UE Videoverarbeitung Winter Term 2011

Exercise 2

1) General Task

The following filters have to be implemented for the second exercise:

1) **effect_add_text.m**
2) **effect_irising_in_out.m**
3) **effect_soft_focus.m**
4) **effect_fast_motion.m**
5) **effect_scene_cut.m**

The filter functions have to be implemented in the above mentioned files. All filters have a different amount of parameters which control the properties of the filters. You can expand the number of parameters for every filter but you should be able to justify each new parameter. During the implementation of each filter meaningful parameter values that result in authentic looking results have to be found.

All the files which are needed for implementing each filter (filter_*.m) are in the /src directory. By calling the function exercise2.m all implemented filters are applied on an image sequence.

Example 1: **exercise2('../images/', '../output/');**
applies all filters on the frames frameXXXX.png in the directory '../images' and stores the resulting images in the directory '../output'.

Example 2: **exercise2('../images/', '../output/', start_frame, end_frame);**
processes only frames 'start_frame' to 'end_frame'.

Example 3: **exercise2({'../images1/','../images2/'}, '../output/');**
reads all images from 2 input directories '../images1' and '../images2' (in this order) and applies the filters on all images.

Example 4: **help exercise2**
displays help text for each file in the command window (in this example for the file exercise2).

2) Submission

The deadline for exercise1 is **28.11.2011**

The submission must include:

- Commented matlab source files of all filters.
- At the beginning of each filter file following fields have to be completed:
  - Short description of the implementation.
  - Short summary of the physical background/reason of the effect.
    For example: What causes Sepia discoloration?
  - Meaningful parameter values.

All files have to be uploaded as a .zip file (UE_GROUPx_EXERCISEy.zip) on TUWEL. Only one submission is needed per group.

3) General information on the implementation of the filters

All exercises are handed out with a Matlab framework. In this exercise the framework consists of the file exercise2.m. The framework takes automatically care of reading, saving and displaying the images. To each filter in this exercise the struct **video** will be passed as the first argument. It has the following structure:

video.original	→ original image in RGB

video.filtered → filtered image in RGB
video.original_frame_nr → frame number of this frame in the input directory (is not modified by any buffer!)
video.frame_nr → frame number of this frame (can change if frames are inserted/removed by a filter)

When implementing a filter the following has to be taken into account:

1) Each filter works on the field **video.filtered** and also saves the result there again. In this way all filters can work sequentially on the same image.

2) To provide access to previous frames a frame buffer is installed. Thus two previous frames can be addressed. You can access the current frame with **video.frame(1).filtered**, the previous one with **video.frame(2).filtered**.

   video.frame(1).filtered → current frame i
   video.frame(2).filtered → frame i-1
   video.frame(3).filtered → frame i-2

3) You can perform any conversion of the input information. But EVERY filter MUST save its results in the RGB format (RGB values between 0 and 255). Thus no distinction and multiple implementation for different image and color formats is needed and the filters can be executed in an arbitrary order.

Example:

```
function filter_test(video)
    img = video.frame(1).filtered;
    …
    …
    …
    video.frame(1).filtered = img;
end
```

In order to pass informations from one filter to another the struct video will be expanded in the course of this lecture. Temporary filters, which operate on multiple images require knowledge about the current status of the filtering (What has been done so far? What has to be done in this step?).

HINT 1:
You can insert any additional fields in the struct video if you need them for the proper working of the filters.

HINT 2:
Matlab manages the images transposed, which means that the image point (x,y) corresponds to the value (y,x) in the image structure. Each image in video.filtered and video.original has the format [H x W x C], where H represents the image height, W the image width and C the color depth. For the RGB color space C = 3 (R/B/G). Here are some examples to access the color information of the images:

Example 1:
r_info = video.filtered(4,3,1); → x = 3, y = 4, c = 1
r_info = 34 → red color information at the position (3,4)

Example 2:
rgb_info = video.filtered(4,3,1:3) → x = 3, y = 3, c = 1:3
rgb_info = [34 12 120] → RGB values at the position (3,4)

Example 3:
r_info = video.filtered(1:10,1:10,1)→ x = 1:10, y = 1:10, c = 1
r_info = 10 x 10 matrix → red color information at the positions (x = 1:10, y = 1:10)

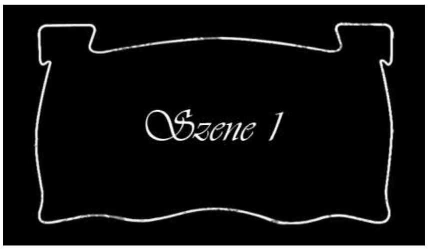
HINT 3:

Try all the filters in an arbitrary order to verify correct functioning.

Effect 1: effect_add_text

General information

As old movies didn't support speech, showing text was a common used technique to overcome this gap. In this task inserting text descriptions to the original sequence has to be realized.



(Frames 1..X)

(Frames X+1,…)

Implementation

For this example the following functions must be implemented:

Function:	effect_add_text
Input:	<video> struct, <text> cell with the effect parameters
Output:	<video> struct with modified list of input files (video.input_files)
File:	effect_add_text.m

1) Read the filenames, positions and durations of the text effects.
2) Copy the filenames of the text frames to the desired positions in video.input_files
3) Take care that frames are shifted correctly and don't get lost.

The effect gets called before the processing of the frames and inserts the appropriate images in the list video.input_files. The variable "text" contains an array with the filenames of the images, the temporal position (relative to the original order of the input frames) of the text frames and the duration in frames.

text{1..N}{1} = filename of the text image
text{1..N}{2} = number of the original input frame before which the text should be added
text{1..N}{3} = duration of the text scene

It is possible to define several positions where text frames should be added (1...N).

Example:

```
effect_add_text(video, {{'../text/scene_text1.png', 1,3},{'../text/scene_text2.png',3,3}, {'../text/scene_text3'
,png,30,3}};
```

This call adds the image 'scene_text1.png' three times before frame 1 in the list video.input_files. Furthermore it adds the image 'scene_text2.png' three times before frame 3 and the image 'scene_text3.png' three times before frame 30.

ATTENTION: The position of the text frames always refers to the original images in the list video.input_files. If the original frame with the number text{i}{1} is not found in the list, the effect gets not applied on this point.

Original order: I1, I2, I3, I4, I5, I6

After calling the above mentioned command the sequence changes to the following one:

T1, T1, T1, I1, I2, I3, **T2, T2, T2**, I4, I5, I6

Ix is the i-th input image and Tx is the t-th text image. Since frame 30 is not present in the original list the application of this effect gets omitted at this place.

Effect 2: effect_fast_motion

General information

By cutting away some frames non-relevant and tedious actions were shorten in old movies. This effect is also used in modern films and is called time-lapse.

Implementation

For this example the following functions must be implemented:

Function:	effect_fast_motion
Input:	<video> struct, <drop> cell with the effect parameters
Output:	<video> struct with modified list of input files (video.input_files)
File:	effect_fast_motion.m

1) Read the start frame and the old and new duration of the sequence.
2) Find the original images in the list video.input_files to which the effect should get applied.
3) Identify the images that should be deleted of the sequence.
4) Delete the images from the list video.input_files

This effect takes a certain area of the video (e.f. frames 1 to 15) and reduces the number of frames by randomly dropping some of the frames. Similar to the 'text' structure in effect_add_text the structure 'drop' contains the start frame and the number of frames to which the effect should be applied. Moreover the amount of frames to which the scene should get shorten gets specified:

drop{1..N}{1} = Start frame where the effect should be applied (refers to the initial number of frames in the input list video.input_files and does not consider inserted text frames by effect_add_text).
drop{1..N}{2} = Number of frames on which the effect should get applied.
drop{1..N}{3} = Number of frames to which the scene should get shorten.

Example: A sequence consists of 6 original frames and 6 frames of two text sequences (2 sequences, each with 3 pictures).

T1, T1, T1, I1, I2, I3, **T2, T2, T2**, I4, I5, I6

Calling the effect with effect_fast_motion(video, {{1,3,2}}) applies the effect (starting at frame 1) to 3 original frames (frames 1-3) and reduces the number to two by randomly dropping one frame. In this example frame I2 gets rejected:

T1, T1, T1, I1, I3, **T2, T2, T2**, I4, I5, I6

Useful MATLAB commands:

rand, sort, find

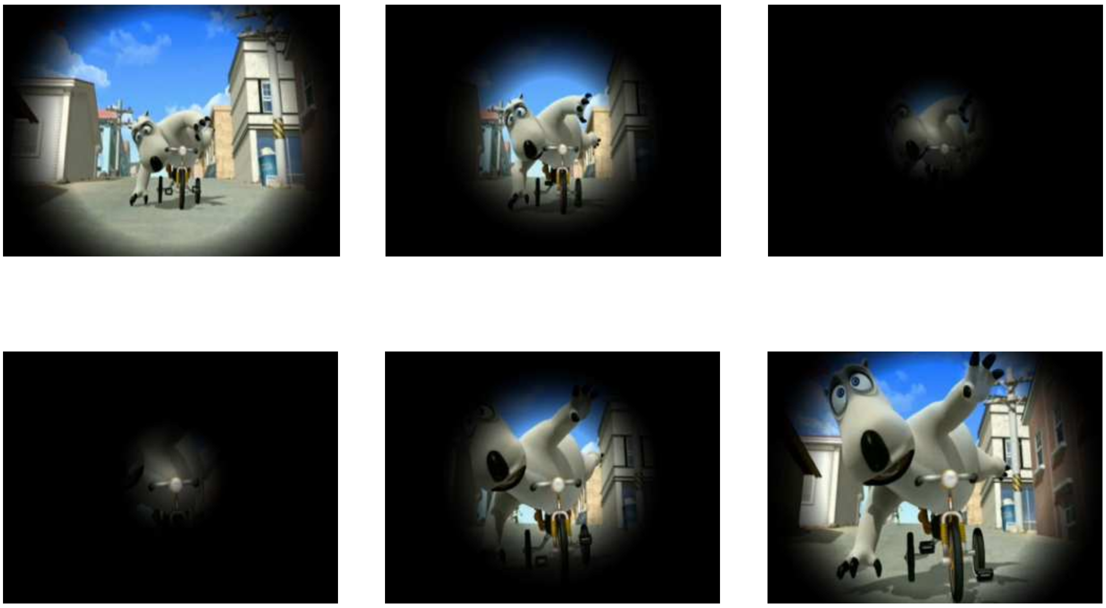
References:

http://en.wikipedia.org/wiki/Time-lapse

Effect 3: effect_irising_in_out

General information

To make the transition between scenes more exiting, the iris was closed at the end of a scene and then reopened. This effect should be replicated here.



Implementation

For this example the following functions must be implemented:

Function: effect_irising_in_out
Input: <video> struct, transition_size, min/max_size, dist_x/y, <fade> cell with the effect parameters
Output: <video> struct with modified frame(1).filtered
File: effect_irising_in_out.m

To implement this function the already existing function filter_iris should be used. Therefore effect_irising_in_out uses the same parameters. In addition, a structure <fade> is passed indicating at which positions in the sequence a fade_in/fade_out occurs and how long it lasts:

fades{1..N}{1} = The frame number where the fade starts. This effect works on all frames (also on the inserted text frames).
fades{1..N}{2} = Duration of the fade_in/fade_out process. During this period the iris should be closed completely and then reopened to the default value.

1) Determine the start and end frames of the effect.
2) Determine appropriate values for the parameters transition_size, min_size, max_size, dist_x, dist_y. When no fading in or fading out occurs the filter_iris should get called with these default settings.
3) In addition, the iris should get closed and reopened at the desired points in the video. If the effect was elected with 10 images, the effect should close the iris in about half the time (5 frames) and reopen in the remaining five frames back to the default value. The iris must be closed completely in the middle of the duration. You can make the transition linearly but also use other function (e.g. sinus). Calculate for each frame the size of the iris and call the filter filter_iris with the obtained values.

Useful MATLAB commands:

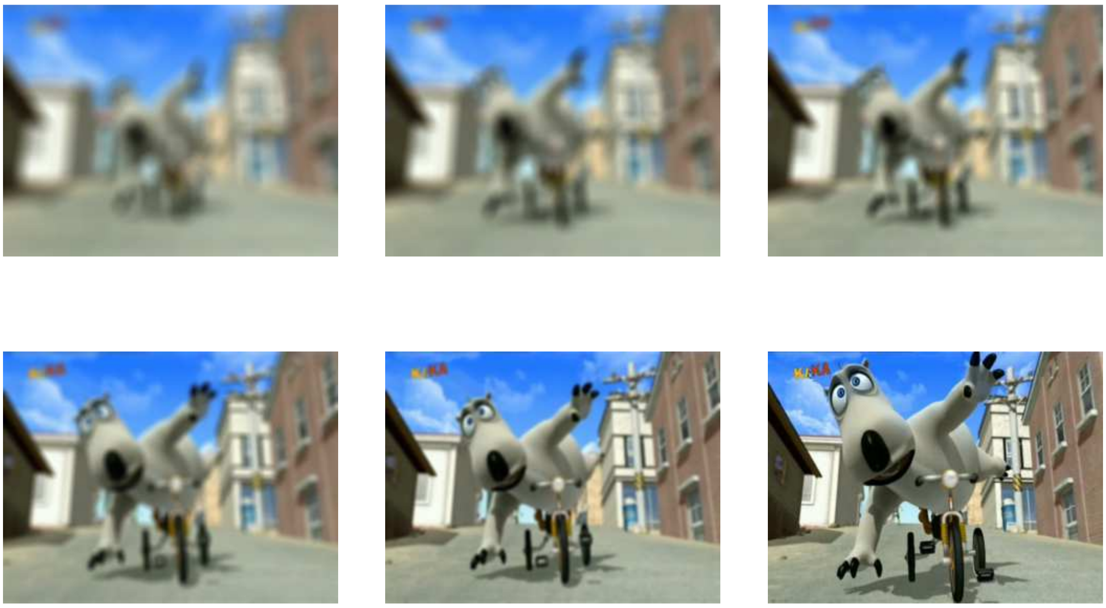
filter_iris

NOTE: The frame number at which the effect starts refers on the video.frame_nr of each image and thus also includes text frames.

Effect 4: effect_soft_focus

General information

This effect was used to draw attention to certain points in the film (f.e. in romantic moments when hero and heroine are facing for the first time in months the scene gets blurred for a moment).



Implementation

For this example the following functions must be implemented:

Function:	effect_soft_focus
Input:	<video> struct, blur factor, <focus> struct
Output:	<video> struct with modified frame(1).filtered
File:	effect_soft_focus.m

A value blur_factor is passed which represents the maximum degree of blurriness. The transition between the fully blurred image (start frame) and the unblurred image (end frame) should be fluent. The struct <focus> indicates at which positions the blur starts and how long it takes:

fades{1..N}{1} = The frame number where the effect starts (biggest blur factor).
fades{1..N}{2} = Duration of the blurring.

1) Determine start and end frames of the effect
2) Determine based on the distance to the start frame the blurriness of the image.
3) Pass the calculated blur factor to the filter_unsharp function or use the MATLAB functions fspecial and imfilter to blur the image accordingly.

Useful MATLAB commands:

filter_unsharp, fspecial, imfilter

Effect 5: effect_scene_cut **(OPTIONAL!!!)**

General information

A filter for the detection of scene changes should be implemented. It considers consecutive frames and recognizes new scenes.

Implementation

For this example the following functions must be implemented:

Function:	effect_scene_cut
Input:	<video> struct
Output:	<video> struct with modified video.frame(X).scene
File:	effect_scene_cut.m

Scene transitions can be identified by differences between adjacent frames (content, color, brightness). If the difference between two frames is sufficiently big, the second image can be considered as a new scene. This difference can be detected by an appropriate evaluation of one or more histograms. The Bhattacharya distance is often used to compare two histograms.

To be able to detect the scene changes it is important to work on the original images of the scene (**video.frame(X).original**). Since the images are only used to for the scene cut detection they are not altered by this effect.

ATTENTION: Any changes in the brightness of a scene may also be caused by changing lightning conditions. A robust scene detection system should be aware of this.

Useful MATLAB commands:

hist, rgb2gray

References:

http://en.wikipedia.org/wiki/Bhattacharyya_distance
http://en.wikipedia.org/wiki/Histogram