Le Squelette

Boero Benoit, Borget Antoine, Crisan Andrei, Dan Anca, Hutzler Eïleen, Revnic Vlad and Roman Tudor.

2018 - 2019

1 Introduction

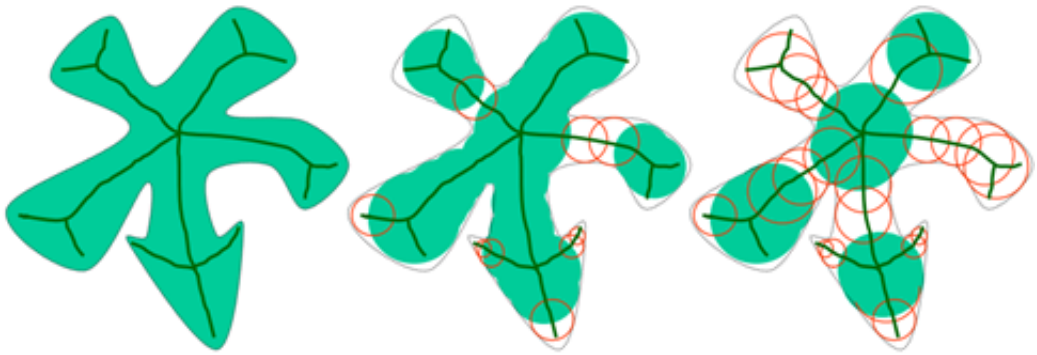
Nous avons travaillé sur le sujet du squelette à partir de l'énoncé suivant : "Le biologiste, Harry Blum a introduit un nouvel outil pour aider à caractériser la forme des individus de la même espèce et à distinguer les espèces les unes des autres. Le squelette de forme S, noté MA(S), est défini par tous les centres des boules maximales contenues dans S. Que peut-on dire sur la forme des squelettes ? Peut-on trouver une forme pour un squelette donné ?"

2 Méthode

2.1 Régles

Pour résoudre ce problème, nous avons commencé, avec nos coéquipiers roumains et français, par définir quelques propriétés du squelette : Le squelette ne peux pas toucher le contour de la forme, cela peut parfois arriver en un seul point. Pour un objet donné, nous pouvons trouver plusieurs squelettes, cependant il ne peut y avoir qu'un squelette qui respecte les règles anatomiques, par exemple le fait que chaque sphère contenue dans la forme ait un rayon maximum tout en restant à l'intérieur de la forme.

Figure 1 : Disques maximaux



Pour que la forme soit la plus précise possible, le squelette doit contenir des ramifications, non représentées dans le squelette osseux. Par exemple, pour un chameau, on peut représenter les bosses seulement avec les ramifications du squelette. La méthode la plus pratique pour représenter le squelette d'un polygone régulier avec un même nombre de côté est de construire le squelette à équidistance du centre de la figure et de son contour. Pour un squelette donné, nous pouvons construire un nombre infini de formes, puisque le centre des sphères ne limite pas leur rayon. Cela arrive lorsqu'on ne sait pas quel type de corps ou contour on recherche.

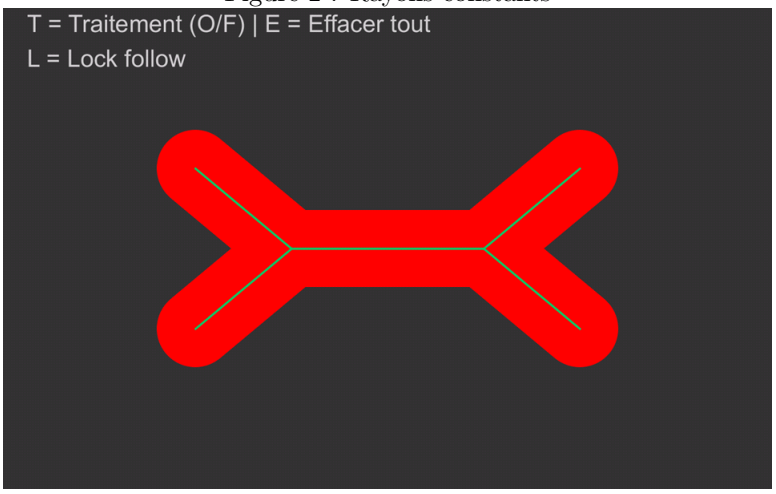
2.2 Programme

Nous avons alors créé un programme qui dessinerait des cercles autour de différents points du squelette afin de voir apparaître des formes possibles pour un squelette donné ou choisi aléatoirement. Nous avons pour cela utilisé différentes fonctions mathématiques :

2.2.1 Cercle de diamètres constants

Une fonction "simple" qui fait des cercles de taille identiques en chaque point du squelette, nous y avons ajouté un curseur pour que l'utilisateur du programme puisse modifier à sa guise la taille du rayon des cercles.

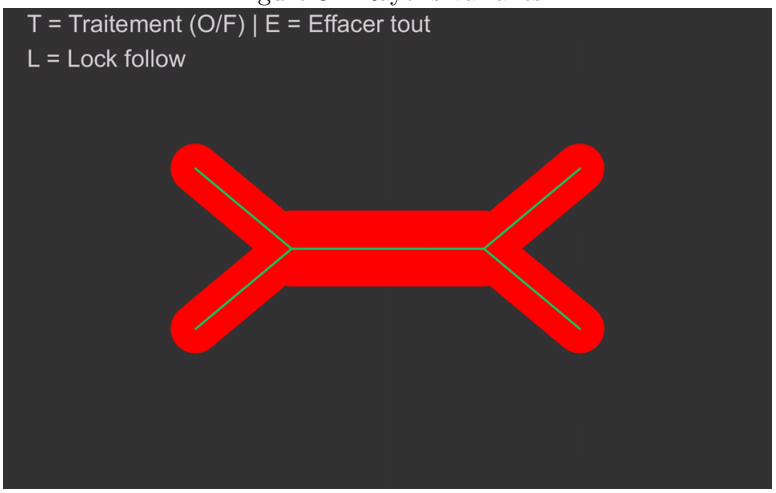
Figure 2 : Rayons constants



2.2.2 Cercle de diamètres variant avec les longueurs

Une fonction suivant une "relation de proportionnalité", qui augmente la taille du rayon lorsque la longueur du trait du squelette croît et inversement quand elle diminue.

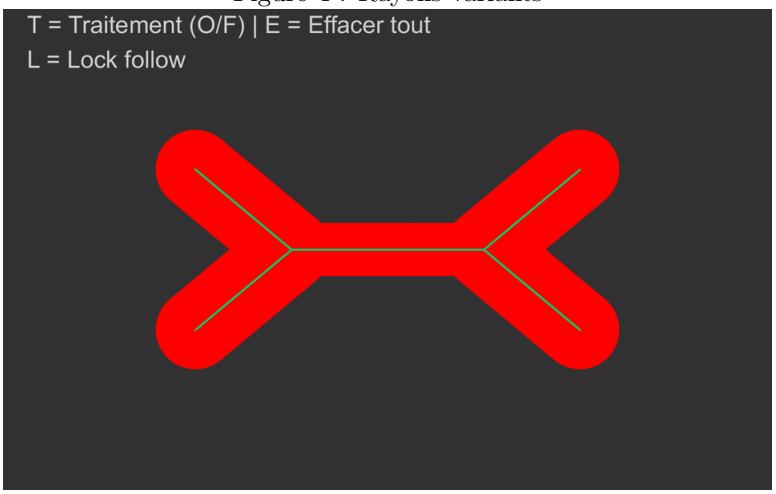
Figure 3 : Rayons variants



2.2.3 Cercle de diamètres variant inversement au longueurs

Une fonction "inverse", qui fonctionne en antithèse par rapport à la fonction précédente, c'est-à-dire que plus la longueur du trait est importante, plus le rayon aura une petite longueur.

Figure 4 : Rayons variants

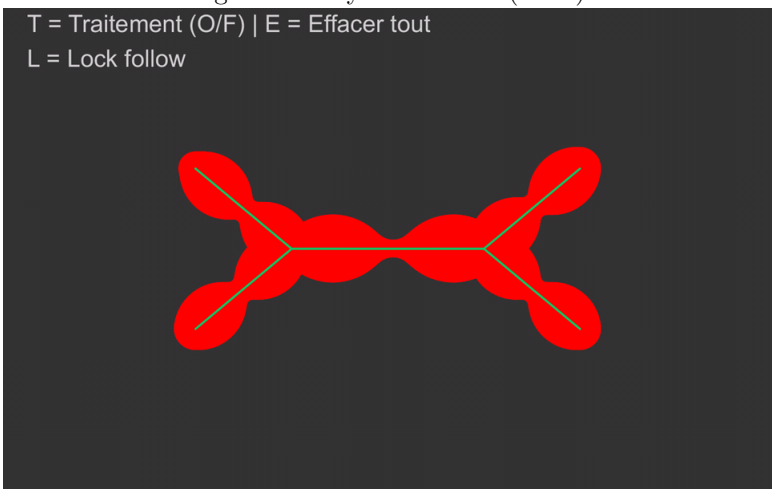


2.2.4 Cercle de diamètres variant avec un sinus

Une fonction "sinus", qui fait varier le rayon des cercles en suivant une fonction de forme sinusoïdale indépendamment de la longueur des traits, cela signifie que la taille du rayon augmentera et diminuera de façon périodique pour former des ondulations autour du squelette que l'on pourrait apparenter aux courbes

du corps en les ajustant correctement. C'est d'ailleurs pour cette raison que nous avons ajouté deux curseurs supplémentaires ayant pour but de contrôler la fréquence et la taille des variations de la fonction "sinus" selon la convenance de l'utilisateur du programme.

Figure 5 : Rayons variants (sinus)



3 Conclusion

Grâce à ces différentes fonctions découvertes au cours de l'année, nous avons pu progresser et bien que le chemin soit encore long, nous pouvons déjà apporter des éléments de réponse aux questions posées par le chercheur, puisque nous avons déterminé des propriétés sur la forme du squelette et en recherché différents types de formes pour un squelette donné.

4 Appendix

4.1 Programme (Processing, java)

```
// tableau des points en m moire
IntList pointsX;
IntList pointsY;
//tableau des lignes en m moire
IntList traitsX;
IntList traitsY;
//coordon  du dernier point le plus proche du curseur
int pointsNearX;
int pointsNearY;
//coordon  du dernier point le plus proche du curseur (LOCK)
```

```
int pointsNearXlock;
int pointsNearYlock;
//variable de boucle
int i=0;
int j=0;
//variables de couleures RVB
int rouge=0;
int vert=0;
int bleu=0;
//distance entre le curseur et son point le plus proche
float distance=width*height;
//traitement (O/F)
boolean traitement=false;
//rayon cercle de traitement (ON)
float rayonCercle=50;
//rayon cercle de traitement (OFF)
float rayonCercle2=width*height/350;
//Objet police
PFont f;
//Lock (ON/OFF)
boolean lock=false;
//fullscreen (ON/OFF)
boolean fullscreen=false;
//texte menu
String texteMenu = "T = Traitement (O/F) | E = Effacer tout\nL = Lock follow";

HScrollbar hs1, hs2;  // Two scrollbars

void setup() {
  //size(640, 360);
  fullScreen();
  //initialisation tableaux
  pointsX=new IntList(0);
  pointsY=new IntList(0);
  traitsX=new IntList(0);
  traitsY=new IntList(0);
  fill(0);
  stroke(255);
  //initialisation objet font f
  f = createFont("Arial", 16, true);
  hs1 = new HScrollbar(0, 8, width/4, 16, 16);
  hs2 = new HScrollbar(0, 32 , width/4, 16, 16);
}

void draw() {
  if (traitement==false) {
```

```
background(0);
textFont(f, height/20);
fill(255);
text(texteMenu, height/20, height/20);
distance=width*height;
strokeWeight(10);

//afficher des lignes entre les points
for (i=0; i<traitsX.size(); i++) {
  stroke(200, 200, 0);
  line(traitsX.get(i), traitsY.get(i), pointsX.get(i), pointsY.get(i));
}
/*points rouge quand la sourie est sur un point
 +affichage des points en m moire
 +v rification du points le plus proche du curseur
 */
strokeWeight(2);
for (i=0; i<pointsX.size(); i++) {
  if (dist(pointsX.get(i), pointsY.get(i), mouseX, mouseY)<=rayonCercle2/2)
    rouge=255;
    vert=0;
    bleu=0;
  }
  stroke(200, 200, 0);
  fill(200, 200, 0, 150);
  ellipse(pointsX.get(i), pointsY.get(i), rayonCercle2, rayonCercle2);
  if (dist(mouseX, mouseY, pointsX.get(i), pointsY.get(i))<distance) {
    distance = dist(mouseX, mouseY, pointsX.get(i), pointsY.get(i));
    pointsNearX=pointsX.get(i);
    pointsNearY=pointsY.get(i);
  }
}

//suivre le curseur
stroke(rouge, vert, bleu);
strokeWeight(4);
fill(rouge, vert, bleu, 150);
ellipse(mouseX, mouseY, rayonCercle2, rayonCercle2);
rouge=0; //rouge par d faut
vert=200; //vert par d faut
bleu=0; //bleu par d faut

//fix du bug ligne (0,0)
if (traitsX.size()==0) {
  pointsNearX=mouseX;
  pointsNearY=mouseY;
```

```
    }

    //afficher une ligne entre le curseur et les points
    if (!lock) {
      line(mouseX, mouseY, pointsNearX, pointsNearY);
    } else {
      line(mouseX, mouseY, pointsNearXlock, pointsNearYlock);
    }
  } else {
    background(50);
    textFont(f, height/20);
    fill(205);
    text(texteMenu, height/20, height/20);
    hs1.update();
    hs1.display();
    hs2.update();
    hs2.display();


    //afficher des lignes entre les points
    for (i=0; i<traitsX.size(); i++) {
      stroke(0, 200, 100);
      line(traitsX.get(i), traitsY.get(i), pointsX.get(i), pointsY.get(i));
    }
    //afficher les points en m moire
    for (i=0; i<pointsX.size(); i++) {

      stroke(100, 200, 100);
      fill(100, 200, 100, 150);
      ellipse(pointsX.get(i), pointsY.get(i), 10, 10);
      stroke(100, 50, 100);
      fill(0, 0, 0, 0);
      int maxSin1 = 100;
      int maxSin2 = 100;
      rayonCercle = map(dist(pointsX.get(i), pointsY.get(i), traitsX.get(i), tra
      for (float op = 0; op<1; op+=0.01) {
        noStroke();
        fill(255, 0, 0);
        ellipse(op*traitsX.get(i)+(1-op)*pointsX.get(i), op*traitsY.get(i)+(1-op
      }
    }
  }
}

void mousePressed() {
  //ajouter un pts
```

```
  if (mouseButton==LEFT&&!traitement) {
    if (!lock) {
      pointsX.append(mouseX);
      pointsY.append(mouseY);
      traitsX.append(pointsNearX);
      traitsY.append(pointsNearY);
    } else {
      traitsX.append(pointsNearXlock);
      traitsY.append(pointsNearYlock);
      pointsX.append(mouseX);
      pointsY.append(mouseY);
    }
    //bug premier points non-li
    if (pointsX.size()==2) {
      traitsX.set(0, pointsX.get(1));
      traitsY.set(0, pointsY.get(1));
    }
  }
  //suprimer un point
  for (i=0; i<pointsX.size(); i++) {
    if (mouseButton==RIGHT&&dist(pointsX.get(i), pointsY.get(i), mouseX, mouseY)<
      traitsX.remove(i);
      traitsY.remove(i);
      pointsX.remove(i);
      pointsY.remove(i);
    }
  }
}
void keyPressed() {
  if (key=='t') {
    if (traitement) {
      traitement=false;
    } else {
      traitement=true;
    }
  }
  if (key=='e') {
    pointsX=new IntList(0);
    pointsY=new IntList(0);
    traitsX=new IntList(0);
    traitsY=new IntList(0);
  }
  if (key=='l') {
    if (lock) {
      lock=false;
    } else {
```

```
      lock=true;
      pointsNearXlock=pointsNearX;
      pointsNearYlock=pointsNearY;
    }
  }
}
class HScrollbar {
  int swidth, sheight;    // width and height of bar
  float xpos, ypos;       // x and y position of bar
  float spos, newspos;    // x position of slider
  float sposMin, sposMax; // max and min values of slider
  int loose;              // how loose/heavy
  boolean over;           // is the mouse over the slider?
  boolean locked;
  float ratio;

  HScrollbar (float xp, float yp, int sw, int sh, int l) {
    swidth = sw;
    sheight = sh;
    int widthtoheight = sw - sh;
    ratio = (float)sw / (float)widthtoheight;
    xpos = xp;
    ypos = yp-sheight/2;
    spos = xpos + swidth/2 - sheight/2;
    newspos = spos;
    sposMin = xpos;
    sposMax = xpos + swidth - sheight;
    loose = l;
  }

  void update() {
    if (overEvent()) {
      over = true;
    } else {
      over = false;
    }
    if (mousePressed && over) {
      locked = true;
    }
    if (!mousePressed) {
      locked = false;
    }
    if (locked) {
      newspos = constrain(mouseX-sheight/2, sposMin, sposMax);
    }
    if (abs(newspos - spos) > 1) {
```

```
      spos = spos + (newspos-spos)/loose;
    }
  }

  float constrain(float val, float minv, float maxv) {
    return min(max(val, minv), maxv);
  }

  boolean overEvent() {
    if (mouseX > xpos && mouseX < xpos+swidth &&
       mouseY > ypos && mouseY < ypos+sheight) {
      return true;
    } else {
      return false;
    }
  }

  void display() {
    noStroke();
    fill(204);
    rect(xpos, ypos, swidth, sheight);
    if (over || locked) {
      fill(0, 0, 0);
    } else {
      fill(102, 102, 102);
    }
    rect(spos, ypos, sheight, sheight);
  }

  float getPos() {
    // Convert spos to be values between
    // 0 and the total width of the scrollbar
    return spos * ratio;
  }
}
```