Bachelor Thesis: 3D Animation of Animal Motion from Still

Guillaume Loranchet

April 2020

Abstract

Animal motion, due to ... is way less documented than human motion. We thus have less usable 3D animated models of animals. Furthermore,

1 Introduction (current situation of 3D modeling)

2 Previous Work

3 2D Animation

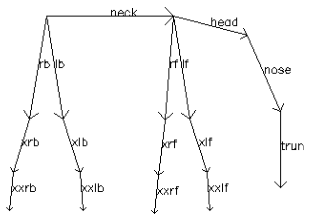


Figure 1: Initial Skeleton

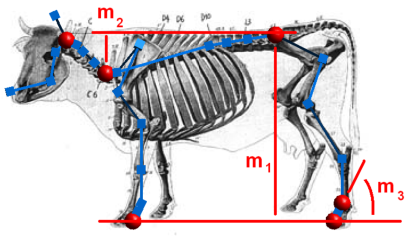


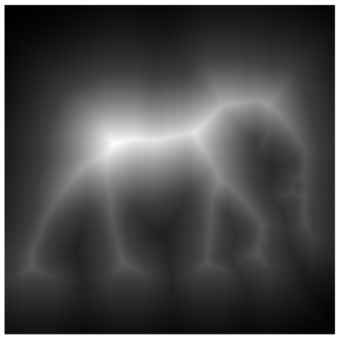
Figure 2: Morphable model

We start by constructing a 3D skeleton. The skeleton is constructed manually but could be made quicker like presented in the paper Morphable model of quadrupeds skeletons for animating by Reveret et al. (2009) [...] (Figure ...). Indeed, by using a database of 8 already made skeletons, and with only 3 key measures, they can create a skeleton for any quadrupeds. Those three measures could also be useful for the second step, that is the placement of the skeleton on the different shapes. We could, for each image, select the hip (equivalent to m1 on the figure) resize the image and translated it so that all images have the same m1. The first two steps could probably be more automated but a small error at this point could compromise the rest of the process. With only three initial points to create the skeleton, and one additional point for each images, we'd make sure that there will not be any important mistake up to this point. Finally, for each bone, we add an ellipse that will allow us, for a reference surface S and a query surface Q, to define a score r as follow:

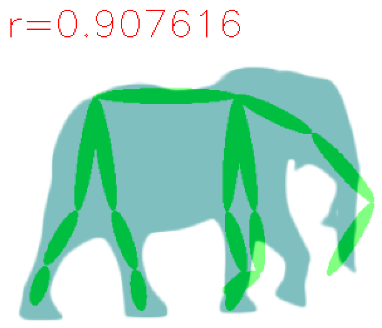
$$r = \frac{S \cap Q}{Q}$$

In other words, r will be maximum (equal to 1) when the entire ellipse is inside the animal shape. In the initial paper from Manon Romain, she uses the Intersection over Union (IOU) which measure how much the skeleton fills up the entire shape. But as the ellipses are not constructed based on the shape of the animal, it would be more accurate to measure only how much the ellipse is inside the shape.

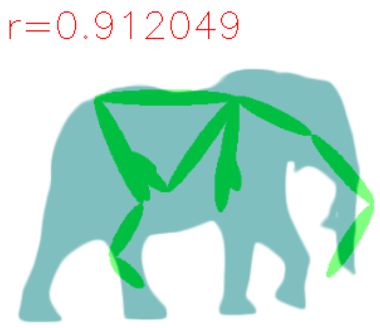
The next step in to make to skeleton fit the shape. The first method we tried was using distance transform. Each pixel of the image is attributed a value between 0 and 1 depending how far it is from the closest boundary. We firstly tried a naive algorithm by simply rotating each bone such that the local score is maximum. However, this method depends too much on the size of the ellipses and the distance transform image itself is pretty far from the correct skeleton.



(a) Distance transform



(b) Apply rotation only on the lower part

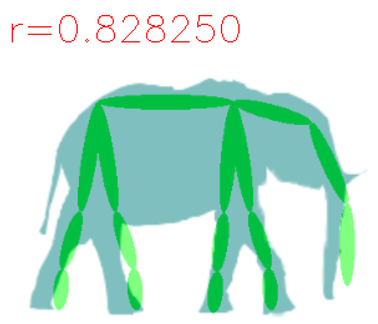


(c) Apply rotation on the entire leg

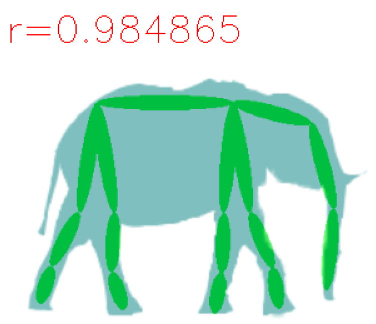
Figure 3: Distance Transform naive algorithm

The second method tries to find the middle of the legs (or trump). For each leg, the algorithm starts by the middle bone (e.g xrf) and finds the two border of the leg. It first detects toward which direction the bone should rotates by looking at the local score and then rotates the bone until the extremity changes of surface (i.e going from outside to inside the shape). We can average the two positions to find a good approximation of the middle of the leg. We also add constraints on the angle of each bone such that if the angle reach one of the boundary, we know the direction was wrong and avoid absurd results. Furthermore, we assume that the extremity is either on the left side of the leg, inside or on the right side, thus one should always be able to find the two borders. Finally, given a good enough initial position, it should accurately dissociates the two legs.

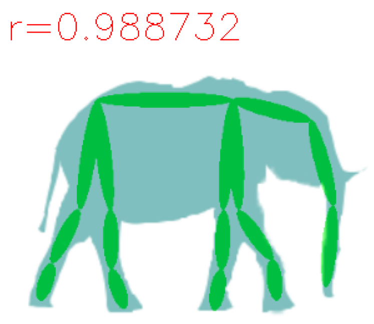
Once the lower part of the leg (xrf and xxrf) are well placed, we can improve a bit the position of the leg by rotating the upper part (rf) such that the score of the leg increases (Figure ...). We also make sure that xxrf keeps the same direction as it is already well positioned. We can see that it doesn't improve a lot the global position of the leg. As before, it is also very dependant on the size of the ellipse.



(a) Initial skeleton position



(b) Middle Point Algorithm



(c) Add improvement on the upper part

Figure 4: Middle Point Algorithm

While this method gives an reasonably fast and accurate mapping of the skeleton, it also has some issues. The first could occur if the skeleton doesn't perfectly fit the shape. Indeed if the skeleton is bigger than the leg (Figure ...), one of our assumption doesn't hold anymore and the extremity will either be block in the "corner" of the leg or rotate until it reaches the maximum angle.

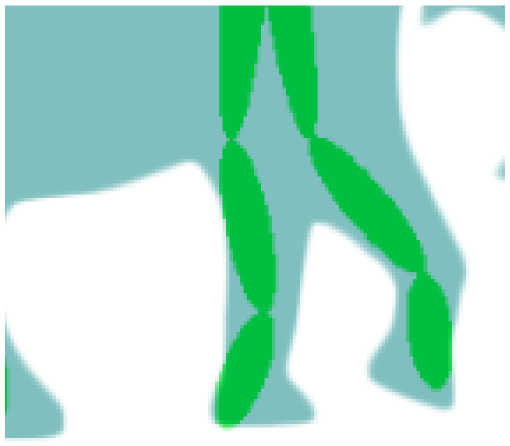
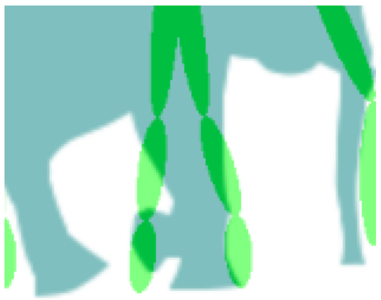
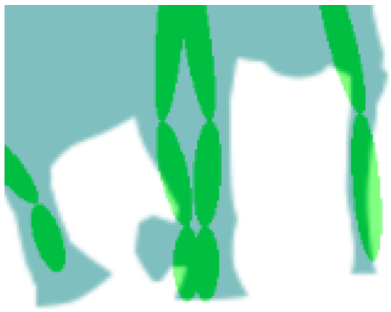


Figure 5: Skeleton size problem

The other issue concerns the position where the two legs are overlapping each other. As shown in Figure ..., the two front legs are mapped to the same leg. This problem could also be due to the imprecise construction of the skeleton or resizing. While we haven't found yet how to fix this specific case, one solution could be to add another constraint on the leg. For example, in a walking movement, xxrf shouldn't be able to be directed toward the front while being behind the left leg, and thus conclude that it should rotate 90 degrees in the other direction.


(a) Initial skeleton position


(b) Middle Point Algorithm

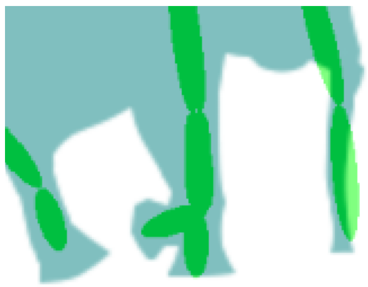

(c) Manual placement

Figure 6: Middle Point Algorithm issue

We now have n different positions for the skeleton. As, by construction, the left leg is always in front of the right one, we also create n new positions by inverting the order of the legs. However, by doing this, we insure that the inversion of the front and of the back legs happen at the same time, which is usually not exactly true. Indeed, most of the time, the back legs cross a bit before the front legs.

We finally need to find an optimal ordering of the $2n$ positions in order to make a realistic movement. For each position, we associate a vector of dimension 4 containing the angle between each leg and the horizontal axis (Figure ...). We can then compute a simple score between two positions by taking the norm of the difference of the two vector. The score measure how different two consecutive positions are. We use a Dijkstra algorithm to minimize the total score of the ordering. The algorithm start with two initials positions and at each step, add the node that minimizes the total score, while keeping in memory the score of the other possible paths. In the worse case, this algorithm has the same complexity as a naive algorithm but most of the time find a minimum before exploring every path. Furthermore, it is guaranteed to find the smallest score. We add a last constraint which is that two opposite positions (same position with an inversion of the legs) can't be juxtaposed. Indeed, the motion is not symmetric whether the leg is moving forward or backward.

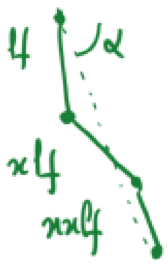
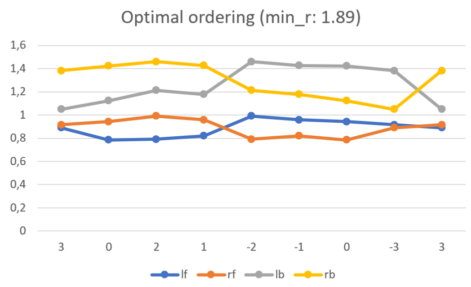
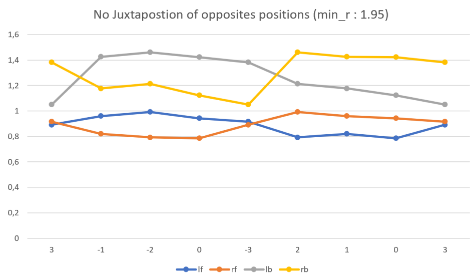


Figure 7: Initial Skeleton



(a) Initial skeleton position



(b) Middle Point Algorithm

Figure 8: Middle Point Algorithm issue

4 3D Render

Algorithm 1: Put several words on two guides

```
function Algo1(Guide, Word):
    function Average(obj):
        return the following attributes of obj: Average in x, Left point, Right point, Median of the lowest
          points
    Average (Guide), Average (Word)
    for point in Word do
        $point.x \leftarrow points.x + (guide.LeftPoint - word.LeftPoint)$
        $point.y \leftarrow point.y + (guide.AverageInY - word.MedianLowestPoint)$
    $Word.update()$
    return
```

Algorithm 2: Put several words on horizontal guides

```
function New_guide ($Guides, Distance$):
    $NewGuide \leftarrow Guides[-1]$
    for point in NewGuide do
        $point.y \leftarrow point.y + distance$
    return NewGuide
function Algo2 ($Guides, Words$):
    if $Word.length() == 0$ then
        $Words.update()$
        return
    $Word \leftarrow Words[0]$
    $Guide \leftarrow Guides[0]$
    $Guide.Size \leftarrow Guide.RightPoint - Guide.LeftPoint$
    $Word.Size \leftarrow Word.RightPoint - Word.LeftPoint$
    if $Word.size < Guide.size$ then
        Algo1 (Guide,Word)
        while $Guide[0].x < Guide.LeftPoint + Word.size + 25$ do
            $Guide.remove(Guide[0])$
        $Words.remove(Word)$
        Algo2 (Guides,Words)
    else
        if $Guides.size > 0$ then
            Algo2 (Guides[1],Words)
        else
            $NewGuide \leftarrow$ New_guide (PreviousGuide, (2ndGuide.AverageInY - 1stGuide.AverageInY))
            Algo2 (NewGuide,Words)
```

5 Conclusion

6 Future work

References

[1] Schilit, B.N., Golovchinsky, G. and Price, M.N. *Beyond paper: supporting active reading with free form digital ink annotations (XLbibris).* CHI 1998, ACM, 1998.
https://www.fxpal.com/research-projects/xlibris/

[2] Fabrice Matulic and Moira C. Norrie, 2012. *Supporting Active Reading on Pen and Touch-operated Tabletops* DOI: http://dx.doi.org/10.1109/TABLETOP.2007.12

[3] Craig Tashman, Cristiano Ghersi, Stephen Dukker, and Dalas Verdugo. 2010. LiquidText.
https://www.liquidtext.net/

[4] Maneesh Agrawala and Michael Shilman. 2005. *DIZI: A Digital Ink Zooming Interface for Document Annotation.* DOI: http://dx.doi.org/10.1007/11555261$_9$

[5] Dongwook Yoon, Nicholas Chen, and François Guimbretière. 2013. *TextTearing: Opening White Space for Digital Ink Annotation.*
DOI: `http://dx.doi.org/10.1145/2501988.2502036`

[6] Hugo Romat, Emmanuel Pietriga, Nathalie Henry-Riche, Ken Hinckley, Caroline Appert. 2019. *SpaceInk: Making Space for In-Context Annotations*
DOI: `https://dl.acm.org/citation.cfm?id=3347934`

https://github.com/guillaume-lrt/inkIntership_cse303