CHAPTER 21

VIDEO SAMPLING RATE AND STANDARDS CONVERSION

WHAT WILL WE LEARN?

- What is sampling rate conversion?
- What are the main practical aspects involved in converting a video sequence from one format to another?
- Which steps are involved in PAL to NTSC (and vice versa) standard conversion?

21.1 VIDEO SAMPLING

As discussed in Chapter 20, although video signals vary continuously in space and time, most TV cameras capture a video sequence by sampling it in the temporal and vertical dimensions. The resulting signal is stored in a 1D raster scan, which is a concatenation of the intensity (or color) variations along successive horizontal scan lines. A third type of sampling (along each line) takes place when an analog video is converted to a digital representation. In all three cases, the fact that selected intensity values are being sampled (while others are being left off) implies that there might be some loss in the process. To minimize such losses, the original video sequence should be sampled at the highest possible sampling rate.

Practical Image and Video Processing Using MATLAB®. By Oge Marques.

Based on Nyquist's sampling theorem, the sampling rate in each dimension (x, y, or t) should be at least twice the highest frequency along that direction. On the other hand, studies of the human visual system (HVS) (see Appendix A) have shown that our visual system cannot distinguish spatial and temporal variations beyond certain high frequencies. Consequently, the *visual cutoff frequencies*—which are the highest spatial and temporal frequencies that can be perceived by the HVS—should be the driving factor in determining the sampling rates for video. After all, there is no need to accommodate frequency components beyond those values. In addition to the frequency content of the underlying signal and the visual thresholds in terms of spatial and temporal cutoff frequencies imposed by the HVS, video sampling rates are also determined by the capture and display device characteristics and the associated processing, storage, and transmission costs.

Video sampling is a complex problem that can be mathematically modeled using *lattice theory*, which is beyond the scope of this book.[1] Lattice theory provides an elegant framework to model relationships between the engineering decisions that require adopting less than ideal sampling rates and the impact of those decisions on the frequency-domain representation of the sampled signals.

One of the consequences of using relatively low sampling rates is the problem of *aliasing*, which can manifest itself within a frame (*spatial aliasing*) or across multiple frames (*temporal aliasing*). Spatial aliasing, which is common to images and videos, particularly the appearance of *Moiré patterns*, was discussed in Chapter 5. Temporal aliasing is a related phenomenon, particularly noticeable in video sequences where wheels sometimes appear to move backward and slower, in what is known as the *wagon wheel effect* (also known as the *reverse rotation effect*).

21.2 SAMPLING RATE CONVERSION

The sampling rate conversion problem consists in converting a video sequence with a certain spatial and temporal resolution into another sequence in which one or more of those parameters have changed. Sampling rate conversion falls within one of the following two cases:

- When the original sequence contains *less* sampling points than the desired result, the problem is known as *up-conversion* (or *interpolation*). Up-conversion usually consists of filling all points that are in the desired video sequence but not in the original one with zeros (a process known as *zero padding*), and then estimating the values of these points, which is accomplished by an interpolation filter. Examples of up-conversion involving contemporary TV and video standards include
  - 480i59.94 to 720p59.94
  - 480i59.94 to 1080i59.94

[1] See Chapter 3 of [WOZ02] for a good explanation of video sampling using lattice theory.

  - 576i50 to 576p50
  - 576i50 to 1080i50

  In practice, up-conversion requires increase in resolution and calls for deinterlacing techniques (see Section 21.3.1).
- When the original sequence contains *more* sampling points than the desired result, the problem is known as *down-conversion* (or *decimation*). This is not as simple as just removing additional samples from the original sequence, which would cause aliasing in the down-sampled signal. Proper down-conversion requires applying a prefilter to the signal to limit its bandwidth, therefore avoiding aliasing. In Tutorial 21.1 (page 548), you will learn how to perform line down-conversion using MATLAB.

21.3 STANDARDS CONVERSION

Sampling rate conversion is also sometimes referred to as *standards conversion*. In the context of this book, a *standard* can be defined as a video format that can be specified by a combination of four main parameters: color encoding (composite/component), number of lines per field/frame, frame/field rate, and scanning method (interlaced/progressive). Hence, standard conversion will be understood as the process of converting one or more of those parameters into another format. Contemporary standard conversion examples include

- Frame and line rate conversion, for example, 1250i50 to 525i60 and 625i50 to 1250i50.
- Color standard conversion, for example, PAL to NTSC (see Section 21.3.2).
- Line and field doubling, for example, 625i50 to 1250i100.
- Film to video conversion, for example, 24 Hz film to 60 Hz video, a process known as *3:2 pull-down* (see Section 21.3.5).

21.3.1 Deinterlacing

The problem of deinterlacing is to fill in the skipped lines in each field, as shown in Figure 21.1.

Deinterlacing can be achieved with the use of relatively simple methods[2] (which can be treated as filters in the spatio-temporal domain) that fall into the following categories:

- Vertical interpolation within the same field
  - *Line Averaging*: a simple filter in which the missing line is estimated by averaging the lines above and below: $D = (C + E)/2$.

[2]In Tutorial 21.2 (page 550), you will learn how to perform basic deinterlacing methods using MATLAB.

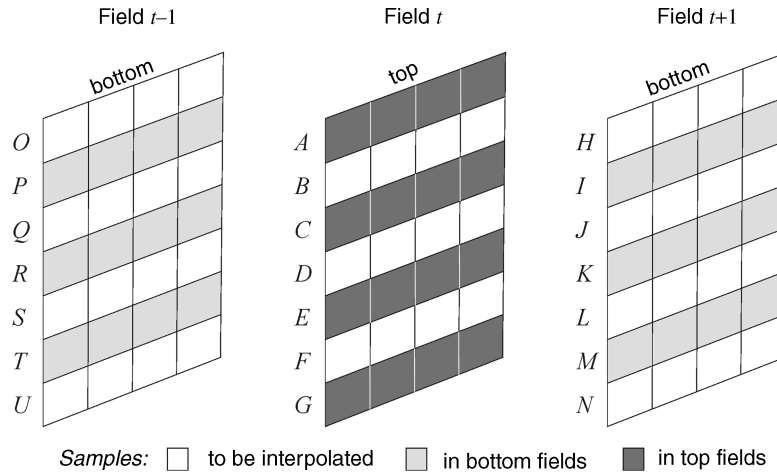

FIGURE 21.1 The deinterlacing process. Fields t and $t + 1$ form one interlaced frame.

  - *Line Averaging Variant*: improves the vertical frequency response upon the previous technique, bringing it closer to the desired (ideal) low-pass filter. In this technique, the missing line is calculated as $D = (A + 7C + 7E + G)/16$.
- Temporal interpolation
  - *Field Merging*: technique by which the deinterlaced frame is obtained by combining (merging) the lines from both fields: $D = K$ and $J = C$. The frequency response of the equivalent filter shows an all-pass behavior along the vertical axis and a low-pass behavior along the temporal axis.
  - *Field Averaging*: improved alternative in which $D = (K + R)/2$, which results in a filter with better (symmetric) frequency response along the vertical axis. The main disadvantage of this technique is that it involves three fields for interpolating any field, increasing storage and delay requirements.
- Temporal interpolation and vertical interpolation combined
  - *Line and Field Averaging*: a technique designed with the goal of achieving a compromise between spatial and temporal artifacts, in which the missing line is calculated as $D = (C + E + K + R)/4$. The frequency response of the equivalent filter is close to ideal, but the method requires storage of two frames.

When there is little or no motion between the two fields, the missing even lines in an odd field should be exactly the same as the corresponding even lines in the previous and the following even field. In these cases, temporal interpolation will yield perfect results. On the other hand, when there is motion in the scene, the corresponding lines in adjacent fields may not correspond to the same object location and temporal interpolation will result in unacceptable artifacts, such as double images. In these cases, although the results of vertical interpolation methods may be acceptable (at the

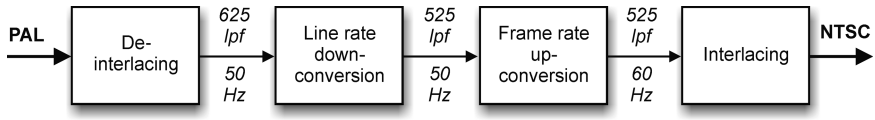

FIGURE 21.2 A practical method for converting PAL to NTSC formats.

cost of lower spatial resolution), the preferred alternative is to use motion-adaptive and motion-compensated interpolation methods.

21.3.2 Conversion between PAL and NTSC Signals

The conversion between PAL and NTSC signals (which in fact should be called *625i50 to 525i60 conversion*) provides a practical example of line number and frame rate conversions combined. This is a particularly challenging technical problem because the vertical and temporal sampling rates of the two video sequences are not integer multiples of each other.

In practice, the problem can be solved in four steps (Figure 21.2):

1. Deinterlacing each field in the PAL signal, resulting in a 625-line frame.
2. Line rate down-conversion from 625 to 525 within each deinterlaced frame.
3. Frame rate up-conversion from 50 to 60 Hz.
4. Splitting each frame into two interlacing fields.

The PAL to NTSC conversion process can be further optimized by

- Replacing the 625-to-525 lines conversion by an equivalent 25-to-21 lines conversion problem.
- Performing the temporal conversion from 50 to 60 Hz by converting every five frames to six frames.

The process of converting NTSC video to its PAL equivalent follows a similar sequence of steps, with obvious modifications: the line rate undergoes up-conversion (from 525 to 625), while the frame rate is down-converted (from 60 to 50 Hz). In Tutorial 21.3 (page 556), you will learn how to perform NTSC to PAL conversion using MATLAB.

21.3.3 Color Space Conversion

Color space conversion is the process of transforming the input color space to that of the output. As discussed previously, there are multiple color spaces in use by different video systems. To ensure the best possible color reproduction quality when converting from one standard to another, it might be necessary to adjust the RGB primaries and the white point to account for the differences in the standards.

In particular, the conversion from Rec. 601 SDTV images to SMPTE 274M HDTV images requires a manipulation of the color matrices to account for, among other things, the different phosphors used in HDTV monitors [Dea01].

The SMPTE 125M—Rec. 601 (Standard Definition) standard uses SMPTE RP145 for chromaticity coordinates:[3]

- Red ($x = 0.6400$, $y = 0.3300$)
- Green ($x = 0.3000$, $y = 0.6000$)
- Blue ($x = 0.1500$, $y = 0.0600$)
- White ($x = 0.3127$, $y = 0.3290$)
- $Y' = 0.2126R' + 0.7152G' + 0.0722B'$
- $U = -0.1146R' - 0.3854G' + 0.5000B'$
- $V = 0.5000R' - 0.4542G' - 0.0458B'$

The SMPTE 274M/296M—1080i/720p (High Definition) standard uses ITU-R BT.709 for all colorimetry measurements:

- Red ($x = 0.6300$, $y = 0.34004$)
- Green ($x = 0.3100$, $y = 0.5950$)
- Blue ($x = 0.1550$, $y = 0.0700$)
- White ($x = 0.3127$, $y = 0.3290$)
- $Y' = 0.2990R' + 0.5870G' + 0.1140B'$
- $U = -0.1690R' - 0.3310G' + 0.5000B'$
- $V = 0.5000R' - 0.4190G' - 0.0810B'$

Color space conversion can be performed by applying a linear transformation to the gamma-corrected components, which produces errors that are still acceptable for most applications. High-end format converters perform additional inverse gamma correction before the linear transformation stage and restore gamma correction after the colors have been linearly mapped to the desired color space, a process that is significantly more complex and expensive to implement [Dea01].

21.3.4 Aspect Ratio Conversion

Aspect ratio (AR) conversion is another aspect that must be addressed when performing certain types of conversion, for example, from SDTV (4:3 aspect ratio) to HDTV (16:9 aspect ratio) or from "widescreen" movies (AR greater or equal to 1.85) to a "full screen" SDTV equivalent (AR equal to 1.33). AR conversion techniques include cropping, stretching, or squeezing each frame, which are not very complex operations. These operations do, however, bring about a number of creative issues

[3] See Section 16.1.2 for an explanation of basic concepts related to chromaticity.

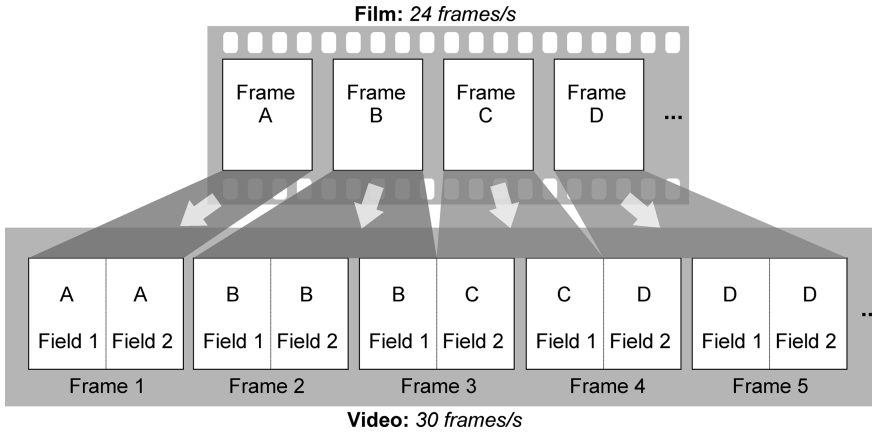

FIGURE 21.3 3:2 pull-down. Redrawn from [Ack01].

related to how frames can be resized and reshaped while keeping most of the contents as intended by the author of the original sequence.

21.3.5 3:2 Pull-Down

3:2 pull-down[4] refers to a technique used to convert material originated at 24 frames per second (fps), typically shot on film, to 30 or 60 fps video for broadcast.[5] In the case of 30 fps interlaced television, this means that each film of four frames is converted to 10 fields or 5 frames of video. In the case of 60 fps progressive scan television, each film of four frames is converted into 10 frames of video.

Figure 21.3 shows the 3:2 pull-down process for converting film material to interlaced video. The film frames are labeled A, B, C, and D; the video frames are numbered 1–5 and the corresponding video fields are called A1, A2, B1, B2, B3, C1, C2, D1, D2, and D3. Note that video frame 3 will contain video from film frames B and C and video frame 4 will contain video from film frames C and D. This is potentially problematic and may cause *mixed frames* to appear in the resulting video sequence when a scene change has occurred between a B and a C frame or between a C and a D frame in the original film sequence.

Another technical issue associated with the conversion from motion picture films to their standard video equivalent (a process known as *telecine*) is the appearance of *judder*. Judder is a motion artifact that causes smooth motion in the original film sequence to appear slightly jerky when telecined. Judder is particularly apparent during slow, steady camera movements. Figure 21.4 shows how the process of copying film frames into video fields using 3:2 pull-down causes the smooth optic flow in the

[4] 3:2 pull-down is also known as *2:3 pull-down* in some references, for example, [Ack01].

[5] The process of 3:2 pull-down is necessary only for video formats based on 60 Hz. In 50 Hz systems, the film is simply transferred from 24 to 25 fps (50 fields/s) by running the telecine 4% faster than normal.

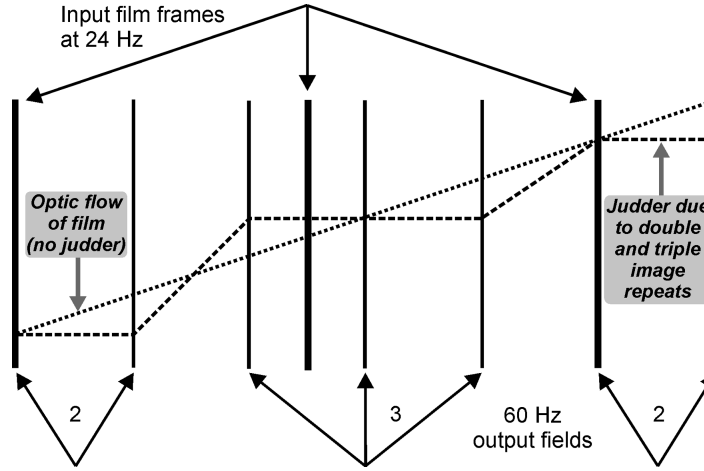

FIGURE 21.4 The problem of judder in telecine using 3:2 pull-down. Redrawn from [Wat94b].

original film sequence to become jerky, with no motion for two or three fields followed by sudden jumps in motion.

21.4 TUTORIAL 21.1: LINE DOWN-CONVERSION

Goal

The goal of this tutorial is to learn how to perform line down-conversion on a video sequence.

Objectives

- Learn how to extract a YUV frame from a video sequence using the `readYUV` function.
- Explore the MATLAB `resample` function to perform up- and down-conversion.
- Explore line down-conversion using the `downConversion` function.

What You Will Need

- MATLAB files `readYUV.m` and `downConversion.m`[6]
- Test file `miss_am.yuv`

[6]These functions were developed by Jeremy Jacob and can be downloaded from the book web site.

Procedure

We will use the `readYUV` function to read an individual frame from a YUV video file. Recall that when we use this function, both a MATLAB video structure and the YUV frame(s) are returned.

1. Read in the first frame of the `miss_am.yuv` file.

```
filename = 'miss_am.yuv';
fileformat = 'QCIF_PAL';
numframes = 1;
[movie, yuv] = readYUV(filename, numframes, fileformat);
```

The function `resample` will resample an image by a factor of p/q, where p and q are defined in the function call. Note that this function will resample only the columns of a matrix, so we must perform the operation twice: once on the transpose of the image and once on the transpose of the transpose, which will bring back the original orientation.

2. Resample the Y component of the frame by a factor of 2.

```
currY = yuv(:, :, 1, 1);
currY_res = double(currY);
currY_res = resample(currY_res', 2, 1);
currY_res = resample(currY_res', 2, 1);
```

Question 1 What is the size of the `currY_res` image? How does this relate to the original image (`currY`)?

To perform line down-conversion, we use the `downConversion` function.

3. Perform line down-conversion on the current frame.

```
currY_dwn = downConversion(currY_res);
figure, subplot(1,2,1), imshow(uint8(currY_res)), ...
    title('2X resampled frame');
subplot(1,2,2), imshow(uint8(currY_dwn)), ...
    title('line down-converted frame');
```

Question 2 How does the size of the converted image compare to the original resampled image?

4. Crop the original image at the bottom and compare it with the line down-converted image.

```
[r, c] = size(currY_dwn);
currY_crp = currY_res(1:r,:);
figure, subplot(1,2,1), imshow(uint8(currY_dwn)), ...
    title('Line down-converted');
subplot(1,2,2), imshow(uint8(currY_crp)), ...
    title('Cropped');
```

Question 3 How does line down-conversion compare to cropping an image?

21.5 TUTORIAL 21.2: DEINTERLACING

Goal

The goal of this tutorial is to learn several methods for deinterlacing a video sequence.

Objectives

- Learn how to obtain the two fields (top and bottom) of an interlaced movie frame.
- Explore deinterlacing by spatial interpolation.
- Explore deinterlacing by line averaging.
- Learn how to perform deinterlacing by temporal interpolation: field merging and field averaging methods.
- Explore deinterlacing by line and field averaging.
- Learn how to create a new *YUV* frame composed of the deinterlaced *Y* frame and the *U*, *V* color component frames.

What You Will Need

- `readYUV.m`
- Test file `foreman.yuv`

Procedure

To begin working with interlaced frames, we will read three frames of a *YUV* video sequence using the `readYUV` function.

1. Set the appropriate variables and read in the first three frames of the `foreman.yuv` video file.

```
filename = 'foreman.yuv';
numframes = 3;
fileformat = 'QCIF_PAL';
[mov,yuv] = readYUV(filename, numframes, fileformat);
```

For the first exercise, we will need only one frame, so we will use the second frame in the sequence. Later we will see that having a frame before and after it will be useful.

2. Extract the *Y* component of the second frame and store its size for later use.

```
currY = yuv(:, :, 1, 2);
[rows, cols] = size(currY);
```

Question 1 Explain how the statement `currY = yuv(:, :, 1, 2);` above extracts the *Y* component of the second frame.

Interlacing

To simulate interlaced frames, we can simply extract the even lines and store them in one variable and the odd lines in another.

3. Create interlaced frames and display them with the original *Y* component.

```
topField = currY(1:2:rows,:);
bottomField = currY(2:2:rows,:);
figure, subplot(2,2,1), imshow(uint8(currY));
subplot(2,2,3), imshow(uint8(topField));
subplot(2,2,4), imshow(uint8(bottomField));
```

4. Restore the interlaced frames to the original size of the frame by filling the blank lines with zeros.

```
topField2 = zeros(rows,cols);
bottomField2 = zeros(rows,cols);
topField2(1:2:rows,:) = topField;
bottomField2(2:2:rows,:) = bottomField;
figure, subplot(2,2,1), imshow(uint8(currY));
subplot(2,2,3), imshow(uint8(topField2));
subplot(2,2,4), imshow(uint8(bottomField2));
```

The lines in the interlaced frames may look distorted due to the resolution at which MATLAB displays the images in the figure. To see a particular frame without distortion, display the frame in its own figure without any subplots.

Line Averaging

We can use spatial interpolation to generate a full frame from an interlaced frame. To do this, we average two lines together and use this to fill in the missing lines.

5. Deinterlace the top and bottom frames using line averaging.

```
deIntField1 = topField2;
deIntField2 = bottomField2;
for m=1:2:rows-2
     for n=1:cols
        lineAverage1(m+1,n) = (deIntField1(m,n) + ...
            deIntField1(m+2,n)) / 2;
     end
end
lineAverage1(1:2:rows,:) = topField2(1:2:rows,:);
for m=2:2:rows-1
     for n=1:cols
        lineAverage2(m+1,n)=(bottomField2(m,n) + ...
            bottomField2(m+2,n)) / 2;
     end
end
lineAverage2(2:2:rows,:) = bottomField2(2:2:rows,:);
```

6. Display the results.

```
figure
subplot(2,2,1), imshow(uint8(currY)), ...
    title('Original frame');
subplot(2,2,3), imshow(uint8(lineAverage1)), ...
    title('Top field averaged');
subplot(2,2,4), imshow(uint8(lineAverage2)), ...
    title('Bottom field averaged');
```

The deinterlaced top frame, stored in `lineAverage1`, is missing one line, as you might have noticed by inspecting its dimensions compared to the original frame. This last line is skipped in the above procedure because there is no line available below (since it is the last line in the frame). Therefore, we will add a line of zeros just to keep the dimensions consistent.

7. Compensate for the missing line in the `lineAverage1` frame.

```
lineAverage1(rows,:) = zeros(1,cols);
```

8. Display the results.

```
figure
subplot(2,2,1), imshow(uint8(currY))
subplot(2,2,3), imshow(uint8(lineAverage1))
subplot(2,2,4), imshow(uint8(lineAverage2))
```

Question 2 This "missing line effect" is also noticeable in `lineAverage2` (the bottom field), but it is at the top of the frame. Why do you think we need to compensate for this in the top field but not the bottom?

Question 3 What are the visual differences between the original frame and the deinterlaced frame?

We can use the `imabsdiff` function to see how the interpolated frames compare to the original.

9. Display the difference between the top and bottom fields to the original frame.

```
dif1 = imabsdiff(double(currY), lineAverage1);
dif2 = imabsdiff(double(currY), lineAverage2);
figure
subplot(1,2,1), imshow(dif1,[]), title('Difference of top field');
subplot(1,2,2), imshow(dif2,[]), title('Difference of bottom field');
```

Because we already have the *U* and *V* components available, we can combine them with one of the new interpolated frames to see what it would look like as an *RGB* image.

10. Convert the interpolated top field to an *RGB* image using the original *U* and *V* components.

```
yuv_deint = yuv(:,:,:,2);
yuv_deint(:,:,1) = lineAverage1;
rgb_deint = ycbcr2rgb(uint8(yuv_deint));
rgb_org = ycbcr2rgb(uint8(yuv(:,:,:,2)));
figure
subplot(1,2,1), imshow(rgb_org), title('Original Frame')
subplot(1,2,2), imshow(rgb_deint), title('Interpolated top field')
```

Question 4 How does the interpolated *RGB* image compare to its original?

Of course, since we have both top and bottom fields, we can merge them, which will result in the original image.

11. Merge the two fields into a single frame.

```
mergeFields = zeros(rows, cols);
mergeFields(1:2:rows,:) = topField2(1:2:rows,:);
mergeFields(2:2:rows,:) = bottomField2(2:2:rows,:);
figure
subplot(1,2,1), imshow(uint8(currY)), ...
    title('Original frame');
```

```
subplot(1,2,2), imshow(uint8(mergeFields)), ...
    title('Merged fields');
```

Question 5 Show that this frame is equivalent to the original by using the `imabsdiff` function.

Field Averaging

Deinterlacing by field averaging requires information from both the preceding frame and the next frame in the sequence. To perform the averaging, we use the lines from the bottom field of the previous frame and the next frame to calculate the missing lines for the top field of the current frame. Similarly, we will use the lines from the top field in the previous and next frames to calculate the missing lines in the bottom field of the current frame.

12. Close any open figures.
13. Display the *Y* components of the previous, current, and next frame.

```
prevY = yuv(:, :, 1, 1);
nextY = yuv(:, :, 1, 3);
figure
subplot(1,3,1), imshow(uint8(prevY)), title('Previous Frame');
subplot(1,3,2), imshow(uint8(currY)), title('Current Frame');
subplot(1,3,3), imshow(uint8(nextY)), title('Next Frame');
```

14. Extract the top and bottom fields for both the previous and next frames.

```
topFieldPrev = zeros(rows,cols);
topFieldNext = zeros(rows,cols);
topFieldCurr = topField2;
bottomFieldPrev = zeros(rows,cols);
bottomFieldNext = zeros(rows,cols);
bottomFieldCurr = bottomField2;
topFieldPrev(1:2:rows,:) = prevY(1:2:rows,:);
topFieldNext(1:2:rows,:) = nextY(1:2:rows,:);
bottomFieldPrev(2:2:rows,:) = prevY(2:2:rows,:);
bottomFieldNext(2:2:rows,:) = nextY(2:2:rows,:);
```

15. Perform the averaging.

```
for m=2:2:rows
    for n=1:cols
        fieldAverage1(m,n)=(bottomFieldPrev(m,n) + ...
            bottomFieldNext(m,n)) / 2;
```

```
    end
end
fieldAverage1(1:2:rows,:) = topFieldCurr(1:2:rows,:);
for m=1:2:rows
    for n=1:cols
        fieldAverage2(m,n) = (topFieldPrev(m,n) + ...
            topFieldNext(m,n)) / 2;
    end
end
fieldAverage2(2:2:rows,:) = bottomFieldCurr(2:2:rows,:);
```

16. Display the results.

```
figure
subplot(1,2,1), imshow(uint8(fieldAverage1)), ...
    title('Top field');
subplot(1,2,2), imshow(uint8(fieldAverage2)), ...
    title('Bottom field');
```

Question 6 How does this technique compare to line averaging? More specifically, under what circumstances does this technique perform better/worse than the line averaging scheme?

Line and Field Averaging

Deinterlacing by line and field averaging also requires information from the previous and the next frame. This method is a combination of line averaging and field averaging. The missing line of the current top field is filled with the previous and next line of the current top field averaged together with the lines from the bottom field in the previous frame and lines from the bottom field in the next frame.

17. Perform line and field averaging using the frame fields previously defined.

```
for m=2:2:rows-2
     for n=1:cols
        lineFieldAverage1(m,n) = (topFieldCurr(m-1,n) + ...
            topFieldCurr(m+1,n)+ ...
            bottomFieldPrev(m,n) + ...
            bottomFieldNext(m,n)) / 4;
     end
end
lineFieldAverage1(1:2:rows,:) = topFieldCurr(1:2:rows,:);
for m=3:2:rows
    for n=1:cols
```

```
        lineFieldAverage2(m,n) = (bottomFieldCurr(m-1,n) + ...
            bottomFieldCurr(m+1,n) + ...
            topFieldPrev(m,n) + ...
            topFieldNext(m,n)) / 4;
    end
end
lineFieldAverage2(2:2:rows,:) = bottomFieldCurr(2:2:rows,:);
figure
subplot(1,2,1), imshow(uint8(lineFieldAverage1)), ...
    title('Top field');
subplot(1,2,2), imshow(uint8(lineFieldAverage2)), ...
    title('Bottom field');
```

Question 7 How does line and field averaging compare to just field averaging?

21.6 TUTORIAL 21.3: NTSC TO PAL CONVERSION

Goal

The goal of this tutorial is to learn how to convert an NTSC formatted video sequence to its equivalent in the PAL format.

Objectives

- Explore the steps in the function `ntsc2pal`, which converts NTSC video to PAL video.

What You Will Need

- MATLAB m-files `readYUV.m`, `ntsc2pal.m`, `downConversion.m`, and `deinterlace.m`
- Test file `whale_show.yuv`

Procedure

The MATLAB function `ntsc2pal.m`[7] performs the NTSC to PAL conversion.

1. Open `ntsc2pal.m` in the MATLAB Editor and answer the questions below.

Question 1 What deinterlacing method is used in this case?

[7]This function was developed by Jeremy Jacob and can be downloaded from the book web site.

Question 2 What other deinterlacing possibilities are available with the `deinterlace` function used in the above code? If we were to use other methods, would we need to change the code for it to work?

Let us now perform an NTSC to PAL conversion of the `whale_show.yuv` video sequence.[8]

2. Load the first 30 frames of the YUV video sequence `whale_show.yuv` using the `readYUV` function.

```
[mov_org, yuv_org] = readYUV('whale_show.yuv', 30, 'NTSC');
```

3. Convert the video sequence to PAL.

```
[mov_new, yuv_new] = ntsc2pal(yuv_org);
```

Question 3 Inspect the dimensions of the original *YUV* array and the new *YUV* array. How do they compare?

Question 4 Why is the number of frames in the new video sequence different from the original?

4. Use the `implay` function to view the original and new video sequences (one at a time, or simultaneously if enough system resources are available). Be sure to set the frame rate appropriately (30 fps for the original and 25 fps for the new video).

Question 5 Subjectively, how does the quality of the new video sequence compare to the original?

21.7 TUTORIAL 21.4: 3:2 PULL-DOWN

Goal

The goal of this tutorial is to learn how to convert from 24 fps film to 30 fps NTSC video using the 3:2 pull-down method.

Objectives

- Explore the steps in the function `pulldown_32`, which converts 24 fps film to 30 fps video.

[8]Note that when the function runs, it uses a lot of memory resources, and so it is recommended that you close any other programs you are not currently using in order to free system resources.

What You Will Need

- `readYUV.m`
- `pulldown_32.m`
- Test file `whale_show.yuv`

Procedure

This tutorial teaches the process of converting 24 fps film to standard 30 fps NTSC video using the 3:2 pull-down method, implemented in function `pulldown_32.m`.[9]

1. Load the `whale_show.yuv` video file, but load only the first 24 frames.

```
[mov_org, yuv_org] = readYUV('whale_show.yuv', 24, 'NTSC');
```

2. Perform a 3:2 pull-down conversion on the video sequence.

```
[mov_new, yuv_new] = pulldown_32(yuv_org);
```

Question 1 Use `implay` to compare the quality between the original and converted video sequences.

Question 2 Write a small script that will simply copy the first four frames and generate the fifth frame by making a copy of the fourth. How does this compare to the 3:2 pull-down method?

WHAT HAVE WE LEARNED?

- Sampling rate conversion is a common technical problem in video processing. Particular cases include up-conversion, down-conversion, deinterlacing, and conversion between PAL and NTSC signals.
- The choice of sampling frequencies to be used in the spatial and temporal directions depends on the frequency content of the underlying signal, the visual thresholds in terms of the spatial and temporal cutoff frequencies, the capture and display device characteristics, and the associated processing, storage, and transmission costs.
- The sampling artifact known as *aliasing* can be minimized—but not eliminated completely—by a proper selection of sampling rates. In specific video sequences (e.g., scenes containing fine striped patterns or moving wheels), some (spatial or temporal) aliasing may still be noticeable.

[9]This function was developed by Jeremy Jacob and can be downloaded from the book web site.

- Converting a video sequence from one format to another is a technologically challenging and economically expensive problem.
- PAL to NTSC conversion involves four main steps: (1) deinterlacing each field in the PAL signal, resulting in a 625-line frame, (2) line rate down-conversion from 625 to 525 within each deinterlaced frame, (3) frame rate up-conversion from 50 to 60 Hz, and (4) splitting each frame into two interlacing fields.

LEARN MORE ABOUT IT

- E. Dubois (in [Bov00a], Chapter 7.2) presents a good summary of video sampling and interpolation.
- Chapters 3 and 4 of [WOZ02] and Chapters 3 and 4 of [Tek95] discuss video sampling and sampling rate conversion using lattice theory as a framework.
- Chapters 16–18 of [Poy03] offer a detailed view of the processes of sampling, filtering, resampling, and reconstruction for 1D, 2D, and 3D signals.
- The booklet by Watkinson [Wat94b] explains standards conversion techniques from an engineering standpoint.
- Two articles by Ackerman [Ack01,Ack02] discuss standards conversion—particularly 3:2 pull-down, color space, and aspect ratio conversion—in more detail.
- Chapter 16 of [Tek95] provides a detailed coverage of standards conversion principles and techniques.
- Chapter 7 of [Jac01] and Chapters 36 and 37 of [Poy03] discuss standards conversion, deinterlacing, and 3:2 pull-down, among many other video processing techniques.

21.8 PROBLEMS

21.1 Explain in your own words the phenomenon known as *aliasing* in video processing. In which situations is aliasing more visible? How can it be prevented/reduced?

21.2 The notion of sampling is usually associated with digitization (analog-to-digital conversion). Give one example of sampling in the analog domain.