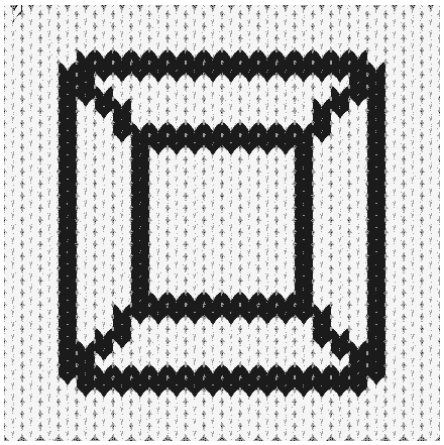
processing, animation, em

Cube en rotation façon tricot

Allez je vais décrire le procédé emberlificoté, alambiqué et nébuleux qui m'a conduit à fabriquer ce gif



L'idée est de représenter une forme en 3D simple, comme les premières images de synthèse produites dans les années 1960, en animant des mailles tricotées, un truc plutôt kitsch en somme.

Comme contrainte imposée, le fichier gif doit faire moins de 1 Mo

Pour réaliser l'animation plusieurs étapes ont été nécessaires :

- fabriquer des images d'un cube en rotation avec une épaisseur de trait importante (processing)
- réduire ces images (ligne de commande)
- à partir des images réduites, fabriquer des images ou chaque pixel est remplacé par une maille (processing)
- assembler cette série d'images sous forme d'animation (ligne de commande)

Tout ça est un peu fastidieux alors j'ai écrit un script qui en automatise une partie. Et puis, ça pourrait être réduit pour n'avoir qu'un seul script processing et une commande pour assembler les images.

Script

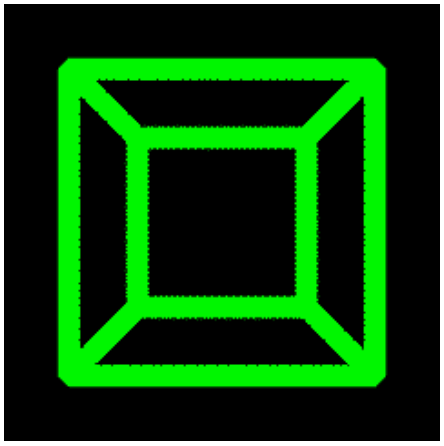
Le script :

```
#!/bin/bash

# 1re ligne ne fonctionne pas : probablement erreur openGL avec xvfb ...
#xvfb-run /home/emoc/processing-3.5.3/processing-java --sketch="/home/emoc/sketchbook/2020_KI/cube_en_rotation/" --run
mogrify -path /home/emoc/Bureau/OPENATELIER_1920/GIF_futur/cube_small -resize 24x36! /home/emoc/sketchbook/2020_KI/cube_en_rotation/*.png
convert -delay 8 -loop 0 /home/emoc/Bureau/OPENATELIER_1920/GIF_futur/cube_small/cube_*.png cube_small_anim.gif
xvfb-run /home/emoc/processing-3.5.3/processing-java --sketch="/home/emoc/sketchbook/2020_KI/TIPL_stitch_001/" --run
"/home/emoc/Bureau/OPENATELIER_1920/GIF_futur/cube_small" "/home/emoc/Bureau/OPENATELIER_1920/GIF_futur/cube_knit"
convert -delay 8 -loop 0 /home/emoc/Bureau/OPENATELIER_1920/GIF_futur/cube_knit/image_*.png -colors 4 cube_knit_anim.gif
```

La première étape est en commentaire car apparemment démarrer processing pour faire de la 3D en mode headless cause une erreur que je n'ai pas cherché à résoudre... Il faut donc créer les images de l'animation en démarrant le script processing par l'interface graphique.
mogrify et convert sont des commandes d'imagemagick

Animation du cube en rotation



cube_en_rotation.pde (cliquer pour afficher le code)

cube_en_rotation.pde

```
/*
  cube en rotation
  processing 3.5.3 @ kirin / Debian Stretch 9.5
  20200510 / pierre@lesporteslogiques.net
 */

float angle = 0;
float steps = 36;
void setup() {
  size(240, 240, P3D);
  frameRate(12);
}

void draw() {
  background(0);
  stroke(0, 255, 0);
  strokeJoin(MITER);
  strokeJoin(SQUARE);
  //fill(0, 0, 0);
  noFill();
  strokeWeight(12);
  push();
  translate(width/2, height/2);
  rotateX(radians(angle));
  rotateY(radians(angle)); //, angle);
  box(width/2, width/2, width/2);
  pop();
  angle += 360 / steps;
  angle %= 360;
  saveFrame("cube_##.png");
  if (angle == 0) noLoop();
}
```

Traitement pour les mailles

TIPL_stitch_001.pde (cliquer pour afficher le code)

TIPL_stitch_001.pde

```
/*
  traitement d'images par lot pour transformer une animation en mailles de tricot!
  processing 3.5.3 @ kirin / Debian Stretch 9.5
  20200508 / pierre@lesporteslogiques.net

  permet de traiter un répertoire d'image complet et de créer une série d'images
  chaque pixel de l'image d'origine est représenté par une maille noire ou blanche selon la luminosité du pixel d'origine
  utilisable en ligne de commande :
    xvfb-run /home/emoc/processing-3.5.3/processing-java --sketch="/home/emoc/sketchbook/2020_KI/TIPL_stitch_001/" --run
"/home/emoc/tipl/orig" "/home/emoc/tipl/dest"
*/

boolean GUIMODE = true;       // GUI ou ligne de commande ? Changé automatiquement si le script est lancé en ligne de commande

String dossier_orig = "/home/emoc/tipl/orig";  // dossier des images à traiter
String dossier_dest = "/home/emoc/tipl/dest";  // dossier des images transformées
String[] fichiers_a_traiter;  // liste des fichiers du répertoire à traiter
int nb_fichiers = 0;          // nombre de fichiers à traiter

PImage img_orig;              // image à traiter
```

```
PGraphics img_dest;           // image résultant du traitement

String fichier_orig = "";     // nom du fichier original à traiter
String fichier_dest = "";     // nom du fichier à créer
String chemin_orig = "";      // chemin complet vers le fichier original
String chemin_dest = "";      // chemin complet vers le fichier à créer
String extension = "png";     // extension et format de fichier à créer
String racine = "image";      // racine du nom de fichier à créer

int compteur = 1;             // numéro du premier fichier, les autres fichiers seront nommés à partir de là
String numero;                // formatage du nombre contenu dans le nom de fichier à créer

// variables spécifiques à ce traitement
float maille_larg = 18;       // largeur d'une maille en pixel
float maille_haut = 12;       // hauteur d'une maille en pixel
float maille_dip  = 6;        // pointe de maille

void setup() {
  size(800, 300);
  init();                     // traitement des arguments associés à la ligne de commande
}

void draw() {

  println("dossier à traiter : " + dossier_orig);
  println("dossier des fichiers traités : " + dossier_dest);
  fichiers_a_traiter = listFileNames(dossier_orig);
  printArray(fichiers_a_traiter);
  nb_fichiers = fichiers_a_traiter.length;

  for (int i = 0; i < nb_fichiers; i++) {

    chemin_orig = dossier_orig + "/" + fichiers_a_traiter[i];
    numero = nf(compteur+i, 4); // numero du fichier formaté 0001, 0002, etc.
    fichier_dest = racine + "_" + numero + "." + extension;
    chemin_dest = dossier_dest + "/" + fichier_dest;
    println("traitement du fichier " + chemin_orig);
    println("fichier à créer : " + chemin_dest);

    img_orig = loadImage(chemin_orig);

    img_dest = createGraphics(img_orig.width * (int)maille_larg, img_orig.height * (int)maille_haut);
    img_dest.beginDraw();
    img_dest.background(127);
    img_dest.stroke(0);

    img_orig.loadPixels();
    for (int j = 0; j < img_orig.width * img_orig.height; j ++) {
      int x = j%img_orig.width;
      int y = floor(j/img_orig.width);
      color c = img_orig.pixels[j];
      float b = brightness(c);
      //img_dest.fill(b);

      if (b > 127) {          // couleur de contraste
         img_dest.fill(255);
      } else {                // couleur de fond
         img_dest.fill(0);
      }
      knit (x * maille_larg, y * maille_haut, maille_larg, maille_haut, maille_dip);
    }
    img_orig.updatePixels();
    img_dest.endDraw();
    img_dest.save(chemin_dest);
  }

  if (!GUIMODE) {
    exit();
  }
  noLoop();
}

// Tracer une maille
// Tracé des courbes d'après sweaterify de Mariko Kosaka https://github.com/kosamari/sweaterify
void knit (float x, float y, float sWidth, float sHeight, float dip) {
  img_dest.beginShape();
    img_dest.vertex(x + (sWidth / 2), y + dip);
    img_dest.quadraticVertex(x + sWidth - (sWidth / 3), y - (sHeight / 12), x + sWidth - (sWidth / 10), y - sHeight / 4);
    img_dest.quadraticVertex(x + sWidth - (sWidth / 50), y, x + sWidth - (sWidth / 70), y + (sHeight / 10));
  img_dest.endShape();

  img_dest.beginShape();
    img_dest.vertex(x + sWidth - (sWidth / 70), y + (sHeight / 10));
    img_dest.bezierVertex(x + sWidth, y + (sHeight / 4), x + sWidth, y + (sHeight * 0.50), x + sWidth - (sWidth / 15), y + (sHeight
* 0.66));
    img_dest.bezierVertex(x + sWidth - (sWidth * 0.3), y + sHeight, x + sWidth - (sWidth * 0.3), y + sHeight, x + sWidth - (sWidth
/ 2) + (sWidth / 20), y + sHeight + sHeight / 3);
  img_dest.endShape();

  img_dest.beginShape();
    img_dest.vertex(x + sWidth - (sWidth / 2) + (sWidth / 20), y + sHeight + sHeight / 3);
    img_dest.quadraticVertex(x + sWidth - (sWidth * 0.55), y + (sHeight * 0.7), x + sWidth - (sWidth / 2), y + dip);
    img_dest.vertex(x + sWidth - (sWidth / 2), y + dip);
  img_dest.endShape();
```

```
  img_dest.noStroke();

  img_dest.beginShape();
    img_dest.vertex(x + sWidth - (sWidth / 2), y + dip);
    img_dest.vertex(x + sWidth - (sWidth / 70), y + (sHeight / 10));
    img_dest.vertex(x + sWidth - (sWidth / 2) + (sWidth / 20), y + sHeight + sHeight / 3);
  img_dest.endShape();

  img_dest.beginShape();
    img_dest.vertex(x + (sWidth / 2) - sWidth * 0.03, y + dip);
    img_dest.quadraticVertex(x + (sWidth * 0.4), y + (sHeight / 12), x + (sWidth / 10), y - sHeight / 4);
    img_dest.quadraticVertex(x + (sWidth / 50), y, x + (sWidth / 70), y + (sHeight / 10));
  img_dest.endShape();

  img_dest.beginShape();
    img_dest.vertex(x + (sWidth / 70), y + (sHeight / 10));
    img_dest.bezierVertex(x, y + (sHeight / 4), x, y + (sHeight * 0.50), x + (sWidth / 15), y + (sHeight * 0.66));
    img_dest.bezierVertex(x + (sWidth * 0.3), y + sHeight, x + (sWidth * 0.3), y + sHeight, x + (sWidth / 2) - (sWidth / 40), y +
sHeight + sHeight / 3);
    img_dest.quadraticVertex(x + (sWidth * 0.56), y + (sHeight + sHeight / 4), x + (sWidth / 2) - sWidth * 0.05, y + dip);
    img_dest.vertex(x + (sWidth / 2) - sWidth * 0.03, y + dip);
  img_dest.endShape();

}

// ***************************************************************************

// Fonction pour traiter les arguments de la ligne de commande
void init() {
  if (args != null) {
    GUIMODE = false;
    println("Arguments : " + args.length);
    for (int i = 0; i < args.length; i++) {
      println(args[i]);
    }
    if (args.length == 2) {
      dossier_orig = args[0];
      dossier_dest = args[1];
    } else {
      println("arguments insuffisants (indiquer dossier de départ et dossier d'arrivée)");
      exit();
    }
  } else {
    println("pas d'arguments transmis par la ligne de commande");
  }
}

// fonction pour lister les fichiers d'un dossier, renvoie un tableau de chaines de caractères
// d'après Daniel Shiffman : https://processing.org/examples/directorylist.html
String[] listFileNames(String dir) {
  File file = new File(dir);
  if (file.isDirectory()) {        // C'est un dossier
    String names[] = file.list();
    names = sort(names);
    return names;
  } else {                         // If it's not a directory
    return null;
  }
}
```