
Shapemaker: Créations audiovisuelles procédurales musicalement synchrones

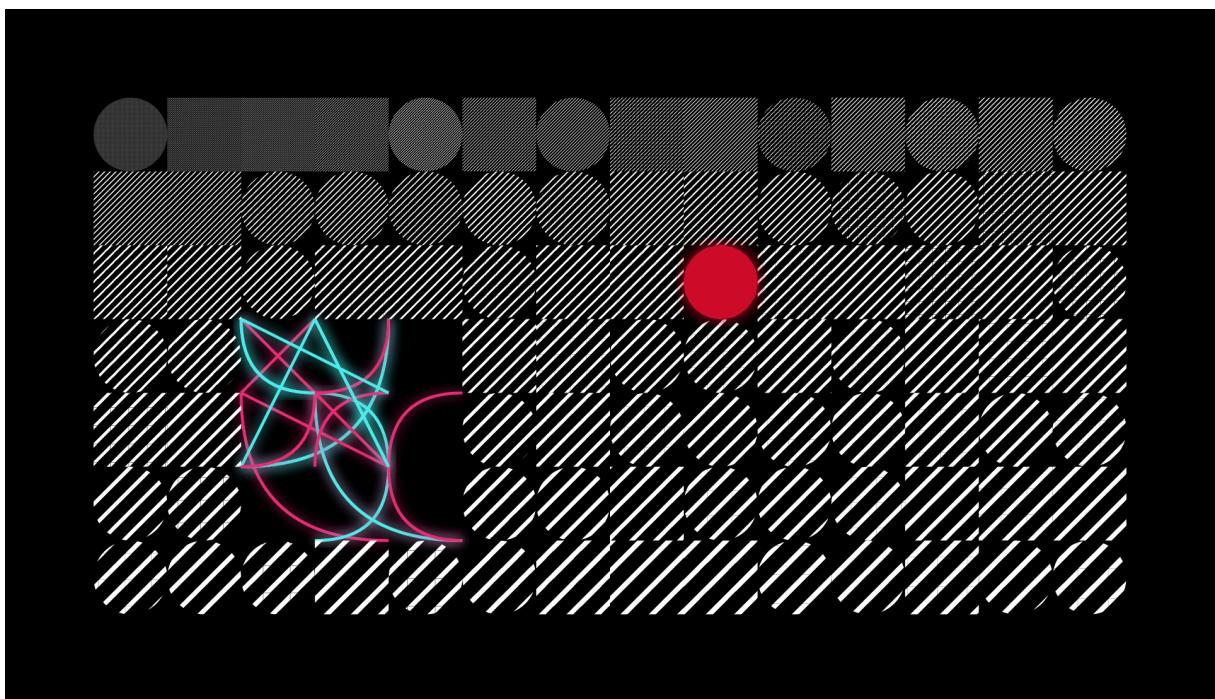
Gwenn Le Bihan

gwenn.lebihan@etu.inp-n7.fr

ENSEEIHT

23 Mars 2025

Mots clés audiovisuel · procédural · SVG · Rust · WASM · WebMIDI · VST



```
use shapemaker::*;

pub fn dna_analysis_machine() -> Canvas {
    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: "#4fec6c".into(),
    });
}
```

```

canvas.set_grid_size(16, 9);
canvas.set_background(Color::Black);

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

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

let red_circle_at =
    Region::from_topright(draw_in.topright()).translated(-3, 0), (4, 3))
    .unwrap()
    .random_point();

let mut hatches_layer = Layer::new("hatches");
let mut red_dot_layer = Layer::new("red dot");

for (i, point) in draw_in.iter().enumerate() {
    if filaments_area.contains(&point) {
        continue;
    }

    if point == red_circle_at {
        red_dot_layer.add_object(
            format!("red circle @ {}", point),
            Object::BigCircle(point)
                .color(Color::Red)
                .filter(Filter::glow(5.0)),
        );
    }

    hatches_layer.add_object(
        point,
        if rand::thread_rng().gen_bool(0.5) || point == red_circle_at {
            Object::BigCircle(point)
        } else {
            Object::Rectangle(point, point)
        }
        .paint(Fill::Hatched(
            Color::White,
            Angle(45.0),
            (i + 5) as f32 / 10.0,
            0.25,
        )),
    );
}

let mut filaments =
    canvas.n_random_curves_within(&filaments_area, 30, "splines");

for (i, object) in filaments.objects.values_mut().enumerate() {
    object.recolor(if i % 2 == 0 { Color::Cyan } else { Color::Pink });
}

filaments.filter_all_objects(Filter::glow(4.0));

canvas.layers.push(red_dot_layer);
canvas.layers.push(hatches_layer);
canvas.layers.push(filaments);
canvas
}

```

Table des matières

| | | |
|-------|--|----|
| 1 | Introduction | 3 |
| 1.1 | À la recherche d'une impossible énumération des formes | 3 |
| 1.2 | Une approche procédurale ? | 5 |
| 1.3 | Excursion dans le monde physique | 6 |
| 1.3.1 | Interprétation collective | 7 |
| 1.4 | Lien musical | 9 |
| 2 | Une <i>crate</i> Rust avec un API sympathique | 10 |
| 3 | Render loop et hooks | 11 |
| 4 | Sources de synchronisation | 11 |
| 4.1 | Temps réel: WASM et WebMIDI | 11 |
| 4.2 | Amplitudes de <i>stems</i> | 12 |
| 4.3 | Export MIDI | 12 |
| 4.4 | Fichier de projet | 13 |
| 4.5 | Dépôt de « sondes » dans le logiciel de MAO | 14 |
| 5 | Performance | 14 |
| 6 | Conclusion | 14 |
| | Bibliographie | 14 |

1 Introduction

1.1 À la recherche d'une impossible énumération des formes

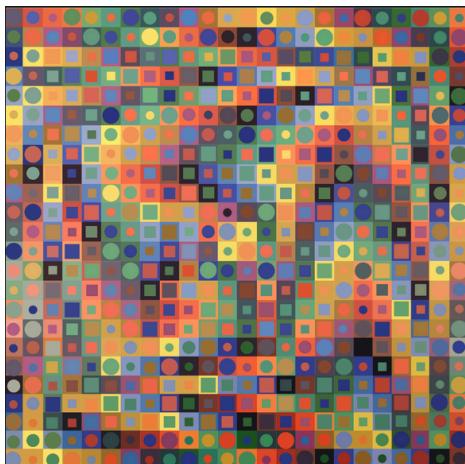


Fig. 1. – MAJUS [1]

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 employant des séries combinaisons simples de formes et couleurs. D'apparence très simple, ces combinaisons sont d'une manières assez fascinantes 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.

Avec cette idée dans la tête, je me mets à gribouiller une ébauche d'« alphabet des formes », qui, naïvement, chercher à énumérer toutes les formes construisibles à partir de formes simples, que l'on peut superposer, pivoter et translater.

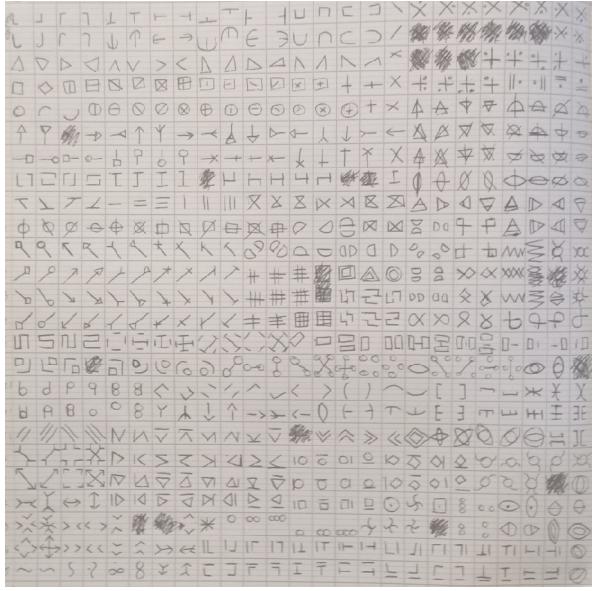


Fig. 2. – Un “alphabet” incomplet

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 est que, parmi les différents formats d'image vectorielles existant, le *SVG*, pour *Scalable Vector Graphics*, est indéniablement le plus populaire, et est un standard ouvert décrivant un format texte.

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

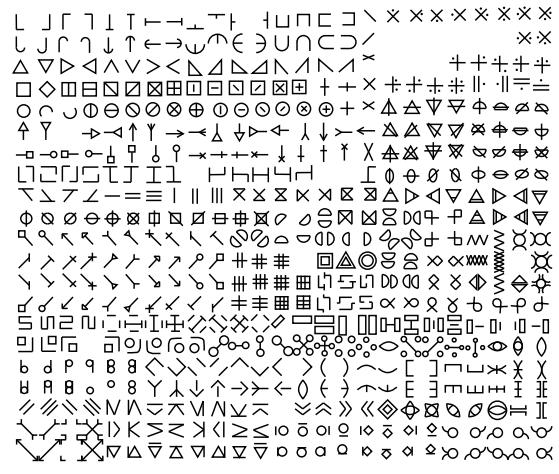


Fig. 3. – Une vectorisation

1.2 Une approche procédurale ?

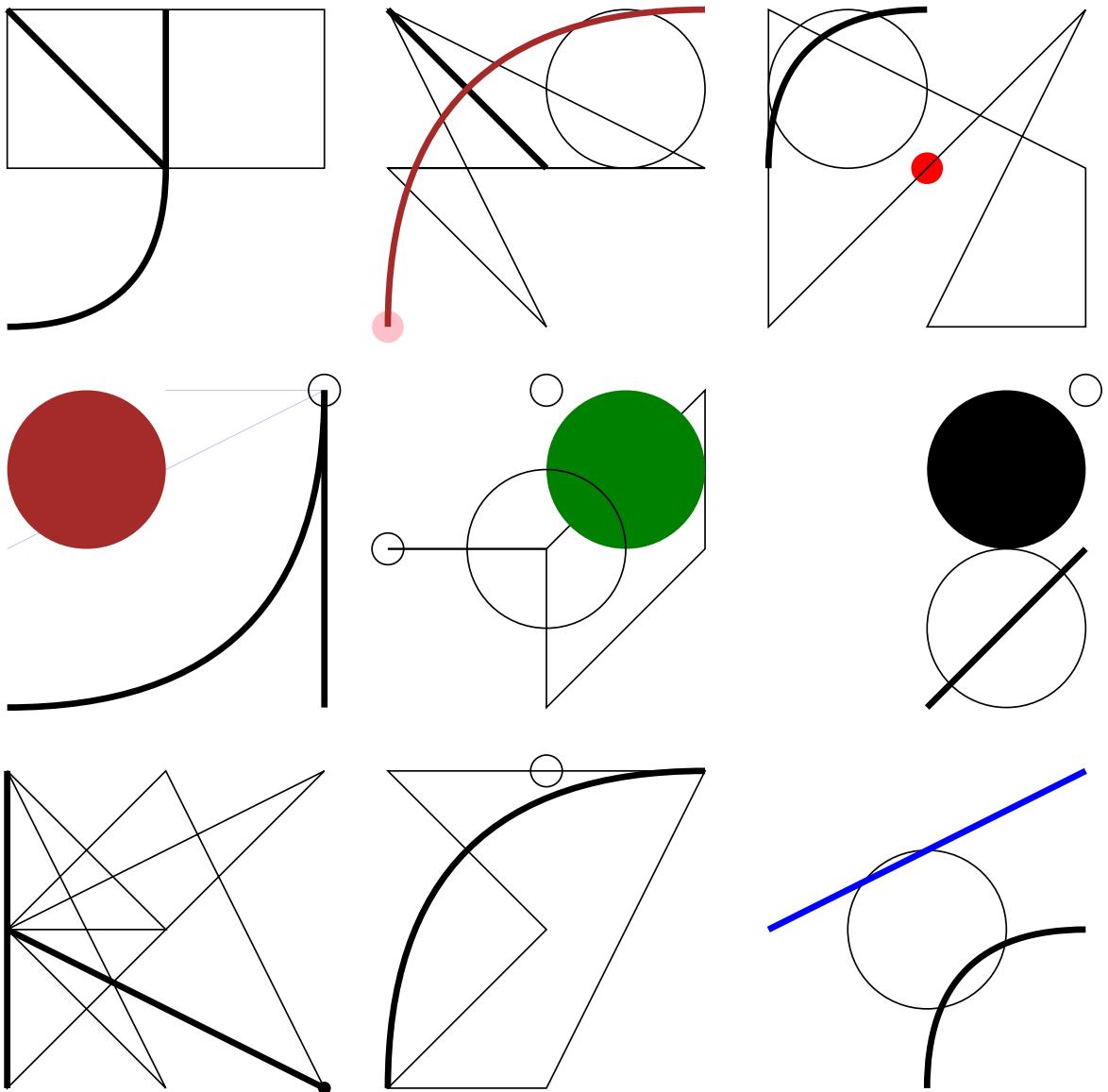


Fig. 4. – Exemples d'œuvres résultant d'une procédure de génération semi-aléatoire, basée sur une grille de 8 “points d'ancrages”

L'étape prochaine dans cette démarche était évidemment 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 préfèrera 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 de d'autres, comme la palette de couleurs, sont des décisions de l'artiste.

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

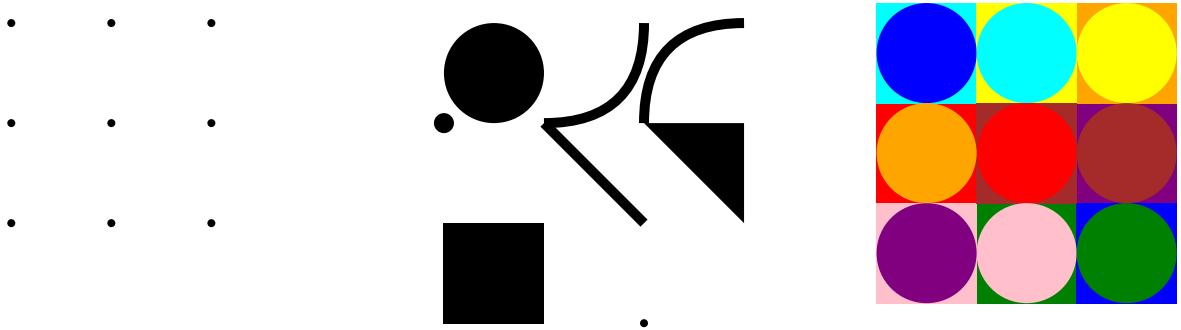


Fig. 5. – Vocabulaire visuel des premières ébauches: grille de placement à 9 points, formes et couleurs

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 devisé 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.

Étant donné la simplicité avec laquelle l'on peut générer de grandes quantités d'œuvres à partir d'un même langage, le *choix d'en sélectionner les meilleures* 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 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 le 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]).



Fig. 7. – « Paramount »



Fig. 8. – « Reflets Citadins », nommée par Enide



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

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

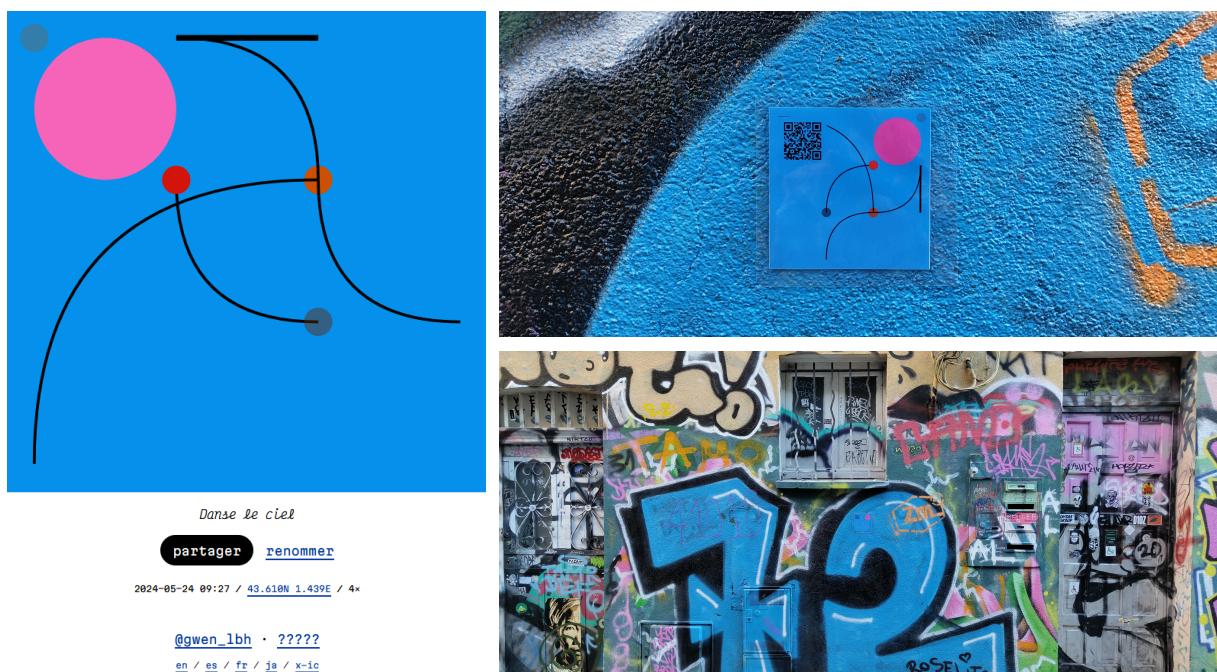


Fig. 10. – « Danse le ciel »

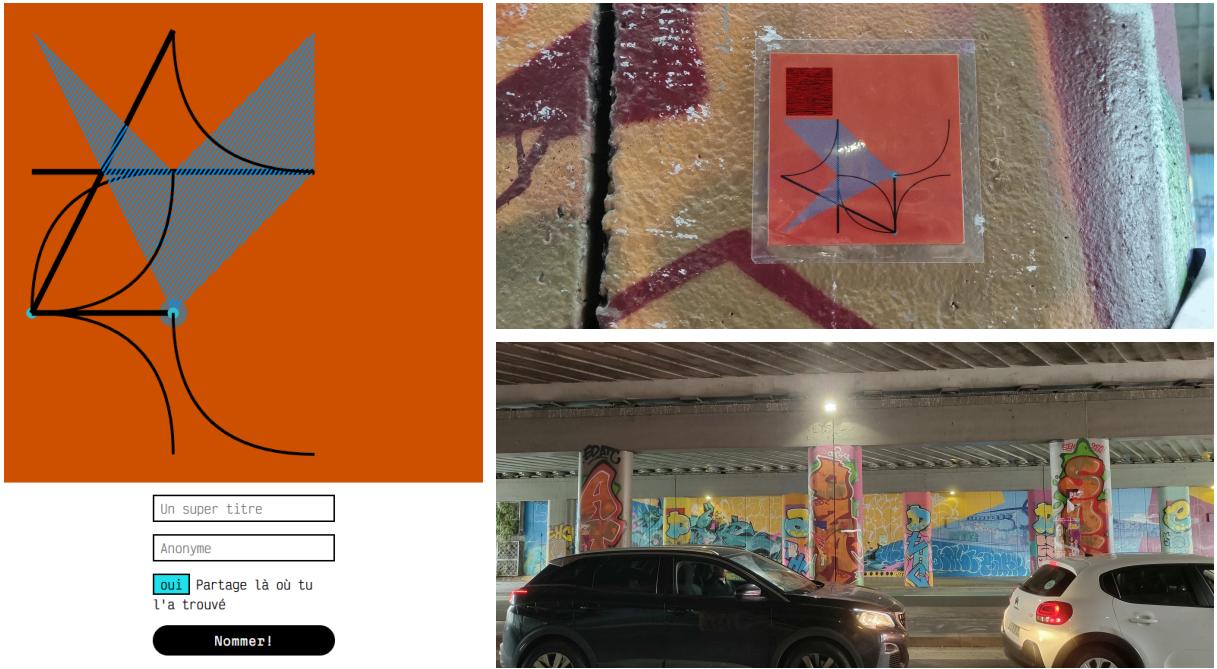


Fig. 11. – *Sans titre*

1.4 Lien musical

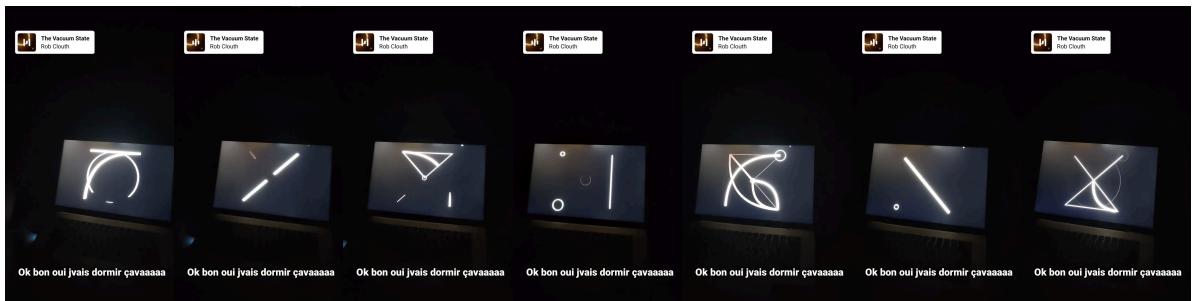


Fig. 12. – Frames d'une *story* Instagram montrant une première esquisse de vidéo

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 permet de se re-charger 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 *réagit à des moments et rythmes clés du morceau*.

2 Une *crate* Rust avec un API sympathique

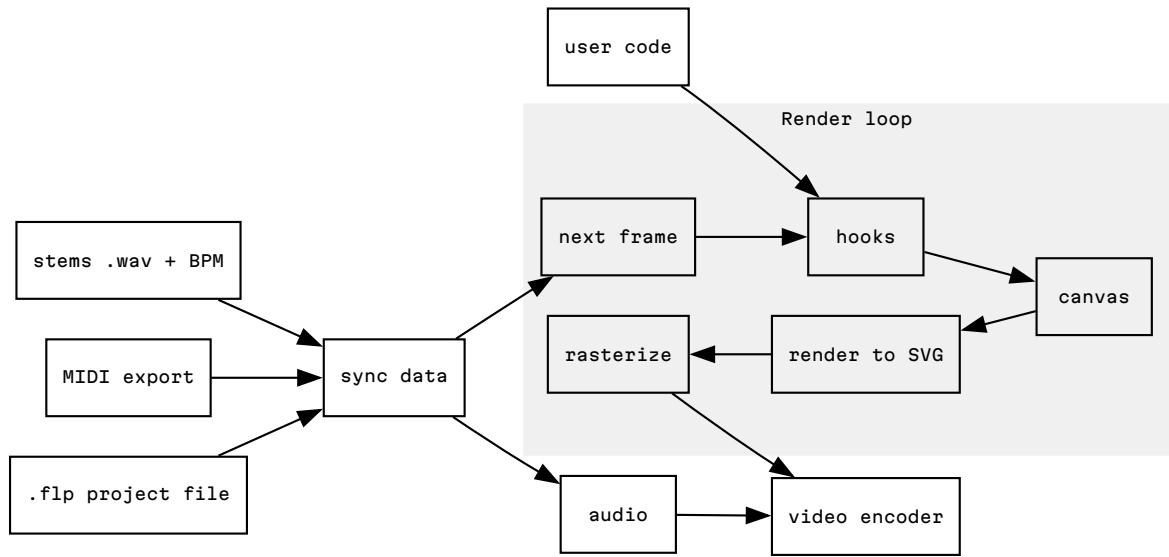


Fig. 13. – Pipeline

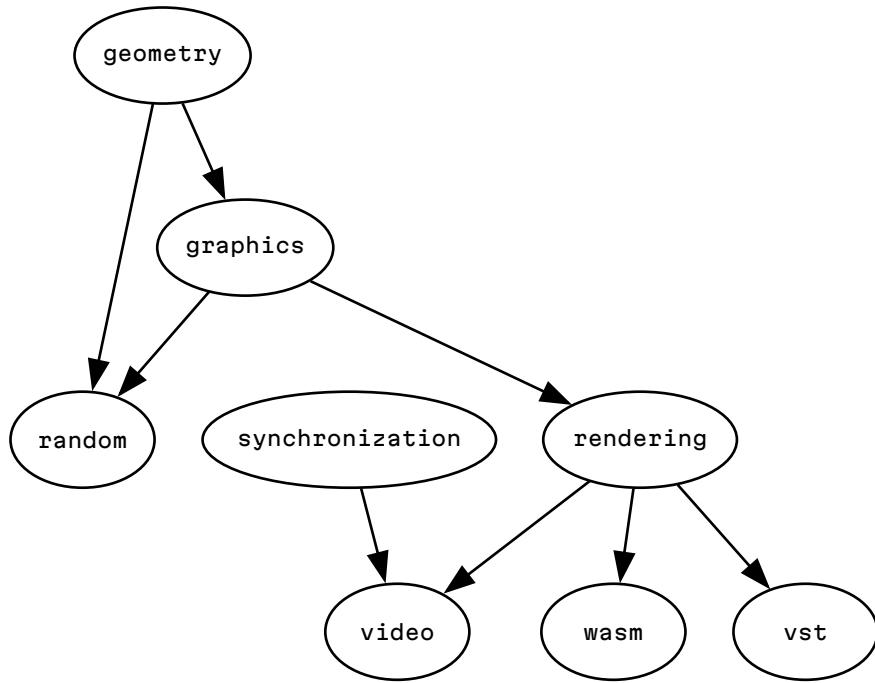


Fig. 14. – Organisation des sous-modules

3 Render loop et hooks

4 Sources de synchronisation

4.1 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 place¹.

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 [7].

Mais bien évidemment, tout le code de Shapemaker, tout 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 [8], qui est une cible de compilation pour quelques des langages compilés plus modernes, comme Go [9] ou Rust [10]

En exportant la *crate* shapemaker en bibliothèque Javascript via wasm-bindgen [11], il est donc possible d'exposer à une balise `<script>` les fonctions de la bibliothèque, et brancher donc celles-ci à des *callbacks* donnés 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 1. – Exposition de fonctions à WASM depuis Rust, et utilisation de celles-ci dans un script Javascript

¹Imaginez, 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.

Au final, on peut arriver à une performance live interactive [12] 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 incluable directement dans le code HTML d'une page web [13], 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.

4.2 Amplitudes de *stems*

```
let mut reader = hound::WavReader::open(path.clone())
    .map_err(|e| format!("Failed to read stem file: {}", e))
    .unwrap();

let spec = reader.spec();

let sample_index_to_frame = |sample: usize| {
    (sample / spec.channels / spec.sample_rate * self.fps) as usize
};

let mut amplitude_db: Vec<f32> = vec![];
let mut current_amplitude_sum: f32 = 0.0;
let mut current_amplitude_buffer_size: usize = 0;
let mut latest_loaded_frame = 0;

for (i, sample) in reader.samples::<i16>().enumerate() {
    let sample = sample.unwrap();
    if sample_index_to_frame(i) > latest_loaded_frame {
        amplitude_db
            .push(current_amplitude_sum / current_amplitude_buffer_size as f32);
        current_amplitude_sum = 0.0;
        current_amplitude_buffer_size = 0;
        latest_loaded_frame = sample_index_to_frame(i);
    } else {
        current_amplitude_sum += sample.abs() as f32;
        current_amplitude_buffer_size += 1;
    }
}

let stem = Stem {
    amplitude_max: *amplitude_db.iter().max().unwrap(),
    amplitude_db,
    duration_ms: (reader.duration() / spec.sample_rate * 1000.0) as usize,
};

// Write loaded stem to a CBOR cache file
Stem::save_to_cbor(&stem, &cached_stem_path);
```

4.3 Export MIDI

```
// Add notes
let mut stem_notes = StemNotes::new();
for (tick, tracks) in timeline.iter().sorted_by_key(|(tick, _)| *tick) {
    for (track_name, event) in tracks {
        if let TrackEventKind::Midi {
            channel: _,
            message,
        } = event.kind
        {
            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()
                },
            },
        );
    }
    _ => {}
}
progressbar.inc(1)
}
}

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

4.4 Fichier de projet

```

def main():
    args = docopt(__doc__)

    project = pyflp.parse(args["<input_flp_file>"])

    out = {
        "info": {
            "name": project.title,
            "bpm": project.tempo,
        },
        "arrangements": {},
    }
    for arrangement in project.arrangements:
        current_arrangement = {"tracks": {}, "markers": {}}
        for track in arrangement.tracks:
            current_track = {}
            for clip in track:
                current_track[clip.position] = {
                    "length": clip.length,
                    "name": clip_name(clip),
                    "data": clip_data(clip),
                }
    }

```

```

        current_arrangement["tracks"][track_name(track)] = current_track
    for marker in arrangement.timemarkers:
        current_arrangement["markers"][marker.position] = marker.name
    out["arrangements"][arrangement.name] = current_arrangement

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

# end

```

4.5 Dépôt de « sondes » dans le logiciel de MAO

```

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(())
}

```

5 Performance

6 Conclusion

Bibliographie

- [1] Victor Vasarely, *MAJUS*. Alvéole 5, 1 avenue Marcel Pagnol, 13090 Aix-en-Provence, France: Fondation Vasarely, 1964.
- [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] 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
- [8] WebAssembly Community Group, « WebAssembly ». Consulté le: 22 mars 2025. [En ligne]. Disponible sur: <https://webassembly.org/>
- [9] Go language authors, « Go Wiki: WebAssembly ». Consulté le: 22 mars 2025. [En ligne]. Disponible sur: <https://go.dev/wiki/WebAssembly>
- [10] 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>
- [11] 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>
- [12] Gwenn Le Bihan, *Démo de piano avec shapemaker*. Consulté le: 1 mai 2024. [En ligne Vidéo]. Disponible sur: <https://www.instagram.com/p/C6byymcou5k/>
- [13] 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>