Leveraging Cloud-based Tools to Talk with Robots

Enas Altarawneh and Michael Jenkin
Electrical Engineering and Computer Science, York University, Toronto, ON, Canada

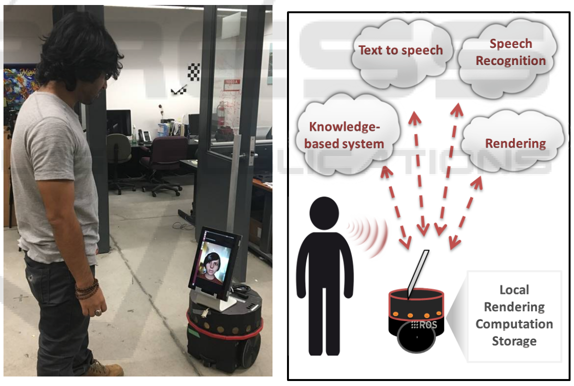
Keywords: Human-robot Interaction, Cloud-based AI, Realistic Human Avatar.

Abstract: Although there has been significant advances in human-machine interaction systems in recent years, cloud-based advances are not easily integrated in autonomous machines. Here we describe a toolkit that supports interactive avatar animation and modeling for human-computer interaction. The avatar toolkit utilizes cloud-based speech-to-text software that provides active listening by detecting sound and reducing noise, a cloud-based AI to generate appropriate textual responses to user queries, and a cloud-based text-to-speech generation engine to generate utterances for this text. This output is combined with a cloud-based 3D avatar animation synchronized to the spoken response. Generated text responses are embedded within an XML structure that allows for tuning the nature of the avatar animation to simulate different emotional states. An expression package controls the avatar's facial expressions. Latency is minimized and obscured through parallel processing in the cloud and an idle loop process that animates the avatar between utterances.

1 INTRODUCTION

This work describes the development of an animated and programmable interactive avatar. The cloud-based Extensible Avatar toolkit (or the EA toolkit for short) is a rendered 3D visual puppet (avatar) through which humans can interact with a robot (see Figure 1). The toolkit relies on a number of components: a speech-to-text module that converts utterances captured by a microphone in proximity to the robot into text, a generic text-to-utterance module that generates natural language speech, and on-robot speaker and display hardware. Augmenting these utterances with a visual avatar requires rendering complex and detailed animations in real time and synchronizing these animations to the utterance. Such rendering typically requires specific and/or resource intensive hardware which may not be available embedded within an autonomous robot. In order to overcome this constraint, this work explores utterance recognition and the rendering of the avatar using cloud-based computational resources. A high level view of this process is shown in Figure 2.

The work described here leverages a number of cloud-based software components. It relies on a speech-to-text recognition module, a knowledge engine, a text-to-speech engine, a 3D character design system, a 3D animation toolkit, and a lip-syncing plugin for the animation program that extracts the sounds in words, maps them to mouth shapes and plots them according to duration and occurrence in the text in real time. An expression package controls the animated character's mood and facial expressions. Rather than seeking to advance our understanding in terms of these aspects, this work considers how to integrate these modules to provide an animated cloud-based avatar. Recognizing that the use of cloud-based resources will introduce unwanted delays in the recognition and rendering process, key technical contributions in this work include (i) the development of an adaptive parallelization strategy to



(a) *The avatar* (b) *Components*

Figure 1: Talking with an interactive avatar-enabled robot. Cloud-based systems are used to augment limited on-board computational and rendering resources.

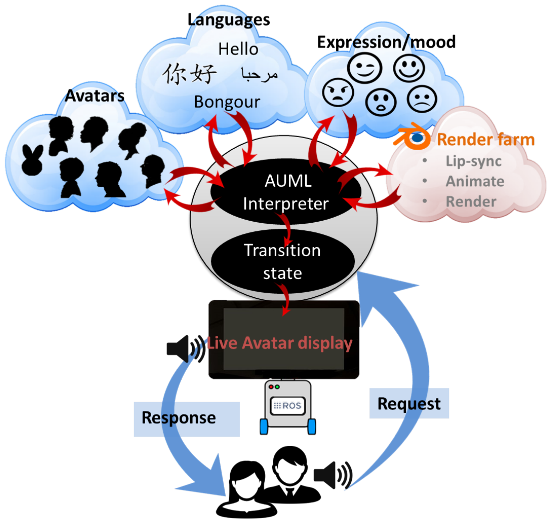


Figure 2: Extensible Avatar (EA) toolkit. *The avatar can be customized for language, appearance and tenor of the conversation. Responses can be reflected through the avatar or through commanded instruction to the vehicle.*

leverage cloud-based rendering resources to minimize the latency itself, and (ii) the development of an "idle loop" process to obscure any resulting latency in the recognition, response and rendering processes. This "idle loop" process animates the avatar puppet between utterances so that the character being rendered is not still but rather appears to interact with external users even when not spoken to directly.

2 PREVIOUS WORK

An artificially intelligent agent is an autonomous entity that observes the environment through sensors and acts upon it using actuators, directing its activity towards achieving a specific set of goals (see Russell and Norvig 2003). A virtual agent or intelligent avatar has applications in almost every field. For example, avatars and virtual agents have been used as an interface for home care monitoring and companionship (see (Shaked, 2017)). There have been a number of previous attempts to create realistic 3D expressive talking heads for intelligent agents and avatars and some have shown encouraging results (e.g. (Bremner et al., 2016), (Wan et al., 2013), (Anderson et al., 2013a), (Anderson et al., 2013b)); however existing systems have not yet achieved the level of realism associated with their 2D counterparts ((Anderson et al., 2013a)). 2D talking heads presently look more realistic than their 3D counterparts, but they are limited in the variety of poses and in the lighting conditions that can be simulated ((Anderson et al., 2013a)). Enabling a talking head to express emotion along with a synchronized utterance is a challenging problem. Model-based approaches have shown some potential in solving this problem. For example, the avatar described by (Anderson et al., 2013a) is driven by text and emotion inputs and generates expressive speech with corresponding facial movements. It uses a Hidden Markov Model (HMM)-based text-to-speech synthesis system ((Zen et al., 2007)) with an active appearance model (AAM)-based facial animation system ((Cootes et al., 2001)). The system utilizes a cluster adaptive training framework to train both the speech and facial parameters which allows for the creation of expressions of different intensity and the combining of different expressions together to create new ones. Results on an emotion-recognition task show that recognition rates given the synthetic output are comparable to those given the original videos of the speaker. (Anderson et al., 2013b) present a similar study that produced a talking head given an input text and a set of continuous expression weights. The face is modeled using an active appearance model, and several extensions that enhance the face. The model allows for normalization with respect to both pose and blink state which significantly reduces artifacts in the resulting synthesized sequences ((Anderson et al., 2013b)).

A central problem in avatar generation involves synchronizing the animated avatar to audio responses generated by the avatar. There exist software systems and human aided approaches that can be used to partially or completely automate the creation of facial and speech animation. One example of such work is the framework for synthesizing a 3D lip-sync speech animation to a given speech sequence and its corresponding text described by (Chen et al., 2012). The first step in this process identifies the key-lip-shapes from a training video that guides the creation of corresponding 3D key-faces. These 3D key-faces

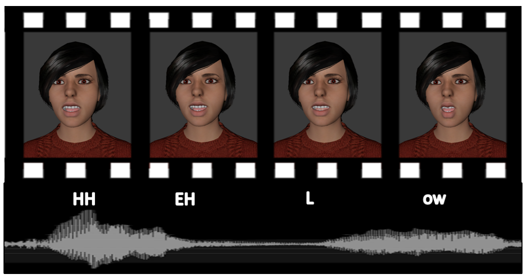


Figure 3: A film strip of an animated character. *The avatar is controlled by control points. These controls are of two types, facial controls and the body rig. Examples of a facial controls include mouth-open, left-brow-right-up, left-brow-mid-down and mouth-right-corner-up. Examples of body rigs include neck, left-eye-lid, jaw and spine.*

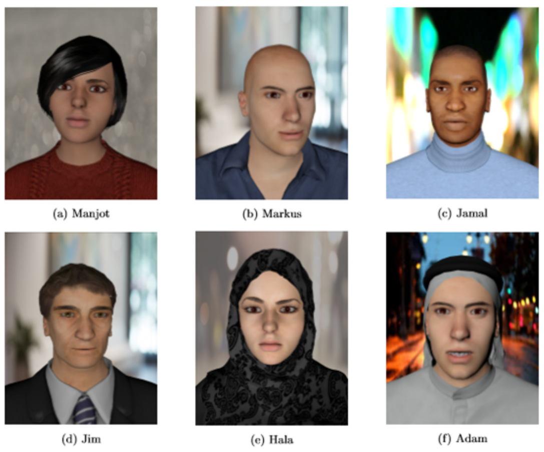


Figure 4: Examples of developed avatars.

are used to construct Dominated Animeme Models (DAM) for each kind of phoneme. Considering the coarticulation effects, which is the articulation of two or more speech sounds together so that one influences or dominates over the other, the DAM computes the polynomial-fitted animeme shape for each key-face and its corresponding dominance weight ((Chen et al., 2012; Huang et al., 2009)). Other approaches exist in the literature. For example, (Wang et al., 2012) describes a statistical, multi-streamed Hidden Markov Model (HMM) trained using super feature vectors consisting of 3D geometry, texture and speech. The HMM is then used to synthesize both the trajectories of head motion animation and the corresponding dynamics of texture. In yet another example, the speech signal, which is represented by Mel-Frequency Cepstral Coefficient vectors, is classified into visemes using a neural network ((Zoric and Pandzic, 2005)). Using genetic algorithms the topology of the neural network is configured automatically. This eliminates the need for manual neural network design and considerably improves viseme classification results.

There exists a range of software tools, plugins and add-on solutions that aid animators with lip-syncing and facial animations. One example is CrazyTalk (CrazyTalk, 2018). Crazy Talk is a 2D real-time facial animation software that uses voice and text to animate facial images. It allows an animator to use their own voice to create their animations in real-time using an automatic motion engine. Another example is Faceshift (Faceshift, 2018), which is a software solution that can capture the user's facial expressions in real time and generate an avatar that mimics the user. Faceshift technology uses off-the-shelf RGBD cameras. Faceshift is compatible with most available 3D software packages via plugins and data export. A Blender (Mullen (2012)) plugin called Quicktalk (Quicktalk, 2017) can semi-automatically lipsync any

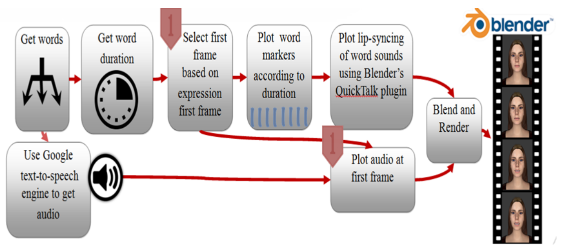

Get words
Get word duration
Select first frame based on expression first frame
Plot word markers according to duration
Plot lip-syncing of word sounds using Blender's QuickTalk plugin
blender
Blend and Render
Use Google text-to-speech engine to get audio
Plot audio at first frame

Figure 5: The process of lip-syncing the spoken words.

selected audio using the MakeHuman MxH2 character (MXH2, 2017). To use this plugin the user manually exports a Makehuman's MXH2 character, adds the sound track, adds the word-sound dictionary, adds the text script to be lip-synced, and then uses the plugin to automatically plot the lip-sync. The word plot markers then need to be manually adjusted to match the audio of each word so that the lip-sync is not out of place.

3 BUILDING THE AVATAR

There exist a number of different suppliers of cloud-based speech recognition and text-to-speech audio generation systems. Here we use an abstract model of these processes. This abstract process has been implemented utilizing a number of different cloud-based and local engines, although the work has concentrated on the Google Engine (Speech Recognition, 2017). The development of this abstract toolkit builds upon substantive previous efforts in this domain. A standard toolkit for local speech recognition can be found online (Speech Recognition, 2017) and can be easily integrated into any ROS robot system. Google provides a toolkit to integrate their recognizer with 3rd party software. The output of this process is a natural language expression as a sequence of words in the recognition language. Similar tools exist for utterance generation. This work uses Google's textto- speech cloud-based engine (Krishnan and Gonzalez (2015)) to generate the audio layer of the utterance and blend it with an animated avatar to match the response.

The Avatar Utterance Markup Language (AUML). Rather than transmitting straight English text, the text to be rendered by the avatar is placed within a structured framework that provides rendering hints for both audio generation and avatar rendering through the Avatar Utterance Markup Language (AUML), a formal language for avatar utterances created in this work. This language is a XML representation; it defines a set of rules for encoding a desired output using a textual data format.

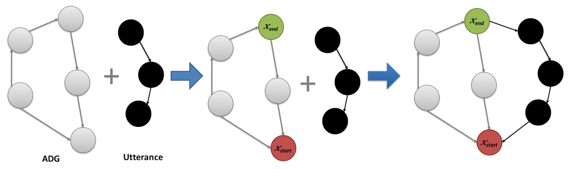


Figure 6: Generating a path in the ADG. *To generate a path for the utterance a start and end node are selected in the ADG, the utterance is then inserted into the graph and rendered to allow smooth transition from and to the utterance.*

Every utterance includes an avatar's detailed description, language, spoken words, expression associated with sub phrases and general mood. The goal of this language is to standardize and facilitate the use of available avatars, languages and expressions. It also allows for extensibility by the simple inclusion of new tags or values.

The Avatars. Avatars are 3D puppets properly rigged for animation. This work utilizes an open source realistic 3D human character design software called MakeHuman(J. Russell and Cohn (2012)). MakeHuman provides the ability to manipulate age, weight, length, gender, and race of the avatar. The software also allows for changes in facial details, hair, eyes, skin and clothes. Users can select from a variety of 3D meshes and bone structures for each character. Characters are exported using the MHX2 rig (MHX2, 2017) which enables MakeHuman structures to be imported into the Blender renderer (Mullen (2012)). Examples of implemented avatars are shown in Figure 4.

Lip-syncing Spoken Words. Spoken words in the utterance are lip-synced with the audio to provide a realistic utterance. A key requirement here is understanding the time indexing of individual events in the utterance. As we know the text used to generate the audio we use the text to help animate the lips. We utilize a dictionary of the sounds in words and use this to compute the timing of events in the utterance. Having prior knowledge of the duration of every possible word (or at least most common words) helps to automate realistic lip-syncing and more generally allows us to predict how long the resulting audio and video sequences will be. In order to obtain the expected duration of utterances we trained our system on the duration of every word in a dictionary using the text-to-speech engine. We assume that the duration $t(x)$ of the spoken word x is independent of its context within which x was used. This simplifies the process of estimating the duration of the spoken phrases. Audio strips generated by a text to audio engine are typically embedded within a quiet clip. The result audio duration usually includes empty audio at the beginning and the end of the audio strip. An audio clip consists of a constant number of frames (f) per second (typically 24) and the pre- and post-clip residue have proven to be of constant duration. In-order to accommodate these effects, the duration of each word is used as a weight for the actual plot time of the word in the lip-sync animation of the sentence. The time marker of each word is calculated using $w(x) = t(x)/\sum_{i=1}^{n}(t(x_i))$. The duration of the word x in the actual sentence $T_s(x)$ is approximated by the weight of the word multiplied by the actual duration of the sentence $t_s(x) = w(x) * t(x)$. The marker of each word in the actual sentence $m(xs)$ is the marker for the first frame (f_0) plus the number of frames $(NF(d))$ in the duration space (d) of every word that comes before it. The frame marker for each word is calculated using $m(x) = f_0 + NF(\sum_{j=1}^{J<i}(w(x_j) * t(x_j)))$.

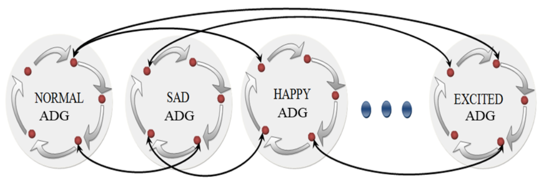


Figure 7: Realistic utterance state transitions.

The vismes in every word are mapped to mouth and lip key-frame shapes. These key-frame shapes are used to plot the vismes associated with each word. We utilize key-frame shapes that are part of the MHx2 facial rig exported from MakeHuman ((Russell and Cohn, 2012)) and imported into Blender ((Mullen, 2012)). This automated lip-sync process is based on a manual process that uses a blender plugin called QuickTalk ((Quicktalk, 2017)). We automated this process and optimized the word markers based on the actual duration of the word instead of using equally divided markers. The QuickTalk plugin creates an indexed dictionary of all of the words in the vismes dictionary for every lip-synced phrase. This work optimizes the vismes retrieval mechanism by using one pass for the words. The sounds plotted using key-frame shapes were based initially on the originally hard coded values for each visme found in MHX2. The original values create exaggerated mouth movements for each visme which did not seem to produce a realistic outcome. These default values were adjusted to create a more desired effect. The lip-syncing process is summarized in Figure 5.

Building a Realistic Utterance State Transition. Between utterances we do not want the avatar to be still. Rather we wish the avatar to engage in apparently normal motion. Furthermore, we wish the

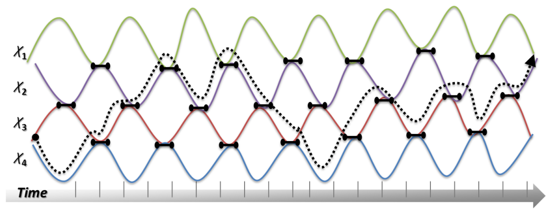


Figure 8: Multiple edge state transition. *Four idle loops x_1 through x_4 are illustrated with transitions between x_1 and x_2, x_2 and x_3, and x_3 and x_4. Potential idle loops in each of x_1 through x_4 shown in different colors an a possible stochastic path is shown as a dotted line.*

avatar to transit from this delay behavior to utterance behavior seamlessly. We accomplish this by pre-rendering and pre-loading to the robot a collection of renderings that can be played when the avatar is idle and which are designed to be combined together to make arbitrarily long sequences of idle behavior. The Avatar Delay Graph (ADG) provides a structure within which to encode short locally cached video sequences that can be played so as to provide an animation of the avatar between utterances. This structure also provides a mechanism within which to obscure rendering and transmission latencies which are unavoidable given the cloud-based rendering of the avatar. We model the ADG as a labeled directed graph $G = (V,E)$, where $V = \{x_1, x_2, ..., x_n\}$ and $E = \{e_1, e_2, ..., e_n\}$. Nodes correspond to points at which specific video sequences can be stitched together smoothly and edges model individual video sequences. Each edge $e = (x_a, x_b)$ is labeled with $\tau(e)$, how long it takes the play the sequence corresponding to e. When the avatar plays the video sequence corresponding to edge e the avatar's representation within the ADG transits from x_a to x_b. Also associated with edge e is an "expressive state" $es = (s_1, s_2, ..., s_p)$ an encoding of the nature of the avatar as it is perceived by a user. The dimensionality of es is avatar dependent.

Initially the avatar is in some node x_0 and has some avatar state S. When the avatar is not uttering an expression it walks the ADG in a stochastic manner as described below. When in node x it chooses from the edges departing from x. For each candidate edge e_i the avatar delay engine computes the difference from S to $es(e_i)$, $d_i = |S - es(e_i)|$. The avatar then chooses randomly from each of the incident edges with a probability inversely proportional to this distance. Specifically, with a probability proportional to $1/(d_i + \varepsilon)$ where ε is a small positive constant to avoid overflow. Once a best edge e_{best} is chosen the avatar's state S is updated using $S' = \lambda S + (1-\lambda) es(e_{best})$. Vertices in the ADG are optionally labeled as being a starting or terminating node to aid in terms of merging ADG

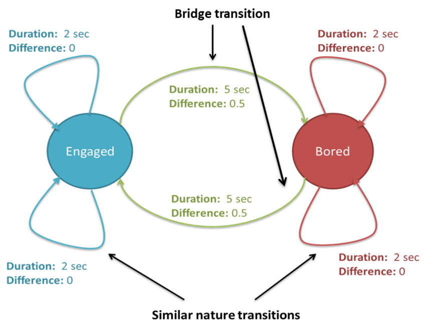


Figure 9: An example ADG, representing two different emotional states.

transitions and renderings with renderings associated with utterances. A node can be both a starting and terminating node. When an utterance is to be generated an appropriate terminating node in the ADG is identified based on the length of the path and to this node the similarity of the chosen transitioning node to the current avatar state as described below.

When the avatar is to render some utterance with state S, a new temporary edge $E = (x_{start}, x_{end})$ is constructed. Here the x_{start} and x_{end} nodes are chosen from the set of starting and terminating nodes in the ADG. The utterance is rendered between node x_{start} and x_{end} of the ADG. To accomplish this, we first identify x_{start} and x_{end} in the ADG. The x_{end} node is chosen such that (i) is a terminating node, and (ii) the mean of $|es((x_{end}, x_k)) - S|$ is minimized. That is, when the utterance is generated it terminates in a state where there is a good exiting edge in the ADG from x_{end}.The choice of start node is similar, but it is also necessary to identify a node that can be accessed quickly in terms of transitions in the ADT in order to avoid delaying the utterance (Figure 6). The x_{start} node is chosen such that (i) x_{start} has a starting label, and (ii) the cost of $\sum \alpha\tau(e) + (1-\alpha)|es(e) - S|$ is minimized, where here the sum is over the path in the ADG from the avatar's current state to the x_{start} node. This chooses a nearby start node such that the es values are similar to the current state of the avatar S. Note that the process of selecting the x_{start} node also enables the computation of the expected delay before it is necessary to start rendering the utterance.

Once the x_{start} and x_{end} nodes have been identified the avatar begins to move deterministically through the ADG to the x_{start} node following the sequence identified in the process of identifying this node. When it reaches x_{start} it then executes the rendered utterance and re-enters the ADG at the x_{end} node. The value of S is unchanged by this process although clearly it would be possible to associate a change in S with each utterance. Once at x_{end} the stochastic walk through the ADG continues until the next utterance is

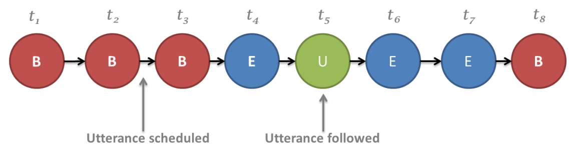


Figure 10: An example transition sequence. *Initially the avatar is bored in the bored state. At t_2 an utterance is scheduled for after t_4. The only start node is in the engaged state, so the avatar transits to the engaged node and executes the utterance at t_5. After the utterance the avatar returns to the engaged state where it continues to walk the graph.*

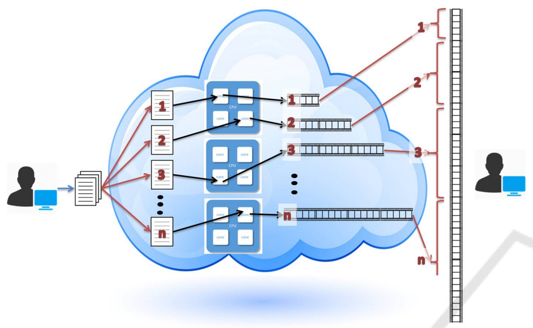


Figure 11: **The multi-process rendering system.** *A clip to be rendered is split into non-uniform length pieces and distributed to the rendering farm. The rendered sections are pieced together in the correct display sequence.*

available and the process continues.

When not generating utterances the avatar continues to animate through one of a number of waiting states to simulate a non-engaged but nevertheless animated speaker. We can structure such waiting states to simulate emotion or mood (Figure 7). Figure 7 also shows how common connectors allow for realistic transition. Figure 8 illustrates how these idle loops are combined stochastically in order to generate smooth idle sequences. The idle loops cross path to form a graph of nodes and edges. To illustrate this point more clearly consider Figure 9. Here the Bored and Engaged state is structured as two idle loops. Suppose that the starting and ending nodes exist only in the "engaged" portion of the ADG. Then it would need a bridge for an utterance while it is in the "bored" portion of the ADG. Figure 10 presents an example of the transition from bored to engaged.

In addition to a simulated general mood in the waiting state the avatar supports automated facial expressions manipulation. We can automate expression based on predefined rotation and translation values for our facial rig or a given set of values at any time. The importance of this automation is that expression manipulation and animation can be done on demand without the need for 3D GUI manipulation which can be extremely time consuming.

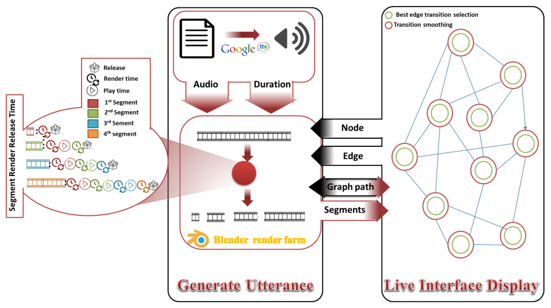


Figure 12: An overview of the parallel multiprocess generation of the utterance to reduce latency and how its result is connected to the display.

4 RENDERING THE AVATAR

We utilize the Google cloud platform to render the avatar. This platform provides a compute engine that allows for the creation of virtual machines with various levels of computation power (number of virtual CPUs, the inclusion of a GPU, size of RAM and disk space) and different choics of operating systems. The virtual machines can be customized based on the user needs. This work requires the creation of K identical virtual machines for multiprocessing and transforming these headless virtual machines into rendering engines. By default the instances created in a project on the Google cloud platform do not include a GUI, graphics display device or an audio playback device. Individual instances are headless servers with computational power and are described as compute engines. This work requires rendering engines. Rendering animations with Blender requires an X server, display screen and audio sink. To create a rendering engine from a compute engine a dummy audio sink needs to be created and activated and an X server needs to be started using a virtual display screen and assigning it values for resolution and color. Unfortunately off screen rendering can not make use of OpenGL which allows for the use of hardware acceleration. In order to enable rendering each instance is provided with VirtualGL (VirtualGL, 2018). VirtualGL is a software that can forward off-screen rending requests to the GPU for hardware acceleration. VirtualGL requires two displays (a 3D display to render from and a 2D display to render to) and a real X server. So, in addition to the previously mentioned virtual display and audio dummy sink, each instance requires a real X server running and a virtual 3D display linked to the GPU driver.

4.1 Distributed Rendering in the Cloud

First, we observe that we can parallelize the rendering process of the avatar. We can break the rendering sequence into smaller pieces, render those pieces in parallel in the cloud, and then present the rendered clips in sequence to the user. If we approximate the relationship between playing time T_p and rendering and network latency time T_r as a multiplicative factor (k), then $T_p = kT_r$ and if we have a pre-defined acceptable rendering latency (T), then we he have T seconds to render the first clip. This latency will result in kT seconds of played video. The second rendering stream also starts at time 0, and has the initial latency plus the time of the first clip's playing time within to render, resulting in $T + kT$ seconds of rendering time and $k(T + kT)$ seconds of rendered video from the second processor. Or more generally, $T_r^n = T + \sum_i^{n-1} T_p^i$ and under the assumption that $T_p = kT_r$ then $T_r^n = T + k\sum_i^{n-1} T_r^i$. Figure 11 illustrates the multi-process rendering system. Second, we observe that delays in the cloud are estimable, but are stochastic. So there is some small probability that the next clip to play may not be available when it is needed. Furthermore, we wish to simplify the problem of stitching the clips together when playing them so arbitrary clip points are to be avoided. So instead of using the break points as identified above we treat the break points as maximum values and seek the next earliest point in the utterance that corresponds to a word break or punctuation. This gives us more natural break points in the rendering. More rigorously, suppose we have a break point T_p identified through the process described above. Then rather than breaking the input at this point we scan backwards looking for the first break in the input, either the first punctuation or space between words. Call the time moving backwards in the clip until the first word break T_B and the time unit the first punctuation T_P. We weight each by k_B and k_P respectively and choose the minimum of T_B, k_B, $T_P k_p$ and V_{max} as the break point. Here V_{max} is a maximum weighted distance to backup. Note that – especially for very short duration clips – one or more of T_B and T_P may not exist. Third, we observe that we can 'stall' the video being generated should it be necessary by rocking the video to be played back and forward a small amount to avoid the avatar becoming 'stuck' or 'stuttering'. Rolling the video backwards and forwards will always be consistent with the video being played and can be used to 'hide' unexpected latency. Figure 12 provides an illustration of the overall rendering and display process. The figure shows how the rendering and display process are connected and the data flow required to achieve a seamless display of introduced response utterances.

5 EXPERIMENTAL SYSTEM

A customized version of the cloud-based avatar for human-robot interaction was used to navigate the robot using a simple chatbox layer that can identify targeted navigational words and numbers in the user's speech within context. This customized version allows for sending navigational commands using speech. These commands include "go left", "go right", "move forward" and "stop". Using these commands the robot moves continuously until given the command "stop". A specified distance can be sent to these commands by adding a number at the end of each command, such as "go forward two". The avatar would respond with an appropriate utterance by saying "going left", "going right", "coming through" and "stopping". The utterances are cached locally on the robot after the first utterance of a kind is sent to the cloud for rendering and is received by the local device for display. An image of a user interacting with this customized version of the interface is shown in Figure 1.

6 ONGOING AND FUTURE WORK

We are currently completing a user study exploring the efficiency of the approach and the potential of exploring locally provided cloud resources specifically tuned for avatar rendering. And we are working to deploy the technology in a greeter robot application.

ACKNOWLEDGMENT

The financial support of CFREF VISTA projects, the Google Cloud Platform and the NSERC's NCFRN and NCRN projects are gratefully acknowledged.

REFERENCES

Anderson, R., Stenger, B., Wan, V., and Cipolla, R. (2013a). An expressive text-driven 3d talking head. In *ACM SIGGRAPH 2013 Posters*, pages 80:1–80:1, New York, NY. ACM.

Anderson, R., Stenger, B., Wan, V., and Cipolla, R. (2013b). Expressive visual text-to-speech using active appearance models. In *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*

(CVPR), pages 3382–3389, Washington, DC. IEEE Computer Society.

Bremner, P., Celiktutan, O., and Gunes, H. (2016). Personality perception of robot avatar tele-operators. In *Proceeding of the 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 141–148, Christchurch, New Zealand.

Chen, Y. M., Huang, F. C., Guan, S. H., and Chen, B. Y. (2012). Animating lip-sync characters with dominated animeme models. *IEEE Transactions on Circuits and Systems for Video Technology*, 22(9):1344–1353.

Cootes, T. F., Edwards, G. J., and Taylor, C. J. (2001). Active appearance models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(6):681–685.

CrazyTalk. (2018). *Create 3D talking heads with CrazyTalk*. Retrieved from https://www.reallusion.com/crazytalk/

Faceshit. (2018). *Faceshift*. Retrieved from http://openni.ru/solutions/faceshift/index.html

Huang, F.-C., Chen, Y.-M., Wang, T.-H., Chen, B.-Y., and Guan, S.-H. (2009). Animating lip-sync speech faces by dominated animeme models. In *SIGGRAPH '09: Posters*, pages 2:1–2:1, New York, NY. ACM.

Krishnan, S. T. and Gonzalez, J. U. (2015). *Building Your Next Big Thing with Google Cloud Platform: A Guide for Developers and Enterprise Architects*. Apress, Berkely, CA, USA, 1st edition.

MHX2. (2017). *Mhx2 documentation.* Retrieved from https://thomasmakehuman.wordpress.com/mhx2-documentation

Mullen, T. (2012). *Mastering Blender*. SYBEX Inc., Alameda, CA, USA, 2nd edition.

Quicktalk. (2017). *Quicktalk lip synch addon.* Retrieved from Available: https://tentacles.org.uk/quicktalk

Russell, J., & Cohn, R. (2012). *Makehuman*. Book on Demand. Retrieved from https://books.google.ca/books?id=TFeaMQEACAAJ

Quicktalk (2017). Quicktalk lip synch addon.

Russell, J. and Cohn, R. (2012). *Makehuman*. Book on Demand.

Shaked, N. A. (2017). Avatars and virtual agents - relationship interfaces for the elderly. *Healthcare Technology Letters 4.3*, pages 83–87.

SpeechRecognition. (2017). *Speechrecognition 3.8.1 : Python package index - pypis.* Retrieved from https://pypi.python.org/pypi/SpeechRecognition/ (Accessed 30- April-2017)

VirtualGL. (2018). *Virtualgl the virtualgl project.* Retrieved from https://www.virtualgl.org/

Wan, V., Anderson, R., Blokland, A., Braunschweiler, N., Chen, L., Kolluru, B., Latorre, J., Maia, R., Stenger, B., Yanagisawa, K., Stylianou, Y., Akamine, M., Gales, M., and Cipolla, R. (2013). Photo-realistic expressive text to talking head synthesis. In *Proceedings of the Annual Conference of the International Speech Communication Association (INTERSPEECH)*, Lyon, France.

Wang, L., Han, W., and Soong, F. K. (2012). High quality lip-sync animation for 3d photo-realistic talking head. In *Proceeding of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4529–4532, Kyoto, Japan.

Zen, H., Nose, T., Yamagishi, J., Sako, S., Masuko, T., Black, A. W., and Tokuda, K. (2007). The HMM-based speech synthesis system (HTS) version 2.0. In *Proceedings of the 7th ISCA Tutorial and Research Workshop on Speech Synthesis (SSW)*, Kyoto, Japan.

Zoric, G. and Pandzic, I. S. (2005). A real-time lip sync system using a genetic algorithm for automatic neural network configuration. In *Proceeding of IEEE International Conference on Multimedia and Expo*, pages 1366–1369, Amsterdam, Netherlands.