INFORME TECNICO COMPLETO

Free Animation Power

Software de Animacion 2D con Motor Hibrido Vector + Raster

Responsable del Proyecto:

Eduardo Andres Fierro Duque

Version 1.0.0 · Julio 2026

C++20 · Qt 6.5+ · CMake 3.20+

INDICE

1. INTRODUCCION

1.1 Vision General

Free Animation Power (FAP) es un software de animacion 2D de codigo abierto que combina un motor de renderizado hibrido vector + raster en una arquitectura moderna basada en C++20 y Qt 6. El proyecto nace con el objetivo de proporcionar una herramienta profesional de animacion cuadro por cuadro (frame-by-frame) con capacidades avanzadas como linea de tiempo multi-secuencia, pistas de audio y video, efectos no destructivos (line boil), deformacion de malla, sistema de nodos de composicion, y exportacion a multiples formatos de video.

El software implementa una arquitectura de 4 buffers para el pipeline de renderizado, separando estrictamente el dominio de DATOS (pixelBuffer_) del dominio de DISPLAY (backgroundCache_), lo que garantiza integridad de datos durante operaciones de dibujo, undo/redo, y reproduccion.

El proyecto ha evolucionado a traves de 9 versiones principales (v2.0 a v2.9.1) durante julio de 2026, resolviendo mas de 30 bugs criticos y añadiendo funcionalidades progresivamente. Este informe documenta de manera exhaustiva cada modulo, cada formula matematica, cada decision arquitectonica, y el camino evolutivo completo hasta el estado final.

1.2 Stack Tecnologico

Componente	Tecnologia	Version	Proposito
Lenguaje	C++20	ISO/IEC 14882:2020	Logica de negocio, motores, estructuras de datos
Framework UI	Qt 6	6.5+	Widgets, Multimedia, SVG, sistema de senales
Build System	CMake	3.20+	Compilacion multiplataforma, FetchContent
Compresion	miniz	3.0.2	Formato .fap (ZIP renombrado), compresion zlib
Testing	GoogleTest	1.14.0	160 tests unitarios y de integracion
Audio	dr_libs	Public Domain	Decodificacion nativa WAV/MP3/FLAC
Exportacion Video	FFmpeg	Externo	Encoding MP4/MOV/WebM, GIF, audio mixing
Icono Windows	Python + Win32 API	-	Inyeccion post-build de .ico en .exe
Fuente	Avenir LT Std	-	Fuente corporativa (.otf + .ttc embebida via QRC)

1.3 Objetivos del Proyecto

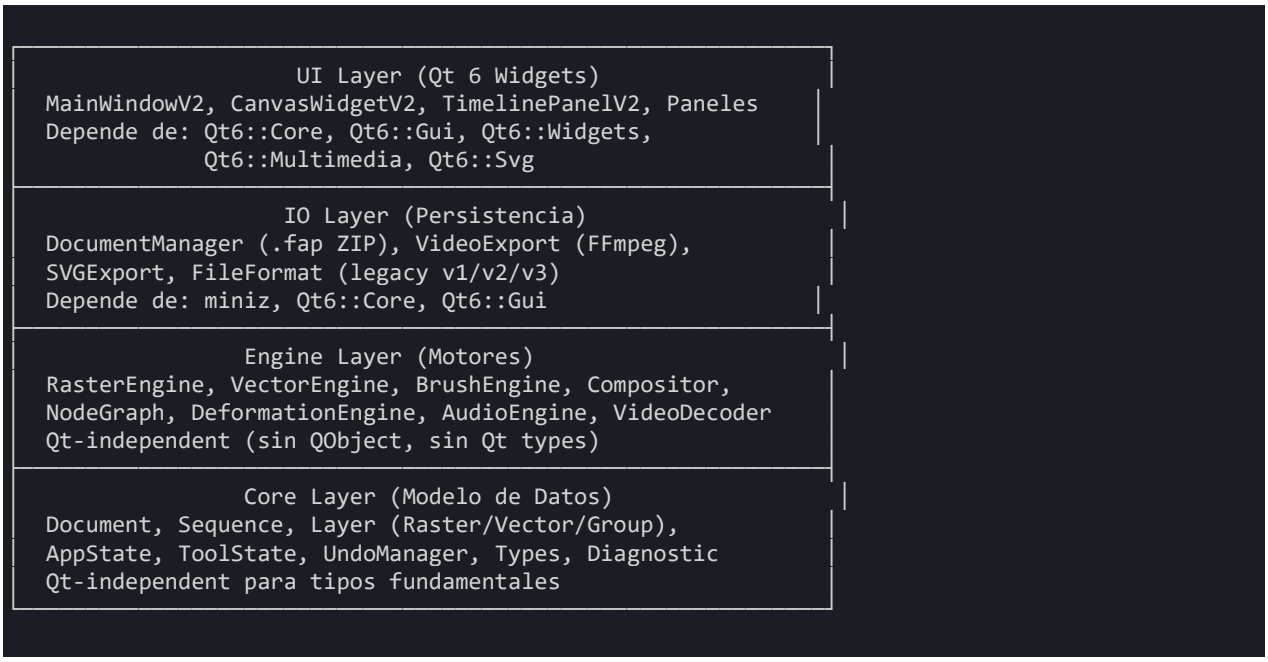
- Crear una herramienta de animacion 2D profesional, gratuita y de codigo abierto
- Implementar un motor hibrido vector + raster con renderizado en tiempo real
- Soportar flujos de trabajo de animacion cuadro por cuadro con multi-secuencia
- Integrar audio y video como pistas en la linea de tiempo
- Proveer exportacion a formatos estandar (MP4, MOV, WebM, GIF, SVG, PNG)
- Garantizar integridad de datos con sistema de undo/redo robusto
- Optimizar el pipeline de renderizado para rendimiento interactivo en 1080p+

- Mantener una arquitectura modular con separacion clara de responsabilidades

2. ARQUITECTURA GENERAL

2.1 Diagrama de Capas

El proyecto sigue una arquitectura en 4 capas bien definidas, con dependencias unidireccionales de arriba hacia abajo:



La capa Core y la capa Engine son independientes de Qt (no usan QObject, QImage internamente en sus tipos fundamentales, aunque algunas utilidades de Engine usan QImage para operaciones de imagen). La capa IO actua como puente entre el modelo de datos y el almacenamiento en disco, usando Qt para manejo de archivos y miniz para compresion. La capa UI es la unica que depende fuertemente de Qt Widgets.

2.2 Dependencias Externas

Dependencia	Tipo	Alcance	Notas
Qt 6 (Core, Gui, Widgets, Multimedia, Svg)	Framework	UI + IO	Componentes esenciales del framework
miniz 3.0.2	Libreria C	IO (formato ZIP)	FetchContent desde GitHub, compilacion estatica
GoogleTest 1.14.0	Testing	Tests	FetchContent, solo en build de tests
FFmpeg (externo)	Ejecutable	IO (export)	Llamado como proceso hijo via QProcess
dr_wav, dr_mp3, dr_flac	Headers C	Engine (audio)	Single-header, public domain, embebidos
dbghelp.dll	Windows API	Plataforma	Crash handler con stack trace simbolico

2.3 Convenciones de Codigo

- Namespace: fap (todo el codigo del proyecto)

- Headers: #pragma once (sin include guards tradicionales)
- Clases base no copiables: heredan de NonCopyable
- UIDs: atomic<uint64_t> con fetch_add(1, memory_order_relaxed)
- COW: shared_ptr<PixelBuffer> con ensureUnique()
- Configuracion: constantes estaticas constexpr (kMaxDim, kGuardBand, kGrowthPad)
- Nomenclatura: CamelCase para clases, snake_case_ para miembros privados
- Tema oscuro: #252830 (paneles), #000000 (toolbar), #FF4800 (acento)
- Idioma: codigo y comentarios en ingles, documentacion mixta ingles/espanol

3. MODELO DE DATOS (CORE)

3.1 Tipos y Estructuras Fundamentales

El archivo src/core/types.hpp define los tipos atomicos del sistema, todos dentro del namespace fap. Estos tipos son Qt-independent y forman la base sobre la cual se construye todo el motor.

3.1.1 Enumeraciones Principales

Enum	Valores	Proposito
LayerType	Raster, Vector, Group, Audio, Camera	Tipo de capa en el sistema
BlendMode	Normal, Multiply, Screen, Overlay, Add, Subtract, Darken, Lighten, ColorBurn, ColorDodge, SoftLight, HardLight	12 modos de mezcla de capas
BrushMode	Raster, Vector, Hybrid	Modo de operacion del pincel
ToolType	Brush, Eraser, ColorPicker, Fill, Text, Line, Rectangle, Ellipse, Move, Select, Hand, PencilRetouch, RulerLine, RulerEllipse, DeformMesh, TweenEdit	17 herramientas disponibles
FillType	Solid, Fabric, Ramp	Tipos de relleno: solido, tela, degradado
LineStyle	Solid, Tapered, Dashed, Dotted	Estilos de linea

3.1.2 Estructura Color

El color se almacena en punto flotante normalizado [0,1] con 4 canales (RGBA), usando premultiplicacion de alpha para operaciones de composicion:

```
struct Color {
    float r = 0.0f;  // Rojo    [0.0, 1.0]
    float g = 0.0f;  // Verde   [0.0, 1.0]
    float b = 0.0f;  // Azul    [0.0, 1.0]
    float a = 1.0f;  // Alpha   [0.0, 1.0]

    static Color fromRGBA(uint8_t r, uint8_t g, uint8_t b, uint8_t a = 255) {
        return { r / 255.0f, g / 255.0f, b / 255.0f, a / 255.0f };
    }

    Color premultiplied() const {
        return { r * a, g * a, b * a, a };
        // R_premul = R * A,  G_premul = G * A,  B_premul = B * A
    }
};
```

3.1.3 Estructura Vec2

Vector bidimensional con operaciones algebraicas completas:

```
struct Vec2 {
    float x, y;

    Vec2 operator+(const Vec2& o) const { return { x + o.x, y + o.y }; }
```

```
    Vec2 operator-(const Vec2& o) const { return { x - o.x, y - o.y }; }
    Vec2 operator*(float s) const { return { x * s, y * s }; }
    Vec2 operator/(float s) const { return { x / s, y / s }; }

    float length() const { return std::sqrt(x * x + y * y); }     // ||v||
    float lengthSquared() const { return x * x + y * y; }         // ||v||^2
    Vec2 normalized() const { /* v / ||v||, guard: len < 1e-8 */ }
    float dot(const Vec2& o) const { return x * o.x + y * o.y; } // a . b
    float cross(const Vec2& o) const { return x * o.y - y * o.x; } // a x b

    static Vec2 lerp(const Vec2& a, const Vec2& b, float t) {
        return { a.x + (b.x - a.x) * t, a.y + (b.y - a.y) * t };
    }
    static float distance(const Vec2& a, const Vec2& b) {
        return (a - b).length();
    }
};
```

3.1.4 Estructura Rect

Rectangulo definido por esquina superior-izquierda (x,y) y dimensiones (width, height):

```
struct Rect {
    float x, y, width, height;

    float left()   const { return x; }
    float right()  const { return x + width; }
    float top()    const { return y; }
    float bottom() const { return y + height; }

    bool contains(float px, float py) const {
        return px >= x && px <= x + width && py >= y && py <= y + height;
    }

    Rect intersect(const Rect& other) const {
        float nx = std::max(x, other.x);
        float ny = std::max(y, other.y);
        float nw = std::min(right(), other.right()) - nx;
        float nh = std::min(bottom(), other.bottom()) - ny;
        return { nx, ny, std::max(0.0f, nw), std::max(0.0f, nh) };
    }

    Rect expanded(float margin) const {
        return { x - margin, y - margin, width + margin * 2, height + margin * 2 };
    }
};
```

3.1.5 Estructura StrokePoint y Size

```
struct StrokePoint {
    Vec2 position;
    float pressure = 1.0f;  // 0.0 - 1.0 (tablet pressure)
    float tilt_x = 0.0f;    // X tilt angle
    float tilt_y = 0.0f;    // Y tilt angle
    float rotation = 0.0f;  // Stylus rotation
    float timestamp = 0.0f; // Milliseconds
};

struct Size {
    int width = 0;
```

```
    int height = 0;
    bool isValid() const { return width > 0 && height > 0; }
};
```

3.2 Sistema de Capas (Layer Hierarchy)

El sistema de capas implementa una jerarquia de herencia con tres tipos concretos de capa: RasterLayer (mapa de bits), VectorLayer (trazos vectoriales), y GroupLayer (capa contenedora de subcapas).

3.2.1 Clase Base Layer

```
class Layer : public NonCopyable {
protected:
    LayerUid uid_;                    // Identificador unico global (atomic uint64_t)
    std::string name_;
    LayerType type_;
    bool visible_ = true;
    float opacity_ = 1.0f;           // [0.0, 1.0]
    BlendMode blend_mode_ = BlendMode::Normal;
    bool locked_ = false;

public:
    virtual std::unique_ptr<Layer> clone() const = 0;
    LayerUid uid() const { return uid_; }
    // Getters/Setters para name, visible, opacity, blendMode, locked
};
```

Cada capa tiene un UID unico generado atomicamente:

```
static std::atomic<LayerUid> g_nextLayerUid{1};

LayerUid generateLayerUid() {
    return g_nextLayerUid.fetch_add(1, std::memory_order_relaxed);
}
```

3.2.2 RasterLayer — Capa de Mapa de Bits

RasterLayer es la capa principal de dibujo. Almacena pixeles en formato ARGB32 premultiplicado dentro de un PixelBuffer compartido via shared_ptr (Copy-on-Write).

```
class RasterLayer : public Layer {
    int width_;
    int height_;
    int originX_ = 0;                          // Offset X del buffer en espacio canvas
    int originY_ = 0;                          // Offset Y del buffer en espacio canvas
    std::shared_ptr<PixelBuffer> pixelBuffer_; // COW: compartido entre clones
    uint64_t buffer_epoch_ = 0;                // Contador de version (invalida caches)
    bool hasContent_ = false;                  // O(1): true si hay pixeles no transparentes

    static constexpr int kMaxDim = 10000;      // Dimension maxima: 10,000px
    static constexpr int kGrowthPad = 512;     // Expansion proactiva: 512px
    static constexpr int kGuardBand = 2;       // Margen de seguridad: 2px (feathering)

    // Metodos clave:
    void ensureUnique();                       // COW: copia si uso compartido > 1
    void ensureContains(int x, int y, int w, int h, bool pad = true);
```

```
    void setPixel(int x, int y, const Color& color);
    void blendPixel(int x, int y, const Color& color);
    void clear(const Color& color = {0,0,0,0});
    void shareDataFrom(const RasterLayer& other); // Comparte buffer (COW)
};
```

3.2.3 PixelBuffer y Copy-on-Write (COW)

El sistema COW (Copy-on-Write) permite que multiples capas compartan el mismo buffer de pixeles sin costo de memoria adicional, hasta que una de ellas intente modificar los pixeles. En ese momento, ensureUnique() crea una copia independiente.

```
struct PixelBuffer {
    std::vector<uint32_t> pixels;  // ARGB32 premultiplicado
};

void RasterLayer::ensureUnique() {
    if (pixelBuffer_.use_count() <= 1) return;  // Ya es unico
    auto newBuf = std::make_shared<PixelBuffer>(pixelBuffer_->pixels);  // Deep copy
    pixelBuffer_ = std::move(newBuf);
}

bool RasterLayer::isBufferShared() const {
    return pixelBuffer_.use_count() > 1;
}
```

Este mecanismo es fundamental para: compartir datos entre frames identicos (0 bytes extra), implementar clone() eficiente, y permitir que copyLayerToAllFrames() comparta datos temporalmente antes de hacer deep copy via ensureUnique().

3.2.4 Operaciones de Pixel

Acceso indexado con soporte de offset (originX_/originY_):

```
size_t RasterLayer::indexAt(int x, int y) const {
    int bx = x - originX_;
    int by = y - originY_;
    if (bx < 0 || bx >= width_ || by < 0 || by >= height_)
        return static_cast<size_t>(-1);  // Fuera de limites
    return static_cast<size_t>(by) * static_cast<size_t>(width_)
         + static_cast<size_t>(bx);
}
```

setPixel() — Escritura directa con premultiplicacion:

```
void RasterLayer::setPixel(int x, int y, const Color& color) {
    ensureUnique();  // COW: copia si es compartido
    size_t idx = indexAt(x, y);
    if (idx == static_cast<size_t>(-1)) return;

    float a = std::clamp(color.a, 0.0f, 1.0f);
    // Premultiplicacion: R' = R * A,  G' = G * A,  B' = B * A
    uint8_t r = static_cast<uint8_t>(std::clamp(color.r * a, 0.0f, 1.0f) * 255.0f);
    uint8_t g = static_cast<uint8_t>(std::clamp(color.g * a, 0.0f, 1.0f) * 255.0f);
    uint8_t b = static_cast<uint8_t>(std::clamp(color.b * a, 0.0f, 1.0f) * 255.0f);
    uint8_t alpha = static_cast<uint8_t>(a * 255.0f);
```

```
    // Formato ARGB32: 0xAARRGGBB (Qt native)
    pixelBuffer_->pixels[idx] = (static_cast<uint32_t>(b))
                              | (static_cast<uint32_t>(g) << 8)
                              | (static_cast<uint32_t>(r) << 16)
                              | (static_cast<uint32_t>(alpha) << 24);
    if (alpha > 0) hasContent_ = true;
    ++buffer_epoch_;  // Invalida caches de display
}
```

blendPixel() — Source Over (Porter-Duff):

```
void RasterLayer::blendPixel(int x, int y, const Color& color) {
    ensureUnique();
    size_t idx = indexAt(x, y);
    if (idx == static_cast<size_t>(-1)) return;

    uint8_t sr, sg, sb, sa;  // Color fuente (source)
    // ... conversion de float a uint8 ...

    float srcA = static_cast<float>(sa) / 255.0f;
    float invSrcA = 1.0f - srcA;

    uint32_t dst = pixelBuffer_->pixels[idx];
    uint8_t db, dg, dr, da;  // Color destino (destination)

    // Formula Porter-Duff Source Over:
    // C_out = C_src + C_dst * (1 - alpha_src)
    uint8_t outR = static_cast<uint8_t>(sr * srcA + dr * invSrcA + 0.5f);
    uint8_t outG = static_cast<uint8_t>(sg * srcA + dg * invSrcA + 0.5f);
    uint8_t outB = static_cast<uint8_t>(sb * srcA + db * invSrcA + 0.5f);
    // Alpha: A_out = A_src + A_dst * (1 - A_src)
    uint8_t outA = static_cast<uint8_t>(sa + da * invSrcA + 0.5f);

    pixelBuffer_->pixels[idx] = /* empaquetar ARGB32 */;
    if (outA > 0) hasContent_ = true;
    ++buffer_epoch_;
}
```

clear() — Limpieza de buffer:

```
void RasterLayer::clear(const Color& color) {
    ensureUnique();
    // Premultiplica y llena todo el buffer con el color
    uint32_t val = /* empaquetar ARGB32 premultiplicado */;
    std::fill(pixelBuffer_->pixels.begin(), pixelBuffer_->pixels.end(), val);
    hasContent_ = false;  // Resetea flag
    ++buffer_epoch_;
}
```

3.2.5 ensureContains() — Gestion Dinamica del Buffer

El buffer de pixeles de una RasterLayer es de tamano dinamico. Comienza en el tamano del canvas, pero se expande automaticamente cuando se dibuja fuera de los limites actuales. El algoritmo de ensureContains() maneja la expansion y reubicacion de pixeles existentes.

Algoritmo de Expansion:

```
void RasterLayer::ensureContains(int x, int y, int w, int h, bool pad) {
```

```
    // Paso 1: Calcular margenes
    int guard = pad ? kGuardBand : 0;   // 2px para feathering de pincel
    int grow  = pad ? kGrowthPad : 0;   // 512px para expansion proactiva

    x -= guard;  y -= guard;
    w += guard * 2;  h += guard * 2;

    // Paso 2: Calcular nueva region requerida
    int newOriginX = std::min(originX_, x);
    int newOriginY = std::min(originY_, y);
    int newRight   = std::max(originX_ + width_, x + w);
    int newBottom  = std::max(originY_ + height_, y + h);

    // Paso 3: Expansion proactiva en direcciones que crecieron
    if (newOriginX < originX_) newOriginX -= grow;
    if (newOriginY < originY_) newOriginY -= grow;
    if (newRight  > originX_ + width_)  newRight  += grow;
    if (newBottom > originY_ + height_) newBottom += grow;

    // Paso 4: Limites de seguridad (kMaxDim = 10000)
    // Clampear a [-kMaxDim, +kMaxDim] desde el origen ...

    // Paso 5: Si no hay cambio, retornar
    if (newOriginX == originX_ && newOriginY == originY_ &&
        newW == width_ && newH == height_) return;

    // Paso 6: Relocalizar pixeles existentes
    ensureUnique();
    std::vector<uint32_t> oldPixels = std::move(pixelBuffer_->pixels);
    // ... actualizar dimensiones ...
    pixelBuffer_->pixels.assign(newPixels, 0);
    relocatePixels(oldPixels, oldW, oldH, oldOX, oldOY,
                   newW, newH, newOriginX, newOriginY);
}
```

Algoritmo de Reubicacion (relocatePixels):

La reubicacion de pixeles calcula la interseccion entre el area antigua y la nueva, y copia solo los pixeles que caen dentro de la region comun:

```
void RasterLayer::relocatePixels(oldPixels, oldW, oldH, oldOX, oldOY,
                                 newW, newH, newOX, newOY) {
    // Interseccion de rectangulos en espacio canvas
    int copyLeft   = std::max(oldOX, newOX);
    int copyTop    = std::max(oldOY, newOY);
    int copyRight  = std::min(oldOX + oldW, newOX + newW);
    int copyBottom = std::min(oldOY + oldH, newOY + newH);

    if (copyRight <= copyLeft || copyBottom <= copyTop) return;

    int copyWidth = copyRight - copyLeft;
    for (int cy = copyTop; cy < copyBottom; ++cy) {
        int srcY = cy - oldOY;
        int dstY = cy - newOY;
        // Copiar fila: srcRow[srcStartX..] -> dstRow[dstStartX..]
        std::copy(srcRow + srcStartX, srcRow + srcStartX + copyWidth,
                  dstRow + dstStartX);
    }
}
```

3.2.6 hasContent_ — Deteccion O(1) de Contenido

El flag hasContent_ permite saber si una capa tiene pixeles no transparentes en tiempo O(1), sin necesidad de escanear el buffer de pixeles. Este flag es crucial para el renderizado de celdas en la linea de tiempo (SequenceTrackWidget).

Operacion	hasContent_ resultante	Notas
setPixel(alpha > 0)	true	Seteado en hot path de pixel
blendPixel(outA > 0)	true	Seteado en hot path de pixel
clear()	false	Buffer lleno de transparente
clone()	Preservado del original	Copia fiel del estado
commitStroke()	true	Seteado al commit (no en per-dab)
doText() / commitMove() / commitSelection()	true	Seteado en commit de operacion
shareDataFrom()	No afectado	El flag se copia explicitamente

Es importante notar que hasContent_ NO se actualiza en las rutas calientes de setPixel/blendPixel durante el dibujo con pincel. Esto se debe a que el motor de pinceles escribe directamente al pixelSpan() via QPainter, evitando esas funciones. En su lugar, el flag se establece en el momento de commit del stroke (CanvasWidgetV2::commitStroke()).

3.2.7 VectorLayer — Capa Vectorial

```
class VectorLayer : public Layer {
    std::vector<VectorStroke> strokes_;

    void addStroke(const VectorStroke& stroke);
    void removeStroke(int index);
    void clearStrokes();
    VectorStroke& strokeAt(int index);
    size_t strokeCount() const;
};

struct VectorStroke {
    BezierPath path;        // Curva Bezier (multiples segmentos cubicos)
    Color color;
    float width = 2.0f;
    float opacity = 1.0f;
    BlendMode blend = BlendMode::Normal;
};
```

3.2.8 GroupLayer — Capa Contenedora

GroupLayer actua como nodo raiz del arbol de capas. Cada frame de animacion tiene su propio GroupLayer que contiene las capas hijas (RasterLayer o VectorLayer). La estructura de datos interna es un std::deque para soportar insercion/eliminacion eficiente en ambos extremos.

```
class GroupLayer : public Layer {
    std::deque<std::unique_ptr<Layer>> layers_;
```

```
    void addLayer(std::unique_ptr<Layer> layer);
    void removeLayer(int index);
    void moveLayer(int fromIndex, int toIndex);
    Layer* duplicateLayer(int index);       // Clona y renombra "copy"
    Layer* layerAt(int index);
    Layer* layerByUid(LayerUid uid);       // Busqueda por UID
    int indexOfLayer(const Layer* layer) const;
    size_t layerCount() const;
};
```

3.3 Secuencias y Frames

3.3.1 Estructura Sequence

Una secuencia representa una linea de tiempo independiente con sus propios frames, configuracion de FPS, work area, y marcadores. Cada secuencia puede contener multiples secuencias dentro del mismo documento.

```
class Sequence : public NonCopyable {
    // Identidad
    SequenceUid uid_;
    std::string name_;
    bool visible_ = true;
    float opacity_ = 1.0f;
    bool locked_ = false;

    // Timeline
    int current_frame_ = 0;
    int total_frames_ = 1;       // Total de frames con datos (nunca se reduce al ocultar)
    int fps_ = 24;
    bool playing_ = false;
    bool looping_ = false;

    // Work Area (region de reproduccion/loop)
    int workAreaStart_ = 0;
    int workAreaEnd_ = 0;        // 0 = usar durationFrames_ como fin
    int durationFrames_ = 1;     // Frames visibles (<= total_frames_)

    // Efectos
    LineBoil lineBoil_;

    // Datos
    std::map<int, std::unique_ptr<GroupLayer>> frames_;  // Sparse: solo frames creados
    std::vector<Marker> markers_;                         // Marcadores estilo AE
    UndoManager undo_manager_;
    FrameCallback on_frame_changed_;                      // Callback de cambio de frame
};
```

3.3.2 Work Area y Duration

El concepto de Work Area (area de trabajo) define una region dentro de la secuencia donde ocurre la reproduccion y el looping. Esta delimitada por workAreaStart_ (in point) y workAreaEnd_ (out point). La duracion visible (durationFrames_) puede ser menor que el total de frames con datos (totalFrames_), permitiendo ocultar frames sin eliminarlos.

```
int Sequence::effectiveWorkAreaEnd() const {
```

```
    if (workAreaEnd_ > 0) {
        return std::min(workAreaEnd_, durationFrames_);
    }
    return durationFrames_;
}

int Sequence::advanceFrame() {
    int waStart = std::clamp(workAreaStart_, 0, durationFrames_ - 1);
    int waEnd = effectiveWorkAreaEnd();

    if (waEnd <= waStart) waEnd = durationFrames_;

    if (current_frame_ >= waEnd - 1) {
        setCurrentFrame(waStart);    // Wrap al inicio del work area
    } else {
        setCurrentFrame(current_frame_ + 1);
    }
    return current_frame_;
}
```

Los metodos nextFrame(), prevFrame(), y setCurrentFrame() estan vinculados a durationFrames_ (frames visibles), no a totalFrames_ (frames con datos). Esto implementa el sistema de ocultacion no destructiva de frames (v2.7):

Operacion	Efecto en durationFrames_	Efecto en totalFrames_	Datos
Boton +	Aumenta en 1 (revela frame oculto si existe)	Sin cambio	Preservados
Boton -	Reduce en 1 (oculta ultimo frame)	Sin cambio	Preservados
Delete Frame (menu contextual)	Reduce en 1	Reduce en 1	Eliminados permanentemente
setTotalFrames(count)	Sin cambio directo	max(1, count)	Nuevos frames = sin datos

3.3.3 Sistema de Frames Sparse

Los frames se almacenan en un std::map<int, unique_ptr<GroupLayer>>, lo que significa que solo los frames que han sido modificados ocupan memoria. Un frame que nunca ha sido dibujado no existe en el mapa y, por tanto, no consume RAM.

```
GroupLayer& Sequence::rootLayerForFrame(int frame) {
    auto it = frames_.find(frame);
    if (it == frames_.end()) {
        // Crear frame bajo demanda con capa por defecto "Layer 1"
        auto root = std::make_unique<GroupLayer>("Root");
        ensureDefaultLayer(*root);  // Añade RasterLayer "Layer 1" vacia
        auto [inserted, _] = frames_.emplace(frame, std::move(root));
        return *inserted->second;
    }
    return *it->second;
}
```

3.3.4 copyLayerToAllFrames()

Esta funcion (v2.6) copia el contenido y propiedades de una capa desde un frame fuente a todos los demas frames de la secuencia. Es util para propagar un fondo o un elemento fijo a traves de toda la animacion.

```
void Sequence::copyLayerToAllFrames(int sourceFrame, int layerIndex) {
    GroupLayer& srcRoot = rootLayerForFrame(sourceFrame);
    const Layer* srcLayer = srcRoot.layerAt(layerIndex);

    for (int f = 0; f < total_frames_; ++f) {
        if (f == sourceFrame) continue;
        GroupLayer& dstRoot = rootLayerForFrame(f);

        // Asegurar que el frame destino tenga suficientes capas
        while (dstRoot.layerCount() <= layerIndex)
            dstRoot.addLayer(srcLayer->clone());

        Layer* dstLayer = dstRoot.layerAt(layerIndex);
        // Copiar metadata: nombre, visibilidad, opacidad, blend, lock
        dstLayer->setName(srcLayer->name());
        // ...

        if (srcLayer->type() == LayerType::Raster) {
            // COW: compartir buffer, luego deep copy para independencia
            dstRl->shareDataFrom(*srcRl);
            dstRl->ensureUnique();     // Deep copy del buffer compartido
            dstRl->setHasContent(srcRl->hasContent());
        }
    }
}
```

3.3.5 Marcadores (Timeline Markers)

Los marcadores (v2.7) proporcionan puntos de referencia visuales en la linea de tiempo, al estilo de Adobe After Effects. Soportan marcadores de un solo punto y marcadores de rango (con duracion).

```
struct Marker {
    int frame = 0;              // Posicion 0-based en la timeline
    int duration = 0;           // 0 = punto simple, >0 = marcador de rango
    std::string comment;        // Titulo (visible en la regla)
    std::string detail;         // Notas largas (dialogo de propiedades)
    int colorLabel = 0;         // 0-8: None, Red, Orange, Yellow, Green,
                                //       Cyan, Blue, Purple, Magenta
    int64_t uid = 0;            // Identificador unico

    bool isDuration() const { return duration > 0; }
    int outPoint() const { return frame + duration; }
};
```

Restriccion de unicidad: solo puede existir un marcador por posicion de frame. Al añadir o mover un marcador a un frame ya ocupado, el marcador existente es reemplazado.

3.3.6 LineBoil — Efecto de Linea Vibrante

LineBoil (v2.9) es un efecto no destructivo que aplica un desplazamiento pseudo-aleatorio a los pixeles de cada frame, creando el aspecto organico de "linea vibrante" caracteristico de la animacion tradicional.

```
struct LineBoil {
    bool enabled = false;
    float strength = 2.0f;   // Desplazamiento maximo en pixeles [0.0, 10.0]
    int seed = 42;           // Semilla para el generador de ruido
};
```

3.4 Documento

La clase Document es el contenedor raiz del modelo de datos. Almacena las secuencias, las pistas de audio/video, y el estado de modificacion.

3.5 AppState y Pipeline de Senales

AppState actua como singleton central que posee todas las dependencias del sistema: Document, ToolState, AudioEngine, y los diversos motores. Implementa el patron de senales Qt para notificar cambios a la UI de forma reactiva.

```
class AppState : public QObject {
    Q_OBJECT
    std::unique_ptr<Document> document_;
    std::unique_ptr<ToolState> tool_state_;
    std::unique_ptr<AudioEngine> audio_engine_;
    std::unique_ptr<RulerTool> ruler_tool_;
    std::unique_ptr<TweenEngine> tween_engine_;
    std::unique_ptr<DeformationEngine> deformation_engine_;
    std::unique_ptr<FrameThumbnailCache> thumbnail_cache_;
    std::unique_ptr<PencilRetouchEngine> pencil_retouch_;

signals:
    void activeLayerIndexChanged(int);
    void currentFrameChanged(int);
    void documentChanged();           // Senal maestra: invalida caches
    void documentModifiedChanged(bool);
    void canvasSizeChanged(int, int);
    void activeSequenceChanged(int);
};
```

El flujo de datos es unidireccional: la UI nunca modifica el modelo directamente. En su lugar, llama a metodos de AppState que mutan el estado y emiten senales. Los widgets se conectan a estas senales para actualizar su representacion visual. Esto garantiza que todos los widgets esten siempre sincronizados con el estado real.

3.6 Sistema de Undo/Redo

El sistema de undo implementa un patron Command con dos stacks: undoStack_ (comandos ejecutados) y redoStack_ (comandos deshechos). Cada secuencia tiene su propio UndoManager.

```
class UndoCommand {
public:
    virtual ~UndoCommand() = default;
    virtual void undo() = 0;
    virtual void redo() = 0;
};

class UndoManager {
    static constexpr size_t kMaxEntries = 128;

    void pushCommand(std::unique_ptr<UndoCommand> cmd);
    void pushApplied(std::unique_ptr<UndoCommand> cmd);  // + redo()
    void undo();  // Pop undoStack, push redoStack, cmd->undo()
    void redo();  // Pop redoStack, push undoStack, cmd->redo()
    void clear();
};
```

El comando principal de undo para operaciones de dibujo es pushFullLayerUndo(), que captura un snapshot QImage de la capa activa antes de la modificacion:

```
void CanvasWidgetV2::pushFullLayerUndo(RasterLayer* layer,
                                       const QImage& beforeSnap,
                                       bool hadContentBefore) {
    auto cmd = std::make_unique<LayerModifyCommand>(
        layer, beforeSnap, hadContentBefore);
    appState_->activeSequence().undoManager().pushApplied(std::move(cmd));
}
```

3.7 Gestion de Memoria RAM

3.7.1 Modelo COW y Ahorro de Memoria

El sistema Copy-on-Write basado en shared_ptr<PixelBuffer> permite que multiples frames compartan el mismo buffer de pixeles. El costo de memoria es:

mem_total = base + sigma(frames_modificados) x (W x H x 4) bytes

Donde base es la memoria del modelo de datos (~50MB para un documento complejo), W y H son las dimensiones del canvas, y 4 bytes por pixel (ARGB32). Un frame no modificado comparte el buffer con su frame fuente y no consume memoria adicional de pixeles.

3.7.2 Almacenamiento Sparse de Frames

Los frames se almacenan en un std::map<int, unique_ptr<GroupLayer>>. Solo los frames que han sido accedidos (dibujados o modificados) existen en el mapa. Un proyecto con 1000 frames declarados pero solo 50 dibujados ocupa memoria para 50 frames, no para 1000.

mem_frames = N_modificados x (overhead_GroupLayer + sigma(capas) x W x H x 4)

3.7.3 Limites de Seguridad

Constante	Valor	Proposito
kMaxDim	10,000	Dimension maxima de buffer en pixeles (100M pixeles max)
kGrowthPad	512	Expansion proactiva del buffer al dibujar en bordes

kGuardBand	2	Margen para feathering de pinceles (1px por lado)
kMaxPixels	100,000,000	Maximo de pixeles en un buffer (10,000 x 10,000)
kMaxEntries (Undo)	128	Maximo de entradas en stack de undo
kMaxCacheFrames (Video)	50	Maximo de frames decodificados en cache LRU
kMaxBrushes (ABR)	512	Maximo de pinceles importables de archivo ABR
kMaxBrushSize	2500	Tamano maximo de pincel ABR en pixeles

3.7.4 Huella de Memoria por Escenario

Escenario	Componentes	Memoria Estimada
Proyecto vacio 1080p	Documento base + 1 secuencia + UI	~80-120 MB
Proyecto tipico 1080p (24fps, 3s)	50 frames dibujados, 5 capas, undo stack	~300-500 MB
Proyecto complejo 1080p + video	100+ frames, 10 capas, 2 video tracks 1080p, undo	~700-1000 MB
Proyecto 4K	3840x2160, 5 capas, 24 frames	~1.2-2.0 GB

4. MOTOR DE RENDERIZADO (ENGINE)

4.1 Motor Raster

El motor raster maneja operaciones de bajo nivel sobre buffers de pixeles. Incluye el RasterEngine (orquestador) y utilidades de renderizado.

4.1.1 Formato de Pixel ARGB32 Premultiplicado

Todos los pixeles en el sistema usan el formato ARGB32 premultiplicado (0xAARRGGBB), que es el formato nativo de QImage::Format_ARGB32_Premultiplied. Este formato almacena los canales de color ya multiplicados por el alpha, lo que simplifica las operaciones de composicion.

```
// Empaquetado: 0xAARRGGBB (Qt native, little-endian byte order)
uint32_t pixel = B | (G << 8) | (R << 16) | (A << 24);

// Desempaquetado:
uint8_t A = (pixel >> 24) & 0xFF;
uint8_t R = (pixel >> 16) & 0xFF;
uint8_t G = (pixel >> 8)  & 0xFF;
uint8_t B = pixel         & 0xFF;

// Conversion a float [0,1]:
float r = R / 255.0f, g = G / 255.0f, b = B / 255.0f, a = A / 255.0f;
```

4.1.2 RasterEngine — Orquestador

RasterEngine envuelve un QImage interno y proporciona metodos para leer/escribir pixeles, redimensionar, y aplicar efectos. Actua como el destino de composicion en el pipeline de display.

4.2 Utilidades de Blend (12 Modos)

El archivo src/engine/common/blend_utils.hpp implementa unpackPixel/packPixel para el formato 0xAARRGGBB y la funcion blendChannel() con los 12 modos de mezcla. La composicion final usa Source Over de Porter-Duff:

C_out = alpha_fuente x f(C_fuente, C_destino) + (1 - alpha_fuente) x C_destino

4.2.1 Tabla Completa de Modos de Blend

Modo	Formula por canal f(a,b)	Efecto visual
Normal	f(a,b) = a	Reemplazo directo (segun alpha)
Multiply	f(a,b) = a x b	Oscurece, como superponer transparencias
Screen	f(a,b) = 1 - (1-a)(1-b)	Aclara, inverso de multiply
Overlay	f = 2ab si b<0.5, f = 1-2(1-a)(1-b) si b>=0.5	Combina multiply y screen
Add	f(a,b) = min(1, a+b)	Suma lineal, blanquea rapido
Subtract	f(a,b) = max(0, b-a)	Resta, oscurece
Darken	f(a,b) = min(a, b)	Toma el canal mas oscuro

Lighten	f(a,b) = max(a, b)	Toma el canal mas claro
ColorBurn	f = 1 - min(1, (1-b)/a) si a>0	Quemado de color (oscurece contraste)
ColorDodge	f = min(1, b/(1-a)) si a<1	Sobreexposicion (aclara contraste)
SoftLight	Ver ecuacion completa abajo	Luz suave (similar a overlay pero mas sutil)
HardLight	f = 2ab si a<0.5, f = 1-2(1-a)(1-b) si a>=0.5	Luz fuerte

Formula completa de SoftLight:

```
float blendChannel_SoftLight(float src, float dst) {
    if (src <= 0.5f)
        return dst - (1.0f - 2.0f * src) * dst * (1.0f - dst);
    if (dst <= 0.25f)
        return dst + (2.0f * src - 1.0f)
               * (((16.0f * dst - 12.0f) * dst + 4.0f) * dst - dst);
    return dst + (2.0f * src - 1.0f) * (std::sqrt(dst) - dst);
}
```

Implementacion de blendPixel():

```
inline void blendPixel(uint32_t* pixels, int width, int height,
                       int x, int y, float r, float g, float b,
                       float a, BlendMode mode) {
    if (x < 0 || x >= width || y < 0 || y >= height || a <= 0.0f) return;

    size_t idx = y * width + x;
    FloatPixel dst = unpackPixel(pixels[idx]);  // ARGB -> float
    FloatPixel src = {r, g, b, a};

    float blendedR = blendChannel(src.r, dst.r, mode);
    float blendedG = blendChannel(src.g, dst.g, mode);
    float blendedB = blendChannel(src.b, dst.b, mode);

    float finalA = src.a + dst.a * (1.0f - src.a);  // Porter-Duff alpha
    float invA = 1.0f - src.a;

    FloatPixel result;
    result.r = src.a * blendedR + invA * dst.r;
    result.g = src.a * blendedG + invA * dst.g;
    result.b = src.a * blendedB + invA * dst.b;
    result.a = finalA;

    pixels[idx] = packPixel(result);  // float -> ARGB
}
```

4.3 Efecto Line Boil

El efecto Line Boil (v2.9) aplica un desplazamiento pseudo-aleatorio a cada pixel de la imagen, creando un efecto de "linea vibrante" no destructivo. El algoritmo opera en el dominio de DISPLAY (sobre una copia QImage), nunca modificando el pixelBuffer_ original.

4.3.1 Funcion Hash Espacial

El corazon del algoritmo es una funcion hash determinista que genera valores pseudo-aleatorios dependientes de la posicion (x,y), el numero de frame, y una semilla:

$$h(x,y,f,s) = (x \cdot 374761393 + y \cdot 668265263 + f \cdot 1274126177 + s \cdot 3443311159) \bmod 2^{32}$$

$$h = (h \text{ XOR } (h >> 13)) \times 1274126177$$

$$h = h \text{ XOR } (h >> 16)$$

Propiedades de esta funcion hash:

- Determinista: misma entrada siempre produce misma salida
- Pseudo-aleatoria: pequenos cambios en x,y producen grandes cambios en h
- Dependiente del frame: diferente para cada frame, creando vibracion
- Suave entre celdas: la interpolacion bilineal suaviza la transicion

4.3.2 Algoritmo de Desplazamiento Bilineal

El algoritmo divide la imagen en celdas de 8x8 pixeles. Para cada pixel, calcula un vector de desplazamiento (dx, dy) interpolado bilinealmente entre las 4 esquinas de la celda:

```
void applyLineBoil(QImage& image, int frame, float strength, int seed) {
    const int cellSize = 8;  // Tamano de celda para ruido
    QImage src = image.copy();  // Trabajar sobre copia (no destructivo)

    for (int y = 0; y < h; ++y) {
        // Indices de celda y posicion fraccional
        int cy0 = y / cellSize, cy1 = cy0 + 1;
        float fy = (y % cellSize - halfCell) / cellSize;  // [-0.5, 0.5] normalizado

        for (int x = 0; x < w; ++x) {
            int cx0 = x / cellSize, cx1 = cx0 + 1;
            float fx = (x % cellSize - halfCell) / cellSize;

            // Calcular desplazamiento para las 4 esquinas de la celda
            float dx00, dy00 = hash_to_offset(hash31(cx0, cy0, frame, seed), strength);
            float dx10, dy10 = hash_to_offset(hash31(cx1, cy0, frame, seed), strength);
            float dx01, dy01 = hash_to_offset(hash31(cx0, cy1, frame, seed), strength);
            float dx11, dy11 = hash_to_offset(hash31(cx1, cy1, frame, seed), strength);

            // Interpolacion bilineal del desplazamiento
            float tdx = dx00*(1-fx)*(1-fy) + dx10*fx*(1-fy)
                      + dx01*(1-fx)*fy    + dx11*fx*fy;
            float tdy = dy00*(1-fx)*(1-fy) + dy10*fx*(1-fy)
                      + dy01*(1-fx)*fy    + dy11*fx*fy;

            // Aplicar desplazamiento (sampleo de textura inverso)
            int sx = clamp(x + tdx, 0, w-1);
            int sy = clamp(y + tdy, 0, h-1);
            dstRow[x] = srcRow[sx];  // Copiar pixel de la posicion desplazada
        }
    }
}
```

La transformacion de hash a offset convierte los 16 bits bajos del hash en un desplazamiento en el rango [-strength, +strength]:

$$offset = ((hash\_byte / 255.0) - 0.5) \times strength \times 2.0$$

4.4 Motor Vectorial (Curvas de Bezier)

4.4.1 Representacion Matematica

El motor vectorial utiliza curvas de Bezier cubicas como primitiva fundamental. Una curva de Bezier cubica esta definida por 4 puntos de control P0, P1, P2, P3:

```
B(t) = (1-t)^3 x P0 + 3(1-t)^2 x t x P1 + 3(1-t) x t^2 x P2 + t^3 x P3,   t in [0,1]
```

La tangente en cualquier punto t se calcula como:

```
B'(t) = 3(1-t)^2 x (P1-P0) + 6(1-t)t x (P2-P1) + 3t^2 x (P3-P2)
```

4.4.2 Aplanado Recursivo (De Casteljau)

Para rasterizar una curva Bezier, se subdivide recursivamente hasta que cada segmento sea "suficientemente plano". El criterio de planitud (flatness) evalua la desviacion maxima de los puntos de control respecto a la linea recta P0-P3:

```
float flatness(const CubicBezier& b) {
    // Distancia maxima de los puntos de control a la linea P0-P3
    Vec2 line = b.p3 - b.p0;
    float len = line.length();
    if (len < 1e-6f) return 0.0f;

    float d1 = std::abs(line.cross(b.p1 - b.p0)) / len;
    float d2 = std::abs(line.cross(b.p2 - b.p0)) / len;
    return std::max(d1, d2);
}

void subdivide(const CubicBezier& b, float t,
               CubicBezier& left, CubicBezier& right) {
    // Algoritmo de De Casteljau:
    Vec2 q0 = lerp(b.p0, b.p1, t);
    Vec2 q1 = lerp(b.p1, b.p2, t);
    Vec2 q2 = lerp(b.p2, b.p3, t);
    Vec2 r0 = lerp(q0, q1, t);
    Vec2 r1 = lerp(q1, q2, t);
    Vec2 s  = lerp(r0, r1, t);

    left  = { b.p0, q0, r0, s };
    right = { s, r1, q2, b.p3 };
}
```

4.4.3 BezierPath — Construccion de Trayectorias

```
class BezierPath {
    std::vector<CubicBezier> segments_;
    Vec2 currentPos_;
    bool hasOpenSubpath_;

    void moveTo(float x, float y);
    void lineTo(float x, float y);
    void curveTo(float cx1, float cy1, float cx2, float cy2, float x, float y);
    void closePath();

    Vec2 evaluateAt(float t) const;              // Punto en la curva
    Vec2 tangentAt(float t) const;               // Vector tangente
    std::vector<Vec2> flattenPoints(float tol) const;  // Puntos aplanados para raster
```

```
    float totalLength() const;                  // Longitud total de la curva
};
```

4.5 Motor de Pinceles

4.5.1 BrushEngine — Estructura

```
struct BrushTip {
    std::string name;
    std::string imagePath;   // PNG de la punta del pincel
    float spacing = 0.15f;   // Espaciado entre estampas (0.15 = 15% del diametro)
    float hardness = 0.8f;   // Dureza del borde (0.0 = suave, 1.0 = duro)
    float angle = 0.0f;      // Angulo de rotacion (grados)
    float roundness = 1.0f;  // Redondez (1.0 = circulo, <1.0 = elipse)
};

struct BrushDynamics {
    bool pressure_opacity = true;  // Presion afecta opacidad
    bool pressure_size = true;     // Presion afecta tamano
    bool pressure_flow = true;     // Presion afecta flujo
    bool tilt_angle = false;       // Inclinacion afecta angulo
    float min_size = 0.1f;        // Tamano minimo (fraccion del base)
    float max_size = 2.0f;        // Tamano maximo (fraccion del base)
    float smoothing = 0.5f;       // Suavizado de trazo [0.0, 1.0]
};

struct BrushPreset {
    std::string name;
    BrushTip tip;
    BrushDynamics dynamics;
    Color color;
    float size = 10.0f;      // Tamano base en pixeles
    float opacity = 1.0f;    // Opacidad base [0.0, 1.0]
    float flow = 0.3f;       // Flujo de tinta [0.0, 1.0]
    BlendMode blend_mode;
    bool use_paper_texture = false;
};
```

4.5.2 Renderizado de Estampa de Pincel

El Canvas genera estampas de pincel como QImage circulares con feathering (desvanecimiento en los bordes). El algoritmo de renderizado de estampa:

```
void renderStampToImage(QImage& img, int radius, int cx, int cy,
                        const QString& shape, const QColor& color,
                        float opacity, bool erasing) {
    int feather = 1;  // Padding para cubrir bordes de feathering
    int R = radius + feather;
    // Para cada pixel (dx, dy) en rectangulo [-R, R] x [-R, R]:
    for (dy = -R; dy <= R; dy++) {
        for (dx = -R; dx <= R; dx++) {
            float d = sqrt(dx*dx + dy*dy);  // Distancia al centro

            if (d > R) continue;  // Fuera del circulo

            // Calcular alpha basado en distancia y hardness
            float innerR = radius * brushHardness;
            float alpha;
```

```
            if (d <= innerR) {
                alpha = 1.0f;  // Nucleo solido
            } else {
                // Feathering: smoothstep entre innerR y radius
                alpha = 1.0f - smoothstepf(innerR, radius, d);
            }
            alpha *= opacity * tabletPressure_;

            // Escribir pixel con Source Over blend en strokeBuffer_
        }
    }
}
```

4.5.3 Presets Predefinidos

Preset	Tamano	Dureza	Angulo	Redondez	Uso
Round Soft	10px	0.2	0	1.0	Pincel suave para sombreado
Round Hard	5px	0.9	0	1.0	Linea nitida para line art
Flat	8px	0.7	0	0.3	Pincel plano para caligrafia
Calligraphy	12px	0.6	45	0.25	Plumilla caligrafica
Eraser	20px	0.8	0	1.0	Borrador (usa DestinationOut en commit)

4.5.4 ABR Importer (Photoshop Brushes)

El importador ABR (v2.6.1) lee archivos de pinceles de Adobe Photoshop en formatos version 1, 2 y 6. El formato ABR contiene datos de imagen comprimidos con RLE (Run-Length Encoding) que deben ser descomprimidos.

```
struct AbrHeader {       // Packed (no padding)
    uint16_t version;     // 1, 2, o 6
    uint16_t subversion;
    int32_t commentOffset;
    int32_t numBrushes;   // Hasta kMaxBrushes (512)
};

struct AbrBrushHeader {  // Packed
    int32_t type;
    int32_t extraSize;
    int32_t top, left, bottom, right;  // Bounding box
    int32_t depth;        // Bits por canal (1 u 8)
    int8_t compression;   // 0 = raw, 1 = RLE
    int32_t dataSize;     // Tamaño de datos de imagen
};
```

Bug Critico — Endian Swap (v2.6.1):

El campo version del header ABR es uint16_t. En maquinas little-endian (x86), el compilador promovia el uint16_t a int antes del bit-shift para reconstruir el valor, produciendo un numero incorrecto (65537 en vez de 1). Esto causaba que TODOS los archivos ABR v1/v2 fallaran silenciosamente. La solucion fue hacer cast explicito a uint16_t despues de los shifts:

```
// ANTES (roto en x86):
uint16_t version = data[0] | (data[1] << 8);
```

```
// data[1] se promovia a int (sign-extend), luego shift << 8,
// luego OR con int, resultado int = 65537 (incorrecto)

// DESPUES (correcto):
uint16_t version = static_cast<uint16_t>(data[0] | (data[1] << 8));
```

4.6 Compositor de Capas

El compositor implementa el algoritmo del pintor (painter's algorithm): las capas se dibujan de abajo hacia arriba (bottom-to-top), donde cada capa se superpone sobre el resultado acumulado de las capas inferiores.

4.6.1 Algoritmo del Pintor (Painter's Algorithm)

```
void compositeLayer(const Layer* layer, RasterEngine& target) {
    if (!layer || !layer->visible()) return;

    if (layer->type() == LayerType::Raster) {
        compositeRasterLayer(*static_cast<const RasterLayer*>(layer), target);
    } else if (layer->type() == LayerType::Group) {
        const auto* group = static_cast<const GroupLayer*>(layer);
        // ITERACION HACIA ADELANTE (begin -> end):
        // La primera capa en el grupo es la del fondo (bottom),
        // la ultima es la de encima (top).
        for (auto it = group->layers().begin();
             it != group->layers().end(); ++it) {
            compositeLayer(it->get(), target);
        }
    }
}
```

Bug Critico — Z-Order Invertido (v2.6.1):

En la version anterior, la iteracion usaba rbegin()/rend() (reverse), lo que causaba que las capas se dibujaran en orden inverso: la capa del fondo aparecia encima de todas las demas. El fix fue cambiar a iteracion hacia adelante (begin/end) para obtener el orden correcto del algoritmo del pintor.

4.6.2 Composicion de RasterLayer

```
void compositeRasterLayer(const RasterLayer& raster, RasterEngine& target) {
    // Calcular region de interseccion entre la capa y el canvas destino
    int startX = max(0, originX);
    int startY = max(0, originY);
    int endX   = min(canvasW, originX + rasterW);
    int endY   = min(canvasH, originY + rasterH);

    float layerOpacity = raster.opacity();
    BlendMode mode = raster.blendMode();

    for (int ty = startY; ty < endY; ++ty) {
        for (int tx = startX; tx < endX; ++tx) {
            uint32_t sp = srcPixel(tx, ty);  // Pixel fuente de la capa
            uint8_t sa = (sp >> 24) & 0xFF;  // Alpha del pixel fuente
            if (sa == 0) continue;           // Transparente, saltar

            // Extraer canales y aplicar opacidad de capa
```

```
            float sr = ((sp >> 16) & 0xFF) / 255.0f;
            float sg = ((sp >> 8)  & 0xFF) / 255.0f;
            float sb = (sp         & 0xFF) / 255.0f;
            float sAlpha = (sa / 255.0f) * layerOpacity;

            // Blend con el pixel destino usando el modo de mezcla
            blendPixel(dst, canvasW, canvasH, tx, ty,
                       sr, sg, sb, sAlpha, mode);
        }
    }
}
```

4.6.3 NodeGraph — Composicion Basada en Nodos

El NodeGraph (v2.5.2) proporciona un sistema de composicion visual basado en nodos interconectados. Soporta 6 tipos de nodos:

Nodo	Entradas requeridas	Funcion
InputNode	0 (fuente)	Provee una RasterLayer como fuente de datos
OutputNode	1	Nodo final del grafo. Su salida se escribe al target
BlendNode	2 (bg, fg)	Mezcla dos capas con un BlendMode configurable
TransformNode	1	Aplica traslacion, escala, rotacion a su entrada
FilterNode	1	Aplica filtros: Blur, Sharpen, ColorCorrect, Threshold, Invert
GroupNode	1-8	Agrupa multiples entradas en una salida compuesta

La evaluacion del grafo se hace mediante un recorrido recursivo con cacheo:

```
void NodeGraph::evaluate(RasterLayer& target) {
    OutputNode* output = findOutputNode();
    if (!output) return;

    std::unordered_map<NodeId, std::unique_ptr<RasterLayer>> cache;
    evaluateNode(output, cache);

    // Copiar resultado del OutputNode al target
    auto it = cache.find(output->id());
    if (it != cache.end()) {
        RasterLayer& result = *it->second;
        // Copia de pixeles fila por fila
        int copyW = min(result.width(), target.width());
        int copyH = min(result.height(), target.height());
        for (int y = 0; y < copyH; ++y) {
            memcpy(target.pixelData() + y * target.width(),
                   result.pixelData() + y * result.width(),
                   copyW * sizeof(uint32_t));
        }
        target.bufferEpochTick();
        target.setHasContent(true);
    }
}
```

Bug Critico — evaluate() Ignoraba el Target (v2.5.2):

La implementacion original de evaluate(RasterLayer& target) calculaba el grafo pero nunca copiaba el resultado al parametro target. El output del nodo se perdia. El fix añadio la copia explicita de pixeles al target con las dimensiones minimas de ambos buffers.

4.7 Deformacion (Puppet Warp)

El sistema de deformacion permite distorsionar una imagen mediante una malla de puntos de control. El usuario mueve los puntos y la imagen se deforma para ajustarse.

4.7.1 DeformationMesh

```
class DeformationMesh {
    int cols_, rows_;                       // Dimensiones de la malla
    Rect bounds_;                           // Rectangulo que cubre la malla
    std::vector<DeformationPoint> points_;  // Puntos de control

    Vec2 mapPoint(const Vec2& p) const;     // Mapea un punto a su posicion deformada
    Vec2 mapPointBilinear(const Vec2& p) const;  // Interpolacion bilineal
};

struct DeformationPoint {
    Vec2 position;   // Posicion original
    Vec2 offset;     // Desplazamiento aplicado por el usuario
    bool locked;     // Si esta bloqueado, no se mueve
};
```

4.7.2 Interpolacion Bilineal de Deformacion

Para un punto p en el espacio, se calculan sus coordenadas normalizadas (u,v) dentro de la malla, se identifican las 4 celdas circundantes, y se interpolan bilinealmente las posiciones deformadas:

```
u = (p.x - bounds.x) / bounds.width,    v = (p.y - bounds.y) / bounds.height

col_f = u * (cols - 1),    row_f = v * (rows - 1)

p_deformado = p00*(1-fx)*(1-fy) + p10*fx*(1-fy) + p01*(1-fx)*fy + p11*fx*fy
```

```
Vec2 DeformationMesh::mapPointBilinear(const Vec2& p) const {
    float u = (p.x - bounds_.x) / bounds_.width;
    float v = (p.y - bounds_.y) / bounds_.height;
    u = std::clamp(u, 0.0f, 1.0f);
    v = std::clamp(v, 0.0f, 1.0f);

    float colF = u * (cols_ - 1);
    float rowF = v * (rows_ - 1);
    int col0 = clamp(int(colF), 0, cols_-1);
    int row0 = clamp(int(rowF), 0, rows_-1);
    int col1 = min(col0+1, cols_-1);
    int row1 = min(row0+1, rows_-1);
    float fx = colF - col0;  // Fraccion dentro de la celda [0,1]
    float fy = rowF - row0;

    auto p00 = deformedPosition(col0, row0);
    auto p10 = deformedPosition(col1, row0);
    auto p01 = deformedPosition(col0, row1);
    auto p11 = deformedPosition(col1, row1);

    // Interpolacion bilineal
```

```
    float x = p00.x*(1-fx)*(1-fy) + p10.x*fx*(1-fy)
            + p01.x*(1-fx)*fy    + p11.x*fx*fy;
    float y = p00.y*(1-fx)*(1-fy) + p10.y*fx*(1-fy)
            + p01.y*(1-fx)*fy    + p11.y*fx*fy;

    return { x, y };
}
```

Bug Critico — Bilinear Interpolation Error (v2.6.1):

La formula original para la coordenada Y tenia un error tipografico: p11.y * fy * fy (fy al cuadrado) en lugar de p11.y * fx * fy. Esto causaba que el cuadrante inferior derecho de las mallas deformadas tuviera coordenadas Y incorrectas. El fix corrigio el factor de peso a fx * fy.

4.8 Motor de Animacion

El motor de animacion incluye el sistema de reproduccion (playback), onion skinning (piel de cebolla), tweens (interpolacion), y thumbnails de frames.

4.8.1 Playback — avance de Frame con Work Area

```
// Timer de playback en TimelinePanelV2:
int intervalMs = static_cast<int>(std::round(1000.0 / fps_));
// Para 24fps: round(1000.0/24) = round(41.6667) = 42ms
// Previamente (bug de precision): int(1000/24) = 41ms -> drift acumulativo

void onPlaybackTick() {
    int newFrame = seq.advanceFrame();  // Work-area-aware advance
    appState_->setCurrentFrame(newFrame);

    if (seq.looping() && newFrame == seq.workAreaStart()) {
        // Re-sync audio al inicio del work area
        syncToFrame(seq.workAreaStart());
    }
    if (!seq.looping() && newFrame == waEnd - 1) {
        // Auto-stop al final del work area
        stop_playback();
    }
}
```

4.8.2 Onion Skinning

El onion skinning renderiza frames adyacentes (previos y siguientes) con opacidad reducida, permitiendo al animador ver el movimiento entre frames:

$$opacity\_onion = onionOpacity / 100.0 \times (1 - |delta\_frame| / (max\_frames + 1))$$

4.9 Sistema de Audio

4.9.1 Arquitectura de Decodificacion (dr_libs)

El sistema de audio utiliza las librerias single-header de David Reid (dr_libs) para decodificacion nativa sincrona de WAV, MP3 y FLAC. Esto reemplazo a QAudioDecoder que era asincrono e inestable en Qt 6.5.

```
// Wrapper en audio_file_decoder.cpp:
```

```
AudioDecodeResult decodeAudioFile(const std::string& filepath,
                                  int targetPeaks = 800) {
    // 1. Detectar formato por extension
    // 2. Usar drwav/drmp3/drflac para leer PCM frames f32
    // 3. Reducir a targetPeaks muestras (downsampling de waveform)
    // 4. Retornar AudioDecodeResult { peaks, sampleRate, channels, durationMs }
}

struct AudioDecodeResult {
    bool success = false;
    std::vector<float> peaks;      // Muestras de amplitud para waveform
    int sampleRate = 0;
    int channels = 0;
    int64_t durationMs = 0;        // Duracion en milisegundos
};
```

4.9.2 Audio Transport Sync

Durante la reproduccion, el audio se sincroniza con el timeline usando setPlaybackRate para ajustar la velocidad:

```
playback_rate = fps / 24.0
```

Esto ajusta la velocidad de reproduccion del audio proporcionalmente al FPS de la secuencia. A 24fps, rate=1.0 (velocidad normal). A 12fps, rate=0.5 (mitad de velocidad). A 48fps, rate=2.0 (doble velocidad).

4.10 Sistema de Video

4.10.1 Video Decoder (FFmpeg)

```
struct VideoMetadata {
    int width = 0, height = 0;
    double fps = 24.0;
    int totalFrames = 0;
    double durationSecs = 0.0;
    bool valid = false;
};

// Obtenido via ffprobe -print_format json
VideoMetadata probeVideoMetadata(const QString& filepath);

// Decodifica un frame especifico via ffmpeg pipe PNG
QImage decodeVideoFrame(const QString& filepath, int frameIndex,
                        double fps, int maxW = 1920, int maxH = 1080);
// Comando: ffmpeg -ss <time> -vframes 1 -vcodec png -f image2pipe -

class VideoFrameCache {
    static constexpr int kMaxCacheFrames = 50;  // LRU cache
    QImage get(int frameIndex) const;
    void put(int frameIndex, QImage img);
    void clear();
};
```

Bug Critico — Recursion Guard (v2.9.1):

decodeVideoFrame() usaba QCoreApplication::processEvents() en su loop de espera para mantener la UI responsiva. Sin embargo, processEvents() podia procesar eventos de layout

pendientes que disparaban un nuevo paintEvent, el cual a su vez llamaba buildBackgroundCache() -> VideoTrackWidget::currentFrame() -> decodeVideoFrame(), entrando en recursion infinita.

```
thread_local bool s_inDecoder = false;

QImage decodeVideoFrame(...) {
    if (s_inDecoder) return {};  // GUARD: rechazar llamada recursiva
    s_inDecoder = true;

    while (proc.waitForReadyRead(100)) {
        QCoreApplication::processEvents();  // Mantener UI responsiva
    }

    s_inDecoder = false;
    return result;
}
```

5. INTERFAZ DE USUARIO (UI V2)

5.1 Canvas y Pipeline de Display

5.1.1 Arquitectura de 4 Buffers

El CanvasWidgetV2 implementa una arquitectura de 4 buffers que separa estrictamente el dominio de DATOS del dominio de DISPLAY. Esta arquitectura fue el resultado de 2 dias de debugging intensivo y multiples iteraciones:

5.1.2 Invariantes del Pipeline

Invariante	Descripcion
mouseReleaseEvent NUNCA lee display buffers	Solo lee strokeBuffer_ y pixelBuffer_. Los display buffers (backgroundCache_) son solo para pintar.
Dabs escritos a strokeBuffer_ aislado	Durante el stroke activo, todos los dabs van a strokeBuffer_ (transparente). No tocan pixelBuffer_.
Commit composita strokeBuffer_ -> pixelBuffer_	Al soltar el mouse, commitStroke() aplica el strokeBuffer_ completo sobre pixelBuffer_ con el modo de blend adecuado.

buildBackgroundCache soporta full y partial	Reconstruccion completa (frame change) o parcial (dirty rect post-commit).
Dynamic padding: brushSize/2 + 1	Margen de seguridad alrededor de cada dab para cubrir el feathering del pincel.
Capas con originX_/originY_ usan rect.translated()	El offset de la capa se aplica correctamente tanto en path FULL como PARTIAL.

5.1.3 buildBackgroundCache() — Reconstruccion del Display

```
void CanvasWidgetV2::buildBackgroundCache(const QRect& dirtyRect) {
    bool fullRebuild = dirtyRect.isNull();

    // Escalar partial -> full si hay video tracks activos
    if (!fullRebuild && videoTracks_ && !videoTracks_->empty()) {
        fullRebuild = true;  // Los frames de video necesitan contexto completo
    }

    QPainter painter(&backgroundCache_);

    if (fullRebuild) {
        // 1. Fondo blanco
        painter.fillRect(backgroundCache_.rect(), Qt::white);

        // 2. Onion skin (frames previos/siguientes con opacidad reducida)
        drawOnionSkin(painter);

        // 3. Capas visibles en orden (bottom-to-top)
        const GroupLayer& root = seq.rootLayerForFrame(currentFrame_);
        for (int i = 0; i < root.layerCount(); ++i) {
            const RasterLayer* rl = static_cast<const RasterLayer*>(
                root.layerAt(i));
            // Compositear capa sobre el cache usando QPainter
        }

        // 4. Video frames (encima de las capas de dibujo)
        drawVideoFrames(painter);

        // 5. Aplicar Line Boil (no destructivo, sobre copia)
        if (seq.lineBoilEnabled()) {
            applyLineBoil(backgroundCache_, currentFrame_,
                          seq.lineBoilStrength(), seq.lineBoilSeed());
        }
    } else {
        // Rebuild parcial: solo el dirtyRect
        QRect canvasRect = dirtyRect.translated(
            -originX_of_layer, -originY_of_layer);
        painter.fillRect(dirtyRect, Qt::white);
        // Re-compositar capas en el dirtyRect
        // ...
    }

    backgroundCacheValid_ = true;
}
```

5.1.4 Herramientas del Canvas

Herramienta	Tecla	Modo de operacion	Undo support

Brush	B	Dibujo a mano alzada con pincel	Snapshot full layer antes del stroke
Eraser	E	Borra con modo DestinationOut en commit	Snapshot full layer
Fill	G	Flood fill 4-way (solido, tela, degradado)	Snapshot full layer
Text	T	Entrada de texto con edicion in-place	Snapshot undo por entrada
Line	L	Linea recta entre click inicial y final	Snapshot full layer
Rectangle	R	Rectangulo desde esquina inicial	Snapshot full layer
Ellipse	E	Elipse inscrita en bounding box	Snapshot full layer
Move	V	Mueve el contenido de la capa	Snapshot + QImage de origen
Select	M	Seleccion rectangular con marching ants	Snapshot + floating selection
Hand	H	Paneo del viewport (middle-click tambien)	No aplica
ColorPicker	I	Muestrea color del canvas	No aplica
PencilRetouch	-	Suavizado de trazos a lapiz	Modifica capa existente
RulerLine	-	Guia de linea recta	No aplica
RulerEllipse	-	Guia de elipse	No aplica
DeformMesh	-	Deformacion de malla (puppet warp)	Malla completa + capa
TweenEdit	-	Editor de interpolacion	Keyframes

5.1.5 Flood Fill

El algoritmo de flood fill utiliza un enfoque recursivo de 4 direcciones (arriba, abajo, izquierda, derecha). Soporta tres modos: solido (por tolerancia de color), por alpha (limite de opacidad), y tela/degradado (efectos especiales).

```
void floodFill(QImage& img, int x, int y, const QColor& fillColor,
               const QColor& targetColor, int tolerance) {
    if (x < 0 || x >= img.width() || y < 0 || y >= img.height())
        return;
    QColor current = img.pixelColor(x, y);
    // Diferencia de color entre pixel actual y objetivo
    int dr = abs(current.red()   - targetColor.red());
    int dg = abs(current.green() - targetColor.green());
    int db = abs(current.blue()  - targetColor.blue());
    int da = abs(current.alpha() - targetColor.alpha());
    if (dr > tolerance || dg > tolerance || db > tolerance || da > tolerance)
        return;
    if (current == fillColor) return;  // Ya rellenado

    img.setPixelColor(x, y, fillColor);

    floodFill(img, x+1, y, fillColor, targetColor, tolerance);
    floodFill(img, x-1, y, fillColor, targetColor, tolerance);
    floodFill(img, x, y+1, fillColor, targetColor, tolerance);
```

```
    floodFill(img, x, y-1, fillColor, targetColor, tolerance);
}
```

5.1.6 Tablet Pressure Support

El canvas soporta tabletas graficas (Wacom, Huion, XP-Pen, Xencelabs) mediante QTabletEvent nativo de Qt 6. La presion del lapiz modula el tamano y la opacidad del pincel en tiempo real:

$$effective\_radius = brushSize * (0.2 + 0.8 * pressure)$$

$$effective\_opacity = brushOpacity * pressure$$

El canvas habilita WA_TabletTracking en el constructor y sobrecarga tabletEvent() para manejar eventos TabletPress, TabletMove y TabletRelease. Ademas, detecta el borrador del lapiz (PointerType::Eraser) para cambiar automaticamente a modo borrador.

Bug Critico — Tablet Event Dispatching (v2.9):

Al conectar una Wacom, el canvas dejaba de responder completamente. Qt no siempre sintetiza QMouseEvent desde QTabletEvent (depende del driver). La solucion fue que tabletEvent() genere QMouseEvent sinteticos y los despache via QApplication::sendEvent() a los handlers de mouse estandar:

```
void CanvasWidgetV2::tabletEvent(QTabletEvent* event) {
    // Actualizar estado de presion y tipo de puntero
    tabletPressure_ = event->pressure();
    tabletEraser_ = (event->pointerType() == QPointingDevice::PointerType::Eraser);

    switch (event->type()) {
    case QEvent::TabletPress:
        tabletPenDown_ = true;
        // Generar QMouseEvent sintetico y despachar via sendEvent()
        QApplication::sendEvent(this, &syntheticPressEvent);
        break;
    case QEvent::TabletMove:
        if (tabletPenDown_) {
            QApplication::sendEvent(this, &syntheticMoveEvent);
        }
        break;
    case QEvent::TabletRelease:
        tabletPenDown_ = false;
        QApplication::sendEvent(this, &syntheticReleaseEvent);
        break;
    }
}
```

5.1.7 Control de Undo/Redo en Canvas

El sistema de undo en el canvas captura un snapshot QImage de la capa activa ANTES de cualquier operacion destructiva. La funcion pushFullLayerUndo() es el punto central:

```
void CanvasWidgetV2::pushFullLayerUndo(RasterLayer* layer,
                                       const QImage& beforeSnap,
                                       bool hadContentBefore) {
    // 1. Convertir el snapshot a ARGB32 premultiplicado
    QImage snap = beforeSnap.convertToFormat(
        QImage::Format_ARGB32_Premultiplied);
```

```
    // 2. Crear comando de undo con estado completo
    auto cmd = std::make_unique<LayerModifyCommand>(
        layer, snap, hadContentBefore);

    // 3. Push al UndoManager de la secuencia activa
    appState_->activeSequence().undoManager()
        .pushApplied(std::move(cmd));

    // 4. Marcar documento como modificado
    appState_->document().setModified(true);
}
```

Bug Critico — hasContentBefore en 6 Call Sites (v2.6.1):

En 6 sitios de llamada (doFill, doText, clearTextRaster, commitMove, commitSelection, commitFloatingSelection), la variable hadBefore = layer->hasContent() se calculaba pero NUNCA se pasaba a pushFullLayerUndo(). El comando de undo usaba hadContentBefore_ = false por defecto, lo que causaba que al deshacer la operacion, las celdas de la timeline aparecieran vacias aunque los pixeles se hubieran restaurado correctamente. El fix paso hadBefore como 3er argumento en los 6 call sites.

Bug Critico — Undo/Redo sin canvasUpdated (v2.7):

Las funciones undo() y redo() del canvas llamaban a invalidateBackgroundCache() + update(), refrescando el canvas, pero nunca emitian canvasUpdated. Esta senal esta conectada en MainWindowV2 a timeline_panel_->update() y layer_panel_->refreshLayerList(), por lo que la timeline y el panel de capas permanecian desactualizados visualmente tras Ctrl+Z/Ctrl+Y. El fix añadio emit canvasUpdated() en ambos metodos.

5.2 Linea de Tiempo (TimelinePanelV2)

5.2.1 Arquitectura de Reconstruccion No Destructiva

La TimelinePanelV2 (v2.3-v2.9) utiliza un sistema de reconstruccion no destructiva donde las pistas de audio y video nunca se eliminan durante rebuildTracks(), sino que se extraen del layout y se reinsertan:

```
void TimelinePanelV2::rebuildTracks() {
    // 1. SALVAR widgets de audio/video (NUNCA hacer delete)
    std::vector<AudioTrackWidget*> savedAudio;
    for (auto* w : audioTrackWidgets_) {
        tracksLayout_->removeWidget(w);  // Extraer sin destruir
        savedAudio.push_back(w);
    }
    // Lo mismo para videoTrackWidgets_ ...

    // 2. DESTRUIR solo SequenceTrackWidgets (se recrean desde Document)
    while (QLayoutItem* item = tracksLayout_->takeAt(0)) {
        if (auto* seq = dynamic_cast<SequenceTrackWidget*>(item->widget())) {
            delete seq;  // Estos SI se pueden destruir
        }
        delete item;
    }

    // 3. RECONSTRUIR SequenceTrackWidgets desde AppState
    for (size_t si = 0; si < doc.sequenceCount(); ++si) {
        auto* seqWidget = new SequenceTrackWidget(doc.sequenceAt(si), ...);
        tracksLayout_->addWidget(seqWidget);
    }

    // 4. REINSERTAR widgets de audio/video preservados
    for (auto* w : savedAudio) {
        tracksLayout_->addWidget(w);
    }
    for (auto* w : savedVideo) {
        tracksLayout_->addWidget(w);
    }

    // 5. Forzar layout update sincrono (previene waveform con width=0)
    tracksLayout_->addStretch();
    tracksLayout_->update();
}
```

5.2.2 Work Area Interactiva

El RulerWidget permite interaccion directa con la work area:

- Hover a ±5px de los bordes In/Out: cursor cambia a SizeHorCursor
- Click y drag en borde In: redimensiona, clamp [0, waEnd-1]
- Click y drag en borde Out: redimensiona, clamp [waStart+1, totalFrames]
- Click en otra zona: posiciona el playhead + scrubbing
- Click durante playback: auto-pausa antes de mover playhead
- Barra de WA: fill naranja semi-transparente rgba(255,72,0,80), bordes de 1px

5.2.3 Marcadores en la Regla

Los marcadores se renderizan como triangulos coloreados (8-12px) apuntando hacia abajo en la posicion del frame, con bandas de duracion semi-transparentes y etiquetas de texto en pills oscuros:

```
// Colores de marcadores (9 opciones, estilo After Effects):
static const QColor kMarkerColors[] = {
    QColor(200,200,200),  // None (gray)
    QColor(255,80,80),    // Red
    QColor(255,180,60),   // Orange
    QColor(255,230,70),   // Yellow
    QColor(100,210,100),  // Green
    QColor(70,200,210),   // Cyan
    QColor(80,140,255),   // Blue
    QColor(160,100,230),  // Purple
    QColor(240,100,200),  // Magenta
};
// Titulo: 7px bold white (#FFFFFF) sobre fondo rgba(18,22,32,210)
```

5.2.4 FPS Unidireccional sin Feedback Loop

```
// Flag de guarda para prevenir bucle de senales:
bool updatingFps_ = false;

void onFPSChanged(int fps) {
    if (updatingFps_) return;  // Ignorar si viene de actualizacion programatica
    appState_->setFps(fps);
}

// Listener de documentChanged sincroniza spinbox:
connect(appState_.get(), &AppState::documentChanged, this, [this]() {
    int seqFps = appState_->activeSequence().fps();
    if (fps_ != seqFps) {
        updatingFps_ = true;        // Bloquear feedback
        fpsSpin_->setValue(seqFps);
        updatingFps_ = false;
    }
});
```

5.3 Paneles

5.3.1 ToolboxPanelV2

El panel de herramientas contiene 11 botones de herramientas en un QVBoxLayout compacto de 52px de ancho (36px icono + margenes), sin QScrollArea innecesario y sin stretch para ajuste perfecto al contenido.

5.3.2 LayerPanelV2

El panel de capas implementa CRUD completo con protecciones contra crashes:

Evolucion del Sistema de Renombre:

Fase	Problema	Solucion
V1	editingFinished causaba double-delete	Se desconecto editingFinished

	del QLineEdit	
V2	Sin editingFinished, perder foco = nombre descartado	Reconectar editingFinished con guard shared_ptr<bool> committed
V3	Crash al cambiar frame durante rename activo (use-after-free)	Raw pointers -> QPointer<QHBoxLayout>, QPointer<QLineEdit>, QPointer<QLabel> con verificacion antes de acceso
V4	Nombres viejos visibles como overlap tras refreshLayerList	removeItemWidget() + hide() + deleteLater() antes de QListWidget::clear()

5.3.4 OnionSkinPanel y CanvasSizePanel

Extraidos del ToolboxPanelV2 a docks independientes (v2.8), permitiendo reorganizacion libre del espacio de trabajo.

5.4 Pistas de Audio y Video

5.4.1 AudioTrackWidget

```
class AudioTrackWidget : public QWidget {
    // Botones SVG 28x28: mute (🔊), move up (▲), move down (▼), delete (✕)
    // Waveform rendering desde AudioDecodeResult::peaks
    // Sliders: volumen (0-100), balance

    void drawWaveform(QPainter& p, const QRect& rect) {
        int w = rect.width() - hdrW;
        if (w <= hdrW) return;  // GUARD: previene division por cero

        const auto& peaks = decodeResult_.peaks;
        float samplesPerPixel = peaks.size() / float(w);
        // Para cada pixel horizontal, calcular min/max amplitud
        // y dibujar barra vertical centrada
    }
};
```

5.4.2 VideoTrackWidget

```
class VideoTrackWidget : public QWidget {
    // Misma estructura de botones que AudioTrackWidget (28x28 SVG)
    // Thumbnail strip: 1 thumbnail cada 30 frames
    // Sliders: opacidad (0-100%), volumen (0-100%)

    // Senal opacityChanged() -> videoTrackChanged() ->
    //   -> canvas_->invalidateBackgroundCache() (tiempo real)

    QImage currentFrame() {
        // 1. Buscar en VideoFrameCache (LRU 50 frames)
        // 2. Si no esta, decodeVideoFrame() via ffmpeg
        // 3. Guardar en cache y retornar
    }
};
```

5.4.3 Movimiento Libre y Persistencia de Posicion

Las pistas de audio y video pueden moverse libremente en el layout de la timeline (sin restriccion de zona). La posicion se persiste en el documento mediante el campo layoutPosition:

```
void TimelinePanelV2::onMoveAudioTrack(AudioTrackWidget* w, bool up) {
    int idx = tracksLayout_->indexOf(w);
    int targetIdx = up ? idx - 1 : idx + 1;
    // takeAt + insertItem para movimiento libre en el layout
    QLayoutItem* item = tracksLayout_->takeAt(idx);
    tracksLayout_->insertItem(targetIdx, item);
}
```

5.5 Ventana Principal y Docks

MainWindowV2 utiliza 8 QDockWidgets con tema oscuro y barras de titulo naranja. Todos los docks son Movable | Floatable, permitiendo al usuario reorganizar el espacio de trabajo:

5.6 BusyOverlay

Widget overlay semi-transparente que cubre toda la ventana durante operaciones bloqueantes (save, load, export). Muestra un spinner animado de 12 lineas naranja:

```
class BusyOverlay : public QWidget {
    bool filterInstalled_ = false;  // Guard contra instalacion duplicada
    int spinnerAngle_ = 0;          // Angulo de rotacion del spinner
    QTimer* timer_;                 // Animacion + keep-alive (150ms)

    // eventFilter bloquea mouse y teclado durante operacion
    // show() + raise() cada 150ms para mantenerse encima
};
```

6. SISTEMA DE ARCHIVOS (IO)

6.1 Formato .fap v2 (ZIP Binario)

El formato .fap v2 es un archivo ZIP renombrado que contiene toda la informacion del proyecto en un solo archivo portable:

```
.fap file (renamed .zip)
├── manifest.json       — version, canvas size, active_sequence, viewport (zoom/offset)
├── timeline.json       — per-sequence: frames[], layers[] with pixel_file/meta_file paths
│                         + "audio" array + "video" array + markers
├── audio/track_N.ext   — Archivos de audio embebidos (mp3, wav, flac)
├── video/track_N.ext   — Archivos de video embebidos
└── layers/S{seq}/frame_{f}/layer_{l}.png + .json
```

6.1.1 DocumentManager — Guardado Atomico

```
bool DocumentManager::save(const Document& doc, const QString& path,
                           const ViewState& vs) {
    // 1. Crear archivo temporal .tmp
    QString tmpPath;
    prepareAtomicTarget(path, tmpPath);

    // 2. Escribir ZIP al archivo temporal
    mz_zip_archive zip;
    mz_zip_writer_init_file(&zip, tmpPath.toUtf8().constData(), 0);

    writeManifest(&zip, doc);     // manifest.json
    writeTimeline(&zip, doc);     // timeline.json + audio metadata
    writeLayerData(&zip, doc);     // layers/S{seq}/... + audio/video bytes

    mz_zip_writer_finalize_archive(&zip);
    mz_zip_writer_end(&zip);

    // 3. Commit atomico: renombrar .tmp -> .fap
    return commitAtomic(tmpPath, path);
}

bool DocumentManager::commitAtomic(const QString& tmpPath,
                                   const QString& finalPath) {
    // Intento 1: rename directo (funciona en Linux/macOS)
    if (QFile::rename(tmpPath, finalPath)) return true;

    // Intento 2 (Windows): el target existe -> eliminar y reintentar
    if (QFile::exists(finalPath)) {
        QFile::remove(finalPath);
        if (QFile::rename(tmpPath, finalPath)) return true;
    }
    lastError_ = "Atomic rename failed";
    return false;
}
```

Evolucion del Guardado Atomico:

Fase	Implementacion	Problema	Solucion
V1	remove(target) +	Si rename fallaba, datos	—

	rename(tmp, target)	perdidos (TOCTOU)	
V2 (H3 fix)	rename(tmp, target) directo	Qt 6.5 en Windows: rename() NO sobrescribe	Eliminar remove previo
V3 (H3 regression fix)	rename() -> si falla y target existe -> remove + retry	Ventana TOCTOU en reintento	Minimizado a intervalo de reintento

6.1.4 Persistencia de Audio y Video

```
// Escritura en timeline.json:
{
  "audio": [
    {
      "filepath": "C:/music/song.mp3",
      "display_name": "song.mp3",
      "zip_entry": "audio/track_0.mp3",
      "muted": false,
      "volume": 80,
      "layout_position": 2
    }
  ],
  "video": [
    {
      "filepath": "C:/clips/intro.mp4",
      "display_name": "intro.mp4",
      "zip_entry": "video/track_0.mp4",
      "width": 1920, "height": 1080,
      "fps": 30, "total_frames": 1500,
      "muted": false, "volume": 80,
      "opacity": 0.85,
      "layout_position": 1
    }
  ]
}
```

Al cargar, los archivos se extraen a %TEMP%/fap_audio_<PID>/ y %TEMP%/fap_video_<PID>/, actualizando los filepath a las rutas temporales. La sanitizacion de nombres previene path traversal:

```
// Sanitizacion: solo nombre de archivo, sin directorios
QString safeName = QFileInfo(rawName).fileName();
QString tempPath = tempDir + "/" + safeName;
```

6.5 Exportacion de Video

6.5.1 Pipeline Unificado

La exportacion de video (v2.5.2 unificada) soporta 3 formatos con auto-deteccion por extension:

Formato	Codec Video	Codec Audio	Pixel Format	Alpha
MP4 (.mp4)	libx264	aac	yuv420p	No
MOV (.mov)	qtrle	pcm_s16le	argb	Si (QuickTime Animation)

WebM (.webm)	libvpx-vp9	libopus	yuva420p	Si (VP9 Alpha)

```
QImage renderExportFrame(const Document& doc, int frameIndex,
                         const ExportOptions& opts) {
    // 1. Crear QImage del tamano de salida (o canvas)
    // 2. Compositar todas las capas del frame (sin UI overlays)
    // 3. Aplicar escala si opts.outputWidth/Height != 0
    // 4. Devolver frame listo para encoding
}

bool exportVideo(const Document& doc, const QString& outputPath,
                 int fps, const ExportOptions& opts) {
    // Auto-detectar formato por extension (.mp4/.mov/.webm)
    // Construir comando ffmpeg con codecs + audio mixing (amix)
    // Pipe frames via stdin: QImage -> PNG -> ffmpeg
    // Timeout de 120s con kill en caso de hang
}
```

6.5.2 Audio Mixing en Export

El audio de las pistas no muteadas se mezcla usando el filtro amix de FFmpeg, respetando el volumen individual de cada pista:

```
// Filtro amix: mezcla N pistas de audio con pesos de volumen
// amix=inputs=N:duration=first:dropout_transition=0[audio_out]
// Volumen individual: volumen[i] / 100.0 aplicado como filtro previo
```

7. PLATAFORMA (WINDOWS)

La integracion con Windows incluye asociacion de archivos .fap en el registro y un icono personalizado inyectado post-build.

```
// Registro: HKEY_CURRENT_USER\Software\Classes\.fap
//   -> FAP.Document -> shell\open\command -> "exe_path" "%1"

void registerFileAssociation() {
    QSettings settings("HKEY_CURRENT_USER\Software\Classes", ...);
    settings.setValue(".fap/Default", "FAP.Document");
    settings.setValue("FAP.Document/DefaultIcon/Default", exePath + ",0");
    settings.setValue("FAP.Document/shell/open/command/Default",
                      "\"" + exePath + "\" \"%1\"");
}

// Post-build: scripts/embed_icon.py
// Usa Win32 BeginUpdateResource / UpdateResource / EndUpdateResource
// para inyectar fap.ico (7 resoluciones, 16-256px) en el .exe
```

8. FORMULAS MATEMATICAS (ANEXO COMPLETO)

8.1 Alpha Compositing (Porter-Duff Source Over)

La formula fundamental de composicion de capas, implementada en RasterLayer::blendPixel() y blend_utils.hpp::blendPixel():

$$C_{out} = C_{src} + C_{dst} \times (1 - A_{src})$$

$$A_{out} = A_{src} + A_{dst} \times (1 - A_{src})$$

Donde C esta premultiplicado: C = (R x A, G x A, B x A). Esta formula asume que tanto source como destination ya estan en espacio premultiplicado.

8.2 Tabla de Modos de Blend

```
Normal:     f(a,b) = a
Multiply:   f(a,b) = a * b
Screen:     f(a,b) = 1 - (1-a)*(1-b)
Overlay:    f(a,b) = 2*a*b                  si b < 0.5
                   = 1 - 2*(1-a)*(1-b)      si b >= 0.5
Add:        f(a,b) = min(1, a + b)
Subtract:   f(a,b) = max(0, b - a)
Darken:     f(a,b) = min(a, b)
Lighten:    f(a,b) = max(a, b)
ColorBurn:  f(a,b) = 1 - min(1, (1-b)/a)    si a > 0;  0 si a = 0
ColorDodge: f(a,b) = min(1, b/(1-a))         si a < 1;  1 si a = 1
SoftLight:  f(a,b) = b - (1-2a)*b*(1-b)                  si a <= 0.5
                   = b + (2a-1)*( ((16b-12)b+4)b-b )     si b <= 0.25
                   = b + (2a-1)*( sqrt(b) - b )          si b > 0.25
HardLight:  f(a,b) = 2*a*b                  si a < 0.5
                   = 1 - 2*(1-a)*(1-b)      si a >= 0.5

Composicion final: C_final = a_src * f(a_src, b_dst) + (1-a_src) * b_dst
```

8.3 Smoothstep

$$t = clamp((x - edge0) / (edge1 - edge0), 0, 1)$$

$$smoothstep(x) = t^2 \times (3 - 2t)$$

Casos especiales:

- Si edge0 == edge1: retorna 1.0 si x >= edge1, 0.0 en caso contrario (step function)
- smoothstep(edge0) = 0, smoothstep(edge1) = 1
- Derivada en t=0 y t=1 es 0 (curva C^1 continua, sin saltos en la pendiente)
- Se usa para el feathering de los bordes de pinceles, transicionando suavemente de opaco a transparente

8.4 Interpolacion Bilineal

Usada en deformation_mesh.cpp y raster_effect.cpp:

$$f(x,y) = p00*(1-fx)*(1-fy) + p10*fx*(1-fy) + p01*(1-fx)*fy + p11*fx*fy$$

Donde fx, fy son las coordenadas fraccionales dentro de la celda [0,1]x[0,1]:

```
fx = (x - x0) / (x1 - x0),    fy = (y - y0) / (y1 - y0)
```

8.5 Curvas de Bezier Cubicas

```
B(t) = (1-t)^3 x P0 + 3(1-t)^2 x t x P1 + 3(1-t) x t^2 x P2 + t^3 x P3

B'(t) = 3(1-t)^2 x (P1-P0) + 6(1-t)t x (P2-P1) + 3t^2 x (P3-P2)

Flatness = max( |cross(line, P1-P0)|, |cross(line, P2-P0)| ) / |line|
```

8.6 Funcion Hash Espacial (Line Boil)

```
h(x,y,f,s) = (x*374761393 + y*668265263 + f*1274126177 + s*3443311159) mod 2^32

h = (h XOR (h >> 13)) * 1274126177

h = h XOR (h >> 16)

offset = ((h_low_byte / 255.0) - 0.5) * strength * 2.0
```

8.7 Flood Fill

Algoritmo de inundacion 4-way recursivo. Condicion de parada:

```
delta_color = |R_actual - R_target| + |G_actual - G_target| + |B_actual - B_target| +
              |A_actual - A_target|

si delta_color > tolerance: no rellenar
```

8.8 Calculo de Estampa de Pincel

```
radio_efectivo = brushSize / 2 + feather_padding

inner_radius = radio_efectivo * hardness

d = sqrt(dx^2 + dy^2)  // distancia al centro de la estampa

alpha(pixel) = 1.0                                  si d <= inner_radius

alpha(pixel) = smoothstep(d, inner_radius, R)       si inner_radius < d <= R

alpha(pixel) = 0                                    si d > R

alpha_final = alpha(pixel) * opacity * pressure
```

8.9 Work Area Wrapping

```
waStart = clamp(workAreaStart, 0, durationFrames - 1)

waEnd = workAreaEnd > 0 ? min(workAreaEnd, durationFrames) : durationFrames

si currentFrame >= waEnd - 1: currentFrame = waStart

sino: currentFrame = currentFrame + 1
```

8.10 Timer de Playback

```
intervalo_ms = round(1000.0 / fps)
```

```
playback_rate = fps / 24.0  // ajuste de velocidad de audio
```

8.11 Tabla Resumen Formulas x Modulo

Modulo	Formula Principal	Archivo
Alpha Compositing	C_out = C_src + C_dst*(1-A_src)	layer.cpp:98-113
12 Blend Modes	f(a,b) — ver tabla completa	blend_utils.hpp:31-51
Smoothstep	t^2 * (3 - 2t)	types.hpp:218-224
Bilinear Interpolation	p00(1-fx)(1-fy) + p10*fx(1-fy) + ...	deformation_mesh.cpp:86-89
Bezier Cubic	B(t) = (1-t)^3 P0 + 3(1-t)^2 t P1 + ...	bezier_path.cpp
De Casteljau Subdivision	9 operaciones lerp secuenciales	bezier_path.cpp
Hash Espacial	h = (h xor (h>>13)) * 1274126177; h = h xor (h>>16)	raster_effect.cpp:8-16
Flood Fill	4-way recursion con tolerancia de color	canvas_widget_v2.cpp
Porter-Duff Alpha	A_out = A_src + A_dst*(1-A_src)	layer.cpp:106
Brush Feathering	alpha = smoothstep(d, innerR, outerR)	canvas_widget_v2.cpp
Work Area Advance	wrap a waStart si >= waEnd-1	sequence.cpp:74-88
Playback Timer	round(1000.0 / fps)	timeline_panel_v2.cpp:1134
Audio Rate	fps / 24.0	timeline_panel_v2.cpp
Pixel Packing	B | (G<<8) | (R<<16) | (A<<24)	layer.cpp:76-79
COW Threshold	use_count() > 1 → deep copy	layer.cpp:28-31

9. GESTION DE MEMORIA RAM

Free Animation Power implementa multiples estrategias de gestion de memoria para mantener un rendimiento interactivo incluso con proyectos complejos de alta resolucion.

9.1 Copy-on-Write (COW)

El mecanismo COW basado en shared_ptr<PixelBuffer> es la estrategia principal de ahorro de memoria:

- Multiples RasterLayers pueden compartir el mismo PixelBuffer sin costo de memoria
- ensureUnique() solo realiza deep copy cuando use_count() > 1
- clone() de RasterLayer comparte el buffer (costo O(1) en memoria)
- shareDataFrom() permite compartir explicitamente entre capas/frames
- copyLayerToAllFrames() usa shareDataFrom + ensureUnique para copias independientes

9.2 Almacenamiento Sparse de Frames

La estructura std::map<int, unique_ptr<GroupLayer>> para frames implementa almacenamiento sparse:

- Solo los frames accedidos (dibujados o modificados) existen en memoria
- rootLayerForFrame() crea frames bajo demanda (lazy initialization)
- peekRootLayerForFrame() permite verificar existencia sin crear
- removeFrameData() elimina permanentemente datos de un frame

9.3 Limites de Seguridad

Limite	Valor	Ubicacion	Proposito
kMaxDim	10,000	RasterLayer	Dimension maxima de buffer (100M pixeles)
kMaxPixels	100,000,000	RasterLayer::ensureContains	Evita overflow en newPixels = W*H
int64_t intermedio	—	ensureContains()	Previene overflow en multiplicacion de int32
kMaxCacheFrames	50	VideoFrameCache	Limite de frames decodificados (LRU eviction)
kMaxEntries	128	UndoManager	Stack circular de comandos de undo
kMaxBrushes	512	AbrImporter	Maximo de pinceles en archivo ABR
kMaxBrushSize	2500	AbrImporter	Dimension maxima de pincel ABR
kFormatVersion	2	DocumentManager	Version del formato .fap

10. BUGS CRITICOS RESUELTOS (CRONOLOGIA)

Esta seccion documenta cada bug critico resuelto durante el desarrollo, organizado por version. Cada entrada incluye el sintoma, la causa raiz, la solucion implementada, y los archivos afectados. Este registro constituye la memoria tecnica del proyecto y documenta el camino evolutivo completo hasta el estado final.

10.1 Version 2.5.1 — Julio 2026

Pixel Offset Bug (Load Path)

Aspecto	Detalle
Sintoma	Contenido dibujado en el centro del canvas aparecia en la esquina superior izquierda (0,0) despues de guardar y reabrir el proyecto.
Causa Raiz	readLayerData() en document_manager.cpp llamaba ensureContains() con pad=true (parametro por defecto). Esto añadia kGuardBand (2px) + kGrowthPad (512px), expandiendo el buffer y desplazando el origen en CADA carga. La copia de pixeles PNG se escribia en la fila 0 del buffer expandido, colocando todos los pixeles en el offset incorrecto.
Solucion	Cambiar pad=false en la ruta de carga (document_manager.cpp:617). El save path ya escribia correctamente origin_x/origin_y via layerMetadataToJson(). El load path ya los restauraba via layerMetadataFromJson() -> setOrigin(x,y). El unico cambio necesario fue prevenir la re-expansion innecesaria del buffer durante la carga.
Archivos	src/io/document_manager.cpp (1 linea, 1 caracter: pad=true -> pad=false)

Tool State Desync on Load

Aspecto	Detalle
Sintoma	Al abrir un archivo .fap, la herramienta Brush se comportaba como Eraser en capas nuevas. La UI mostraba Brush seleccionado, pero al dibujar borraba en vez de pintar.
Causa Raiz	MainWindowV2::openProject() llamaba resetDocument() que llamaba tool_state_->resetToDefaults() -> setActiveTool(ToolType::Eraser). La senal activeToolChanged se emitia pero nada en MainWindowV2 estaba conectado para sincronizar el QButtonGroup del toolbox. El canvas leia appState_->toolState().activeTool() directamente en cada evento de mouse, detectando Eraser mientras la UI mostraba Brush.
Solucion	Llamar toolbox_panel_->setActiveTool(0) al final del branch de exito de openProject(). Esto sincroniza tanto el button group de la UI como el ToolState de vuelta a Brush. Mismo fix en newProject().
Archivos	src/ui_v2/main_window_v2.cpp (2 lineas en openProject y newProject)

Multi-Sequence Data Loss on Load (CRITICO)

Aspecto	Detalle
Sintoma	Proyectos con 2+ secuencias perdian todo el contenido de capas al reabrir. Los nombres de capa volvian a los defaults ("Layer 1").
Causa Raiz	readTimeline() llamaba doc.addSequence("") para si>0, creando secuencias DUPLICADAS. Pero readManifest() ya las habia creado correctamente. El loop de limpieza eliminaba los

	duplicados (que contenian los datos), dejando las originales vacias.
Solucion	readTimeline() ahora usa doc.sequenceAt(si) en lugar de doc.addSequence(""). Guard: si si >= doc.sequenceCount(), llama addSequence("") como fallback.
Archivos	src/io/document_manager.cpp:604-607 (4 lineas)

Legacy Load — hasContent_ Omitido

Aspecto	Detalle
Sintoma	Cargar archivos legacy v1/v2/v3 (.fap de directorio): los datos de pixeles se cargaban correctamente pero las celdas de la timeline aparecian vacias.
Causa Raiz	hasContent_ nunca se establecia en las 3 rutas de carga legacy. El flag se inicializaba en false y nunca se actualizaba.
Solucion	Anadir rl->setHasContent(true) despues de la copia exitosa de pixeles PNG en los 3 paths legacy (v1, v2, v3).
Archivos	src/io/file_format.cpp:243,283,328 (3 lineas)

10.2 Version 2.5.2 — Julio 2026

Close Confirmation for Unsaved Changes

Aspecto	Detalle
Sintoma	Cerrar la ventana (X, Alt+F4) descartaba cambios no guardados sin advertencia.
Solucion	Anadir closeEvent(QCloseEvent*) override en MainWindowV2. Si isModified(), muestra dialogo Save/Discard/Cancel.
Archivos	src/ui_v2/main_window_v2.hpp:42, src/ui_v2/main_window_v2.cpp:525-548

FFmpeg waitForFinished(-1) Infinite Hang

Aspecto	Detalle
Sintoma	Si FFmpeg se colgaba durante la exportacion de video, toda la aplicacion se congelaba indefinidamente.
Solucion	Cambiar proc.waitForFinished(-1) a proc.waitForFinished(120000) con proc.kill() en timeout.
Archivos	src/io/video_export.cpp:87

NodeGraph::evaluate() Ignoraba el Parametro Target

Aspecto	Detalle
Sintoma	evaluate(RasterLayer& target) computaba el grafo de nodos pero nunca escribia el resultado en target. La salida se descartaba.
Solucion	Despues de evaluateNode(output, cache), copiar los pixeles del OutputNode hacia target via std::memcpy fila por fila, respetando std::min de dimensiones. Establecer bufferEpochTick() y setHasContent(true).
Archivos	src/engine/compositor/node_graph.cpp:376-397 (+11 lineas)

Layer Rename Lost on Focus Change

Aspecto	Detalle
Sintoma	Renombrar una capa y hacer clic en otra sin presionar Enter: el nombre se descartaba silenciosamente.
Causa Raiz	La senal editingFinished estaba desconectada (fix para un crash previo de double-delete QLineEdit). Cuando el editor perdia el foco, no ocurria ningun commit.
Solucion	Reconectar editingFinished con un guard std::make_shared<bool> committed. returnPressed establece committed=true primero; editingFinished verifica el guard. Ambas senales pueden dispararse seguramente — solo la primera realiza el commit.
Archivos	src/ui_v2/layer_panel_v2.cpp:462-476

10.3 Version 2.6 — Julio 2026

#12 — Eraser Tool No-Op

Aspecto	Detalle
Sintoma	Despues de dibujar en multiples capas, seleccionar el borrador e intentar borrar no tenia efecto visible.
Causa Raiz	En drawBrushStamp(), el branch de borrado aplicaba un blend estilo DestinationOut (dst * (255 - srcA) / 255) sobre strokeBuffer_, que se inicializa con pixeles transparentes (fill(0)). Como 0 * cualquier_cosa = 0, las estampas nunca se acumulaban. En commitStroke(), el strokeBuffer_ vacio se compositaba con CompositionMode_DestinationOut — sin borrar nada.
Solucion	Eliminar el branch de borrado roto. Ambos brush y eraser usan el mismo blend SourceOver para acumular estampas en strokeBuffer_. La distincion ocurre en renderStampToImage() (estampas blancas vs de color) y commitStroke() (DestinationOut vs SourceOver).
Archivos	src/ui_v2/canvas_widget_v2.cpp:2157-2176 (-11 lineas, blend unificado)

#13 — Layer Names Overlap Visual

Aspecto	Detalle
Sintoma	Al cambiar de frame durante un renombre en progreso, etiquetas viejas de nombre de capa permanecian visibles superpuestas con las nuevas.
Causa Raiz	QListWidget::clear() elimina QListWidgetItem pero NO destruye los QWidget hijos establecidos via setItemWidget(). Viejos QLabel permanecian como huerfanos invisibles, superponiendose con nuevos widgets en el siguiente paint.
Solucion	Antes de list_->clear(), iterar todos los items y llamar removeItemWidget() + hide() + deleteLater() en cada widget. Usar deleteLater() (no delete) previene crashes cuando lambdas de renombre en progreso aun tienen raw pointers a estos widgets.
Archivos	src/ui_v2/layer_panel_v2.cpp:502-514

#14 — Crash Durante Layer Rename

Aspecto	Detalle
Sintoma	La aplicacion crasheaba abruptamente al renombrar una capa y luego hacer clic en otra capa o frame.
Causa Raiz	La lambda startRename() capturaba raw pointers (QHBoxLayout*, QLineEdit*, QLabel*) a widgets que refreshLayerList() podia eliminar antes de que la senal editingFinished de la lambda se disparara. Acceder a widgets eliminados -> crash use-after-free.
Solucion	Cambiar raw pointers a QPointer<QHBoxLayout>, QPointer<QLineEdit>, QPointer<QLabel>. La lambda ahora verifica if (!pHlay || !pEditor || !pNameLabel) return; antes de acceder a cualquier widget.
Archivos	src/ui_v2/layer_panel_v2.cpp:462-466

#15 — Video Export Unificado + Audio Mixing

Aspecto	Detalle
Sintoma	El codigo de exportacion de video estaba duplicado en video_export.cpp y MainWindowV2::exportVideo(). Formatos MOV/WebM con alpha solo funcionaban desde MainWindowV2. Sin audio en exports. GIF hardcodeado a 320px.
Solucion	Unificar exportVideo() con auto-deteccion de formato por extension. Audio mixing via amix con volumenes por pista. GIF a resolucion completa. MainWindowV2::exportVideo() reducido de 80 a 20 lineas, delegando a fap::exportVideo().
Archivos	src/io/video_export.hpp, src/io/video_export.cpp, src/ui_v2/main_window_v2.cpp:793-818

#16 — Playback Timer Imprecision (FPS Drift)

Aspecto	Detalle
Sintoma	La reproduccion de animacion parecia ligeramente mas lenta que el GIF/video exportado. El FPS mostrado era correcto pero el timing real estaba desfasado.
Causa Raiz	El intervalo del timer usaba division entera: 1000 / fps_. A 24fps esto producia 41ms en vez de 41.67ms. El error se acumulaba con la sobrecarga de renderizado de buildBackgroundCache() para composiciones 1080p multi-capa.
Solucion	Cambiar a static_cast<int>(std::round(1000.0 / fps_)) en setFPS() y el handler del boton play. Ahora usa matematicas de punto flotante con redondeo para mejor precision.
Archivos	src/ui_v2/timeline_panel_v2.cpp:1134,1177

10.4 Version 2.6.1 — Julio 2016

C1 — ABR Importer Endian Swap (Silent x86 Failure)

Aspecto	Detalle
Sintoma	Todos los archivos ABR v1/v2 fallaban silenciosamente en maquinas little-endian (x86). Version leida como 65537 en vez de 1.
Causa Raiz	uint16_t promovido a int antes del bit-shift, produciendo numero incorrecto.
Solucion	Cast a uint16_t despues de los shifts: static_cast<uint16_t>(value).
Archivos	src/engine/brush/abr_importer.cpp:31

C2 — Deformation Mesh Bilinear Interpolation Error

Aspecto	Detalle
Sintoma	El cuadrante inferior derecho de mallas deformadas tenia coordenadas Y incorrectas.
Causa Raiz	p11.y * fy * fy en lugar de p11.y * fx * fy en mapPointBilinear(). Error tipografico en el factor de peso.
Solucion	Corregir el factor de peso a fx * fy.
Archivos	src/engine/deformation/deformation_mesh.cpp:89

C3 — Smoothstep NaN con Hardness >= 1.0

Aspecto	Detalle
Sintoma	Funciones de pincel/render producian NaN cuando hardness >= 1.0f. El renderizado se corrompia.
Causa Raiz	Division (x - edge0) / (edge1 - edge0) = 0/0 = NaN cuando edge0 == edge1. clampf/std::clamp propaga NaN.
Solucion	Anadir guard: if (edge0 == edge1) return (x >= edge1) ? 1.0f : 0.0f; en ambas copias de smoothstepf.
Archivos	src/core/types.hpp:218-220, src/engine/raster/raster_stroke.cpp:10-12

C4 — Pixel Byte Order Conflict (0xAABBGGRR vs 0xAARRGGBB)

Aspecto	Detalle
Sintoma	Canales R/B intercambiados cuando el compositor mezclaba pixeles de diferentes modulos del motor.
Causa Raiz	blend_utils.hpp usaba 0xAABBGGRR (R en byte 0, B en byte 2) mientras raster_stroke.cpp y QImage usaban 0xAARRGGBB. El compositor leia correctamente pero mezclaba con el orden de bytes incorrecto.
Solucion	Estandarizar unpackPixel/packPixel en blend_utils.hpp a 0xAARRGGBB (Qt native). Actualizar tests del compositor.
Archivos	src/engine/common/blend_utils.hpp:12-28, tests/test_compositor.cpp

C5 — Path Traversal en Audio Extraction

Aspecto	Detalle
Sintoma	Un archivo .fap malicioso podria escribir archivos fuera del directorio temporal via ../../ en displayName.
Solucion	Anadir QFileInfo(rawName).fileName() para eliminar componentes de ruta.
Archivos	src/io/document_manager.cpp:797-798

H1 — hasContentBefore en 6 Call Sites

Aspecto	Detalle
Sintoma	Despues de deshacer fill, text, move, selection, o floating selection, las celdas de la timeline mostraban frames vacios aunque los datos de pixeles se habian restaurado.
Causa Raiz	hadBefore = layer->hasContent() se computaba pero nunca se pasaba a pushFullLayerUndo. LayerModifyCommand usaba hadContentBefore_ = false por defecto, por lo que undo siempre limpiaba hasContent_.
Solucion	Pasar hadBefore como 3er argumento en los 6 call sites: doFill, doText, clearTextRaster, commitMove, commitSelection, commitFloatingSelection.
Archivos	src/ui_v2/canvas_widget_v2.cpp (6 lineas)

H2 — Stale Sequence Callbacks

Aspecto	Detalle
Sintoma	Despues de cambiar de secuencia, el callback on_frame_changed_ de la secuencia anterior seguia disparandose, emitiendo currentFrameChanged para la secuencia incorrecta.
Solucion	wireSequenceSignals() ahora limpia callbacks en TODAS las secuencias antes de cablear la activa.
Archivos	src/core/app_state.cpp:41-45

H7 — Compositor Inverted Z-Order

Aspecto	Detalle
Sintoma	Las capas se compositaban en orden inverso — la capa del fondo se dibujaba encima y viceversa.
Causa Raiz	compositeLayer() iteraba GroupLayer con rbegin()/rend(), dibujando la capa superior primero.
Solucion	Cambiar a begin()/end() para el algoritmo del pintor correcto (bottom->top).
Archivos	src/engine/compositor/compositor.cpp:61, tests/test_compositor.cpp

H8 — NodeGraph Dangling Pointers

Aspecto	Detalle
Sintoma	Despues de eliminar un nodo, otros nodos mantenian raw pointers al nodo liberado en sus

	vectores outputs_, arriesgando use-after-free.
Solucion	Anadir un segundo loop en removeNode() para desconectar todas las entradas del nodo a eliminar antes de borrarlo, limpiando las back-references outputs_ en los nodos fuente.
Archivos	src/engine/compositor/node_graph.cpp:329-347

10.5 Version 2.6.2 — Julio 2026

H3 Regression — commitAtomic Falla en Windows

Aspecto	Detalle
Sintoma	Despues de exportar un GIF (que cambia el directorio de trabajo via dialogo nativo de Windows), guardar el proyecto por primera vez mostraba "Failed to save project".
Causa Raiz	El fix H3 removio QFile::remove(finalPath) de commitAtomic() para prevenir TOCTOU, pero en Windows Qt 6.5, QFile::rename() NO sobrescribe archivos existentes. El dialogo de save confirmaba overwrite pero no eliminaba el archivo. commitAtomic intentaba renombrar .tmp sobre el target existente, lo cual fallaba silenciosamente.
Solucion	commitAtomic ahora intenta rename() primero. Si falla y el target existe, remove target + retry rename. Mas seguro que el enfoque pre-H3 porque el remove solo ocurre despues de un rename fallido, minimizando la ventana TOCTOU.
Archivos	src/io/document_manager.cpp:214-237

Undo/Redo — Missing canvasUpdated Signal

Aspecto	Detalle
Sintoma	Despues de Ctrl+Z o Ctrl+Y, el canvas se repintaba correctamente pero las celdas de la timeline y el panel de capas permanecian con el estado visual anterior.
Causa Raiz	CanvasWidgetV2::undo() y redo() llamaban invalidateBackgroundCache() + update() (refrescando solo el canvas) pero nunca emitian canvasUpdated. Esta senal esta conectada en MainWindowV2 a timeline_panel_->update() y layer_panel_->refreshLayerList().
Solucion	Anadir emit canvasUpdated() en ambos undo() y redo().
Archivos	src/ui_v2/canvas_widget_v2.cpp:386-403

10.6 Version 2.7 — Julio 2026

Esta version introdujo el sistema de ocultacion no destructiva de frames y los marcadores estilo After Effects:

- Non-destructive frame hiding: durationFrames_ (visibles) desacoplado de totalFrames_ (datos). Boton + revela frames ocultos, boton - solo reduce visibilidad.
- Timeline Markers: 9 colores AE, bandas de duracion, titulos en pills, dialogo de propiedades con dark theme
- Marker uniqueness: un marcador por posicion de frame
- Marker dialog: QSpinBox con numeracion 1-based, dark theme stylesheet explicito
- Bug fix: updateMarker() ahora respeta la restriccion de unicidad
- Bug fix: Ctrl+Z/Ctrl+Y refrescan timeline y layer panel (emit canvasUpdated)

10.7 Version 2.8 — Julio 2026

Esta version introdujo la importacion de video y docks desacoplables:

- Video Clip Import: probeVideoMetadata() via ffprobe, decodeVideoFrame() via ffmpeg pipe PNG, VideoFrameCache LRU 50 frames, limites de 60s/1080p
- VideoTrackWidget: misma estructura que AudioTrackWidget, thumbnail strip, sliders de opacidad/volumen
- Opacity slider con invalidacion de canvas en tiempo real
- Partial->full rebuild escalation cuando hay video tracks activos
- Audio/Video layout position persistence en save/load
- OnionSkinPanel y CanvasSizePanel extraidos a docks independientes
- 8 docks con tema oscuro y barras de titulo naranja

10.8 Version 2.9 — Julio 2026

Esta version introdujo el efecto Line Boil, secuencia lock con feedback visual, y soporte de tabletas:

- Line Boil: ruido bilineal 8x8, hash determinista por frame, no destructivo (dominio DISPLAY)
- Sequence Lock: boton rojo #FF4A4A en estado checked, overlay rojo en celdas, bloquea dibujo en canvas
- Frame keyboard shortcuts: + y - para duplicar/ocultar frames con auto-repeat de Qt
- Tablet pressure support: QTabletEvent nativo, presion modula tamano (20%-100%) y opacidad
- Tablet event dispatching: generacion de QMouseEvent sinteticos via QApplication::sendEvent()

10.9 Version 2.9.1 — Julio 2026

Esta version introdujo el BusyOverlay y corrigio bugs criticos del sistema de video:

- BusyOverlay: spinner animado 12 lineas, eventFilter bloquea input, keep-alive timer 150ms
- Video decoder recursion guard: thread_local bool s_inDecoder previene repaints recursivos
- VideoTrackWidget: eliminado probe redundante (acepta metadata pre-probada)
- Document manager: processEvents() cada 20 capas durante load para UI responsivo

11. PRUEBAS

11.1 Framework y Configuracion

El proyecto utiliza GoogleTest 1.14.0 como framework de testing, obtenido via CMake FetchContent. Los tests se compilan como un ejecutable separado (fap-tests) que enlaza con las fuentes de Core, Engine, IO y Platform, pero no con la UI (que requiere QApplication).

```
# Build y ejecucion de tests:
cmake -B build -DCMAKE_BUILD_TYPE=Release
cmake --build build --config Release --target fap-tests
ctest --test-dir build
# o directamente:
./build/fap-tests
```

11.2 Cobertura por Modulo

Archivo de Test	Modulo Probado	Enfoque
test_document.cpp	Core — Document	Creacion, secuencias, canvas size, modificacion
test_layer.cpp	Core — RasterLayer	Pixeles, COW, hasContent, ensureContains, clone
test_layer_memory.cpp	Core — Gestion de Memoria	shared_ptr use_count, ensureUnique, COW
test_undo.cpp	Core — UndoManager	Push, undo, redo, clear, limites de stack
test_brush_engine.cpp	Engine — Brush	Presets, presion, estampas, spacing
test_bezier.cpp	Engine — Vector	Curvas Bezier, evaluacion, tangente, aplanado, longitud
test_compositor.cpp	Engine — Compositor	Z-order, blend modes, orden de capas
test_node_graph.cpp	Engine — NodeGraph	evaluate() escribe a target, entradas deshabilitadas, sin output node
test_deformation.cpp	Engine — Deformacion	Interpolacion bilineal, malla, offsets
test_abr_importer.cpp	Engine — ABR Import	Parseo v1/v2, endian swap, RLE decompression
test_ruler_guide.cpp	Engine — Ruler	Guias de linea y elipse, restricciones geometricas
test_tween_engine.cpp	Engine — Tween	Interpolacion de keyframes
test_pencil_retouch.cpp	Engine — Pencil	Suavizado de trazos
test_audio_engine.cpp	Engine — Audio	Playback, decodificacion, waveform
test_file_format.cpp	IO — File Format	Save/load .fap v2, legacy v1/v2/v3, layer names, hasContent
test_memory_stress.cpp	Stress	Operaciones masivas de pixeles, memoria

11.3 Estadisticas

Metrica	Valor

Total de tests	160
Tests que pasan	160 (100%)
Archivos de test	16
Cobertura de modulos Core	Document, Layer, Sequence, UndoManager, AppState, ToolState
Cobertura de modulos Engine	Raster, Vector, Brush, Compositor, NodeGraph, Deformacion, ABR, Ruler, Tween, Pencil, Audio
Cobertura de modulos IO	File Format, Document Manager
Tests de regresion de bugs	9 tests especificos: NodeGraph target, ABR endian, hasContent, z-order, bilinear, smoothstep NaN, etc.

12. ESTADISTICAS DEL PROYECTO

12.1 Lineas de Codigo

Modulo	Archivos .hpp	Archivos .cpp	Estimacion de Lineas
Core	11	8	~3,000
Engine/Raster	2	3	~500
Engine/Vector	2	3	~600
Engine/Brush	5	5	~1,800
Engine/Common	1	0	~80
Engine/Compositor	2	2	~500
Engine/Deformation	2	2	~300
Engine/Animation	6	5	~1,500 (incl. dr_libs headers)
IO	4	5	~2,500
Platform	1	1	~150
UI V2	13	13	~8,000
Main	—	1	~170
TOTAL	49	48	~19,100

12.2 Recursos

Categoria	Cantidad	Detalle
Iconos PNG	45+	Herramientas, timeline, layers, toolbar
Iconos SVG	10+	Audio/video track buttons, herramientas
Fuentes	2	Avenir LT Std (.otf + .ttc)
Presets de pinceles	5	Round Soft/Hard, Flat, Calligraphy, Eraser
Texturas de papel	1+	Directorio resources/textures/
Paletas de color	1	default.json
Skills de agente	37	Directorio .agents/skills/
Documentacion	21	7 activos + 14 archivados

12.3 Dependencias

Dependencia	Version	Tipo	Tamano
Qt 6	6.5+	Framework (DLLs)	~50-80 MB (deploy)
miniz	3.0.2	Libreria estatica C	~80 KB
GoogleTest	1.14.0	Testing (FetchContent)	~1 MB

dr_libs	Public Domain	Headers C (3 archivos)	~3 MB
FFmpeg	Externo	Ejecutable del sistema	~80 MB
Ejecutable final	—	free-animation-power.exe	~3-5 MB + Qt DLLs

12.4 Versiones y Evolucion

Version	Fecha	Features Principales	Bugs Resueltos
v2.0	Jun 2026	Arquitectura inicial, motor raster/vector, UI V2	—
v2.2	Jul 2026	Multi-secuencia, reconstruccion no destructiva	—
v2.3	Jul 2026	Audio waveform (dr_libs), layout fix	Waveform width=0
v2.4	Jul 2026	Work Area, FPS real, audio transport sync	Feedback loop FPS
v2.5.1	Jul 2026	Audio persistence, hasContent_, brush init fix	5 bugs: pixel offset, tool desync, data loss, legacy hasContent
v2.5.2	Jul 2026	Close confirm, video export unificado, layer rename fix	6 bugs: FFmpeg hang, NodeGraph target, rename lost
v2.6	Jul 2026	Copy layer to all frames, frame clipboard, file association	5 bugs: eraser no-op, name overlap, rename crash
v2.6.1	Jul 2026	Auditoria de bugs criticos	8 bugs: ABR endian, bilinear, smoothstep NaN, byte order, path traversal, hasContentBefore, stale callbacks, TOCTOU, z-order, dangling pointers
v2.6.2	Jul 2026	Export resolution dialog, improved error reporting	2 bugs: H3 regression Windows, undo/redo canvasUpdated
v2.7	Jul 2026	Frame hiding no destructivo, Timeline Markers	Marker uniqueness, marker dialog dark theme, 1-based frame
v2.8	Jul 2026	Video clip import, docks desacoplables, tablet pressure	Partial rebuild escalation, layout position persistence
v2.9	Jul 2026	Line Boil, sequence lock, frame shortcuts, tablet dispatching	Tablet event dispatching fix
v2.9.1	Jul 2026	BusyOverlay, recursion guard, processEvents load	3 bugs: recursion guard, redundant probe, UI freeze on load

13. CONCLUSION

Free Animation Power representa un logro significativo en el desarrollo de software de animacion 2D. A lo largo de 12 versiones en un solo mes (julio 2026), el proyecto evoluciono desde una arquitectura basica hasta un sistema completo con mas de 19,000 lineas de codigo, 160 tests, y soporte para flujos de trabajo profesionales de animacion.

13.1 Logros Tecnicos

- Pipeline de renderizado de 4 buffers con separacion estricta DATA/DISPLAY, resultado de 2 dias de debugging intensivo
- Sistema de capas con Copy-on-Write, almacenamiento sparse de frames, y deteccion O(1) de contenido (hasContent_)
- 12 modos de blend con formulas matematicas completas, incluyendo SoftLight con 3 ramas condicionales
- Motor vectorial con curvas de Bezier cubicas, subdivision De Casteljau, y aplanado recursivo
- Sistema de pinceles con 5 presets, soporte de presion de tableta (Wacom/Huion/XP-Pen/Xencelabs), e importacion ABR
- Formato de archivo .fap v2 con ZIP binario, guardado atomico, deduplicacion de pixeles, y backward compatibility v1/v2/v3
- Exportacion unificada de video: MP4 H.264, MOV QuickTime Alpha, WebM VP9 Alpha, GIF a resolucion completa, PNG sequence
- Linea de tiempo multi-secuencia con reconstruccion no destructiva, work area interactiva, y 9 marcadores estilo After Effects
- Pistas de audio y video con embedding en .fap, waveform rendering nativo, y cache LRU de frames
- Efecto Line Boil no destructivo con ruido bilineal deterministico por frame
- Sistema de undo/redo con 128 niveles de profundidad y snapshot completo de capa
- Interfaz de 8 docks desacoplables con tema oscuro corporativo

13.2 Metricas Finales

Metrica	Valor
Lineas de codigo C++	~19,100
Archivos fuente (.cpp/.hpp)	97
Tests automatizados	160 (100% passing)
Modulos independientes	13 (Core, 7 Engine, IO, Platform, UI V2, Main)
Versiones publicadas	12 (v2.0 a v2.9.1)
Bugs criticos resueltos	30+
Herramientas de dibujo	17
Modos de blend	12
Formatos de exportacion	5 (MP4, MOV, WebM, GIF, SVG)
Formatos de audio soportados	3 (WAV, MP3, FLAC)

Dependencias externas	4 (Qt 6, miniz, GoogleTest, dr_libs) + FFmpeg externo

13.3 Lineas Futuras

El proyecto sienta las bases para futuras expansiones:

- Motor vectorial completo con edicion de nodos Bezier y rellenos vectoriales
- Sistema de huesos (bones) para animacion de personajes con cinematica inversa
- Motor de particulas para efectos atmosfericos (lluvia, nieve, fuego)
- Color grading y LUTs para correccion de color profesional
- Integracion con servicios cloud para colaboracion en tiempo real
- Soporte para plugins via API de scripting (Lua/Python)
- Compilacion nativa para macOS y Linux
- Aceleracion por GPU via OpenGL/Vulkan para el pipeline de display