David (Yang) Jing
301287553

CMPT 365 – Self Proposed Project Report

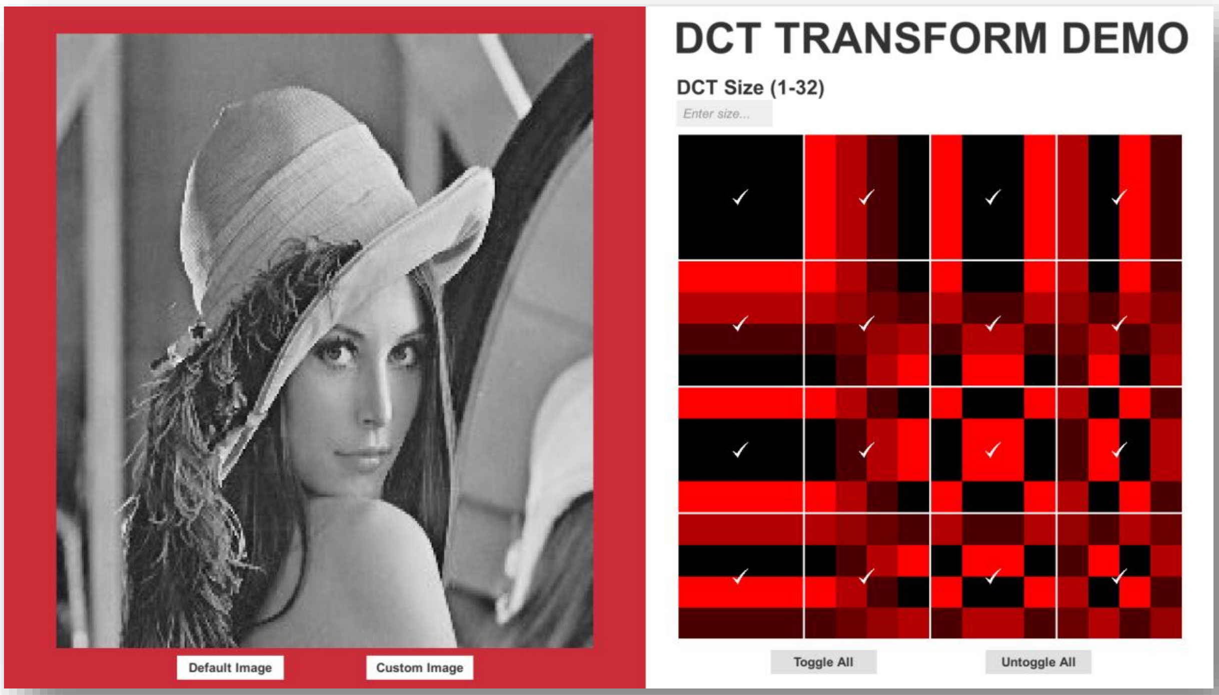


Figure 1 - My interactive DCT transform demo built in the Unity Game Engine and with C#.

Project Files

https://vault.sfu.ca/index.php/s/GUjbU595JcnjWfd

The Unity Game Engine source files (coded in C#) and builds can be access via the link above.

Abstract

Using the Unity Game Engine, I created an interactive program in which users can explore and observe the effects of DCT components on a greyscale image. In this program, users can toggle any number of individual DCT components in a wide range of N-point DCTs (right of Figure 1). Additionally, the contributions of each DCT component is visually shown in their respective togglable block. Upon toggling or untoggling a DCT component, the effects of the removal or inclusion of that DCT component is immediately reflected in the presented image (left of Figure 1). Defaulting to the Lena image, the user has the option to load other PNG or JPG files for further experimentation.

Introduction

Inspired by CMPT 365's Project 2's Question 2, and slides 38 and 44 of 6-LossyCompression.ppt, I wanted to examine and experiment with the effects and contributions of DCT components in a wide range of N-point DCTs—not just 2, 4, and 8; hence, I created the DCT transform demo as shown in Figure 1. I choose to utilize the Unity Game Engine because I wanted to maximize interactive options and create more interesting UI designs—something that Python cannot easily provide.

Solution

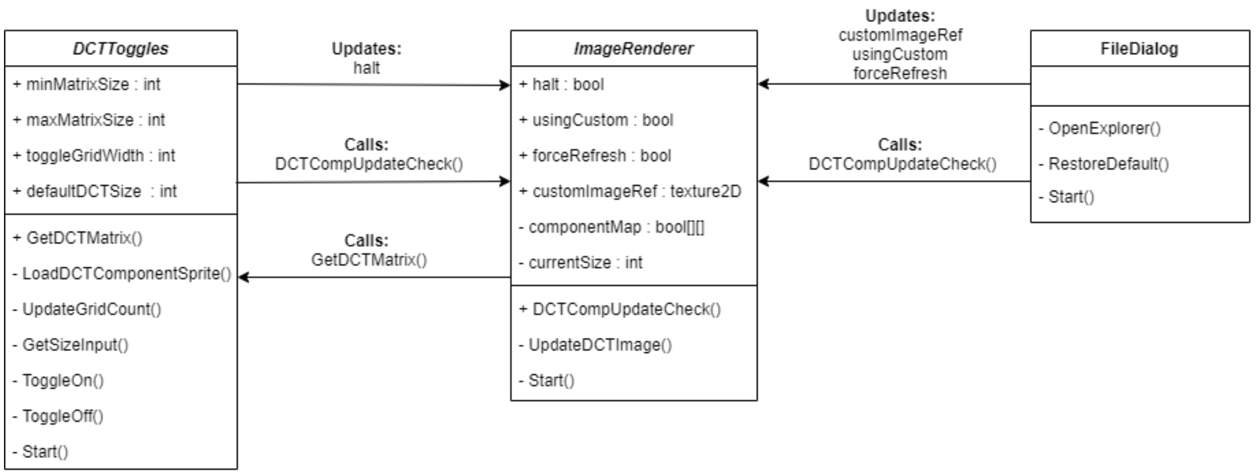


Figure 2 - Class diagram of the underlying system used. Note that GameObjects relating solely to on-screen items are excluded from the diagram.

Three main classes are used in this project with each controlling over a major function in the system:

1. **DCTToggles** manages the user input, the N-by-N DCT toggles, and the “Toggle All” and “Untoggle All” buttons. Upon the user’s input, the class creates and displays N-by-N toggles with each displaying their respective DCT component image. The displayed component images are calculated upon initialization and are drawn pixel-by-pixel. With an N-by-N zero matrix, by placing the maximum DC value of that N-point DCT in the component that is to be imaged, the corresponding DCT component image can be generated.

2. The **ImageRenderer** communicates with FileDialog and DCTToggles to refresh the displayed image. When refreshing the image, the class would generate DCT transforms of each N-by-N pixel block from a reference image and make the untoggled DCT components zero. After zeroing the corresponding components, the DCT transform is converted back to an image and is then drawn pixel-by-pixel onto the screen.

3. **FileDialog** manages the “Default Image” and “Custom Image” buttons. When the user clicks on the “Custom Image” button, a file dialog box prompts the user to select a PNG or JPG file. If the file is valid, the image would be loaded and converted to greyscale pixel-by-pixel. The greyscale image is then saved to memory and used as a reference by ImageRenderer. The “Default Image” button reverts the displayed and referenced image to Lena.

Sample Results

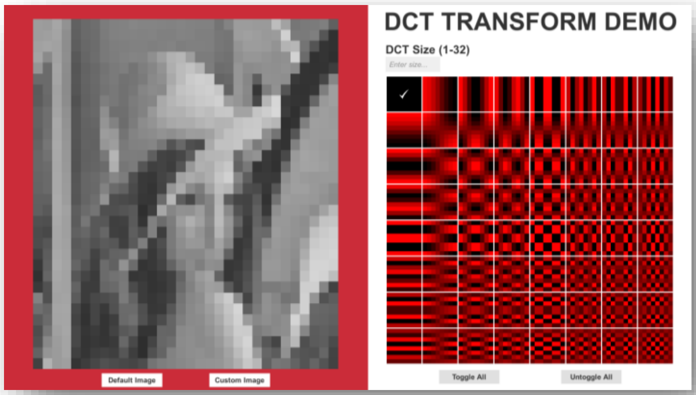


Figure 3 – 8-Point DCT on the default Lena image. With everything but the DC component untoggled, the image's resolution is effectively reduced by 8.

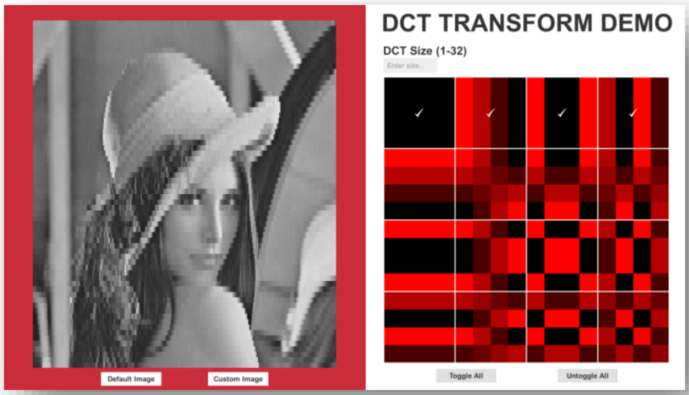


Figure 4 – 4-Point DCT on the default Lena image. With only the "vertical stripe" components toggled, the image appears to have disjointed "horizontal stripes".

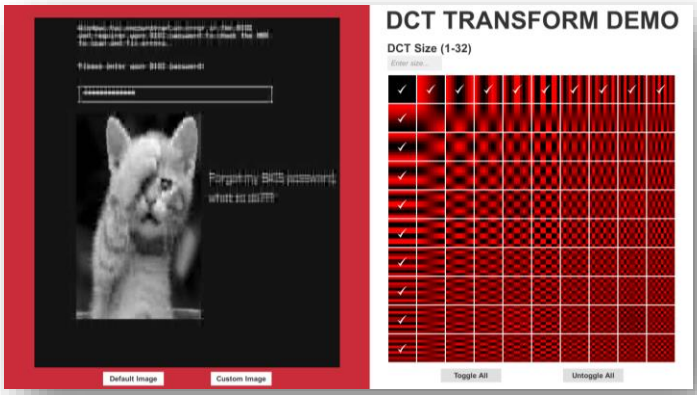
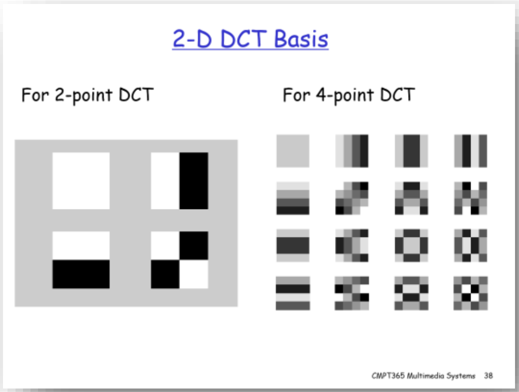


Figure 5 – 10-Point DCT on the BIOS image. With only the "vertical stripe" and "horizontal stripe" components toggled, details in the image can be generally identified.

References

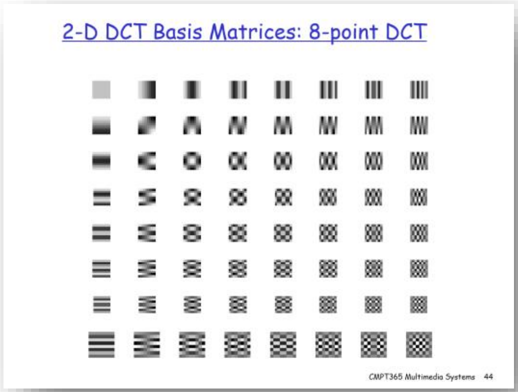


2-D DCT Basis

For 2-point DCT

For 4-point DCT

CMPT365 Multimedia Systems 38



2-D DCT Basis Matrices: 8-point DCT

CMPT365 Multimedia Systems 44

Figure 6 – References from 6-LossyCompression.ppt (slide 38 and 44) used in the creation of the DCT transform demo.

By referencing the images shown in Figure 6, I was able to verify my results; this proved to be extremely valuable for debugging my DCT transform demo. By referencing the slides, I was able to realize that pixels are indexed bottom-up instead of the top-down for Unity’s Texture2D—which caused bugs in the program. Also, I made use of the RGB-to-greyscale conversion ratio presented in slide 16 of 3-MediaRepresentation-Image.ppt for converting inputted images to greyscale.