Google Summer of Code

Proposal for: GSoC 2026

Project: Kdenlive

Proposal title: Improving effect widgets for powerful effects

Length: ~350 Hours (Large - 12 weeks)

Mentor: Jean-Baptiste Mardelle (jb@kdenlive.org)

Student details:

Name: Yash Bavadiya

Email: krbavadiya11@gmail.com

LinkedIn: Yash Bavadiya

GitHub: Yash Bavadiya

GitLab(KDE) : Yash Bavadiya

IRC/Matrix handle: @xevrion:matrix.org

Phone Number: +91 9879163975

Time Zone: Indian Standard Time (IST) (UTC +05:30)

Institution: Indian Institute of Technology, Jodhpur

Program: B.Tech in Computer Science and Engineering

Introduction:

Kdenlive is the acronym for KDE Non-Linear Video Editor. It works on Linux, Windows, macOS, and BSD. The effect system in Kdenlive gives options to users to apply different visual and audio filters to their clips, effects like color correction, blur or speed remapping. Most of the effects are powered by external libraries but the UI for controlling their properties and state is built inside Kdenlive. But when the UI is poor or missing, users can not properly use powerful effects that are already available.

Each widget in Kdenlive's effect panel is backed by `AssetParameterModel` (`src/assets/model/assetparametermodel.cpp`), which stores effect parameter values as a `QDomElement` tree and acts as the bridge between the UI and MLT's filter pipeline. When a user adjusts a widget such as dragging a curve point, repositioning a color stop, or modifying a speed curve handle, the widget emits a `valueChanged` signal. `AssetParameterWidget` catches this and calls `AssetParameterModel::setParameter()`, which serializes the new value into the XML representation and emits `dataChanged()`. This triggers the timeline to call `Mlt::Filter::set()` with the updated parameter string, causing MLT to re-process the affected frames and update the monitor preview in real time.

Every new widget built in this project will inherit from `AbstractParamWidget` and implement `slotRefresh()` for model-to-widget sync and `valueChanged` emission for widget-to-model sync. Serialization formats will be designed to match the exact string formats expected by the underlying MLT filters, for the curves widget, this means preserving the existing per-channel point-list format while extending it to carry independent data for all channels, maintaining backward compatibility with existing `.kdenlive` project files.

Currently,
The Curves have a drop-down menu for selecting the channel and also to adjust three channels such as “Red”, “Green” and “Blue” independently, we need to add the “Curves” effect three times to the same clip, one per channel.
In the Gradient effect, I noticed that there are only two stops for color, we can’t add more color stops.
The Time Remapping panel has a visual dual-timeline representation, but transitions between keyframes are linear only, there are no bezier handles and no easing presets. Creating a smooth speed ramp currently requires manually placing many closely-spaced keyframes.

To solve these problems, this project will build a proper widget that allows editing curves with tabs to handle each color channel individually. The color gradient effect will have the option to create an arbitrary number of color stops. Adding curves to the speed ramp editor would make it easier for users to see how the speed is changing and we can also add different presets like Ease In, Ease Out, Ease In/Out, and Linear.

Why Kdenlive?

As a developer who also creates tutorial and project videos, I've been using Kdenlive as my go-to video editor on Linux. What makes it special to me is that it's not just software I use, it's software I've contributed to. Every time I open it now, I see pieces of work I've done in the codebase, which makes the whole experience feel personal.

What surprised me digging into the codebase was how cleanly the effect system is designed, every widget plugs into `AssetParameterModel` through the same interface, yet the rendering power comes from MLT and `libavfilter` underneath. That separation between UI and pipeline is elegant, and it's what makes this project's improvements meaningful rather than cosmetic.

When I looked at the GSoC ideas for Kdenlive, the "Improve Effect Widgets" project immediately caught my attention because it sits exactly at the intersection of what I'm good at, Qt/C++ for the backend logic and strong UI sensibility from my frontend experience. The current effect UI has real gaps that frustrate users daily, and fixing them would make Kdenlive genuinely more powerful for everyone.

Project goals:

- **Curves Widget** → Replace the channel dropdown in `CurveParamWidget` with a `QTabWidget`, one tab per channel (RGB, Red, Green, Blue, Alpha, Luma). Each channel maintains independent curve state using `std::map<CurveModes, QString>`, so switching tabs no longer discards edits. Currently users need to apply the effect three times for independent per-channel control, this reduces it to a single effect instance. Per-channel data is stored as separate kdenlive:-namespaced parameters (`kdenlive:curve_red, kdenlive:curve_green`, etc.), fully backward-compatible with existing project files.
- **Color Gradient Widget** → Build a standalone `GradientEditorWidget` in `src/assets/view/widgets/` supporting an arbitrary number of draggable color stops, replacing the hardcoded two-stop system currently tightly coupled to the title clip editor in `src/titler/gradientwidget.cpp`. Stops are stored as `QList<QPair<double, QColor>>` and serialized to MLT's expected format. Integrates with `AssetParameterModel` like any other effect widget with real-time `QLinearGradient` preview.
- **Speed Ramp** → Add bezier curve handles to the existing `RemapView` connector lines in `src/dialogs/timeremap.cpp`, enabling smooth easing transitions between keyframe speeds. Reuses `BezierSplineEditor` for handle rendering. Handle data is stored in a separate MLT parameter, leaving `time_map` untouched for full backward compatibility. Includes presets: Linear, Ease In, Ease Out, Ease In/Out.
- **QML Monitor Overlays (stretch goal)** → Extend the pattern established by `MonitorRotoScene.qml` to allow direct manipulation of effect parameters (such as

gradient position or curve points) on the monitor preview, with changes propagating back through the controller to `AssetParameterModel`. Scope and feasibility to be finalized with Jean-Baptiste during Community Bonding.

Implementation:

Curve Widgets:

Current Architecture

The `frei0r.curves` effect uses a Channel parameter (value 0=Red, 0.1=Green, 0.2=Blue, 0.3=Alpha, 0.4=Luma, 0.5=RGB) alongside a single curve parameter named `kdenlive:curve` storing point data in `"x/y;x/y;"` format. The `frei0r.bezier_curves` variant uses parameter 5 in `"h1x;h1y#px;py#h2x;h2y|..."` format. In both cases, the widget stores only one channel's curve at a time. Switching the channel dropdown discards any unsaved edits to the previous channel. So currently, there is no per-channel memory in the widget layer.

Proposed Changes

The Channel dropdown in `CurveParamWidget` will be replaced with a `QTabWidget`, one tab per channel (RGB, Red, Green, Blue, Alpha, Luma). Per-channel curve state will be stored in a `std::map<CurveModes, QString>` inside the widget. When the user switches tabs, the current channel's curve string is saved to the map before loading the new channel's data into the editor:

```
// On tab switch:
m_channelData[previousMode] = m_edit->toString();
m_edit->fromString(m_channelData[newMode]);
m_currentMode = newMode;
```

When the user edits a curve, `valueChanged` emits the full serialized multi-channel state so `AssetParameterModel` can update all relevant parameters in one operation.

Serialization and Backward Compatibility

Each channel's curve will be stored as a separate kdenlive:-namespaced parameter in the effect XML: `kdenlive:curve_red`, `kdenlive:curve_green`, `kdenlive:curve_blue`, `kdenlive:curve_alpha`, `kdenlive:curve_luma`, with `kdenlive:curve` preserved for the RGB composite channel. The existing Channel parameter is kept for MLT filter compatibility.

On `slotRefresh()`, the widget checks for the presence of `kdenlive:curve_red`. If absent (old project), it loads `kdenlive:curve` into all channels as the default, the identity curve `"0/0;1/1"`, preserving original behavior exactly. If present (new project), each

channel loads independently. Old .kdenlive project files require no migration and open without data loss.

Gradient Widgets:

The existing gradient widget lives in `src/titler/gradientwidget.cpp` and is tightly coupled to the title clip editor, it supports only two fixed color stops (start and end) and cannot be reused in the effects panel. The goal is to build a standalone `GradientEditorWidget` in `src/assets/view/widgets` that supports an arbitrary number of draggable color stops and integrates with `AssetParameterModel` like any other effect widget.

The widget will render a gradient bar using `QLinearGradient`, updating in real time as stops are added, moved, or removed. Each stop is represented as a draggable handle on the bar. Clicking a handle opens a `QColorDialog` to change its color. A minimum of two stops is enforced. Stops are stored as `QList<QPair<double, QColor>>` where the double is the normalized position (0.0 to 1.0).

```
class GradientEditorWidget : public AbstractParamWidget {
    void addStop(QColor color, double position);
    void removeStop(int index);      // enforces minimum 2 stops
    void moveStop(int index, double newPosition);
    QString toString() const;        // serializes to MLT-compatible string
    void fromString(const QString&); // deserializes on project load

    QList<QPair<double, QColor>> m_stops;
};
```

Serialization will use the format `"position=#rrggbb;position=#rrggbb;..."` ,for example `"0.0=#ff0000;0.5=#00ff00;1.0=#0000ff"` , which maps directly to what MLT's gradient filter expects as its colors and positions parameters. `toString()` produces this string and passes it through `valueChanged` to `AssetParameterModel`. `slotRefresh()` calls `fromString()` to reconstruct stop positions and colors when the project loads, ensuring round-trip fidelity.

Backward compatibility: the existing two-stop gradient parameter in old project files is a valid instance of this format with exactly two stops, so old projects load without any special handling.

Speed Ramp:

The Time Remapping panel (`src/dialogs/timeremap.cpp`) already provides a dual-timeline `RemapView` widget where keyframes map source clip time to output timeline time, with the speed at each keyframe derived from the slope of the connector line between them. However, transitions between keyframes are currently **linear only,** the speed changes

instantaneously at each keyframe boundary rather than easing smoothly into or out of the new speed. There are no bezier handles on the connector lines, no easing presets, and no way for users to create smooth slow-motion ramps without manually placing many closely-spaced keyframes.

The proposed enhancement adds **bezier curve handles** to the existing `RemapView` connector lines, allowing smooth easing transitions between keyframe speed values. This reuses the existing `BezierSplineEditor` (src/assets/view/widgets/curves/bezier/beziersplineeditor.cpp) for handle rendering and interaction logic. Each keyframe pair will expose two tangent handles, one controlling ease-out from the left keyframe and one controlling ease-in to the right keyframe. Common presets (Ease In, Ease Out, Ease In/Out, Linear) will be available via a `QComboBox`, each mapping to predefined bezier handle positions.

The serialized `time_map` format currently stores linear keyframe pairs as `"time=value;time=value"` via `RemapView::getKeyframesData()`. To store bezier handle positions without breaking existing project file compatibility, handle data will be stored in a **separate parameter** on the MLT timeremap link, leaving `time_map` untouched and backward-compatible with existing `.kdenlive` files. Projects created before this change will open correctly with linear interpolation as the default.

The main technical risk is integrating bezier handle state into RemapView's existing `paintEvent` without disrupting the current keyframe rendering pipeline. This will be the first thing prototyped during Community Bonding, before any other Speed Ramp work begins, so that design issues surface early with time to course-correct with Jean-Baptiste.

QML Monitor Overlays (Stretch Goal):

Kdenlive's monitor overlays live in `src/monitor/view/` as QML components. Currently overlays like `OverlayThirds.qml` and `MonitorSafeZone.qml` are purely visual. `MonitorRotoScene.qml` is the most advanced example, it uses `K.MonitorOverlay` with a `controller.overlayType` binding that connects QML interaction back to C++ via `setParameter()`.

The stretch goal extends this pattern to the new effect widgets, allowing users to manipulate parameters like gradient stop positions or curve points directly on the monitor preview by dragging handles. QML handles emit position changes, the C++ controller receives them and calls `AssetParameterModel::setParameter()`. This follows the exact same architecture already established in `MonitorRotoScene.qml`. The scope and specific widgets to target will be decided with Jean-Baptiste during Community Bonding before any implementation begins.

Unit Testing Approach:

Each widget will have unit tests written alongside the implementation, not at the end. For the curves widget, tests will verify that switching tabs preserves independent curve data per channel and that serialization/deserialization round-trips correctly. For the gradient widget, tests will cover add/remove/reposition of stops and edge cases like minimum 2 stops. Tests will be added to Kdenlive's existing test suite in the `tests/` directory.

Documentation Plan:

Code documentation will be written inline as each widget is implemented, not deferred to the final week. Each new class and public method will have doxygen-style comments explaining its purpose and parameters. At the end of the project, I will write a summary blog post on KDE Planet covering what was built, design decisions made, and how to use the new widgets.

Availability:

I plan to dedicate 30-45 hours per week to the project and will be most active from 10 AM to 10 PM IST. My end-semester exams end by April 29, so I'll be completely free from then onwards, meaning I can actively participate during the Community Bonding Period and fully enjoy the coding period.

Timeline:

Time Frame	Tasks
Community Bonding (May 1 – May 24)	Deep dive into `curveparamwidget.ipp`, `gradientwidget.cpp`, `timeremap.cpp`, and `bezierssplineeditor.cpp`. Set up weekly communication cadence with Jean-Baptiste. Present proposed widget APIs and serialization formats to Jean-Baptiste for feedback and approval. API design is done before bonding ends, not during coding. Study existing QML overlays in src/monitor.
Coding officially begins! Week 1 (May 25 – May 31)	Audit `CurveParamWidget`, understand how `CurveModes` enum works, how the current dropdown switches channels, and how curve data is stored. Identify exactly what changes are needed.
Week 2 (Jun 1 – Jun 7)	Design the `QTabWidget` structure for per-channel curves. Define the `std::map<CurveModes, CurveData>` storage. Implement tab switching with independent curve state per channel.

Week 3 (Jun 8 – Jun 14)	Connect per-channel curve data to the effect parameter model. Ensure serialization and deserialization of all channels works correctly. Fix `ValueLabel` axis color rendering per active tab.
Week 4 (Jun 15 – Jun 21)	Real-time preview feedback when switching tabs. Write unit tests for curve data per channel. Address mentor review feedback. Document the curves widget.
Week 5 (Jun 22 – Jun 28)	Buffer for curves review feedback. Begin gradient widget skeleton, create class in `src/assets/view/widgets/`, implement basic stop data structure and `QLinearGradient` rendering.
Week 6 (Jun 29 – Jul 5)	Complete basic gradient bar rendering. Implement mouse hit detection for stop handles. Begin drag-to-reposition logic with position clamping between adjacent stops.
Midterm (Jul 6 – Jul 10)	**Submit midterm evaluation. Curves widget should be complete and in review. Gradient widget in progress.**
Midterm evaluation deadline July 10 – 18:00 UTC	**Mid-Term Evaluation Submission**
Week 7 (Jul 11 – Jul 17)	Implement add/remove color stops in gradient widget. Implement drag to reposition stops. Real-time gradient preview update.
Week 8 (Jul 18 – Jul 24)	Wire gradient widget to effect parameter model. Handle edge cases, minimum 2 stops, color picker integration. Get mentor review.
Week 9 (Jul 25 – Jul 31)	Buffer for gradient, address review feedback. Polish and finalize gradient widget. Begin speed ramp research and `RemapView paintEvent` analysis to plan bezier handle integration.
Week 10 (Aug 1 – Aug 7)	Implement bezier handle rendering on `RemapView` connector lines. Wire handle positions to a separate parameter on the MLT timeremap link. Implement preset `QComboBox` (Linear, Ease In, Ease Out, Ease In/Out) with predefined bezier handle positions for each preset.
Week 11 (Aug 8 – Aug 14)	Connect bezier handle data to the separate timeremap MLT parameter. Write unit tests for preset handle positions and backward compatibility with existing linear keyframe projects. Verify real-time preview updates correctly on monitor.
Week 12 (Aug 15 – Aug 24)	Address all pending review feedback. Final testing of all three widgets. Code cleanup and documentation. Submit final work product.

I will communicate with Jean-Baptiste daily on Matrix. I will post a weekly progress update in the `#kdenlive-dev:kde.org` Matrix channel every Sunday.

Measurable Success Criteria:

Curves widget is complete when: all 6 channels (RGB, Red, Green, Blue, Alpha, Luma) have independent curve state that persists across tab switches; serialization round-trips correctly, saving and reloading a project produces identical `QPointF` coordinates for all channels within floating-point epsilon; and old project files containing only kdenlive:curve open without any data loss.

Gradient widget is complete when: stops can be added, removed, and repositioned freely with a minimum of 2 stops enforced; the gradient preview updates in real time on every change; `toString()/fromString()` round-trips are lossless; and the widget integrates with `AssetParameterModel` identically to any other effect widget.

Speed Ramp is complete when: bezier handles are draggable on `RemapView` connector lines; all 4 presets produce correct handle positions; the handle data round-trips through the separate MLT parameter without corrupting `time_map`; and existing projects with linear keyframes open correctly with no visual change.

About me:

I'm Yash Bavadiya, a 2nd year B.Tech CSE student at IIT Jodhpur. I love building things that solve real problems, not just code that sits unused. My stack includes C/C++, React, TypeScript, and JavaScript, and I'm comfortable working across both frontend UI and systems-level code.

Before I joined college, I started coding during the Covid pandemic when the whole world slowed down and I finally had time to experiment. I began with Python, building small games in Pygame, nothing fancy, just enough to feel the magic of making something move on a screen. That curiosity pulled me deeper, and I moved to C and then C++, where I discovered something I hadn't expected: how close you can get to the machine, how every byte matters.

Coming to IIT Jodhpur, I properly studied C++ and object-oriented programming, and started understanding how UIs are built from the ground up. When I switched to Linux, I had to leave Adobe Premiere Pro behind, and while looking for an alternative, I stumbled upon Kdenlive. I've been using it ever since for my tutorial and project videos. What started as a replacement quickly became something I genuinely loved, a powerful, open-source video editor built by a community that actually cares.

My Open Source Journey started in October 2025 during HacktoberFest. That was the moment it clicked, you can contribute to software you use every single day. Real code, real users, real impact. I was amazed and excited at the same time. I started contributing to KDE, first to the developer documentation site and then to Kdenlive itself. When I saw the "Improve Effect Widgets" project in GSoC 2026, I knew this was the right project as it sits at the exact intersection of Qt/C++ widget development and UI design, and the problems are ones I've personally hit as a Kdenlive user.

Some projects I've built:

1. **DaemonDoc** → AI documentation engine that auto-updates READMEs on every push using webhooks and Redis queues. Built with Python and deployed live.
2. **BreathClean** → Route scoring engine that combines AQI data, weather, and geospatial indexing to suggest healthier walking/cycling routes. Live project.
3. **Peek-a-Repo** → Browser extension for GitHub that lets you browse repository files without clicking into folders, built using GraphQL and DOM interception.
4. **Chronapse** → Timelapse recorder combining a Go TUI, Python capture layer, and FFmpeg rendering pipeline.
5. **DocuMentor** → Fully offline RAG pipeline using FAISS and 4-bit quantized LLM inference. Currently used by 15+ students at IIT Jodhpur.

These projects span systems programming, AI tooling, and developer utilities, reflecting the same mindset I bring to Kdenlive: build things that actually get used.

KDE Developer Tutorial and Article Site → Worked through a comprehensive search UX improvement issue, delivering 8 merged MRs (!733, !734, !735, !736, !737, !741, !742, !744). Contributions included: lazy-loading the offline search index to reduce page load time, adding keyboard shortcuts (/ and Ctrl+K) to focus the search field, implementing keyboard navigation for search results (↑↓ to navigate, Enter to open, Esc to close), fixing the search icon overlapping input text, replacing inline styles with Bootstrap utility classes, updating lunr.js to the latest version, and using Bootstrap list-group for more compact search results.

Kdenlive MR !824 (MERGED) → Added 6 missing HD-ready (1366×768) project profiles at standard frame rates (25, 29.97, 30, 50, 59.94, 60 fps) to `data/profiles/`, fixing a gap reported by users who couldn't find this resolution in the New Project dialog.

Kdenlive MR !827 (MERGED) → Fixed duplicate profiles appearing in the New Project dialog. Root cause: both MLT system profiles and Kdenlive's own profiles were loaded without deduplication. Rewrote ProfileRepository::refresh() in `src/profiles/profilerepository.cpp` to deduplicate by spec key (width, height, framerate, colorspace), with Kdenlive profiles taking priority over MLT ones.

Kdenlive MR !826 (MERGED) → Added common video, audio, and image MIME types to `data/org.kde.kdenlive.desktop`, allowing Kdenlive to appear as an option when opening media files from file managers.

Kdenlive MR !831 (MERGED) → Implemented configurable clip type colors (video, audio, title, image, slideshow) in the Colors and Markers settings page. Added 5 new `KColorButton` entries to `configcolors_ui.ui`, new color settings to `kdenlivesettings.kcfg`, and modified 5 color functions in `timelinecontroller.cpp` to respect user preferences while falling back to theme defaults when transparent.

Kdenlive MR !835 (MERGED) → Fixed broken tab order in the Render dialog. The <tabstops> block in `renderwidget_ui.ui` was both incomplete (10 widgets missing) and

misordered, causing Tab to jump randomly across all three dialog tabs. Reordered existing entries and added missing widgets so Tab follows the visual layout top-to-bottom.

Kdenlive MR !837 (IN PROGRESS) → Implemented the "Identify Gaps" feature requested in bug #352409. Adds an action to the Timeline menu and track header right-click menu that automatically detects gaps in the timeline and places ranged guide markers spanning the full gap length. Gap detection uses a union-of-intervals algorithm across all video tracks, a gap is only flagged when all video tracks are empty at that position. Iterated through multiple rounds of feedback from Eugen Mohr and Bernd Jordan, implementing: renamed action to "Identify Gaps", switched to ranged markers via addRangeMarker(), added per-track option ("Selected Track" / "All Tracks") in the track header context menu.

Carl Schwan (KDE maintainer, Marknote, `https://matrix.to/#/@carl:kde.org`) reviewed my frontend contributions to the KDE Developer Tutorial site and can speak to my engagement with the KDE community.

Community Presence

Since joining the Kdenlive community, I've been actively engaging beyond just submitting code. I introduced myself in the `#kdenlive-dev:kde.org` Matrix channel, commented on bug reports to confirm reproducibility on Linux, and participated in MR review discussions with Jean-Baptiste Mardelle and Eugen Mohr. I diagnosed the root cause of the drag buttons disappearing in the clip monitor when effects are applied (issue #2151), traced it through the QML codebase, and posted my findings in the issue thread asking for guidance before writing any code. I understand that open source is about collaboration first, code second.

I'm comfortable working independently with a remote mentor across time zones, and I communicate proactively. English is not my native language (Gujarati/Hindi), but I work and write in English daily without difficulty. I am applying only to KDE and would choose KDE if accepted.

Before and After

	Before	After
Curves	Single shared curve for all channels. Switching channel dropdown discards previous edits. Need to apply effect 3× for independent per-channel control.	One tab per channel. Each channel stores its own independent curve. All channels adjustable in a single effect instance.
Gradient	Only available in title clip editor. Effect panel gradient limited to two fixed color stops.	Standalone widget in effects panel. Arbitrary number of draggable color stops. Real-time `QLinearGradient` preview.
Speed Ramp	RemapView keyframe transitions are linear-only. Speed changes	Bezier handles on connector lines. Smooth ease-in/out between keyframe

	abruptly at keyframe boundaries. No easing.	speeds. Presets: Linear, Ease In, Ease Out, Ease In/Out.

Conclusion

Kdenlive is software I use, love, and have already contributed to in meaningful ways. I've spent time understanding its codebase, engaging with its community, and shipping real fixes, not to pad a resume, but because I genuinely care about making it better. This project sits exactly at the intersection of my skills and my passion, and I'm ready to dedicate my full summer to it. I would be honored to work with Jean-Baptiste and the Kdenlive team to make these effect widgets a reality.