# 8.2 Deriving Video Semantics

## 8.2.1 Cut Detection

A very simple and very basic scheme for the investigation of video semantics is **cut detection**. A **cut** denotes the border between two **shots** in a movie. During a shot the camera films continuously.

One distinguishes **hard cuts** and **soft cuts** (fade out, fade in, resolve,....).

The most important use of cut detection is to decompose a video into smaller units which can then be analyzed. A sample application is the use of shots as the "atomic" units for storage and retrieval in a video archive.



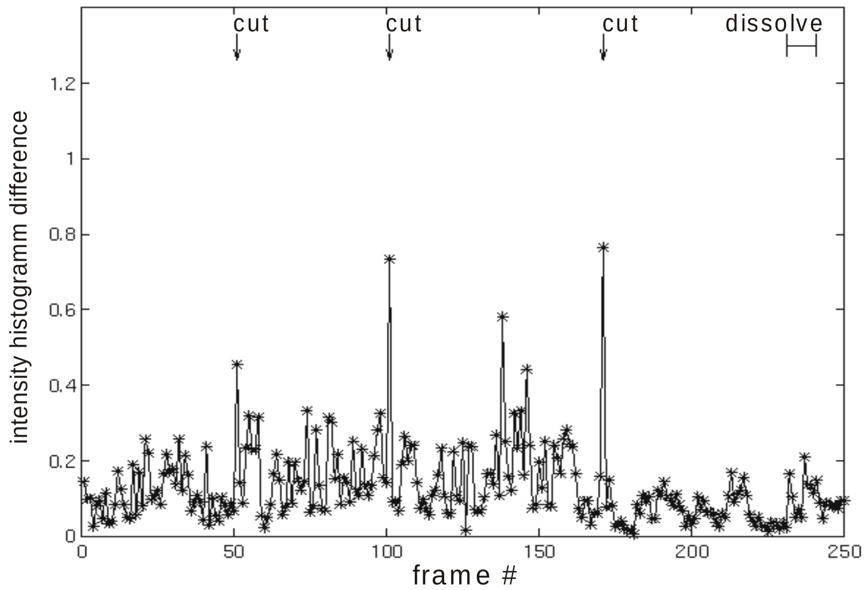
## Cut Detection with Color Histograms

The simplest approach to cut detection is by means of color histograms: If the color histograms of two successive video frames i and i+1 differ at least by a threshold value T, a hard cut is detected.

Let H(r,g,b,i) be the value of the histogram for a color triple (r,g,b) in a frame i, i.e., the number of pixels in frame i that have color (r,g,b). A cut is detected if and only if:

$$\sum_{r,g,b} (H(r,g,b,i) - H(r,g,b,i+1))^2 \geq T$$

# Example: Cut Detection with Gray-scale Histogram Differences






# Typical Detection Errors

If we use color histogram differences, the rate of successful detection of hard cuts is 90% to 98% for a typical video.

This method, however, fails when the colors change significantly between two adjacent frames even if no cut is present. **False hits are much more common than misses.**

**Examples**

- The light is switched on in a room,
- an explosion occurs,
- the camera pans very quickly.

# Cut Detection with the Edge Change Ratio

Generally, the edges in the first frame after a cut differ largely from the edges in the last frame before the cut. Thus the ECR can be used to detect hard cuts.

Let $ECR_i$ be the edge change ration between frame *i* and frame *i+1*. A cut is detected if and only if

$$ECR_i \geq T$$

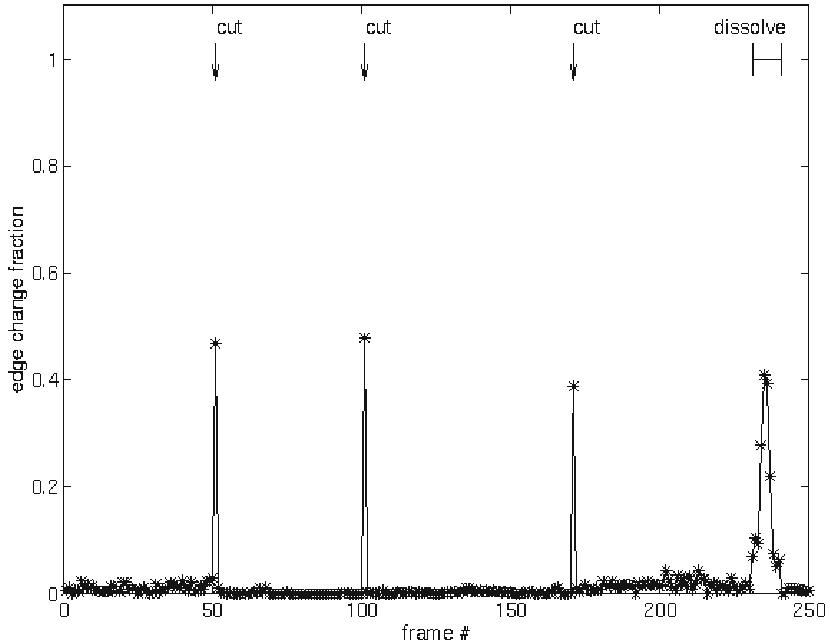
where $T$ is a threshold.

A drawback of this approach is that it requires the computation of motion compensation for a video: fast camera panning or zooming or fast object movements lead to a high ECR for successive frames even if there is no cut.

Object motion or camera operations can be distinguished from a hard cut since they always last for several frames.



# Example: Cut Detection with ECR

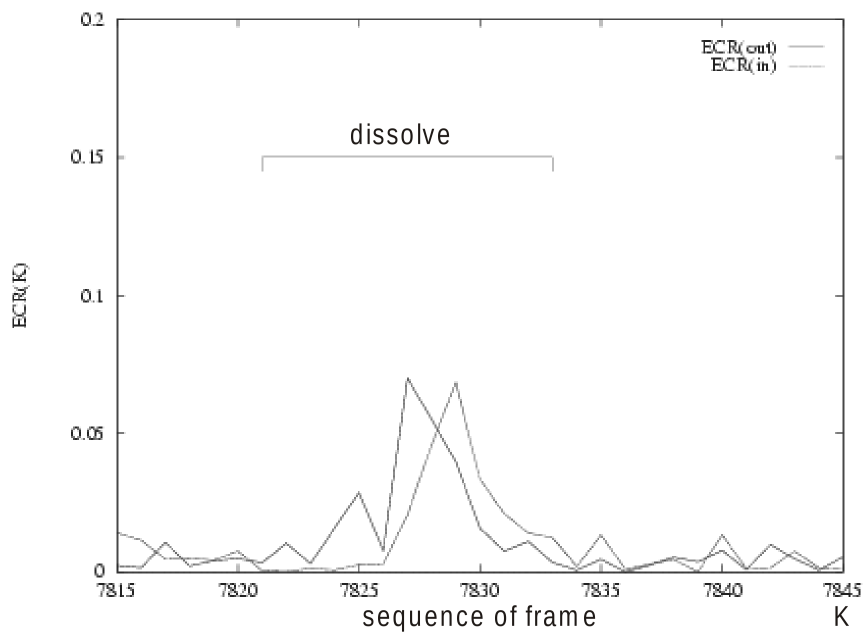


# Detecting Soft Cuts

A fade between successive scenes is much harder to detect than a hard cut. One possibility is to try to detect a characteristic behaviour of the **edge change ratio** (ECR) in the area of fade outs, fade ins and resolves.

**Example:**

During a **resolve**, the edges of the old shot disappear with a constant ratio. At the same time, the edges of the new shot gradually appear. We observe a characteristic pattern of the ECR:



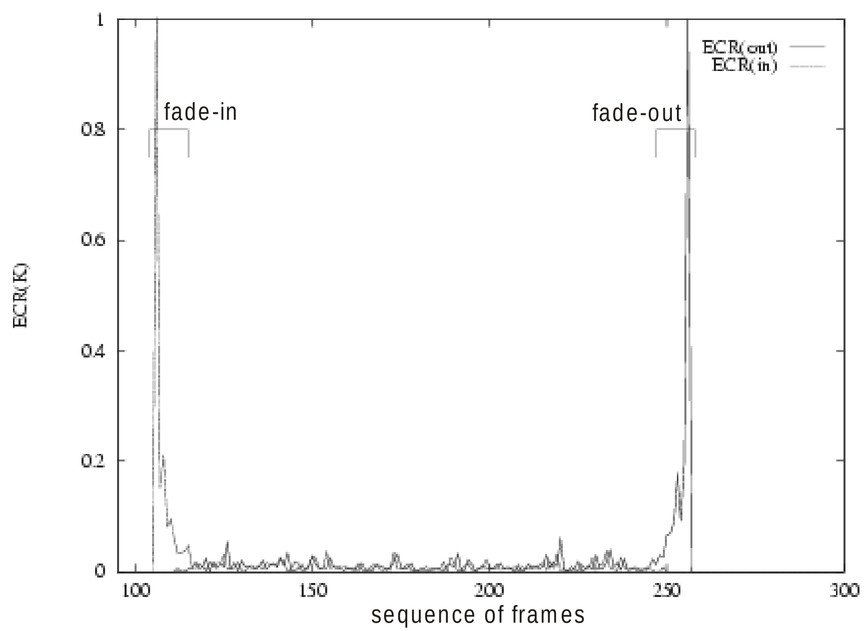



# Fade-ins and Fade-outs (1)

In a similar fashion, fade-ins and fade-outs in a video can be determined: When fading out, the number of pixels located on an edge will be zero in the last frame of the fade. When fading in, the number of pixels located on an edge is zero in the first frame.

**Example:**

ECR during a fade-in and and a fade-out.



## Could We Detect Soft Cuts With Color Histograms?

It is obvious that soft transitions between **color histograms** are frequent in a video, and can have many reasons. Thus, it practically impossible to detect soft cuts with color histogram differences.



## 8.2.2 Action Intensity

The intensity of action in a video shot is an interesting parameter. For example, it can be used to distinguish between different genres of TV broadcasts, such as newscasts vs. music clips.

Action intensity can easily be estimated by means of **motion vectors:** one computes the average absolute value of all vectors in each frame of the shot; this includes both the motion of objects within the scene and camera operations. If the video was compressed using motion vectors, this physical parameter can be computed with minimum cost.

The **edge change ratio** ECR can also be used as an indicator for action: long and relatively static shots have a low ECR while shots with a lot of motion have a high ECR.

## 8.2.3 Detection of Camera Operations

The notion of a camera operation encompasses panning, zooming, etc. It is characteristic for camera operations that they affect all pixels of a frame in a similar, deterministic manner.

**Example 1**

When the camera pans, all pixels move into the same direction, by the same amount, between two frames.

**Example 2**

When the camera zooms in, all pixels (except the one in the center of the frame) move from the center to the boundaries.



## Algorithm for the Detection of Camera Operations

**Algorithm** **Detect-Camera-Operations**

- Use the motion vectors of the underlying compression algorithm (e.g., MPEG-2), or compute the optical flow for a sequence of frames.
- Determine if the direction and the length of the motion vectors fit to the scheme of a pre-defined camera operation.

The detection of camera operations works well when the shot under consideration contains no or little object motion. When camera operation and object motion appear simultaneously (which is quite common in practice), automatic detection becomes very difficult.

# Deriving a Panoramic Image from Camera Motion

When camera motion can be detected automatically, it is also possible to compute a panoramic image from a panning operation.

**Example**

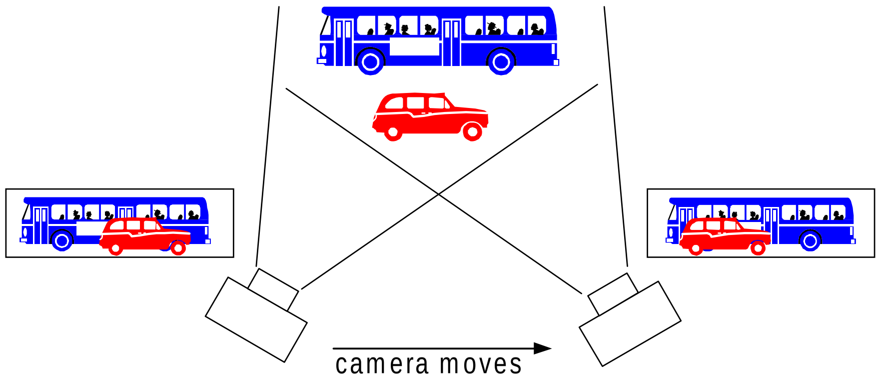
Computation of panoramic images from a video that was recorded by a camera placed on a person‘s head (Steve Mann, MIT Media Lab, 1996)



# Panoramic Image Generation

We want to generate a panoramic image of our environment from a fixed position.

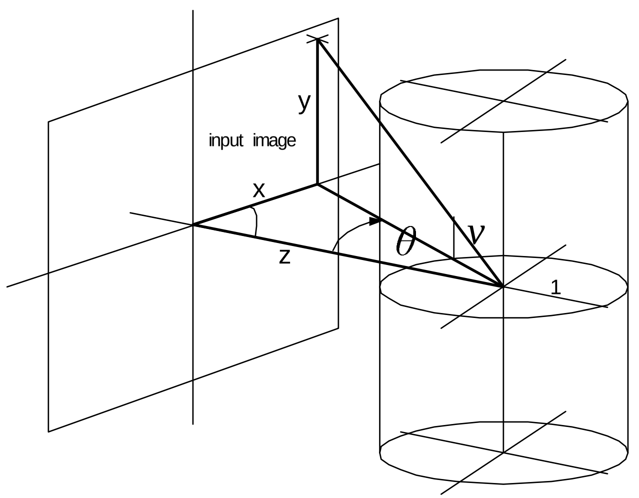
**Parallax effect:** If the camera moves objects at different distances appear shifted against each other.
-> the camera must not change position.




The easy approach: a **cylindrical panorama**. The camera only rotates around its vertical axis. Images are mapped onto a virtual cylinder around the camera position.

# Cylindrical Panoramas (1)

Map the image to cylindrical coordinates.




$$\theta = \arctan(x/z)$$

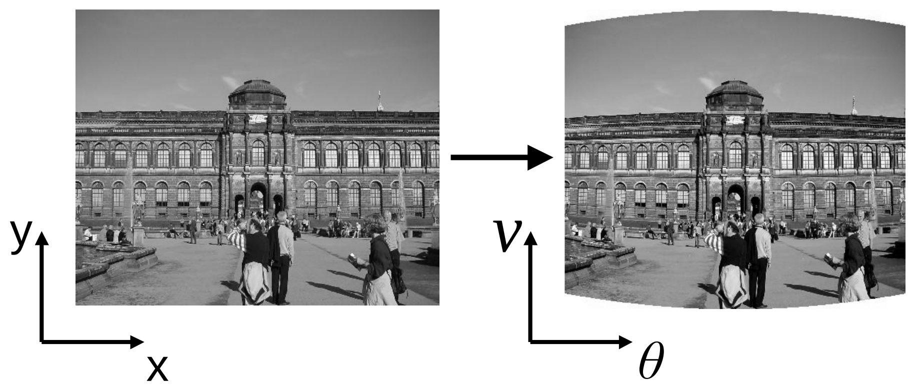
$$v = y/\sqrt{x^2+z^2}$$

Note that the focal-length z has to be known for the coordinate transform.



# Cylindrical Panoramas (2)

Image example



# Cylindrical Panoramas (3)

Transformed images have to be aligned by matching the image content in the overlapping areas.

Find translation vector ($t_x$;$t_y$) by minimizing the matching error between image f and image g:

$$Err(t_x, t_y) = \sum_{x,y} | f(x,y) - g(x+t_x, y+t_y) |^2$$

The minimum can be found by an exhaustive search over ($t_x$;$t_y$). However, computational complexity for image size *NxN* pixels and search range *MxM* pixels is:

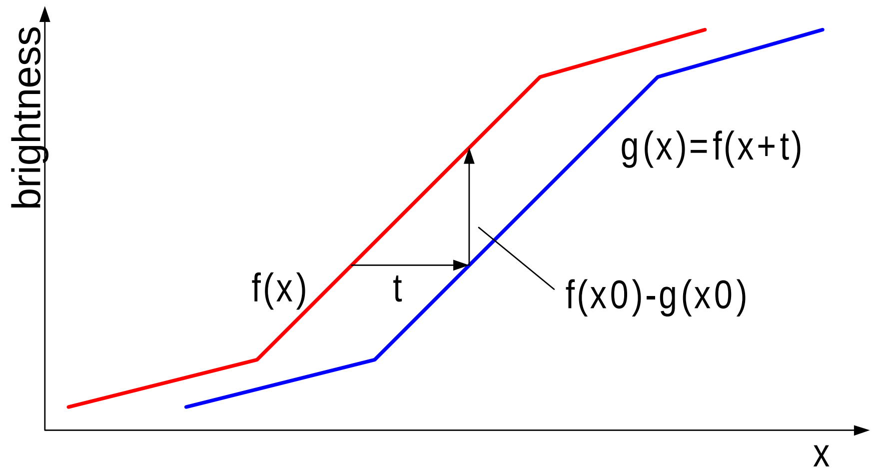
$$O(N^2 M^2) \approx O(N^4)$$

We conclude that we need faster algorithms.



# Cylindrical Panoramas (4)

**Pel-Recursive Motion Estimation:** For simplicity consider the one-dimensional case. Let g(x) be an exact copy of f(x), displaced by a constant t.




With the spatial derivative $\frac{df}{dx}$ of f, we obtain t as:

$$t = \frac{g(x_0) - f(x_0)}{\frac{df}{dx}(x_0)}$$

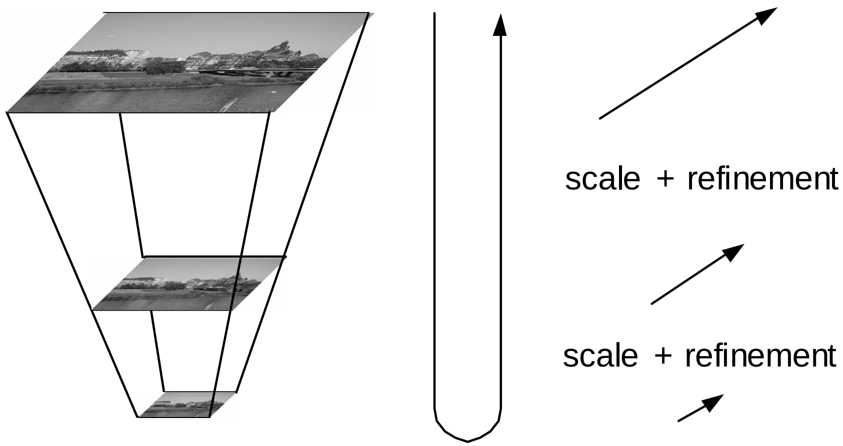
However, this only works for small displacements t.

# Cylindrical Panoramas (5)

## Hierarchical Motion Estimation

**Algorithm**

- Scale the original image down by a constant factor to build a pyramid of the image at different resolutions.
- Do motion estimation on the lowest resolution layer.
- Scale the obtained motion model upward to the next resolution level and refine the motion model.



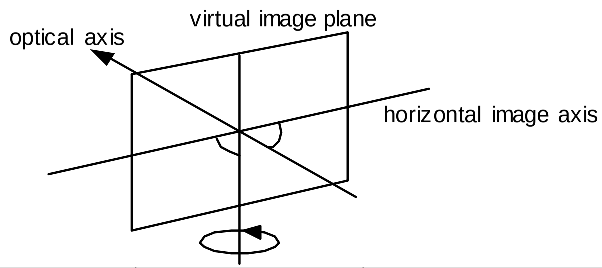



# Cylindrical Panoramas (6)

Example result of a cylindrical panorama:

**Disadvantages of cylindrical panoramas**

- Distortion of straight lines to curved lines
- Focal length (zoom) has to be known or estimated
- Rotation axis has to be perpendicular to the optical axis and to the horizontal image axis:
  - Standing on a hill and looking down is not possible
  - Camera may not be rotated around horizontal axis (but mathematical extension to *spherical panoramas* is possible).



## Full-Perspective Panoramas (1)

Assume that your environment is painted on a single, large plane (environment painted on a glass plane).

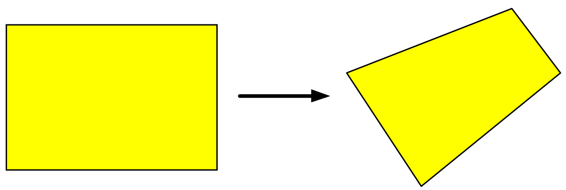
Describe the motion that the solid plane can perform in space.

**2D:** affine motion (translation, rotation, scaling)

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \underbrace{\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}}_{\text{rotate, scale}} \begin{pmatrix} x \\ y \end{pmatrix} + \underbrace{\begin{pmatrix} t_x \\ t_y \end{pmatrix}}_{\text{translate}}$$

**3D:** perspective transformation (affine + 2 additional parameters for perspective projection)

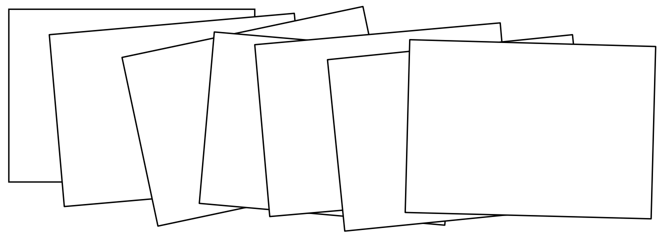
$$x' = \frac{a_{11}x + a_{12}y + t_x}{b_1 x + b_2 y + 1} \qquad y' = \frac{a_{21}x + a_{22}y + t_y}{b_1 x + b_2 y + 1}$$





## Full-Perspective Panoramas (2)

Think of our camera taking images of the glass plane from different orientations. To align the images, the relative transformation between the images has to be estimated.



Exhaustive search for the parameters is not possible because the parameter space is very large (8 parameters).

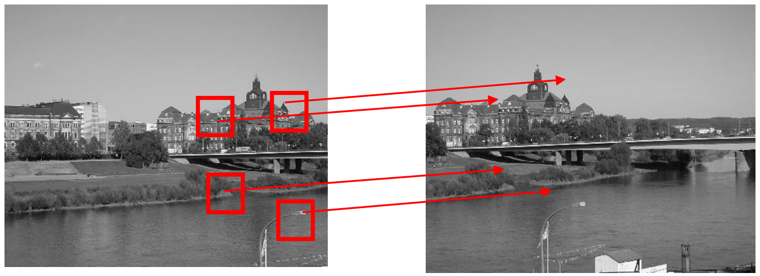
Possible approach for **parameter estimation:**

- Find an estimate with a lower-dimensional model (e.g., translatorial only to coarsely compensate motion)
- Determine an initial estimate for the perspective model (see next slide)
- Use the gradient-descent technique for fine alignment.

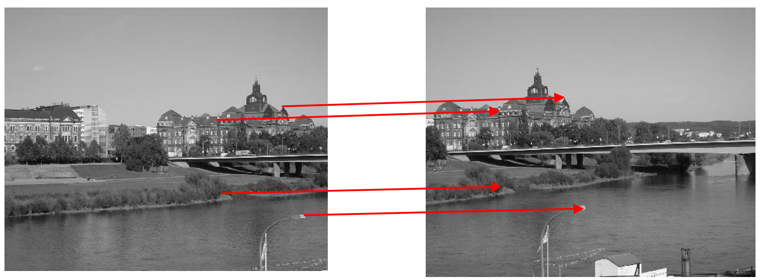
# Full Perspective Panoramas (3)

**Determine initial estimation** of the perspective model:

1. Search for four blocks with characteristic features.



2. Even though we have compensated translatorial motion the features will not match perfectly. Refine matching positions of the blocks in the second image by block-matching.In general, the motion vectors will be different for the four blocks.



3. These four motion vectors (with two parameters each) can be used to determine the eigth parameters of the perspective motion model (eight equations for eight unknowns).



# Full-Perspective Panoramas (4)

**Fine alignment**

1. Start with the parameter vector estimated in last step:

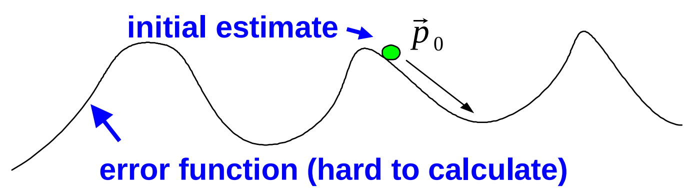
$$p_0 = (a_{11}; a_{12}; a_{21}; a_{22}; t_x; t_y; b_1; b_2)$$

2. Consider the matching-error depending on the transformation parameters:

$$Err(\vec{p}) = \sum_{x,y} | f(x,y) - g(x',y') |^2$$

3. Do a gradient descent search to find a better match.

$$\vec{p}_{i+1} = \vec{p}_i - \nabla Err(\vec{p}_i)$$



# Full-Perspective Panoramas (5)

**Image example**




# 8.2.4 Text Detection

**Goal**

Extraction of text appearing in a video. Motivation: text is rich in semantics.

We distinguish between artificially generated text (such as a movie title) and text appearing in a scene.

**Algorithm**

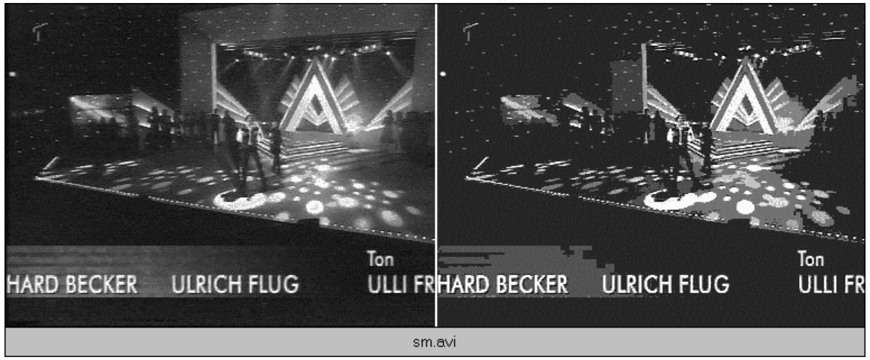
- Detect regions of the frame containing text,
- cut them out,
- run an OCR algorithm (optical character recognition.

**Characteristics of generated text in a video**

- monochrome
- rigid
- in the foreground
- the letters have a minimum and maximum size
- either stationary or moving linearly
- high contrast to background
- appears repeatedly in successive frames
- letters appear in groups.

# Text Segmentation (1)

**Text is monochrome**




**original frame of a video** **segmentation into regions**



# Text Segmentation (2)

Apply thresholds for the size of the letters and the contrast.


**previous result** **after thresholding**

# Text Segmentation (3)

The text must be either stationary or moving linearly (here: horizontally).


**previous result** **after application of the rule of horizontal motion**



# Experimental Results

Detection of text regions in a video

| test video | number of frames | number of letters | successful findings |
|---|---|---|---|
| **title sequences** | 7372 | 6423 | 99% |
| **advertisements** | 6858 | 1065 | 99% |
| **newscast** | 18624 | 1411 | 97% |

(from the dissertation of Rainer Lienhart, Praktische Informatik IV, U. Mannheim)

# 8.2.5 Face Detection

## Goal

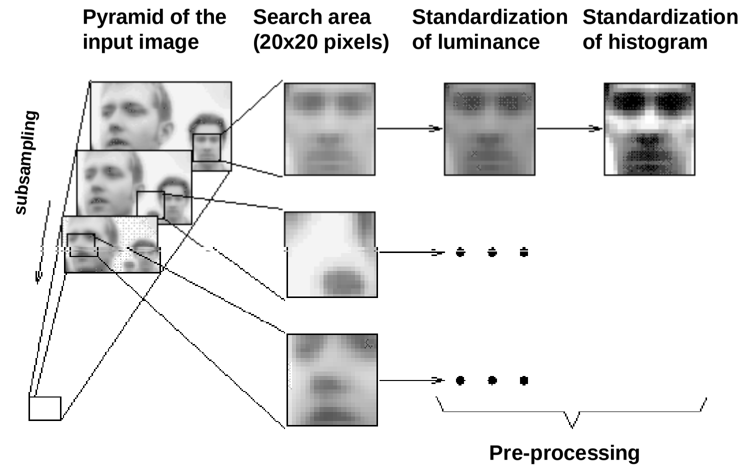
Detection of areas in a video frame that show the frontal view of a human face.

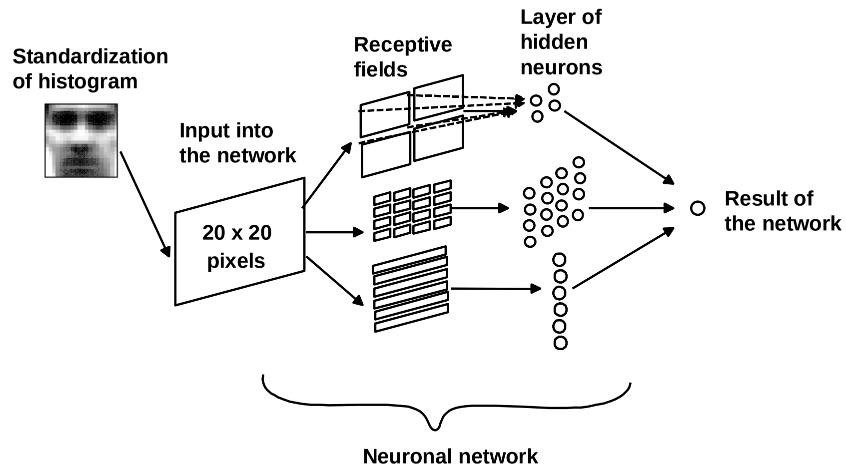
## Approach

- Construct a neural network for the detection of texture
- Train this network with thousands of faces, where the line between the eyes as well as an orthogonal line towards the tip of the nose are marked
- Process an unknown image with search areas in varying sizes:
  - pre-processing / filtering / standardization of the illumination
  - test on the hypothesis “face” with the trained neural network.

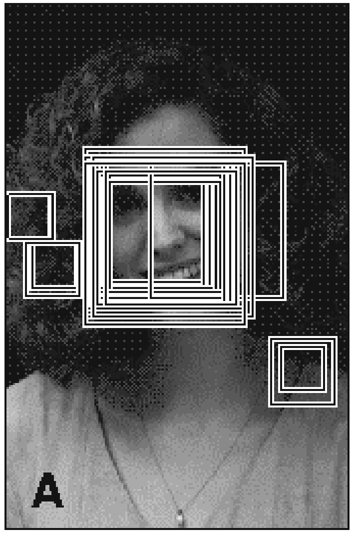


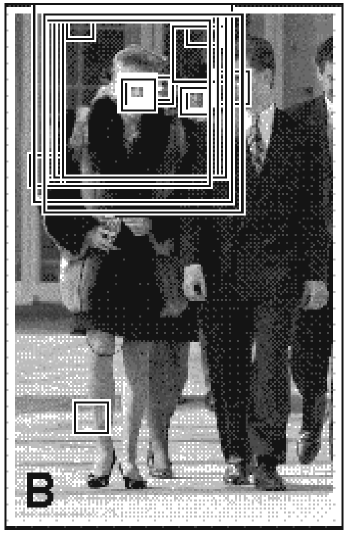
# Algorithm ”Face Detection”



# Multiple Detection






# Visualization of Results