Design Documentation

Edit New page

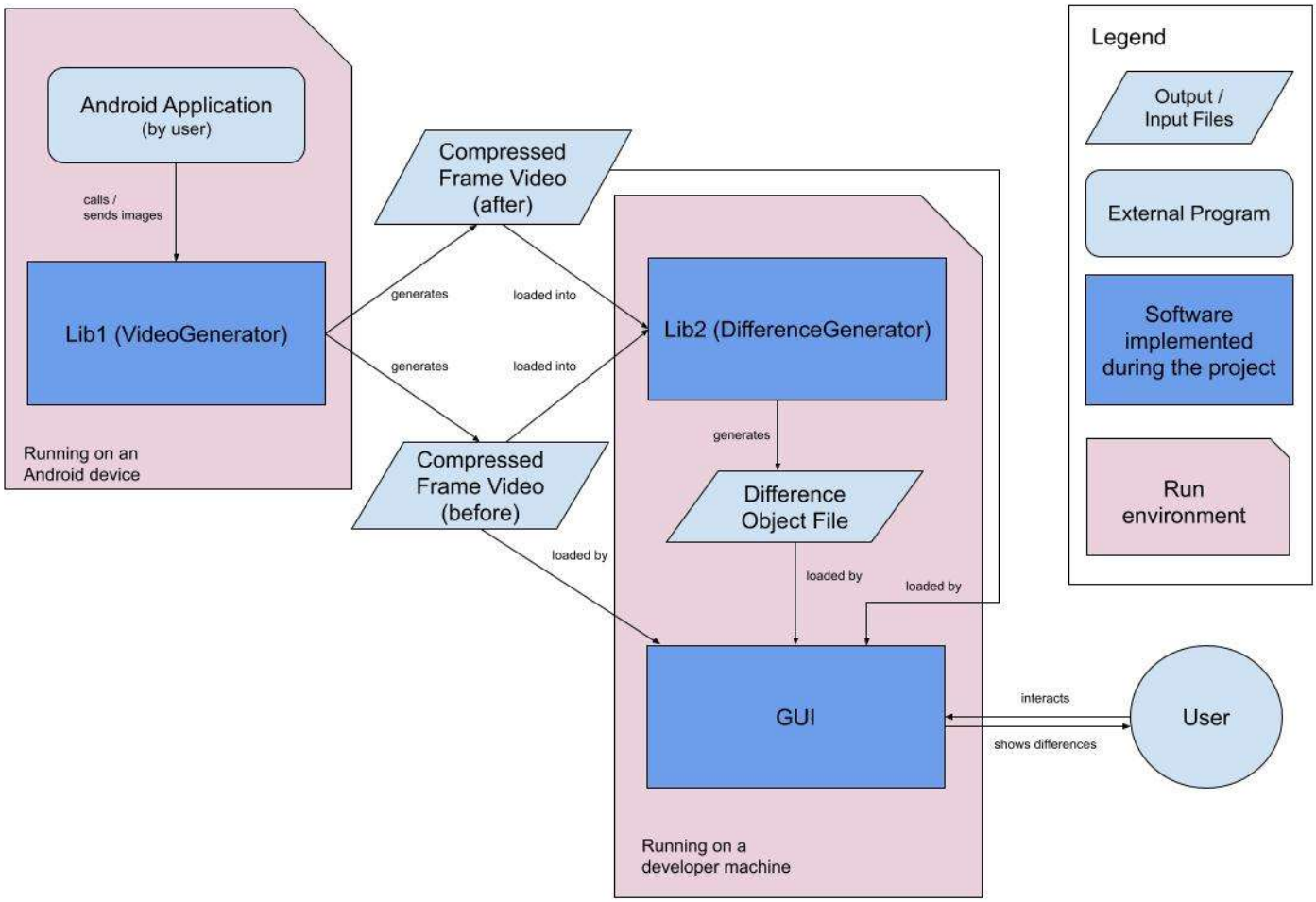
Jump to bottom

a-miscellaneous edited this page 6 hours ago · 22 revisions

Runtime Components

The software is comprised of three main components:

- **VideoGenerator Library:**
  - Saves sequences of images as compressed video files.
  - Usable by both Android and desktop applications.
- **DifferenceGenerator Library:**
  - Compares video files to find differences between frames.
  - Utilizes an AlignmentAlgorithm for sequence alignment.
  - Supports masking to exclude irrelevant content.
- **Graphical User Interface (GUI):**
  - Allows users to load two video files.
  - Uses the DifferenceGenerator library to display matching images, differences, and added/deleted frames.

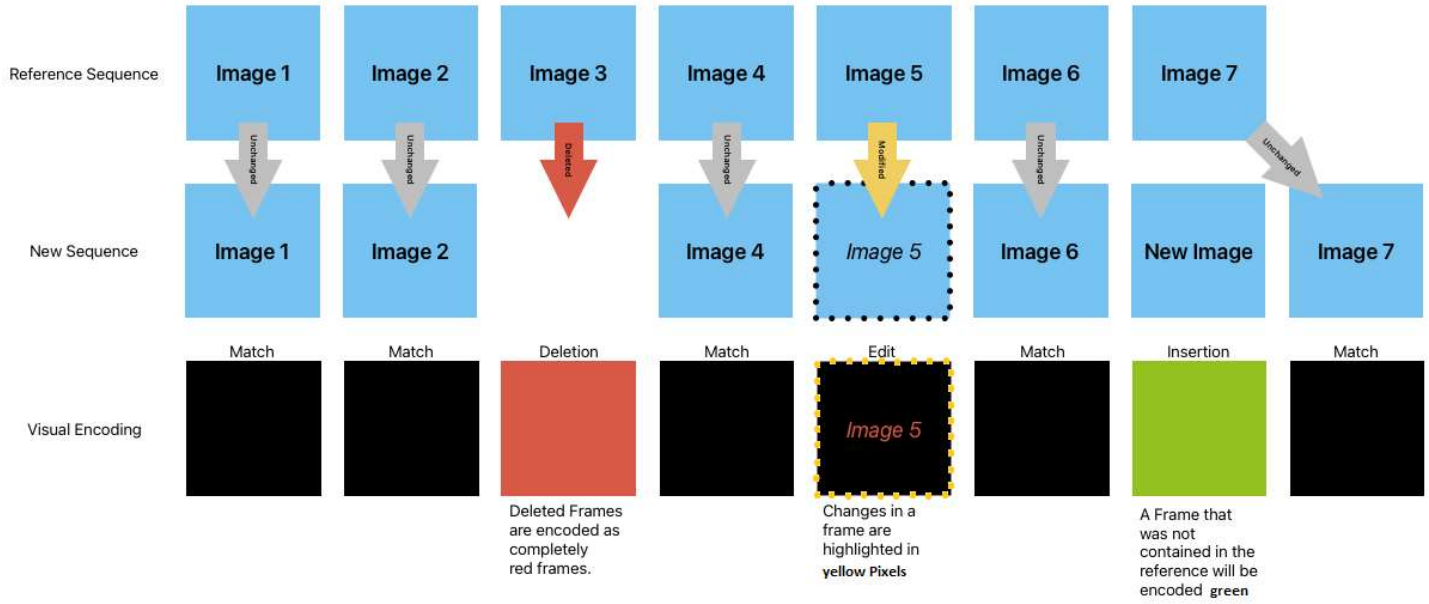


For interoperability, all subprojects are written in `Kotlin` and built with `Gradle`.

DifferenceGenerator Library

This library focuses on comparing two sequences of images to identify inserted, deleted, or edited images. A modified version of the Gotoh algorithm, a sequence alignment algorithm popular in bioinformatics, is utilized. Dynamic programming is employed to find a good alignment (though not optimal in this case). To speed up the process, a divide and conquer approach is implemented that splits the problem at every perfectly matching frame.

The DifferenceGenerator library exposes the `DifferenceGenerator` class, taking an `AlignmentAlgorithm` object, two input video file paths, an output file path, and an optional mask file path.



Assumptions

Assumptions about input videos:

- Images can occur multiple times.
- Images stay in the same order if included in both videos.
- Similarity of two images computed by differing pixels.
- An image can be added/deleted.
- Many frames will be exactly the same in both videos.

Alignment Elements

- Match - Two images are equal
- Edit - An image in the new sequence was edited
- Insertion - A new image was added to the new sequence
- Deletion - An image from the reference sequence was deleted

Alignment

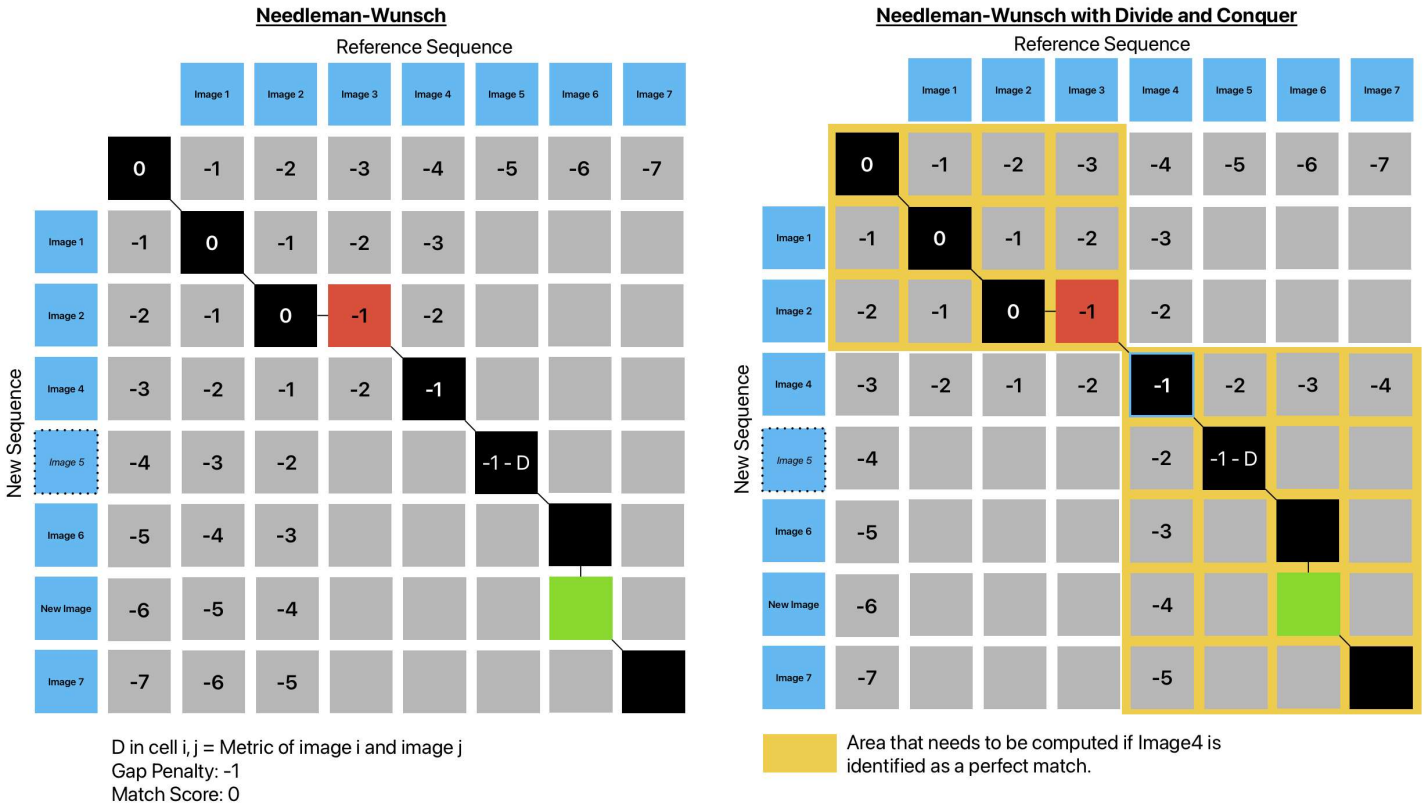
The result of the comparison is an alignment. This does not contain the original images but only the information about how these sequences are best aligned to each other.

This alignment can be expressed as a sequence of the terms defined above (e.g., Match, Match, Deletion, Match, Edit, Match, Insertion, Match).

The alignment can also be expressed as an image sequence of black(match), red(deletion), green(insertion), and black/yellow(edit) frames (see Visual Encoding in picture above).

Calculating the Alignment

Info: The problem faced here is similar to sequence alignment problems common in Bioinformatics. Thus, many bioinformatics algorithms are capable of solving these kinds of issues. The easiest of this is the **Needleman-Wunsch** algorithm (NW).



Needleman-Wunsch Algorithm

Even though the **Gotoh algorithm** is used in this project as a sequence alignment algorithm, the NW algorithm will be explained first using an example(Picture above). NW is a dynamic programming algorithm on a 2D matrix, each sequence on one axis. The algorithm calculates a score for every possible pair and for every element being an insertion or deletion.

For the calculation, a `gap penalty` is set and applied when introducing a gap into the alignment. Additionally, a `mismatch penalty` is subtracted when the algorithm matches two values that are not equal.

In normal bioinformatic sequence alignment, two elements or characters can either be equal or not. Thus, in the vanilla NW algorithm, there is a `match score` and a `mismatch score`.

For the application on images, this is different. A **Metric** is introduced that measures how similar two images are. One possible metric is the `Pixel Count Metric` (Count pixel differences between the images). In this case, the metric is used as a continuous function instead of the two discrete values match/mismatch.

Recursive Formula for Needleman-Wunsch using a Metric

$$n = |ReferenceSequence|$$
$$m = |NewSequence|$$
$$i \in [0..n], j \in [0..m]$$

Initialisation

$$M(i,0) = i * gapScore \quad \forall i \in [0..n]$$
$$M(0,j) = j * gapScore \quad \forall j \in [0..m]$$

Computation of the matrix M

$$M(i,j) = \max \begin{cases} M(i-1, j-1) - Metric(i-1, j-1) \\ M(i, j-1) - GapScore \\ M(i-1, j) - GapScore \end{cases}$$

Starting from the upper left cell of the matrix, for every cell, three options are considered and their maximum is taken.

- **Match:** take the value from the diagonal (left, upper) cell and subtract the `metric`
- **Deletion:** take the value from the left cell and subtract the `gap penalty`
- **Insertion**: take the value from above and subtract the `gap penalty`

info: If the alignment moves vertically or horizontally, a gap (insertion or deletion) is introduced because the alignment only proceeds with one sequence (on one axis). If it moves diagonally, it proceeds on both axis, thus the sequences are matched.

Upon arriving in the bottom right cell, the actual alignment needs to be **traced back**. This is done by starting from the bottom right cell and always going in the direction of the maximum value of the three preceding cells.

Gotoh Algorithm

The Gotoh algorithm builds upon the NW and adds the possibility to account for the gap length. Meaning that a sequence of `n` Insertions or Deletions can be punished less (or more) than just `n * gap_score`. In our case this makes particular sense because it is likely that multiple frames are added at the same time. There are now a `gap open score` and a `gap extension score`. But for this, it needs 3 matrices instead of 1.

The algorithm uses three matrices: **score**, **gapA**, and **gapB**. These matrices are used to keep track of the scores for matching images, opening a gap in sequence A, and opening a gap in sequence B, respectively (More information and interactive example).

Divide and Conquer

A divide and conquer approach is added to the algorithm to speed up calculation and improve accuracy(see picture above). This involves finding perfect matching frames first and using them to exclude large fractions of the matrix from calculation. This is possible because, if there is a perfect match, the alignment has to pass this cell. Thus, everything to the upper right and lower left of the cell can be excluded.

Each image is hashed to be able to find identical images fast without using a pixel-wise comparison. We ignore frames if they appear multiple times in a video, since then it it unknown with identical match is correct.

Masking

A mask is an image applied to a frame to mask out parts deemed irrelevant for generating differences. We use opaque pixels for areas that are to be ignored and clear pixels for areas that are taken into account.

Output

After computing the alignment sequence, an output video is generated with color coding:

- **Match:** Black pixels for matching, red for differing.
- **Insertion:** All pixels are green.
- **Deletion:** All pixels are blue.
- **Perfect Match:** All pixels are black.

The GUI can also access the alignment produced by the DifferenceGenerator directly.

GUI

The usage is explained in User Documentation

Compose works on components, these are rerendered depending on listeners. Our main Application is a listener that is responsible for changing screens. It listens on the global state for a change of the active screen.

Video Select Screen

All file interaction uses either an open or save dialog depending on the use case. These Dialogs remember where they were last opened using the state. Before the difference calculation is started we check for compatibility (creation date, frame size, mask compatibility ect.) If we find a discreptancy then a warning or confirmation is thrown, depending on the severity. If an error occurs then we show the user an error and prevent the program from crashing.

Settings Screen

The settings screen saves a copy of the current state and then either keeps these changes or discards them, depending on the users decision.

Differences Screen

- On creation a `navigator` object is created, this keeps track of which frame is currently active and updates images. The displayed images have a listener that rerenders the component once the image is changed.
- Since Windows are also components these can be actively rerendered (window position and size must be saved) and they can interact with the navigator.
- Keyboard listeners only propagate from parents to children, hence the screen has to be focused to intercept keypresses and interact with the navigator.
- If the screen is rerendered we have to explicitly dispose of the navigator object, to free the video files
- the timeline renders grey frames as a placeholder while the thumbnails load.

Currentness of Documentation

This documentation is written on the basis of Release 12. Changes beyond this release are not yet documented here.

Legal Information

The entire project operates under the MIT license, making it open and free for use.