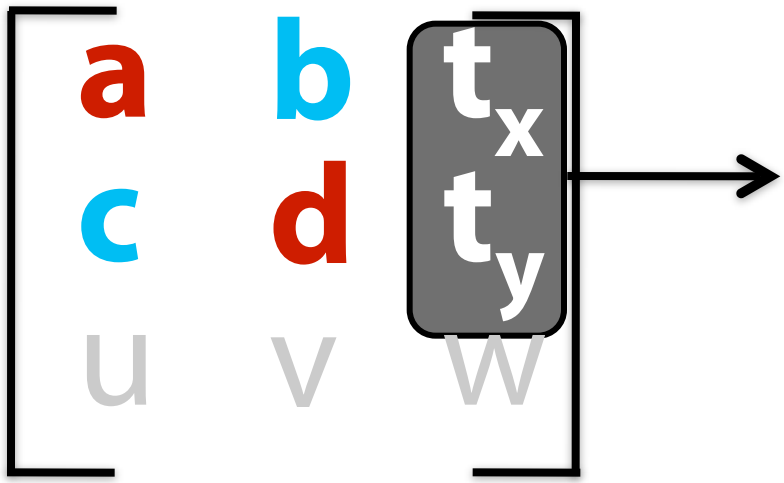
- **Transformation matrix**



Y and X skew values
0 = no skew
1 = skew value equal to the width or height of an object at a scale of 1

Matrix Transformations

- **Transformation matrix**



Y and X translation values
value in pixels

Matrix Transformations

- **AND ROTATION ? ??**

$$\begin{bmatrix} a & b & t_x \\ c & d & t_y \\ u & v & w \end{bmatrix}$$

Rotation = combination of a, b, c and d !

Example :

rotate around angle q

$$a = \cos(q)$$
$$b = \sin(q)$$
$$c = -\sin(q)$$
$$d = -\cos(q)$$

Matrix Transformations

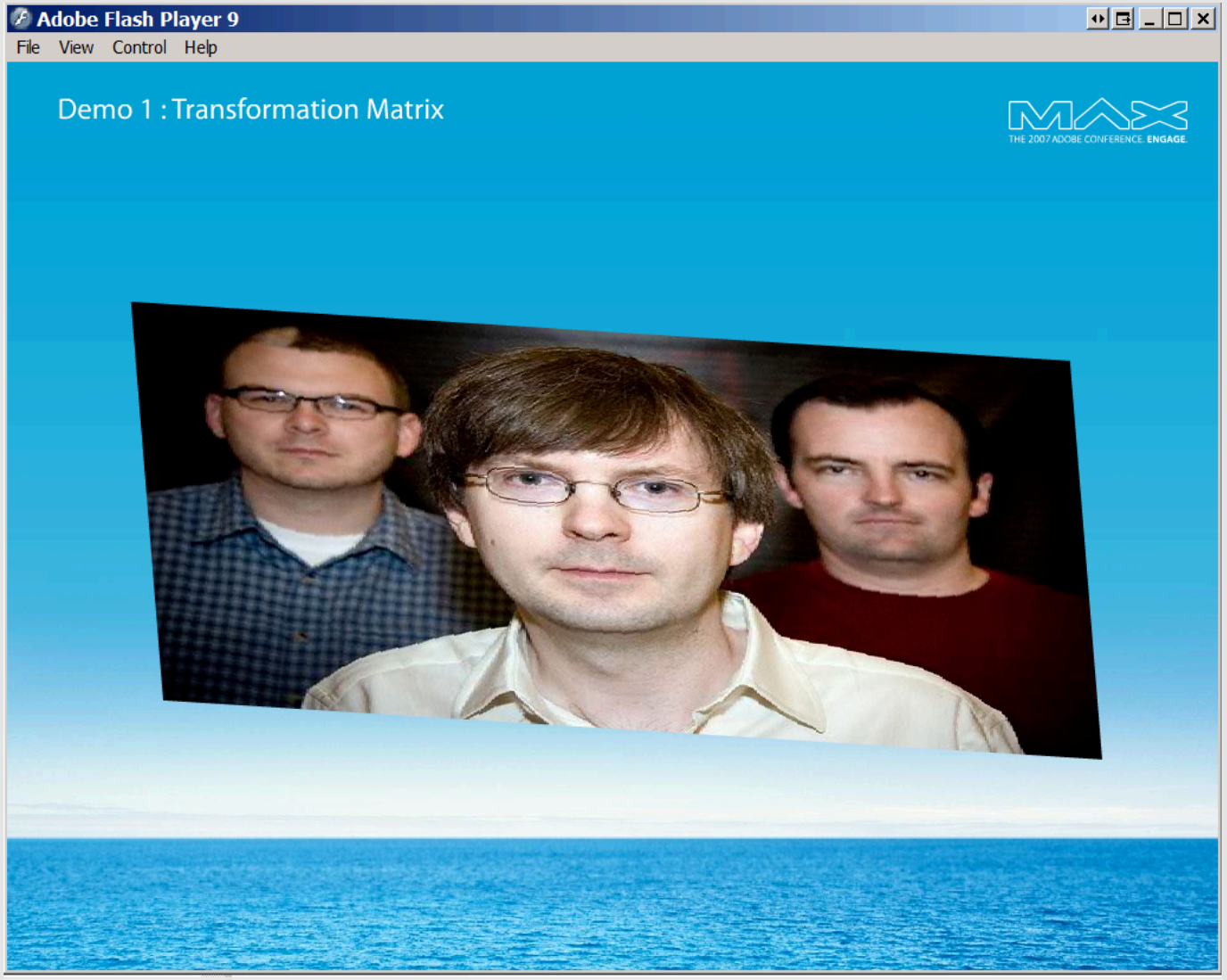
- Matrix class makes it easier:
  - Matrix.**createBox**(scaleX, scaleY, rotation, tx, ty);
  - Matrix.rotate(rotation);
  - Matrix.scale(scaleX, scaleY);
  - Matrix.translate(tx,ty)

Matrix Transformations

“We want to see this in action !”

time for demo 1

Demo 1 : Transformation Matrix

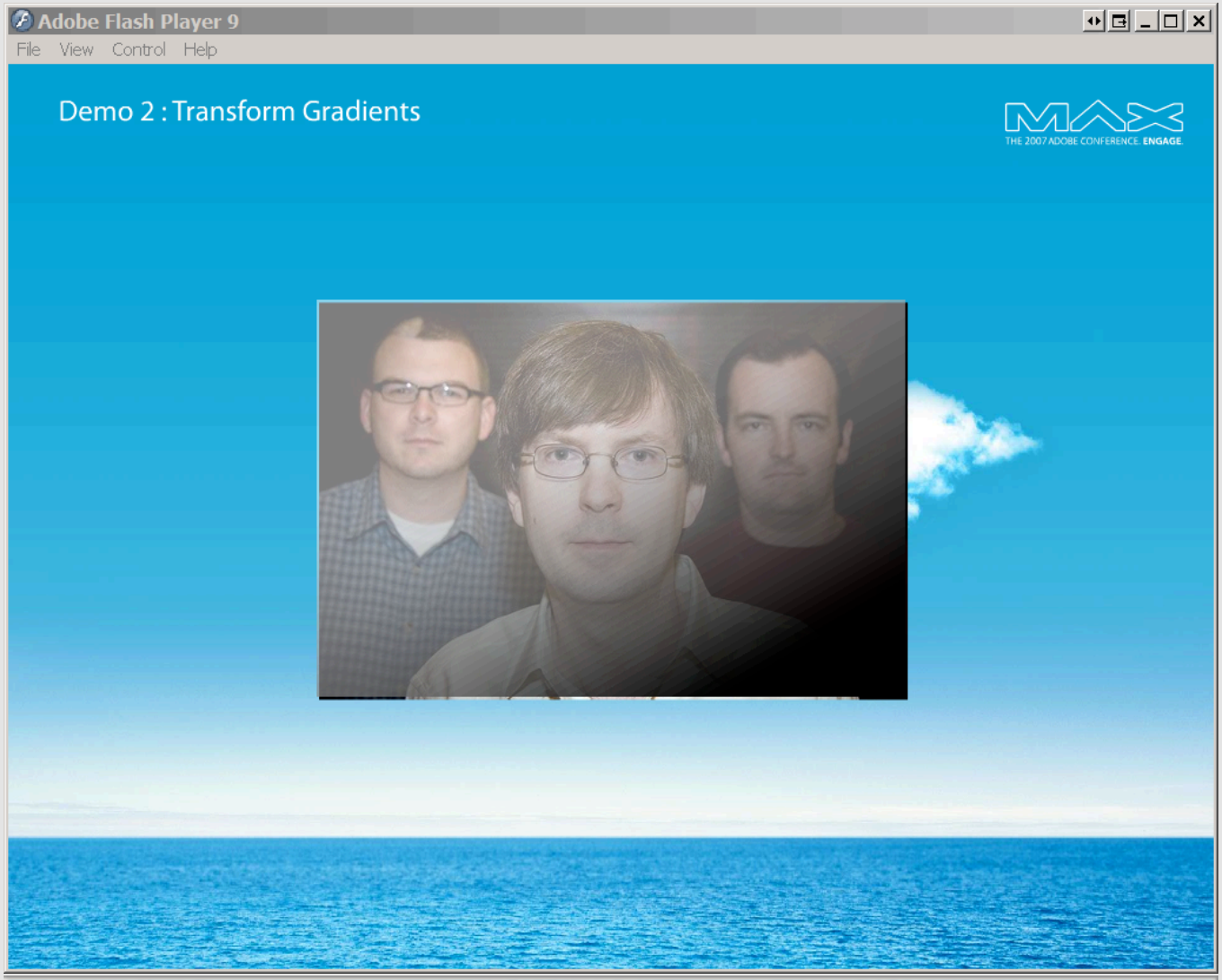


Matrix Transformations

- Transform Gradients
  - Transformations affected by the **Matrix.createGradientBox** method
  - First create gradient in Actionscript (Graphics.beginGradientFill) and use Matrix to define the gradient's translation, rotation, scale and rotation

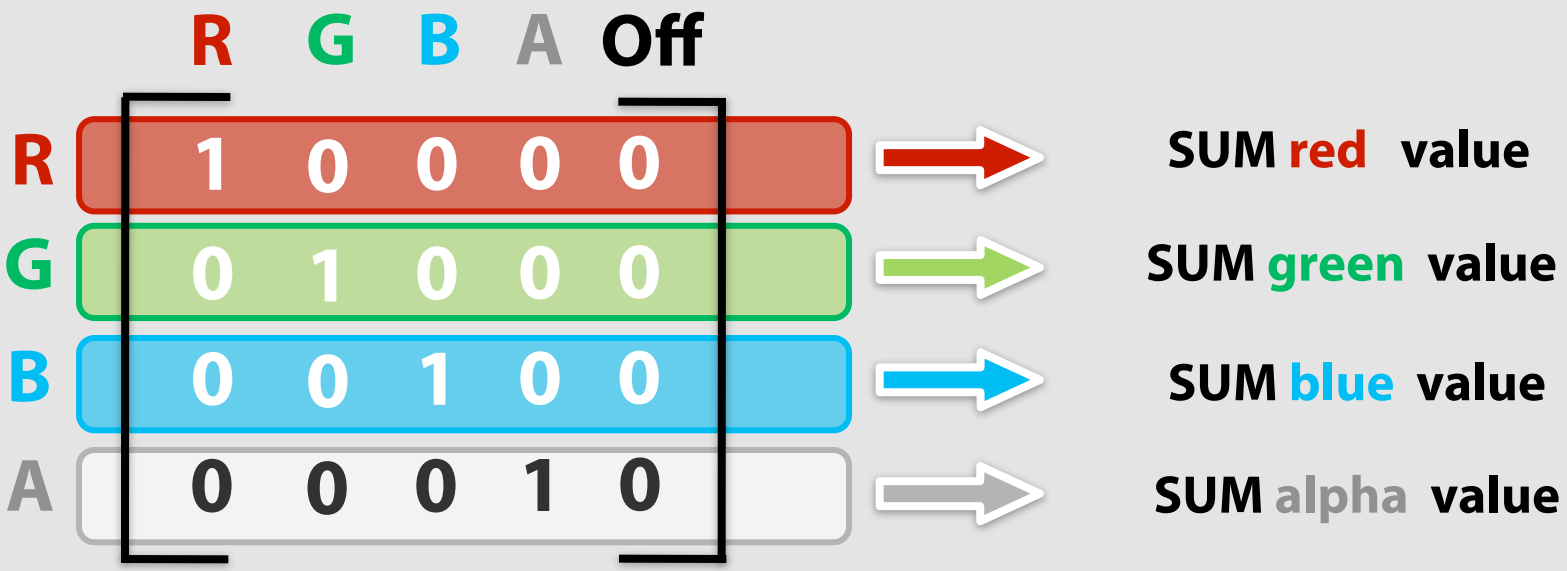
…time for demo 2

Demo 2 : Transform Gradients



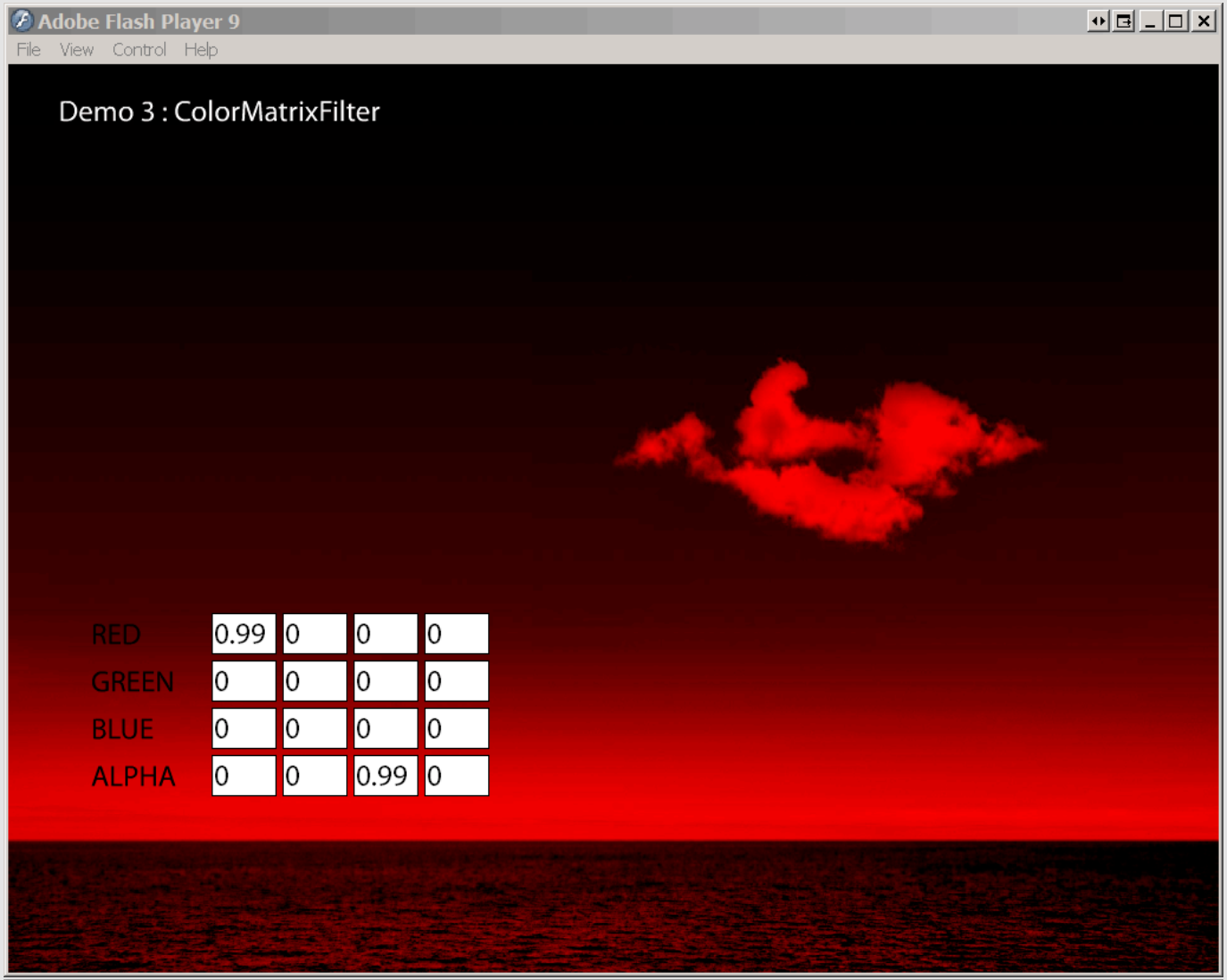
ColorMatrix

- Using Matrices to manipulate colors
  - Flash.filters.ColorMatrixFilter !!
  - ColorMatrixFilter accepts a 4x5 matrix



Demo 3 : ColorMatrix Filter

@



Convolution Filter

- = **Combines** PIXEL data in a bitmap with data from neighboring pixels (cfr moving mask with weight)

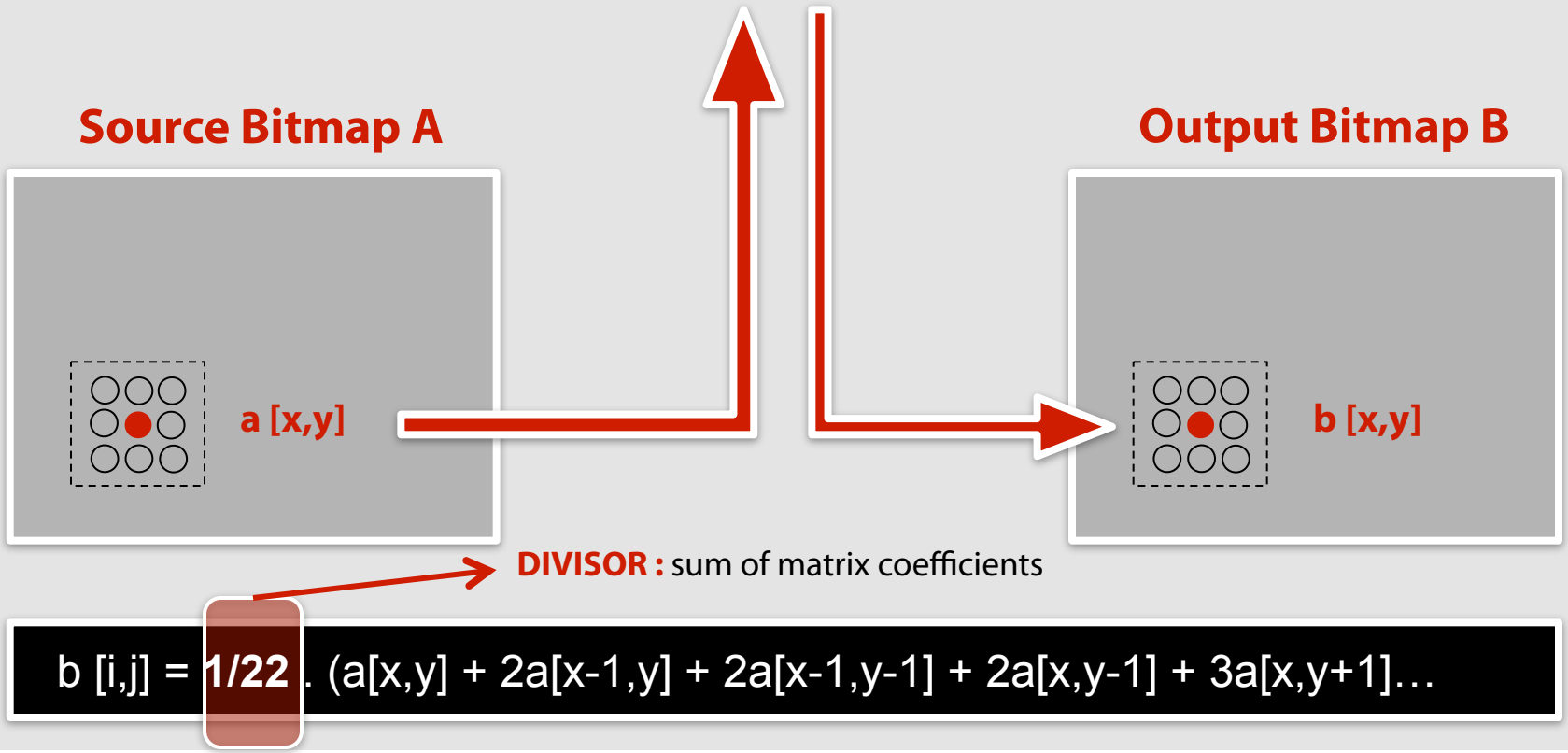
- = **FULL CONTROL on Pixel level** !
  - Blurring, Beveling, Embossing, Sharpening,…
  - ALL DONE BY CONVOLUTION MATRICES !

- ConvolutionFilter's matrix property does not have a set number of rows and columns (depends on the effect)

Convolution Filter

Convolution Matrix :

$$\begin{bmatrix} 2 & 2 & 3 \\ 2 & 1 & 3 \\ 3 & 3 & 3 \end{bmatrix}$$



A few common filters : line detection

- Horizontal edges :

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

- Vertical edges :

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

- Left diagonal edges :

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

- Right diagonal edges :

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

A few common filters : smoothing a blurring

- Basic Smooth 3x3 :

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

- Basic Smooth 5x5 :

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 4 & 4 & 4 & 1 \\ 1 & 4 & 12 & 4 & 1 \\ 1 & 4 & 4 & 4 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

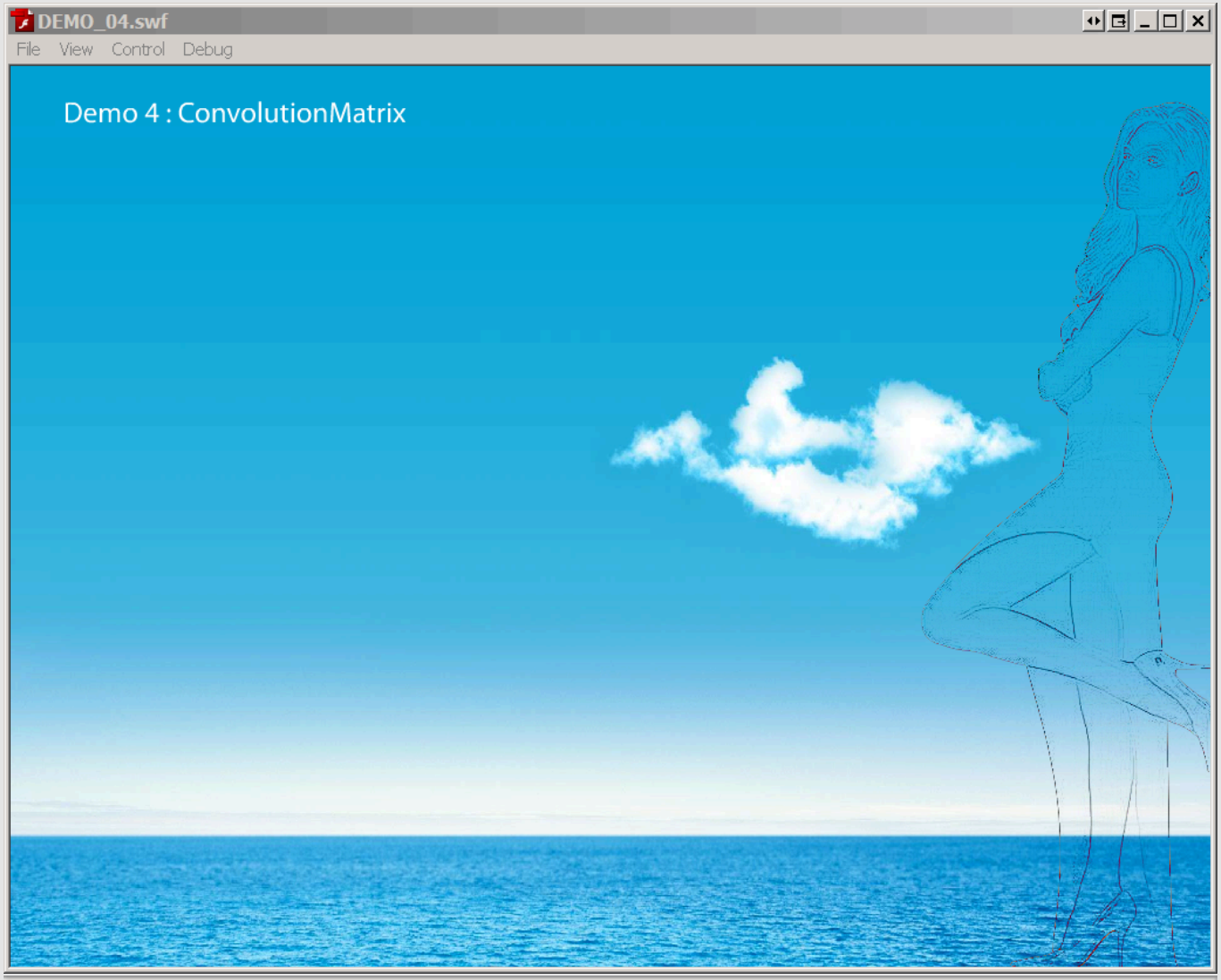
- Basic High Pass filter 3x3 :

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

- Basic High Pass filter 5x5 :

$$\begin{bmatrix} 0 & -1 & -1 & -1 & 0 \\ -1 & 2 & -4 & 2 & -1 \\ -1 & -4 & 13 & -4 & -1 \\ -1 & 2 & -4 & 2 & -1 \\ 0 & -1 & -1 & -1 & 0 \end{bmatrix}$$

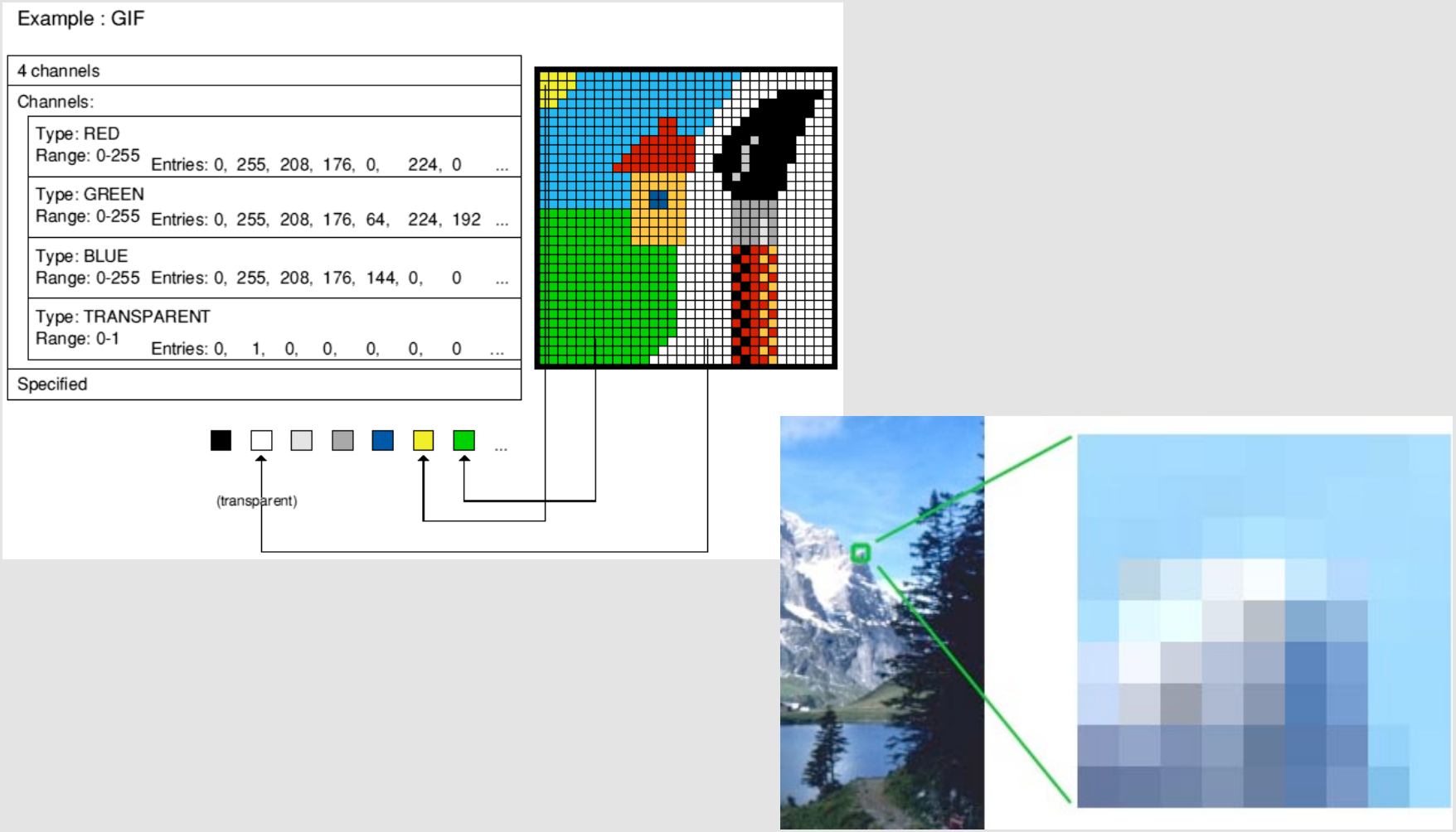
Demo 4 : ooooh… want to see this in action !



BitmapData class

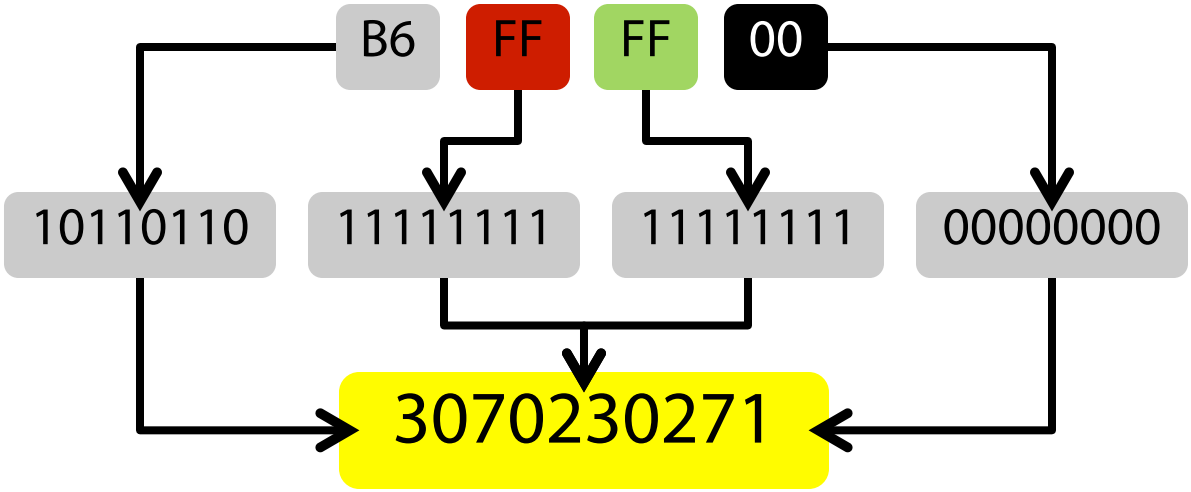
- Create and manipulate 32-bit images at runtime using Actionscript
- Access the BitmapData for a Bitmap image that you load with the flash.display.Loader class
- Import flash.display.BitmapData
- BITMAP: digital image format that describes an image using a grid of color values
  - A cel = a pixel with specific color value
  - Color = binary number (32 bit)
    - Color Information = 4 channels (Red, Green, Blue and Alpha)

BitmapData class



BitmapData class

- Hexadecimal color : 32 bit color values (<>24 bit)
  - = 4 pairs of hexadecimal digits : **ARGB** format
    - Each pair defines the intensity of each of the color channels (Red, Green, Blue and Alpha)
    - Intensity : range 0-255 (FF = full intensity; 00 is no intensity)



BitmapData class

- Instantiation
  - var myBMPData:BitmapData = new BitmapData(width, height,transparent, fillColor);
  - Default : white image with no transparency
  - Limitations: 2880 pixels width and height (=32 MB RAM)
  - TIP: Free the memory using myBMPData.dispose() when possible
- How to show the bitmapData instance on Stage?
  - Assign it the bitmapData property of an existing Bitmap instance.
  - Pass it as parameter when initiating a new Bitmap object.

BitmapData class

- Important method from bitmapData class:
  - BitmapData.applyFilter
    (source:BitmapData,rect:Rectangle,destPoint:Point,filter:BitmapFilter)
  - Example with displacementmapfilter!

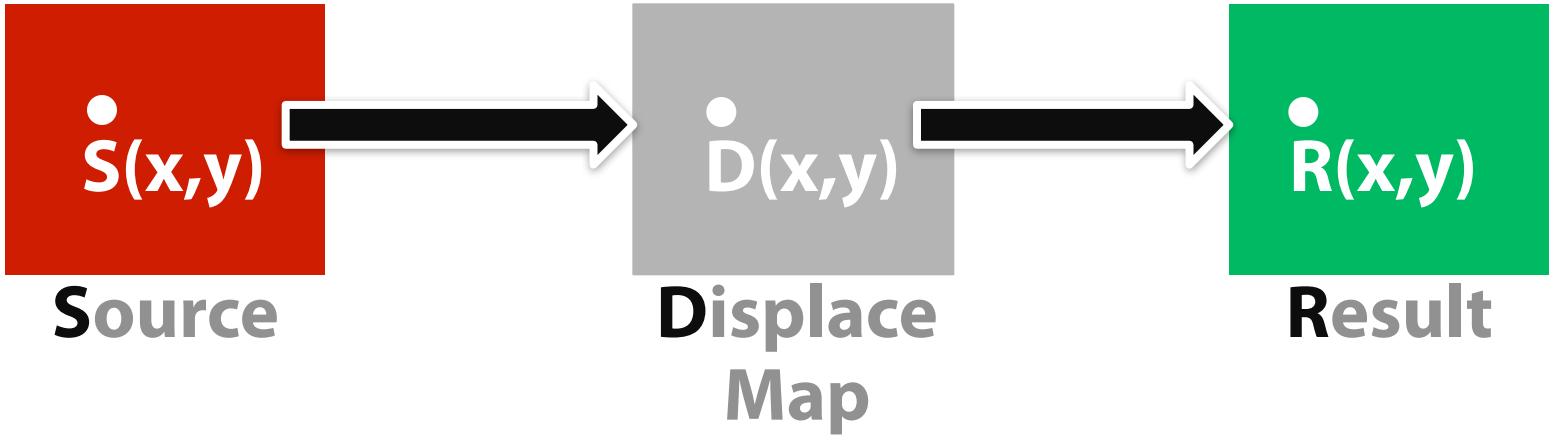
BitmapData class

- **DisplacementMapFilter** : uses **pixel color values** from a displacement map to do displacement in x and y direction !
  - Displacement map = BitmapData object !
  - Works also directly on all DisplayObjects !

Understanding the DisplacementMapFilter

- New DisplacementMapFilter(**mapBitmap**:BitmapData, **mapPoint**:Point, **componentX**:uint, **componentY**:uint, **scaleX**:Number, **scaleY**:Number, **mode**:String, color:uint, **alpha**:Number)
  - componentX and componentY are a **bitmapDataChannel** constant :
    - **Constants : ALPHA = 8 ; BLUE = 4 ; GREEN =2; RED =1**
- **Next slide rocks… ☺ !**

Understanding the DisplacementMapFilter



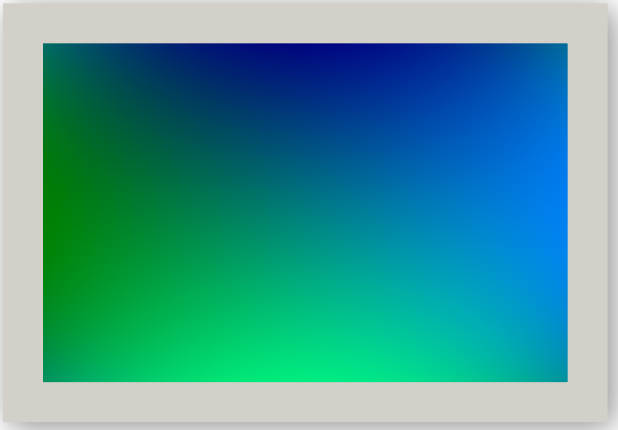
$$R[x, y] = S\Big[x + \big((doCompX(x, y) - 128) * scaleX\big) / 256,\ y + \big((doCompY(x, y) - 128) * scaleY\big) / 256\Big]$$

Multipliers to SCALE the X- and Y displacement

doCompX = get componentX value from D(Sx-Dx , Sy-Dy)
doCompY = get componentY value from D(Sx-Dx , Sy-Dy)

Understanding the DisplacementMapFilter

- Example : displ. map for pinch effect:



- Example: displ. map for perspective

Time for some displacement demos

Original image

Other interesting methods of BitmapData class

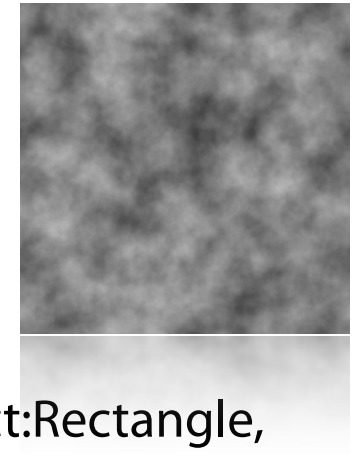
- **colorTransform**(rect:Rectangle, colorTransform:ColorTransform):void
  - Adjusts the color values in a specified area of a bitmap image by using a ColorTransform object.
- **compare**(otherBitmapData:BitmapData):Object
  - Compares two BitmapData objects.
- **copyChannel**(sourceBitmapData:BitmapData, sourceRect:Rectangle, destPoint:Point, sourceChannel:uint, destChannel:uint):void
  - Transfers data from one channel of another BitmapData object or the current BitmapData object into a channel of the current BitmapData object.

Other interesting methods of BitmapData class

- **copyPixels(**sourceBitmapData:BitmapData, sourceRect:Rectangle, destPoint:Point, alphaBitmapData:BitmapData = null, alphaPoint:Point = null, mergeAlpha:Boolean = false):void
  - Provides a fast routine to perform pixel manipulation between images with no stretching, rotation, or color effects.
- **dispose()**:void
  - Frees memory that is used to store the BitmapData object.
- **draw**(source:IBitmapDrawable, matrix:Matrix = null, colorTransform:ColorTransform = null, blendMode:String = null, clipRect:Rectangle = null, smoothing:Boolean = false):void
  - Draws the source display object onto the bitmap image, using the Flash Player vector renderer.

Other interesting methods of BitmapData class

- **perlinNoise**(baseX:Number, baseY:Number, numOctaves:uint, randomSeed:int, stitch:Boolean, fractalNoise:Boolean, channelOptions:uint = 7, grayScale:Boolean = false, offsets:Array = null):void
  - Generates a Perlin noise image.



- **pixelDissolve(**sourceBitmapData:BitmapData, sourceRect:Rectangle, destPoint:Point, randomSeed:int = 0, numPixels:int = 0, fillColor:uint = 0):int
  - Performs a pixel dissolve either from a source image to a destination image or by using the same image.

Other interesting methods of BitmapData class

- **threshold**(sourceBitmapData:BitmapData, sourceRect:Rectangle, destPoint:Point, operation:String, threshold:uint, color:uint = 0, mask:uint = 0xFFFFFFFF, copySource:Boolean = false):uint
  - Tests pixel values in an image against a specified threshold and sets pixels that pass the test to new color values.

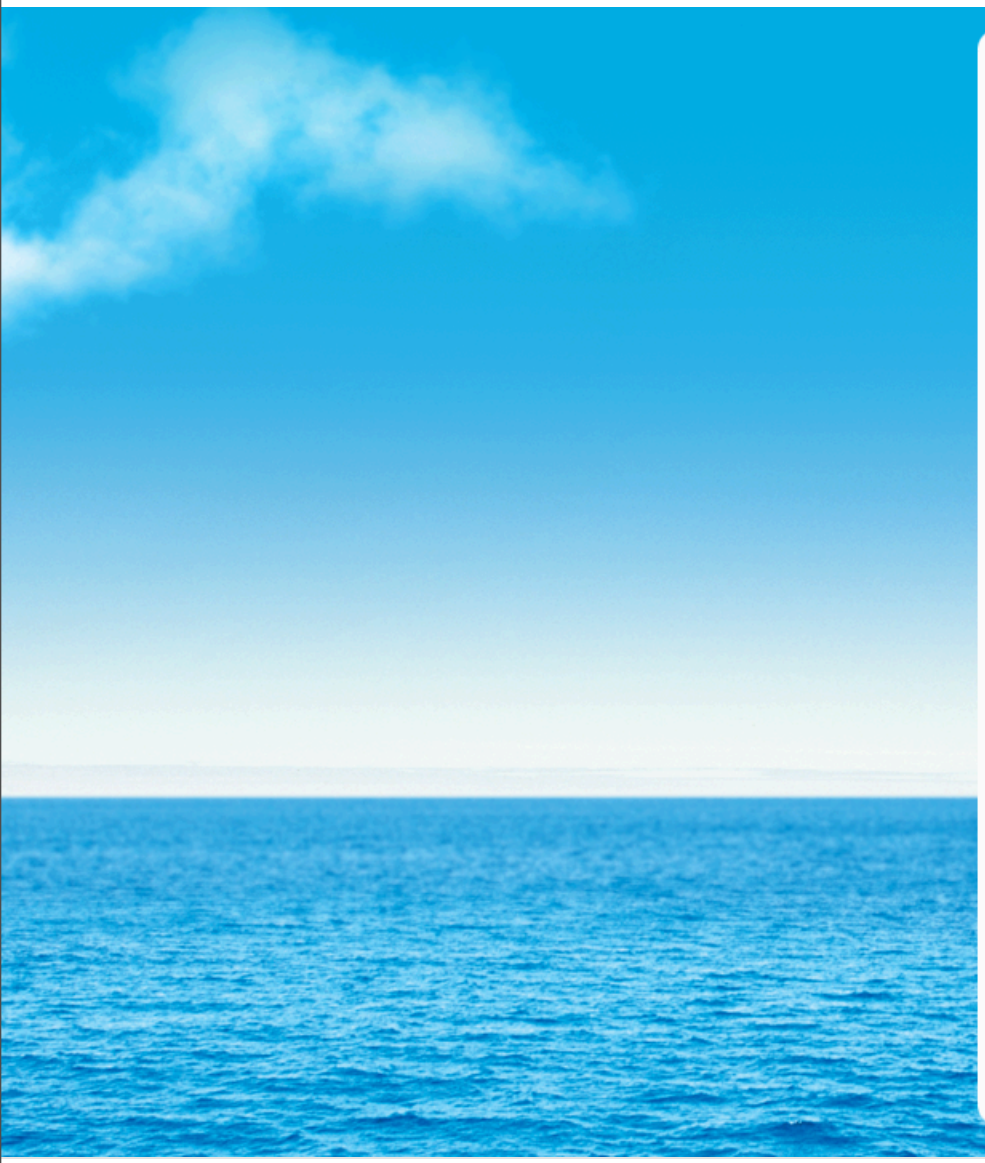
- **and so much more….**

Combining it all = FUN !!

- Webcam combination (games) possible: basketball, ping pong, pictionary, …

EXAMPLES…

THANKS YOU ALL !!



- **OOOOH THE END ??**
  - **START EXPERIMENTING NOW !!**
- **CONTACT :**
  - **info@newmovieclip.com**
- **DEMO FILES :**
  - Available at:
  - **www.newmovieclip.com**

Q ??