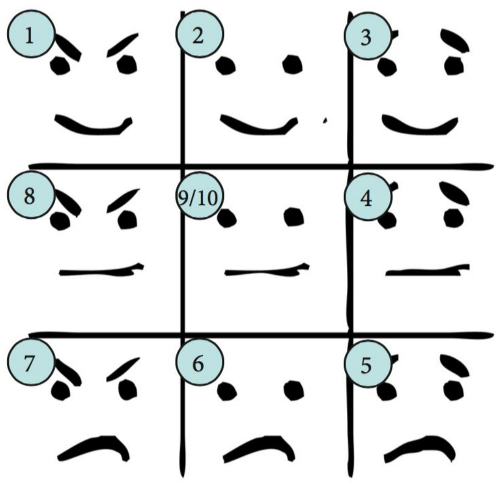
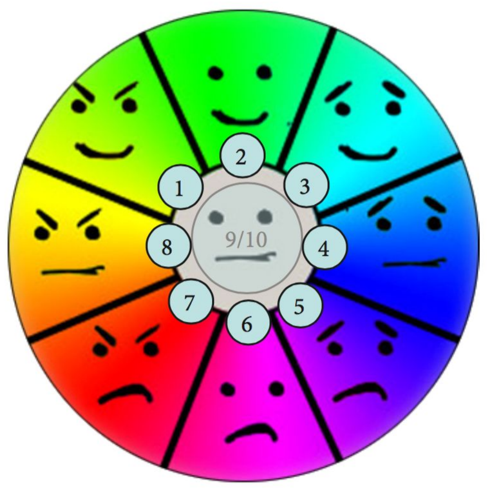
Botanic Technologies

Avatar & Animation Creation Best Practices for ACTR





The purpose of ACTR is to drive real-time animations that are contextually appropriate to spoken words. This may include animating an avatar based on the output of a natural language processing system, driving a VR scene graph via voice control, or other purposes. The following specifications detail animating avatars that are connected to chatbots.

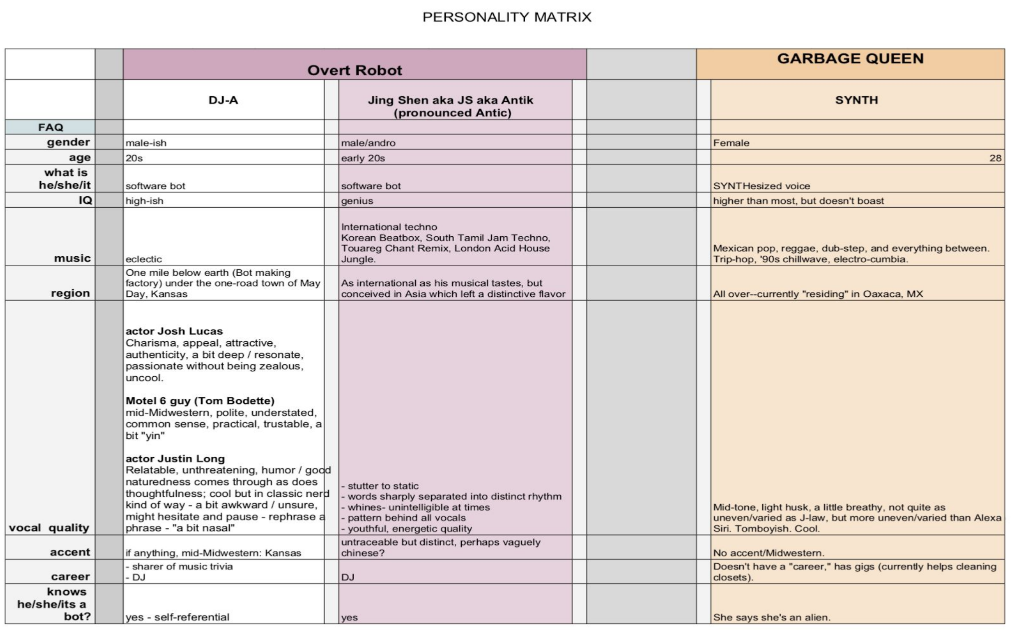
Table of Contents

Preface

The process of building Autonomous Character & Topology Resource (ACTR) avatars that are able to provide real-time animation across a range of channels and devices is a process that requires a detailed working recipe, as outlined below. This is not intended to be an exhaustive list, rather guidelines for best-practices. Notes and ongoing dialogue around these processes will help us improve the methods as technologies and talents continue to emerge.

Character Personality Brief

Who is your character? This is a very important question to answer, seeing as it influences many choices that follow. Make sure to take time to research who your character is. So, where to start? Thinking about the end user and potentially interviewing them will give a lot of insight into designing the best fit personality. The goal of the application will also play a major role. For example, a learning app might require a personality that presents competency and intelligence while a friendly and approachable personality might work best for an app for an airline or travel company. The type of information gathered will influence things like shape of the character, facial features, wardrobe, and animation.



PERSONALITY MATRIX

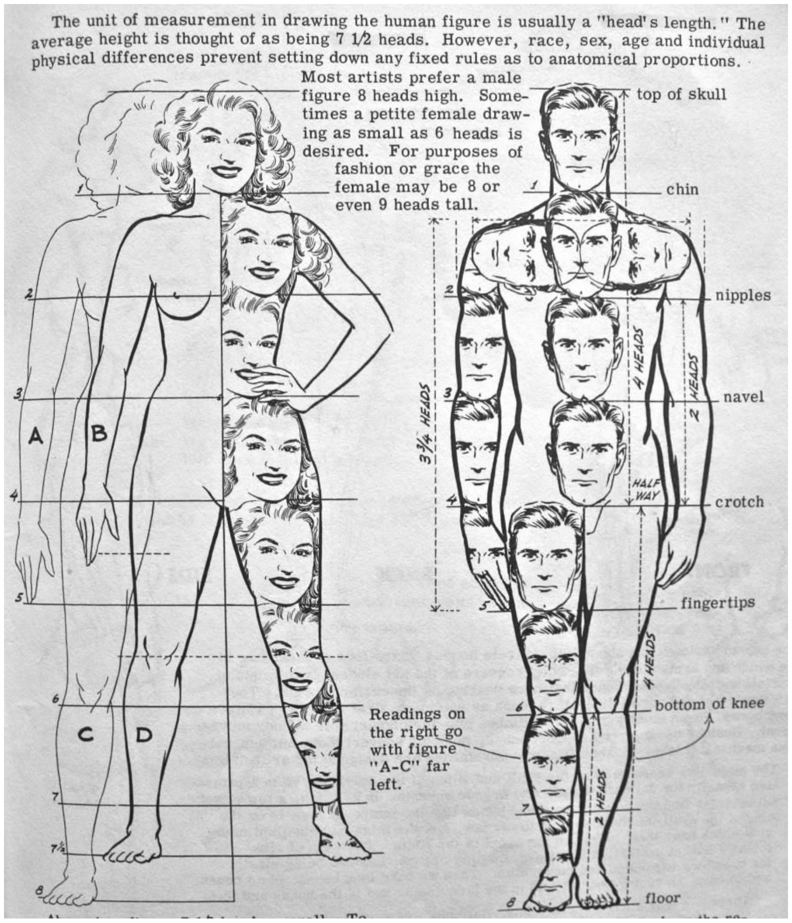
	Overt Robot		GARBAGE QUEEN
	DJ-A	**Jing Shen aka JS aka Antik (pronounced Antic)**	**SYNTH**
FAQ			
gender	male-ish	male/andro	Female
age	20s	early 20s	28
what is he/she/it	software bot	software bot	SYNTHesized voice
IQ	high-ish	genius	higher than most, but doesn't boast
music	eclectic	International techno Korean Beatbox, South Tamil Jam Techno, Touareg Chant Remix, London Acid House Jungle.	Mexican pop, reggae, dub-step, and everything between. Trip-hop, '90s chillwave, electro-cumbia.
region	One mile below earth (Bot making factory) under the one-road town of May Day, Kansas	As international as his musical tastes, but conceived in Asia which left a distinctive flavor	All over--currently "residing" in Oaxaca, MX
vocal quality	**actor Josh Lucas** Charisma, appeal, attractive, authenticity, a bit deep / resonate, passionate without being zealous, uncool. **Motel 6 guy (Tom Bodette)** mid-Midwestern, polite, understated, common sense, practical, trustable, a bit "yin" **actor Justin Long** Relatable, unthreatening, humor / good naturedness comes through as does thoughtfulness; cool but in classic nerd kind of way - a bit awkward / unsure, might hesitate and pause - rephrase a phrase - "a bit nasal"	- stutter to static - words sharply separated into distinct rhythm - whines- unintelligible at times - pattern behind all vocals - youthful, energetic quality	Mid-tone, light husk, a little breathy, not quite as uneven/varied as J-law, but more uneven/varied than Alexa Siri. Tomboyish. Cool.
accent	if anything, mid-Midwestern: Kansas	untraceable but distinct, perhaps vaguely chinese?	No accent/Midwestern.
career	- sharer of music trivia - DJ	DJ	Doesn't have a "career," has gigs (currently helps cleaning closets).
knows he/she/its a bot?	yes - self-referential	yes	She says she's an alien.

An example of a Personality Matrix character design with three characters, of two types, shown.

Character Concept Design

Concepting with art fundamentals such as form and shape in mind will take you far. Use tools that don't get in the way and design for appeal while exploring what brings out the character's personality.

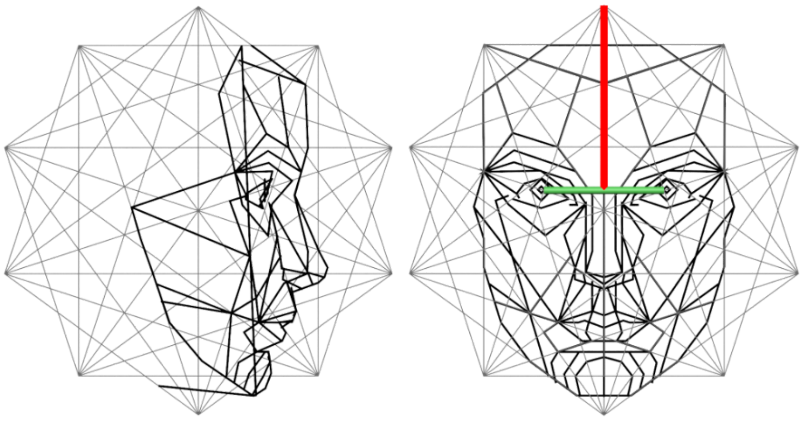
The first round of characters being released will be of two different approaches. One based on the "ideal" body and facial proportions and one of a more stylized approach. For the "ideal" approach, female body proportions follow a height of 7 ½ heads and males follow an 8 head height proportion. The "ideal" facial proportions are based on Marquardt's Phi Mask, also known as the "Golden Ratio Mask." Our initial animation libraries will also be based on these ratios so to get a perfect transfer of animation, we recommend matching these proportions.

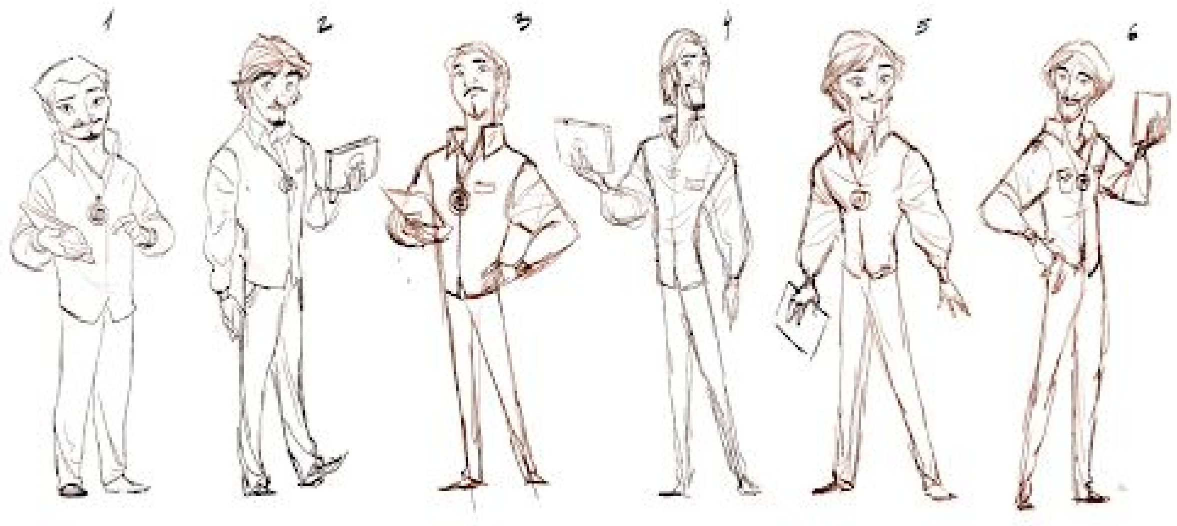


The unit of measurement in drawing the human figure is usually a "head's length." The average height is thought of as being 7 1/2 heads. However, race, sex, age and individual physical differences prevent setting down any fixed rules as to anatomical proportions.

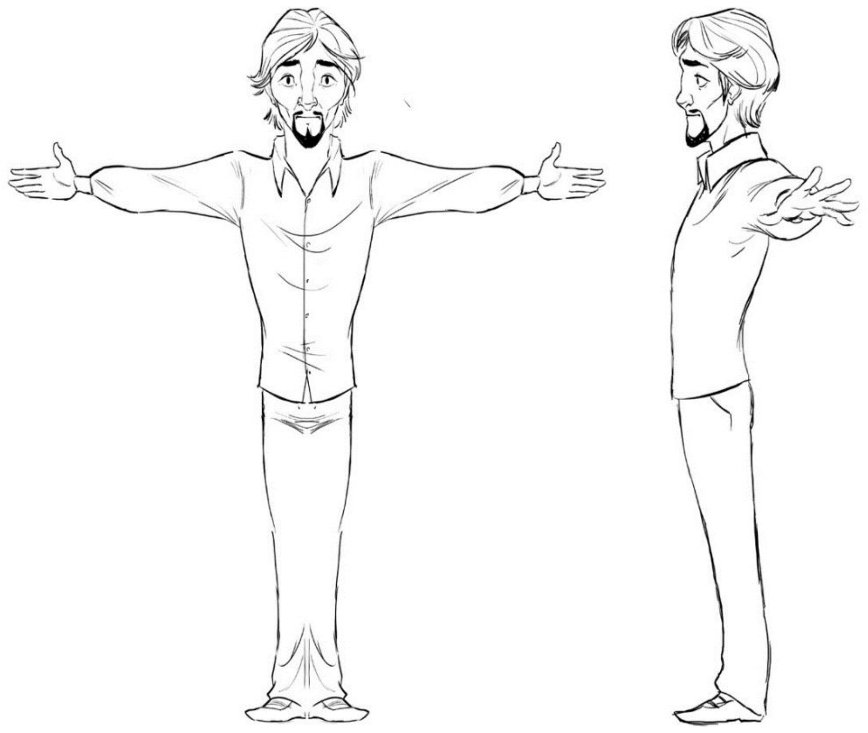
Most artists prefer a male figure 8 heads high. Sometimes a petite female drawing as small as 6 heads is desired. For purposes of fashion or grace the female may be 8 or even 9 heads tall.

Readings on the right go with figure "A-C" far left.





There is a great deal of creative freedom in the “stylized” character approach. This method allows you to explore various shape and line combinations. It’s good practice to create more than one option to choose from. Narrow it down to whichever concept you think best matches the Personality Brief. From there, you may choose to create orthographic sketches than can be used in the modeling phase.



Modeling

If you are methodical in what you create it doesn't matter how you get there. Efficient and intelligent topology, optimized UV space and scene hierarchy are extremely important "whats" that ensure the avatar will appear, perform and deform well in the end application. Our approach at Botanic is a high to low poly baking method using various software including, ZBrush, Maya and Marmoset Toolbag.

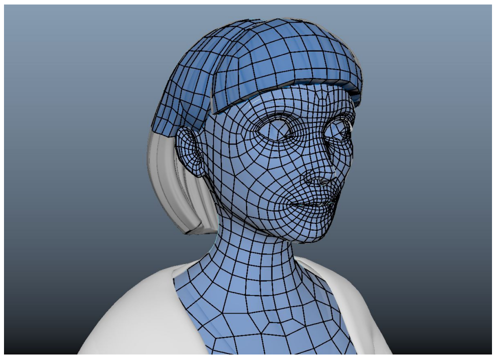
Modeling Process Overview

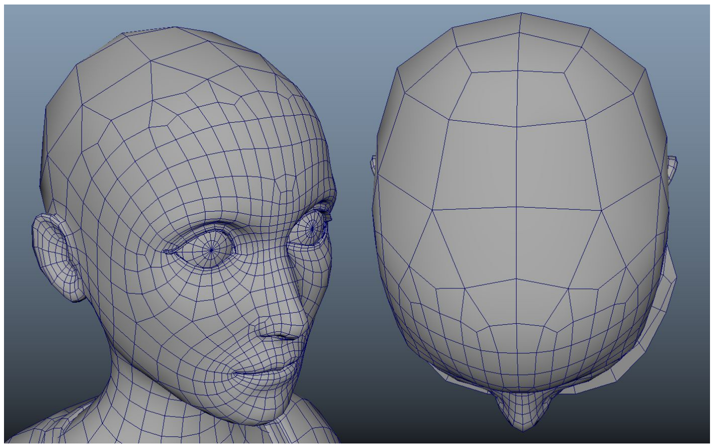
1. High-poly sculpt (ZBrush)
2. Retopology (Maya)
3. Model assembly and smoothing normals (Maya)
4. UV layout (Maya)
5. Setup for export to Zbrush (Maya)
6. Assemble high and low poly meshes (ZBrush)

Topology

Some people like to focus on topology at the beginning of character modeling. When we start, we focus on creative things like shape, form and appeal without technical limitations in mind. After our character sculpt is nailed down we then focus on the technical side and move forward with retopology.

The following are some examples of good topology workflows. Match the number of top and bottom edge loops for the mouth and eyelids. This ensures a proper seal when shut as well as being able to make similar shapes. If modeling eyelashes, match edge loops exactly to the eyelids so it properly follows along with the eyelids. Penetrating geometry is okay, just be smart about placement. Face normals should always face outwards and be smoothed. In order to keep a happy rigging artist, we highly recommend having a symmetrical mesh. It‘s possible to have an asymmetrical mesh if absolutely necessary. However, in some circumstances, a likely better solution would be offsetting the look with a texture and normal map.

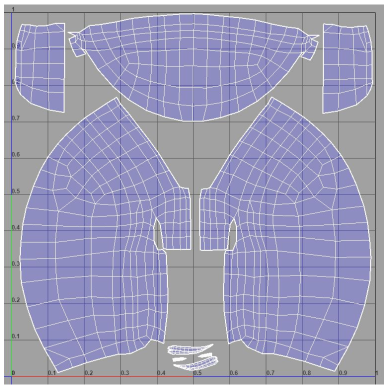


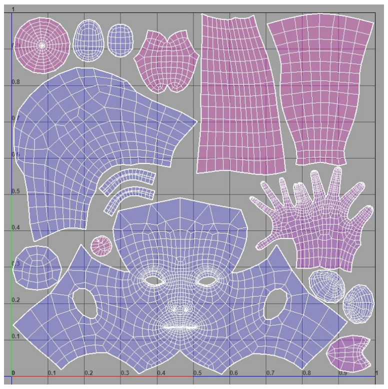


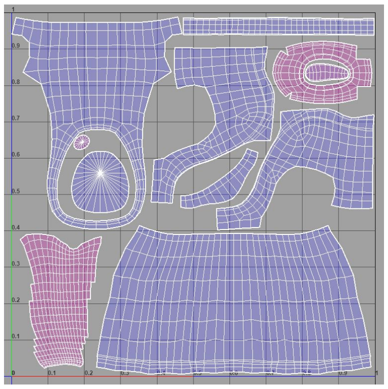
Poly count is a highly debated topic and limits can vary depending on factors such as device model/type, material shaders, scene complexity, etc. While the following number isn't set in stone, we recommend not exceeding 10K polys. If you're efficient with your topology, 10K polys should be more than enough to create a good-looking character while performing well on most mobile devices. Using purposeful kiting will allow you to have more polygons in important areas such as the face and fewer polygons in less important areas such as the top of the head. You may think the top of the head would be better optimized by deleting those faces because it's covered by hair. In many cases that would be true. Our goal in this specific example was to create a versatile character that could be used by many people for many uses. With that in mind, keeping that portion of the head proves useful as it allows numerous hairstyles including baldness.

UV's

To make a character more easily customizable we recommend splitting your model into UV sets strategically to specific regions. In this example we have one for the body, hair, and outfit. This allows for swapping parts of the model out while still being efficient with your texture space. It wouldn't make much sense, for example, to have the hair UV's share the same texture space as the body because if a new model of hair were to be put on the old model UV's would be taking up valuable texture space that could otherwise be used for displaying the body better. It may also be desirable to have symmetrical parts, such as the hands, eyes, shoes, etc., share the same UV space to maximize pixels given to other regions of the model. The reddish-purple UV islands, pictured below, show this overlapping method.





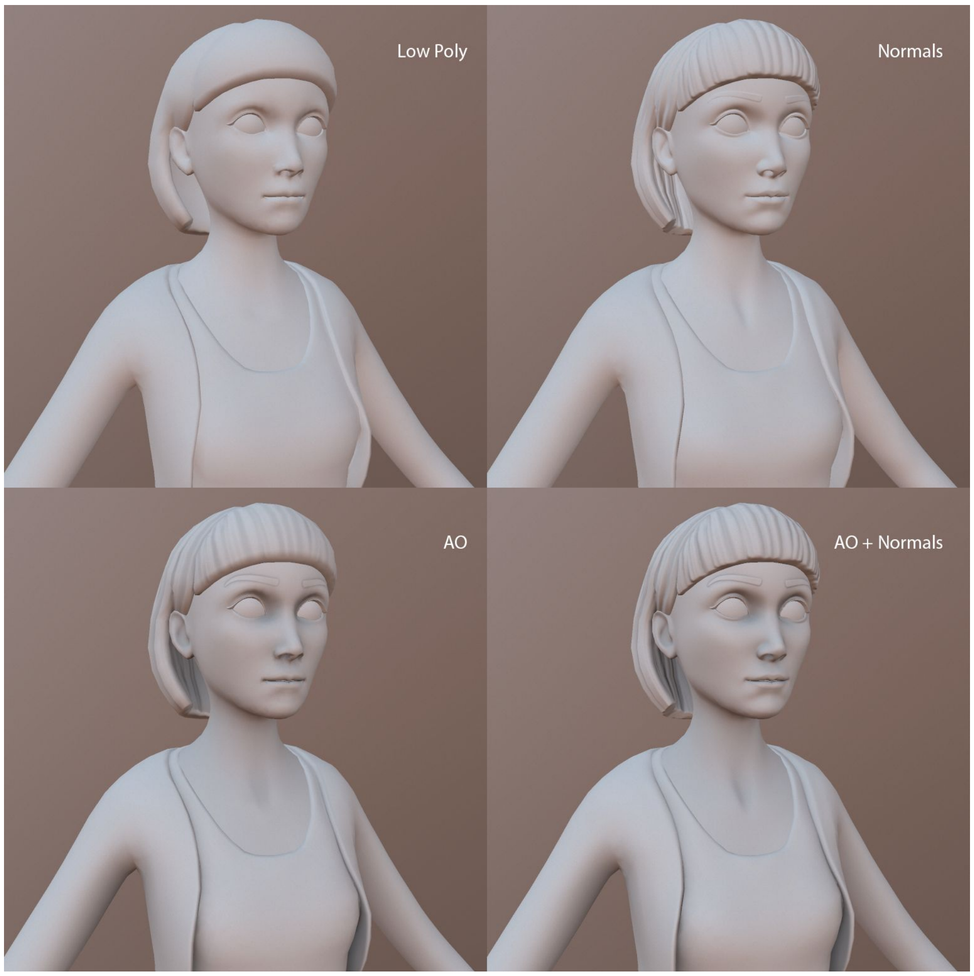


Shading

Depending on the style you are hoping to achieve, the techniques used can widely vary. For instance, on the simple side, using a toon shader might not require textures or UV's at all because the output can solely be a shader based on how light interacts with geometry. Conversely, developing a Physically Based Rendering (PBR) character is more complex and will require optimized UV's and many more textures including albedo, normal and roughness maps.

Shading Process Overview

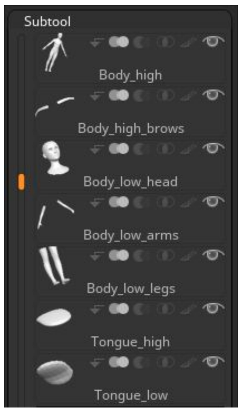
1. Export FBX high and low poly meshes for Marmoset (ZBrush)
2. Bake maps (Marmoset)
3. Texturing (Mudbox, Photoshop)
4. Shading networks (Marmoset Toolbag/Real-time Engine)



Baking

If the character hierarchy is set up according to its specifications, Marmoset Toolbag will allow you to bake details without too much extra work. There is a lot of great information out there on how to wield the power of Marmoset Toolbag so here we'll only focus on a few basics to get you up and running to bake maps.

Follow the proper naming convention: region_meshtype_regionDescription. Please note that you may have multiple low and high meshes that could make a more complex baking set. Each file should represent one texture file. So, if your outfit and skin have different textures maps, save one file with all your skin geometries and another with all your outfit geometries. Below are some examples of possible baking sets. To export from ZBrush, simply use the FBX ExportImport tool making sure the SNormals option is ticked. You may use any software package you'd like for this process, but we use ZBrush because it can handle very high poly count high resolution geometries without getting bogged down.



Body_high

Body_high_brows

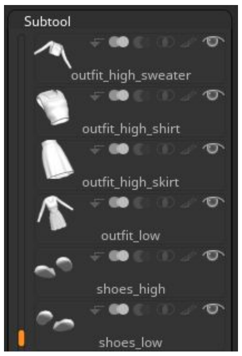
Body_low_head

Body_low_arms

Body_low_legs

Tongue_high Shoes_low

Tongue_low



Outfit_high_sweater

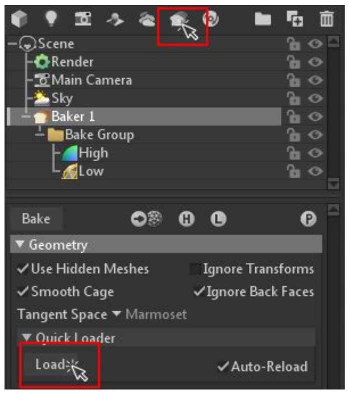
Outfit_high_shirt

Outfit_high_skirt

Outfit_high_shoes

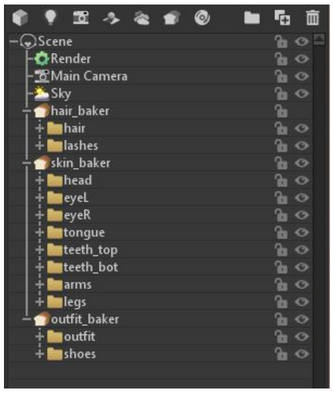
Shoes_high

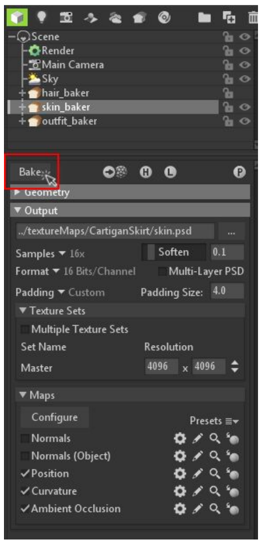
Loading your meshes into Marmoset is very easy. Press the bread icon to create a baking group and then press Load and choose a FBX file you created in the last step. Each baker group will generate maps one UV space.



✓Use Hidden Meshes Ignore Transforms
✓Smooth Cage ✓Ignore Back Faces
Tangent Space Marmoset
✓Auto-Reload

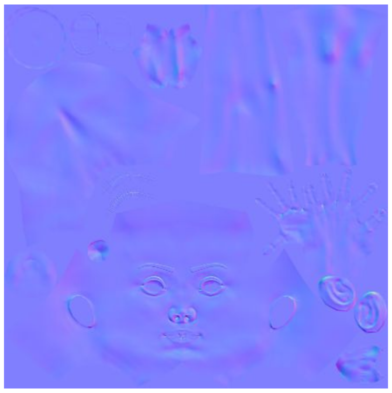
for

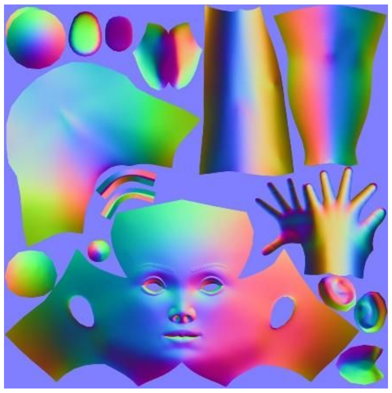


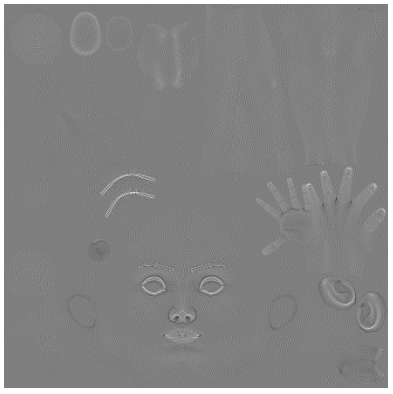


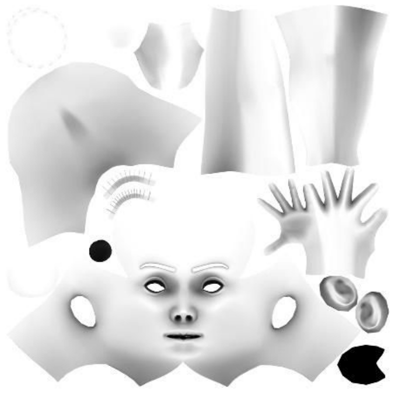
../textureMaps/CartiganSkirt/skin.psd
Samples 16x · Soften 0.1
Format 16 Bits/Channel · Multi-Layer PSD
Padding Custom · Padding Size: 4.0
Multiple Texture Sets
Set Name: Master · Resolution: 4096 x 4096
Normals
Normals (Object)
✓ Position
✓ Curvature
✓ Ambient Occlusion

Choosing which maps to use should be based on the style you want to achieve and so it is completely up to you. After choosing which maps you'd like to create and adjust the proper settings, you're ready to start baking.

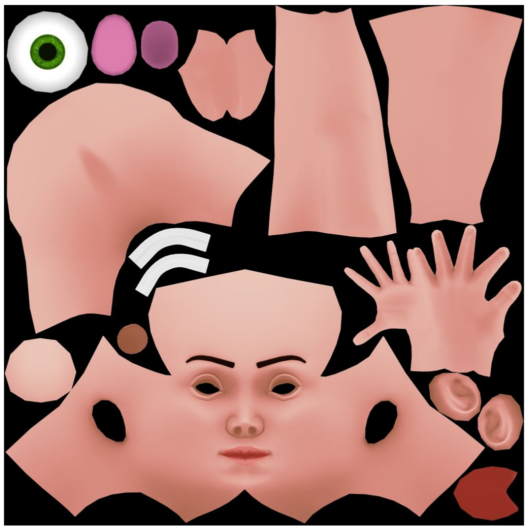






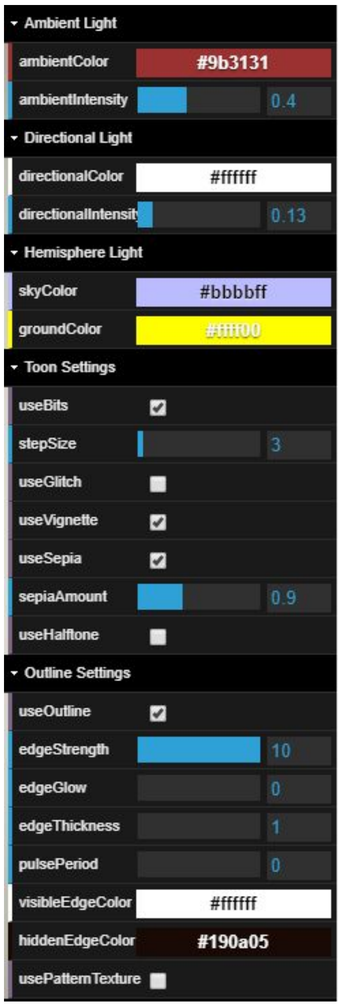


Baking maps sets you in a great place to start texturing. From there, you're able to add and paint in thinks like, skin tone, clothing fabric and anything else you'd like to add. This discipline is huge, and the details are dependent upon software chosen and aspired style to get into details here. Books and the internet will take you down many roads if you so choose to learn.



Toon Shading in Hadron

The visual effect of the avatar is a combination of controlled lighting, shadows and shaders.



Ambient Light	
ambientColor	#9b3131
ambientIntensity	0.4
Directional Light	
directionalColor	#ffffff
directionalIntensit	0.13
Hemisphere Light	
skyColor	#bbbbff
groundColor	#ffff00
Toon Settings	
useBits	✓
stepSize	3
useGlitch	☐
useVignette	✓
useSepia	✓
sepiaAmount	0.9
useHalftone	☐
Outline Settings	
useOutline	✓
edgeStrength	10
edgeGlow	0
edgeThickness	1
pulsePeriod	0
visibleEdgeColor	#ffffff
hiddenEdgeColor	#190a05
usePatternTexture	☐

The first state uses lights that are a combination of ambient, directional and hemisphere that helps cast shadows and smooth features in the avatar by washing it out slightly. The color, intensity and position of the lights can be adjusted.

The second stage is a step shader that reduces the number of colors in different areas of the avatar and applying the new color palette to the avatar. The step shader can be adjusted to allow between 2 and 15 colors. The shader itself allows more steps if necessary but this smaller range seems to work well with models.

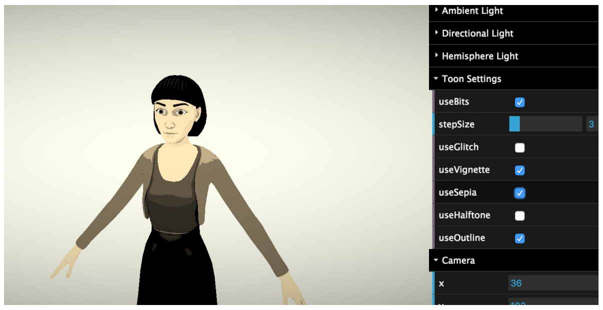
The third stage is a sepia shader that has been adjusted to let some of the colors from the second stage to show while dampening them to give a softer appearance.

The fourth stage is an outline shader that finds the outer edges of the mesh in the avatar and then adds a computed outline to the avatar.

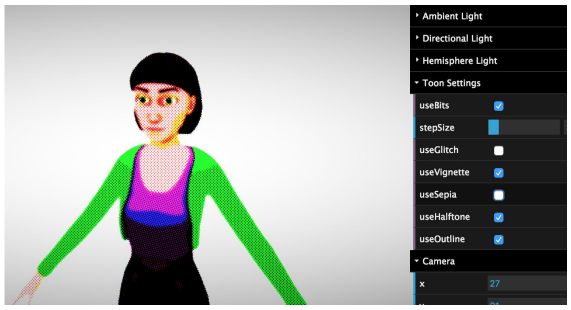
There are other effects that can be added to this stack but these are the four primary steps. The demo page has a GUI that can tweak the output to get the desired appearance.

Our plan is to continue extending ACTR so it sends these values with an avatar and let designers tweak settings using this demo GUI. This would help them customize the effects for their avatar.

When the toon shaded avatar is loaded the mesh and materials in the avatar are recreated on the fly. The materials are swapped with others that force a flatter look, remove any shininess and applies custom specular, shininess and bump values. This is technically the first step.



Ambient Light	
Directional Light	
Hemisphere Light	
Toon Settings	
useBits	✓
stepSize	3
useGlitch	☐
useVignette	✓
useSepia	✓
useHalftone	☐
useOutline	✓
Camera	
x	36



Ambient Light	
Directional Light	
Hemisphere Light	
Toon Settings	
useBits	✓
stepSize	
useGlitch	☐
useVignette	✓
useSepia	☐
useHalftone	✓
useOutline	✓
Camera	
x	27

Rigging

The characters in our initial release will be a joints-only approach. While blendshape and blendshape/joint hybrid rigs have many great functionalities, it currently isn't supported as we plan on this being an option after future development.

In theory, you should be able to use any rigging system you'd like, in any software you'd like. If the joint hierarchy naming convention matches ours and you export the same file types as us (we currently support glTF only). Please know that if you do so, you'll be blazing your own trail, so support will be either limited or non-existent.

Rigging Process Overview

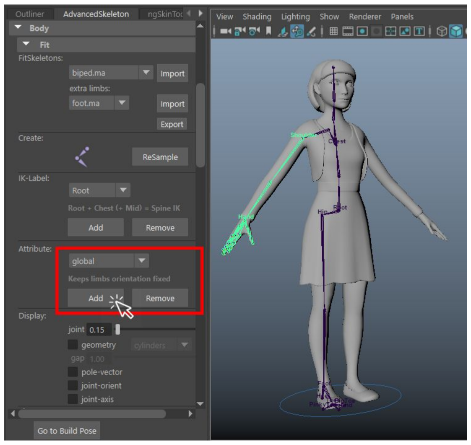
01. Body (Maya/Motionbuilder)
    a. Advanced Skeleton: bone placement, custom attributes, build settings, build
    b. Skinning (ngSkinTools)
    c. Pose-space deformers (if compatible with real-time engine)
    d. Mocap retarget rig (if using mocap)
02. Face (Maya)
    a. Joint Placement
    b. Control System
    c. Naming Convention
    d. Skinning (ngSkinTools)
03. Simplifying the Rig: Real-time Engine Compatible
    a. ActorTools (Braverabbit): Create rig constrained to the Advanced Skeleton rig
    b. Delete unnecessary joints
    c. Zeroing-out facial joints
04. Publish animation rig
05. Publish engine rig
06. Assemble and test in real-time engine

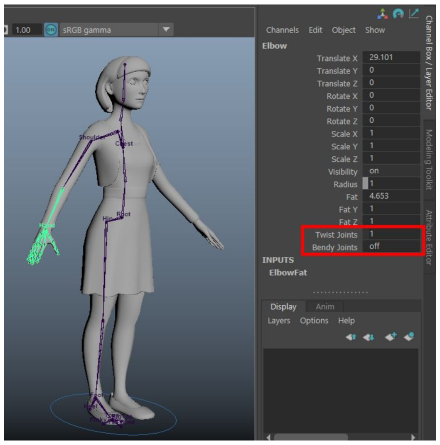
Tools

There are many auto-rigging tools from which to choose. Advanced Skeleton offers great functionality and has been a very smooth experience because of they level of support they offer. For those reasons it's become an essential tool in our pipeline. Braverabbit's ActorTools is also an important tool that ensures that our real-time character will work in engine. We also heavily use ngSkinTools and Michael Comet's "Comet Scripts." We will be giving a brief overview of Advanced Skeleton and Braverabbit but won't be covering ngSkinTools or Comet Scripts in much detail. To get a more in-depth understanding feel free to check out their websites as well as watch tutorials on YouTube.

Setup (Body)

Advanced Skeleton by default has a few attributes on the build skeleton that we want to change. Items to change include, twist joints, bendy joints and inbetween joints. For the elbow and shoulder there should be one twist joint and bend joints turned off. The hip and neck will have zero twist joints and bendy joints off. Lastly, the root and spine1 joints have a setting of one for inbetween joints. Optionally, you may add a global attribute to the shoulder and head joints. Many animators love this feature, so we include it. Settings need to happen per joint as you can't change multiple joints at a time.



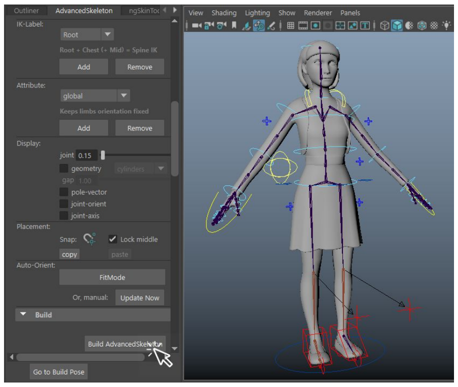


Elbow

Translate X	29.101
Translate Y	0
Translate Z	0
Rotate X	0
Rotate Y	0
Rotate Z	0
Scale X	1
Scale Y	1
Scale Z	1
Visibility	on
Radius	1
Fat	4.653
Fat Y	1
Fat Z	1
Twist Joints	1
Bendy Joints	off

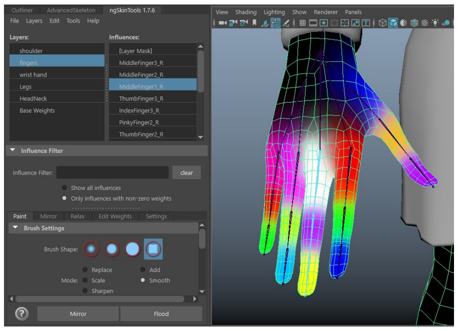
INPUTS
ElbowFat

Build (Body)

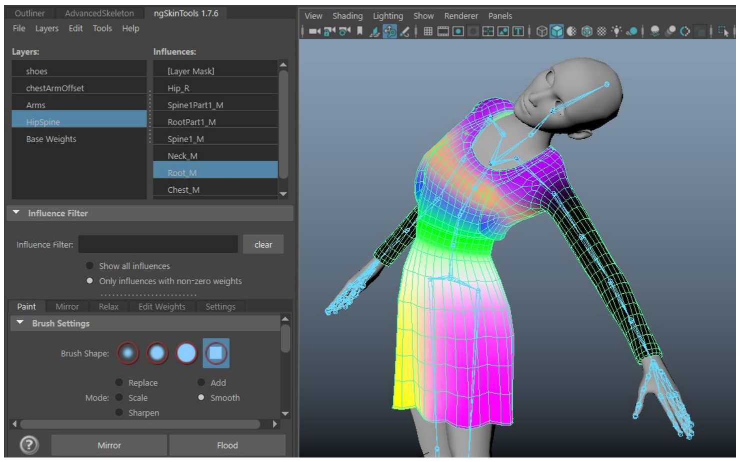


After adjusting and adding our attributes we can move on to the build phase. Click the Build AdvancedSkeleton button and voilà, there you have it. Now we can move on to skinning.

Skin Weights

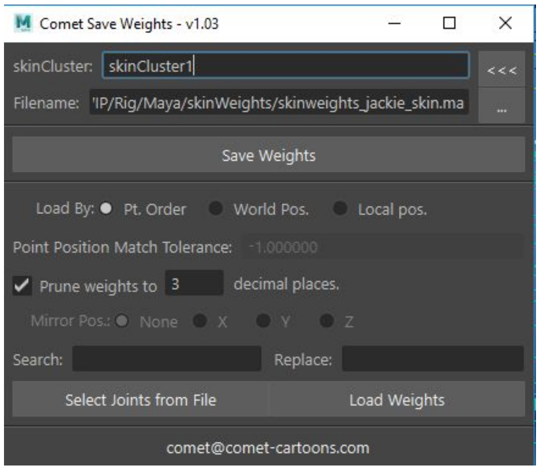


If you've never used ngSkinTools to paint weights in Maya you are in for a treat! Those familiar with skinning in Maya will know the pains of either forgetting to lock weights while painting or the tedium of locking and unlocking influences while painting regions of a mesh. ngSkinTools brings intuitive functionality into Maya that behaves much like layers do within Photoshop utilizing opacity and masking. It also allows you to mirror weights in any pose and handles smoothing extremely well.



When painting weights, it's generally a good practice to have range of motion (ROM) animation on the character. That way you are able to see how the skin weights behave in motion and adjustments can be made in context. You may either make you own or use the AnimationTester tool under the Tools tab in Advanced Skeleton.

Whether you're taking our base model and making shape adjustments, or the model has updated UV's, being able to swap out your character mid-process is something that is important to be able to do. While there are multiples ways of doing this, the open-source toolset, Comet Scripts has some amazing functionalities including a handy weight saving tool. The beautiful thing about the script is that it's based on vertex number, so it is 100% accurate. Say you needed to delete half your model and mirrored it over causing your vertex ID's to change, no problem. You may also load it based on world or local space.


Comet Save Weights - v1.03

skinCluster: skinCluster1

Filename: 'IP/Rig/Maya/skinWeights/skinweights_jackie_skin.ma

Save Weights

Load By: [✓ Pt. Order] World Pos. Local pos.

Point Position Match Tolerance: -1.000000

[✓] Prune weights to 3 decimal places.

Mirror Pos.: [✓ None] X Y Z

Search: Replace:

Select Joints from File Load Weights

comet@comet-cartoons.com

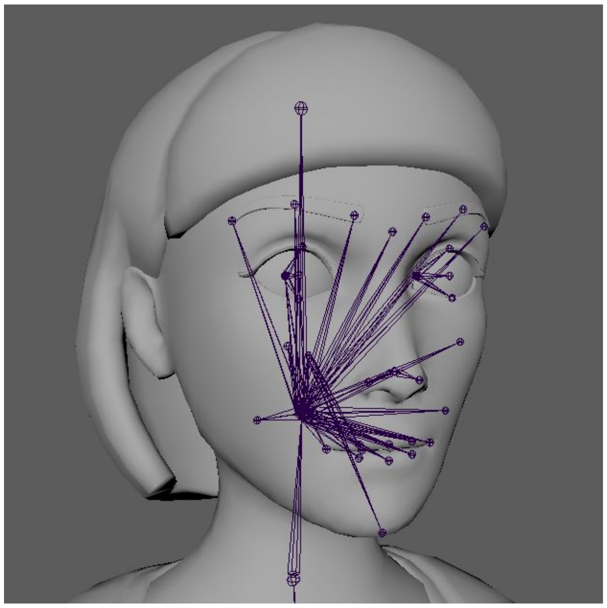
Facial Rigging

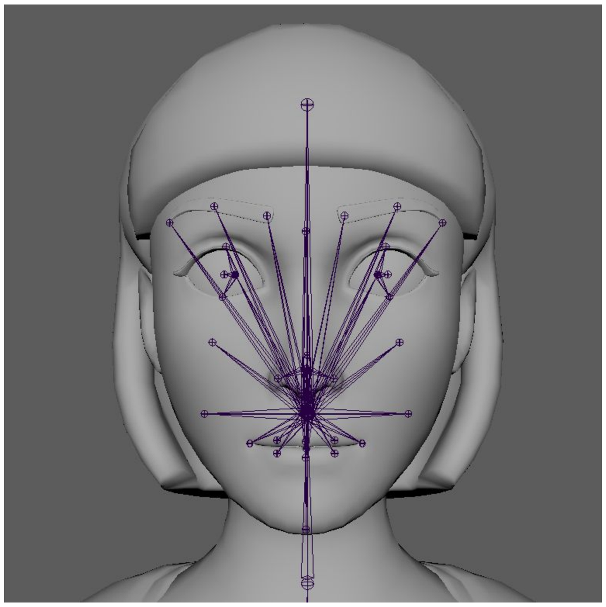
To follow will be two ways to approach facial rigging. First, we'll show you how to create our method from scratch. At a future time, the process of taking our system and applying it to a custom character will be shown. Keep in mind, you are not bound by our system. If you would like to create your own facial rig, you may do so and attach it to either our body rig or a body rig of your creation. If you do choose to create your own facial system, then you would need to create the necessary animations files with it.

Joint Placement

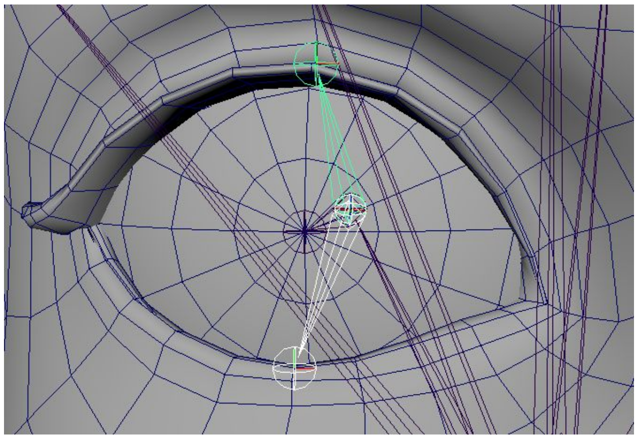
Our facial system has 37 joints in it. 30 of those joints will have skinning information and the other 7 are used for generating desired orientation axes. Optionally, those 7 joints may be deleted in the real-time rig. We will cover that in the *Simplifying the Rig: Real-time Engine Compatible* section. For now, let's focus on the 37 joints.

Our placement of joints follows the thinking that to create a curve you need a minimum of three points. Since the rig is optimized for real-time we go with that minimum number for important deforming regions (brows and sides of the mouth). The eyelids are somewhat of an exception as there is only one joint per lid. While the corners of the eye aren't skinned, they account of the other two points to make that curve.





An important thing to note about rigging the eyes is to align your end joint to the center edge loop of each lid. This is where the earlier note about matching edge loop count comes in handy. Make sure to test out how the joints line up when rotated down into the blink position and make any adjustments to ensure that the end joints stack on top of one another as best you can. This will ensure a nice seal when blinking occurs because the edge loops should match from top to bottom.

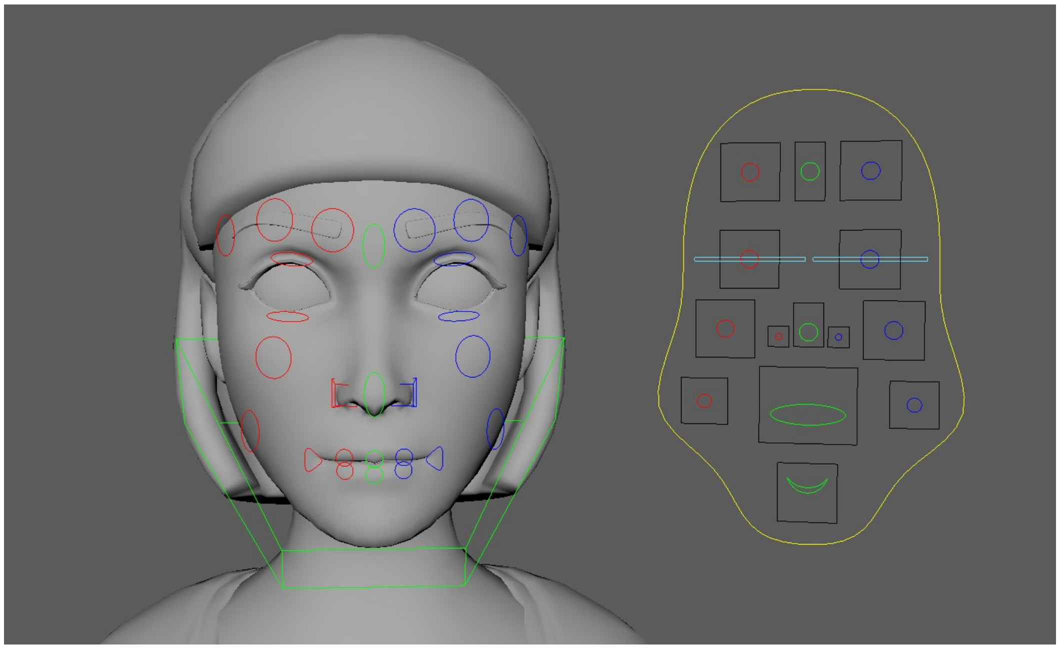


Control System

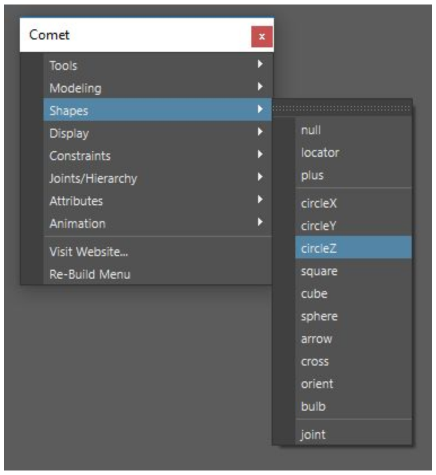
The methods used below are completely optional. You may use any type of control system you'd like because the joint data is what will be baked down and used in the end. How you get there is up to you. The method we show gives animators multiple options of how to animate. Since people have varying preferences and styles it's nice to have more than one option. This system allows the animator to use the GUI and on-face controllers individually or a combination of the two. To start, we begin with making the two individual systems. After those are setup we build the system that will bridge the two.

GUI

The GUI is a common driven key controller box. Each box corresponds to a region of the face and the controller has translation limits most often from -1 to 1 in X and Y. The exception to this is the main mouth controller that drives some viseme shapes. The attributes are limited to being controlled in the Attribute Editor. Quick tip: If you rotate the controller box regions on the right side of the face you will get mirrored behavior when moving two symmetrical controllers simultaneously.



On-face Controllers

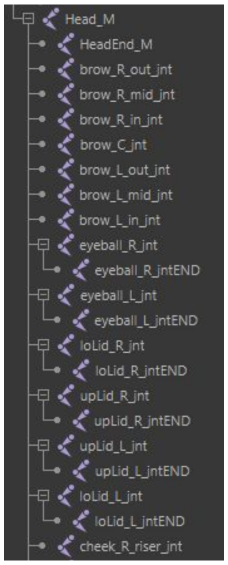


To make the on-face controllers, first decide what type of controller you want and where. In this case, we are using two types of primitive shapes, a circle and cube. Now we need to match the placement of the curves to the joints. You could do this by hand, but that adds more tedium. Comet Scripts has a tool for this! Select all the joints that you'd like to have a circle controller for. Then, in the toolbar go to shapes->circleZ. Z corresponds to the Z axis of the joint and will align the newly created controller accordingly. Do this same step for the nostrils and jaw but choose the cube shape. Don't worry about the transform data on the control curve, we will take care of those pesky numbers with a zero group in the system bridging stage. Now that we have our controllers placed it's time to shape and size them. Move the curve CVs to get your desired shapes.

Naming Convention

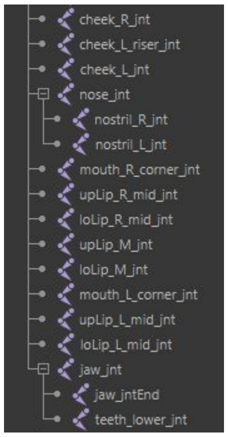
Joint Naming Convention

```
Head_M
  brow_R_out_jnt
  brow_R_mid_jnt
  brow_R_in_jnt
  brow_C_jnt
  brow_L_out_jnt
  brow_L_mid_jnt
  brow_L_in_jnt
  eyeball_L_jnt
    eyeball_L_jntEnd
  eyeball_R_jnt
    eyeball_R_jntEnd
  loLid_R_jnt
    loLid_R_jntEND
  loLid_L_jnt
    loLid_L_jntEND
  cheek_R_riser_jnt
  cheek_R_jnt
```



```
Head_M
  HeadEnd_M
  brow_R_out_jnt
  brow_R_mid_jnt
  brow_R_in_jnt
  brow_C_jnt
  brow_L_out_jnt
  brow_L_mid_jnt
  brow_L_in_jnt
  eyeball_R_jnt
    eyeball_R_jntEND
  eyeball_L_jnt
    eyeball_L_jntEND
  loLid_R_jnt
    loLid_R_jntEND
  upLid_R_jnt
    upLid_R_jntEND
  upLid_L_jnt
    upLid_L_jntEND
  loLid_L_jnt
    loLid_L_jntEND
  cheek_R_riser_jnt
```

```
cheek_L_riser_jnt
cheek_L_jnt
nose_jnt
  nostril_R_jnt
  nostril_L_jnt
mouth_R_corner_jnt
upLip_R_mid_jnt
loLip_R_mid_jnt
upLip_M_mid_jnt
loLip_M_mid_jnt
mouth_L_corner_jnt
upLip_L_mid_jnt
loLip_L_mid_jnt
jaw_jnt
  jaw_jntEnd
  teeth_lower_jnt
```



```
cheek_R_jnt
cheek_L_riser_jnt
cheek_L_jnt
nose_jnt
  nostril_R_jnt
  nostril_L_jnt
mouth_R_corner_jnt
upLip_R_mid_jnt
loLip_R_mid_jnt
upLip_M_jnt
loLip_M_jnt
mouth_L_corner_jnt
upLip_L_mid_jnt
loLip_L_mid_jnt
jaw_jnt
  jaw_jntEnd
  teeth_lower_jnt
```

More to come about the control system's naming convention concerning offset and zeroing nodes.

Bridging the Two Controller Systems

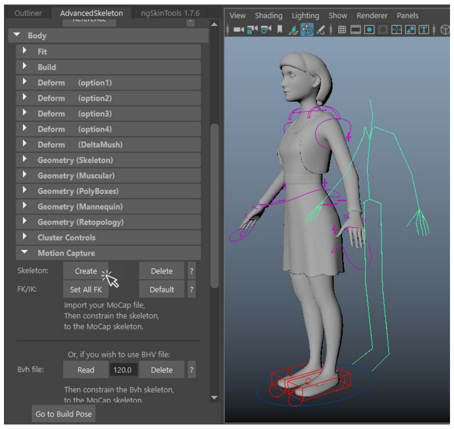
Now that we have our GUI and On-face controller systems it’s time to link everything up so we can start driving the placement of the joints. So what controls what? In short, the on-face controllers directly control the joint through point and orient constraints. Each on-face controller has nodes above it that are controlled with driven keys by the GUI system. This allows the on-face controllers to move without transform data being plugged into it, allowing for addition animation on the actual on-face controller for tweaking facial performance if/when necessary. Side note: by using point and orient constraints instead of parent constraints it becomes more animator friendly because it will have higher performance. For instance, if you have a parent constraint and you only translate, orientation will still be calculated and vice-a-versa. If you have many controllers you may begin to see how it could slow things down. By splitting it up it you only calculate what is being used and don’t overtax the machine with unnecessary calculations.

More specifics to come about how to setup up driven keys with offset nodes.

As a final note to this chapter, while we recommend the tools we’re accustomed to this doesn't mean they’re the only tools you can use. The final output is the key, so the list of tools we suggest you use are just one of the many production paths to the goal.

Motion Capture Retarget Skeleton (Optional)

Retargeting animation to Advanced Skeleton is straightforward using its built-in Motion Capture tools. To create a skeleton that drives the character rig simple go to the Motion Capture tab in the Body section and press create. Then use Maya's HIK system to characterize the skeleton so you're able to retarget animation. Now the main rig has a skeleton linked to it that may receive mocap data. If you'd like to retarget using Motionbuilder at this point, skin your mesh to the "MoCap" skeleton (for previewing purposes) and export the skeleton and mesh as an FBX file that can be imported into Motionbuilder. We will show how to bring animation back from Motionbuilder into Maya and cover best practices for baking that data on to the main rig in the following Animation section.

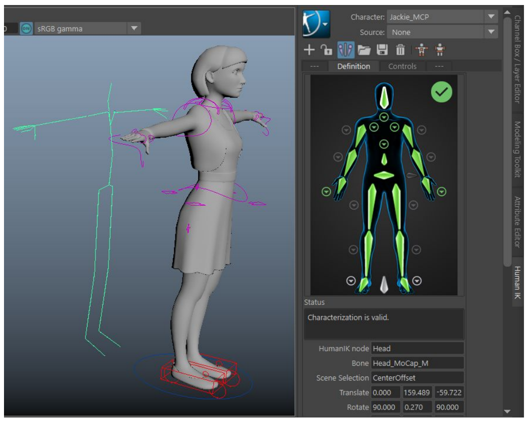


Import your MoCap file,
Then constrain the skeleton,
to the MoCap skeleton.

Or, if you wish to use BHV file:

120.0

Then constrain the Bvh skeleton,
to the MoCap skeleton.



Character: Jackie_MCP
Source: None

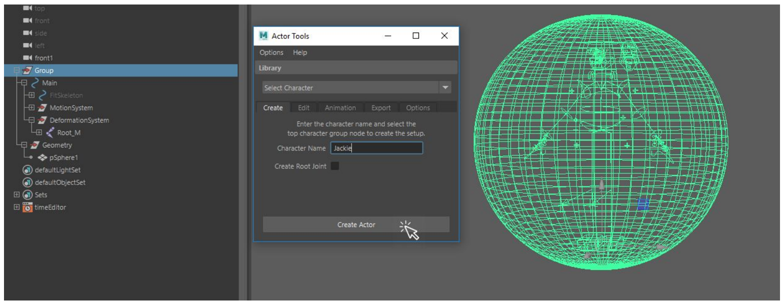
Characterization is valid.

HumanIK node: Head
Bone: Head_MoCap_M
Scene Selection: CenterOffset
Translate: 0.000 159.489 -59.722
Rotate: 90.000 0.270 90.000

More to come regarding how to setup the Advanced Skeleton rig to drive the final export skeleton/rig from ActorTools.

Simplifying the Rig: Real-time Engine Compatible

Why simplify? To make our lives easier and to avoid headaches while trying to make node behaviors match from one package to another! This step uses ActorTools from Braverabbit and duplicates the existing skeleton and makes is a child of the existing rig. This ensures rig capability to the engine and makes the animation export process much simpler, hence faster. This stage is techy and has a lot of moving pieces so make sure to take close attention.

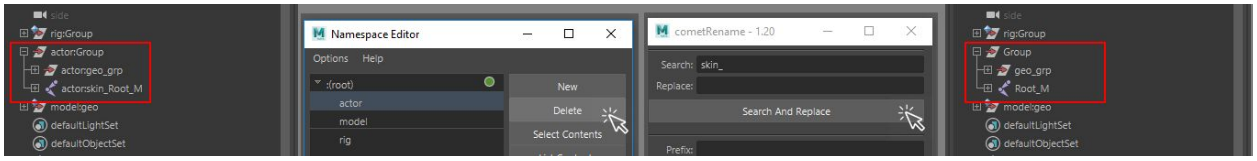


Enter the character name and select the
top character group node to create the setup.

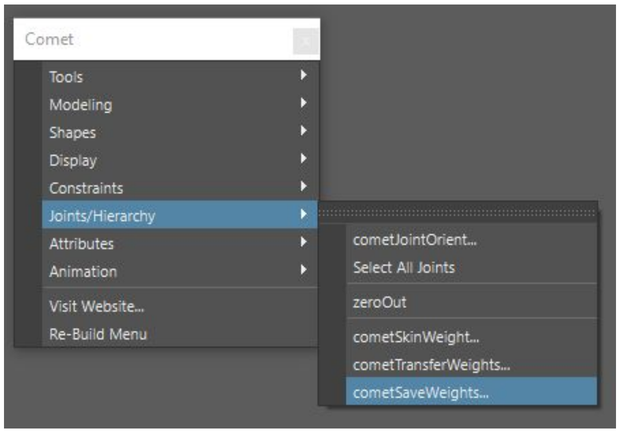
Character Name: Jackie

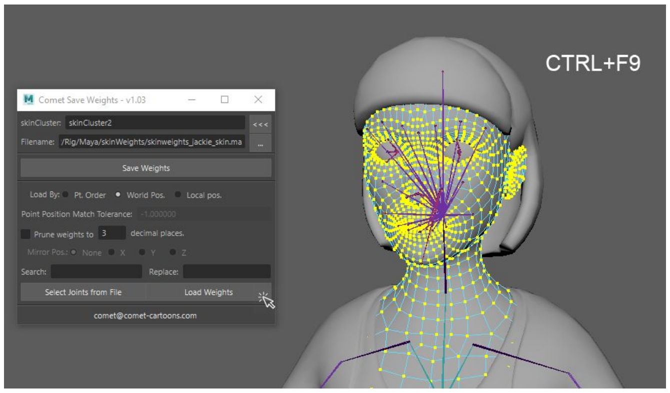
From the date of the writing of the document (August 2018) there is one bug we found while using ActorTools with Maya 2018. The bug is that if you open a file and try to create the export rig it won't read the skinCluster nodes. This will give you an error and not let you proceed when you click the Create Actor button. There is a simple workaround for this. It may seem like a lot but what ActorTools gives you is worth the little extra work. Start by saving your weights with Comet Scripts as shown earlier, unbind ALL your geometry, unload the referenced model, create a sphere and bind it to the skeleton. Now when you click Create Actor and Continue, it works! A new scene will be created that references the two files that were made, a rig file and an actor file. The rig file is the Advanced Skeleton rig and the actor file is the newly made skeleton that is driven by the Advanced Skeleton rig.

We need to clean up the scene by matching the Advanced Skeleton naming convention, so we can skin our character geometry to the actor rig skeleton. Start by referencing your published model file. Next, delete the "actor" namespace by deleting it in the Namespace Editor by clicking "delete," then "Merge with Parent." After that, select the root joint and select the hierarchy below. Using the cometRename script, delete the skin prefix.



Using the Come Save Weights script to transfer skin weights streamlines the process. Begin by loading the associated skin weights file that you saved when exporting your weights. Click the "Select Joints from File" button and select the corresponding geometry. Then use the Maya "Bind Skin" tool. The weights will be garbage but that's okay. We just want a skinCluster node with all the joints from all of our previous rigging efforts. With the geometry selected, load the skin cluster. Next with the geometry selected, press CTRL+F9. This selects all the vertices of that geometry. Then press "Load Weights." Do this for all pieces of geometry until your character is fully skinned. Phew! Just one more step!




Comet Save Weights - v1.03

skinCluster: skinCluster2

Filename: /Rig/Maya/skinWeights/skinweights_jackie_skin.ma

Load By: Pt. Order [✓ World Pos.] Local pos.

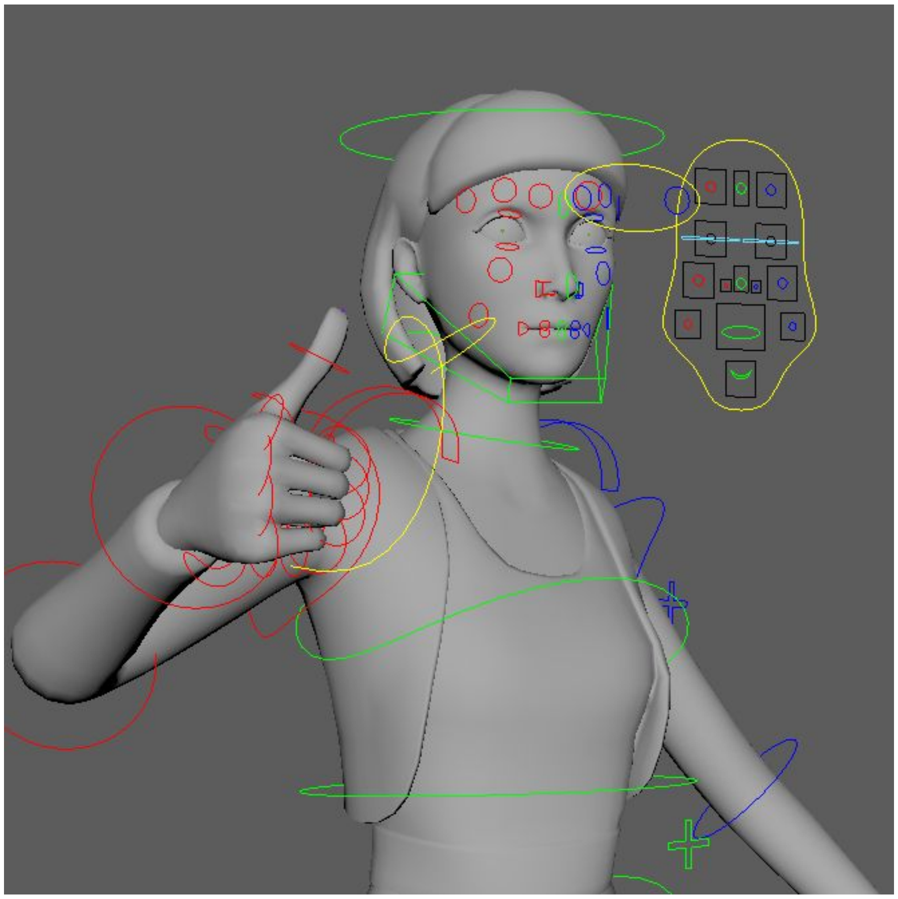
Point Position Match Tolerance: -1.000000

Prune weights to 3 decimal places.

Mirror Pos.: [✓ None] X Y Z

Search: Replace:

The only thing left to do is delete joints that don't have any skinning information on it. The reason for this is to minimize the amount of data in our animation files. Lesser the joints, lesser the data so, lesser the file size. Delete the following joints from the actor rig: HeadEnd_M, eyeball_L_jntEND, eyeball_R_jntEND, upLid_L_jnt, loLid_L_jnt, upLid_R_jnt, loLid_R_jnt, jaw_jntEnd, ThumbFinger4_L, IndexFinger4_L, MiddleFinger4_L, RingFinger4_L, PinkyFinger4_L, ThumbFinger4_R, IndexFinger4_R, MiddleFinger4_R, RingFinger4_R, PinkyFinger4_R, ToesEnd_L, ToesEnd_R. Woohoo! What's left is a fully ready character to be exported and brought into our real-time engine, Hadron!



Animation

Machine Learning may soon produce all the animation we'll ever need. Until then, keyframing and performance capture animation techniques will be used. Due to the amount of animations needed to build an ACTR compliant animation library, motion capture may help if you have access to a system. Keyframing can work just as well and, especially for characters that are not humanoid, like a bouncing desk lamp, will be better than MoCap. Regardless, the core animations are as follows.

Animation Process Overview

01. Body
    a. Fibonacci sequence
    b. Iconic gestures
    c. Idles/Neutrals
02. Face
    a. Fibonacci sequence
    b. Iconic gestures
    c. Idles/Neutrals
03. Mouth shapes (visemes). Please note that the amount of shapes needed is dependent upon the chosen language to be used.
04. Facial expressions
05. Build animation clips with proper naming convention
06. Setup base character
07. Integrate animation clips into scene
08. Export clips with Game Exporter Tool

Planning

More to come during animation phase.

Keyframing

More to come during animation phase.

Motion Capture

More to come during animation phase.

Integration with Real-time Engine

Integration Process Overview

01. Assemble rigged character with shaders
02. Environment setup
03. Lighting

More to come about the specifics and best practices of Real-time Engine integration.

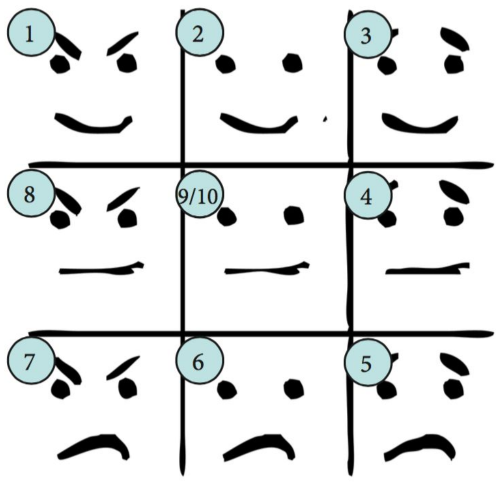
Pro Tips

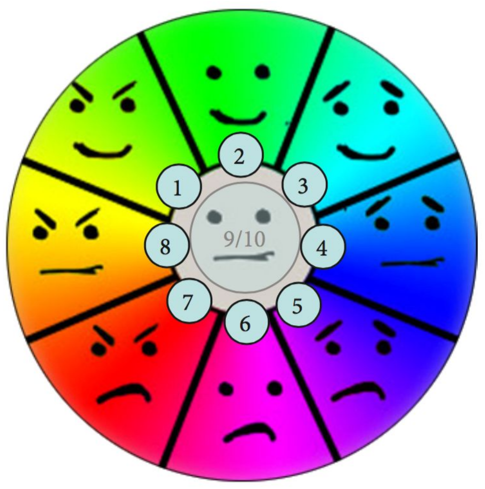
Tips to be covered in this section: general optimizations, etc.

ACTR Protocol

Function

The purpose of ACTR is to drive real-time animations that are contextually appropriate to arbitrary natural language input or output. This may include animating an avatar based on the output of a natural language processing system, driving a VR scene graph via voice control, or other purposes. Ultimately, ACTR generates real-time cinema from natural language strings. The following specifications detail animating avatars that are connected to chatbots.





The Library Preparation

We began with a breakout of 8 emotions (though this number can be different it has been convenient for us) and a single, nul / non-emotion. For a very similar model see Plutchik's wheel of emotions (which doesn't work as well for reasons beyond our scope in this explanation). We'll call these "emotion buckets." We do not name the emotions as it is both unimportant and can even be detrimental to the development process.

We then built eight 33-second animations. Each of the eight animations corresponded to the list of eight emotions. Two nul / non-emotion animations of the same duration were made, giving us a total of ten animations. Each of the 33 second animations were split into a Fibonacci sequence (1, 1, 2, 3, 5, 8, and 13 second durations). These were saved for later use, intended as a library for later real-time concatenation during the conversation with the avatar. The link animation is then assigned a name based on these values, something like d13_e7 for emotion number seven at 13 seconds. This is intended as a reference for the player to then interpolate with other animations.

(Please see the last page of this document for related patent applications)

The Real-time Operation

The NLP system produces a block of output text in human-readable format. In order to leverage the prepared ACTR library (above) we need a duration value and an affect value. Duration is the time it takes to speak that text and is calculated by assuming that one second equals 7 bytes. Second, the undetermined emotion (the affect or emotion of the output NLP text) is calculated by an external service (such as WordNet Affect, manually entered, or others) and the affect report is delivered to the PHP script that is monitoring MultiMedia tag generation. The script then produces a control file that roughly matches that emotion and duration without having repeated animations happening adjacent to one another.

Generating the Chain Animation

A block of output text is evaluated so as to determine two values

1. **The duration report**. Listed in seconds, this is based on the number of bytes if using a TTS system, or recording length, or whatever to determine how long it takes to speak it.
2. **The affect report**. Listed as an integer from 1-10, to correspond with our emotional model, that corresponds to the emotion number.

These two values are then used to determine the duration and emotion of the composed, or chained, animation. This chained animation is a sequence of the links mentioned above generated by interpolating between the end-values and start-values of successive link animations.

- Note: *This multimedia script simultaneously prevents recurring animations. An example of what needs to be avoided is if we have two responses that are each the same emotion and the same duration. So consider the possibility that the system produces a block of text that amounts to 9 seconds of emotion #2, then another block of text, on the subsequent output, that is also 9 seconds of emotion #2. We do not want the avatar to have precisely the same animation so the system might use 8+1/#2 to generate the first animation, but we do not want to run that same (8+1/#2) again, so we use instead anything but that. One option would then be 5+1+3/#2. This avoids repeats.*

Iconic Gestures

At particular moments that require particular emphasis we use Iconic Gestures to break this up and bring attention to the words being said such that the Iconic Gesture matches the duration and sentiment of what is being said. ACTR provides 50 default iconic gestures (see below) however there is a spectrum of integration / customisation which is usually gated by scope.

Additional Uses (Beyond Driving Avatars)

In the shortest description, the middleware responds to requests by returning information about the avatar, lighting, camera, scenes, etc. The system is extensible so the standard behavior is not all it can do but can be easily extended.

The middleware receives either audio and textual phrases which are parsed for contextual information and JSON responses are returned. At its heart, the middleware is a REST API that accepts remote commands that can either be driven by a user interaction, a bot interaction or an avatar requesting a behavior. The server then responds in a standardized way despite how it was directed. The objects that are returned are identical in format which type flags that determine the type of object and target.

If the phrase “Pass me the butter” is sent to our servers, that text is analyzed for sentiment and content. The text is broken into phonemes and animation sequences are created that define mouth shapes, mouth shape changes, start time for the animation, duration, intended character. The start time for animation can be defined too in cases where there is an initial silence/pause. For example, the spoken words may start at 0.30 seconds so that time is returned in the animation response and the individual animations are compensated to match. The following JSON is representative of a sequence you may receive. This is a simplified object, a typical response could contain dozens or hundreds of similar objects based on complexity and time and also control information that is not shown.

```
{
control_group: “mouth_neutral_d500",
timestamp: 1463760818.7082,
settings: {
power_on: false,
level: 0,
color: “”,
uri: “”,
order: 5,
animation_track: “mouth”,
duration: 97,
start_delay: 0,
animation_loop_fill: true,
animation_loop_sequence: false
},
control_group_human_readable: “mouth_neutral_d500"
},
```

The control_group in the example is a mouth animation with a neutral emotion and a maximum duration of 500 milliseconds. The actual duration in this example is 97 milliseconds with no start delay and is the 5th animation in the current sequence.

You’ll also see that the animation object is a generic type that is able to receive other types of data such as lighting which has a master on/off, level and RGB color. The light is considered to be a generic object that has its place and direction set in the avatar scene.

The uri field can be used to provide a link to extra media and could be used to trigger the playing of a youtube video as one possible example.

Animation is not limited to mouth but can also direct head and body.

```
{
```

```
control_group: “head_e10_d5000”,
timestamp: 1463760818.721,
settings: {
start_delay: 0,
animation_loop_fill: true,
animation_loop_sequence: false,
order: 1,
animation_track: “head”,
duration: “5000”
},
control_group_human_readable: “head_e10_d5000"
}


{
control_group: “body_e10_d5000”,
timestamp: 1463760818.721,
settings: {
start_delay: 0,
animation_loop_fill: true,
animation_loop_sequence: false,
order: 1,
animation_track: “body”,
duration: “5000”
},
control_group_human_readable: “body_e10_d5000"
}
```

These examples show a command for a stock sequence running for 5000 milliseconds. They are considered to be running alongside mouth so the start order of 0 for body and head means they are the first animations and start running alongside the mouth order 0. The combined animation time of head, body and mouth will always match.

If the control was for a light, the command would be similar to this.

```
{
control_group: “kitchen”,
timestamp: 1463760818.7082,
settings: {
power_on: true,
level: 80,
color: “FFFFFF”,
uri: “”,
order: 0,
animation_track: “lights”,
duration: 97,
start_delay: 0,
animation_loop_fill: true,
animation_loop_sequence: false
},
control_group_human_readable: “kitchen”
}
```

Control groups are either built in groups like kitchen, window or they can be specific objects that have been defined in either global or user contexts. When these groups or objects are detected, the server attempts to discover intent and pass a command structure that describes what the user wanted and what should be acted upon.

The control group represents the target light and in this case, it is a kitchen light that is being turned up to 80% and is white (RGB FFFFFF) and a start delay of zero. You can add a delay and choose to bring the light to the target level for example. You can also receive a command that targets ALL lights in a room which common light characteristics.

The middleware recognizes things like “go to the kitchen”, “move to the living room” and returns an animation sequence that specifies that the object type is camera and tells the avatar the location. You can also so “pan left”, “pan right” and other commands but they behave as relative movements that the avatar chooses to implement how it sees fit.

You can say “change the background to mountains” and it will send a command sequence to the avatar to change the scene.

The middleware recognizes commands like “reset world” which tells the avatar to reset to a known point. When this happens, the server sends the avatar a series of commands ranging from chosen camera, lighting options, initial avatar state and other data.

At a deeper level, we can enable sentiment analysis which looks at text and/or images and determines the mood of the user and sends this emotion data to the avatar in two forms. The first is that a control group can have an embed emotion, neutral in the above case, but it can also return a JSON array that gives the emotion in multiple vectors over time. The emotion is not just the current emotion but is a composite of text and image sentiment over time.

The middleware is able to expand what it knows and uses NLP to discover intent and generates responses. If it is unsure it can respond in any number of ways determined by the chat bots character.

Obviously, animating VR (or AR components) is a big topic, so let’s set that aside for the moment and return to our primary focus: animating the avatar.

ACTR JSON Format Overview

The ACTR JSON format is designed to allow the communication of avatar control data to an ACTR capable client for the purpose of rendering synchronized mouth movement, emotional expression, and iconic gestures.

Required animations:
- for each of type = [head|body]
    - 10 emotions labeled 1 through 10, 9 and 10 are alternate neutrals
    - durations (in milliseconds) 1000a, 1000b, 2000, 3000, 5000, 8000, 13000 for each emotion (there are two with duration = 1000)
- result is 70 animations each for head and body
- labels in the model are {type}_e{emotion}_d{duration} or iconic_{name}_d{duration}
    - e.g. head_e4_d1000a, body_e5_d2000, iconic_cheers_d4000
- standard iconics are: thumbsup, onethumbup, cheers, wave, etc
- visemes
  - fall under "mouth" label
  - all characters must have a neutral mouth (e.g. mouth_neutral_d500)
  - all viseme animations are 500ms long with peak shape at 250ms to allow animation blending
  - each language will have a standard minimum set of visemes with its own phoneme to viseme mapping

Base English language visemes

Visemes for tongue motion sounds are collapsed into root sounds:

o	[o]
a	[a]
c	[cdgknrsy]
e	[e]
f	[fv]
m	[m]
u	[u]
neutral	

Other Details

- each sequence is targeted to a specific scene element (i.e. character name)
- each sequence is a combination of speech elements with head and body emotion animations
- all the animations will be presented to fill the total duration
- all sequences delivered in the same server response have the same start time
- sequences with a start_delay must have their playback delayed by the indicated amount
- speech visemes and entries playback is always synchronized
- phoneme string is rendered in the International Phonetic Alphabet (http://en.wikipedia.org/wiki/International_Phonetic_Alphabet). See http://upodn.com/phon.asp for example transcodings.
- viseme string is mapping of phoneme to available viseme animations
- entries of different types (e.g. body and head) in the entries array are rendered in synchronization, according to order, from the start time

Elements

```
{ "sequence": {
        "element": string,                  //name of the scene element that is
                                            // the target of this sequence
        "start_delay": integer,             //start time delay from "0" for this sequence(millisecs)
        "total_duration": integer,          //duration of all animations together (milliseconds)
        "speech": string,                   //Human readable speech text
        "phoneme": string,                  //International Phonetic Alphabet version of speech
        "visemes": array of strings,  //array of viseme animation names
                                            //mapped by server from phonemes
        "entries: [                         //array of animation entries
                {
                        "type": string,               //entry type [head|body|iconic]
                        "label": string,              //label used in the model
                        "description": string,        //human readable helper description
                        "duration": float,            //duration of this entry in seconds.
                        "order": integer,             //playback order of this entry
                                                      //(preserved within type)
                        "info": {                     //non-common fields, custom per type
                                "type": string,       //should match parent type
                                "emotion": string, //emotion label, typically 1 through 10
                                "duration": string, //duration label
                                                      // [1000a,1000b,2000,3000,5000,8000,13000]
                                "mouth": float,       //used for custom mixing of mouth variable
                                "eyes": float         //used for custom mixing of eye variable
                        }
                }
        ]
        }
}
```

```
/*
EXAMPLE 1: greeting with neutral body and head responses
triggers the following sequences in Unity:

melissa_mouth_neutral_500
melissa_mouth_e_500
melissa_mouth_o_500
melissa_mouth_u_500
melissa_mouth_neutral_500
melissa_mouth_m_500
melissa_mouth_a_500
melissa_mouth_c_500
melissa_mouth_neutral_500

melissa_head_e9_d1000a
melissa_head_e10_d1000a
melissa_head_e10_d2000

melissa_body_e9_d2000
melissa_body_e9_d1000b
melissa_body_e9_d1000a

*/

{ "sequence": {
        "element": "melissa",
        "total_duration": 4000,
        "start_delay": 0,
        "speech": "Hello, Mark.",
        "phoneme": "həlo, mɑrk",
        "visemes": [ "e", "o", "u", " ", "m", "a", "c"],
        "entries": [
                {
                        "type": "head",
                        "label": "head_e9_d1000a",
                        "description": "neutral1",
                        "duration": 1000,
                        "order": 1,
                        "info": {
                                "type": "head",
                                "emotion": "9",
                                "duration": "1000a",
                                "mouth": 0.5,
                                "eyes": 0.5
                        }
                },
                {
                        "type": "head",
		"label": "head_e10_d1000a",
		"description": "neutral2",
		"duration": 1000,
		"order": 2,
		"info": {
			"type": "head",
			"emotion": "10",
			"duration": "1000a",
			"mouth": 0.5,
			"eyes": 0.5
		}
	},
	{
		"type": "head",
		"label": "head_e10_d2000",
		"description": "neutral2",
		"duration": 2000,
		"order": 3,
		"info": {
			"type": "head",
			"emotion": "10",
			"duration": "2000",
			"mouth": 0.5,
			"eyes": 0.5
		}
	},
	{
		"type": "body",
		"label": "body_e9_d2000",
		"description": "neutral1",
		"duration": 2000,
		"order": 1,
		"info": {
			"type": "body",
			"emotion": "9",
			"duration": "2000"
		}
	},
	{
		"type": "body",
		"label": "body_e9_d1000b",
		"description": "neutral1",
		"duration": 1000,
		"order": 2,
		"info": {
			"type": "body",
			"emotion": "9",
			"duration": "1000b"
		}
```

```
                },
                {
                        "type": "body",
                        "label": "body_e9_d1000a",
                        "description": "neutral1",
                        "duration": 1000,
                        "order": 3,
                        "info": {
                                "type": "body",
                                "emotion": "9",
                                "duration": "1000a"
                        }
                }
        ]
        }
}

/*
EXAMPLE 2: iconic gesture as the animation response
triggers the following animations in Unity:

melissa_mouth_neutral_500
melissa_mouth_a_500
melissa_mouth_e_500
melissa_mouth_c_500
melissa_mouth_neutral_500
melissa_mouth_m_500
melissa_mouth_e_500
melissa_mouth_c_500
melissa_mouth_f_500
melissa_mouth_c_500
melissa_mouth_neutral_500
melissa_iconic_thumbsup_d2000

*/

{ "sequence": {
        "element": "melissa",
        "total_duration": 2000,
        "start_delay": 0,
        "speech": "That's perfect!",
        "phoneme": "ðæts pərfɛkt!",
        "visemes": [ "a", "e", "c", " ", "m", "e", "c", "f", "c"],
        "entries": [
                {
                        "type": "iconic",
                        "label": "iconic_thumbsup_d2000",
                        "description": "both thumbs up",
                        "duration": 2000,
                              "order": 1,
                              "info": {
                                        "type": "iconic",
                                        "name": "thumbsup",
                                        "duration": "2000"
                              }
                    }
          ]
          }
}
```

Full List of ACTR Animations

 * The animation clips should be split into multiple glTF files per markup on the list
 * Each glTF file MUST use the same RIG
 * For organizational use, each file should contain one animation clip per named animation assigned to it
 * For final export, each animation category (i.e., e1, e2, admiring, encouraging, etc.) should be sequenced one after the other in descending order and exported as one glTF file.

 * durations are in milliseconds
 * @ 30 frames per second, the here are the mappings:

```
  * 500ms   = 15F
  * 1000ms  = 30F
  * 2000ms  = 60F
  * 3000ms  = 90F
  * 4000ms  = 120F
  * 5000ms  = 150F
  * 8000ms  = 240F
  * 13000ms = 390F
```

Mouth entries are so short that ACTR will just use LERP to transition between them.

Emotions are defined according to the emotion wheel (see related image file).
For head, emotions are essentially the various combinations of smile/frown with eyebrows up/down.

Here are the names of the emotions to map to the numbers
/* NOTE: Avoid using the names when possible.
e1 - Mischievous
e2 - Happy
e3 - Kind/Sympathetic
e4 - Worried/Uncertain
e5 - Sad/Hurt
e6 - Disapproving/Disappointed
e7 - Angry
e8 - Determined
e9,10 - Neutral

Here is the full set of categories:
e1,e2,e3,e4,e5,e6,e7,e8,e9,admiring,encouraging,appreciative,congratulatory,positive,cheers,wave,hello,goodbye,thumbsup,onethumbup,pout

NLP can send comma-delimited sets of the categories to be played back and ACTR will generate a playlist of the according length combining the different categories.

e.g. affect=thumbsup,e2

Head emotions should also include blinking, etc.
For body, emotions should include hand gestures, breathing, etc.

Neutral emotions e9 and e10 will be "loopable" on the base neutral keyframe and so do not need transitions. Transitions to emotions and transitions from emotions must be played before each emotion animation sequence. This is true for both head emotion tracks and body emotion tracks.

Head and body for a given emotion will start and end in the "emotion" state and so are loopable, but they require a transition to emotion animation and transition from emotion animation to be played before a given emotion sequence is played.

Iconics are also loopable, starting and ending on the "neutral" keyframe to avoid the need for specific iconic transition animations.

****
BELOW IS THE FULL SET OF EMOTION ANIMATIONS AND TRANSITIONS IN ORDER OF IMPLEMENTATION PRIORITY.
****

Please NOTE the naming convention for the A and B variants on the 1 second versions of each emotion.
Please NOTE the markup indicating which glTF file should contain the animation.

<mouth_and_iconic.gltf>

mouth_o_d500
mouth_a_d500
mouth_c_d500
mouth_e_d500
mouth_f_d500
mouth_m_d500
mouth_u_d500
mouth_neutral_d500

</mouth_and_iconic.gltf>

<e9.gltf>

head_e9-A_d1000
head_e9-B_d1000
head_e9_d2000
head_e9_d3000
head_e9_d5000
head_e9_d8000
head_e9_d13000

body_e9-A_d1000
body_e9-B_d1000
body_e9_d2000
body_e9_d3000
body_e9_d5000
body_e9_d8000
body_e9_d13000

</e9.gltf>

<e2.gltf>

head_transition-to-e2_d500
head_transition-from-e2_d500
body_transition-to-e2_d500
body_transition-from-e2_d500

head_e2-A_d1000
head_e2-B_d1000
head_e2_d2000
head_e2_d3000
head_e2_d5000
head_e2_d8000
head_e2_d13000

body_e2-A_d1000
body_e2-B_d1000
body_e2_d2000
body_e2_d3000
body_e2_d5000
body_e2_d8000
body_e2_d13000

</e2.gltf>

<e10.gltf>

head_e10-A_d1000
head_e10-B_d1000
head_e10_d2000
head_e10_d3000
head_e10_d5000
head_e10_d8000
head_e10_d13000

body_e10-A_d1000
body_e10-B_d1000
body_e10_d2000
body_e10_d3000
body_e10_d5000
body_e10_d8000
body_e10_d13000

</e10.gltf>

<e3.gltf>

head_transition-to-e3_d500
head_transition-from-e3_d500
body_transition-to-e3_d500
body_transition-from-e3_d500

head_e3-A_d1000
head_e3-B_d1000
head_e3_d2000
head_e3_d3000
head_e3_d5000
head_e3_d8000
head_e3_d13000

body_e3-A_d1000
body_e3-B_d1000
body_e3_d2000
body_e3_d3000
body_e3_d5000
body_e3_d8000
body_e3_d13000

</e3.gltf>

<e4.gltf>

head_transition-to-e4_d500
head_transition-from-e4_d500
body_transition-to-e4_d500
body_transition-from-e4_d500

head_e4-A_d1000
head_e4-B_d1000
head_e4_d2000
head_e4_d3000
head_e4_d5000
head_e4_d8000
head_e4_d13000

body_e4-A_d1000
body_e4-B_d1000
body_e4_d2000
body_e4_d3000
body_e4_d5000
body_e4_d8000
body_e4_d13000

</e4.gltf>

<mouth_and_iconic.gltf>

iconic_admiring-A_d1000

iconic_admiring-A_d2000
iconic_admiring-A_d3000
iconic_admiring-A_d4000
iconic_admiring-A_d5000
iconic_encouraging-A_d1000
iconic_encouraging-A_d2000
iconic_encouraging-A_d3000
iconic_encouraging-A_d4000
iconic_encouraging-A_d5000
iconic_appreciative-A_d1000
iconic_appreciative-A_d2000
iconic_appreciative-A_d3000
iconic_appreciative-A_d4000
iconic_appreciative-A_d5000
iconic_congratulatory-A_d1000
iconic_congratulatory-A_d2000
body_transition-from-e6_d500

head_e6-A_d1000
head_e6-B_d1000
head_e6_d2000
head_e6_d3000
head_e6_d5000
head_e6_d8000
head_e6_d13000

body_e6-A_d1000
body_e6-B_d1000
body_e6_d2000
body_e6_d3000
body_e6_d5000
body_e6_d8000
body_e6_d13000

</e6.gltf>

<e5.gltf>

head_transition-to-e5_d500
head_transition-from-e5_d500
body_transition-to-e5_d500
body_transition-from-e5_d500

head_e5-A_d1000
head_e5-B_d1000
head_e5_d2000
head_e5_d3000
head_e5_d5000
head_e5_d8000

iconic_congratulatory-A_d3000
iconic_congratulatory-A_d4000
iconic_congratulatory-A_d5000
iconic_positive-A_d1000
iconic_positive-A_d2000
iconic_positive-A_d3000
iconic_positive-A_d4000
iconic_positive-A_d5000

</mouth_and_iconic.gltf>

<e6.gltf>

head_transition-to-e6_d500
head_transition-from-e6_d500
body_transition-to-e6_d500

head_e5_d13000

body_e5-A_d1000
body_e5-B_d1000
body_e5_d2000
body_e5_d3000
body_e5_d5000
body_e5_d8000
body_e5_d13000

</e5.gltf>

<e1.gltf>

head_transition-to-e1_d500
head_transition-from-e1_d500
body_transition-to-e1_d500
body_transition-from-e1_d500

head_e1-A_d1000
head_e1-B_d1000
head_e1_d2000
head_e1_d3000
head_e1_d5000
head_e1_d8000
head_e1_d13000

body_e1-A_d1000
body_e1-B_d1000
body_e1_d2000
body_e1_d3000

body_e1_d5000
body_e1_d8000
body_e1_d13000

</e1.gltf>

<e8.gltf>

head_transition-to-e8_d500
head_transition-from-e9_d500
body_transition-to-e8_d500
body_transition-from-e9_d500

head_e8-A_d1000
head_e8-B_d1000
head_e8_d2000
head_e8_d3000
head_e8_d5000
head_e8_d8000
head_e8_d13000

body_e8-A_d1000
body_e8-B_d1000
body_e8_d2000
body_e8_d3000
body_e8_d5000
body_e8_d8000
body_e8_d13000

</e8.gltf>

<e7.gltf>

head_transition-to-e7_d500
head_transition-from-e7_d500
body_transition-to-e7_d500
body_transition-from-e7_d500

head_e7-A_d1000
head_e7-B_d1000
head_e7_d2000
head_e7_d3000
head_e7_d5000
head_e7_d8000
head_e7_d13000

body_e7-A_d1000
body_e7-B_d1000
body_e7_d2000
body_e7_d3000
body_e7_d5000
body_e7_d8000
body_e7_d13000

<mouth_and_iconic.gltf>

iconic_cheers_d3000
iconic_wave_d3000
iconic_hello_d3000
iconic_goodbye_d3000
iconic_thumbsup_d3000
iconic_onethumbup_d3000
iconic_pout_d3000

</mouth_and_iconic.gltf>

Iconic Gestures (Default List)

//Iconics are also loopable, starting and ending on the "neutral" keyframe to avoid the
//need for specific iconic transition animations
iconic_cheers_d1000
iconic_wave_d1000
iconic_hello_d1000
iconic_goodbye_d1000
iconic_thumbsup_d1000
iconic_onethumbup_d1000
iconic_pout_d1000

	User Says (NLP_IN)	Gesture Description	Gesture Name / ID
1		(idiosyncratic)	iconic_interestedA_d1000
2		(idiosyncratic)	iconic_interestedB_d2000
3		(idiosyncratic)	iconic_interestedC_d3000
4		(idiosyncratic)	iconic_interestedD_d4000
5		(idiosyncratic)	iconic_interestedE_d5000
6		head cocked to left	iconic_thinkingA_d1000
7		chin raised, eyes narrowed	iconic_thinkingB_d2000
8		chin lowered, looks at user	iconic_thinkingC_d3000
9		holds chin in right hand, folds left arm across abdomen	iconic_thinkingD_d4000
10		rests chin on right fist, folds left arm across abdomen	iconic_thinkingE_d5000
11		Eyes widen/eyebrows raise	iconic_surprisedA_d1000
12		(idiosyncratic)	iconic_surprisedB_d2000
13		(idiosyncratic)	iconic_surprisedC_d3000
14		(idiosyncratic)	iconic_surprisedD_d4000
15		(idiosyncratic)	iconic_surprisedE_d5000
16		Right finger to chin	iconic_listeningA_d1000
17		Right finger to right cheek	iconic_listeningB_d2000
18		Eyes squint, nod	iconic_listeningC_d3000

19		one eyebrow raised	iconic_listeningD_d4000
20		head cocked to right	iconic_listeningE_d5000
21		Puts hands behind body and nods head	iconic_concernedA_d1000
22		Cocks head, chin moving to left, and furrows forehead	iconic_concernedB_d2000
23		Holds chin in left hand, folds right arm across abdomen, left elbow resting on top of right hand	iconic_concernedC_d3000
24		Places left palm across left cheek, folds right arm across abdomen	iconic_concernedD_d4000
25		Jut chin forward and up, furrow brow	iconic_concernedE_d5000

Prioritisation (top ten in order of most useful down with an evenly scattered range of duration)

1. iconic_listeningA_d1000
2. iconic_listeningB_d2000
3. iconic_listeningC_d3000
4. iconic_listeningD_d4000
5. iconic_listeningE_d5000
6. iconic_thinkingA_d1000
7. iconic_thinkingB_d2000
8. iconic_thinkingC_d3000
9. iconic_thinkingD_d4000
10. iconic_thinkingE_d5000

	Bot Says (NLP_OUT)	Gesture Description	Gesture Name
1		Eyebrows raise, small smile	iconic_admiringA_d1000
2		Smile, nod head	iconic_admiringB_d2000
3		Eyebrows raise, head swivel so that chin moves down and to the left (head cock)	iconic_admiringC_d3000
4		Holds up right hand index finger, points at user (You!)	iconic_admiringD_d4000

5		Thumbs up, right hand	iconic_admiringE_d5000
6		(idiosyncratic)	iconic_encouragingA_d1000
7		(idiosyncratic)	iconic_encouragingB_d2000
8		(idiosyncratic)	iconic_encouragingC_d3000
9		Nods head	iconic_encouragingD_d4000
10		Folds hands in front of body and nods head	iconic_encouragingE_d5000
11		Thumbs up, right hand	iconic_appreciativeA_d1000
12		OK sign, right hand	iconic_appreciativeB_d2000
13		(idiosyncratic)	iconic_appreciativeC_d3000
14		(idiosyncratic)	iconic_appreciativeD_d4000
15		(idiosyncratic)	iconic_appreciativeE_d5000
16		Thumbs up	iconic_congratulatoryA_d1000
17		Smile, thumbs up	iconic_congratulatoryB_d2000
18		Claps (five beats)	iconic_congratulatoryC_d3000
19		Nod, smile, thumbs up	iconic_congratulatoryD_d4000
20		Fist pump in the air, smile	iconic_congratulatoryE_d5000
21		Nod head	iconic_positiveA_d1000
22		Close eyes, nod head two beats, open eyes	iconic_positiveB_d2000
23		(idiosyncratic)	iconic_positiveC_d3000

24		(idiosyncratic)	iconic_positiveD_d4000
25		(idiosyncratic)	iconic_positiveE_d5000

Prioritization (top ten in order of most useful down with a mostly-evenly scattered range of duration)

1. iconic_admiringA_d1000
2. iconic_admiringB_d2000
3. iconic_admiringC_d3000
4. iconic_admiringD_d4000
5. iconic_admiringE_d5000
6. iconic_appreciativeA_d1000
7. iconic_appreciativeB_d2000
8. iconic_positiveA_d1000
9. iconic_positiveB_d2000

10 iconic_encouragingD_d4000

Miscellaneous Notes

Prioritization of Affect

In 100 outputs the avatar will be neutral (#9 & #10) and happy (#2) the majority of the time. Generally speaking the left side of the chart is less commonly used, so the avatar is rarely seen with its eyebrows down, and even less commonly seen with both eyebrows and mouth down.

#9	20%
#2	20%
#10	20%

= 60% of animations

#3	12%
#4	9%
#6	6%
#5	6%
#1	3%
#8	2%
#7	2%