Creating character Poses based on Lines of Action

Martin Bergerès

***Abstract*—The Line of Action (LOA) is a tool used by cartoonists and illustrators to define the pose or the action of a character. It allows the illustrator to give a general idea to the pose and draw the character around or on this line. The general expression of the body of the character is then based on a simple and aesthetic line. This project is inspired by the paper *The Line of Action: an Intuitive Interface for Expressive Character Posing* published by Martin Guay, Marie-Paule Cani and Rémi Ronfard presenting an interface to match 3D character poses to LOAs. It presents a new interface that allows the user to create 3D poses based on LOAs while respecting the general geometry of the skeleton. This interface offers more flexibility in the chosen body line and a faster computation.**

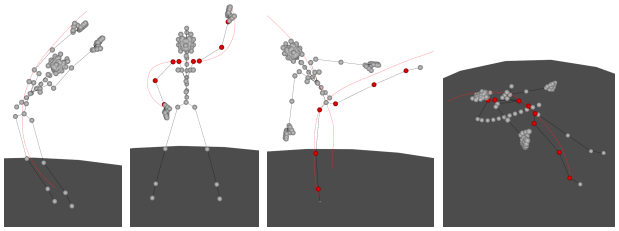

Fig. 1. Expressive poses created with simple Lines of Action.

I. Introduction

Animating a 3D character is a challenge and requires animation skills. Several rules should be respected: geometry of the skeleton, coherence of all the body parts, etc... It's also very time consuming to manually create 3D poses or animations based on the character's skeleton. On the top of that, in order for the pose to be expressive and dramatic, the different body part should remain coherent and follow a general expressive idea. Inspired by the practice of the cartoonists and their Lines of Actions (LOA), and a previous paper implementing character posing based on LOAs, we propose an interface that allows the user to generate 3D character poses based on simple sketch lines. It should allow any illustrator to easily and rapidly create simple and expressive poses.

While prior work focused on posing a 3D character from a LOA in solving optimization problems, this paper explores the alternative of creating these poses by recursive rotations of the joints of a subset of the skeleton called *skeleton segments*. We explain how to transform a selection of joints into separate *skeleton segments* in order to be able to match entire *body lines*. The computation for these poses are much faster and allows more type of *body lines* than prior work.

First, we present the general theory that we used in order to generate the character poses. Then, we present the interface created and the features available for the user. We finally present the results of our interface and discuss possible further improvements.

II. General theory

A. Skeleton

In order to assure the 3D character can be properly shaped from a sketch line (the LOA), we have to make sure the character is rigged with a *correct* skeleton.

A skeleton $\mathcal{S}$ is defined as a set of N joints

$$\mathcal{S} = \{s_0, s_1, ..., s_{N-1}\}$$

Each joint s_i has exactly one parent noted $p(s_i)$ and can have several children. The only joint allowed to have no parent is the *root* of the skeleton $\mathcal{S}$, noted s_r. To simplify, we declare that $p(s_r) = s_{-1}$. In the 3D scene, each joint s_i has its own position define by its translation $s_i^T = (s_i^x, s_i^y, s_i^z)$ and rotation s_i^R.

B. Skeleton segment

Once the skeleton is properly defined, we want to spatially match a subsection of this skeleton to a sketched line. But because the line carries few information compared to the entire skeleton, it's impossible to match the entire skeleton at once with only one sketch line. We must limit ourselves to certain types of subsection called *skeleton segments*, that the user will be able to select to be matched with the sketch lines.

A skeleton segment $L_\mathcal{S}$ relative to the skeleton $\mathcal{S}$ of length $k > 1$ is a subsection of joints respecting the conditions:

$$L_\mathcal{S} = \{j_0, j_1, ..., j_{k-1}\},\ j_i \in \mathcal{S} \tag{1}$$

$$\forall i \in [1, k-1]\ ,\ p(j_i) = j_{i-1}$$

j_0 is called the *root of the segment* $L_\mathcal{S}$, not to be confused with the root of the skeleton $\mathcal{S}$. It has no parent *in* $L_\mathcal{S}$ (but can have one in $\mathcal{S}$).

More practically, a skeleton segment for a 3D character can be an arm, a leg, or even the trunk (from the hips to the head). However, if the joint of the hips is the *root* of the skeleton, a segment from the left foot to the right foot passing by the hips is not a valid skeleton segment, because the hips has no parent in that segment (it has no parent at all). An example is provided in figure 2.

C. Sketch line

In order for the user to create Lines of Action, the code provided by the professor allows the user to draw *sketch lines*. A *sketch line* l is a set of K points in the 3D scene that are on a common 2D plan. They are drawn from point 0 to point $K-1$ such that

$$l = \{p_0, p_1, ..., p_{k-1}\}$$

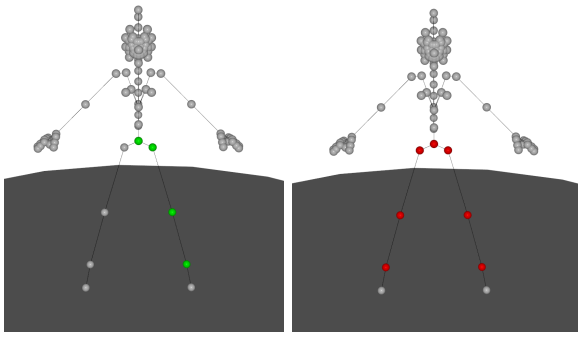

Fig. 2. A valid skeleton segment (left), and an incorrect one (right)

For convenience and future considerations, we provide each line l with curvilinear x-axis. Each point p_i is indexed by a curvilinear index y_i.

$$y_i = \frac{i}{K}$$

$$l = \{p_{y_0}, p_{y_1}, ..., p_{y_{K-1}}\}$$

III. Methods to match a LOA

A. Naive method: translation matching

With the theory proposed in section II, we can already think of a simple way to match a body line to a sketched line: *translation matching*.

The first translation matching and the simplest one is *near matching*. Each joint of the body line is matched with the nearest point on the sketched line. This method can be correct for LOA that are close to the body line but cannot be applied for more sophisticated LOAs.

The second translation matching is the *curvilinear matching*. In the case of a skeleton segment, $L_{\mathcal{S}}$, we provide the segment with a curvilinear x-axis similar to the one of the sketched line.

$$L_{\mathcal{S}} = \{j_{x_0}, ..., j_{x_{k-1}}\}$$

Provided that the sketch line l is drawn beginning from the root of the segment, we can then simply apply

$$\forall i \in [0, k-1],\ j^T_{x_i} = p_{y_{j_i}},\ j_i = \underset{o}{\operatorname{argmin}} |x_i - y_o|$$

Less formally, we match the joints of the segment with the points on the LOA with the closest curvilinear index.

These methods are very simple, fast and easy to implement. However, the present so much problems that they are not viable:

- They don't respect the geometry of the skeleton: the bones can be extended or compressed and the joints can be rotated in unnatural direction.
- They don't preserve the relative orientation of each joint. The joints are only translated and never rotated, making skinning complicated or impossible. (Appendix figure 3)

B. Prior work: Optimisation problem

The method presented in the paper on which this project is inspired is essentially based on an optimisation problem. Without going into the details, the constraints imposed are: relative position of the selected body parts to the LOA, the rotation locks of certain joints (in order to respect the geometry of the body), the importance of certain specific bones and the connectivity of the joints. The authors then compute the optimal position of each bones, taking into consideration every constraints and their coefficients. This method has several advantages:

- It perfectly respects the general geometry of the body (inextensible bones, rotation lock of certain joints like elbows and knees, bones cannot over overlap).
- The computation is fast (a few seconds for most of the LOA.

However, it also shows some limitations:

- Only a pre-defined set of body lines are available.
- The LOA should be approximately the length of the body line and not too far from it.
- For 1 LOA, several body poses are possibles. The user sometimes has to remove the ambiguity himself.
- it is hard to implement, and is parametrized for one specific skeleton.

C. Rotation matching

In order to solve most of the problems encountered with translation matching, this project proposes another alternative: *rotation matching*.

This method is a recursive method that rotates each joints in order to match the skeleton segment on the sketched line while preserving the length of the bones and the coherence of the joints allowing to do skinning after the matching.

The idea is the following: if we supposed that j_i is correctly matched with the point $p_{y_{j_i}}$ on the line l, we can match j_{i+1} while preserving the general rotation of the skeleton by rotating the rest of the skeleton segment $j_i, j_{i+1}, ..., j_{k-1}$ around the joint j_i without translating it. The rotation is chosen so j_{i+1} is correctly matched with $p_{y_{j_{i+1}}}$ After this step, j_{i+1} is correctly matched and become the new *pivot*.

To initialize the algorithm, we virtually match j_0, the root of the skeleton segment, by translating the LOA in order to match the root of the LOA (the first drawn point for the moment). Here is a pseudo-algorithm to describe this algorithm.

Algorithm 1 Rotation matching

Inputs: $L_{\mathcal{S}} = \{j_0, ..., j_{k-1}\}$, $l = \{p_{y_0}, p_{y_1}, ..., p_{y_{K-1}}\}$
Initialise: $l = l + \overrightarrow{p_{y_0} j_{x_0}}$
for Pivot i = 0,..., k-2 **do**
  Rotation $rot = Rot\_vectors(\overrightarrow{j_{x_i} j_{x_{i+1}}}, \overrightarrow{j_{x_i}, p_{y_{j_{i+1}}}})$
  for Children c = i,..., k-1 **do**
    $j^T_c = j^T_i + rot * (j^T_c - j^T_i)$
    $j^R_c = rot * j^R_c$
  end for
end for

This algorithm respects the length of the bones, their respective orientation (which allows to do skinning afterwards) and is very fast to compute. However, it doesn't restrict the rotation of the joints. As a consequence, it can create distortions that are not natural for a human body. Because this algorithm stays very general, it is up to the illustrator to ensure that drawn sketch lines more or less respect the articulations of the body. Also note that this method works correctly as long as the sketched line begins near the root of the segment and is more or less the same length.

For the moment, we have only spoke and studied the case of *skeleton segments* described in (1). It is restrictive when we want to match entire body lines with LOAs, but we will see in the next section that we can split a body line in different skeleton segments and have nice results.

IV. Implementation and Interface

A. Drawing on the Scene

The user is able to draw 2D sketch lines directly in the 3D scene. Because we are loosing 1 dimension, we have to project the 2D line in the 3D scene. That why we had to make certain assumptions according to the position and the orientation of the plane.

The central point of the scene always belongs to the plane. Then, we have to determine the normal to the plane. If the user choose *free drawing*, the plan will be orthogonal to the camera (which moves spherically). It is practical if the user wants to create an aerial pose for instance.

If the user choose the *Z-lock drawing*, the normal of the plane is collinear to plane of the ground. It is useful if the user wants to create poses where the character is standing up.

The user can draw multiple lines. However, only the last drawn line will be used as the LOA to be matched with.

B. Joint selection and Segment computation

In order to match the sketch line, the user must select the joints that should be rotated to match the line. The proposed interface allows the user to freely select the joints, in an arbitrary number. The next step is to compute the skeleton segments corresponding to the selected joints. A single selection can correspond to one or several skeleton joints.

First, we seek a *root* j_r in the selected joints (e.g. a joint whose parent is *not* selected). For each selected children j_c, with $p(j_c) = j_r$, the couple (j_r, j_c) is the beginning of a skeleton segment, because the j_r can be the root of several skeleton segments. We then recursively seek for maximal skeleton segments in the selected joints beginning by the couples (j_r, j_c). However, if we found a selected joint j_i that is *not* a root and has 2 children selected, we are forced to declare the selection incorrect. That is because we shouldn't allow T shapes in the selection, except is the center of the T is a root (then the root is the root of 3 skeleton segments). Once the skeleton segments having j_r as root are all computed, we check if all the selected joints have been assigned to a segment. If it's not the case, we seek another root and execute this recursive computation again

If the selection is declared valid, we obtain one or several skeleton segments that exactly correspond to the selected joints.

C. Skeleton segment rotation

Now that we have computed the skeleton segments, we can apply algorithm 1. However, because we eventually want to match *several* skeleton segments on a *single* sketch line (finally, the LOA), we have to clarify what part of the LOA corresponds to what selected skeleton segment.

In section III-C, we matched the root of the skeleton segments with the first drawn point of the sketch line. It's not possible any more. Assuming that the LOA is not drawn *too far* from the skeleton segment, we will take the closest point of the LOA to be matched with the root of the segment.

Now that the root is matched, we have to determine in which *direction* on the LOA should the segment be aligned. In fact, because the root could be matched with a point on the middle of the LOA (which is the case for the hips for instance), it is not completely clear if the skeleton segment should align on the first middle of the line or on the second middle of it. To determine the direction of the skeleton segment, there are several possibilities: scalar product (match with the part which is currently closer), length of the matched part of the LOA (similar length will be matched together), etc... It depends on the way the illustrator wants to create the poses.

D. Singular point: root of the skeleton

The last point to consider to be considered is how we treat the root of the skeleton. With the rotation matching, a joint is rotated around a direct or indirect parent. But the root of the skeleton has no direct or indirect parent.

One solution, proposed as the feature *Rotation lock* for the user, is to declare that the root of the skeleton will never be moved or rotated. It can be useful to create simple poses where the character moves his arms, legs, head and trunk. However, the character will always seem to be standing up, and some more sophisticated poses will look weird and unnatural.

Another solution, also featured for the user, is to match the orientation of the root with the local tangent of the LOA. If the root is matched with the point p_r, the local tangent is defined as the vector $\overrightarrow{p_{r-\epsilon}p_{r+\epsilon}}$ with a relative *little* ϵ. We then rotate the root of the skeleton so the *up* vector of this joint matches the local tangent.

E. Skinning

Because we took care of conserving the relative rotation and orientation of each joint relative to its children, we *can* apply skinning to the skeleton. In this project, we use Linear Blend Skinning which has some limitations (candy wrapping problem, unnatural curves at critical points). One improvement could be to implement Dual-Quaternion skinning, but it is not the purpose of this project.

F. Limitations

The *rotation matching* can be seen as an improvement of *translation matching* which is very naive. It allows the user to create credible poses, with some constraints and limitations.

- It doesn't prevent unnatural rotation of joints like the knees for instance.
- It doesn't prevent bone overlapping.
- In order for the matching to be correct, the LOA shouldn't be *too far* from the skeleton, especially for a long body line matching (from foot to hand for instance). An example of mismatching due to an ambiguous line is provided in appendix, figure 4.
- If the LOA doesn't have the same length as the selected body line, the matching will not be perfect (mainly due to the curvilinear indexing).
- It requires the user to select correct skeleton segments.

Despite these constraints, this methods remains very general, can be applied to any correct skeleton described in II-A and gives good results, provided that the user knows the notion of skeleton segment and draws natural LOAs.

V. Conclusion

This project presents a simple way to match a 3D with LOAs drawn in 2D. It uses *rotation matching* which rotates the joints to match the LOAs. This method is very fast (less than a second for each matching) and efficient, provided that the user respect certain rules in choosing the rotated joints and draws the LOA not too distant from the skeleton. It is intentionally very general and can be useful for any type of skeleton (we could think of a rigged tree for instance). Further work would eventually be to better determine the direction corresponding to each skeleton segment which is still a weakness of this method, but is inherent to the LOA: projecting a 3D skeleton on a 2D line brings a necessary ambiguity. Also, we could better determine p_{y_i} for each j_i of a segment. Relying only on the curvilinear index is a problem for a LOA that is longer or shorter than the segment.

Appendix

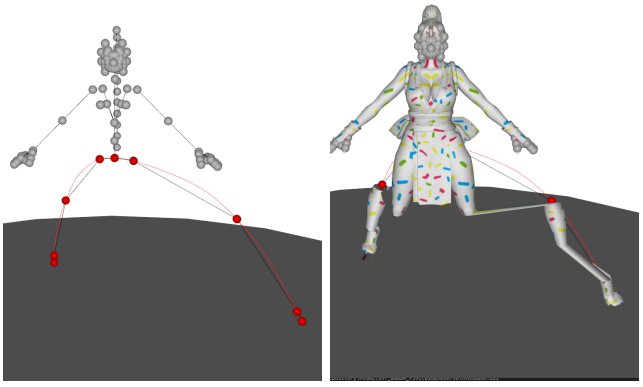

Fig. 3. *Translation matching* creating massive deformations of the skeleton (left) impossible to skin (right).

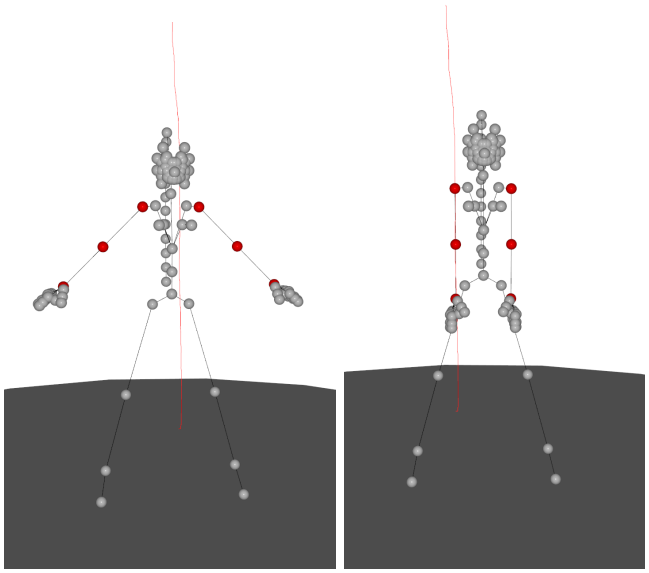

Fig. 4. An ambiguous line leading to a mismatching (matching in the wrong direction).