Puppeteering 2.5D Models

João Coutinho
Universidade Federal do ABC, Brazil
joao.coutinho@ufabc.edu.br

Bruno A. D. Marques
Departamento de Computação
Universidade Federal Fluminense, Brazil
brunodortamarques@gmail.com

João Paulo Gois
Universidade Federal do ABC, Brazil
joao.gois@ufabc.edu.br

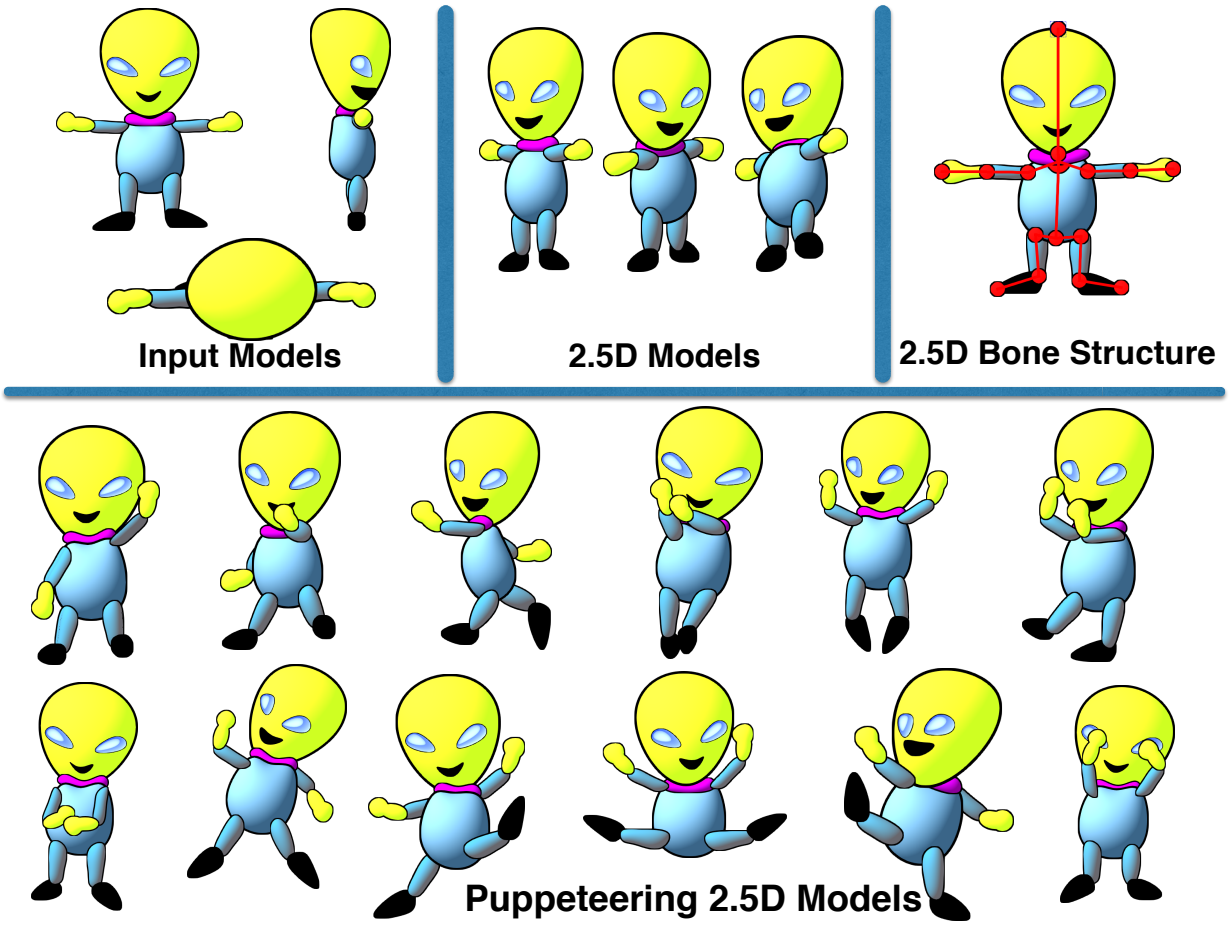

Fig. 1. Puppeteering 2.5D Models: [top row] input models required by the 2.5D Modeling techniques (left); new views generated by the 2.5D Modeling techniques (center); definition of our bone structure for driving the computation of new poses (right); [bottom row] poses generated with our puppeteering technique.

***Abstract*—A laborious task for animators is the redrawing of 2D models for each new required view or pose. As a consequence, several applications have been proposed to make this task easier. A successful approach is the Cartoon 2.5D Models. Its goal is the automatic computation of new views – by the simulation of 3D global rotation – from user-provided 2D models. However, previous work of 2.5D Models does not have support to calculate new poses efficiently, *i.e.*, the user redraws the input views again in the new pose. We present a novel approach that allows the user to produce both new views and new poses easily for the 2.5D Models, thus *puppeteering* the 2.5D Models. It makes use of a hierarchical bone structure that explores the methodology of the 2.5D Models, ensuring coherence between the bone structure and the model. The usability of the present approach is intuitive for users acquainted with previous 2.5D Modeling tools.**

***Keywords*-cartoon modeling; computer animation; vector-art drawing; geometric transformations.**

I. Introduction

The availability of applications for 2D and 3D modeling is no longer limited to the big companies of cinema and games. Indie artists – among the new beneficiaries of those tools – have revealed their talents by utilizing PCs, tablets, and web-based applications [1] to create 2D and 3D content.

Compared to 3D modeling, the 2D drawing provides a considerable artistic freedom for generating stylized cartoons. However, one significant difference between the creation of 2D and 3D models depends on the fact that all the views of a 3D model can be easily achieved by just rotating the virtual camera of the application. On the other hand, in the 2D drawing, for each point-of-view, the artist must redraw, at least partially, the model, leading to a laborious work. Although artists depend mainly on their artistic intuition to draw the additional point-of-views required, it is not uncommon the employment of automatic drawing tools to ease the task.

In this sense, the Cartoon 2.5D Modeling aims to simulate 3D rotations from a set of different views of 2D vector-art drawings of a cartoon [2]. In this technique, the user inputs a set of views of a 2D model – for instance, top, side, and front – to automatically compute new views through a 2D interpolation and an automatic depth estimate for the strokes of the model.

However, despite the support of 2.5D Models for the generation of new views of the model, the definition of a new pose, in general, requires substantial redrawing of the input models. In Fig. 1 we exemplify this issue and show how we address it. The top-left section presents the three input drawings used. The top-center section shows new views automatically produced by a 2.5D Modeling technique [3]. The top-right section illustrates the proposed bone structure for defining new poses. Finally, the bottom row shows both new views and poses generated with the proposed technique.

When the artist designs complex models with several articulations, a user-friendly interface to alleviate the computations of the new poses becomes necessary. The present work tackles this issue by reducing the number of the required new drawings for the 2.5D Models. We start from the axiom that some strokes of the drawing in a particular view are the same strokes in another view, in which they only differ from their global positions. For instance, in Fig. 2-(top row), the Bunny is (globally) rotated with open arms. In Fig. 2-(bottom row) the Bunny, in the front view, points its left arm forward. Notice that the drawings of the left arm in (b) and (d) are the same. Therefore, assuming that new poses of the models are built upon local rotations, our goal is to determine adequately such rotations in the 2.5D methodology. In other words, assuming the possible strokes established in other views, we efficiently propose a 2.5D approach that reuses these drawings for defining new poses, hence diminishing the user's labor.

Contribution: In the present work, we propose a novel approach for the generation of new poses for the 2.5D Models [2]. Through the manipulation of a hierarchical *bone system* that exploits the 2.5D methodology, users can determine new poses, within an interactive interface, hence reducing their efforts.

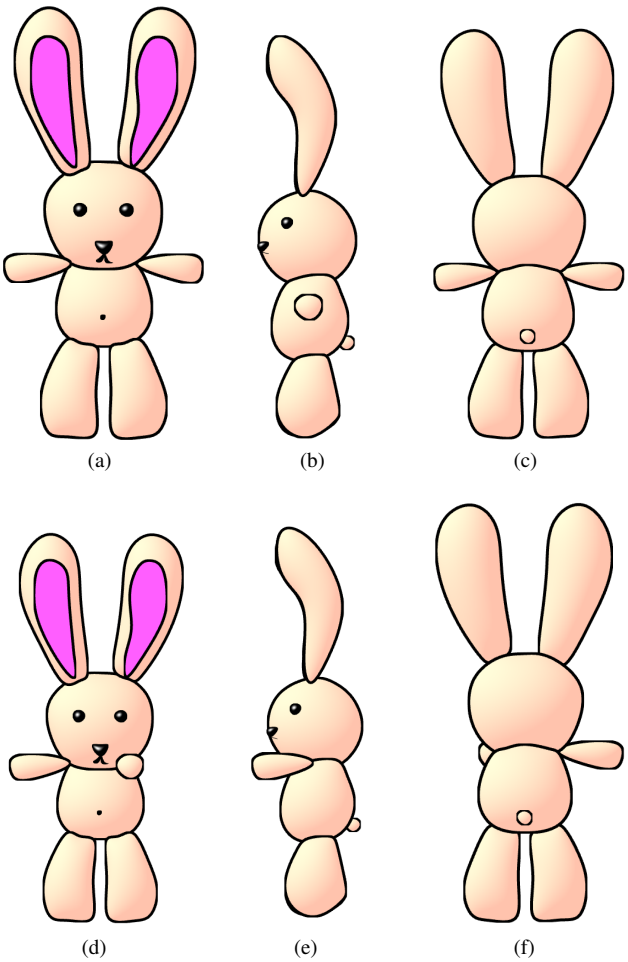

Fig. 2. Reducing the number of new strokes: on the top (a–c), a Bunny with open arms; on the bottom (d-f) its left arm points forward. Observe that the left arm of the Bunny on the side view on (b) is the same drawing of the left arm on the front view on (d). The only difference between both draws is the position of the arm.

II. Related work

Methods for improving the visual aspects of 2D drawing have received considerable attention. In particular, methods for texturing, lighting, and shading [4], [5], [6], [7], [8] brought well-studied effects in the 3D context to the 2D drawings. Concerning the animations of 2D drawings, the problems of inbetweening, generation of new poses, and physically-based simulations have also been broadly explored [9], [10], [11], [12], [13].

One important set of studies related to 2D applications relies on the generation of new views of a 2D drawing. Yeh and colleagues [12] approached the 3D simulation effects for 2D drawings. This work focused on performing twist, roll, and fold of the drawings based on optimization strategies. For most of the cases, the results are convincing. One of the limitations is the manipulation of concave drawings or drawings with large protrusions.

The Cartoon 2.5D Modeling, proposed by Rivers *et al.*, simulates 3D rotations and performs animations from a set of different views of a 2D cartoon [2]. This technique has received attention and extensions [14], [3]. In particular, Gois *et al*. extended the 2.5D Models to support interactive 3D visual effects. In this work, the authors incorporated several

interactive 3D effects in the context of 2.5D Model, *e.g.*, lighting, texturing and fur simulation [3]. Yet again those 2.5D Models [2], [3] present the same issue of adequately simulating rotation with drawings with concavity and large protrusions.

In the context of 2D drawing, the literature is full of approaches as well as numerous successful softwares, for the animation, edition, creation of new poses and deformation of drawings and pictures [15], [9], [10], [16], [17]. In particular, approaches that use hierarchical structures – or skeletons – have also been explored for articulating 2D drawings and photos. Those structures become one of the most used approaches for 3D animation [1]. In addition, they are present in popular 3D modeling tools [18]. Hierarchical structures have also been explored in 2D for shading and lighting of drawings [19], [20]. In the present work, we introduce a hierarchical approach for the creation of new poses for 2.5D Models.

III. Puppeteering 2.5D Models

Previous 2.5D Modeling techniques simulate 3D rotations from an input set of 2D drawings. Some of these input drawings are provided by the user while others are automatically generated by reflections and rotations of these user-provided inputs (Fig. 3). Each drawing links to a 2D point in a plane parametrized by the *pitch-yaw* angles [2], [3]. The set of these points is then triangulated (Delaunay) and the simulation of the 3D rotation is calculated by two ingredients: the Barycentric interpolation among the vertices of the triangles and an automatic computation of the depth order of the *strokes* that compose the model. It assumes that each stroke is present in each drawing of each input. It is also worth to note that strokes are composed of 2D polygonal curves. After this setup, the user obtains new views of the model by drag-and-drop operations either on the viewport of the screen application or on the pitch-yaw pane (Fig. 6).

Thus, we build upon the 2.5D Method a three-stage approach for computing new poses:

- Definition of the bones: in this stage the user provides the bone structures of the strokes that will be locally rotated;
- Computation of the local rotation of the strokes: this stage finds the pitch, yaw – and now – roll of the strokes, taking into account the hierarchy of the model and the current global view of the 2.5D Model. Computationally, this stage employs the obtained pitch and yaw to determine the correct shape of the strokes by the Barycentric interpolation and the roll to provide its 2D orientation;
- Displacement of the positions of the strokes: this stage is responsible for ensuring that the strokes are properly placed in the 2.5D Model while the local rotations are performed.

A. Definition of the bones

Our method must be as general as possible to allow the user to define new poses independently of the current view of the model. One can argue that, at least in some predetermined views, for instance, the front view, it could be relatively straightforward to compute new poses. However, artists analyze and edit their drawings using multiple views. Accordingly, we need to define a natural approach for calculating the new poses of the 2.5D Model independently of its current view. To this end, we propose a bone system that is similar to the tools for 3D modeling and animation, where the user defines *joint points* at the articulations of the model. Thus, in our approach, for each stroke, we define a *bone* with two joint points: the *initial joint*, which is responsible for being the reference of the rotations of the stroke and the *final joint*, which indicates the end of the stroke. Specifically, we consider the initial joint as the origin for the rotation of the final joint, *i.e.* the 2.5D rotations of the final joint orbits the initial joint.

However, our method differs, in several aspects, from the 3D skeletons techniques. The first is that our bone system is based on the 2.5D methodology. Computationally, we define our joint points as small circles that are considered as 2.5D strokes. This allows to exploit the 2.5D properties, *e.g.* the automatic definition of the bones in the other views computed by the 2.5D Models [3], to estimate the depth order of the strokes. In other words, considering the joint points as 2.5D strokes ensures the coherence of the rotations between the 2.5D Models and the proposed bone system.

Also, the use of strokes as the joint points of the bones ensures two important properties: the first implies that migrating to 3D space to define the bones is not necessary, what – up to our understanding – could be very hard to 2.5D Models. The second, based on the computational analysis of Gois *et al.* [3], the use of strokes for the joint points filled with simpler shadings, *e.g.* Flat or Phong shadings, does not significantly affect the time consuming of the application.

In Fig. 4 we detail the steps that the user performs to create a new pose. In (a), the initial model is represented with open arms. In (b) the user selects the first stroke where the bone will be defined. In (c) the user defines both the initial and final joint points (red circles) while the *bone root* (blue circle) in the center of the model (origin) is automatically provided. In (d) the user selects a second stroke and in (e) its initial and final joint points are shown. Observe that the final joint of the first stroke is the initial joint of the second stroke. In (f) the user rotates the strokes, defining thus a new pose. Finally, in (g-i) new views of the model are presented, considering the new user-defined pose.

B. Computation of the local rotation of the strokes

We must ensure that the local rotations of the strokes are appropriately combined as well as combined with the global rotation of the model. We present in this section the set of 3D rotations that provides the coherence with the 2.5D Modeling.

Gois *et al.* also used 3D rotations in their 2.5D Models [3]. In this work, the authors incorporated interactive shading effects into the 2.5D Models. They explored the graphics pipeline to infer relief and to simulate 3D rotations of the shading effects in real-time. The presented effects were widely variable, *e.g.* Gooch and cel shadings, environment mapping, fur simulation, hatching and texture animations. To this end,

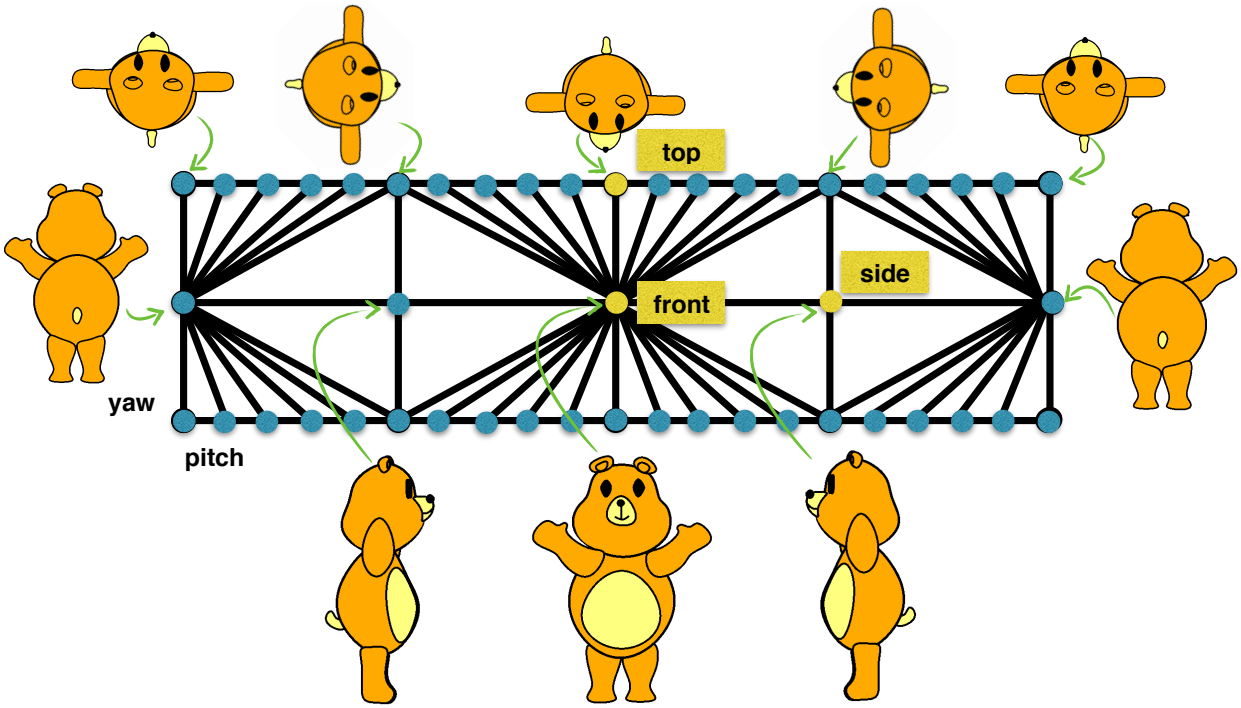

Fig. 3. Setup of the 2.5D Model: each input drawing is either associated with an user-defined drawing (at the yellow vertices) or automatically generated from the user inputs (at the blue vertices) of the (triangulated) pitch-yaw parameter plane.

the authors incorporated into the method of the 2.5D Model a *3D reference geometry* for each stroke of the model where shading effects were computed and appropriately transferred to the 2.5D Model in real-time. The method assigns the pitch-yaw coordinates to the 3D rotations of the reference geometry. This transfer guarantees the simulation of the 3D virtual trackball with shaded 2.5D Models.

Here, we follow a reverse path: given a 3D rotation of the stroke, we need to find its current pitch, yaw and roll parameters in order to obtain its shape and orientation. In the following, we show how to build this 3D rotation to adequately obtain these parameters.

We firstly need to compute the 3D local rotation of a stroke:

$$R_\ell = R_x(pitch) * R_y(yaw) * R_z(roll), \quad (1)$$

where R_ℓ denotes the 3D local rotation of the stroke and R_x, R_y and R_z are the rotations with respect the euclidean axis by the pitch, yaw and roll values interactively provided by the user.

Next, we need to combine the local rotation of the stroke with the rotations of the stroke hierarchically above it as well as the global rotation:

$$R_f = R_\ell * R_p * R_g, \quad (2)$$

where R_f is the final rotation of the stroke, R_p is the rotation of its parent, and R_g is the global rotation.

Finally, we compute from R_f the Euler angles (pitch, yaw, roll) of the stroke. Afterwards, the pitch and yaw values determine the shape of the stroke by the Barycentric interpolation while the roll provides its 2D orientation.

Previous computation ensures that the stroke will have the desirable shape, but does not ensure that the stroke will be adequately placed on the model. Now, we need to perform the last stage of our approach: the heuristics that guarantees the strokes of being properly placed on the model.

C. Displacement of the positions of the strokes

Previous 2.5D Models only support global rotations. However, we need to ensure that the strokes can be plausibly rotated locally. Specifically, we propose a heuristic (depicted in Fig. 5) to establish that local rotations are performed with respect the initial joint point. Fig. 5-(a) shows the initial model while (b) presents the bone definition of the stroke that will be locally rotated. The initial and final joint points of the bone are depicted in white and red colors, respectively. At this moment, the bounding box of the stroke is also illustrated. The user finds the desirable shape and orientation of the stroke by tuning the pitch, yaw and roll parameters (described in Sec. III-B) in the interface application. However, the current position of the new shape of the stroke, which was computed by the 2.5D global rotation, probably will be not in the desirable position (c). We thus need to move the stroke adequately to its initial joint point. The issue, at this moment, is to determine which part of the stroke must be the closest to the initial joint point. Notice that the bone, which is rotated considering its initial joint point, provides a hint to a coherent displacement of the stroke. To adequately displace the stroke, we take into account the hypothesis that the center of the bounding box of the stroke

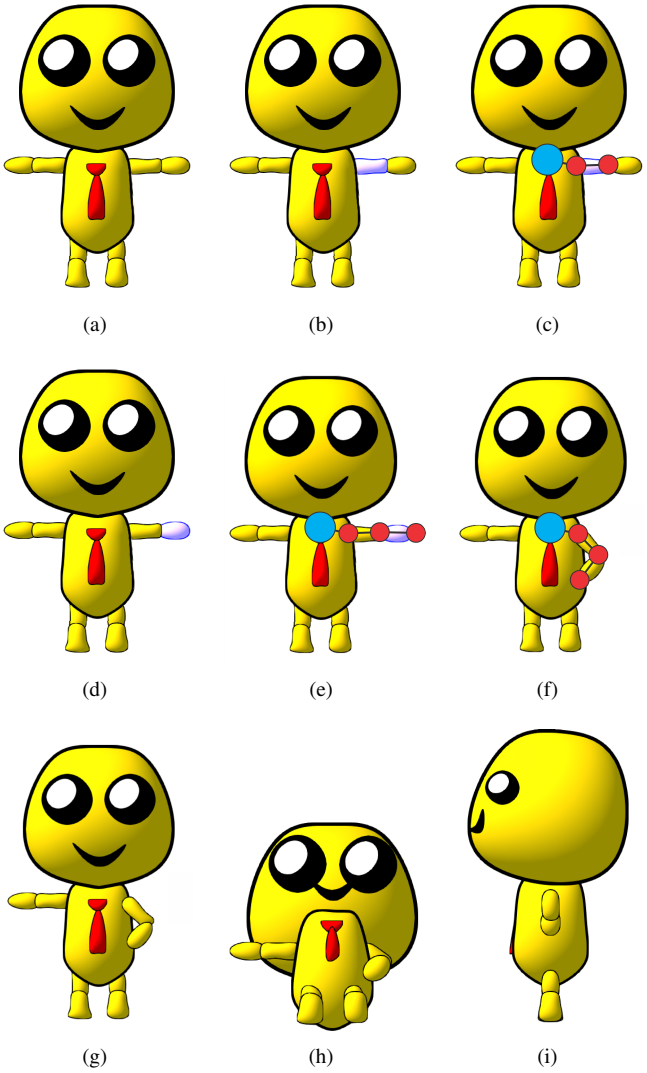

Fig. 4. Bone Interface: (a) the initial model; (b) the user selects the first stroke; (c) the user defines both the initial and final joint points (red circles). The blue circle (*bone root*) is the center of the model (origin) and is automatically provided; (d) the user selects a second stroke and (e) the initial and final joint points are computed to it; (f) the user rotates the strokes, defining thus a new pose; (g-i) new views of the model considering the new pose.

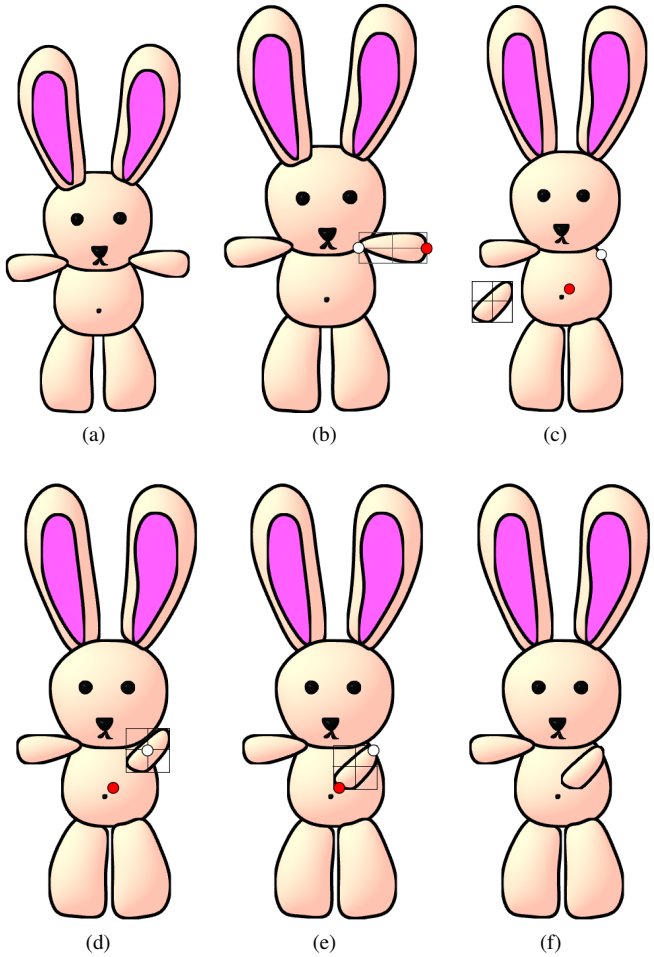

Fig. 5. Displacement of the positions of strokes: step-by-step of our heuristic that ensures the locally rotated strokes are suitably placed in the 2.5D Models. (a) initial model; (b) definition of the bone and bounding box of the stroke; (c) real position of the stroke when the 2.5D rotation is computed; (d)-(e) heuristics to displace coherently the stroke: first move the center of the bounding box to the initial joint point of the bone (white circle) and second move half of the diagonal of the bounding box along the bone; (f) the stroke will be plausibly placed according to its initial joint point.

is, in general, close to the center of the stroke. It follows that the translation of the stroke is performed in two steps: firstly, we move the center of the bounding box to the initial joint point (d); secondly, we translate the bounding box by half of its diagonal along the line segment of the bone (e). The use of this heuristics has ensured that the local rotation of the strokes remains coherent to the initial joint point (Fig. 5-(f)).

IV. Results

We developed our application with the C++ language, the Qt framework (version 5.6) and the shading language GLSL (version 4.2). All the results, even with shaded strokes on, achieved around 25 frames per second in a PC equipped an nVidia GTX 750Ti graphics card. We computed the geometrical rotations as well as the conversion to Euler angles, presented in Sec. III-B, using the Quaternions class provided by Qt. Singularities were expected when converting from quaternions to Euler angles at angles near the 90 degrees [21], [22]. We simply tackled this issue by considering a small perturbation close to these cases.

Figure 6 presents the interface of the application. Observe that on the top-right there are two view cubes that track the rotations: the left cube tracks the local rotation of the currently selected stroke, whereas the right cube tracks the global rotation. On the bottom, it can be observed the selector for the local and the global rotations as well as the pane for the pitch-yaw and the slider for the (local) roll rotation.

Fig. 7 presents the analysis of the method when requiring a complex bone structure. In (a-c) it is shown the input model and in (d) a rotated view at the original pose. In (e) the bones are defined. Observe that it incorporates not only the limbs but also the head of the Ant. In (f) the model was rotated, and

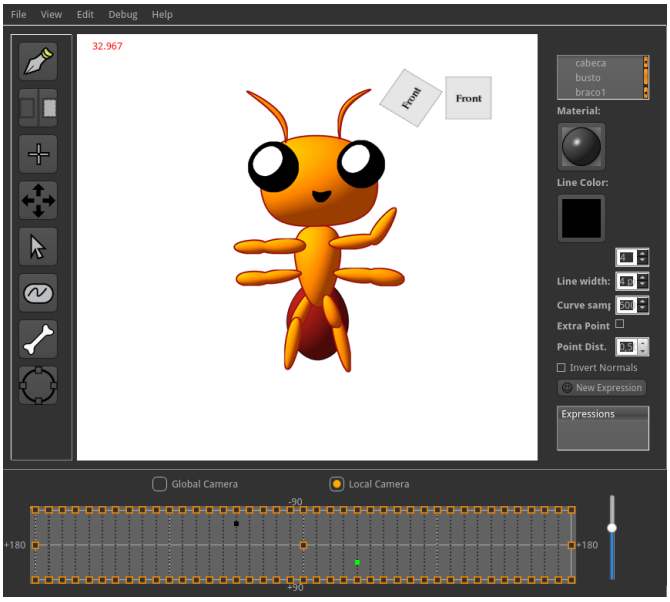

Fig. 6. The interface of the application: the global rotation is obtained by the drag-and-drop at the viewport widget (white window) or by the drag-and-drop at the pitch-yaw pane, the rectangle at the bottom. Local rotations are computed through the navigation at the pitch-yaw pane and by the roll slider (right side of the pitch-yaw pane).

its bones were manipulated. Other views of the model with its bones are presented in (g-h). From (i) to (p) several poses are performed.

In addition, our bone structure can be used to rotate hierarchically drawn strokes. For instance, the head of the models, in Fig. 7 and Fig. 8 have inside them other strokes, *e.g.* eyes and mouths. So with our technique, rotating a bone defined in the head of the model will also rotate coherently all of its child strokes (eyes, mouth, etc).

In Fig. 9, we show global rotations of the Bunny in both poses presented in Fig. 2: in (a-b) we introduce the model with open arms, whereas in (c-d) it points its left arm forward. In Fig. 10, we briefly illustrate the displacement of a stroke. In (a) we present the initial model with a bone on the (green) arm. In (b) we give the global rotation of the model while in (c) we rotate only the arm. Notice that the stroke of the arm in Fig. 10-(b-c) not only is the same but it is also in the same place on the plane. In (d) we show the final position of the stroke, after applying our heuristic of displacement of strokes.

A. Limitations and Future Work

Despite the recent improvements on 2.5D Models, there are important issues to be studied. The first is related to the bone structure. It is built upon the 2.5D methodology, *i.e.*, the joint points are manipulated as 2.5D strokes. Consequently, modifications may be required at the bone for each user input. Obviously even with the local modifications of the bone for each user-provided pose, the great benefit of defining new poses still remains. However, this fact does not prohibit the exploration, in a future work, for an approach that better fits the bones in all input views automatically.

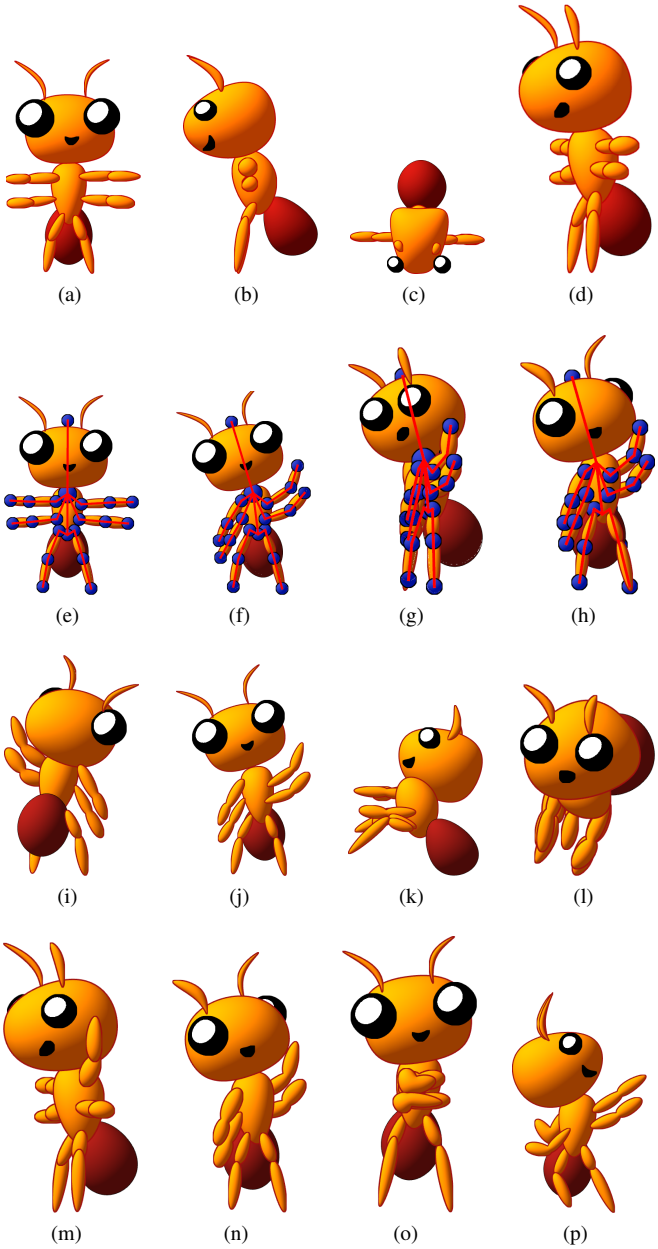

Fig. 7. Model with a bone structure with multiple joints: (a-c) user inputs; (d) rotated model at the initial pose; (e-h) definition of the bones and computation of new poses and views; (i-p) several views of different poses.

The current bone system is built supposing the space parameterization suggested by Gois *et al.* [3], in which yaw is continuously computed from -180 to 180 degrees. On the other hand, in the work by Rivers and colleagues [2], when yaw is zero, the strokes are reflected. In a future work we aim to consider this property of the Rivers approach into the bone system.

It is also important to provide some comments about the heuristics for displacement of the stroke. One could argue that more sophisticated methods could be employed, for instance, methods that explore an optimization technique or principal

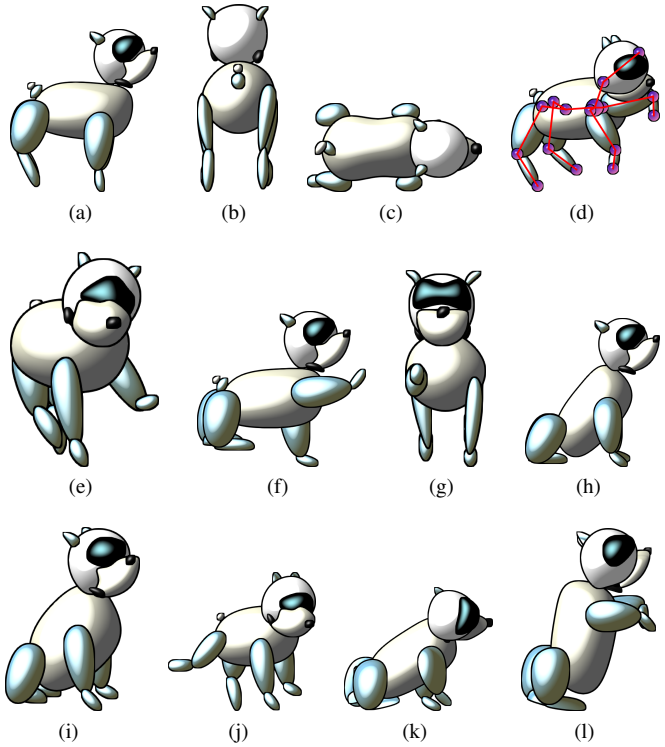

Fig. 8. Robot Dog Model: (a-c) views at the original pose; (d) a view with the bone system; (e-l) new views and poses of the model.

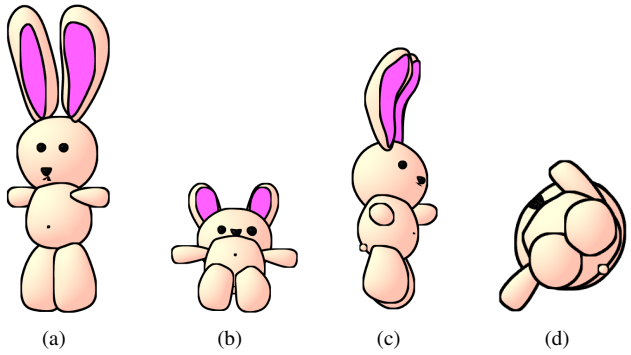

Fig. 9. Global rotation of the Bunny in two distinct poses: (a-b) rotations of the Bunny represented with open arms; (c-d) global rotations of the Bunny with its left arm pointing forward.

component analysis. However, up to this moment, the achieved results are satisfactory and the computational time is not compromised.

There is a limitation related to the user interface. Currently, we control the local rotations of the bones by a pane (pitch and yaw) and a slider (roll). However, for arbitrary views and poses, this kind of interface can become nonintuitive. In a future work, we shall to develop an approach to edit the bones directly in the view pane.

Another important issue is related to the interpolation of sharp and highly concave strokes. This issue can be addressed for some shapes, as suggested by Rivers [2], by

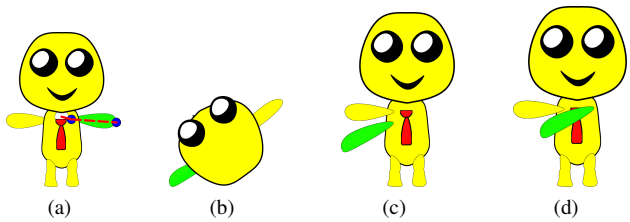

Fig. 10. Displacement of the strokes: (a) initial stroke with the bone in the arm; (b) example of global rotation of the model; (c) local rotation of the arm. Notice that the shape and position of the green arm in (b) and (c) are the same; (d) the final position of the stroke after applying our heuristic of displacement of strokes.

the decomposition of the stroke into mostly convex strokes followed by a Boolean union. Thus, following such suggestion, we implemented a technique to guide the decomposition of the stroke. Our approach is based on the Hertel-Mehlhorn algorithm [23] for triangulation of polygons. From the vertices of the triangulation of the polygon, our method computes a set of curves (ellipses in our tests) overlapping the original stroke. After this step, the user deforms the curves to better fit the original stroke. In Fig. 11-(a-c), the user adjusts the curves at the side, front, and top views. In (d) we present a rotated view of the decomposed Santa Claus' hat. In Fig. 11-(e-h) we illustrate the result of the Boolean union of the strokes.

However, this technique presents a limitation related to the computation of the depth order for complex shapes. At this moment, the depth order is a single value to the stroke. In complex geometries, the current versions of the 2.5D Models experienced difficulties to compute the order of the strokes adequately.

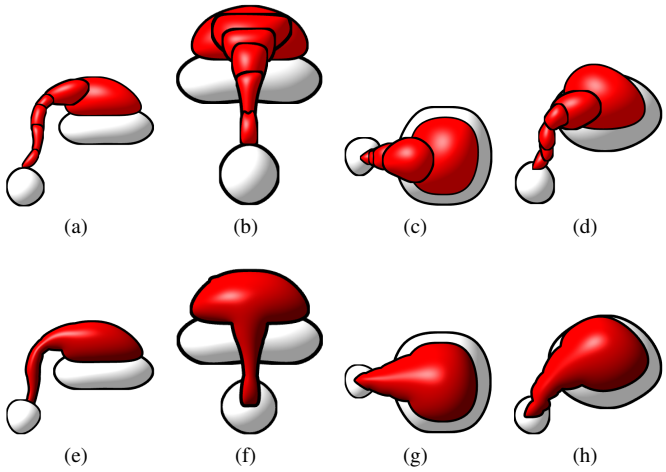

Fig. 11. The method of decomposition of highly concave strokes employed to the Santa Claus' hat: (a-c) decomposed user input (side, front and top views); (d) a rotated view of the model. (e-h) the Boolean operation of the curves that decomposed the model.

A further study related to the problem of complex shapes is the investigation of new methods of interpolation. Currently, we have been using the Barycentric interpolation, motivated by its simplicity and computational efficiency. However, it has

been shown, not only in this work but also in the previous ones [3], [2] that such interpolation is inefficient to handle complex shapes. Thus, it is apparent that we must investigate new methods of interpolation for 2.5D Models. One possibility is the multi-scale geometry interpolation [24].

V. Conclusion

The 2.5D Models [3], [2] simulate 3D global rotation by a 2D interpolation among provided views of the 2D model and an automatic depth-order computation of the strokes of the model.

In this work, we provided a step forward in making 2.5D Models a more attractive approach for the modeling and animation of 2D drawings by incorporating a user-friendly and interactive technique to determine new poses: puppeteering. The usability of the present approach is intuitive for users familiar with previous 2.5D Modeling tools.

The goal of our technique is to shorten the artists' redrawing work. Specifically, instead of redrawing the strokes for the new required poses, our method allows the users to locally rotate the strokes, *i.e.* perform a local simulation of the 3D rotation of the strokes. The local rotation of the strokes is done by our method in three steps: the formation of a bone structure, the computation of the pitch, yaw and roll parameters of the strokes and the displacement of the position of the strokes. The first is related to the definition of a hierarchical structure – in a similar way to 3D animation tools – while the other two are linked to the computation of the coherent rotation of the strokes within the 2.5D methodology. All in all, the presented results affirm the flexibility and the potential of puppeteering 2.5D Models.

Acknowledgment

The authors thank to São Paulo Research Foundation – FAPESP (proc. 2014/11067-1), the National Council of Technological and Scientific Development (CNPq), and Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) for the financial support of this work.

References

[1] A. Jacobson, "Breathing life into shapes," *IEEE Computer Graphics and Applications*, vol. 35, no. 5, pp. 92–100, Sept 2015.

[2] A. Rivers, T. Igarashi, and F. Durand, "2.5d cartoon models," *ACM Trans. Graph.*, vol. 29, no. 4, pp. 59:1–59:7, Jul. 2010. [Online]. Available: http://doi.acm.org/10.1145/1778765.1778796

[3] J. P. Gois, B. A. D. Marques, and H. C. Batagelo, "Interactive shading of 2.5d models," in *Proceedings of the 41st Graphics Interface Conference*, ser. GI '15. Toronto, Ont., Canada, Canada: Canadian Information Processing Society, 2015, pp. 89–96. [Online]. Available: http://dl.acm.org/citation.cfm?id=2788890.2788907

[4] R. Prévost, W. Jarosz, and O. Sorkine-Hornung, "A vectorial framework for ray traced diffusion curves," *Computer Graphics Forum*, vol. 34, no. 1, pp. 253–264, 2015.

[5] D. Sýkora, L. Kavan, M. Čadík, O. Jamriška, A. Jacobson, B. Whited, M. Simmons, and O. Sorkine-Hornung, "Ink-and-ray: Bas-relief meshes for adding global illumination effects to hand-drawn characters," *ACM Transaction on Graphics*, vol. 33, no. 2, p. 16, 2014.

[6] J. Lopez-Moreno, S. Popov, A. Bousseau, M. Agrawala, and G. Drettakis, "Depicting stylized materials with vector shade trees," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 118:1–118:10, Jul. 2013. [Online]. Available: http://doi.acm.org/10.1145/2461912.2461972

[7] M. Finch, J. Snyder, and H. Hoppe, "Freeform vector graphics with controlled thin-plate splines," *ACM Trans. Graph.*, vol. 30, no. 6, pp. 166:1–166:10, Dec. 2011. [Online]. Available: http://doi.acm.org/10.1145/2070781.2024200

[8] D. Sýkora, M. Ben-Chen, M. Čadík, B. Whited, and M. Simmons, "Textoons: Practical texture mapping for hand-drawn cartoon animations," in *Proceedings of International Symposium on Non-photorealistic Animation and Rendering*, 2011, pp. 75–83.

[9] A. Jacobson, I. Baran, J. Popović, and O. Sorkine, "Bounded biharmonic weights for real-time deformation," *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH)*, vol. 30, no. 4, pp. 78:1–78:8, 2011.

[10] A. Jacobson, Z. Deng, L. Kavan, and J. P. Lewis, "Skinning: Real-time shape deformation," in *ACM SIGGRAPH 2014 Courses*, ser. SIGGRAPH '14. New York, NY, USA: ACM, 2014, pp. 24:1–24:1. [Online]. Available: http://doi.acm.org/10.1145/2614028.2615427

[11] B. Jones, J. Popovic, J. McCann, W. Li, and A. Bargteil, "Dynamic sprites," in *Proceedings of Motion on Games*, ser. MIG '13. New York, NY, USA: ACM, 2013, pp. 17:39–17:46. [Online]. Available: http://doi.acm.org/10.1145/2522628.2522631

[12] C.-K. Yeh, P. Song, P.-Y. Lin, C.-W. Fu, C.-H. Lin, and T.-Y. Lee, "Double-sided 2.5d graphics," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 2, pp. 225–235, 2013.

[13] F. Di Fiore, P. Schaeken, K. Elens, and F. Van Reeth, "Automatic in-betweening in computer assisted animation by exploiting 2.5d modelling techniques," in *Proceedings of the Fourteenth Conference on Computer Animation*. IEEE, 2001, pp. 192–200.

[14] F. An, X. Cai, and A. Sowmya, "Perceptual evaluation of automatic 2.5d cartoon modelling," in *Proceedings of the 12th Pacific Rim conference on Knowledge Management and Acquisition for Intelligent Systems*, ser. PKAW'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 28–42. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-32541-0_3

[15] "Spine: 2d animations for games," http://esotericsoftware.com/, accessed: 2015-11-24.

[16] R. H. Kazi, F. Chevalier, T. Grossman, S. Zhao, and G. Fitzmaurice, "Draco: Bringing life to illustrations with kinetic textures," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '14. New York, NY, USA: ACM, 2014, pp. 351–360. [Online]. Available: http://doi.acm.org/10.1145/2556288.2556987

[17] A. Jacobson, T. Weinkauf, and O. Sorkine, "Smooth shape-aware functions with controlled extrema," *Computer Graphics Forum (proceedings of EUROGRAPHICS/ACM SIGGRAPH Symposium on Geometry Processing)*, vol. 31, no. 5, pp. 1577–1586, 2012.

[18] C. Totten, *Game Character Creation with Blender and Unity*, ser. EBL-Schweitzer. John Wiley & Sons, 2012. [Online]. Available: https://books.google.com.br/books?id=DICrcWznTmwC

[19] R. Nascimento, F. Queiroz, A. Rocha, T. I. Ren, V. Mello, and A. Peixoto, "Colorization and illumination of 2d animations based on a region-tree representation," in *24th SIBGRAPI Conference on Graphics, Patterns and Images*. Los Alamitos, CA, USA: IEEE Computer Society, 2011, pp. 9–16.

[20] D. Sýkora, D. Sedlacek, S. Jinchao, J. Dingliana, and S. Collins, "Adding depth to cartoons using sparse depth (in)equalities," *Computer Graphics Forum*, vol. 29, no. 2, pp. 615–623, 2010. [Online]. Available: http://dx.doi.org/10.1111/j.1467-8659.2009.01631.x

[21] J. Lee, "Representing Rotations and Orientations in Geometric Computing," *IEEE Computer Graphics and Applications*, vol. 28, no. 2, pp. 75–83, March 2008.

[22] J. Vince, *Rotation Transforms for Computer Graphics*. Springer, 2011.

[23] S. Hertel and K. Mehlhorn, "Fast triangulation of the plane with respect to simple polygons," *International Conference on Foundations of Computation Theory – Information and Control*, vol. 64, no. 1, pp. 52 – 76, 1985. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0019995885800449

[24] T. Winkler, J. Drieseberg, M. Alexa, and K. Hormann, "Multi-scale geometry interpolation," *Computer Graphics Forum*, vol. 29, no. 2, pp. 309–318, 2010. [Online]. Available: http://dx.doi.org/10.1111/j.1467-8659.2009.01600.x