Animating TTS Messages in Android using Open-Source Tools

Ronald Yu
School of Computer Science and Engineering
University of California, Irvine
ronaly1@uci.edu

Tong Lai Yu
School of Computer Science and Engineering
California State University, San Bernardino
tyu@csusb.edu

Ihab Zbib
School of Computer Science and Engineering
California State University, San Bernardino
zbibi@csusb.edu

ABSTRACT

We describe in this paper how to use open-source resources to design and implement an Android application that renders a three-dimensional model of a human head to animate the lip movements of human speech from input text. The application utilizes the Android Text-To-Speech (TTS) engine[1] to convert any input text, which can be entered by the user in a text box or chosen from a menu of predefined messages, to human speech in English. Animation of the speech is carried out by a 3D graphics model of a human head composed of polygon meshes[20]. Blender[7, 30], a popular open-source graphics suite, is employed to create the 3D model and save its mesh data in the COLLAborative Design Activity (COLLADA) format[22], which is also an open graphics format.

We use Java language to develop a parser[25, 42] to extract coordinates of polygons from a COLLADA file and organize the data into a format that can be rendered effectively by OpenGL ES, the graphics rendering library used by Android. The producer-consumer paradigm is employed to synchronize the animated lip movements and the speech generated by TTS. When the application is lying idle, it moves the head randomly to simulate other facial expressions such as blinking the eyes and yawning.

Keywords

Open-source, 3D Graphics, Animation, Text-To-Speech, TTS, Android

1. INTRODUCTION

Open-source software has been playing a critical role in the advent of technology. A lot of breakthroughs in technology development and application such as Watson's Jeopardy win[5] and the phenomenal 3D movie Avatar[4] are based on open-source software. It is a significant task to explore the usage of available open-source tools to develop software applications for research or for commercial use[40, 41]. The Android application reported in this paper is developed with free software resources, which are mainly open-source.

Mobile devices have become ubiquitous and in the last couple of years, Android, an open-source software stack for running mobile devices, has become the dominant platform of many mobile devices such as mobile phones and smart phones[14]. There has been exponential growth in mobile applications in recent years. Speech simulation is one of the areas that enjoy rapid growth, and a significant amount of research has been done on speech animation using 3D graphics models[32]. The video compression standard MPEG-4 also has specifications on facial animation for synthesized speech[28, 13].

Though speech simulation is still an ongoing research topic, it already has numerous commercial applications including game development and customer service[6], and it contributes to both the developments of acoustic and visual applications[12, 9].

Our work reported in this paper develops and merges the audio and visual technologies into one application by making use of open-source technologies. The main tool we use for rendering graphics is OpenGL for embedded systems (ES). The graphics library OpenGL[33] is the industry standard for developing 2D and 3D graphics applications[2, 8], and OpenGL ES[23, 3, 27] is OpenGL modified for embedded systems. There is a major difference between OpenGL ES 1.X and OpenGL ES 2.X. While the 1.X version shares the same functionality and syntax of the traditional OpenGL APIs and, like early OpenGL, has a fixed pipeline and operates as a state machine, the 2.X version has adopted a programmable pipeline architecture that allows users to program vertex and fragment shaders[24, 31], the equivalent of OpenGL Shading Language (GLSL)[18]. The vertex shader is responsible for processing geometry. The fragment shader works at the pixel level, processing incoming fragments to produce colors including transparency.

Mobile devices are characterized by small display size[29], limited memory capacity and limited computing power. All of these aspects affect the graphic animation experience of the mobile user. These limitations make the design and implementation of a TTS animation application in a mobile device very different from that of an application running on

WORLDCOMP'13:CGVR13 July 22–25, 2013, Las Vegas, Nevada, USA.

a desktop PC.

Another problem one must address is the audio-video synchronization. For traditional video compression of natural scenes, MPEG standard uses timestamps to synchronize audio and video streams[19]. MPEG-4 also addresses coding of digital hybrids of natural and synthetic, aural and visual information[28, 32]. Doenges et al. mentioned in their paper[13] that special attention must be paid to the synchronization of acoustic speech information with coherent visible articulatory movements of the speakers mouth in MPEG-4 synthetic/natural hybrid coding (SNHC) for animated mixed media delivery. However, they did not present the details of synchronization in the paper. Our synchronization problem of video and audio is different from that of MPEG-4 as our application does not involve any data encoding and decoding, and data transmission. Therefore, we do not use timestamps to synchronize audio and video. Instead, the synchronization is done using the producer-consumer paradigm[35], which works effectively in this situation.

The application is developed for Android-based mobile devices. Android provides a Text-to-Speech (TTS) engine (PICO) with limited APIs[1]. The main thread of the application presents a text box to the user for entering texts; prepared sample texts are also available as items on the application menu, and a sample can be chosen by clicking on a button of the menu. The input text is used for the speech simulator that plays the sound using the Android TTS APIs and renders the corresponding visemes while performing a lip-synchronization action, keeping the audio and video synchronized. Visemes, which can be considered as visual counterpart of phonemes in audio, are visually distinct mouth, teeth, and tongue articulations for a language.

Besides the main thread, the application has three other threads. One of them is responsible for voice synthesis and speech simulation by making use of the Android Text-to-Speech(TTS) engine[1]. Another thread controls the 3D rendering and animation of a human head. This thread implements the OpenGL ES function calls and has to decide which object to render based on the input data. The last thread is the input text thread that handles the insertion of the data into the text buffer. This thread implements the producer in the Producer-Consumer problem. Figure 1 is a UML diagram showing these components and their connections, where the TTS Thread is Consumer 1 and the Animation Thread is Consumer 2.

2. GRAPHICS 3D MODEL

The 3D model is initially imported from Google SketchUp 3DWarehouse[16] and is shown in Figure 2. We use the free version of Sketchup[36], a 3D drawing tool, to convert it to a COLLADA file, which can be then imported by Blender[7, 30]. Blender is a free 3D graphic suite for creating, rendering and animating graphics models[30]. It supports a variety of formats such as COLLADA(.dae), Wavefront(.obj), 3D Studio(.3ds), and others.

To generate a new facial expression, the model is modified by deforming the mesh, and a different copy is created and passed to the COLLADA parser to create a metafile. The viseme, or the shape of the mouth that corresponds to each

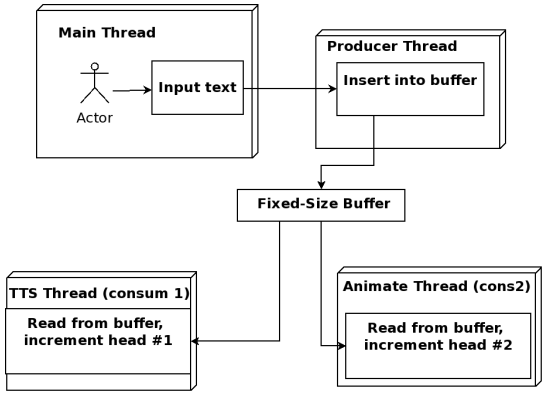

Figure 1: Components Diagram

Figure 2: Model From Google 3DWarehouse

phoneme, is based on the lip-sync phonetic-based animation[38, 15] used in animation movies. Figure 3 shows the lip shapes for phonemes that we have adopted.

In addition to the mouth shapes, other facial expressions are created to help simulate a more human-like agent. These facial expressions include eye blinking, eyebrow movements and yawning. These expressions are presented to keep the user entertained when the application is idle. Figure 4 shows the Android emulator running the 3D Face at rest position.

Java is used to develop a COLLADA parser[25], which parses a COLLADA file and extracts the necessary information for rendering and animating the graphic models. The faces of the model are meshes of polygons of three or more edges. Because OpenGL ES 1.0 can only render triangles, the parser has to extract the indices of every polygon, convert them into triangles, and recalculate the normal vector for every triangle by performing a cross product of the vectors along two of the triangle's edges.

Since the COLLADA file is essentially an XML document, the parser needs to make use of an XML library to carry out the parsing. Java APIs provide wide support for XML parsing and a variety of libraries to choose from such as JAXP, JDOM and SAX. Most of these libraries support the XML Path Language (XPath) [17]. While the Document Object Model (DOM) [39] is a more complete tree structure representation of the document, XPath is a straightforward language that allows the selection of a subset of nodes based on their location in the document [17]. The parsing program described here makes use of the Java package *javax.xml.xpath* to extract the necessary nodes from the

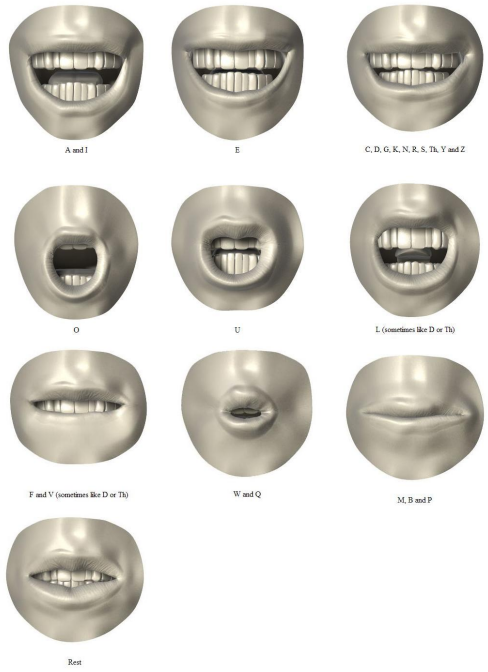

Figure 3: Preston Blair Phoneme Series

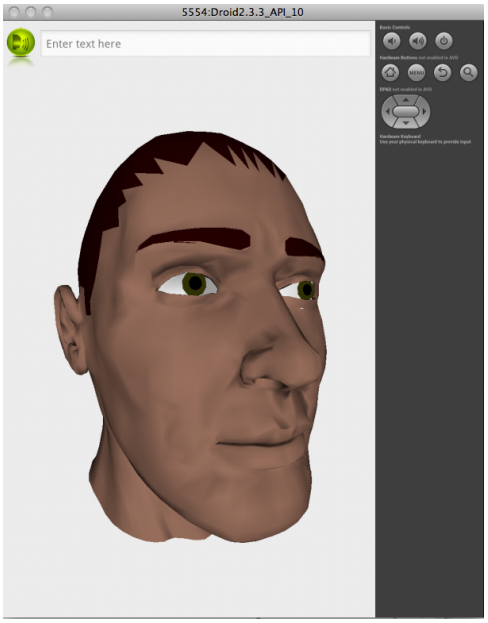

Figure 4: Resting Position

COLLADA document.

The parser parses the data of a COLLADA file into a meta-file containing a set of vertices coordinates, their normal vectors, the indices of the triangle and normalized color codes. The following data sample is an example of the data of a meta-file.

Meta-file Data Sample:

```
#upper_head.dae
object:base_039-mesh
{
  vertices:2517
  {
    0 2.8f,1.38f,-0.66f,
    1 2.83f,1.31f,-0.72f,
    2 2.78f,1.31f,-0.72f,
    ...
  }
  normals:2517
  {
    0 0.02f,0.62f,-0.78f,
    1 0.03f,1.0f,-0.08f,
    2 -0.74f,0.26f,-0.61f,
    ...
  }
  indices:4311
  {
    0 0,1,2,
    1 3,4,5,
    2 6,2,1,
    ...
  }
  materials:6
  {
    0 3509,0.51f,0.37f,0.31f,1.0f,
    1 50,0.41f,0.41f,0.41f,1.0f,
    2 690,0.13f,0.0f,0.0f,1.0f,
    ...
  }
}
```

The data labeled materials represent the color codes in red, green, and blue (RGB) of the affected faces. The first number is the table index and the second number indicates the number of faces that this color is applied to. The next three numbers are normalized RGB color codes. The color codes are normalized by dividing every component by 255. And the last number is the transparency, with values between 0 and 1, a value of 1 meaning opaque, and 0 meaning total transparency.

Every facial expression requires a separate graphic file that has to be loaded by the Android application. In order to reduce the amount of data, if the meta-file is a variation of the base model, the parser will compare it to the base model and export only the differences. This helps to reduce the start-up time of the Android application, as it does not need to create a different graphic object for every variation. The application can duplicate the original model and apply the changes in coordinates.

As mentioned earlier, OpenGL ES is the industry standard for embedded 3D graphics applications. This project makes use of OpenGL 1.0, which is supported by most of the commercial devices with an Android operating system. The minimum version of Android required is the Gingerbread, Android 2.3.3 API 10. One of the limitations of OpenGL ES 1.0 is that it only renders triangles. To overcome this issue, the COLLADA parser transforms a generic polygon into triangles and recalculates the normal vectors.

There are two ways to render a 3D object (or 2D for that matter) with OpenGL ES 1.0. One is array-based, and the other is element-based. To render the model with the array-based method, the vertices have to be inserted in the right order, so that OpenGL can render them in that sequence. The element-based approach is more flexible, as it does not require changes to the vertices buffer. A pointer to the indices buffer can be manipulated to render certain portions of the model at the time. This allows the program to apply certain attributes, such as color codes, to specific faces of the model without the need to load a complete color buffer with redundant information. The following is the code that renders the 3D model.

3D Model Rendering:

```
public void draw(GL10 gl) {
  //Enable drawing
  gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
  gl.glEnableClientState(GL10.GL_COLOR_ARRAY);
  gl.glEnableClientState(GL10.GL_NORMAL_ARRAY);
  gl.glFrontFace(GL10.GL_CW);
  gl.glVertexPointer(3, GL10.GL_FLOAT, 0,
                                vertexBuffer);
  gl.glNormalPointer(GL10.GL_FLOAT, 0,
                               normalsBuffer);
  int offset = 0;
  for(int i=0; i<materials.length; i++){
    gl.glColor4f(materials[i].rgb[0],
      materials[i].rgb[1],materials[i].rgb[2],
                          materials[i].rgb[3]);
    int length = materials[i].length;
    //!!!!very important
    indexBuffer.position(offset);
    int mode = GL10.GL_TRIANGLES;
    gl.glDrawElements(mode,length*3,
           GL10.GL_UNSIGNED_SHORT,indexBuffer);
    offset += length * 3;
  }
  //!!!!very important
  indexBuffer.position(0);
  //Disable drawing
  gl.glDisableClientState(GL10.GL_VERTEX_ARRAY);
  gl.glDisableClientState(GL10.GL_COLOR_ARRAY);
  gl.glDisableClientState(GL10.GL_NORMAL_ARRAY);
}
```

The application starts by loading the meta-files data of the 3D model previously prepared by the COLLADA parsing program into memory. The 3D model is composed of two parts. One is the upper head, and the other is composed of the mouth and jaws. Combined, they constitute a complete 3D model of a human head. While the application is loading the base model of each part to represent the resting position, a parallel thread is created to load the rest of the meta-files for different expressions. That reduces the startup time to half of what it would be if all the models are loaded in sequence. Once the meta-files are loaded into appropriate arrays and buffers, they are cached using a key-map structure for efficient access.

When there is no input, the application assumes itself to be in an idle situation and starts a timer. The application will periodically monitor the timer, and will randomly replace the resting models with animated ones, creating a frame-based movement effect. As soon as the user enters a text and sends the execute command, the application switches to the speech simulation mode, starts the TTS activity, and synchronizes the mouth animation to create the visual speech effect.

3. LIPS-AUDIO SYNCHRONIZATION

The producer-consumer paradigm[19, 34, 37], a well-studied synchronization problem in Computer Science, is employed to synchronize lip movements with the speech. A classical producer-consumer problem has two threads (one called producer, the other consumer) sharing a common bounded buffer. The producer inserts data into the buffer, and the consumer takes the data out. In our case, the buffer is a queue where characters are entered at the tail and are read at the head. Physically, the queue is a circular queue[19]. Logically, one can imagine it to be a linear infinite queue. The head and tail pointers are always advancing (incrementing) to the right. (To access a buffer location, the pointer is always taken the mod of the physical quenue length, e.g $tail \ \% \ queue\_length$.) If the head pointer catches up with the tail pointer (i.e. $head = tail$), the queue is empty, and the consumer must wait. If the difference between *head* and *tail* is equal to the length of the buffer, the queue is full, and the producer must wait.

In the application, the problem is slightly modified: it has one producer and two consumers, each has its own head pointer. The producer is the thread that accepts the input text and puts it in the queue, and the consumers are the Android TTS engine and the animation thread with routine calls to OpenGL ES. The text stream input thread controls the *tail* of the buffer and waits. Every time a new character is entered, *tail* is incremented. The TTS thread and the animation thread read and process the data while each of them is incrementing its own *head*. When the distance between the tail and one of the heads is larger than or equal to a certain empirical constant C, the producer stops and waits for the heads to catch up. When both heads reach for the tail, the producer starts inserting new data into the buffer. To further improve the algorithm, the TTS *head* and the animation *head* wait for each other, which forces the speech and the animation to be more in sync as shown in Figure 5. Below is a Java-like pseudo code for the synchronization of the TTS with the animation using the producer-consumer algorithm.

Producer-Consumer Code:

```
long tail = ttsThread = animationThread =0;
```

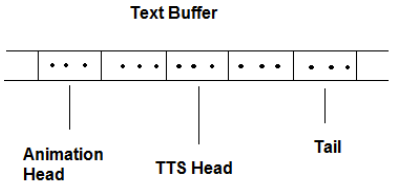

Figure 5: Producer-Consumer Data Buffer

```
//Fixed size buffer
char[] textBuffer = new char[offset];
//Producer thread
while(true){
  if(tail-head1>=offset||tail-head2>=offset)
    inputThread.sleep(100);
  else{
    textBuffer[tail % offset]=ch;
    tail++;
  }
}

//TTS thread
while(true){
  if(head1 == head || head1 >= head2+C1)
  ttsThread.sleep(100);
  else{
    char ch = textBuffer[head1 % offset];
    tts.speak(ch);
    head1++;
  }
}
//Animation thread
while(true){
  if(head2==tail || head2>=head1+C2)
    animationThread.sleep(100);
  else{
    char ch = textBuffer[head2 % offset];
    animation.render(ch);
    head2++;
  }
}
```

4. CONCLUSIONS AND DISCUSSIONS

We have presented the design and implementation of a speech animator for the Android mobile platform using exclusive open-source technologies. A parser written in Java is used to parse a COLLADA file containing 3D model data to a meta-file which can be rendered by OpenGL ES programs. Normally, only the difference between a 3D model and the base model are read from the meta-file. The final model is obtained by overlaying the scene created with the difference data on the base model. The producer-consumer paradigm is used to achieve lips-audio synchronization. The code of the application will be available for students and developers who want to use it as a starting point for further development.

There are unlimited ways of extending and enhancing the application. In particular, it is a significant task to explore the application of the Active Shape Models(ASM) or Active Appearance Model (AAM) developed by Tim Cootes and Chris Taylor in the 1990s[11, 10] to create more realistic 3D models. ASM and AAM are statistical models for image processing, in particular facial recognition. Using ASM or AAM, a system can be trained to generate new sets of data from a reduced covariance matrix and the vector representing the pose model, or the mean shape. In this case, the principal components can be applied to produce the different facial expression by training the model with a sample of visemes, representing the mouth shapes for the different phonemes. That will result in a more realistic movement of the mouth when simulating the visual speech. Some researchers have explored this approach and obtained good results[21].

Another significant enhancement to the application could be an interface enhancement with text messaging feature. Instead of reading the message, the user could listen to it and watch the simulation. It could also be interfaced with a live video streaming application. Instead of transmitting audio and video data which might be huge even after compression, one can transmit only the text of the talking person, along with some control data. The other person can watch and listen to a real-time simulation of the conversation. To realize this, speech recognition capabilities are required at the sender side. The Android platform supports this feature using Google's speech-recognition service[26]. Of course the transmitted text can be compressed by the sender and decompressed by the receiver but no synchronization is needed for the transmitted data as the animation is driven by the text and the lip-speech synchronization is done using the producer-consumer paradigm at the receiver end.

5. ACKNOWLEDGMENTS

We would like to thank Mr. Ted Benic, who helped enhance many of the 3D model images using Blender.

6. REFERENCES

[1] Android Open Source Project: TextToSpeech, *http://developer.android.com/reference/android/speech/*

[2] E. Angel, *Interactive Computer Graphics: A Top-Down Approach Using OpenGL*, Fourth Edition, Addison-Wesley, 2005.

[3] D. Astle and D. Durnil, *OpenGL ES Game Development*, Thomson Course Technology, 2004.

[4] Jun Auza, *The Technology Behind Avatar (Movie)*, *http://www.junauza.com/2010/01/technology-behind-avatar-movie.html*, Jan 2010.

[5] Charles Babcock, *Watson's Jeopardy Win A Victory For Mankind*, Information Week, Feb 2011.

[6] Koray Balc, *Xface: Open source toolkit for creating 3d faces of an embodied conversational agent*, pp. 263-266, Smart Graphics, 2005.

[7] Blender Foundation. Blender.org *http://www.blender.org/*, 2013.

[8] S. R. Buss, *3-D Computer Graphics: A Mathematical Introduction with OpenGL*, Cambridge University Press, 2003.

[9] C. Bregler, M. Covell, and M. Slaney, *Video Rewrite: Driving Visual Speech with Audio*, p.353-360, SIGGRAPH'97 Proceedings, ACM Press, 1997.

[10] T.F. Cootes, G. J. Edwards, and C. J. Taylor, *Active appearance models*, p. 484-498, ECCV, 2, 1998.

[11] T.F. Cootes, C.J. Taylor, D.H. Cooper and J. Graham *Active Shape Models - Their Training and Application*, Computer Vision and Image Understanding, p. 38-59, Vol. 61, No. 1, Jan. 1995.

[12] E. Cosatto, H.P. Graf, and J. Schroeter, *Coarticulation method for audio-visual text-to-speech synthesis*, US Patent 8,078,466, Dec 2011.

[13] P.K. Doenges et al., *MPEG-4: Audio/video and synthetic graphics/audio ifor mixed media*, p.433-463, Signal Processing: Image Communication, ELSEVIER, 9, 1997.

[14] Forbes Magazine, *Android Solidifies Smartphone Market Share*, *http://www.forbes.com/*, Jan., 2013.

[15] Gary C. Martin, *Preston Blair phoneme series*, *http://www.garycmartin.com/mouth_shapes.html*, 2006.

[16] Google Inc. Trimble Navigation Limited. 3D Warehouse. *http://sketchup.google.com/3dwarehouse/*, 2013.

[17] E.R. Harold. *Processing XML with Java: a guide to SAX, DOM, JDOM, JAXP, and TrAX*, Addison-Wesley Professional, 2003.

[18] S. Hill, M. Robart, and E. Tanguy, *Implementing Opengl ES 1.1 over OpenGL ES 2.0*, Consumer Electronics, 2008, ICCE 2008, Digest of Technical Papers, International Conference, IEEE, 2008.

[19] F. June, *An Introduction to Video Compression in C/C++*, Createspace, 2010.

[20] F. June, *An Introduction to 3D Computer Graphics, Stereoscopic Image, and Animation in OpenGL and C/C++*, Createspace, 2011.

[21] G. A. Kalberer, P. Muller, and L.V. Goo, *Modeling and Synthesis of Realistic Visual Speech in 3D*, p. 266-294, 3D Modeling & Animation, edited by N. Sarris and M. G. Strintzis, IRM Press, 2005.

[22] The Khronos Group Inc.,*https://collada.org/*, 2011.

[23] The Khronos Group Inc., *OpenGL ES The Standard for Embedded Accelerated 3D Graphics*, *http://www.khronos.org/opengles/*, 2013.

[24] The Khronos Group Inc., *OpenGL Shading Language*, *http://www.opengl. org/documentation/glsl/*, 2013.

[25] M. Milivojevic, I. Antolovic, and D. Rancic, *Evaluation and Visualization of 3D Models Using Collada Parser and Webgl Technology*, p. 153-158, Proceedings of the 2011 International Conference on Computers and Computing, World Scientific and Engineering Academy and Society (WSEAS), 2011.

[26] S. Mlot, *Google Adds Speech Recognition to Chrome Beta*, *http://www.pcmag.com/article2/0,2817,2414277,00.asp* , PC Magazine, Jan. 2013.

[27] A. Munshi et al., *OpenGL ES 2.0 Programming Guide*, Addison-Wesley Professional, 2008.

[28] I.S. Pandzic and R. Forchheimer, *MPEG-4 Facial Animation:The Standard, Implementation and Applications*, John Wiley & Sons, 2002.

[29] Thomas Rist, and Patrick Brandmeier, *Customizing Graphics for Tiny Displays of Mobile Devices*, p.260-268, Personal and Ubiquitous Computing, 6, 2002.

[30] T. Roosendaal and S. Selleri, *The Official Blender 2.3 guide: free 3D creation suite for modeling, animation, and rendering*, No Starch Press, 2004.

[31] R. J. Rost et al., *OpenGL Shading Language*, Third Edition, Addison-Wesley, 2009.

[32] N. Sarris and M.G. Strintzis, *3D Modeling & Animation*, IRM Press, 2005.

[33] D. Shriener et al., *OpenGL Programming Guide*, Eigth Edition, Addison-Wesley, 2013.

[34] A. Silberschatz et al., *Operating System Concepts*, Addison-Wesley, 1998.

[35] M. Singhal and N.G. Shivaratri, *Advanced Concepts in Operating Systems*, McGraw-Hill, 1994.

[36] Sketchup, *http://www.sketchup.com/intl/en/product/gsu.html*, 2013.

[37] A.S. Tanenbaum, *Modern Operating Systems*, Third Edition, Prentice Hall, 2008.

[38] University of Maryland, *Blendshape Face Animation*, *http://userpages.umbc.edu/bailey/Courses/Tutorials/ModelNurbsHead/BlendShape.html*, 2009.

[39] L. Wood et al., *Document object model (dom) level 1 specification*, W3C Recommendation, 1, 1998.

[40] T.L. Yu, "Chess Gaming and Graphics using Open-Source Tools", *Proceedings of ICC2009*, p. 253-256, Fullerton, California, IEEE Computer Society Press, April 2-4, 2009.

[41] T.L. Yu, D. Turner, D. Stover, and A. Concepcion, "Incorporating Video in Platform-Independent Video Games Using Open-Source Software", *Proceedings of ICCSIT*, Chengdu, China, July 9-11, IEEE Computer Society Press, 2010.

[42] I. Zbib, *3D Face Animation with OpenGL ES: An Android Application*, CSE Master Project Report, School of Computer Science and Engineering, CSUSB, 2013.