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

Category: Research

Figure 1: Side-view of a run-cycle stick figure with its corresponding edge-filtered 3D-posed models. Our program computes the joint angles of a set of key body joints before applying the euler rotations to the 3D model. The model is then rasterized and edge-detected to be used as reference for keyframes in 2D hand-drawn animations. Bottom row of frames were sketched out using the middle row reference keyframes and colored in.

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 the contour-detected rasterizations. The image sequence can be used as a stand-alone outline for character animations, or can be brought into any external drawing program where users without drawing experience can add additional sketch-details and inbetweens from these character keyframes to create their own animations. 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.

Index Terms: Human-centered computing—Human computer interaction (HCI)—Interaction paradigms—Graphical user interfaces; Applied computing—Arts and humanities—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 [27]. 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. Instead, the 2D interface uses stick-figure drawings in order to pose a 3D character that can be rasterized down to 2D reference keyframes. We attempt to make the 2D interface similar to drawing on paper so that users can comfortably create animation cycles based on their previous knowledge of drawing stick figures on pen and paper from early grade-school days. 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.

Contributions: 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 the reference keyframes needed to draw characters quickly and consistently. 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.

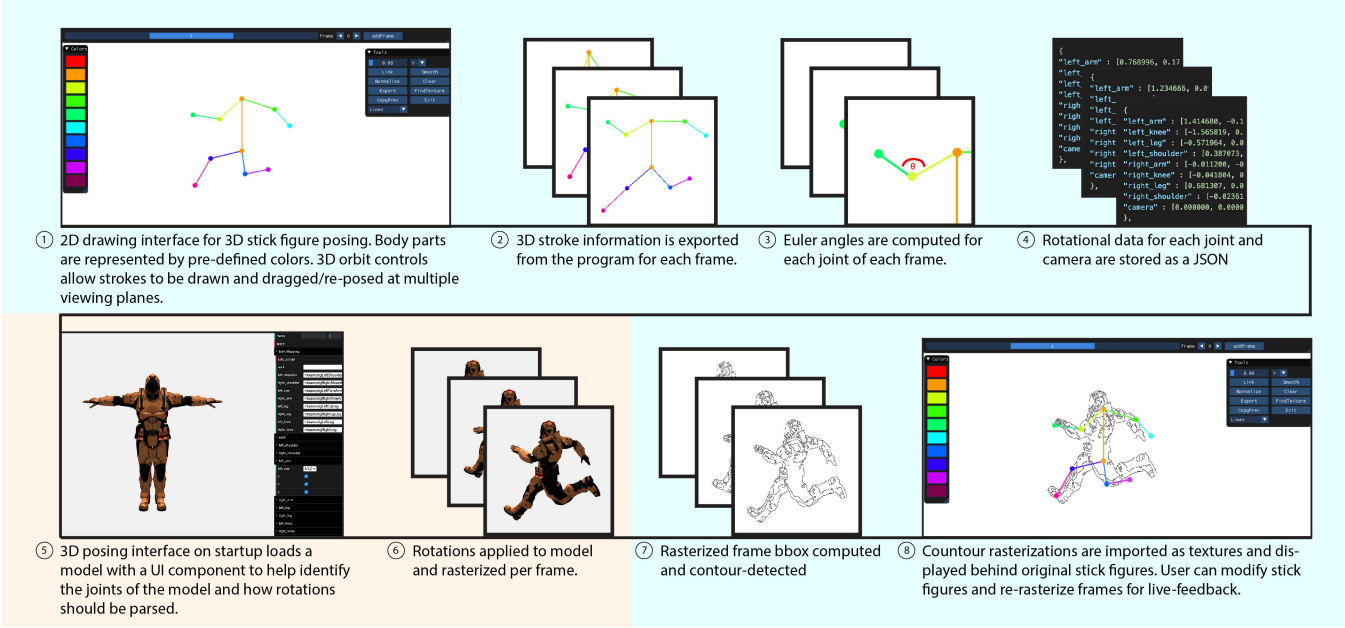

Figure 2: Block diagram of the interface. Blue regions refer to the sketch interface while orange regions refer to the posing interface. The joint rotations of the sketch in 3D space are used to pose a user-imported 3D model. These model frames are then rasterized, cropped, and contour-detected before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

2 Related Works

Making 2D animation easier has been a persistent challenge, pushing creators to spend more time dreaming and less time drawing. A few tools exist in helping to assist the 2D animation pipeline by combining traditional animation with vector software to reduce the need for drawing ability. Disney Research's Mender was developed as a vector/raster hybrid interface for computing in-betweens quickly [32] 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 [14] [29], 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 [25], but this is limited in viewpoint as we cannot consider 3D rotations. We chose to have a 3D-pipeline for assisting 2D animation after seeing how much more flexible 3D animation is at animating various joint and body turns that similar 2D vector-based animation tools struggle with.

When using our 3D-pipeline as a reference for 2D hand drawn animation, it is important that the 3D motions still obey a 2D-like movement. 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 [33] [15] or squash and stretch [16] 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 [18] [17] 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 [21], 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.

We look at trying to use 3D animation to assist in the 2D animation pipeline, yet the inverse problem of 2D-assisted 3D animation has also been done from rough 2D skeletons [5] [9] or from detailed artist drawings [12]. 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 [20] [28], 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 [22] [26] and to define spatial trajectories of character motions [30] [7] [8]. These same 3D drawing environments can also be used for creating 3D human character models [23] or various other 3D models [4] [35] [34] 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 [13] 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.

Once a 3D character model has been posed, there are techniques to help make 3D character models more cartoon-like as well, giving them a more 2D feel. Toon Shading, also known as Cel Shading, is a non-photorealistic rendering style that helps give 3D animation a 2D cartoon-like look [10] 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 [24] [31] [19]. When rasterizing 3D-posed

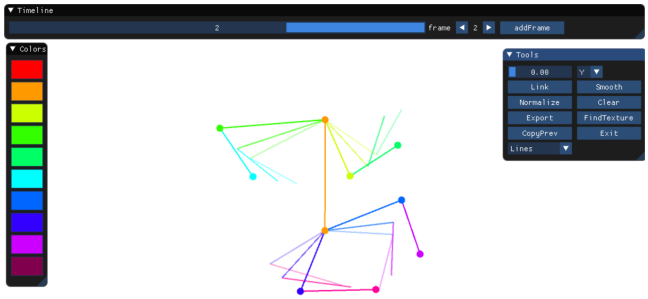

Figure 3: 2D drawing interface. The left column lists the different colors for each body part while the right column lists different functionalities for rotating the camera, linking joints and more. The timeline at the top allows the user to draw out multiple frames and use onion-skinning to view previous frames at a lower opacity to help with temporal consistency in movements.

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. Rather than manually hand-drawing frames, this approach saves users time by algorithmically applying a hand-drawn effect to frames to get hand drawn visuals per frame while also achieving hand-drawn motions from posing the 3D model with the hand-drawn skeleton.

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 angles that comprise the stick figure. This data is sent to the 3D interface where these rotations are applied to a rigged character model and rasterized. The 2D interface can query for this rasterized frame and compute the contours before displaying the results behind the stick figure. Here, the user has a chance to make any corrections and re-rasterize any frames they are not happy with. Once the user is happy with the set of frames, they can sketch their character animation using these frames as keyframe reference, where all the character proportions and posing has already been given to them. Alternatively, users can use the hand-drawn program we discuss in section 3.4 to algorithmically give the character a sketch-like feel without having to take the time to re-sketch every frame.

3.1 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. These colors are necessary for telling the interface which strokes correspond to which body parts. 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. The camera can also be orbitally rotated along the x, y, or z-axis to view the stick figure from different angles 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. This way, only one instance of each body part exists 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. We can also copy previous frames and drag around endpoints for the current frame, ultimately reducing stick figure animation cycles to just drawing one frame and iteratively dragging the joints around for each keyframe.

When the joints have been drawn and posed, 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 contour-detected 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. Once satisfied, the user can close the application and use the resulting frames as keyframe drawing reference for their animations.

Algorithm 1 Joint Rotations

for baseID, endID in jointMap **do**
  Vec3 base0, base1 ← bodyParts[baseID]
  Vec3 end0, end1 ← bodyParts[endID]
  //normalize and center strokes
  Vec3 base ← normalize(base1 - base0);
  Vec3 end ← normalize(end1 - end0);
  // compute xyzw quaternion
  Vec3 xyz ← base × end;
  float w ← $\sqrt{(\text{base}\cdot\text{base})*(\text{end}\cdot\text{end})}$ + (base · end)
  float norm ← $\sqrt{(\text{xyz}\cdot\text{xyz})+(\text{w}*\text{w})}$
  xyz ← xyz / norm
  w ← w / norm
  // convert to euler angles
  Vec3 euler ← quatToEuler(xyz.x, xyz.y, xyz.z, w)
end for

3.2 3D Interface

When the user is done drawing several frames of stick figures, the joint rotations for each frame are sent to the 3D interface. Each joint maps to a base stroke and end stroke, where we seek to obtain the rotations to move a vector from the base stroke to end stroke. For each joint, we move both strokes to the origin, normalize them, and compute the quaternion rotation from the base stroke to the end stroke [11]. We convert the quaternions to euler angles as in Algorithm 1 and send them over to our 3D Interface [6]. We can also use linear interpolation between successive joint angles, allowing us to draw fewer keyframes while still obtaining similar results using lerp.

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. The user can also specify the euler rotation order [XYZ, XZY, YXZ, etc...] and orientation [CW, CCW] per joint. Because these are model-specific parameters, they only need to be computed once per model as shown in Figure 4.

The rotations are then applied per joint, where we first save the initial joint rotations of the model, then add to these initial rotations our modified joint rotations. We can also apply camera rotations per frame, where we capture the view angle when we were in the 2D interface and apply the resulting rotations to an orbital camera centered around the model. Once each frame of the model has been posed, we then rasterize the frames and save them to the device.

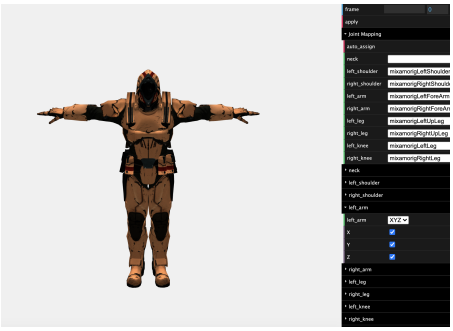

Figure 4: 3D posing interface with configurable parameters. The user specifies the joint mappings between the essential joints in the 2D setting with the joint names for the specific model, as well as the joint parsing order and direction per angle.

3.3 Importing to 2D Interface

We can search for the rasterized frames on the device and run a sobel filter over the image to edge detect them. We then compute the bounding box of the edge-detected 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 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. Once the user is happy with the frames, they can exit the program and find the original-posed and edge-detected frames to use as keyframe drawing reference.

3.4 Sketch Effect

We can generate a hand-drawn feeling along the contours of each frame by applying rough sketch strokes to these contours. Each sketch is comprised of a bezier curve of several nearby points. The number of points per spline determines the length of the stroke. We first apply an inverse choke convolution on the edge detected frames to expand the edges that we will sample points from for our stroke. In path-first search, 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. In point-first search, we generate all points for a spline within the same region in

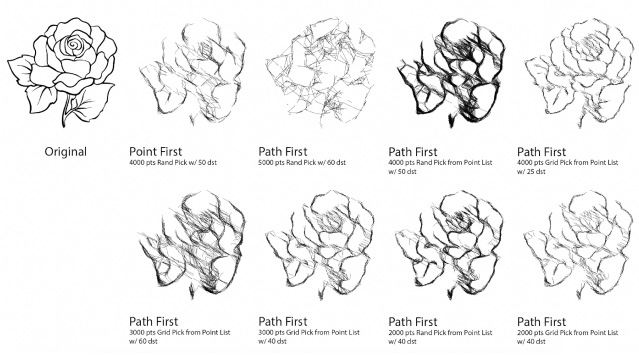

Figure 5: Sketching results. Increasing number of points or distance between points leads to a deeper sketch result. In areas of high or concentrated contours, sketches produce a scribble-like shading effect.

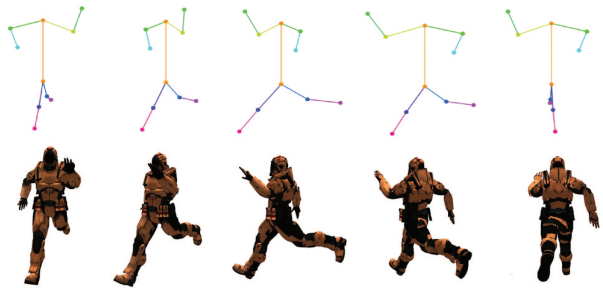

Figure 6: A stick figure pose visualized at different angles.

parallel, making sure they are roughly the same distance away before connecting them together. This approach leads to more stochastic and rougher edge strokes compared to path-first search. 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. Some visual results comparing the styles can be seen in figure 5.

4 Results

We evaluated our system on a few core animations cycles, such as jumping-jacks and run cycles. We begin by drawing 2D keyframes in the xy plane, and editing their depth in the z-plane. The resulting run cycles can be seen in Figure 1, Figure 7 and Figure 8.

5 participants with varying artistic backgrounds were selected to conduct animation test trials. Some participants had prior art abilities from traditional courses while others had no prior art or doodling experience. 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 provided by the Mixamo database, saving time for many users. Configuring the model's rotations only had to be done once so the participants did not have to configure the rotations, but rather were provided pre-configured models for their animations.

Subjects were then given time to draw out their animations as stick figures in our pipeline. The draw time for creating and modifying the various animation cycles are recorded in Table 1. The table reports the time in minutes each user spent drawing the frames and then editing drawings to look more temporally consistent between frame. The time taken to draw the animation cycle is roughly linear with the frame rate, although some cycles like running require more complex and diverse posing per frame, taking substantially longer than other cycles. The sketch and color time for certain cycles that were hand-drawn after are also provided. (.) means the sketch results were algorithmically generated.

We used the resulting edge-detected frames for the run cycle in Figure 1 and Figure 7 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. Figure 9 demonstrates the difference between various automatic sketching results compared to the hand-drawn case from our posed reference frame.

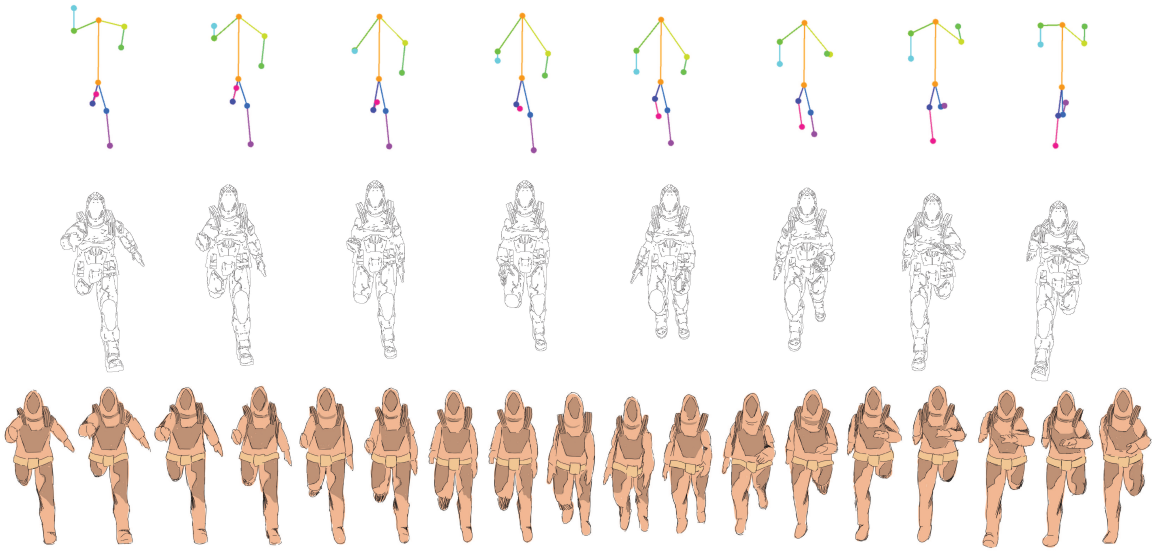

Figure 7: Side-view and front-view run cycles of first half of cycle frames using the same stick figures sequence exported at different views.

Table 1: Animation Cycle Timed (in mins)

Cycle	Frame Count	Drawing	Editing	Sketching
Run	24	16.4	9.4	34.2
Run (Side)	25	16.4	9.4	29.8
Walk	11	7.3	8.9	(.)
Jump	10	4.2	2.2	(.)
Jump (Side)	11	8.19	7.1	(.)
Kick	17	10.2	4.0	(.)
Stretch	9	7.3	5.2	(.)
Dab	9	5.1	2.2	(.)
Warrior	8	6.2	2.5	(.)
Tree Pose	20	8.3	6.3	(.)

5 Discussion

5.1 Ease of Use

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

5.2 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.

We find that the resulting posing is independent of the body part lengths. In some drawings, legs are purposely shortened or elongated, but that does not affect the posing of characters. Our algorithm only considers the angle of the normalized adjacent body parts, so any notion of length is disregarded. This is beneficial in the cases where the user may not have good proportional drawing abilities with the various body parts of the stick figure. One potential downside is that the resulting pose is very rigid in geometry, and that the user cannot squash or stretch different body parts using this method. With elements such as squash-and-stretch are desired attributes in hand-drawn animation, it would be nice to have the flexibility to define how certain parts of the body would elongate or shorter. Yet, there are also many cases when the user may mistakenly make something longer than it actually is, with no intent of actually changing its length. Since the later case is more likely, the algorithm adopted a rotation-based strategy that conserves length.

Another potential issue with long or potentially inconsistent stroke lengths in the stick figures is that they give the wrong impressions of orientations/rotations. For example, consider two strokes where one appears longer in the xy-plane. We would assume based

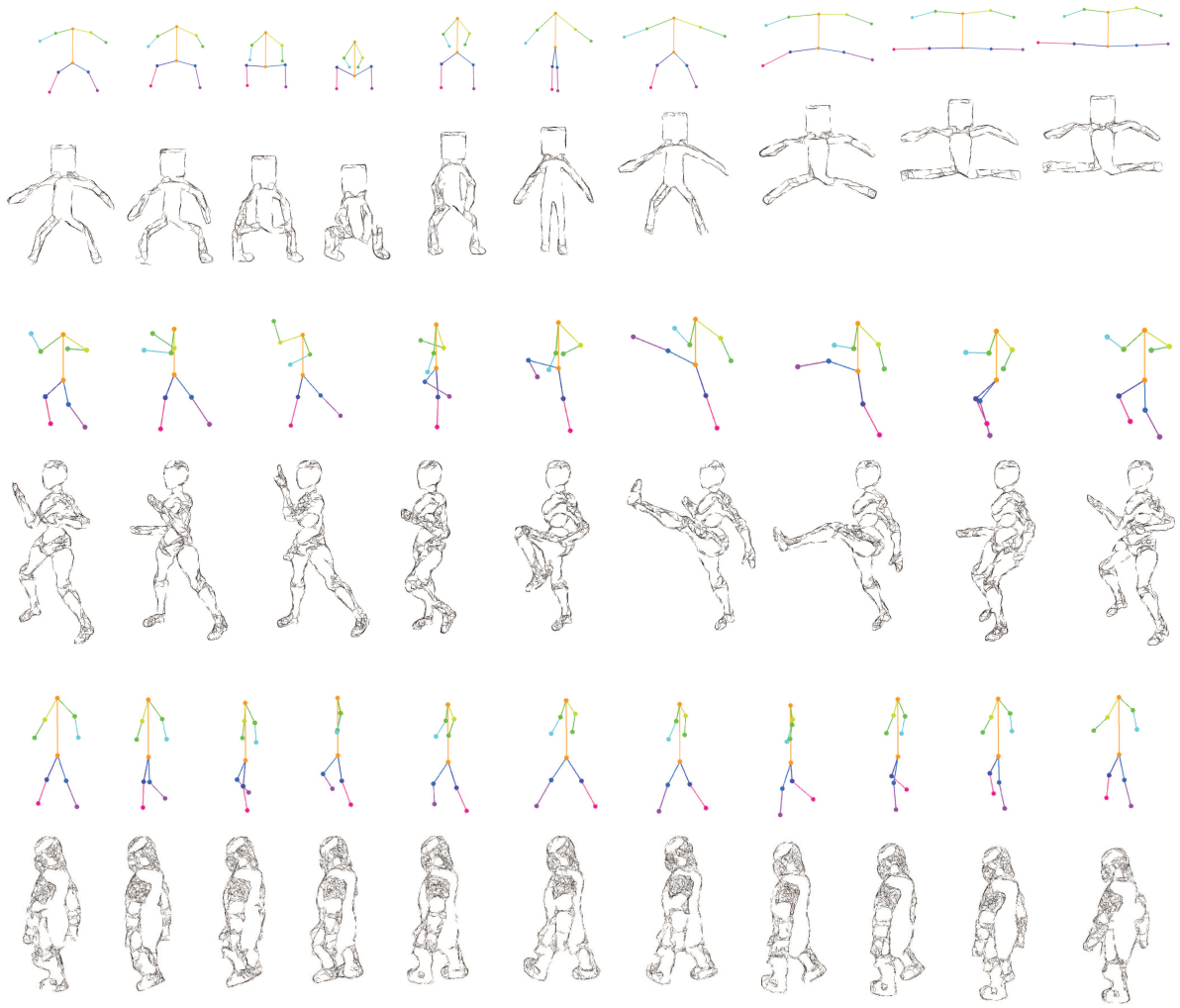

Figure 8: Participant-generated stick figure animations and the resulting posed output. Sketches generated with 1000 20px-mean length strokes.

on our perceptions from the real world that the shorter stroke must be more elongated in the z-axis (not visible to the xy-plane), but it may also be the case that the longer stroke in the xy-plane is even longer in the z-axis too since it was just drawn very long in that direction. We allow the user to normalize each stroke by the average stroke length of the scene so that each stroke is the same length. We make this assumption since each main bone in the human body has roughly the same length. By making every stroke the same length, we can get a better hold of what joints are longer in the current viewing plane, and what joints are longer in perpendicular viewing planes.

6 Future Work

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 it would come time to compute the 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 euler rotation of the vector from its rest position.

7 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. Rather than requiring that users have the

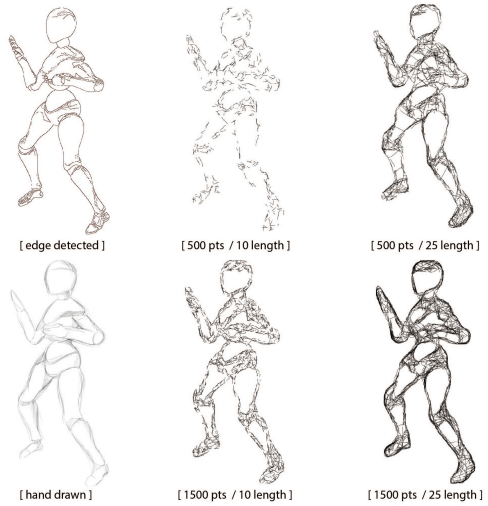

Figure 9: Comparisons of different sketch results. Hand drawn was done by an inexperienced artist in Photoshop using the edge detected frame as drawing reference. The other results were generated from the edge detected frame using our sketch algorithm.

skills to draw complex characters, users can simply draw a sequence of segmented stick-figure frames that the 3D interface uses to pose their favorite character model and export back for the 2D interface to use as reference to draw over or to automatically create sketches for. 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.

[3] Aristidou, Andreas and Joan Lasenby. *Graphical Models*, pp. 243–260, 2011.

[4] B.-Y. Chen, Y. Ono, and T. Nishita. Character animation creation using hand-drawn sketches. *The Visual Computer*, 21:551–558, 09 2005. doi: 10.1007/s00371-005-0333-z

[5] J. Davis, M. Agrawala, E. Chuang, Z. Popović, and D. Salesin. A sketching interface for articulated figure animation. p. 320–328, 2003.

[6] M. Day. Extracting euler angles from a rotation matrix. 2014.

[7] M. Guay, R. Ronfard, M. Gleicher, and M.-P. Cani. Adding dynamics to sketch-based character animations. p. 27–34, 2015.

[8] M. Guay, R. Ronfard, M. Gleicher, and M.-P. Cani. Space-time sketching of character animation. *ACM Trans. Graph.*, 34(4), July 2015. doi: 10.1145/2766893

[9] R. Hecker and K. Perlin. Controlling 3D objects by sketching 2D views, 1992. doi: 10.1117/12.131636

[10] M. Hudon, R. Pagés, M. Grogan, J. Ondrej, and A. Smolic. 2d shading for cel animation. 08 2018. doi: 10.1145/3229147.3229148

[11] N. Hughes. Quaternion to/from euler angle of arbitrary rotation sequence direction cosine matrix conversion using geometric methods. 07 2017.

[12] E. Jain, Y. Sheikh, and J. Hodgins. Leveraging the talent of hand animators to create three-dimensional animation. p. 93–102, 2009. doi: 10.1145/1599470.1599483

[13] Y.-B. Jia. Quaternions and rotations. 2013.

[14] R. H. Kazi, T. Grossman, N. Umetani, and G. Fitzmaurice. *Motion Amplifiers: Sketching Dynamic Illustrations Using the Principles of 2D Animation*, p. 4599–4609. Association for Computing Machinery, New York, NY, USA, 2016.

[15] J.-y. Kwon and I.-K. Lee. Rubber-like exaggeration for character animation. pp. 18–26, 10 2007. doi: 10.1109/PG.2007.25

[16] J.-y. Kwon and I.-K. Lee. The squash-and-stretch stylization for character motions. *IEEE transactions on visualization and computer graphics*, 18:488–500, 03 2011. doi: 10.1109/TVCG.2011.48

[17] J. Lander. Making kine more flexible. *Game Developer Magazine*, pp. 15–22, 11 1998.

[18] J. Lander. Oh my god, i inverted kine! *Game Developer Magazine*, pp. 9–14, 9 1998.

[19] D. Liu, M. Nabail, A. Hertzmann, and E. Kalogerakis. Neural contours: Learning to draw lines from 3d shapes. 2020.

[20] P. Lv, P. J. Wang, W. W. Xu, J. X. Chai, M. M. Zhang, Z. G. Pan, and M. L. Xu. A suggestive interface for sketch-based character posing. *Computer Graphics Forum*, 34(7):111–121, 2015. doi: 10.1111/cgf.12750

[21] M. Mahmudi, P. Harish, B. Le Callennec, and R. Boulic. Artist-oriented 3d character posing from 2d strokes. *Computers Graphics*, 57:81 – 91, 2016. doi: 10.1016/j.cag.2016.03.008

[22] M. Mahmudi, P. Harish, B. Le Callennec, and R. Boulic. Artist-oriented 3d character posing from 2d strokes. *Computers Graphics*, 57:81–91, 2016. doi: 10.1016/j.cag.2016.03.008

[23] C. Mao, S. F. Qin, and D. Wright. A sketch-based approach to human body modelling. *Computers Graphics*, 33(4):521–541, 2009. doi: 10.1016/j.cag.2009.03.028

[24] U. R. Muhammad, Y. Yang, Y. Song, T. Xiang, and T. M. Hospedales. Learning deep sketch abstraction. *CoRR*, abs/1804.04804, 2018.

[25] J. Pan and J. Zhang. Sketch-based skeleton-driven 2d animation and motion capture. *T. Edutainment*, 6:164–181, 01 2011. doi: 10.1007/978-3-642-22639-7_17

[26] E. Steger. Sketch-based animation language.

[27] M. Stein. Will disney ever return to making hand-drawn animated films? *The Mickey Mindset*, 2016.

[28] Z. Tang, J. Xiao, Y. Feng, X. Yang, and J. Zhang. Human motion retrieval based on freehand sketch. *Computer Animation and Virtual Worlds*, 25(3-4):271–279, 2014. doi: 10.1002/cav.1602

[29] F. Thomas. The illusion of life : Disney animation. 1995.

[30] M. Thorne, D. Burke, and M. van de Panne. Motion doodles: An interface for sketching character motion. *ACM Trans. Graph.*, 23(3):424–431, Aug. 2004. doi: 10.1145/1015706.1015740

[31] Z. Tong, X. Chen, B. Ni, and X. Wang. Sketch generation with drawing process guided by vector flow and grayscale. 2020.

[32] B. Whited, E. Daniels, M. Kaschalk, P. Osborne, and K. Odermatt. Computer-assisted animation of line and paint in disney's paperman. 08 2012. doi: 10.1145/2343045.2343071

[33] T.-T. Wu. Character rigs for motion exaggeration. 2006.

[34] R. Yang and B. Wünsche. 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.

[35] S. Yonemoto. A sketch-based skeletal figure animation tool for novice users. pp. 37–42, 2012. doi: 10.1109/CGIV.2012.18