Method for Custom Facial Animation and Lip-Sync in an Unsupported Environment, Second Life™

Eric Chance and Jacki Morie

USC Institute for Creative Technologies, 13274 Fiji Way,
Los Angeles, California, USA 90292
{chance,morie}@ict.usc.edu

Abstract. The virtual world of Second Life™ does not offer support for complex facial animations, such as those needed for an intelligent virtual agent to lip sync to audio clips. However, it is possible to access a limited range of default facial animations through the native scripting language, LSL. Our solution to produce lip sync in this environment is to rapidly trigger and stop these default animations in custom sequences to produce the illusion that the intelligent virtual agent is speaking the phrases being heard.

Keywords: lip-sync, Second Life™, virtual world.

1 Introduction

Our current project is an intelligent virtual agent that identifies and greets users, and instructs them how to interact with a game-like environment constructed in the virtual world of Second Life™ (SL). Our intelligent agent can easily print text to screen, but we decided that we could offer greater immersion if we could make the intelligent agent's avatar lip-sync to voice recordings that conveyed strong affective content. This is a problem because SL does not offer support for lip-sync. While SL does allow for many custom avatar animations (in bvh format), the hands and the face of the avatar, specifically, do not allow for customized animations. The user can access a limited range of about 18 default facial animations such as "surprised," "afraid," and "kiss," but there is no supported way to do complex facial animations, such as those needed for lip-sync.

SL has implemented some experimental architecture to accommodate a basic "mouth-flap" animation that moves the avatar's mouth open and closed if a user is streaming sound (voice) through their microphone. The agent could use this feature and stream our recorded audio phrases through the mic channel, but this has disadvantages. It requires the user to have voice enabled in their client program, and the simplistic result does not provide any emotional range, nor does it attempt to match appropriate mouth movements to the rhythm of the speech.

2 Solution

Our solution is to trigger the default facial animations in rapid succession using the native scripting language, LSL. If two facial animations are put together back to back, in rapid succession, we can suggest various morphemes or their constituents. For instance, the puckered lips of avatar_express_kiss can lead into avatar_express_open_mouth, to suggest the sound "bo."

Animations in SL all have an assigned priority, and most of the facial animations have the same priority. This means that the facial animations cannot simply be stacked against each other back-to-back because each subsequent animation will not interrupt the previous one. An animation must be triggered, allowed to play for the appropriate duration, and then that animation must be disabled before the next one is immediately triggered.

This does not cause animations to appear excessively choppy. Most animations in SL, have ease-in and ease-out times, and the facial animations appear to be no exception. What this means is that if an animation is interrupted, the avatar will slowly ease back into the default animation, or the next animation that is triggered, rather than snap back in a jarring manner.

2.1 Example Code

```
    float w=0.1; //seconds to delay before starting next
animation

    llPlaySound( "SoundFile" , 1.0);
    llStartAnimation( "express_anger_emote" );
    llSleep(w);
    llStopAnimation( "express_anger_emote" );
    llStartAnimation( "express_repulsed_emote" );
    llSleep(w*2);
    llStopAnimation( "express_anger_emote");
      //continue until end of audio
```

It is not an elegant solution, but it is far more effective than anything normally possible within the limitations of the platform.

2.2 Other Concerns

Two additional concerns should be considered when using this method: script time dilation (the server dynamically allocates processor time to scripts based on overall simulator performance), and the loading time for each sound. Script time dilation isn't a major concern in most locations, but it could be very minimally improved in certain situations if the value "w" from the example above is altered in response to the value returned by llGetRegionTimeDilation().

The major concern of the two is sound loading time. Most content in SL is streamed to the user, so the sound will play after a brief lag time when the animation sequence is first played. This can be mitigated if the intelligent agent tracks new users and triggers llPreloadSound() for each new user that approaches.