Video Encoder

Michael Clifford, Patrick Dillon, June Hua, Frank Tranghese
Group 3

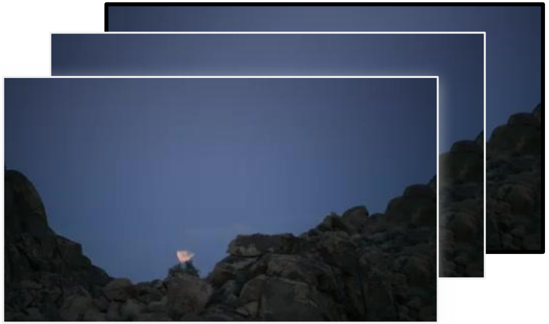
EC 504 - Advanced Data Structures Final Project
December 2017

BU Department of Electrical & Computer Engineering

Video Compression

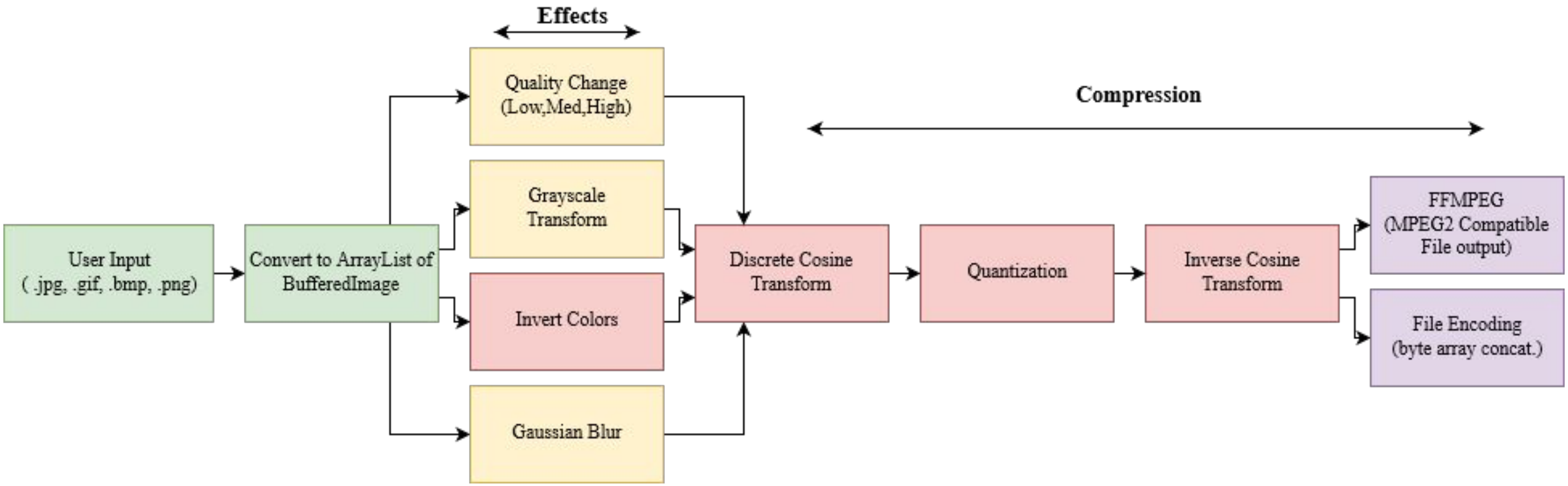
- The process of concatenating still images into moving video.
- For an image of 1920 x 1800 pixels with 24 bits per pixel, played at 30 frames per second: Over 2000 Mbps!
- Video compression algorithms increasingly necessary to provide services like YouTube, Skype, and other video sharing APIs.

Problem Statement

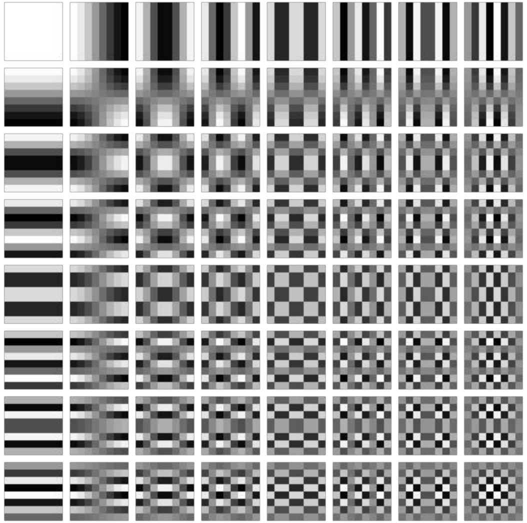


- Encode up to 100 JPEG images (of the same dimensions) into one encoded file in under 5 minutes.
- File size no more than the sum of the individual images
- Ability to play back at least 10 images per second.

Overall Design



Discrete Cosine Transform



8 x 8 DCT on JPG

https://en.wikipedia.org/wiki/Discrete_cosine_transform#/media/File:DCT-8x8.png

$$D(i,j) = \frac{1}{\sqrt{2}} C(i)C(j) \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} I(x,y)\cos\left[\frac{(2x+1)i\pi}{2N}\right]\left[\frac{(2x+1)j\pi}{2N}\right]$$

$$\text{Where: } C(u) = \begin{cases} \frac{1}{\sqrt{2}} & if \;\; u = 0 \\ 1 & if \;\; u > 0 \end{cases},$$

Discrete Cosine Transform Formula

Khedr and Abdelrazek. **Image Compression using DCT upon Various Quantization**. *International Journal of Computer Applications (0975–8887)Vol. 137 No.1, March 2016*

Raid, AM et al. **Jpeg Image Compression Using Discrete Cosine Transform - A Survey**. *IJCSES. Vol.5, No.2, April 2014*

Quantization

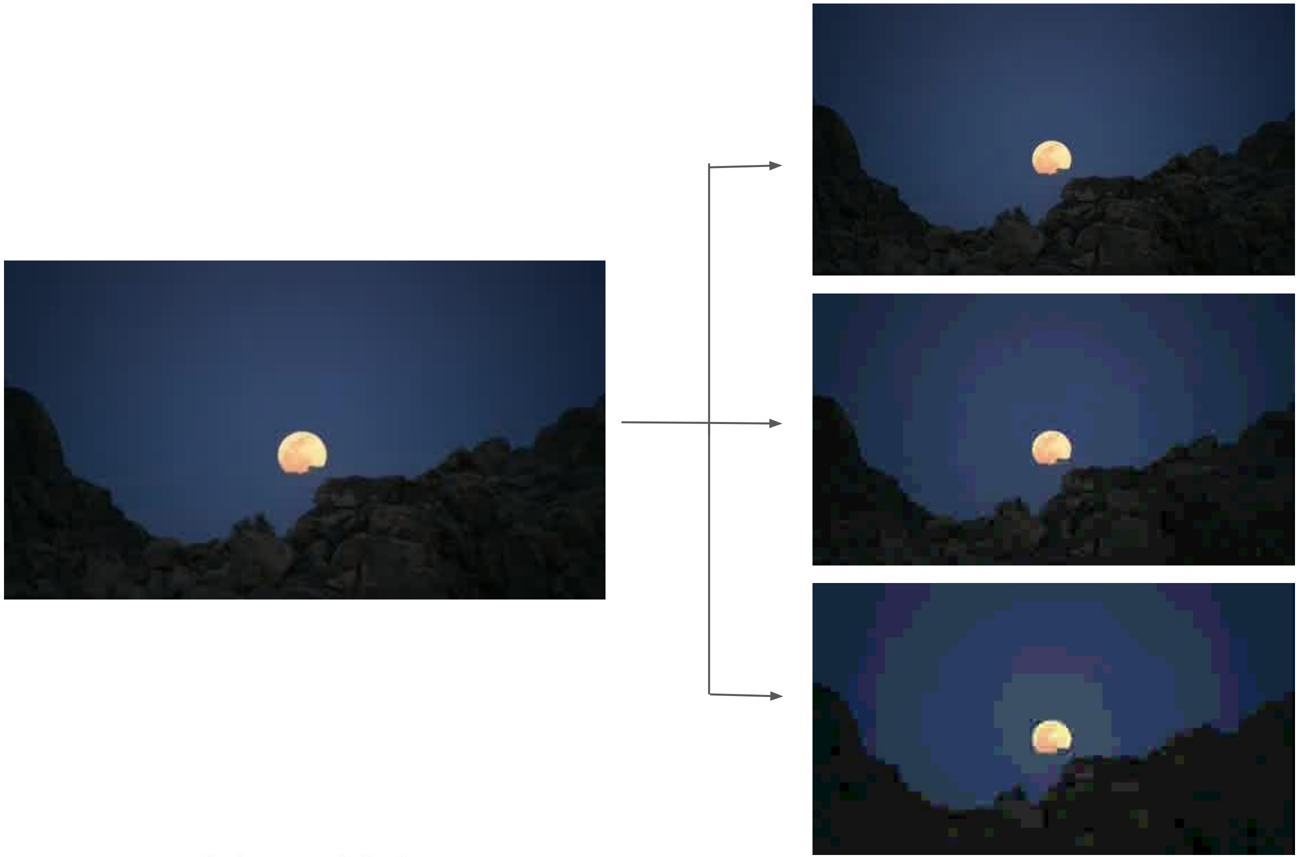
$$Q_{50} = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Quantization Matrix

- JPEG standard quantization matrix
- Vary quantization matrix scale to change quality

Khedr and Abdelrazek. **Image Compression using DCT upon Various Quantization**. *International Journal of Computer Applications (0975–8887) Vol. 137 No.1, March 2016*

Quantization Scale Factor

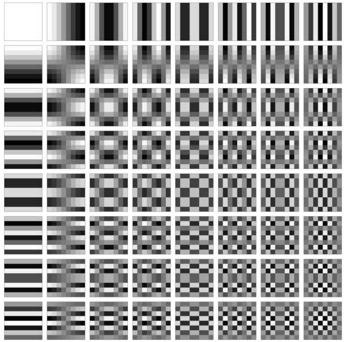


QScale = 1

QScale = 5

QScale = 10

Inverse Discrete Cosine Transform



IDCT $\longrightarrow$

$$f(x,y) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u)C(v)F(u,v) \cos\left[\frac{\pi(2x+1)u}{2N}\right] \cos\left[\frac{\pi(2y+1)v}{2N}\right]$$

for $x = 0,\dots,N-1$ and $y = 0,\dots,N-1$ where $N = 8$

Raid, Khedr, El-dosuky, and Ahmed.. **Jpeg Image Compression using Discrete Cosine Transform - A Survey**. *International Journal of Computer Science & Engineering Survey (IJCSES) Vol. 5 No.2, April 2014*

DCT and Quantization Time Complexity

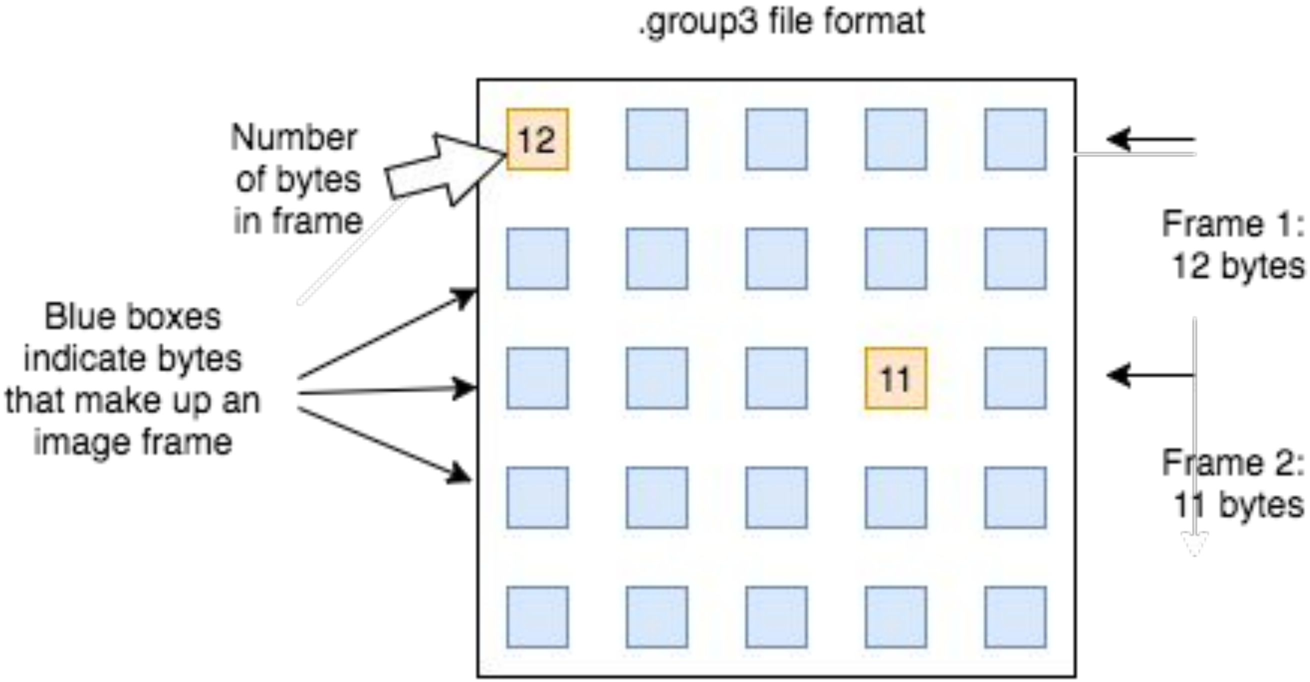
$\Theta(n^2)$, where n = number of pixels in a single image

For m images,

- $m \ll n \Rightarrow$ Runtime: $\Theta(n^2)$
- $m \sim n \Rightarrow$ Runtime: $\Theta(mn^2)$
- $m \gg n \Rightarrow$ Runtime: $\Theta(m)$

Bitstream Encoding/Decoding: the .group3 filetype

After compression, the bytes of each image are appended into a .group3 file.

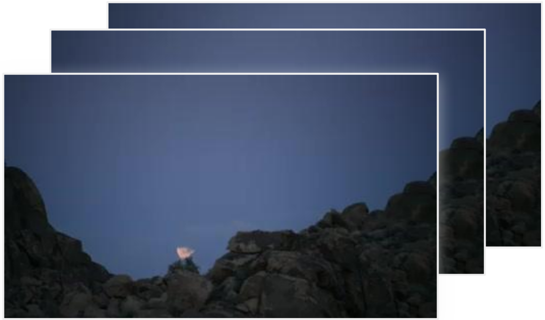


Features

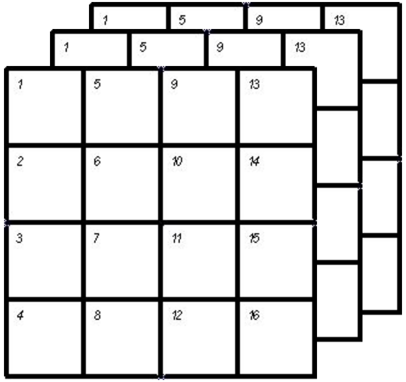
- Takes .jpeg, .bmp, .png, and .gif (static)
- User can change quality of the output video at the cost of file size.
- Three real-time video transforms
  - Grayscale
  - Color Inversion
  - Gaussian Blur
- Option for MPEG-2 compliant output.

File Input Types and BufferedImage

- Initial images converted to an ArrayList<BufferedImage>
- BufferedImage is a built in Java.awt type that supports JPEG, BMP, PNG, and GIF file types.
- All code designed around this data structure choice.



Input Images



ArrayList<BufferedImage>

Quality Change

- Uses Java built-in convolution operator for BufferedImage call ConvolveOp.
- Convolves images with kernels that average groups of nearby pixels to the same value.
- Making neighboring colors similar reduces the amount of information and increased the compression of DCT and quantization.

$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$
$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$
$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$
$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$

4 x 4 Averaging Kernel

Real Time Transforms

Original Image

Inverse

Grayscale

Gaussian Blur

Gaussian Blur

- Similar to Quality changes, uses Java's built in ConvolveOp to convolve a kernel of values according to the 2D Gaussian Equation with original images.
- Values are normalized so the entire kernel adds to 1, which ensures the intensity values of the images are not changed.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

2D Gaussian Formula

Shapiro, L. G. & Stockman, G. C: **"Computer Vision"**, page 137, 150. Prentice Hall, 2001

Color Inversion

- Uses BufferedImage built in functions .getRGB and .setRGB to access each pixel's color RGB values and change them to their inverted values, as seen below.

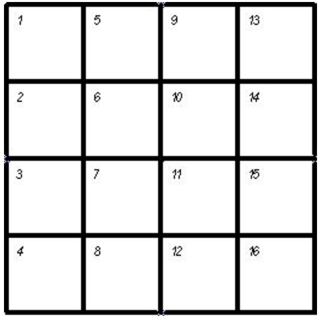
```
Inv R = 255 - R

Inv G = 255 - G

Inv B = 255 - B
```

Formula for Inverting RGB values

Grayscale

- Uses Java's built in Graphics class to take the initial BufferedImage (RGB type) and redraw it into another BufferedImage of Gray type, which causes all color values to be lost.

Original



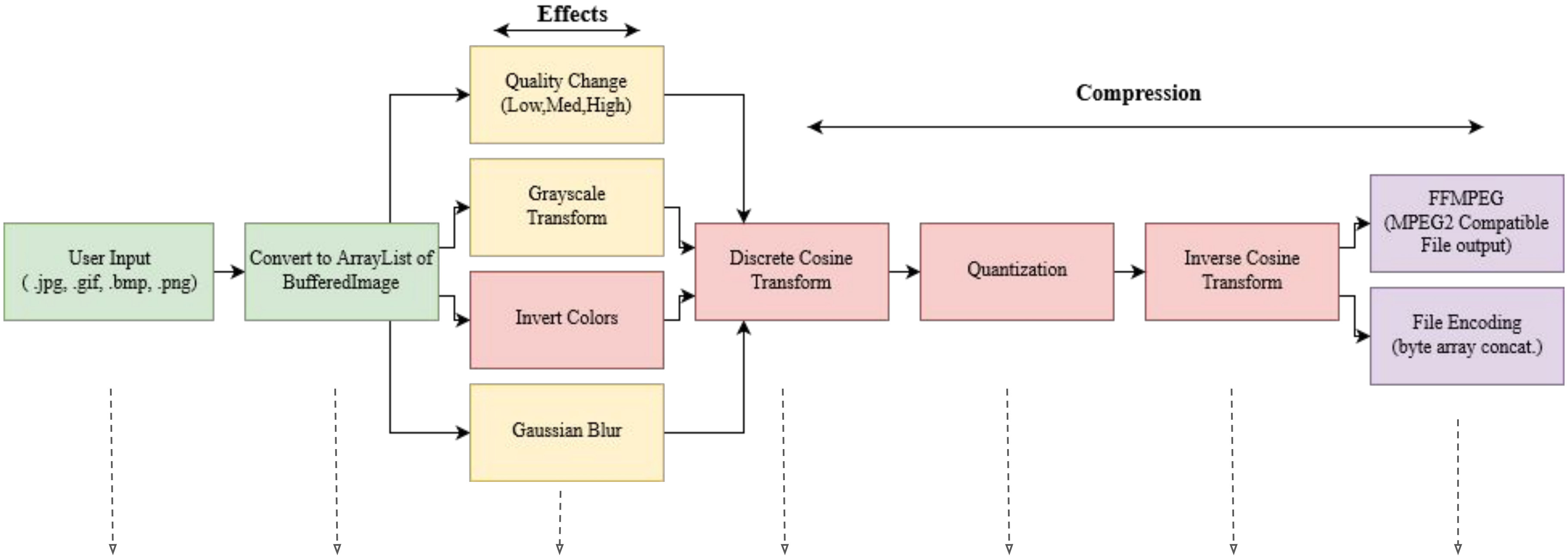
BufferedImage (gray)

Redrawn Output

MPEG-2 Compliant Output

Our program employs FFMPEG to convert preprocessed images into .mpeg format as the final step, after the input image set has been compressed.

Overall Time Complexity



$$\Theta(mn) + \Theta(mn) + \Theta(mn^2) + \Theta(mn^2) + \Theta(mn^2) + \Theta(mn^2) + \Theta(mn)$$

Overall: ~ $\Theta(mn^2)$

And now an example of our program in action...