**PROJECT**

**VIDEO2GIF**

SONG, Chen

301324024

ZHOU, Jiahui

301215554

Instructed by

CHEN, Xiaochuan

**­­Abstract**

Converting video type files to GIF image file is worth, because GIF has been widely used in Internet in order to denote product. This report introduces our project that can convert most video type files to GIF image file.

**1. Introduction**

This report is going to demonstrate the detail about GIF encoder and how our video2gif project works. GIF format and basic encoding algorithm will be introduced first. Then, the features of our video2gif project will be listed. In the end, the steps to use this project will be shown. The features we added to this project is text adding tool and resolution selection tool, which will be indicated after encoding algorithms being introduced.

**2. Background**

**Language:** Java

**Front-End:** JavaFX

**Back-End:** OpenCV

**Framework:** MVC Framework (Model, View, Controller)

**3. Encoding Steps & Algorithm**

**3.1 Get Video Frames from Controller**

1. **GIFEncoder(Mat[] image):** A new gifencoder object will be created with the input video information. Video information is a 3D matrix which contains the pixel information of each frame.
2. **\*.encoder(short fps):** This is the major encoding function which writes GIF data to buffer. The following is the steps of writing GIF data.

a. Write Header: ‘GIF89a’***(See 3.2)***

b. Write Screen Descriptor ***(See 3.3)***

c. Write Graphic Control Extension ***(See 3.4)***

d. Write Frame Data (include Color Table & LZW compression algorithm). ***(See 3.5, 3.6, 3.7)***

e. Write a trailer to show the end. ***(See 3.8)***

**3.2 Header**

‘G’ ‘I’ ‘F’

‘8’ ‘9’ ‘a’

Signature

Version

3 bytes

3 bytes

*(Figure 3.2: Header)*

**Signature**: Identify this is GIF data stream. Therefore, here gets the fixed value “GIF”.

**Version**: Identify the version number of this GIF data stream, which is “89a”.

This Header declares the type and the version of the file. In our project, it is ‘GIF89a’.

**Function:** writeHeader(); (in file *“GIFEncoder.java”*)

**3.3 Screen Descriptor**

Screen Width

Screen Height

Unsigned

Unsigned

0

Background Color

Byte

0

Pixel Aspect Ratio

Byte

Bit

0

111

0

000

Packed

*(Figure 3.3: Screen Descriptor)*

*Packed:*

*1. Global Color Table disable: 0 (1 bit)*

*2. Color Resolution: 111 (3 bits)*

*3. Sort disable: 0 (1 bit)*

*4. Global Color Table Size: 000 (3 bits)*

**Screen Width**: Indicate the screen width of the displaying device. The numbers are represented in pixel. The type of the data is “unsigned”.

**Screen Height**: Indicate the screen height of the displaying device. The numbers are represented in pixel. The type of the data is “unsigned”.

**Packed**: Packed includes the information about Global Color Table Ability, Color Resolution information, Sort Ability, Global Color Table Size.

1. Global Color Table Ability: Zero means Disabled, and One means Enabled. This information only need one bit. In this project, we did not use Global Color Table.
2. Color Resolution: Indicates the number of colors used in the graphic. This takes 3 bits. In our case, we used 256 colors.
3. Sort Ability: Indicate whether the Global Color Table ordered according to the color frequency. The color with the highest frequency at the first position. This takes 1 bit. In our case, because there is no Global Color Table, sort is disabled.
4. Global Color Table Size: Indicate the size of Global Color Table. This takes 3 bits. Because Global Color Table has not been used in our project, the size is 0x0.

**Background Color**: Indicate the color of the pixels which are not covered by image on the screen. If Global Color Table is disabled, this have to be 0, which means it has been ignored. In our case, because Global Color Table has been disabled, the value has been set to 0.

**Pixel Aspect Ratio**: This is used to calculate the approximation of the aspect ratio of the pixel in original image. Zero means there is no aspect ratio information, which has been used in our project.

This is the screen descriptor that indicates the screen detail. This includes Screen Size, Global Color Table information, Background Color information, and Pixel Aspect Ratio information.

**Function:** writeScreenDescriptor(); (in file *“GIFEncoder.java”*)

**3.4 Graphic Control Extension**

0x21

Extension Introducer

Byte

0xf9

Graphic Control Label

0x04

Block Size

Packed

Delay Time (fps)

0x00

Color Index

0x00

Terminator

Byte

Byte

Byte

Byte

Bit

Unsigned

000

000

0

0

Delay Time (fps)

Unsigned

*(Figure 3.4: Graphics Control Extension (for controlling the fps))*

*Packed:*

1. *Reserved: 000 (3 bits)*
2. *Disposal Method: 000 (3 bits)*
3. *User Input Flag: 0 (1 bit)*
4. *Color Flag: 0 (1 bit)*

**Extension Introducer**: Indicate that the beginning of the Graphic Control Extension. “0x21” is the fixed signature for the introducer.

**Graphic Control Label**: Indicate that this block is the Graphic Control Extension. “0xf9” is the fixed value for this label.

**Block Size**: Indicate the number of bytes in the block but without block terminator. Here contains the fixed value “0x04”.

**Packed**:

1. Reserve: Reserved is always zero(0x0). This takes 3 bits.
2. Disposal Method: If 0, there will be no special disposal method. If 1, this means do not do any disposal method. If 2, the image area must be restored to background color. If 3, the image area must be restored to the area with the previous color. If 4 to 7, this means you can self-define a method. This takes 3 bits. In our case, we only use the default disposal method.
3. User Input Flag: If 1, this means process will continue after user finishes his input. If 0, there will be no user input expected. This takes 1 bit. In our case, we do not expect user input.
4. Transparency: Indicate the transparency is given or not. This takes 1 bit. In our case, we do not use transparency.

**Delay Time**: (?) Indicate the number of 1/100 of a second to wait before continuing with process. The clock start immediately after the image is rendered.

**Transparency Index**: (?) This byte will only be used if Transparency Flag in Packed is set to 1.

**Block Terminator**: Indicate here is the end of the Graphic Control Extension block. It takes the fixed value “0x00”.

**Function**: writeGraphicsControlExtension(short delayInMilliSeconds); (in file *“GIFEncoder.java”*)

**3.5 Image Descriptor**

0x2C

Image Left Postion

Separator

Byte

Image Top Postion

Unsigned

Unsigned

Image Width

Image Height

Unsigned

Unsigned

Packed

Bit

1

0

0

00

111

*(Figure 3.5: Image Descriptor)*

*Packed:*

*1. Local Color Table enable: 1 (1 bit)*

*2. Progress scanning: 0 (1 bit)*

*3. Sort disable: 0 (1 bit)*

*4. Reserved: 00 (2 bits)*

*5. Local Color Table size: 111 (3 bits)*

**Separator**: Indicate this block is Image Descriptor. It takes the fixed value “0x2C”.

**Image Left Postion**: Indicate the leftmost column on the screen, in pixel.

**Image Top Postion**: Indicate the topmost row on the screen, in pixel.

**Image Width:** Width of the image, in pixel.

**Image Height**: Height of the image, in pixel.

**Local Color Table Ability**:

1. Local Color Table Flag: Indicate whether Local Color Table has been used. Zero means Local Color Table is disabled, and One means Local Color Table is enabled. In our case, we are used Local Color Table. Therefore, it is one. This takes 1 bit.
2. Interlace Flag: Indicate whether interlace scanning has been used. If 0, it means progressive scanning has been used. If 1, interlace scanning has been used. In our project, we used progressive scanning. This takes 1 bit.
3. Sort Flag: Indicate whether the Local Color Table ordered according to the color frequency. The color with the highest frequency at the first position. This takes 1 bit. In our case, because there is no Local Color Table, sort is disabled.
4. Local Color Table Size: Indicate the size of Local Color Table. This takes 3 bits. Because Local Color Table has not been used in our project, the size is 0x0.

**Function**: writeImageDescriptor(Mat inputimage); (in file *“GIFEncoder.java”*)

**3.6 Local Color Table**

Red 0

Green 0

Blue 0

Red 1

Green 1

Blue 1

.

.

.

.

.

.

Red 255

Green 255

Blue 255

Byte

Byte

Byte

Byte

Byte

Byte

Byte

*(Figure 3.6 Color Table)*

The color table contains a sequence of bytes which represents red, green, and blue color tuples. Each image has its own color table.

Local color table is formed by the collection of information about red, green and blue color from all the pixels. Then, it use the median-cut algorithm to figure out a better way to use 8 bits in order to represent higher resolution image.

The following is the step to figure out color table:

1. Collect the color information and select the color channel with the largest variance.

**Function**: findTheMostScatteredColor(int dataStart, int dataEnd); (in file “*ColorTable.java*”)

1. Sort according the channel which contains the largest variance in 3 channel.
2. Keep the recursion from Step 1 to 3 if there is no 7 times yet. If the times of recursion is 7, the next step will be processed.

(The following step will be performed for 8 times, since this is a recursion function.)

1. Use the current median to separate the pixels into two separate parts.
2. Choose the median color for the channel which contains the largest variance in these two parts.
3. Sort the rest two channels, and separate them using the same median as the one in the previous step.
4. Choose the median color for these two channels in these two parts.

As the result, we will set all the pixels with the values from “0000 0000” to “1111 1111” for RGB channels.

**Function:** writeColorTable(ColorTable colorTable); (in file *“GIFEncoder.java”*) & ColorTable.java & ColorTableEntry.java

**3.7 LZW encoding algortihm**

LZW Minimum Code Size

Image Data

Byte

*(Figure 3.7: LZW Image data)*

**LZW Minimum Code Size:** The encoded LZW output length must be the multiple of LZW Minimum Code Size. Here, we choose 8.

Using LZW will make the GIF file size reduce dramatically. This compression method contains two main parts: the dictionary and the data stream.

The following is the explanation about how to build the dictionary rely on color table and the image data:

1. Initialize the dictionary

Add a code for each color entry in color table. (In our case, it relies on the local color table.) For GIF file, there are two special code which must be included in the dictionary. One is “CLEAR CODE”, and the other is “END OF INFORMATION”. “CLEAR CODE” is a symbol for reinitializes the dictionary. “END OF INFORMATION” means the end of image data. The value of these two special types of code depends on LZW minimum code size. Because LZW minimum code size here is 8, which is the same as color table size. Therefore, the code 0 to 255 will be saved for colors. “CLEAR CODE” can be 256, and “END OF INFORMATION” can be 257.

2. Prepare 3 variable lists (N/A, 1990) (Graphics Technology Department)

**Input**: This contains the color entry for each pixels.

**Output List**: This contains the output code.

**Buffer**: This contains the current color entry.

3. Start with output “CLEAR CODE” to “Output list”.

4. Read first Input, and use it in Buffer.

5. Loop

1. Read next Input
2. Check if “Buffer + (nextInput)” is in dictionary.

If so, add nextInput to the end of Buffer. Keep loop.

If no, add “Buffer + (nextInput)” to dictionary. Add the same code to Output List. Now, Buffer will be set to nextInput. “nextInput” is null for now. Keep loop.

6. Output the last content in Buffer

7. Add “END OF INFORMATION” at the end of Output

Because this may result with an extremely large dictionary, “CLEAR CODE” will be used whenever the code length hits 13 bits. When the code length hits 13 bits, the code of “CLEAR CODE” will be sent to output list, and the dictionary will become its initialization (the table we build in STEP 1).

**Function**: compressImageData(int[] uncompressedData); (in file *“GIFEncoder.java”*) & LZWImageData.java

**3.8 Trailer**

0x3B

Trailer (The end of data stream)

Byte

*(Figure 3.8: Trailer)*

This is GIF trailer that represents the end of GIF data stream. It has the fixed value “0x3B”.

**Function**: writeTrailer(); (in file *“GIFEncoder.java”*)

**4. File Format**

Header

Screen Descriptor

Graphic Control Extension

Image Descriptor

Local Color Table

Image Data

Trailer

Keep loop for each frame

*(Figure 4: Overall File Format)*

This is final overall file format that has been stored in GIF file.

**5. Feature**

**5.1 Convert video to GIF**

This is the main goal of our project, which can convert most of video files to GIF file.

(Flickinger, 2015)

**5.2 The interval of your video**

You can choose different time interval, and convert it into GIF format.

**5.3 The resolution of your video**

You can choose the resolution of your GIF file. The choices are “1/4” of the original one, “1/2” of the original one, and the same resolution as the original one.

**5.4 Add your own subtitle**

You can add your own subtitle onto the original video, which will be shown on the GIF file.

**6. Conclusion:**

“Video2GIF” uses the GIF89a format. This is the version of GIF that user can customize the frame per second. In “Video2GIF”, it also contains the features like adding subtitle, editing video interval and choosing GIF resolution selection.

**Reference:**

# Flickinger, M. (2015, Jan 24). *What's In A GIF - LZW Image Data*. Retrieved Jan 24, 2015, from Matthew Flickinger : http://www.matthewflickinger.com/lab/whatsinagif/lzw\_image\_data.asp

# Graphics Technology Department. (n.d.). *GIF File Format Summary*. Retrieved from File Format Info: http://www.fileformat.info/format/gif/egff.htm

N/A. (1990, July 31). *Cover Sheet for the GIF89a Specification*. Retrieved July 31, 1990, from W3C: https://www.w3.org/Graphics/GIF/spec-gif89a.txt