



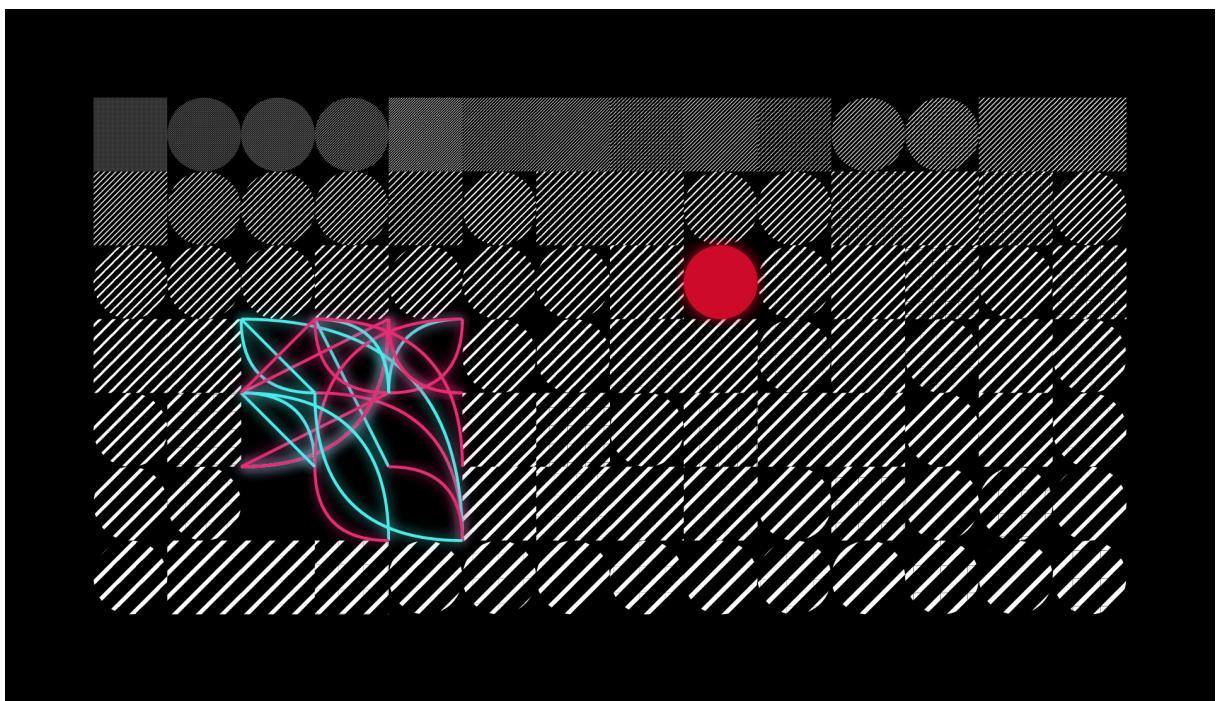
Shapemaker: Créations audiovisuelles procédurales musicalement synchrones

Gwenn Le Bihan

gwenn.lebihan@etu.inp-n7.fr

ENSEEIHT

27 Mars 2025



```

use shapemaker::*;
use rand;

pub fn main() {
    let mut canvas = Canvas::with_colors(ColorMapping {
        black: "#000000".into(),
        white: "#ffffff".into(),
        red: "#cf0a2b".into(),
        green: "#22e753".into(),
        blue: "#2734e6".into(),
        yellow: "#f8e21e".into(),
        orange: "#f05811".into(),
        purple: "#6a24ec".into(),
        brown: "#a05634".into(),
        pink: "#e92e76".into(),
        gray: "#81a0a8".into(),
        cyan: "#4fecac".into(),
    });
    canvas.set_grid_size(16, 9);
    canvas.set_background(Color::Black);

    let draw_in = canvas.world_region.resized(-2, -2);

    // Strands

    let strands_in =
        Region::from_bottomleft(draw_in.bottomleft().translated(2, -1), (3, 3))
            .unwrap();

    canvas.add_layer(canvas.n_random_curves_within(&strands_in, 30, "strands"));

    for (i, obj) in canvas.layer("strands").objects.values_mut().enumerate() {
        obj.recolor(if i % 2 == 0 { Color::Cyan } else { Color::Pink });
        obj.filter(Filter::glow(4.0));
    }

    // Red dot

    let red_dot = Object::BigCircle(
        Region::from_topright(draw_in.topright().translated(-3, 1), (4, 3))
            .unwrap()
            .random_point(),
    )
    .colored(Color::Red)
    .filtered(Filter::glow(5.0));

    canvas.new_layer("red dot").add(red_dot.clone());

    // Hatched circles & squares

    let hatches = canvas.new_layer("hatches");

    for (i, point) in draw_in.except(&strands_in).enumerate() {
        if red_dot.region().contains(&point) {
            continue;
        }
        if rand::random() {
            Object::BigCircle(point)
        } else {
            Object::Rectangle(point, point)
        }
        .filled(Fill::Hatches(
            Color::White,
            Angle(45.0),
            (i + 5) as f32 / 10.0,
            0.25,
        ))
        .add_to(hatches);
    }

    canvas.render_to_png("dna-analysis-machine.png", 480).unwrap();
}

```

Table des matières

1	Introduction	5
1.1	À la recherche d'une impossible énumération des formes	5
1.2	Une approche procédurale ?	6
1.3	Excursion dans le monde physique	7
1.3.1	Interprétation collective	7
1.4	Lien musical	9
2	Une <i>crate</i> Rust avec un API sympathique	10
2.1	Découpage en modules	11
3	Rendu en images	11
4	Render loop et hooks	13
5	Sources de synchronisation	16
5.1	Amplitudes des <i>stems</i>	16
5.1.1	Accessibilité	17
5.1.2	Performance	17
5.1.3	Faisabilité	17
5.2	Export MIDI	17
5.2.1	Faisabilité	17
5.2.1.1	Ticks MIDI	17
5.2.2	Performance	19
5.2.3	Accessibilité	19
5.2.4	Conclusion	20
5.3	Fichier de projet	20
5.3.1	FL Studio	20
5.3.1.1	Performance	21
5.4	Dépôt de « sondes » dans le logiciel de MAO	21
5.5	Temps réel: WASM et WebMIDI	23
6	Performance	24
6.1	Rastérisation parallèle	26
6.2	Encodage H.264 parallèle?	26
6.3	Pixmap et frames HWC: 100ms de standards	26
6.3.1	Aller plus loin	28
6.4	SVG vers string vers SVG	28
7	Conclusion	28
7.1	Pistes d'améliorations	28
7.1.1	Feedback loop	28
7.1.2	Un langage de scripting	29
7.2	Code source	29
7.3	Exemples	29
Remerciements	29	
Bibliographie	29	
Annexes	33	
A Marqueurs dans un logiciel de MAO	33	
B Série « interprétation collective » 1	33	

1 Introduction

1.1 À la recherche d'une impossible énumération des formes

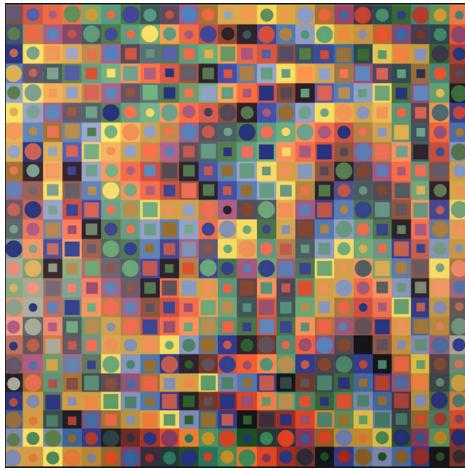


Fig. 1. – *MAJUS* [1]

Avec cette idée en tête, je me mets à gribouiller une ébauche d'« alphabet des formes », qui, naïvement, cherche à énumérer toutes les formes constructibles à partir de formes simples, en autorisant les superpositions, rotations et translations.

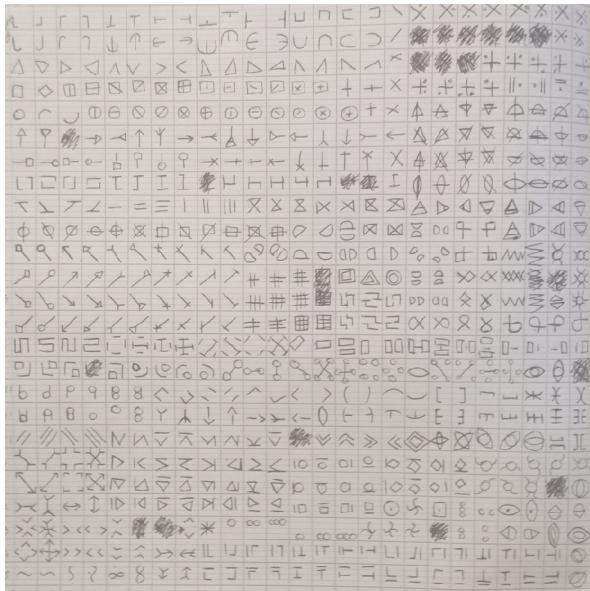


Fig. 2. – Un “alphabet” incomplet

Principalement par simple intérêt esthétique, je vectorise cette page via Illustrator. Vectoriser signifie convertir une image bitmap, représentée par des pixels, en une image vectorielle, qui est décrite par une série d'instructions permettant de tracer des vecteurs (d'où le nom), leur ajouter des attributs comme des couleurs, des règles de remplissage (Even-Odd, Non-Zero, etc.), des effets de dégradés, etc.

Un aspect intéressant des images vectorielles est que, parmi les différents formats les décrivant, le *SVG*, pour *Scalable Vector Graphics*, est indéniablement le plus populaire, et est un standard ouvert décrivant un format texte.

Fascinée depuis longtemps par les œuvres du plasticien et artiste Op-Art *Victor Vasarely*, j'ai été saisie par une de ses périodes, la période « Planetary Folklore », pendant laquelle il a expérimenté à travers plusieurs œuvres autour de l'idée d'un alphabet universel fait de combinaisons simples de formes et couleurs. D'apparence très simple, ces combinaisons sont, d'une manières assez fascinante, uniques, d'où l'idée d'alphabet [2].

En particulier, un tableau, *MAJUS*, implémente à la fois ce concept, et est également une transcription d'une fugue de Bach.

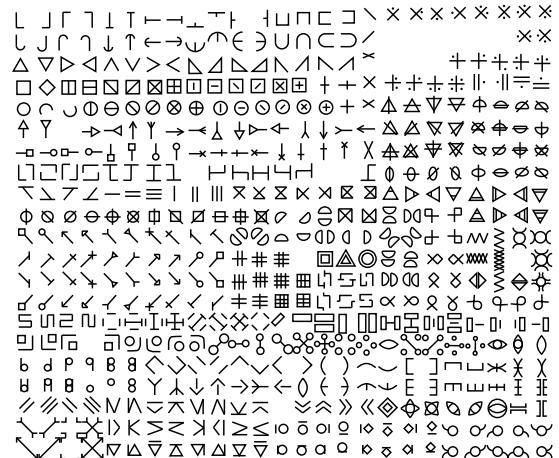


Fig. 3. – Une vectorisation

Il est donc très facile de programmatiquement générer des images vectorielles à travers ce format.

1.2 Une approche procédurale ?

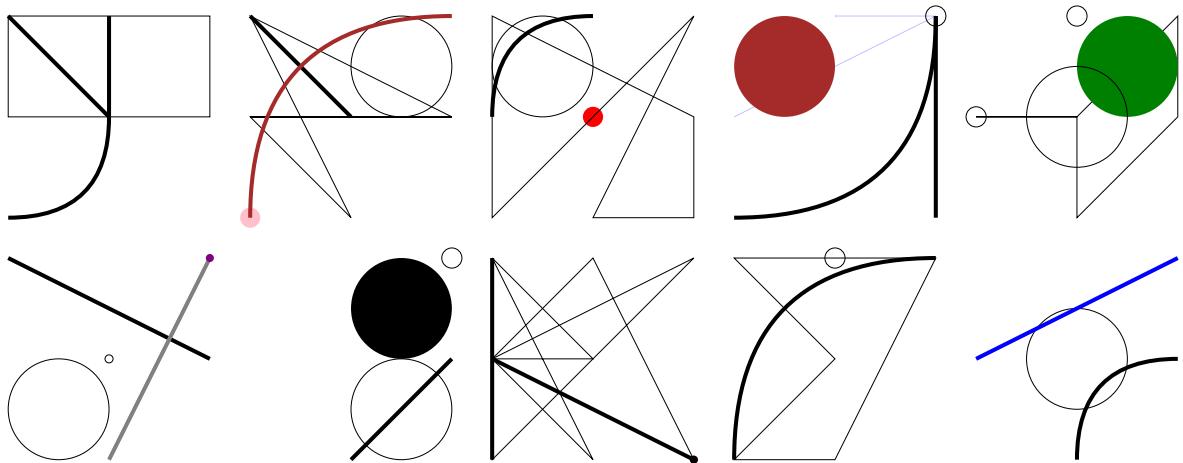


Fig. 4. – Exemples d'oeuvres résultant d'une procédure de génération semi-aléatoire

L'étape prochaine dans cette démarche était donc de générer procéduralement ces formes. Afin d'avoir des résultats intéressants, et devant l'évidente absurdité d'un projet d'énumération complète de toutes les formes, on préférera des générations procédurales dites « semi-aléatoires », dans le sens où certains aspects du résultat final sont laissés à l'aléatoire, comme le placement des formes élémentaires, tandis que d'autres, comme la palette de couleurs, sont des décisions de l'artiste.

Le modèle choisi dans les premières ébauches de Shapemaker est le suivant:

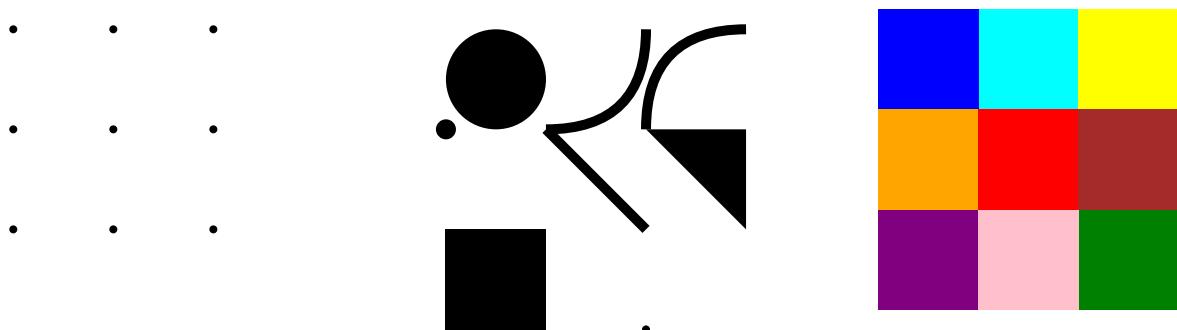


Fig. 5. – Vocabulaire visuel des premières ébauches: grille de placement à 9 points, formes et couleurs

L'idée est donc de limiter la part d'aléatoire à des choix dans des ensembles prédéfinis d'éléments, que ce soit dans le choix des couleurs, des placements ou des formes élémentaires.

Cette méthode amène donc l'artiste à définir, d'une certaine manière, son *propre langage visuel*, où les éléments de langage sont les couleurs, formes, placements et post-traitements (flou, rotations, etc) utilisables.

La part aléatoire engendre une infinité réduite d'œuvres, qui naissent dans les confins du langage visuel défini par l'artiste.

1.3 Excursion dans le monde physique



Fig. 6. – Planches d'impression (merci à Relais Copies [3])

Bien évidemment, les décisions dans le processus créatif ne s'arrêtent pas au choix du vocabulaire visuel utilisé par le processus de génération. zdz Étant donné la simplicité avec laquelle l'on peut dériver de grandes quantités d'œuvres à partir du même langage, la *sélection des meilleures œures* influe évidemment sur la série exposée et/ou partagée.

C'est dans cette optique que j'ai réalisé une série d'impressions de 30 générations, dont certaines ont été légèrement retouchées après génération.

1.3.1 Interprétation collective

Avec 30 œuvres (cf Annexe B) abstraites sans nom, je me suis posé la question de comment les nommer. J'aurais pu les nommer au gré de ma propre imagination, mais j'ai trouvé intéressant le faire de laisser cette décision au grand public, qui tomberait né à né avec ces manifestations de pseudo-hasard virtuel.

Le choix du nom d'une œuvre, en particulier quand elle est aussi abstraite et dénuée de contexte explicite, peut se faire parmi une potentielle infinité de titres, du littéral, au descriptiviste au poétique.

Les œuvres possèdent toutes un QR code amenant sur une page web qui permet de (re)nommer l'œuvre, en y apposant optionnellement son nom, en l'adoptant jusqu'à ce que lea prochain-e n'en prenne la garde.

J'ai donc laissé le public trouver ces œuvres, cachées à travers la ville, dans l'esprit des fameux *Spaces Invaders* de Paris [4] (qui d'ailleurs étendent leur colonisation bien au-delà de Paris, allant même jusqu'à l'ISS [5]).

Certaines ont été souvent renommées, beaucoup ont disparues, et certaines restent encore inconquises.

Fig. 7. – « Reflets Citadins », nommée par *Enide*

Fig. 8. – « Paramount »

Fig. 9. – « l'envolée du Cerf-Volant », nommée par *Nicolas C.*

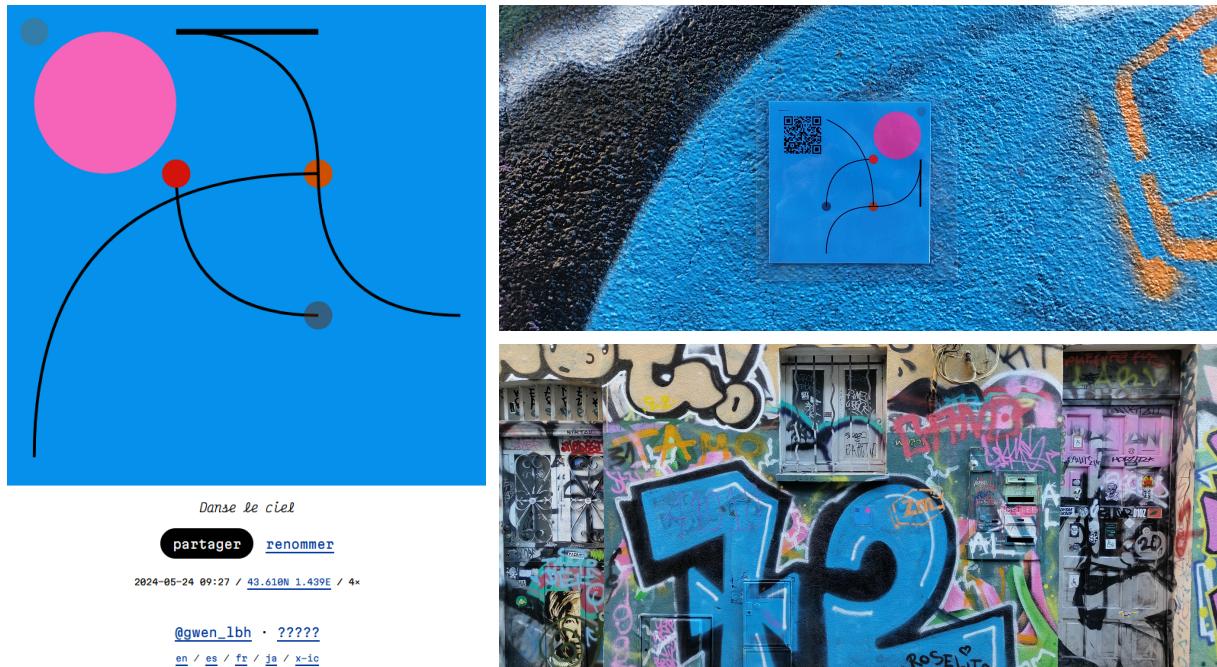
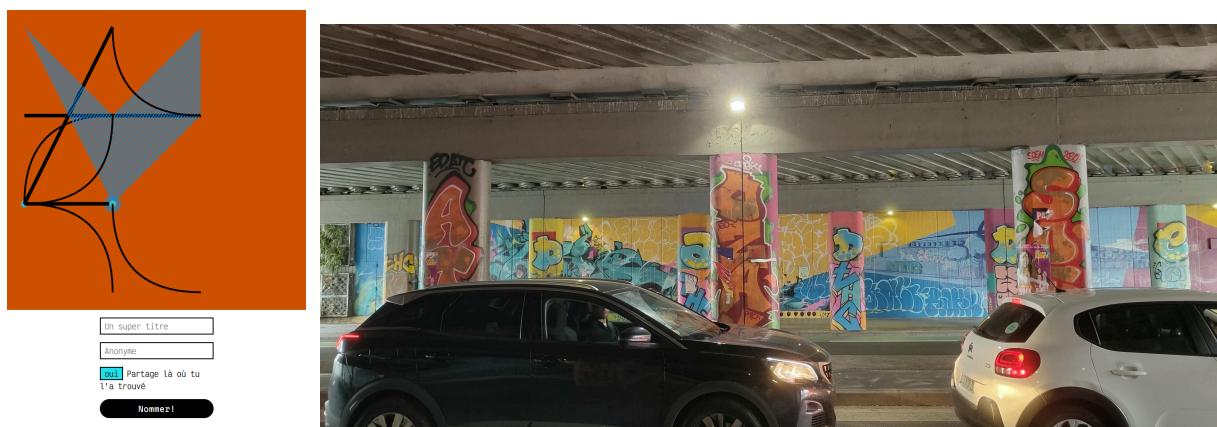


Fig. 10. – « Danse le ciel »

Fig. 11. – *Sans titre*

1.4 Lien musical

Fig. 12. – Frames d'une *story* Instagram montrant une première esquisse de vidéo

À force de générer des centaines de petites images géométriques, il m'est venu à l'idée de les transformer en frames d'une *vidéo*.

Afin d'évaluer à quoi pourrait ressembler une telle chose, j'ai commencé par simplement faire une boucle, écrasant un même fichier .png à un intervalle de temps régulier, fichier ouvert dans XnView [6], qui se recharge automatiquement quand le fichier affiché change.

Bien évidemment, surtout s'il s'agit d'une vidéo synchronisée à sa bande son, il ne suffit pas de générer une frame aléatoire chaque seconde. Il faut pouvoir *réagir à des moments et rythmes clés du morceau*.

2 Une *crate* Rust avec un API sympathique

Pour implémenter cette génération, il faut donc donner un moyen à l'artiste de décrire son langage visuel.

Ainsi, Shapemaker est une bibliothèque, ou *crate* dans l'écosystème Rust [7], dont l'on peut se servir pour créer son script, dont un exemple est montré page 3.

La procédure est conceptualisée par un canvas, composé de une ou plusieurs couches ou *layers* d'objets. Ces objets sont *colorés* (possèdent une information sur la manière dont il faut les remplir: bleu solide, hachures cyan, etc.), et peuvent également subir des filtres et transformations¹. Ils sont aussi *placés* dans l'espace du canvas: le canvas possède une information de *région*, un intervalle 2D de points valables. Les objets se placent dans cette région, en stockant en leur sein les coordonnées de *points* marquant leur positionnement dans l'espace (par exemple, `Object::Rectangle` stocke deux `Point` pour définir ses coins)

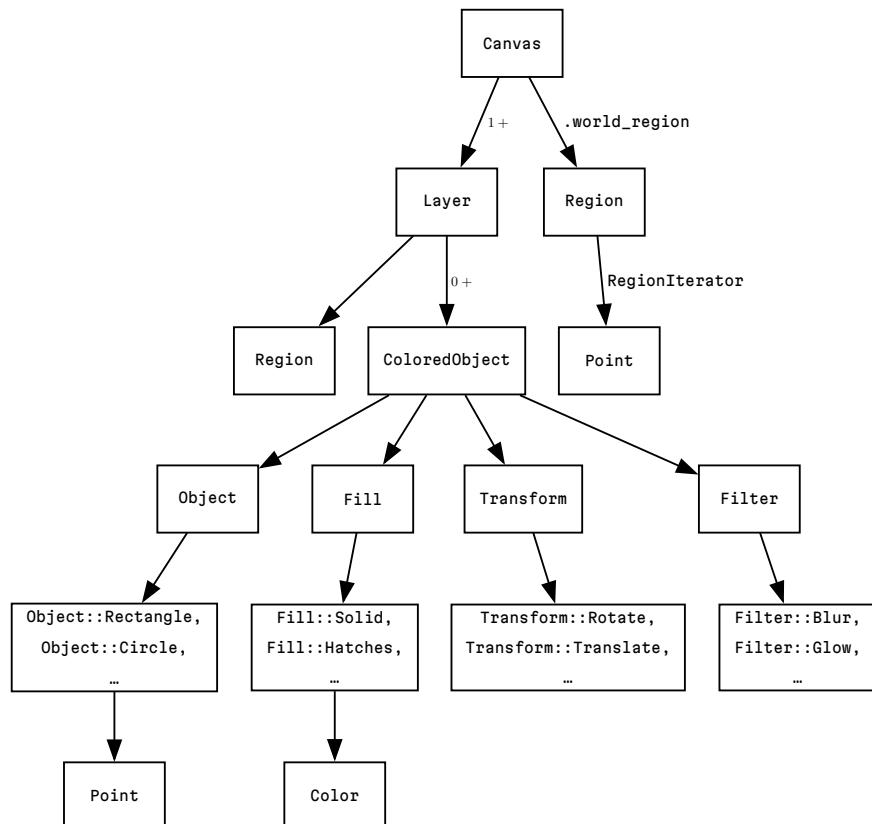


Fig. 13. – Modèle objet du Canvas

¹Avec un peu de recul, le terme d'objet texturé est plus approprié, mais le code n'a pas encore changé

Ce modèle mental permet de travailler plus efficacement car il est bien plus proche de la manière dont on a tendance à penser l'art visuel: sur Illustrator par exemple, ce sont des objets, organisés en plusieurs couches, qui possèdent des attributs dictant leur remplissage.

Les concepts de transformations et de filtres sont également très proche de ce qu'on peut retrouver dans des logiciels de traitement d'images raster, comme Photoshop.

2.1 Découpage en modules

Pour rendre la bibliothèque plus claire, et pouvoir éventuellement séparer la crate en plusieurs sous-crates et ainsi améliorer la vitesse de compilation [8] dans le futur, la crate est découpée en plusieurs modules:

geometry partie purement géométrique, définissant `Point`, `Region` et leurs opérations associées

graphics définitions des objets et tout leurs aspects visuels (`Fill`, `Transform`, `Filter`, `Color`, `Object`, `ColoredObject`)

random fonctions de génération aléatoire, permettant d'introduire facilement et de manière plus ou moins granulaire, une part d'aléatoire dans le processus de génération: `Region.random_point()`, `Color::random()`, etc.

rendering implémentation du rendu en SVG et PNG

video cf Chapitre 4

synchronization cf Chapitre 5

vst cf Chapitre 5.4

wasm cf Chapitre 5.5

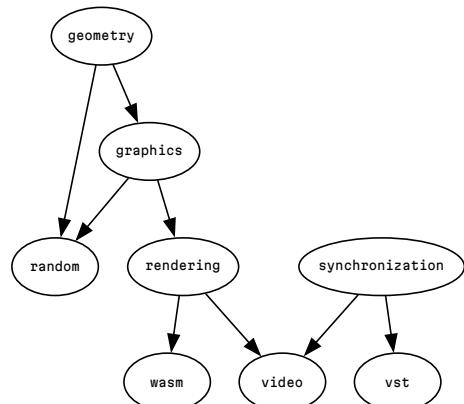


Fig. 14. – Dépendances entre les modules de la bibliothèque

3 Rendu en images

Maintenant que l'on a cette structure, il est bien évidemment essentiel de pouvoir l'exporter en un fichier image exploitable, en PNG par exemple.

L'idée est d'utiliser le standard SVG et tout l'écosystème existant autour, pour éviter d'avoir à ré-implémenter un moteur de rastérisation à la main: SVG possède déjà énormément de fonctionnalités, et faire ainsi nous permet également d'avoir un « escape hatch » et de fournir à Shapemaker des fragments de code SVG pour des cas spécifiques que la bibliothèque ne couvrirait pas, à travers `Object::RawSVG`, qui prend en argument un arbre SVG brut.

Ce processus de rendu est réalisé via l'implémentation d'un *trait*, une sorte d'équivalent en Rust des interfaces présentes dans les langages orientés objet [9]:

```
pub trait SVGRenderable {
    fn render_to_svg(
        &self,
        colormap: ColorMapping,
        cell_size: usize,
        object_sizes: ObjectSizes,
        id: &str,
    ) -> Result<svg::node::element::Element>;
}
```

Ce *trait* est ensuite implémenté par la plupart des structures de `shapemaker::graphics`, de la façon suivante:

Canvas rendu de toutes ses **Layer**, en prenant garde à les ordonner correctement pour que les premières couches soit dessinées par dessus les dernières

Layer rendu de l'ensemble des **ColoredObject** qu'elle contient, en les regroupant dans un groupe SVG `<g>`, ce qui garanti l'ordre de superposition des objets qu'elle contient

ColoredObject rendu de l'**Object** qu'il contient, en appliquant les transformations et filtres

Object dépend de la variante: `Object::Rectangle` est rendu comme un `<rect>`, `Object::Circle` est rendu comme un `<circle>`, etc.

Fill dépend de la variante: simple attribut SVG `fill` pour `Fill::Solid`, utilisation de `<pattern>` pour `Fill::Hatches`, etc.

Transform attribut SVG `transform`

Filter définition d'un `<filter>` avec les attributs correspondants

Color utilise le **ColorMapping** donné pour réifier sa variante² en une valeur de couleur SVG (en notation hexadécimale)

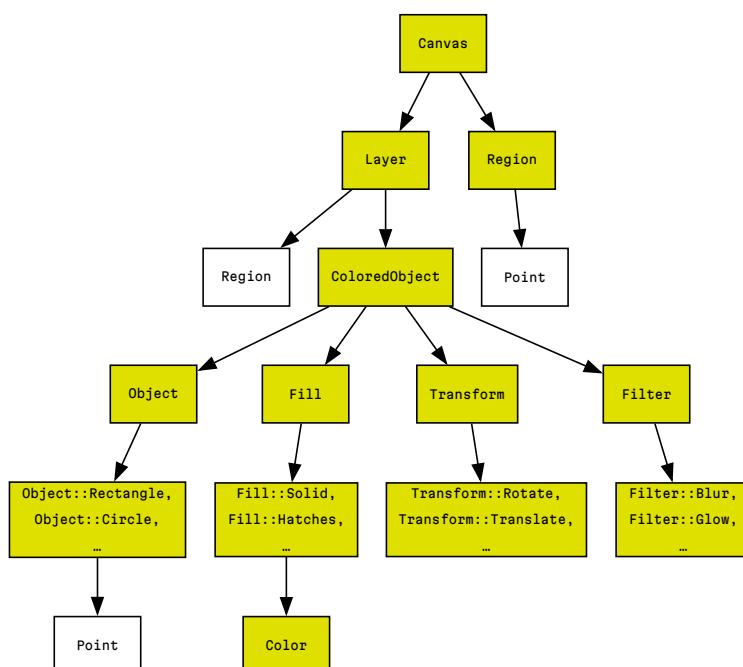


Fig. 15. – Objets rendables en SVG

² « variante » dans le sens des *variantes d'un enum*, **Color** étant un enum de couleurs nommées, `Color::Black`, `Color::Pink`, etc.

Les arguments `cell_size` et `object_sizes` permettent de réaliser en valeur concrètes (pixels) les valeurs de taille abstraites: la distance unitaire entre deux points est définie par `cell_size`, et les tailles des objets, qui, par choix, ne sont pas finement contrôlables, sont définies par `object_sizes`.

```
pub struct ObjectSizes {
    pub empty_shape_stroke_width: f32,
    pub small_circle_radius: f32,
    pub dot_radius: f32,
    pub default_line_width: f32,
}
```

Liste 1. – Définition du type de `ObjectSizes`

En suite, pour convertir en PNG, on utilise une autre bibliothèque, `resvg`, qui implémente presque complètement la spécification SVG 1.1, et l'implémente même mieux que Firefox, Safari et Chrome [10]. L'arbre SVG que l'on a construit est sérialisé en string, puis parsé par `resvg`³, qui le transforme en un arbre de rendu, qui est ensuite rasterisé en une pixmap⁴, qui est finalement encodée en PNG puis écrite dans un fichier.

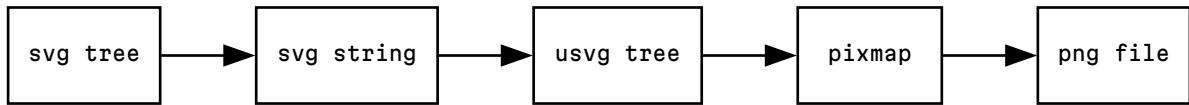


Fig. 16. – Rendu d'un canvas SVG en PNG

4 Render loop et hooks

On peut maintenant rastérer un canvas. Passer à l'étape vidéo consiste donc à réaliser cette opération sur chaque *frame* de la vidéo finale. Cependant, la vidéo devant se synchroniser au son, la tâche est rendue plus difficile: en effet, il ne suffit pas d'exposer à l'artiste une fonction `render_frame`, qui prendrait en argument le numéro de frame actuel et permettrait de définir le canvas pour chaque frame: on a besoin de *réagir* à des moments clés de la musique.

Pour donner les moyens à l'artiste d'exprimer cela, on utilise un concept assez commun en programmation, les *hooks*, nommés ainsi car, essentiellement, ils permettent à du code utilisateur de s'immiscer dans certains moments de l'exécution d'une bibliothèque [11].

Dans notre cas, on va donner les hooks suivants:

each_beat Appelé sur chaque battement de la musique

on_note Appelé à chaque début de note jouée, par un ou des instruments en particulier à préciser

at_timestamp Appelé une fois, à un instant précis de la vidéo

... et pleins d'autres

Un **Hook** est constitué de deux fonctions: `when` pour savoir si le hook doit être exécuté à un instant donné, et `render_function` qui décrit les modifications à effectuer sur le canvas.

³Ce choix à première vue étonnant, qui consiste une perte de performance, est discuté au Chapitre 6.4, page 28

⁴Matrice plate de pixels RGBA

```

pub struct Hook<C> {
    pub when: Box<HookCondition<C>>,
    pub render_function: Box<RenderFunction<C>>,
}

pub type HookCondition<C> = dyn Fn(&Canvas, &Context<C>, BeatNumber, FrameNumber) -> bool;

pub type RenderFunction<C> = dyn Fn(&mut Canvas, &mut Context<C>) -> Result<()>;

```

Un hook reçoit notamment une référence mutable au Canvas, `&mut Canvas`, car il *modifie le canvas de la frame en cours*. Le moteur de rendu vidéo ne possède en effet qu'un seul canvas, qui est successivement modifié au cours de la vidéo.

Le paramètre générique `<C>` existe car l'artiste peut définir des données additionnelles à stocker dans le contexte, ce dernier étant partagé entre les différentes exécutions des hooks. Par exemple: « quelle a été la dernière ligne de parole affichée? il faut passer à la prochaine »

On met également à disposition une méthode `with_hook`, qui rajoute un hook à la liste, permettant de facilement les définir:

```

impl Video<C> {
    ...
    pub fn with_hook(self, hook: Hook<C>) -> Self {
        let mut hooks = self.hooks;
        hooks.push(hook);
        Self { hooks, ..self }
    }
}

```

Voici par exemple la définition du hook `on_note`:

```

impl Video<C> {
    ...
    pub fn on_note(
        self,
        stems: &'static str,
        render_function: &'static RenderFunction<C>,
    ) -> Self {
        self.with_hook(Hook {
            when: Box::new(move |_, ctx, _, _| {
                stems
                    .split('!')
                    .map(|stem_name| ctx.stem(stem_name.trim()))
                    .any(|stem| stem.notes.iter().any(|note| note.is_on())))
            }),
            render_function: Box::new(render_function),
        })
    }
}

```

Le moteur de rendu vidéo est donc une boucle qui, à chaque itération, regarde dans l'ensemble des *hooks* enregistrés, exécute ceux qui le demande, puis rastérise le canvas en une frame qui est ensuite donnée à l'encodeur vidéo:

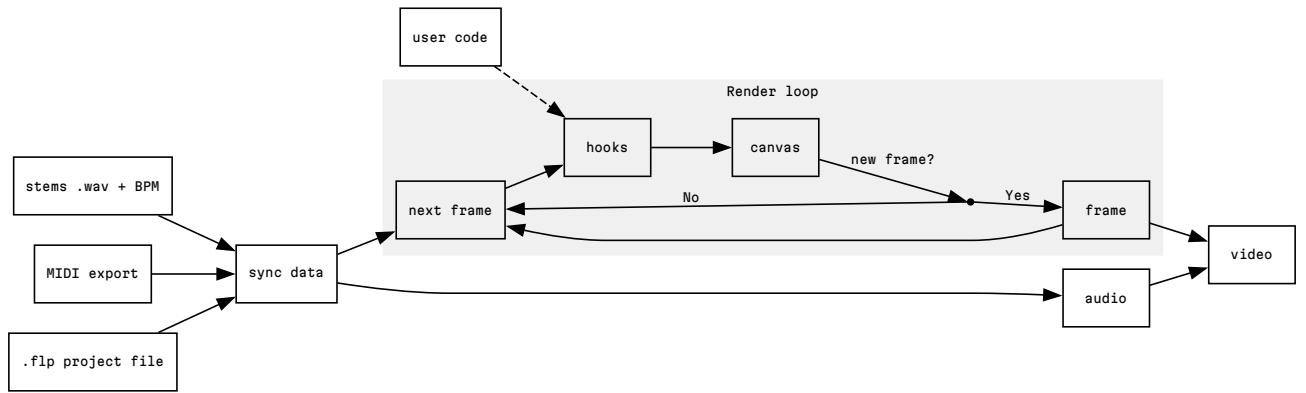


Fig. 17. – Pipeline

La boucle de rendu en elle-même itère sur **les instants de la vidéo, milliseconde par milliseconde, et non pas les frames**. C'est important pour garder la vidéo en synchronisation avec le son. J'avais initialement fait itérer la boucle sur les frames, et la vidéo se décalait progressivement de sa bande son⁵.

```
let render_ms_range = self.start_rendering_at..self.duration_ms();

let mut frames_to_encode: Vec<(Time, String)> = vec![];

for _ in render_ms_range.into_iter() {
    context.ms += 1_usize;
    context.frame = self.fps * context.ms / 1000;
```

On exécute bien les hooks à chaque itération de la boucle, mais par contre on ne rend une nouvelle frame uniquement si le numéro de frame change:

```
if context.frame != previous_rendered_frame {
    frames_to_encode.push((
        Time::from_secs_f64(context.ms as f64 * 1e-3),
        stringify_svg(canvas.render_to_svg(
            canvas.colormap.clone(),
            canvas.cell_size,
            canvas.object_sizes,
            """",
        )?),
    ));
    written_frames_count += 1;

    previous_rendered_beat = context.beat;
    previous_rendered_frame = context.frame;
}
```

La rastérisation et l'encodage sont réalisés après la fin de la boucle de rendu pour pouvoir paralléliser la rastérisation (voir Chapitre 6.1).

⁵Ma théorie est qu'il faut itérer sur un sorte de dénominateur commun des deux pas temporels, sachant les informations de synchronisation de la musique ont un pas de temps bien plus court que le FPS de la vidéo

5 Sources de synchronisation

On a pu voir dans les exemples de code précédents que les hooks reçoivent deux arguments essentiels dans leur fonctions: le *canvas*, discuté précédemment, et un *contexte*.

Ce contexte, en plus de quelques informations déposées par la boucle de rendu (milliseconde actuelle, numéro de frame actuel, etc), contient surtout *des informations musicales sur l'instant présent*, comme les notes actuellement jouées, les amplitudes instantanées de chaque piste, etc.

Afin d'obtenir ces informations, il faut bien analyser quelque chose—la question est donc: de quels fichiers ou signaux tirer parti pour construire ces informations de synchronisation?

Les sous-sections suivantes traitent des différentes approches explorées:

Amplitudes des *stems* utilisation des signaux audio bruts depuis des exports piste par piste du morceau

Analyse de fichiers MIDI utilisation d'un standard stockant les notes jouées dans le temps.

Analyse de fichiers .flp utilisation des fichiers de projet de FL Studio, un logiciel de production musicale. C'est l'équivalent d'un fichier source en programmation, là où l'export .mp3 serait l'équivalent d'un exécutable.

Sondes dans le logiciel de MAO⁶ utilisation de plugins VST pour envoyer des informations de synchronisation potentiellement arbitraire, directement depuis le logiciel de production musicale.

Temps réel utilisation de signaux MIDI en « live », solution contournant le problème de la synchronisation et toute la partie rendu vidéo et rastérisation. Plutôt prévue pour un autre cas d'usage, les concerts et installations live

Dans chacun de ces cas, l'objectif est de pouvoir inférer depuis ces ressources les informations suivantes:

- Le BPM⁷, avec éventuellement des évolutions au cours du morceau
- Des marqueurs temporels, permettant de réagir à des changements de phrases musicales (par exemple, la classique construction *build-up / drop / break* en EDM⁸), sans avoir à coder en dur un timestamp dans le code de la vidéo. Ces marqueurs sont placés dans le logiciel de production musicale (cf Fig. 26, page 33)
- Pour chaque instrument, et à chaque instant:
 - Les notes jouées: pitch⁹ et vitesse¹⁰
 - Des éventuelles évolutions de paramètres influant sur le timbre de l'instrument (ouverture d'un filtre passe bas pour un synthétiseur, pédale de sustain pour un piano, etc)

5.1 Amplitudes des *stems*

Cette approche consiste à demander à l'artiste de fournir un fichier audio par piste du morceau de musique. La définition de « piste » est ici assez vague. Plus le nombre de fichiers est grand, plus il est possible de réagir à des changements d'amplitudes individuels. En général, une piste à un instrument.

⁶MAO: Musique Assistée par Ordinateur

⁷Beats per minute, aussi appelé tempo

⁸Electronic Dance Music

⁹hauteur

¹⁰intensité avec laquelle la note a été jouée

5.1.1 Accessibilité

Exporter un projet en fichiers audios piste-par-piste, des *stems*, est une pratique plutôt courante, par exemple lors de concours de remix [12], pour fournir aux participant·e·s les éléments du morceau séparés et ainsi faciliter la création d'un remix.

On pourrait faciliter encore plus l'usage en, par exemple, proposant de faire de la séparation de source par réseaux neuronaux si l'artiste ne peut pas ou ne souhaite pas faire un export en stems [13]. Cette approche resterait pertinente même en cas de résultats habituellement considérés comme insatisfaisants dans le domaine de la séparation de sources, étant donné que l'on ne s'en sert qu'à des fins d'analyse pour de la synchronisation – l'export audio final du morceau fournit à lui seul la bande son de la vidéo.

5.1.2 Performance

Néanmoins, ce processus demande de remplir une structure de donnée avec des amplitudes à chaque *instant*, ce qui est assez coûteux, que ce soit en temps de calcul ou en mémoire.

5.1.3 Faisabilité

La correspondance signal \mapsto note jouée est beaucoup moins évidente qu'elle ne le paraît. Un signal peut être décomposé en amplitude et fréquence, mais une note possède deux caractéristiques bien plus utiles aux musicien·ne·s:

Vélocité \leftrightarrow amplitude Les amplitudes d'un signal sont très variables, et il est difficile de déterminer un seuil de détection efficace pour considérer qu'une note a été jouée, surtout en la présence d'effets (en particulier de l'echo ou de la réverbération).

Pitch \leftrightarrow fréquence Pour obtenir le pitch d'une note, il faut effectuer une analyse fréquentielle du signal. Ceci pourrait à priori ne pas être trop complexe, mais n'a pas été tenté étant donné les difficultés soulevées par le point précédent. Il est en plus très difficile de séparer plusieurs notes d'un accord.

5.2 Export MIDI

Cette méthode consiste d'une certaine manière à prendre le problème « à l'envers » par rapport à la méthode précédente: on part d'information *sur les notes jouées*, desquelles on peut dériver les amplitudes, depuis la vélocité.

5.2.1 Faisabilité

Le format MIDI [14] permet de spécifier:

- Pour chaque piste: les notes jouées (pitch et vélocité)
- Pour le morceau dans sa globalité, le BPM

Bien que l'on puisse assez facilement inférer une sorte d'amplitude simulée à partir des vélocités, le problème opposé se pose: si l'on veut animer un objet en prenant en compte les échos, par exemple, MIDI ne peut pas nous aider.

Mais pour de nombreux usages, le résultat final paraît beaucoup plus « en réaction avec la musique » qu'avec une approche par amplitudes réelles, certainement grâce à la précision apportée par le fait d'utiliser les événements de notes jouées « à la source ».

Ticks MIDI

Pour l'implémentation, rien de bien compliqué, on rajoute les notes une à une dans notre structure de données en partant des évènements MIDI:

```
match message {
    MidiMessage::NoteOn { key, vel } | MidiMessage::NoteOff { key, vel } => {
        stem_notes
            .entry(absolute_tick_to_ms[tick] as u32)
            .or_default()
            .insert(
                track_name.clone(),
                Note {
                    tick: *tick,
                    ms: absolute_tick_to_ms[tick] as u32,
                    key: key.as_int(),
                    vel: if matches!(message, MidiMessage::NoteOff { .. }) {
                        0
                    } else {
                        vel.as_int()
                    },
                },
            );
    }
    _ => {}
}
```

...Sauf que les coordonnées temporelles MIDI sont en *deltas de ticks MIDI*. Les ticks sont indépendant du BPM, et les deltas sont de simples différences du nombre de ticks passés entre deux évènements.

La durée d'un tick est aussi dépendante du *PPQ*, ou *Pulse per quarter*, qui correspond à la résolution temporelle d'un fichier MIDI – c'est l'équivalent des FPS en vidéos ou de la fréquence d'échantillonnage en audio [15].

```
fn midi_tick_to_ms(tick: u32, tempo: usize, ppq: usize) -> usize {
    let with_floats = (tempo as f32 / 1e3) / ppq as f32 * tick as f32;
    with_floats.round() as usize
}
```

Pour passer de ticks à des millisecondes réelles, il faut réifier ces ticks en lisant le BPM, **qui peut changer au cours du morceau**. Les changements de BPM sont des évènements MIDI parmi le stream plat du fichier.

```

// Convert deltas to absolute ticks
let mut track_no = 0;
for track in midifile.tracks.iter() {
    track_no += 1;
    let mut absolute_tick = 0;
    for event in track {
        absolute_tick += event.delta.as_int();
        timeline
            .entry(absolute_tick)
            .or_default()
            .insert(track_names[&track_no].clone(), *event);
    }
}

// Convert ticks to ms
let mut absolute_tick_to_ms = HashMap::<u32, usize>::new();
let mut last_tick = 0;
for (tick, tracks) in timeline.iter().sorted_by(|(tick, _)| *tick) {
    for event in tracks.values() {
        if let TrackEventKind::Meta(MetaMessage::Tempo(tempo)) = event.kind {
            now_tempo = tempo.as_int() as usize;
        }
    }
    let delta = tick - last_tick;
    last_tick = *tick;
    now_ms += midi_tick_to_ms(delta, now_tempo, now_ticks_per_beat as usize);
    absolute_tick_to_ms.insert(*tick, now_ms);
}

```

5.2.2 Performance

L'inférence d'amplitudes à partir des vélocités est assez coûteuse. La raison de ce coût n'a pas encore été étudiée.

5.2.3 Accessibilité

Malheureusement, là où l'export d'un projet musical en stems se résume à un simple clic dans un menu, l'export en MIDI est souvent plus complexe. Par exemple, sur FL Studio, il demande à créer *une copie du projet, avec toutes les pistes converties en « instruments MIDI »*, ce qui est fastidieux:

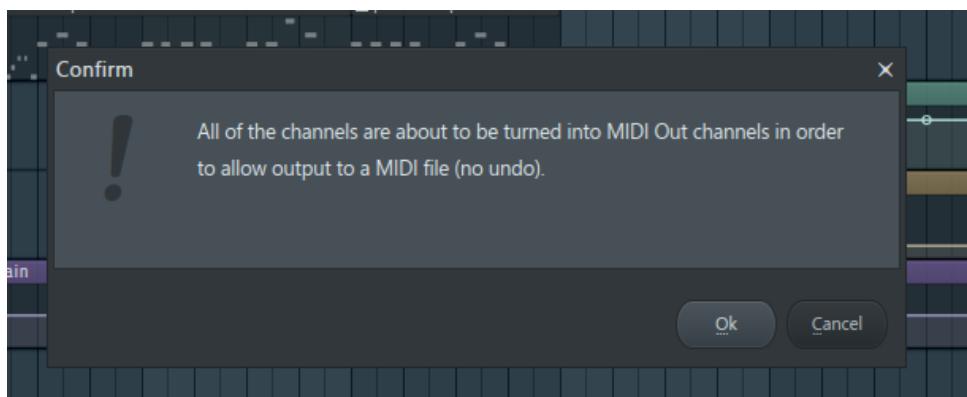


Fig. 18. – Dialogue d'avertissement lors de l'utilisation de la macro « Prepare for MIDI export » dans FL Studio

5.2.4 Conclusion

Cette méthode, malgré l'aspect fastidieux de sa mise en place, est une amélioration nette par rapport à l'approche par amplitude

Commit 7ae7a14a90f16f664edee3f433ade9b8c5019ffa

Figure out a POC to get notes from MIDI file into note[ms][stem_name]

And the conversion from MIDI ticks to milliseconds does not drift at all, after 6 mins on a real-world track (see research_midi/source.mid), it's still fucking _spot on_, to the FUCKING CENTISECOND (FL Studio can't show me more precision anyways).

So beautiful.

aight, imma go to sleep now

5.3 Fichier de projet

Étant donné l'aspect fastidieux de la solution précédente, il est intéressant de se pencher sur les fichiers de projet des logiciels de production musicale, afin de *remonter totalement à la source du morceau de musique*: le fichier qui est ouvert par l'artiste, celui sur lequel il travaille.

Malheureusement, les logiciel libres sont très loin derrière les standards de l'industrie en terme de production musicale, et il est aujourd'hui assez irréaliste de penser pouvoir produire de la musique avec des alternatives libres qui possède des formats de fichier de projet ouverts.

On doit donc se tourner vers de la rétro-ingénierie, et avoir une implémentation d'un « adaptateur » pour chaque logiciel de production musicale que l'on souhaite supporter.

5.3.1 FL Studio

Il existe une bibliothèque Python, pyflp [16], qui permet de parser les fichiers de projets FL Studio, et d'en extraire la quasi totalité des informations intéressantes.

```
import pyflp

def main():
    args = docopt(__doc__)
    project = pyflp.parse(args["<input_flp_file>"])

    out = {
        "info": {"name": project.title, "bpm": project.tempo},
        "arrangements": {},
    }

    for a in project.arrangements:
        out["arrangements"][a.name] = {
            "tracks": [
                track_name(track): serialize_track(track) for track in a.tracks
            ],
            "markers": [
                marker.position: marker.name for marker in a.timemarkers
            ],
        }

    Path(args["<output_json_file>"]).write_text(json.dumps(out, indent=4))
```

Cependant, l'auteur·ice de la bibliothèque n'a malheureusement plus le temps de la maintenir [17], et, étant donné l'évolution de FL Studio, le parser est voué à progressivement ne plus supporter les dernières versions du logiciel.

Étant donné que je suis utilisatrice de FL Studio, je n'a pas cherché de potentielles solutions pour d'autres logiciels de MAO.

Performance

Étant donné que l'adapter est en Python, l'intégrer proprement dans Shapemaker consisterai à éventuellement utiliser une solution de FFI¹¹ comme PyOxide [18], ce qui demanderait beaucoup de travail d'adaptation.

5.4 Dépôt de « sondes » dans le logiciel de MAO

Cette dernière solution, dont l'implémentation est encore en cours, consiste à donner la possibilité aux artistes d'exposer directement des signaux depuis leur logiciel, en les exfiltrant à Shapemaker à travers un VST¹² [19] dédié.

L'avantage de cette approche est qu'elle est agnostique au logiciel de MAO: en effet, VST est *le* standard de plugins audio, supporté par tous les logiciels.

C'est via cette technologie que les artistes peuvent jouer des instruments virtuels, allant des pianos physiquement simulés [20], en passant par vocaloïdes¹³ (comme par exemple Hatsune Miku [21]), aux synthétiseurs additifs, soustractifs ou à wavetables (dont un exemple très populaire est Serum [22]).

C'est aussi cette technologie qui est utilisée pour appliquer des effets aux signaux audio créés par les instruments (on parle de VST *effets*, contrairement aux VST *générateurs*), allant des modélisations de pédales d'effets de guitare ou de compresseurs analogiques à tube, aux simulations de compression digitale de signaux (« bitcrushing »), aux égaliseurs fréquentiels.



Fig. 19. – Un VST Shapemaker servant de sonde, dans une chaîne d'effets sur FL Studio

Il est donc possible de recevoir du signal, **autant audio que MIDI**, en entrée d'un VST.

Autre possibilité, qui s'avère utile parmi nos objectifs: les VSTs peuvent exposer à l'hôte (le logiciel de MAO) des paramètres changeables, ce qui permet de faire évoluer le timbre d'un instrument, l'intensité d'une réverbération, etc. Faire varier des paramètres au cours du temps est un aspect essentiel de la musique, en particulier électronique, qui contribue à « donner vie » à un morceau.

¹¹Foreign Function Interface, permettant d'appeler des fonctions écrites dans un autre langage de programmation

¹²Virtual Studio Technology, un standard de plugins audio

¹³simulateurs de parole chantée, cas à application musicale de la synthèse vocale

On peut donc également définir des paramètres sur notre VST-sonde, qui peuvent servir à, par exemple, automatiser des changements de couleurs en suivant une évolution dans le timbre d'un instrument, depuis la source directement (il suffit d'envoyer le signal d'automatisation au VST-sonde, en plus de l'instrument lui-même).

On exfiltre ensuite ces données hors du logiciel vers un « beacon », via un simple API WebSocket, qui permet une communication instantanée beaucoup plus performante que des requêtes HTTP, et est plus approprié à l'envoie de potentiellement plusieurs milliers de points de données par secondes: en effet, le VST-sonde s'immisçant dans la chaîne de traitement audio, il ne doit pas la ralentir considérablement, sous peine de rendre le logiciel de MAO inutilisable.

```
pub fn connect_to_beacon<T: FnMut(&ws::Sender) -> ()>(&mut action: T) -> Result<()> {
    ws::connect(beacon_url(), |out| {
        action(&out);
        move |_msg| out.close(ws::CloseCode::Normal)
    })?;
    Ok(())
}

pub fn register_probe(probe: Probe) -> Result<()> {
    connect_to_beacon(|beacon| {
        beacon
            .send(format!(
                "+ probe {}",
                serde_json::to_string(&probe).expect("Failed to serialize probe")
            ))
            .expect("Failed to send register probe message");
    })?;
    Ok(())
}
```

Liste 2. – Implémentation de la fonction permettant à une probe de se signaler auprès du beacon

Enfin, on utilise la crate *nih-plug* [23] pour exporter la partie VST de notre code en un fichier **.vst3**, chargeable dans un logiciel de MAO.

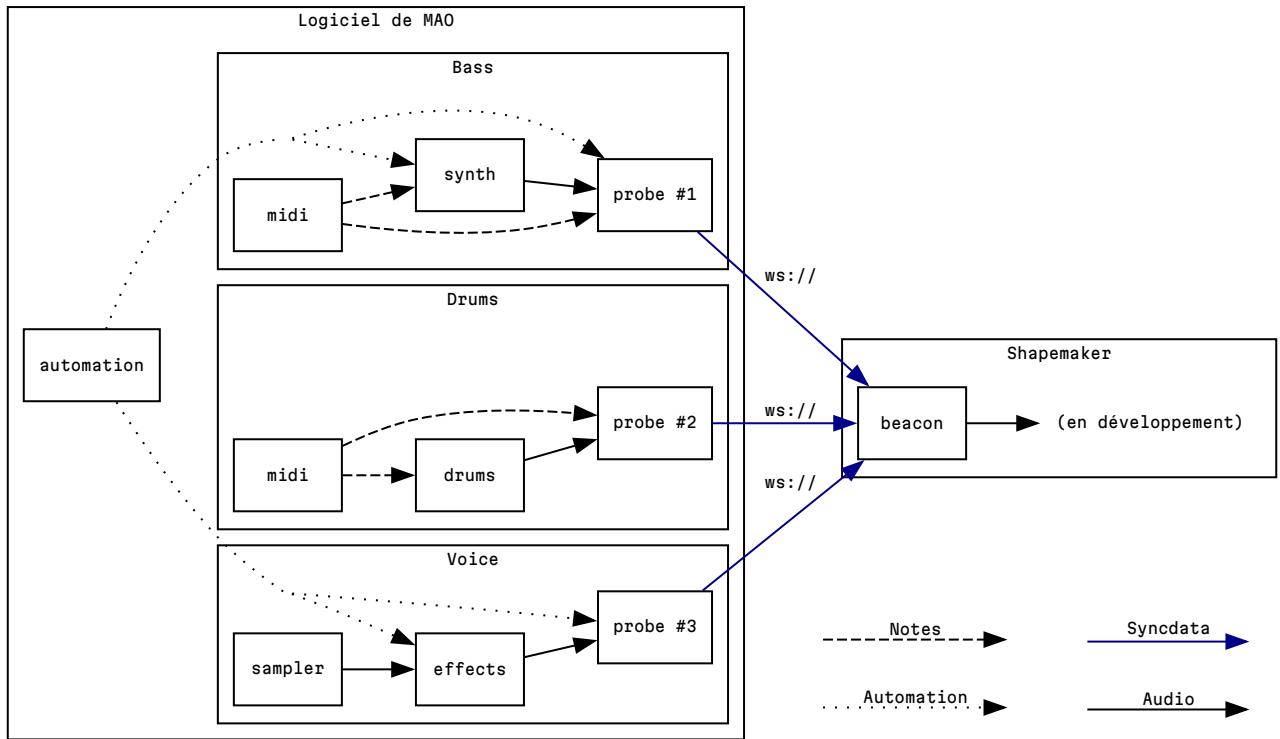


Fig. 20. – Exfiltration de données depuis la chaîne de traitement du logiciel de MAO

5.5 Temps réel: WASM et WebMIDI

Il est possible de réagir en temps réel à des pressions de touches sur des appareils conçus pour la production musicale assistée par ordinateur (MAO): des claviers, des potentiomètres pour ajuster des réglages affectant le timbre d'un son, des pads pour déclencher des sons et, par exemple, jouer des percussions, etc.

Ces appareils sont appelés « contrôleurs MIDI », du protocole standard qui régit leur communication avec l'ordinateur.

S'il est évidemment possible d'interagir avec ces contrôleurs depuis un programme natif (c'est après tout ce que font les logiciels de production musicale), j'ai préféré tenté l'approche Web, pour en faciliter l'accessibilité et en réduire le temps nécessaire à la mise en place¹⁴.

Comme pour de nombreuses autres technologies existant à la frontière entre le matériel et le logiciel, les navigateurs mettent à disposition des sites web une technologie permettant de communiquer avec les périphériques MIDI connectés à la machine: c'est l'API WebMIDI [24].

Mais bien évidemment, tout le code de Shapemaker, toutes ses capacités de génération de formes, sont implémentées en Rust.

Il existe cependant un moyen de « faire tourner du code Rust » dans un navigateur Web: la compilation vers WebAssembly (WASM), un langage assembleur pour le web [25], qui est une cible de compilation pour quelques des langages compilés plus modernes, comme Go [26] ou Rust [27].

¹⁴Imaginez, votre ordinateur a un problème 5 minutes avant le début d'une installation live, et vous aviez prévu d'utiliser Shapemaker pour des visuels. En faisant du dispositif un site web, il suffit de brancher son contrôleur à l'ordinateur d'un-e ami-e, et c'est tout bon.

En exportant la *crate* shapemaker en bibliothèque Javascript via wasm-bindgen [28], il est donc possible d'exposer à une balise `<script>` les fonctions de la bibliothèque, pour les brancher à un *callback* donné par l'API WebMIDI:

```
#[wasm_bindgen]
pub fn render_image(opacity: f32, color: Color) ->
Result<(), JsValue> {
    let mut canvas = /* ... */
    *WEB_CANVAS.lock().unwrap() = canvas;
    render_canvas_at(String::from("body"));

    Ok(())
}

import init, { render_image } from "./shapemaker.js"
void init()
navigator.requestMIDIAccess().then((midi) => {
    Array.from(midi.inputs).forEach((input) => {
        input[1].onmidimessage = (msg) => {
            const [cmd, ...args] = [...msg.data];
            if (cmd !== 144) return;

            const [pitch, velocity] = args;
            const octave = Math.floor(pitch / 12) - 1;

            render_image(velocity / 128, colors{octave});
        }
    })
})
}
```

Liste 3. – Exposition de fonctions à WASM depuis Rust, et utilisation de celles-ci dans un script Javascript

Au final, on peut arriver à une performance live interactive [29] intéressante, et assez réactive pour ne pas avoir de latence (et donc de désynchronisation audio/vidéo) perceptible.

Les navigateurs Web supportant nativement le format SVG, qui se décrit notamment comme directement inclurable dans le code HTML d'une page web [30], il est possible de simplement générer le code SVG, et de laisser le navigateur faire le rendu, ce qui s'avère être une solution très performante.

6 Performance

Les premiers prototypes de Shapemaker avaient une implémentation serielle, où le code Rust s'occupait seulement de la partie génération de formes et sérialisation en SVG. Chaque frame SVG était sauvegardée dans un fichier, puis converti en PNG en ligne de commande via ImageMagick [31]. Les frames étaient ensuite concaténées en une vidéo via FFmpeg, également en ligne de commande.

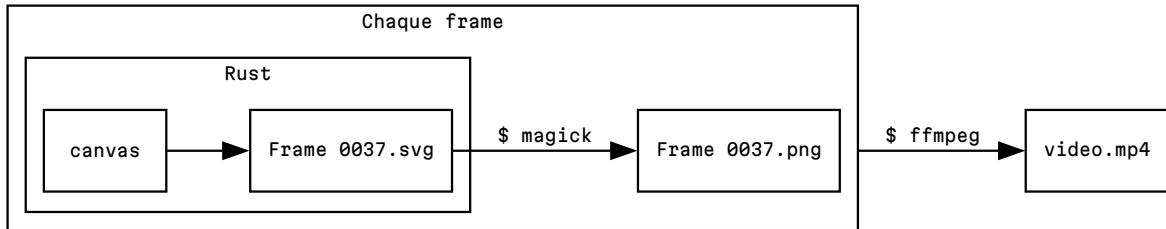


Fig. 21. – Pipeline de rendu, premier prototype

Un des plus gros gains de performance a été achevé en éliminant le plus d'I/O¹⁵ possible ainsi qu'un encodage/décodage PNG, en passant des pixmap (matrices de pixels) directement.

¹⁵Input/Output

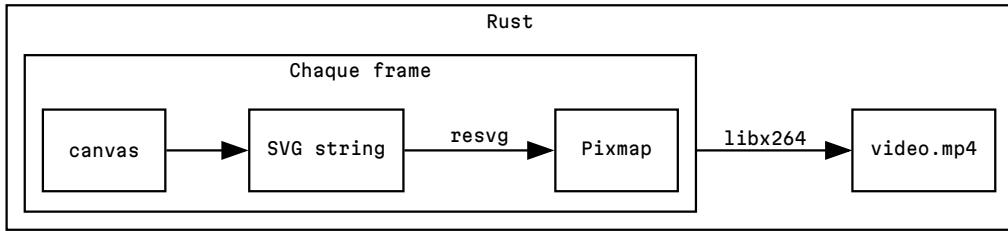


Fig. 22. – Pipeline de rendu sans *shell-out*¹⁶s

L'inconvénient est que, pour la partie encodage vidéo, il n'existe pas encore vraiment d'encodeur H.264¹⁷ en pur Rust, la plupart des solutions étant des bindings¹⁸ vers des bibliothèques C, notamment ffmpeg [32].

Cela rend l'installation de la bibliothèque beaucoup plus complexe, notamment sur Windows (les logiciels de production musicale sont très rares à fonctionner correctement sur Linux, surtout quand on prend en compte que les VSTs doivent eux aussi fonctionner sur Linux):

```

Compiling ffmpeg-sys-next v7.1.0
error: failed to run custom build command for `ffmpeg-sys-next v7.1.0`
note: To improve backtraces for build dependencies, set the
CARGO_PROFILE_DEV_BUILD_OVERRIDE_DEBUG=true environment variable to enable debug
information generation.

Caused by:
  process didn't exit successfully: `C:\Users\...
\projects.local\shapemaker\target\debug\build\ffmpeg-sys-next-
d2108b58b450b79e\build-script-build` (exit code: 101)
  --- stdout
  Could not find ffmpeg with vcpkg: Could not look up details of packages in
vcpkg tree could not read status file updates dir: The system cannot find the
path specified. (os error 3)

```

Liste 4. – Erreur rencontrée pendant la compilation des bindings Rust à libx264

Malgré plusieurs guides contradictoires d'installation, utiliser *vcpkg* [33] pour installer ffmpeg a fini par fonctionner

Une fois cette optimisation faite, qui a **divisé par 10** le temps de rendu, on peut se pencher sur le détail de la boucle de rendu pour identifier les potentiels gains de performance

¹⁶Invoquer un programme en ligne de commande (dans un shell), au lieu de faire tourner du code dans le programme courant

¹⁷Codec vidéo, très souvent utilisé pour les fichiers MP4, par exemple

¹⁸bibliothèque utilisant des FFIs pour donner un accès idiomatique à une bibliothèque provenant d'un autre langage de programmation

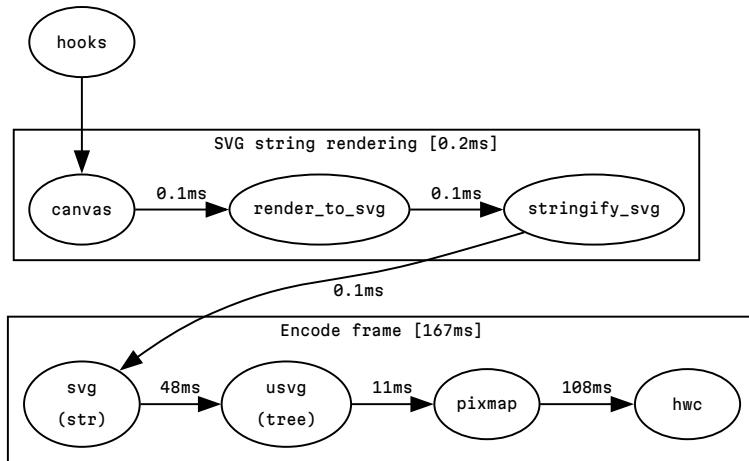


Fig. 23. – Détail de la boucle de rendu

Tâche	Δt	#
render_to_svg	0.127	150
stringify_svg	0.135	150
create_pixmap	0.251	150
setup_encoder	5.160	1
usvg_to_pixmap	10.823	150
svg_to_usvg	47.558	150
pixmap_to_hw	107.686	150
load_midi_notes	119.540	1
load_fonts	148.610	1
encode_frame	167.082	150

Tableau 1. – Durées d'exécution par tâche, pour une vidéo de test de 5 secondes (millisecondes)

6.1 Rastérisation parallèle

Si la partie `render_to_svg` n'est pas parallélisable car il faut bien faire exécuter tous les hooks dans l'ordre, la rastérisation des SVG sortants, elle, est bien parallélisable. Malheureusement, le gain de performance n'a pas été significatif, au contraire: rastériser toutes les frames, avant de commencer à encoder, implique de remplir la RAM avec des pixmaps – une par frame – ce qui conduit à une utilisation complète de toute la RAM de la machine, et bloque ainsi le système. Une approche avec une *queue* de taille maximale limitée, de laquelle l'encodeur pourrait récupérer les pixmaps rastérisées, reste à explorer.

6.2 Encodage H.264 parallèle?

Si l'on est bien capable de donner à l'encodeur nos frames dans le désordre, tout en lui indiquant le timestamp de chaque frame, l'encodeur doit recevoir les frames dans l'ordre [34]. Il est donc impossible de paralléliser l'encodage.

6.3 Pixmap et frames HWC: 100ms de standards

L'encodage vidéo étant fait par une bibliothèque totalement séparée de celle s'occupant de la rastérisation SVG, il y a un risque d'incompatibilité entre les formats de pixmap utilisés par les deux bibliothèques, ce qui est le cas ici.

En effet, les SVG rasterisés sont stockées dans un array plat de valeurs RGBA [35]:

```
[R, G, B, A, R, G, B, A, ...]
```

Tandis que la bibliothèque utilisée, *video-rs*, attend une matrice HWC, ou height-width-channels, de pixels RGB [36], [37], [38]:

```
[
  [ [R, G, B], [R, G, B], ... ],
  [ [R, G, B], [R, G, B], ... ],
  ...
]
```

Il est donc nécessaire de convertir entre ces deux formats, ce qui est lent car demande de copier les données.

La solution initiale utilisait `video_rs::Frame::from_shape_fn`:

```
Ok(video_rs::Frame::from_shape_fn(
    pixmap.height() as usize, pixmap.width() as usize, 3),
    |(y, x, c)| {
        let pixel = pixmap
            .pixel(x as u32, y as u32)
            .expect(&format!("No pixel found at x, y = {x}, {y}"));
        match c {
            0 => pixel.red(),
            1 => pixel.green(),
            2 => pixel.blue(),
            _ => unreachable!(),
        }
    },
))
```

Cependant, cette solution est très lente car *non parallélisée*, je l'ai donc réimplémentée avec de la parallélisation sur chaque pixel:

```
pub fn pixmap_to_hwc_frame(
    &self,
    size: (usize, usize),
    pixmap: &tiny_skia::Pixmap,
) -> anyhow::Result<video_rs::Frame> {
    debug_time!("pixmap_to_hwc_frame");
    let (width, height) = size;
    let mut data = vec![0u8; height * width * 3];

    data.par_chunks_exact_mut(3)
        .enumerate()
        .for_each(|(index, chunk)| {
            let x = index % width;
            let y = index / width;

            let pixel =
                pixmap.pixel(x as u32, y as u32).unwrap_or_else(|| {
                    panic!("No pixel found at x, y = {x}, {y}")
                });

            chunk[0] = pixel.red();
            chunk[1] = pixel.green();
            chunk[2] = pixel.blue();
        });
}

Ok(video_rs::Frame::from_shape_vec([height, width, 3], data)?)
```

On effectue toujours de la copie, mais la conversion est nettement plus rapide ainsi.

Bien évidemment, il ne faut pas faire d'erreur dans les calculs des coordonnées des pixels, ce qui peut donner des résultats surprenants, même si éventuellement artistiquement intéressants:

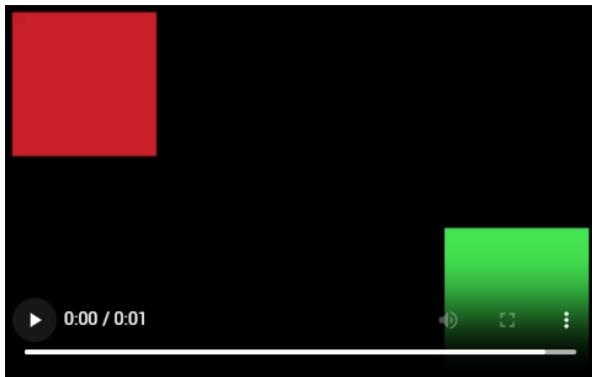


Fig. 24. – Frame cible correcte



Fig. 25. – Interversion de % et /

6.3.1 Aller plus loin

L'opération reste de loin la plus coûteuse de la chaîne de rendu.

Une solution serait de passer à une bibliothèque plus bas niveau et voir s'il est possible de donner directement les données de pixmap à l'encodeur, sans conversion, ou tout du moins sans avoir à copier les données.

Une autre solution est de proposer une contribution à la bibliothèque de rendu utilisée par *resvg*, *tiny_skia*¹⁹, pour y ajouter la possibilité d'instrumentaliser les lectures et écritures à sa pixmap, et ainsi stocker la représentation voulue par libx264 directement.

6.4 SVG vers string vers SVG

Comme on peut le remarquer, un gain de performance assez conséquent est possible si l'on parvient à utiliser usvg, non seulement pour la rastérisation, mais également pour la construction de l'arbre SVG: sur une boucle de rendu de 167 ms, **on passe 29% du temps à parser un arbre SVG sérialisé, alors que l'on vient de construire cette arbre.**

7 Conclusion

Malgré les multiples solutions de synchronisation audio-vidéo testées, avec certaines s'avérant infructueuses, l'approche par VST-sondes semble prometteuse, et permettrait de remplir presque tous les objectifs fixés au début du Chapitre 5.

L'approche WASM/WebMIDI explorée au Chapitre 5.5 est une solution appropriée pour des installations live, qui mérite d'être davantage explorée, notamment en vue de la création d'une solution de scripting pour VJing²⁰

7.1 Pistes d'améliorations

7.1.1 Feedback loop

Enfin, un des points les plus importants à améliorer reste la « feedback loop » *pendant la conception d'une procédure de génération*, qui reste extrêmement longue à cause de la lenteur de compilation de Rust, et du fait que, contrairement à un logiciel de montage vidéo, par exemple,

¹⁹Tiny-skia est notamment utilisé par Typst [39], [40], l'alternative moderne à LaTeX sur laquelle ce papier a été typeset

²⁰Visual Jockeying, l'art de mixer des visuels en live, souvent en concert ou en boîte de nuit

on ne peut que re-rendre la vidéo en MP4 (même si l'on peut décider de rendre qu'une petite partie), ouvrir le fichier, et regarder le résultat.

Une idée serait de, là aussi, utiliser le backend WASM/WebMIDI pour fournir une sorte de preview du code en temps réel: une interface simple permettrait de placer une tête de lecture à un instant pour y montrer la frame, et se rafraîchirait quand le code change. Avec éventuellement la possibilité de faire « play ».

Encore faut-il que la vitesse de recompilation de Rust le permette, même si ce serait à priori possible tant que la crate utilisant Shapemaker (celle que l'artiste écrit) reste légère.

7.1.2 Un langage de scripting

Rust étant un des langages de programmation les plus difficiles à utiliser, on pourrait éventuellement exposer l'API de Shapemaker à un langage de scripting plus léger, comme Lua par exemple, ce qui permettrait également de rendre le projet plus accessible.

Cela permettrait éventuellement aussi d'améliorer la vitesse de compilation de la crate écrite par l'artiste, qui pourrait, si elle est trop faible, empêcher l'implémentation de la solution de feedback loop telle qu'évoquée plus tôt. Des projets comme Tauri embarque un système de HMR²¹, non pas pour leur bibliothèque Rust, mais pour les bindings JavaScript exposé aux utilisateur·ice·s de la bibliothèque [41].

On pourrait même envisager afficher cette *preview* dans le logiciel de MAO, en tant qu'un 2e VST, « Shapemaker Preview ». Ceci demande d'implémenter encore un backend de rendu, autre que H.264 ou WASM, mais serait certainement la meilleure solution en terme d'UX²².

7.2 Code source

Le code source du projet est disponible en ligne sur Github:

[gwennlbh/shapemaker](https://github.com/gwennlbh/shapemaker)

Le répertoire `paper/` contient la source de ce papier, écrit en Typst

7.3 Exemples

Le projet n'étant pas encore terminé, il n'y a pas encore de clips musicaux publiés. Cependant, voici des liens vers quelques tests:

- https://youtu.be/3lx6VAz_UKM
- <https://instagram.com/p/C62JfogoUt9>

Remerciements

Bibliographie

- [1] Victor Vasarely, *MAJUS*. Alvéole 5, 1 avenue Marcel Pagnol, 13090 Aix-en-Provence, France: Fondation Vasarely, 1964.

²¹Hot Module Replacement, permettant de recharger du code en temps réel sans recharger la page, technologie assez prévalente dans le développement web frontend

²²expérience utilisateur·ice

- [2] Fondation Vasarely, « Planetary Folklore Period ». Consulté le: 22 mars 2025. [En ligne]. Disponible sur: <https://www.fondationvasarely.org/en/planetary-folklore-period/>
- [3] « Relais Copies ». Google Maps, 30 Rue Pharaon, 31000 Toulouse.
- [4] Invader, *Invader - Paris*. Consulté le: 23 mars 2025. [En ligne]. Disponible sur: <https://www.space-invaders.com/world/paris/>
- [5] The European Space Agency, « Space Invaders », 14 mars 2015. Consulté le: 23 mars 2025. [En ligne]. Disponible sur: https://www.esa.int/Space_in_Member_States/France/Highlights/Space_Invaders
- [6] « Visionneuse photo et conversion d'images par lot | XnView ». Consulté le: 23 mars 2025. [En ligne]. Disponible sur: <https://www.xnview.com/fr/>
- [7] Steve Klabnik, Carol Nichols, et Chris Krycho, *The Rust Programming Language § 7.1 Packages and Crates*. Consulté le: 23 mars 2025. [En ligne]. Disponible sur: <https://doc.rust-lang.org/book/ch07-01-packages-and-crates.html>
- [8] Brian Anderson, « Rust's Huge Compilation Units », 22 juin 2020, *TiDB*. Consulté le: 24 mars 2025. [En ligne]. Disponible sur: <https://www.pingcap.com/blog/rust-huge-compilation-units/>
- [9] Steve Klabnik, Carol Nichols, et Chris Krycho, *The Rust Programming Language § 10.2 Traits: Defining Shared Behavior*. Consulté le: 23 mars 2025. [En ligne]. Disponible sur: <https://doc.rust-lang.org/book/ch10-02-traits.html>
- [10] Yevhenii Reizner, *linebender/resvg: An SVG rendering library*. Linebender. Consulté le: 23 mars 2025. [En ligne]. Disponible sur: <https://github.com/linebender/resvg>
- [11] Jorge Israel Peña, « What are hooks? ». Consulté le: 24 mars 2025. [En ligne]. Disponible sur: <https://softwareengineering.stackexchange.com/a/13748>
- [12] LabelRadar et Armada Music, « Will Clarke & Armada Music Launch 'Midnight Mass' Remix Contest ». Consulté le: 24 mars 2025. [En ligne]. Disponible sur: <https://www.beatportal.com/articles/866279-will-clarke-armada-music-launch-midnight-mass-remix-contest>
- [13] Daniel Stoller, Sebastian Ewert, et Simon Dixon, « Wave-U-Net: A Multi-Scale Neural Network for End-to-End Audio Source Separation », 14 septembre 2018, *19th International Society for Music Information Retrieval Conference (ISMIR 2018)*. doi: [10.48550/arXiv.1806.03185](https://doi.org/10.48550/arXiv.1806.03185).
- [14] MIDI Association, « Summary of MIDI 1.0 Messages ». Consulté le: 24 mars 2025. [En ligne]. Disponible sur: <https://midi.org/summary-of-midi-1-0-messages>
- [15] Craig Anderton, *MIDI for Musicians*. Amoco Publications, 1986, p. 8-10. Consulté le: 24 mars 2025. [En ligne]. Disponible sur: https://archive.org/details/midiformusicians_0000ande
- [16] demberto, *pyflp 2.2.1*. (5 juin 2023). PyPI. Consulté le: 24 mars 2025. [En ligne]. Disponible sur: <https://pypi.org/project/pyflp/>
- [17] kingkongjames, « 🐍 Python 3.12 Enum issue », 7 novembre 2023, *GitHub*. Consulté le: 24 mars 2025. [En ligne]. Disponible sur: <https://github.com/demberto/PyFLP/issues/183>

- [18] David Hewitt *et al.*, *PyO3*. GitHub. Consulté le: 24 mars 2025. [En ligne]. Disponible sur: <https://github.com/PyO3/pyo3>
- [19] Steinberg, « VST 3 Interfaces Documentation ». Consulté le: 24 mars 2025. [En ligne]. Disponible sur: https://steinbergmedia.github.io/vst3_doc/vstinterfaces/index.html
- [20] Modartt, « Pianoteq Overview ». Consulté le: 24 mars 2025. [En ligne]. Disponible sur: https://www.modartt.com/pianoteq_overview
- [21] Crypton Future Media, « About HATSUNE MIKU ». Consulté le: 24 mars 2025. [En ligne]. Disponible sur: https://ec.crypton.co.jp/pages/prod/virtualsinger/cv01_us
- [22] Xfer Records, « Serum: Advanced Wavetable Synthesizer ». Consulté le: 24 mars 2025. [En ligne]. Disponible sur: <https://xferrecords.com/products/serum>
- [23] Robbert van der Helm, « nih_plug ». Consulté le: 24 mars 2025. [En ligne]. Disponible sur: https://nih-plug.robbertvanderhelm.nl/nih_plug/
- [24] MDN Authors, « Web MIDI API ». Consulté le: 22 mars 2025. [En ligne]. Disponible sur: https://developer.mozilla.org/en-US/docs/Web/API/Web_MIDI_API
- [25] WebAssembly Community Group, « WebAssembly ». Consulté le: 22 mars 2025. [En ligne]. Disponible sur: <https://webassembly.org/>
- [26] Go language authors, « Go Wiki: WebAssembly ». Consulté le: 22 mars 2025. [En ligne]. Disponible sur: <https://go.dev/wiki/WebAssembly>
- [27] Rust authors, « WebAssembly — Le langage de programmation Rust ». Consulté le: 22 mars 2025. [En ligne]. Disponible sur: <https://www.rust-lang.org/fr/what/wasm>
- [28] Rust and WebAssembly Working Group, *The `wasm-bindgen` Guide*. Consulté le: 22 mars 2025. [En ligne]. Disponible sur: <https://rustwasm.github.io/wasm-bindgen/introduction.html>
- [29] Gwenn Le Bihan, *Démo de piano avec shapemaker*. Consulté le: 1 mai 2024. [En ligne Vidéo]. Disponible sur: <https://www.instagram.com/p/C6byymcou5k/>
- [30] Amelia Bellamy-Royds, Chris Lilley, Tavmjong Bah, Dirk Schulze, et Eric Willigers, « Scalable Vector Graphics (SVG) 2 § 1.1 About SVG ». W3C Editor's Draft. Consulté le: 8 mars 2023. [En ligne]. Disponible sur: <https://svgwg.org/svg2-draft/intro.html#AboutSVG>
- [31] « ImageMagick – Mastering Digital Image Alchemy ». Consulté le: 26 mars 2025. [En ligne]. Disponible sur: <https://imagemagick.org/index.php>
- [32] « FFmpeg ». Consulté le: 26 mars 2025. [En ligne]. Disponible sur: <https://ffmpeg.org/>
- [33] Microsoft, « vcpkg - Open source C/C++ dependency manager ». Consulté le: 24 mars 2025. [En ligne]. Disponible sur: <https://vcpkg.io/en/>
- [34] popo2 popo2, « Dynamic pipeline, "non-strictly-monotonic PTS" warning with x264enc after switching source », 26 novembre 2012, *Narkive Mailing List Archive, gstreamer-devel@list.freedesktop.org*. Consulté le: 26 mars 2025. [En ligne]. Disponible sur: <https://gstreamer-devel.narkive.com/LJuu7yM3/dynamic-pipeline-non-strictly-monotonic-pts-warning-with-x264enc-after-switching-source>

- [35] Yevhenii Reizner et Bruce Mitchener, *src/tiny_skia/pixmap.rs*, lignes 30 à 33. Linebender. Consulté le: 24 mars 2025. [En ligne]. Disponible sur: https://docs.rs/tiny-skia/0.11.4/src/tiny_skia/pixmap.rs.html#30-33
- [36] Gerwin van der Lugt, « Encoder in video_rs::encode, 0.10.3 ». Oddity AI. Consulté le: 24 mars 2025. [En ligne]. Disponible sur: https://docs.rs/video-rs/0.10.3/video_rs/encode/struct.Encoder.html#method.encode
- [37] Gerwin van der Lugt, « Frame in video_rs::frame, 0.10.3 ». Oddity AI. Consulté le: 24 mars 2025. [En ligne]. Disponible sur: https://docs.rs/video-rs/0.10.3/video_rs/frame/struct.Frame.html
- [38] Ulrik «bluss» Sverdrup et Jim Turner, « Array3 in ndarray 0.16.1 ». rust-ndarray. Consulté le: 24 mars 2025. [En ligne]. Disponible sur: <https://docs.rs/ndarray/0.16.1/ndarray/type.Array3.html>
- [39] « Dependencies · typst/typst ». Consulté le: 24 mars 2025. [En ligne]. Disponible sur: <https://github.com/typst/typst/network/dependencies?q=tiny-skia>
- [40] *Cargo.toml at main · typst/typst*. Typst. Consulté le: 24 mars 2025. [En ligne]. Disponible sur: <https://github.com/typst/typst/blob/1b2714e1a758d6ee0f9471fd1e49cb02f6d8cde4/Cargo.toml#L124>
- [41] « Tauri 2.0 Stable Release », 2 octobre 2024, *Tauri*. Consulté le: 24 mars 2025. [En ligne]. Disponible sur: <https://v2.tauri.app/blog/tauri-20/#hot-module-replacement>
- [42] Postamble, *Onset (Album Version)*. 2024. Consulté le: 24 mars 2025. [En ligne]. Disponible sur: https://gwen.works/emoti*ns

Annexes

A Marqueurs dans un logiciel de MAO



Fig. 26. – Marqueurs dans FL Studio: INTRO END, BLOCK 1, BREAK 1, BUILDUP 1, ...
Fichier de projet pour *Onset* de Postamble [42]

B Série « interprétation collective » 1

