3A2A: A Character Animation Pipeline for 3D-Assisted 2D-Animation

Oscar Dadfar[1] and Nancy Pollard[1]

Carnegie Mellon University, Pittsburgh PA 15213, USA

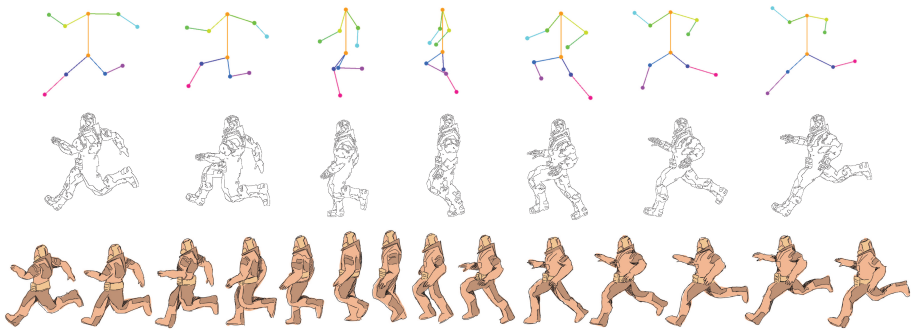

Fig. 1. Our program computes the joint angles of a set of key body joints to pose any 3D model. The posed model is toon-shaded and sketched over to help give a hand-drawn animation. Bottom row of frames were sketched and colored using the middle row reference keyframes.

Abstract. 2D hand-drawn character animation requires a substantial amount of time and drawing experience, turning away many interested in the field based on their lack of drawing ability. In an effort to make this type of animation more accessible to new creators without drawing experience, we provide a pipeline that creates character keyframe drawing references from stick figures. Our 2D interface allows users to draw and edit stick figures in 3D space that are then used to pose and rasterize any 3D character model in order to get keyframe reference images. We give these reference images a hand-drawn effect by applying a sketch-like filter to a toon-shaded output and compose it with a blotchy shading effect. We evaluate our work on multiple animation cycles and various models, demonstrating the program's ease-of-use for individuals with or without drawing experience. This strategy helps non-artists create 2D hand-drawn human character animations by reducing the entire hand-drawn character pipeline to simply drawing stick figures. We believe that if anyone can draw stick figures, then anyone can animate.

Keywords: Human-Centered Computing · Graphical User Interfaces · Fine Arts.

1 Introduction

Since Disney's release of Snow White and the Seven Dwafts in 1937, traditional hand-drawn animation gained momentum as one of the most entertaining and inspiring mediums of creativity and storytelling of its time. Disney followed up this success with countless other hand-drawn films in the next few decades, asserting the capabilities traditional animation had in conveying narratives and capturing the audience's attention. A few decades later, many animation studios have yet to put out any traditional animation films, with Disney having no plans for releasing any 2D animation films in the future [30]. The cost and skill associated with drawing out every frame, a practice originally done on cel-sheet paper due to the lack of technology, was too expensive compared to its computer-generated counterpart that requires no drawing experience. As easier 3D tools became more widely accessible, more novice content creators would avoid the art of hand-drawn animations in favor of 3D animation, citing that they do not have the necessary drawing skills to create traditional animations [1].

A few notable qualities make hand-drawn animation more favorable compared to its 3D counterpart. Traditional animation has a human-like touch, where every stroke is carefully thought out rather than interpolated. Characters are not limited to human constraints and can bend and rotate limbs in any fashion. Yet in order to create such character animations, individuals need to know how to draw characters accurately (per frame) and consistently (across frames), putting forth a high artistic barrier for those interested in the field.

In order to alleviate the high requirements of 2D animation, we developed an animation pipeline that does not require any artistic ability from the user when creating hand-drawn animations. Our pipeline is the first of its kind to use 3D models to help go from stick figures to a 2D character animation. We can model realistic body turns and proper object perspectives that are otherwise difficult for 2D character animators to draw, giving beginner animators an easy way to generate and stylize characters in a toon-like fashion. We demonstrate the program's ease-of-use in creating several different hand-drawn animation cycles applied to various character models and evaluate the system on a number of individuals with and without traditional artistic backgrounds. The final pipeline allows anyone to, regardless of artistic ability, create 2D hand-drawn animations, encouraging individuals without artistic backgrounds to hand-drawn animations despite their lack of drawing experience.

2 Related Works

Making 2D animation easier has been a persistent challenge, pushing creators to spend more time dreaming and less time drawing. Disney Research's Mender was developed as a vector/raster hybrid interface for computing in-betweens quickly [35] and Adobe's Flash Professional software integrated a similar pen tool to vectorize brush strokes, but both approaches suffered from the same artifacts when trying to adjust to complex geometries like face and body turns. Researchers

at Autodesk tried expanding on the vector graphics domain of 2D animation by incorporating "Motion Amplifiers" that apply a set of transformations on a vector to model the motions found in the 12 Principles of Animation [17,32], although the resulting vector still suffers from its inability to exhibit 3D rotations. 2D sketches can also be used to to pose 2D vectored characters [28], but this is limited in viewpoint as we cannot consider 3D rotations.

We can edit 3D motions to obey the 12 Principles by computing a pass over the kinematics data and applying filters or convolutions to exaggerate [36,18] or squash and stretch [19] these 3D motions, making them look more 2D like. While these filters are a step in the right direction, the viewer can often see that the results were generated by a filter and not by hand animation, thus leading the viewer to loose interest in the work. An alternative to this is to use inverse kinematics on the joint angles [21,20] or bone segments [3] to pose the characters in the exaggerated ways we see in 2D animation. One solver in particular attempts to use 2D sketches in the model's 3D environment as a basis for 3D character posing using inverse kinematics [24], making 3D character posing and animating a 2D constrained problem, given that all target movements lie along the same plane. We use this mechanism of editing 3D data along the 2D camera plane as a part of the backbone of our drawing interface, as it provides a way to pose 3D characters from 2D-constrained sketches.

The inverse problem of 2D-assisted 3D animation has been done from rough 2D skeletons [9,7,12] and from detailed artist drawings [15]. Converting 2D sketches to 3D posing data is an under-constrained problem, leading to many possible poses per 2D sketch. Rather than querying the user to pick the best pose from a sketch or searching for similar 3D poses from a database [23,31], we attempt to properly constrain the problem by drawing 2D stick figures in 3D space. These 3D interfaces for 2D sketches have been used previously for iterative character re-posing [25,29] and to define spatial trajectories of character motions [33,10,11]. These same 3D drawing environments can also be used for creating 3D human character models [26] or various other 3D models [5,38,37] from 2D sketches. They provide an interface for drawing within 3D space that leads to a fully-constrained conversion between sketch and 3D posing, and rotations in these 3D sketch spaces can be achieved by easily computing the quaternion rotations [16] between joints. We model a similar 3D-based 2D sketch environment that allows users to draw in 2D but drag and edit strokes in 3D space in order to achieve the same fully-constrained conversion that can be used for one-shot posing based on computing the quaternions of these joints.

Toon Shading, also known as Cel Shading, is a non-photorealistic rendering style that helps give 3D animation a 2D cartoon-like look [13,4] by thresholding on the shader's color value to give a rendering high-contrast shading effects that we see in 2D imaging and paintings. While an individual frame of a Cel Shader may look to be drawn and shaded by hand, the resulting sequence of frames still uses solid rigid outlining that keeps it from looking cartoon-like. Hand-drawn shading is an alternative that integrates sketch-like strokes around the contours of an image to give it a more natural, hand-drawn feel [6,27,34,22].

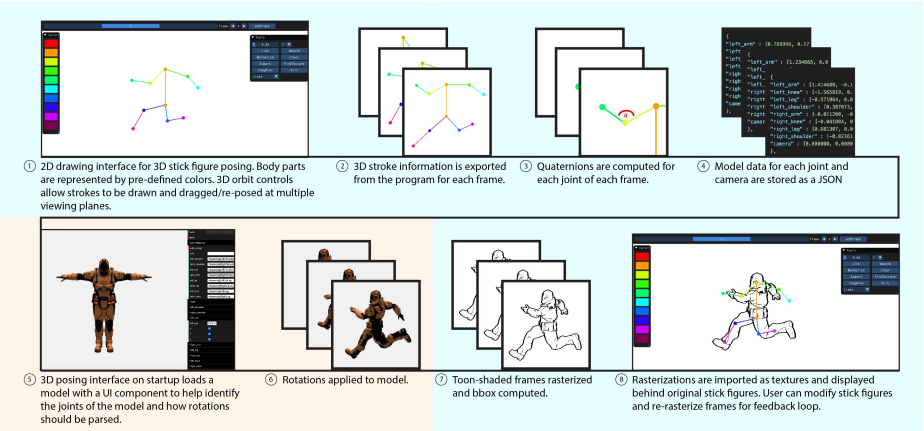

Fig. 2. Block diagram of the interface. Blue regions refer to the sketch interface while orange regions refer to the posing interface. The joint rotations of the sketch in 3D space are used to pose a user-imported 3D model. These model frames are toon-shaded and cropped before being loaded behind the original stick figure, providing a live feedback-loop.

When rasterizing 3D-posed reference frames, we can apply these hand-drawn shading styles to the contours of our exported 3D character animation sequence to help make it look more like a 2D sketch while also introducing the stochasticity and roughness of human sketches that we see between frames.

3 Approach

Figure 2 shows a high-level description of the animation interface segmented into two parts: 2D and 3D. The 2D drawing interface is the user side that is responsible for gathering the joint vectors that comprise the stick figure. This data is sent to the 3D interface where these rotations are computed and applied to a rigged character model that is toon shaded and rasterized before being sent back to the 2D interface. The user has a chance to make any corrections and re-pose any frames they are not happy with. Once done with their animation cycle, users can sketch their character animation using these frames as keyframe reference or can use the hand-drawn program we discuss in the paper to algorithmically give the character a sketch-like feel.

2D Interface. We construct a 2D interface that can sketch multi-color strokes in 3D space. The different stroke colors refer to different body parts, including the back, left-leg, left-ankle, right-leg, right-ankle, left-shoulder, left-arm, right-shoulder, and right-arm. The user can draw on the plane parallel to the current camera's view and drag endpoints of existing stokes parallel to the same camera plane. We install an orbital camera to view the stick figure from different angles

Fig. 3. 3D posing interface with configurable parameters. The user specifies the joint mappings between the joints in the 2D setting with the joint names for the specific model.

and to help redraw/drag existing strokes. This mechanism allows users to draw in the native xy-plane that they are used to drawing in 2D on pen and paper while still allowing the user to modify depths of strokes in the z-pane. If the user draws a stroke that already exists on the current frame, the previously existing stroke is cleared, allowing for only one instance of each body part per frame. The interface's provided timeline coupled with the onion skinning tool allows for drawing in-betweens with ease after drawing the main keyframes.

The user can link together joints, where the program attaches strokes if they share a common joint. Some common joints include the shoulder connected by the back and left/right-shoulder or the knee connected by the left/right-leg and left/right ankle. This way, the user does not have to make sure strokes connect when drawing them, but can auto-connect joints afterwards.

The rotational data of the joints are used to pose and display a toon-shaded rasterization of the user-selected model behind the user's stick figure for each frame. This allows the user to see in full perspective how their posing looks and allows them to edit their stick figures in realtime.

3D Interface. Each joint maps to a base vector and end vector, where we seek to obtain the rotation transforming a joint along the base vector to the end vector. For each joint, we fetch the initial base vector of the model before any transformations are applied. This helps us compute the quaternion rotation from the base's rest vector to the current base vector [14,8]. We apply the inverse of this quaternion to the base and end vectors for the current joint, allowing us to reset the base vector to its rest position before computing the current joint's quaternion rotation from base to end in its initial space. This is because we will be applying the quaternion rotation to each joint from it's rest position, so we want to compute quaternions when the base vector is aligned with it's rest position.

Each quaternion can be broken into a pitch, yaw, and roll. Because we do not define orientations for our joints, the roll is left undefined when computing

Algorithm 1 Joint Rotations

```
for (curJoint, parentJoint) in jointMap do
    Vec3 base = normalize( curJoint.base )
    Vec3 end = normalize( curJoint.end )
    Vec3 initBase = normalize( modelInit[curJoint.id].base )
    //reset base/end to init orientation
    Quat q0 = quaternionFromUnitVec( base, initBase );
    Vec3 baseR = base.rotateByQuaternion( q0.inv() );
    Vec3 endR = end.rotateByQuaternion( q0.inv() );
    Quat q = quaternionFromUnitVec( endR, baseR );
    curJoint.rotateQuaternion( q );
    //remove rollAxis rotation from parent
    Vec3 current = curJoint.rollAxis;
    Vec3 target = end;
    Vec3 axis = target.cross( current );
    float angle = acos( target.dot( current ));
    parentJoint.rotateAxisAligned( parentJoint.rollAxis, angle );
end for
```

the quaternion. This can lead to random rotations that affect the child joint, offsetting target rotations by some amount along the roll of the parent joint. To fix this, we remove the roll component from each joint's parent quaternion. We compute the current axis as the child joint's roll axis and the target axis as the world-space end vector of the child joint and attempt to minimize the dot product between these two vectors. We can compute the rotation angle between the current and target axis, and apply an axis-aligned rotation onto the parent joint's roll axis for the computed rotation angle. This effectively removes the roll component of the parent's quaternion and correctly aligns the child joint to its target location.

The user can import any rigged 3D model to pose using their stick-figure joint rotations. Since the model joint names and rotations differ from model to model, our interface traverses the model's scene graph and provides a list of all joint names to the user. The user can then configure which local joints map to which model joints. Because these are model-specific parameters, they only need to be computed once per model as shown in Figure 3.

When rasterizing the frames, we use toon-shading where we extrude the geometry of back-facing normals that are shaded black to give it an outline effect. We interpolate the vertex normals of the model along the faces to create smooth normals that we can threshold on in order to compute hard shadows for our figure. We combine this with a movable directional light to give it a flat-shading effect. For each frame, we pose the model using the quaternions computed per joint and then rasterize the frames and save them to the device.

We can search for the rasterized frames on the device and compute the bounding box of the toon-shaded image in order to crop the image and display it as a texture behind the bounding box of the stick figure. This allows the user to see in real-time how the frame posing aligns with their stick-figure and allows them

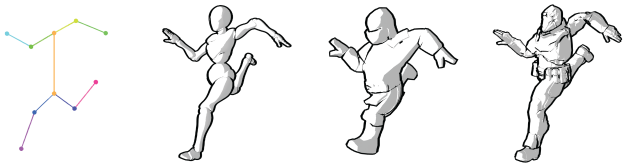

Fig. 4. Applying the same joint rotations to multiple toon-shaded models

to make quick edits to body parts and re-rasterize frames if they need to. This iterative process helps users experiment and refine their animations cycles.

Sketch Effect We can generate a hand-drawn feeling along the contours from the backface shading of each frame by applying rough sketch strokes to these contours. Each sketch is comprised of a bezier curve of several nearby points. We generate a random point along the contour of the frame and search for the next point within a given radius of the previous point using uniform rejection sampling, making sure the next point falls within the contour of the frame as well. To help remove extraneous strokes that cut across non-outline regions, we also verify the midpoint between every two points also lies along the contours. We can generate N random strokes in the bounding box of the frame or generate N/C strokes per cell in a $C \times C$ uniform grid to promote the uniformity of strokes distributed.

We can separate the shading components from our toon-shaded models and run a thresholded convolution filter that bins colors into either a light or a shaded color depending on which color the average of the convolution is closer to per pixel. This provides blotchier, smoother details that replicate hand-painted effects for shading. We can apply both light and dark shadows by duplicating the blotchy shading and apply a choke convolution that narrows the region, creating darker shadows in more concentrated regions. We finish by adding a displacement map to the shading, where the displacements are read from a perlin noise texture

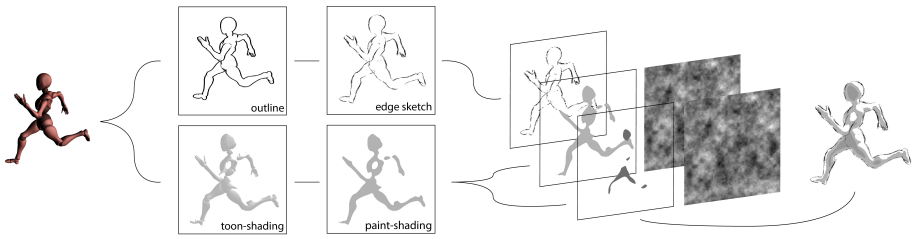

Fig. 5. Computing the hand-drawn and paint details separately before compositing them. A displacement map sampled from perlin noise is used on the paint details to give them additional temporal inconsistency.

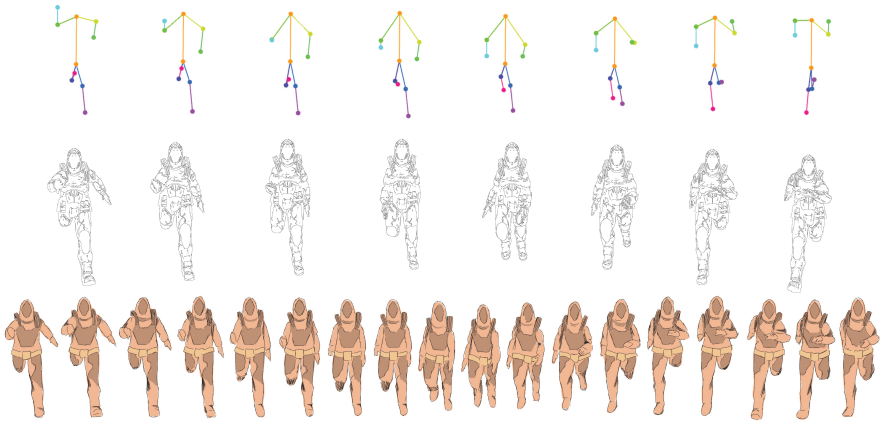

Fig. 6. Front-view run cycles. Bottom row colored-in by novice user.

to provide smooth jittery distortions to the shading per frame to give shading a slight temporal inconsistency, similar to what would be found in hand-painted frames. The results are composited with the stroke effect in Figure 5 to provide both hand-drawn strokes and shading.

4 Results

We evaluated our system on several core animations cycles, such as jumping-jacks and run cycles, which can be seen in Figure 1, Figure 6 and Figure 7.

5 participants with varying artistic backgrounds were selected to conduct animation test trials. Each participant was given a demo of the software and instructed to think of an animation cycle that they wanted to animate. The models provided by Adobe Mixamo [2] used the same joint rotation orientations, so we only had to configure the joint rotations once in order for the rotations to work for most models.

We used the resulting edge-detected run-cycle frames in Figure 1 and Figure 6 as drawing reference by sketching over them manually and shading them to give them a more hand-drawn feel. Sketching over the reference frames required no previous drawing experience, as it became a task of simply drawing strokes over the the edges of the image. Yet, sometimes drawing over every frame can take a long time, so we can also use our sketching algorithm on the same contour frames frames to generate quick automatic sketches of these animation cycles in Figure 7. Our comprehensive demo of video animations can be found here.

5 Discussion

Participants with prior art experience were able to create more comprehensive animations with advanced posing, mostly due to the fact that they have practiced

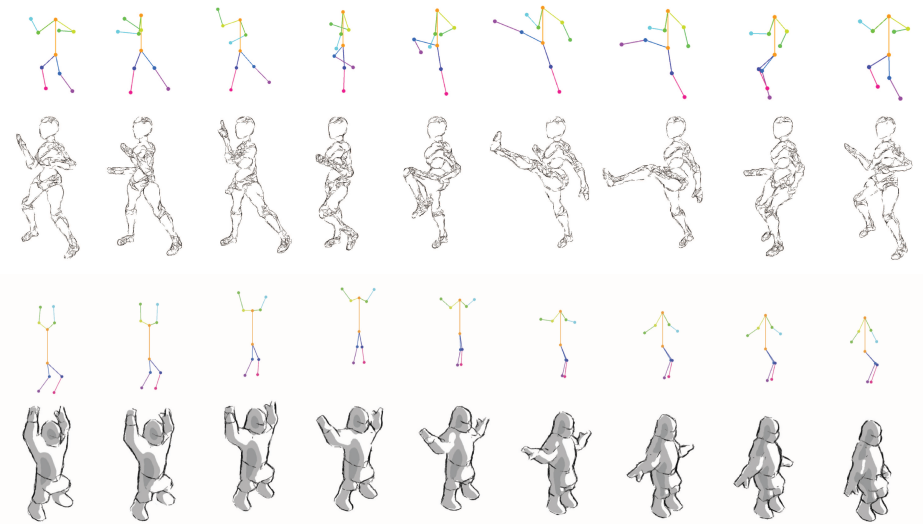

Fig. 7. Participant-generated stick figure animations and the resulting posed output. Top uses edge detection while bottom uses our sketch-painting algorithm.

Table 1. Animation Cycle Timed (in mins)

Cycle	Frames	Drawing	Editing	Cycle	Frames	Drawing	Editing
Run	24	16.4	9.4	Kick	17	10.2	4.0
Run (Side)	25	16.4	9.4	Stretch	9	7.3	5.2
Walk	11	7.3	8.9	Dab	9	5.1	2.2
Jump	10	4.2	2.2	Warrior	8	6.2	2.5
Jump (Side)	11	8.19	7.1	Tree Pose	20	8.3	6.3

character posing in their artistic career. Those without prior art experience were still able to create fluid animation cycles, mainly because the animation interface required them to draw stick figures, something that many of us are familiar with since elementary school. This, coupled with the fact that it is easy for many of us to think of animations as the overall skeletal posings rather than individual joint rotations make an interface such as this a lot simpler for many inexperienced animators than traditional posing techniques such as forward kinematics.

In some cycles in Table 1 the drawing time was substantially longer than the editing time. This happened in cycles such as run and kick that were visually more challenging to think and draw the posings for. Most cycles had lower editing times, mainly due to the fact that it is easier for users to see and edit inbetweens of primitive color-coded stick figures using onion skinning than of larger, more complex 3D character rigs, allowing users to quickly identify and edit changes to their work whenever they noticed a temporal inconsistency in the motion.

Following the study, users were asked about their impressions about the pipeline. Multiple users found it easy to think about their animations as stick figures when producing their character animation cycles, as stick figure drawings were ubiquitous in their pre-school and elementary school periods. When asking about their thought process during the animation stage, some individuals who did not have prior art or posing experience said they would move around in front of a mirror and copy down their motions as stick figures in order to easily create their animation cycles from their own movements. They found that real-life was an easy reference to them, and that converting real-life posings as stick figures came naturally for them.

Limitations. Our interface only allows drawing line strokes, prohibiting the generation of curved strokes for arc-like posing. This is because we assume a rigid-like rigging interface for our model. While some models may have many pivot points along the back to make it bend and curve, other models may lack these pivot points. In order to generalize our interface to posing multiple models, we use line strokes, not worrying about curved or arc-like body parts.

When computing joint angles, the stick figure from the 2D interface is only concerned with aligning body parts in the 3D model to match their stick figures. Yet this is still an under constrained problem, since our algorithm does not account for the fact that users should be able to rotate joints around themselves that do not change the character pose, but change body part orientations. A common example of such rotation is twisting an arm around itself. These body part orientations are not specified in the original 2D interface. Future work could explore adding a normal to each stroke to help visualize the orientation, and moving this normal around would change the orientation. When computing joint angles, this changes our alignment strategy from aligning vectors to aligning planes since each joint is described by their unique direction and normal. In this case, there always exists a unique rotation between two planes.

Future work could also look into adding in support for head rotations as well. Most artists represent heads in basic sketches as circle with a cross, where the cross's intersection represents where the nose is oriented. Our program would analyze where this point is relative to the center of the circle, as well as whether the cross is bent inwards or outwards to denote whether the nose is facing towards or away from the camera. This information would be enough to construct a unit vector from the center of the face to the cross point and compute the quaternion rotation of the vector from its rest position.

6 Conclusion

Our implementation bridges a 2D drawing interface with a 3D posing and rendering interface in order to assist with the process of creating 2D hand-drawn animations. We also introduce a novel toon-based shading scheme that builds on classic cel-shading to create hand-drawn effects and shading. We believe that with our interface, if anyone can draw stick-figures, then anyone can animate.

References

1. Why we're seeing less 2d animated movies and why they probably won't make a comeback. Bloop Animation (2019)
2. Adobe: Mixamo (2021), https://www.mixamo.com/#/
3. Aristidou, Andreas, Joan Lasenby: Graphical Models pp. 243–260 (2011)
4. Bénard, P., Hertzmann, A.: Line drawings from 3d models. CoRR **abs/1810.01175** (2018), http://arxiv.org/abs/1810.01175
5. Chen, B.Y., Ono, Y., Nishita, T.: Character animation creation using hand-drawn sketches. The Visual Computer **21**, 551–558 (09 2005). https://doi.org/10.1007/s00371-005-0333-z
6. Curtis, C.: Loose and sketchy animation (1998), http://otherthings.com/uw/loose/sketch.html
7. Davis, J., Agrawala, M., Chuang, E., Popović, Z., Salesin, D.: A sketching interface for articulated figure animation p. 320–328 (2003)
8. Day, M.: Extracting euler angles from a rotation matrix (2014)
9. Dvorožňák, M., Sýkora, D., Curtis, C., Curless, B., Sorkine-Hornung, O., Salesin, D.: Monster Mash: A single-view approach to casual 3d modeling and animation. ACM Transactions on Graphics **39**(6) (2020)
10. Guay, M., Ronfard, R., Gleicher, M., Cani, M.P.: Adding dynamics to sketch-based character animations p. 27–34 (2015)
11. Guay, M., Ronfard, R., Gleicher, M., Cani, M.P.: Space-time sketching of character animation. ACM Trans. Graph. **34**(4) (Jul 2015). https://doi.org/10.1145/2766893, https://doi.org/10.1145/2766893
12. Hecker, R., Perlin, K.: Controlling 3D objects by sketching 2D views (1992). https://doi.org/10.1117/12.131636, https://doi.org/10.1117/12.131636
13. Hudon, M., Pagés, R., Grogan, M., Ondrej, J., Smolic, A.: 2d shading for cel animation (08 2018). https://doi.org/10.1145/3229147.3229148
14. Hughes, N.: Quaternion to/from euler angle of arbitrary rotation sequence direction cosine matrix conversion using geometric methods (07 2017)
15. Jain, E., Sheikh, Y., Hodgins, J.: Leveraging the talent of hand animators to create three-dimensional animation p. 93–102 (2009). https://doi.org/10.1145/1599470.1599483, https://doi.org/10.1145/1599470.1599483
16. Jia, Y.B.: Quaternions and rotations (2013), http://graphics.stanford.edu/courses/cs348a-17-winter/Papers/quaternion.pdf
17. Kazi, R.H., Grossman, T., Umetani, N., Fitzmaurice, G.: Motion Amplifiers: Sketching Dynamic Illustrations Using the Principles of 2D Animation, p. 4599–4609. Association for Computing Machinery, New York, NY, USA (2016), https://doi.org/10.1145/2858036.2858386
18. Kwon, J.y., Lee, I.K.: Rubber-like exaggeration for character animation. pp. 18–26 (10 2007). https://doi.org/10.1109/PG.2007.25
19. Kwon, J.y., Lee, I.K.: The squash-and-stretch stylization for character motions. IEEE transactions on visualization and computer graphics **18**, 488–500 (03 2011). https://doi.org/10.1109/TVCG.2011.48
20. Lander, J.: Making kine more flexible. Game Developer Magazine pp. 15–22 (11 1998)
21. Lander, J.: Oh my god, i inverted kine! Game Developer Magazine pp. 9–14 (9 1998)

22. Liu, D., Nabail, M., Hertzmann, A., Kalogerakis, E.: Neural contours: Learning to draw lines from 3d shapes (2020)
23. Lv, P., Wang, P.J., Xu, W.W., Chai, J.X., Zhang, M.M., Pan, Z.G., Xu, M.L.: A suggestive interface for sketch-based character posing. Computer Graphics Forum **34**(7), 111–121 (2015). https://doi.org/https://doi.org/10.1111/cgf.12750, https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12750
24. Mahmudi, M., Harish, P., Le Callennec, B., Boulic, R.: Artist-oriented 3d character posing from 2d strokes. Computers Graphics **57**, 81 – 91 (2016). https://doi.org/https://doi.org/10.1016/j.cag.2016.03.008, http://www.sciencedirect.com/science/article/pii/S0097849316300218
25. Mahmudi, M., Harish, P., Le Callennec, B., Boulic, R.: Artist-oriented 3d character posing from 2d strokes. Computers Graphics **57**, 81–91 (2016). https://doi.org/https://doi.org/10.1016/j.cag.2016.03.008, https://www.sciencedirect.com/science/article/pii/S0097849316300218
26. Mao, C., Qin, S.F., Wright, D.: A sketch-based approach to human body modelling. Computers Graphics **33**(4), 521–541 (2009). https://doi.org/https://doi.org/10.1016/j.cag.2009.03.028, https://www.sciencedirect.com/science/article/pii/S0097849309000594
27. Muhammad, U.R., Yang, Y., Song, Y., Xiang, T., Hospedales, T.M.: Learning deep sketch abstraction. CoRR **abs/1804.04804** (2018), http://arxiv.org/abs/1804.04804
28. Pan, J., Zhang, J.: Sketch-based skeleton-driven 2d animation and motion capture. T. Edutainment **6**, 164–181 (01 2011). https://doi.org/10.1007/978-3-642-22639-7$_1$7
29. Steger, E.: Sketch-based animation language
30. Stein, M.: Will disney ever return to making hand-drawn animated films? The Mickey Mindset (2016)
31. Tang, Z., Xiao, J., Feng, Y., Yang, X., Zhang, J.: Human motion retrieval based on freehand sketch. Computer Animation and Virtual Worlds **25**(3-4), 271–279 (2014). https://doi.org/https://doi.org/10.1002/cav.1602, https://onlinelibrary.wiley.com/doi/abs/10.1002/cav.1602
32. Thomas, F.: The illusion of life : Disney animation. (1995)
33. Thorne, M., Burke, D., van de Panne, M.: Motion doodles: An interface for sketching character motion. ACM Trans. Graph. **23**(3), 424–431 (Aug 2004). https://doi.org/10.1145/1015706.1015740, https://doi.org/10.1145/1015706.1015740
34. Tong, Z., Chen, X., Ni, B., Wang, X.: Sketch generation with drawing process guided by vector flow and grayscale (2020)
35. Whited, B., Daniels, E., Kaschalk, M., Osborne, P., Odermatt, K.: Computer-assisted animation of line and paint in disney's paperman (08 2012). https://doi.org/10.1145/2343045.2343071
36. Wu, T.T.: Character rigs for motion exaggeration (2006)
37. Yang, R., Wünsche, B.: Life-sketch - a framework for sketch-based modelling and animation of 3d objects. Conferences in Research and Practice in Information Technology Series **106** (01 2010)
38. Yonemoto, S.: A sketch-based skeletal figure animation tool for novice users pp. 37–42 (2012). https://doi.org/10.1109/CGIV.2012.18