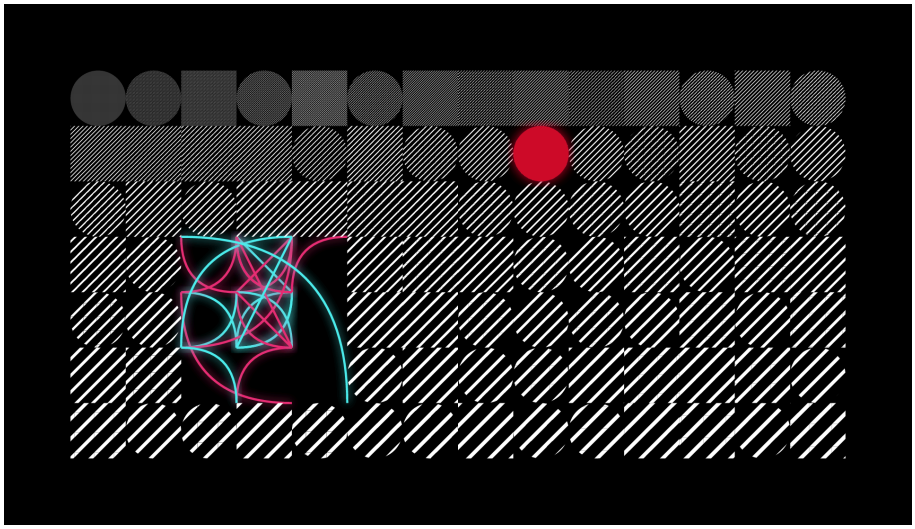
Shapemaker: Créations audiovisuelles procédurales musicalement synchrones

Gwenn Le Bihan
gwenn.lebihan@etu.inp-n7.fr
ENSEEIHT

22 Mars 2025

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



```rust
use std::iter;

use shapemaker::*;
use rand::Rng;

pub fn shapes_shed() -> Canvas {
    let mut canvas = Canvas::new(vec![]);

    canvas.set_grid_size(3, 3);
    canvas.set_background(Color::White);

    let root = canvas.layer("root");

    root.add_object(
        "1",
        Object::BigCircle(Point(0, 0)).color(Fill::Solid(Color::Black)),
    );
    root.add_object(
        "2",
        Object::CurveOutward(Point(1, 1), Point(2, 0), 5.0)
            .color(Fill::Solid(Color::Black)),
    );
    root.add_object(
```

```
        "3",
        Object::CurveInward(Point(2, 1), Point(3, 0), 5.0)
            .color(Fill::Solid(Color::Black)),
    );
    root.add_object(
        "4",
        Object::SmallCircle(Point(0, 1)).color(Fill::Solid(Color::Black)),
    );
    root.add_object(
        "5",
        Object::Line(Point(1, 1), Point(2, 2), 5.0)
            .color(Fill::Solid(Color::Black)),
    );
    root.add_object(
        "6",
        Object::Polygon(
            Point(2, 1),
            vec![
                LineSegment::Straight(Point(3, 1)),
                LineSegment::Straight(Point(3, 2)),
            ],
        )
        .color(Fill::Solid(Color::Black)),
    );
    root.add_object(
        "7",
        Object::Rectangle(Point(0, 2), Point(0, 2))
            .color(Fill::Solid(Color::Black)),
    );
    root.add_object(
        "8",
        Object::Dot(Point(2, 3)).color(Fill::Solid(Color::Black)),
    );

    canvas
}

pub fn colors_shed() -> Canvas {
    let mut canvas = Canvas::new(vec!["circles"]);
    canvas.set_grid_size(4, 3);
    canvas.canvas_outter_padding = 0;

    let all_colors = vec![
        Color::Gray,
        Color::Black,
        Color::Blue,
        Color::Cyan,
        Color::White,
        Color::Yellow,
        Color::Orange,
        Color::Red,
        Color::Brown,
        Color::Purple,
        Color::Pink,
        Color::Green,
    ];

    let foregrounds = all_colors.iter();
    let backgrounds = all_colors.iter().cycle().skip(1);
    let colors = iter::zip(foregrounds, backgrounds);

    for ((color, bgcolor), point) in
        iter::zip(colors, canvas.world_region.iter())
```

```
    {
        println!("{}: {:?} {:?}", point, color, bgcolor);
        canvas.layer("circles").add_object(
            color.name(),
            Object::BigCircle(point).color(Fill::Solid(*color)),
        );
        canvas.layer("root").add_object(
            format!("{}_bg", color.name()),
            Object::Rectangle(point, point).color(Fill::Solid(*bgcolor)),
        );
    }

    canvas
}

pub fn dna_analysis_machine() -> Canvas {
    let mut canvas = Canvas::new(vec![]);

    canvas.colormap = 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: "#4fecec".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(Fill::Solid(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)
                }
                .color(Fill::Hatched(
                    Color::White,
                    Angle(45.0),
                    (i + 5) as f32 / 10.0,
                    0.25,
                )),
            );
        }

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

        for (i, object) in filaments.objects.values_mut().enumerate() {
            object.recolor(Fill::Solid(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

1.1 À la recherche d'une impossible énumération des formes

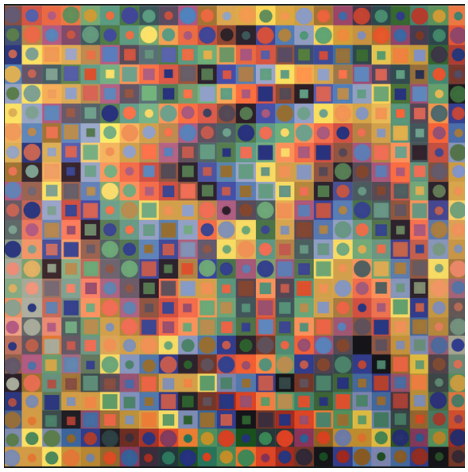


Fig. 1. – MAJUS (Victor Vasarely 1964)

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.

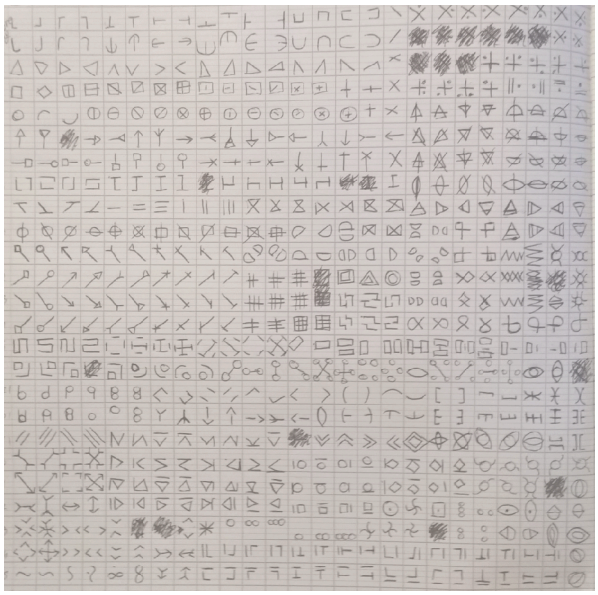


Fig. 2. – Un “alphabet” incomplet

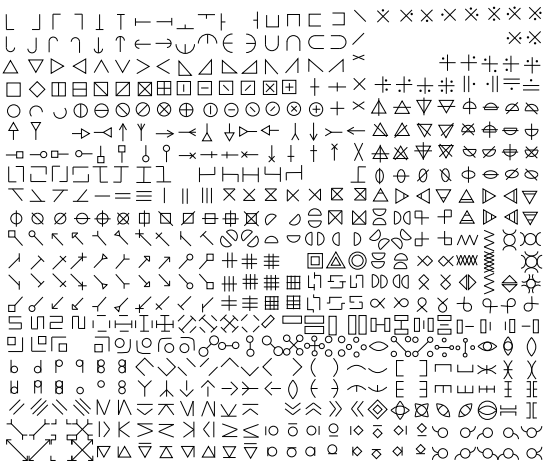


Fig. 3. – Une vectorisation

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 intéréssant 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.

1.2 Une approche procédurale ?

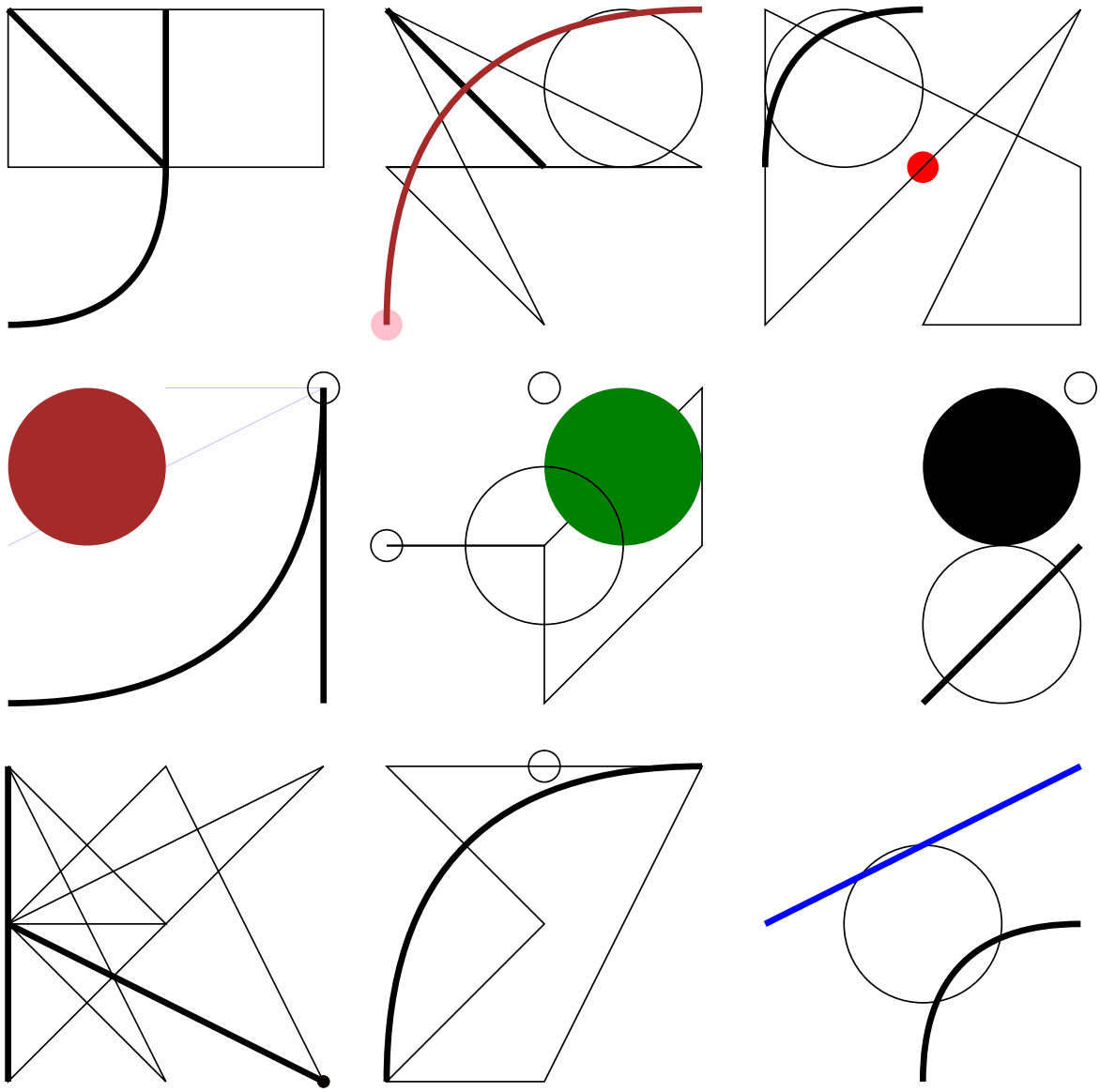


Fig. 4. – Exemples d'œuvres résultant d'une procédure de génération semi-aléatoire, basée sur une grille de 8 “points d'ancrages”

L'étape prochaine dans cette démarche était évidemment donc de générer procéduralement ces formes. Afin d'avoir des résultats intéréssants, 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:

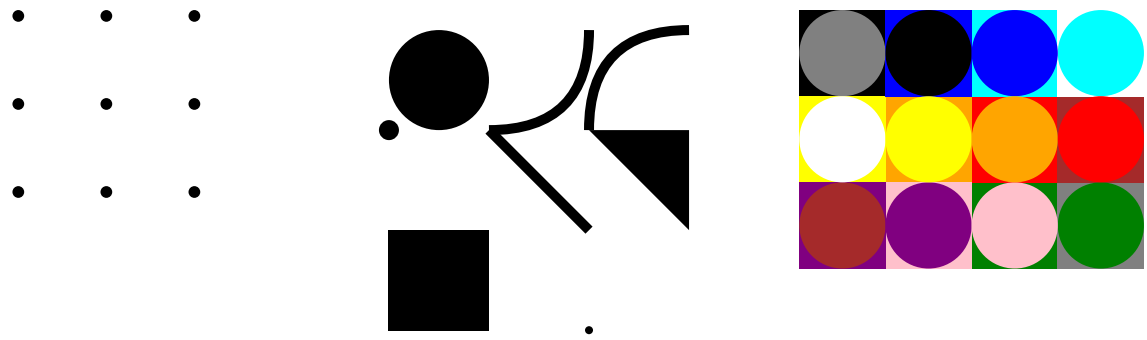


Fig. 5. – Vocabulaire visuel des premières ébauches: grille de placement à 9 points, formes et couleurs

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

1.3.1 Interprétation collective

https://shapemaker.gwen.works/soon.noredir

1.4 Lien musical

2 Une *crate* Rust avec un API sympathique

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 potentiomères 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'interagit 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 nécéssaire à la mise en place[1].

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 [3].

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 [4], qui est une cible de compilation pour quelques des langages compilés plus modernes, comme Go [5] or Rust [6]

En exportant la *crate* shapemaker en bibliothèque Javascript via wasm-bindgen [7], il est donc possible d'exoser à une balise `<script>` les fonctions de la bibliothèque, et brancher donc celles-ci à des *callbacks* donnés par l'API WebMIDI:

```rust
#[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(())
}
```

Liste 1. – Exposition de fonctions à WASM depuis Rust

```javascript
import init, { render_image } from "./shapemaker.js"

void init()

navigator.requestMIDIAccess().then((midiAccess) => {
  Array.from(midiAccess.inputs).forEach((input) => {
    input[1].onmidimessage = (msg) => {
      const [cmd, ...args] = [...msg.data]
      if (cmd !== 144) return

      // Touche enfoncée
      const [pitch, velocity] = args

      // get octave from pitch
      const octave = Math.floor(pitch / 12) - 1

      if (velocity === 0) {
        fadeOutElement(frameElement(color))
      } else {
        render_image(velocity / 128, octave)
      }
    }
  })
})
```

Liste 2. – Utilisation des fonctions exposées dans un script Javascript

Au final, on peut arriver à une performance live interactive [8] intéréssante, 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 [9], 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.

[1] Imaginez, 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 dispostif un site web, il suffit de brancher son contrôleur à l'ordinateur d'un · e ami · e, et c'est tout bon.

4.2 Amplitudes de *stems*

4.3 Export MIDI

4.4 Fichier de projet

4.5 Dépôt de « sondes » dans le logiciel de MAO

5 Performance

6 Conclusion

Bibliographie

[1] Victor Vasarely, *MAJUS*. 1964.

[2] Fondation Vasarely, « Planetary Folklore Period ». [En ligne]. Disponible sur: https://www.fondationvasarely.org/en/planetary-folklore-period/

[3] MDN Authors, « Web MIDI API ». [En ligne]. Disponible sur: https://developer.mozilla.org/en-US/docs/Web/API/Web_MIDI_API

[4] WebAssembly Community Group, « WebAssembly ». [En ligne]. Disponible sur: https://webassembly.org/

[5] Go language authors, « Go Wiki: WebAssembly ». [En ligne]. Disponible sur: https://go.dev/wiki/WebAssembly

[6] Rust authors, « WebAssembly — Le langage de programmation Rust ». [En ligne]. Disponible sur: https://www.rust-lang.org/fr/what/wasm

[7] Rust and WebAssembly Working Group, *The `wasm-bindgen` Guide*. [En ligne]. Disponible sur: https://rustwasm.github.io/wasm-bindgen/introduction.html

[8] Gwenn Le Bihan, *Démo de piano avec shapemaker*, (1 mai 2024). [En ligne Vidéo]. Disponible sur: https://www.instagram.com/p/C6byymcou5k/

[9] 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, 8 mars 2023. [En ligne]. Disponible sur: https://svgwg.org/svg2-draft/intro.html#AboutSVG