Interactive Image Processing: Discrete Cosine Transformation

Peter Plaimer
Johannes Kepler University Linz
dct-tool@tk.jku.at

January 30, 2018

Abstract

This paper looks into methods of teaching Discrete Cosine Transformation (DCT) in the context of images. DCT is a mathematical transformation used in lossy compression of images and video. As such, it is a topic addressed in university curricula for computer science. While it is obvious to approach teaching DCT from a mathematical point of view, this approach is not well-suited for all students alike. We propose and develop a software application that supports students in learning DCT by offering an alternative, visual and interactive means to explore forward an inverse DCT on arbitrary images. We further provide exercises and concrete questions the students can work on, in order to deepen their understanding of DCT based on example images built into that software.

Contents

1 Introduction

Many contemporary and everyday multimedia systems make use of Discrete Cosine Transformation (DCT) at their very core. DCT [ANR74]is the basis for lossy compression of images in JPEG *ITU Recommendation T.81*, of videos in MJPEG and Theora, and, in a modified version, for audio data in MP3 and Vorbis [Dis]. The undergraduate curriculum for computer science at the Johannes Kepler University Linz [cs.17] reflects the relevance of this technique in the respective field. The curriculum comprises multiple classes where DCT is an important topic, for example, lectures and exercises on multimedia systems and a course on digital image processing.

From our first hand experience we know that the level of mathematical skills of students in these classes varies significantly, and so does their success in understanding DCT. Some colleagues struggle to get their heads around the role of coefficients in the frequency domain matrix and their role in the image domain matrix or vice-versa. Informal interviews suggest that most of these students are seeking a visual connection between the coefficients in the two domains. They express the desire to see such connections especially when DCT is applied to blocks of "just" 8 x 8 pixels during JPEG compression and decompression.

We plan to contribute teaching material for classroom demonstration, exercise assignments, and self-study, which aids in understanding DCT in a visual way. Our approach consists of two major parts. The first one is a software application that shows DCT in the context of image processing and JPEG. It offers an image representation of the spatial domain matrix and the frequency domain matrix. Its user interface allows for interactive manipulation of any coefficients in each matrix, respectively the images, using a brush metaphor controlled by the mouse pointer and other means. Changes in one domain trigger instant execution of the DCT, providing instant visual feedback in the other domain's image. The second part is this paper which briefly introduces the topic, explains the application and its user interface, and proposes exercises and questions for deepening the understanding of DCT.

The remainder of this paper offers a short recap of the definition of the DCT and its application in JPEG in section 2. In section 3 we present the software application in the style of a user manual. That section moves on with a list of exercises we propose. It ends with an overview of the application's components and technical requirements. Section 5 evaluates the application based on the planned contribution. We use section 6 to propose topics for future

work based on the application or the general theme of visualizing interactive image processing demos. Section 7 concludes our work.

2 Related work

This section gives a brief overview of the related work. First we look into the the original definition of DCT, and then we highlight the part of the JPEG standard which defines the application and role of DCT in the context of compressing images. As we try to gain understanding of the nature of DCT in a visual way, we are not interested in implementations optimized for speed or memory consumption.

2.1 DCT

According to its original paper [ANR74] from 1974, DCT was defined as a tool in digital image processing for the purpose of feature selection in pattern recognition and for scalar-type Wiener filtering. The goal was to create an orthogonal transformation along with an algorithm that allows its fast computation. After the mathematical definition of the DCT, the paper develops an algorithm which uses the fast Fourier transform (FFT) as the basis for the DCT and its inverse operation, the IDCT. The paper states, that DCT can be used in multiple dimensions, that is, the rows and columns of an image. It does, however, not mention the idea of using DCT for image compression.

2.2 JPEG

JPEG is today's de-facto standard for lossy image compression of real-life photography material in consumer electronics, multimedia applications, and the World Wide Web in general. It uses DCT in it very core.

The 1992 JPEG standard [Itu] defines two classes of encoding and decoding processes, lossless and lossy processes. For us, the latter ones are of interest, as these use DCT for compression and decompression. Figure 1 provides an overview of the encoder employed, for the special case of a single-component image.

The part we take a closer look at in this paper is the *Forward DCT* (denoted FDCT), and its counterpart in the decoder, the *Inverse DCT* (IDCT). In the simplest amongst the DCT-based processes, the *baseline sequential* process, the encoder separates the image into blocks of 8 by 8 pixels and feeds every single into the FDCT. The FDCT's result comprises 64 coefficients representing the frequency domain projection of the spatial domain image block.

In section A.3 the standard defines details for the FDCT. Inside a block, the pixels are referenced by a two-part index. The index components x and y denote the pixel's horizontal and vertical position, from left to right, and from top to bottom, starting with 0 for both components. That section also gives an *informative* specification for the FDCT and IDCT using equations 1, 2 and 3. The indexes u and v serve to address the coefficients in the resulting 8 by 8 matrix holding the 64 coefficients, in the same way as x and y address pixels the image block.

$$\text{FDCT: } S_{vu} = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} s_{yx} \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16} \tag{1}$$

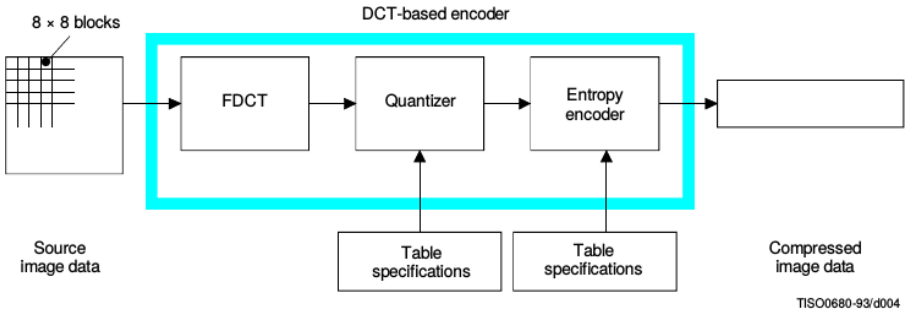


Figure 1: JPEG DCT-based encoder, simplified diagram

$$\text{IDCT: } S_{yx} = \frac{1}{4}\sum_{u=0}^{7}\sum_{v=0}^{7} C_u C_v s_{vu} \cos\frac{(2x+1)u\pi}{16} \cos\frac{(2y+1)v\pi}{16} \tag{2}$$

$$\text{where: } c_u, c_v = \begin{cases} 1/\sqrt{2} & u, v = 0 \\ 1 & \text{otherwise} \end{cases} \tag{3}$$

The relatively simple structures of these equations provide a good opportunity for a coding exercise that aims to implement FDCT and IDCT of one 8 by 8 block extracted from a single image component. This, however, is not in the scope of this paper.

3 Interactive Software Application

This section presents the application in the form of a user manual. This form can be helpful when doing the exercises proposed in section section 4. It furthermore provides a technical description of the application including a description of its components, libraries, source code, and implementation highlights.

3.1 User Manual

This section describes the elements of the user interface, explains their effects, and outlines relevant use cases. The user interface comprises multiple windows of different types. The GUI overview in figure 2 depicts a typical situation during DCT exploration and serves as the reference for the following description.

The top left window contains all settings for the brush, the tool for interactive manipulation of the matrices via image windows. The other two windows in the top row are image windows, each one showing an image representation of a matrix. The left window depicts the spatial domain matrix, and right one the frequency domain matrix. The bottom row has two table windows. A table window shows the components of a matrix as table, and allows value manipulation via standard means of input. Their matrices correspond to those in the image windows above.

Any manipulation of a matrix value via an image window or via a table window will instantly affect the other matrix, and reflect in all windows of all matrices. Manipulation of the spatial domain matrix triggers a forward DCT (FDCT) and changes the frequency domain matrix. Manipulation of the frequency domain matrix triggers an inverse DCT (IDCT) and changes the spatial do main matrix.

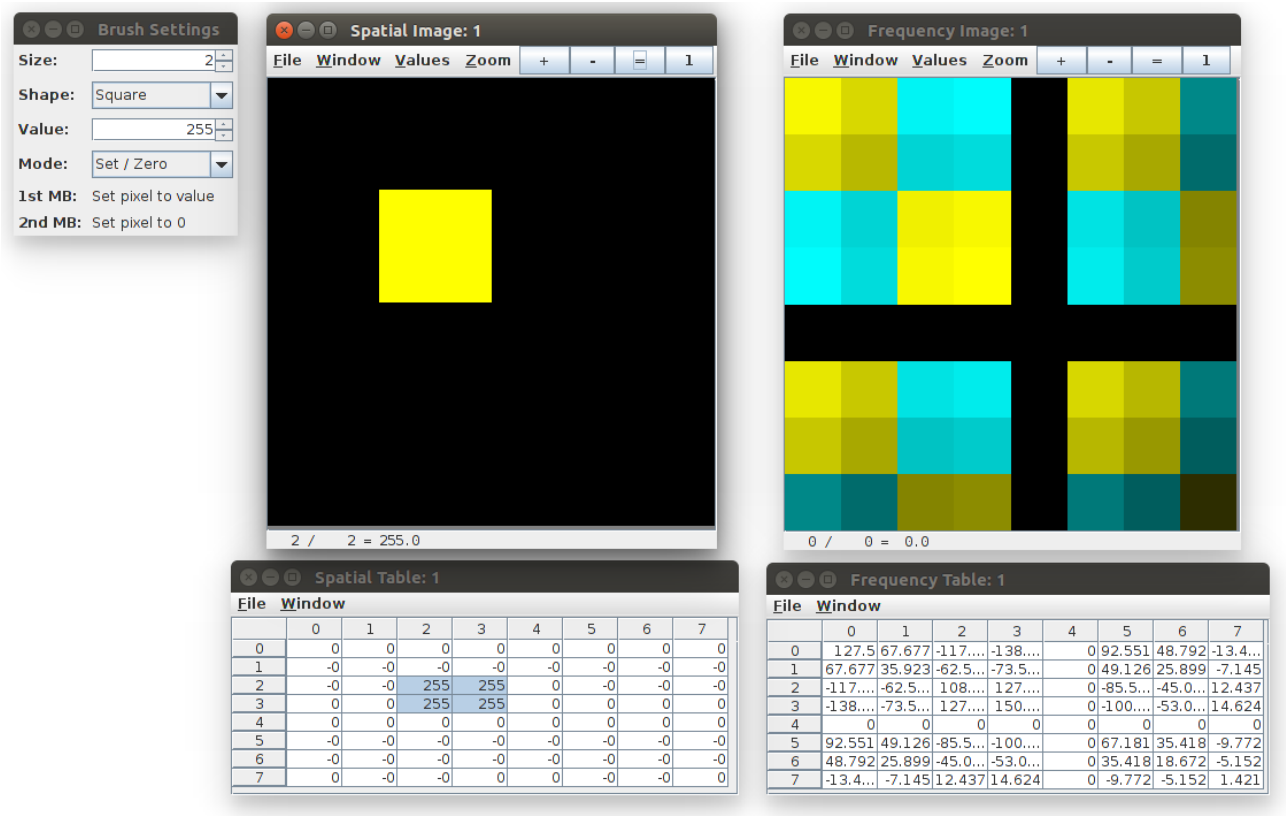


Figure 2: This GUI overview is an example of a typical situation during DCT exploration.

3.1.1 Image Window

Each image window provides the same functions, regardless of the matrix it is displaying. The window uses a viewport showing all or a part of the image representation of a matrix, see section 3.1.2 for details. In this representation, one pixel corresponds to one component of the matrix.

The window features a set of buttons [+], [−], [=], and [1] to control the magnification of the image, in a manner similar to image viewer applications. Spinning the mouse wheel has the effect of zooming in and out, too. The application provides scrollbars to navigate the image, if the magnified view of the image does not fit into the space inside the window. The status bar shows the value of the matrix component beneath the mouse pointer.

Each image window features a *File Menu*, which operates on the respective matrix and its visualization. The most important menu items from top to bottom:

New Matrix creates a new zero matrix. A dialog will query the size. As a consequence of the nature of DCT, this will also create the other matrix anew, and all components will be zero as well.

Open Example Image >> offers a sub menu containing a list of built-in example images for easy opening, with the same behavior like the next menu item.

Open Image File ... opens an image file from the file system via the standard file dialog. Images must be of type JPEG, PNG or TIFF, with 8 bits per pixel grayscale, and of even

height and even width. The image window scales the matrix visualization to fit into the window.

Re-Open Last Image quickly opens the last image opened from the examples list or the file system, discarding all changes to the matrix. This is a very basic way of undoing changes.

Save Matrix As CSV ... saves the matrix components into a comma-separated-values file for external processing.

Save Visible Image As PNG ... saves the whole matrix visualization image to the file system, ignoring the image window's viewport transformation. The image file is always of type PNG, no matter what extension the file name features. It will be an 8 bit grayscale image if the matrix visualization does not use color, so it is ready to be opened with the menu item above. We used this function to create the image files used in figure figure 3

Copy Visible Image To Clipboard copies the whole matrix visualization image into the system clipboard, ignoring the image window's viewport transformation.

The window's menu bar features a Window Menu, containing items to create further image windows and table windows.

3.1.2 Matrix Visualization

DCT is a transformation on a matrix, yielding another matrix. In order to create the image representation of a matrix, the application creates an image of the same size in pixels as the matrix has components. One image pixel corresponds to one matrix component in the same row and column as the pixel.

Each component of both matrices is a real number, thus it has decimal digits and a sign, which may be negative. We can not just use the component value for the brightness of a pixel, since contemporary computer displays can not show negative brightness, and positive brightness is limited to integer values from 0 to 255 inclusive.

The application offers multiple strategies to overcome this crucial limitation, amongst which the user may choose one in each image window's *Value Menu*.

Original mat clipped grey implements simple clamping of values in a grayscale image. Negative component values show as brightness 0 in the pixel, values of 255 and above will have brightness 255.

Auto contrast/brightness grey will uniformly and linearly scale the matrix values such that their resulting brightness exactly fits into the range 0 to 255, if any component exceeds these values. In image processing terms, this roughly corresponds to an automatic contrast and brightness correction.

Log(1+abs(v)) to grey is based on on a method we picked up in the *Digital Image Processing* class. That method employs a logarithmic scale for the conversion to a grayscale image. In order to avoid the impossible calculation of the logarithm of values equal or less than zero, it proceeds in two steps. First, it uses only the absolute value of the component, so the negative values are not lost, but look like their positive counterparts. The second step adds 1 just before the log operation. This way, the intuitive mapping of component

value 0 to pixel brightness 0 is still intact, and higher values yield brighter pixels from the logarithmic scale. Our actual implementation employs automatic contrast adaption after that, in order to use the maximum brightness value 255.

Log(1+v) to +yellow/-cyan is our extension of the aforementioned strategy, additionally encoding the sign of the component value in the color of the pixel. See figure 2 for an example. Positive component values use yellow color, negative values use cyan, zero is black still.

We want to highlight the design process regarding the latter visualization option. We chose these colors with contemporary RGB displays and human perception in mind, and we aimed for high brightness and contrast, while still retaining good color separation for positive and negative signs. The green color component of pixels in these displays is the color with the best reception and resolution in the human eye, yielding high brightness and contrast. We use green for both signs in order to leverage these favorable attributes in all cases. With the same brightness, we add either red or blue, depending on the sign. This has multiple advantages. We further gain brightness and contrast, vital for visual value estimation. We leave a minimum of available brightness and contrast capability of the display unused. We achieve high color separation, due to red and blue being on opposite sides of green in the color spectrum. We add no crosstalk, because we use a separate display sub-pixel per sign.

3.1.3 Brush Window

Interactive manipulation of a matrix via its image representation inside an image window uses the brush metaphor. Pressing the primary mouse button (PMB) while the mouse pointer is inside the image viewport will modify the matrix by applying the brush's primary function, the secondary mouse button (SMB) applies the brush's secondary function, respectively. Dragging the mouse pointer while pressing a mouse button will repeat the function once for every matrix component entered with the mouse pointer, thus enabling a natural way of painting on the matrix.

The Brush Window allows for configuration of a number of brush attributes:

Size determines the size of the brush. Its exact semantics depend on the brush shape.

Shape chooses a shape of the brush and indirectly the strength of its impact on the components affected. The available shapes are:

Square is a filled square with hard edges with length *size*. The mouse pointer determines the center component of the square.

Hard Circle is a disk of diameter *size* with a hard edge. The mouse pointer determines the center component of the disk. The brush function affects all components covered by the brush equally.

Soft Circle is a disk of diameter *size* with a soft edge. The mouse pointer determines the center component of the disk. The brush function fully affects the center component. The strength of the effect on other components decreases with their distance from the center of the brush. It is minimal for components on the edge of the disk.

Value determines the strength of the brush function. Can be negative. Its exact semantics depend on the brush mode.

Mode chooses the pair of functions the brush applies to the matrix:

Set Value sets the affected components value to equal *brush value* when pressing the PMB. The SMB sets the components value 0.

Add Value adds *brush value* to the components value when applied with the PMB, and the SMB subtracts the *brush value* from the component value.

Multiply % interprets *brush value* as percentage and applies the resulting number to the component value using a multiplication for the PMB and a division for the SMB. For example, value 125 represents 125 percent, which equals the number 1.25 in decimal notation. It will therefore increase the component value by one fourth when using the PMB and reduce it by one fifth when applied by the SMB.

3.1.4 Table Window

A table window shows the matrix and its component values in a table. Standard GUI interaction allows for changing single component values via keyboard. When drawing the brush in an image window, table windows of the same matrix change their cell selection such that it highlights the components affected by the brush. A table window's menu bar offers the File Menu and the Window Menu as described in section 3.1.1, minus the image export functions.

3.2 Technical Description

This section provides a list of the application's runtime requirements, the components and libraries it uses, and the tools and environments used to create it. It gives an overview of the most important parts in the application source code and highlights selected elements of the implementation.

3.2.1 Dependencies and Tools

The application depends on a *Java 7 SE* (or later) runtime to execute; we used *OpenJDK* version 7 during development. For the user interface, we rely on the *Swing API*, its programming model, its GUI components, and its extensibility.

We chose *Open Source Computer Vision Library* [Opea] (OpenCV version 2.4.9) to provide for both the DCT operations. In order to use this particular version of OpenCV, we depend on matching Java bindings provided by the *Maven* fragment *nu.pattern.opencv* [Opeb]version 2.4.9-7 by *Pattern-consulting.*

For the Java development tasks we employed the *Eclipse Luna 4.4* IDE, with plug-ins for *Maven* and *git*. We relied on LyX and therefore on LaTeX for creating this paper. All applications used and created run on *Ubuntu 14.04 LTS*, which in turn runs on a standard notebook from 2011.

3.2.2 OpenCV Library

The OpenCV library is obviously overkill for the DCT alone, and its use has a historical background. We created the first version of this application in an effort to kill two birds with one stone while we attended the aforementioned classes in the same semester. The exercises in multimedia systems required us to implement a novel multimedia system of our choice; and the class on digital image processing required us to give a presentation on an application or library in that field, where we chose to look into OpenCV. In order to provide a first hand experience report we started tinkering with the library and quickly realized the potential to use it to create a novel multi application which is at the same time a demo on how to use OpenCV. Both course instructors affirmed the plan.

3.2.3 Source Code

The github repository consists of one eclipse project which includes all sources and images used to build and run the application. The project root contains one directory dedicated to the paper. TODO: not smart. We use Maven as build system mainly to manage dependencies and pull the relevant files on the one hand, and on the other hand it allows us to create a JAR to deploy the application as single file. TODO: true? ... That is, minus the OpenCV library runtime required to be installed on the target machine.

All application code resides in the package `cx.uni.jk.mms.iaip`, which is is therefore the package which all the names of packages and classes in the technical description are relative to. All package, classes and methods feature Javadoc and other comments we deem suitable for anyone diving into the code in order to enhance the application or just out of curiosity.

The application's `main()` method resides in `main.DCT` which on startup creates a `main.MainModel` and a `main.MainController` instance. The latter creates one `image.ImageController` and one `table.MatTableController` for each of the two `mat.MatModels` representing the spatial domain matrix and the frequency domain matrix. It creates as a `brush.BrushController` with a new `brush.brushModel`. Next, the controller for the brush is called to create a `brush.BrushView` and the image controllers are to create one `image.ImageView` each. This `main.MainController` instance places all these views onto the screen in order to create the initial layout comprising the top three windows in figure figure 2. From this description one can see that we follow Swing's separable-model-and-view design [Swi] flavor of MVC in practically all parts of the application.

Our `image.MouseAdapterScaling` class acts as an adapter or proxy converting Swing's window-relative mouse pointer screen coordinates we receive in `image.ImageView`. Based on the image's current zoom factor, it converts screen coordinates into matrix-component coordinates we can use to manipulate the underlying matrix with the brush.

While the `brush` package contains the brush model and configuration GUI, the actual manipulation of the matrix based on the model is part of the `tools.SimpleBrushTool`class. The concrete implementation uses alpha blending of multiple copies of the area under the brush in order to interpolate the brush's effect, especially when using the *soft circle* shape.

The `rectangularJTable` package's sole purpose is to enhance Swing's `JTable` and its `TableModel` with a way to inform listeners about a data change event concerning a rectangular region of cells, instead of just single cells, a range of rows in one column, or all cells. We use this enhancement in order to give visual feedback in matrix table views, by highlighting the matrix components which were affected by the brush.

In the `filter` package one finds the implementations of the available matrix visualizations described in section section 3.1.2. It contains the `MatFilter` interface implemented by all these visualizations, a well as a manager class.

4 Exercises

This section is a collection of exercises covering basic topics of DCT and its application in JPEG. These exercises can also serve as a basis for classroom demonstration or deepening self-studies. In each exercise we introduce the context, give instructions to reconstruct the situation under discussion. A set of images helps to clear out ambiguities. In addition to that, we pose additional questions in order to excite the students' curiosity. The instructions often refer to the Spatial Domain (SD), Spatial Domain Image (SDI), Frequency Domain (FD), and the Frequency Domain Image (FDI), using the respective abbreviations.

4.1 Identity Matrix

The identity matrix clearly shows the matrix transformation nature of the DCT. When loaded into the SD or the FD, the resulting matrix is always the identity matrix.

Exercise: Load the example image identity-8 (figure 3g) into either one matrix.

Additional questions: Are there any other simple matrices showing visual similarities in the SDI and FDI?

4.2 JPEG DCT

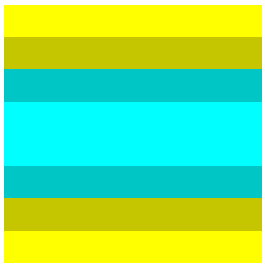
JPEG compression feeds 8 by 8 blocks of pixels into an FDCT to obtain the FD matrix. Similarly, the decompression feeds an 8 by 8 matrix into its IDCT in order to reconstruct the SDI. Visual explanations of these step often use a diagram showing a grid of SDI patterns that every single FD component adds to the image [Fil; Dis]. This application allows to reproduce the elements of these diagrams interactively. Furthermore, one can study the superposition of the patterns from multiple coefficients.

Exercise: Configure a size 1 square brush with value 255 in *set/zero* mode. Set SDI visualization to *yellow/cyan*. Apply the PMB on single components of the FDI, use SMB to remove the value.

Additional questions: What is the effect of FD component in row 0, column 0 on the SDI? How do FD components n/0 and 0/n relate to each other (see figures figure 3a and figure 3b for n=2)? How and why is the SDI for the combination of FDI n/0 and 0/n different to n/n alone (see figuresfigure 3c and figure 3d for n=2)?

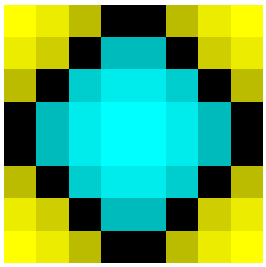
4.3 JPEG Quantization Loss / Low-Pass Filter

JPEG compression is lossy due to the quantization step after the FDCT. This step basically discards high-frequency components with small absolute values, in other terms, it is a low-pass filter. When losing too much information, the decompressed image shows blurred edges, blocky colors, ringing, and other types of artifacts. We can simulate this loss of information in quantization and immediately see its effect in the SDI.

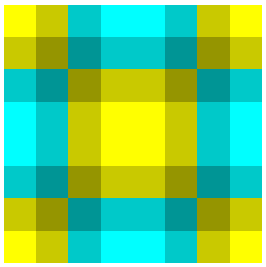


(a) JPEG DCT: SDI for FD component 0/2

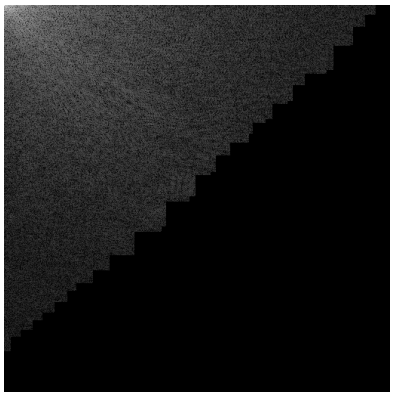
(b) JPEG DCT: SDI for FD component 2/0



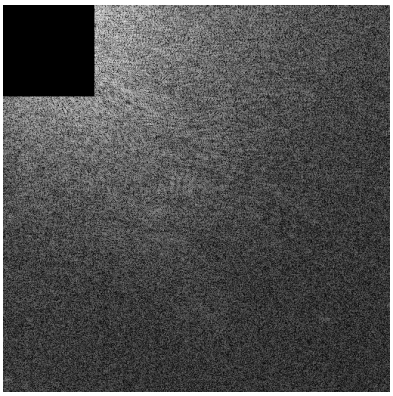
(c) JPEG DCT: SDI for FD component 0/2 and 2/0 combined



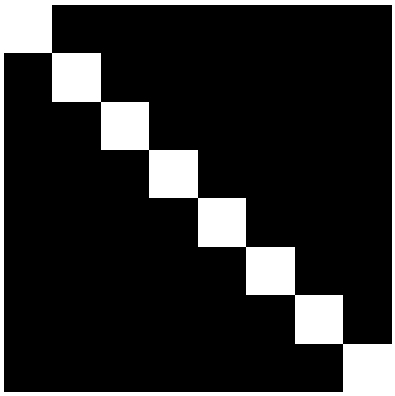
(d) JPEG DCT: SDI for FD component 2/2



(e) Low pass: 60 % of Lenna's FDI deleted diagonally

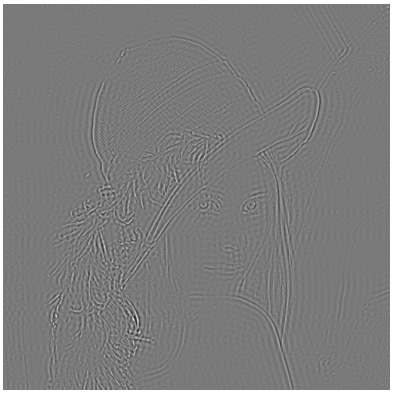


(f) High pass: Lenna's FDI with a block of low-frequency components deleted

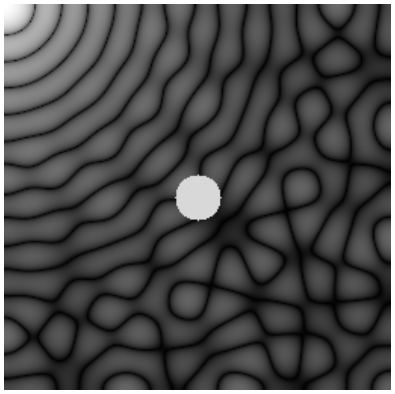


(g) The Identity matrix looks the same in SDI and FDI

(h) Low pass: SDI of Lenna with 90 % FD deleted diagonally



(i) High pass: SDI shows highlighted edges



(j) Self-Similarity: This SDI looks very similar to its own FDI

Figure 3: SDI and FDI displaying intermediate and final steps from the exercises

Exercise: In an SDI window, open example image lena-512 in an SDI window, *zoom to original size*, choose visualization *original*. Configure a size 80 square brush and choose mode *set/zero*. In an FDI set visualization *log-gray*. In that FDI, start applying the brush in the bottom right corner. Press and hold SMB and start sweeping in a diagonal direction from bottom-left to top-right and back. While still sweeping, slowly advance to the top-left corner, deleting lower and lower frequencies (figure figure 3e). You are most likely to see no detrimental effect in the SDI up until the point where the bottom-right half of the FDI is cleared. Keep sweeping and watch closely as the first artifacts become visible (figure figure 3h).

Additional questions: What is similar or different to the effect seen before, if one starts clearing coefficients at the bottom edge of the FDI, sweeping from left to right and back, slowly moving to the top?

4.4 Edge detection / High-Pass Filter

Edges in SDI correspond to high amplitudes in high-frequency coefficients. One can use this property to find edges by manipulating the FDI mimicking a high-pass filter.

Exercise: In an SDI window, open example image lena-512 in an SDI window, choose visualization *auto-contrast/brightness*. Configure a size 240 hard circle brush and choose mode *set/zero*. In an FDI apply the brush by SMB at component 0/0 (figures figure 3f). The SDI immediately shows highlighted edges (figure figure 3i).

Additional questions: How does the brush size influence the kind of edges this method finds? When clearing the coefficient 0/0 only, how does it change the character of the filter operation?

4.5 Large-Scale Self-Similarity

Exercise: Create a new matrix of size 256 by 256. Configure brush size 32, *hard circle* shape, value 255, mode *add/subtract*. Choose *log-gray* matrix visualization for SDI and FDI. Apply the brush in SDI at 128/128 using PMB. This produces a pattern of dozens of wiggly lines and loops in the FDI. Now apply the brush in the FDI at 128/128. Now SDI (figure figure 3j) and FDI show striking similarities. Question: What are the major reasons for this result?

Additional questions: What patterns do you expect when applying the other brush types instead? Or different brush sizes?

5 Evaluation

We have created an interactive application presenting DCT in an image processing context. During its design and implementation phases, we faced unforeseen challenges. For example, the need for alpha blending in order to apply a soft brush to the matrix, and incompatibilities of image file processing libraries in the Java API with the similar methods in the OpenCV library. Not only have we overcome these technical challenges, we have also grown with them. Above that, we designed and implemented some unique GUI features. For example, using a brush on a matrix interactively, providing visual feedback of changes in a table, and the various matrix visualization strategies. We consider the yellow/cyan matrix visualization to be a novel approach in understanding matrices.

In the current state the application is suitable for classroom demonstration, as well as for exercise assignment and self-studies of the DCT. In order to aid all three scenarios, we provide

a set of exercises covering a range of basic topics of DCT and its application in JPEG. These exercises demonstrate that the application satisfies the needs of our fellow students. While thus list is a pool for a course instructor to pick from, it is also not exhaustive.

An extensive user manual explains all functions and options, and for some items gives insight into their rationale.

For the technically-oriented, we offer an overview of the application components, the libraries and tools involved, we skim all parts of the implementation and take a closer look at some interesting details. With this information, one will be able to build the application from the source code we release in its entirety.

Finally, we provide the source codes of this paper, too, so that anyone extending the application can extend this document as well, for the sake of consistency.

6 Future Work

While we deem the application feature complete regarding the initial plan, we present an informal list of ideas that did not make it into the application, but can be part of anyones future work. Allow switching to FFT (additional phase-image) or wavelet transformation (JPEG 2000 [Jpe]). Replace calling the OpenCV DCT with your own implementation, or with an interface, so that many students can implement the DCT algorithm in a single method, as part of an exercise assignment. Enable importing CSV files. Open color images and interpret red an blue channel with different sign, according to the matrix visualization chosen. Add a simple way to copy the SD matrix over FD and vice-versa. Make the brush border visible while hovering over the image representations. Apply DCT or any of the other transforms to audio data, with graphical representations of amplitude over time and frequency spectrum over time, and similar interactive manipulation of these representations.

7 Conclusion

We have successfully created the interactive application we planned in the beginning. We are confident that it aids in teaching and in understanding DCT in a visual way, not only by supporting the exercises we deliver in this paper, but also by offering the freedom to load arbitrary images and manipulate these using the brush. On top of that, we encourage anyone who is interested to extend this set of learning material in any useful way. Therefore we release the source code and images of the application, and the source code and images of this paper, under the GPL 3.0 license on github: https://github.com/jkutk/iaip-dct

References

[ANR74] N. Ahmed, T. Natarajan, and K. R. Rao. "Discrete Cosine Transform". In: *IEEE Transactions on Computers* C-23.1 (Jan. 1974), pp. 90–93. ISSN: 0018-9340. DOI: 10.1109/T-C.1974.223784.

[cs.17] cs.jku.at. *CURRICULUM GUIDE BACHELOR IN COMPUTER SCIENCE. valid as of WS 2017/18*. 2017. URL: http://cs.jku.at/teaching/curricula/2017/CS_Bachelor_Guide_en.pdf (visited on 01/23/2018).

[Dis] *Discrete cosine transform*. In: *Wikipedia*. Page Version ID: 817587251. Dec. 29, 2017. URL: https://en.wikipedia.org/w/index.php?title=Discrete_cosine_transform&oldid=817587251 (visited on 01/28/2018).

[Fil] *File:Dctjpeg.png*. In: *Wikipedia*. URL: https://en.wikipedia.org/wiki/File:Dctjpeg.png (visited on 01/28/2018).

[Itu] *ITU Recommendation T.81 : Information technology - Digital compression and coding of continuous-tone still images - Requirements and guidelines*. Sept. 1992. URL: https://www.itu.int/rec/T-REC-T.81-199209-I/en (visited on 01/28/2018).

[Jpe] *JPEG 2000*. In: *Wikipedia*. Page Version ID: 821758115. Jan. 22, 2018. URL: https://en.wikipedia.org/w/index.php?title=JPEG_2000&oldid=821758115 (visited on 01/28/2018).

[Opea] *OpenCV library*. 2017. URL: https://opencv.org/ (visited on 01/21/2018).

[Opeb] *opencv: OpenCV Java bindings packaged with native libraries, seamlessly delivered as a turn-key Maven dependency*. original-date: 2014-05-16T12:37:05Z. Dec. 30, 2017. URL: https://github.com/PatternConsulting/opencv (visited on 01/29/2018).

[Swi] *A Swing Architecture Overview*. URL: http://www.oracle.com/technetwork/java/architecture-142923.html (visited on 09/01/2016).