Kelir - Technical Notes

Adhi Hargo

---

1 Introduction

It has been noted that CG animation, though more popular in recent years, only dominates big-budgeted feature film production. Beyond that, 2D animation is still relied upon for its faster and cheaper production cycle, especially in serialized TV titles. Even titles using CG animation still incorporate 2D animation workflow for creating animatics, and sometimes effects animation, too. A robust 2D animation tool is therefore crucial for efficient production of an animated product.

Kelir [1] is a 2D animation software written towards the goals of *simplicity*, *ease of use*, and *robustness*. It's important to have a good codebase to facilitate experiments in different fields of production, thus Kelir's release with a *GNU GPL v2* free software license, also striving for a consistent and readable code.

2 Build Process

Building Kelir only requires a C++ compiler and full Qt library installation. It uses Qt's `qmake` [2] utility to manage build process, all contained in `kelir.pro`[1] project file. To compile, simply execute:

```
> qmake
> make
```

[1] `<srcdir>/kelir.pro`

3 Code Structure

Kelir is the author's first attempt at creating an animation software. With no experience and few references, it is largely an experiment prone to large rewrites. The following is an overview of Kelir's current code structure to facilitate any newcomer's understanding of Kelir's source code, and will be updated infrequently to reflect any significant update or restructure.

3.1 Document

A `Document`[2] contains global settings like default render resolution and output format, and a non-empty list of scenes. Each `Scene`[3] contains a non-empty list of layers. Various types of layers can be contained in a scene, all of them inherited from `AbstractLayer`, but only those further inherited from `DrawableLayer`[4] can contain any frame.

There is already a `VectorLayer`, but at this moment, only `BitmapLayer` is used and supported. No vector-based geometric operations for the drawing tools yet. Both `VectorFrame` and `BitmapFrame` are to be inserted to their respective layer class only.

3.2 Canvas

The `PaintCanvas`[5] class implements an "infinite" canvas, by way of an internal transformation matrix. It's also responsible for storing the result of tool operations in the right frame. The canvas holds a list of editing tools deriving from `AbstractTool`[6], keeping track of which one is active, and disabling them all if current layer isn't editable (locked or not a `DrawableLayer`).

---

[2] `<srcdir>/data/document.hpp`

[3] `<srcdir>/data/scene.hpp`

[4] `<srcdir>/data/drawablelayer.hpp`

[5] `<srcdir>/main/paintcanvas.hpp`

[6] `<srcdir>/abstracttool.hpp`

Currently `PaintCanvas`'s drawing operation is unoptimized and not supporting multi-layered drawing operation, the only reason why Kelir is still limited to single-layer. A `FrameCache`[7] meant to speed up multi-layer frame building by asynchronous update is already prepared, but a way to integrate it cleanly to `PaintCanvas`'s drawing routine haven't been figured out yet.

3.3 Timeline

Along with the canvas, timeline area is arguably the most important part of an animation software, both being the areas an animator interacts with most often. Current capability is easy graphical manipulation of multiple frames on multiple layers (still limited to visible area).

Kelir's `TimeLineArea`[8] is divided into 3 main internal spaces: header, and splitter-delimited layer and frame space. This area's implementation borrows from several other animation softwares. For example, there's a thin interval overlay above frames to help readability (drawing from John K's critique [3] of Toon Boom Animate's interface):

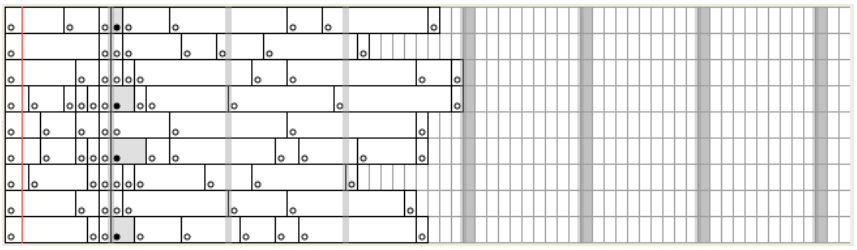

Figure 1 Frame space section of the timeline area.

3.4 Input/Output

Currently Kelir is only capable of saving to PNG image sequence with JSON-formatted index, handled by `ImageSequence`[9]. The arrangement isn't ideal yet,

[7] `<srcdir>/main/framecache.hpp`

[8] `<srcdir>/areas/timelinearea.hpp`

[9] `<srcdir>/io/imagesequence.hpp`

and weaknesses of `Document`'s data structure are apparent while implementing this class. This class still has no file read function.

4 Future Development

There are lots of things already planned to be implemented in Kelir. This section enumerates some of them, in no particular order.

4.1 Document Structure

a. Support multi-scene document editing.
b. Design a native (text/binary) file format flexible enough for any planned feature, breaking backward compatibility if need be.

4.2 Drawing Tools

a. Optimizing multi-layered, random-seekable draw routine.
b. Adding in several more standard drawing tools: Bézier curve, bucket fill, paintbrush.
c. Simultaneously develop vector-based version of these tools.
d. Implement robust curve interpolation ("line smoothing") and graphics tablet support.

4.3 External Libraries

a. Integrating FFMPEG to support reading and writing various media formats. Crucial for audio/video layer, and scene rendering feature.
b. Integrating SDL for optimized media playback. Crucial at least for audio playback (in simple tests, SDL Audio is more flexible and portable than Phonon or Qt Multimedia).
c. Integrating libgif, to support exporting whole document as GIF animation.

5 References

[1] Kelir's GitHub repository [URL]

[2] Qt Project - qmake Manual [URL]

[3] John Kricfalusi - "Toonboom Bugs 1: Timeline Hard To Read" [URL]