Lipz File Format v1.0

By Erik Hermansen

Overview

The Lipz file format is a simple description of lip animation and subtitles corresponding to an audio file for use in games and other software that performs character animation. The format has the assumption that the game will be responsible in choosing assets and how to animate, and concerns itself with providing just enough information as to be useful for the game.

Here's a quick example of a Lipz file to give you the gist:

```
{
    "text-en_us" :  "Now, boys and girls, I'd like you to put away your readers|and get ready for a very special treat.",
    "visemes" :     "ABBBCABBFGGBBCAABBCDFFGBAAAABAACDEFFFCCBBBABBCCCDB|CCCBBAAGGBBCCDDDBBAACCBBBDDBBCCAAABCCC---",
    "viseme_type" : "toonboom",
    "fps" : 24
}
```

The Lipz file above is associated with one audio file containing the spoken dialogue it describes. The game uses information in the Lipz file to show subtitles and synchronize lip animation of the character as the audio file plays.

The `text` element contains text for the game to display on the screen as subtitles. There is a special "segment delimiter" character (`|`) that shows a point in the text that can be used to segment the text into separate parts for subtitle display. In this case, first a subtitle with "Now, boys and girls, I'd like you to put away your readers" would be shown, followed by a subtitle with "and get ready for a very special treat."

The `visemes` element contains codes that indicate specific visemes (lip frames) to show at different points in time. The segment delimiter (`|`) appears here to mark the point in the audio that corresponds to the same point in the `text` element marked by a segment delimiter.

`viseme_type` and `fps` are used to interpret the codes in the `visemes` element correctly.

Goals of Lipz Format

- Cover use cases for character animation in games including:
  - Synchronizing lip frames (visemes) to audio
  - Subtitles with localization
  - Trigger in-game animation that is timed to audio

- Easy to write parsing code in game software
- Human-readable format that can be hand-edited
- Extensible to cover different viseme types, e.g. mouth charts

Non-Goals

- *Does not* contain information about specific art assets used for animation.
- *Does not* contain settings and information used by animation authoring or rendering tools.
- *Does not* store motion-capture or facial position information.
- *Does not* define animation beyond events triggered by character speech.

Association with Audio File

A Lipz file is associated with one audio file that is expected to contain spoken dialogue. There is no concept of multiple characters or tracks inside of a Lipz file, so typically, the Lipz file will have information for just one character's speech.

The association of the Lipz file to an audio file is managed externally (not specified by Lipz file). An easy convention is to name the Lipz file to match the audio file, e.g. if you have an audio file called `samson-hello.wav` then you could name its corresponding Lipz file `samson-hello.lipz`. And whatever code you have to open up and play audio files, would also look for a Lipz file with a common filename but `.lipz` file extension.

Visemes

Visemes (often mistakenly described as "phonemes") are the visual expression corresponding to a spoken part of dialogue. For example, when you say the word "bowl", your lips will come close together for the "B" sound (a phoneme), and that visual expression is a viseme.

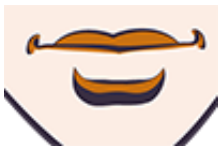
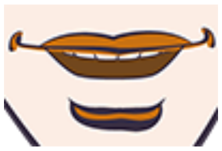
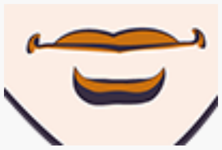
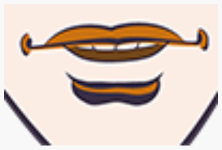
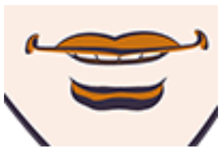
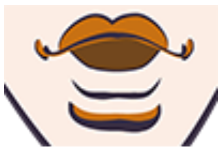
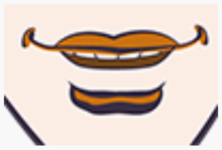
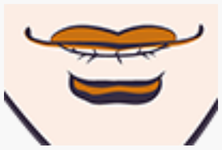
In this version of the Lipz format, three viseme types are defined:

- **Blair** - The classic 9-viseme (+1 for default/resting) mouth chart from Disney animator Preston Blair.
- **Toon Boom** - This viseme type is based on the Toon Boom software's popular 7-viseme mouth chart. It's essentially the same as Preston Blair's mouth chart, but separate "L" and "U" visemes aren't included.
- **RMS** - This viseme type represents an amplitude average (RMS) for each frame of animation. If the audio is louder, the mouth will be open wider. If the audio is silent or below a certain threshold, the mouth will be

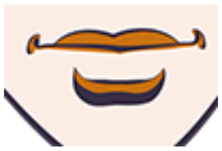
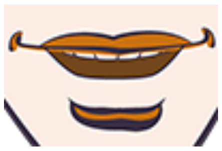
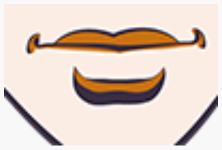
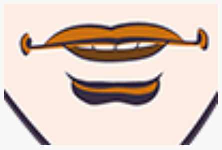
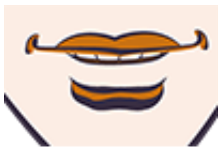
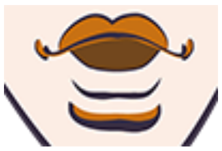
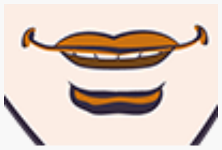
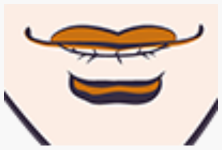
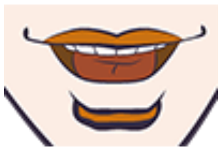
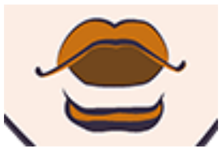
closed.

The hyphen (`-`) viseme code is reserved across all viseme types to indicate a resting mouth position, or in other words, the character is not actively speaking. It differs from visemes that specifically have the mouth closed, because the character may have a resting mouth position that is not closed. Imagine, for example, an energetic politician that grins constantly while in public.

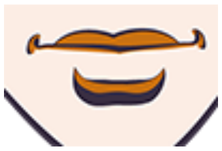
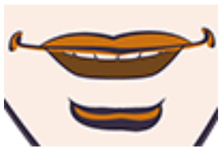
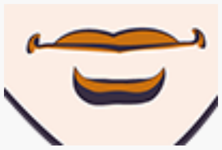
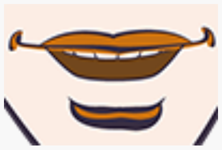
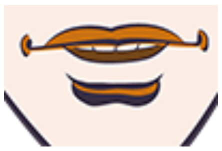
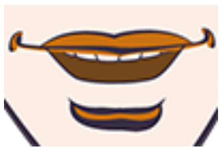
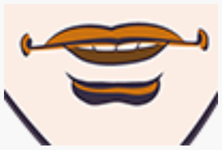
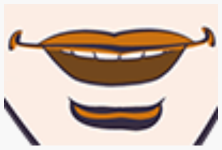
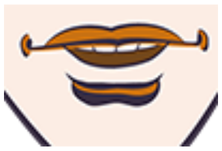
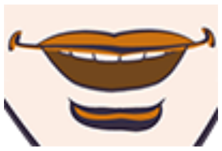
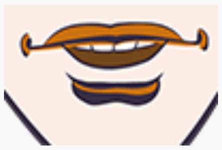
Toon Boom Viseme Codes

Code	Example Image	Description	Code	Example Image	Description
`-`		Resting	`D`		A and I
`A`		M, B, and P	`E`		O
`B`		Most consonants	`F`		W and Q
`C`		E	`G`		F and V

Blair Viseme Codes

Code	Example Image	Description	Code	Example Image	Description
`-`		Resting	`A`		A and I
`M`		M, B, and P	`O`		O
`S`		Most consonants	`W`		W and Q
`E`		E	`F`		F and V
`L`		L	`U`		U

RMS Viseme Codes

Code	Example Image	Description	Code	Example Image	Description
–		Resting	5		
0		Closed	6		
1		Minimally open	7		"Yell" open
2			8		
3			9		Fully open
4					

File Format Definition

The Lipz file format uses JSON. It is UTF-8 encoded, so may contain Unicode characters. Though JSON supports a hierarchical structure, all elements are declared flat under the root element. Elements that can be used are described in the table below. All elements are optional, but the usefulness of the file to a game relies on at least some of the elements to be present.

Element	Type	Constraints	Description
`fps`	*number*	Between 1 and 1000	Frame rate of viseme specification expressed in frames per second. Used to interpret values of `viseme` elements, where one character equals one frame. Note that the game frame rate is not tied to this value. If unspecified, the value is supplied externally, e.g. a known constant value in game code or a general settings file.
`text_` *ll* `-` *cc*	*string*	Segment delimiter count must match `visemes`.	Dialogue text that corresponds to audio with the additional specification of a language locale. The language locale consists of a lower-case, 2-character, ISO 639-1 language code followed by a hyphen (`-`) followed by a lower-case, 2-character, ISO 3166-1 country code. Example: `text_es-mx` (Mexican Spanish). Text may be segmented with use of the `|` character (see more about this in "Segmenting" section). Custom meta-information not intended for display can be stored with the text inside of curly braces (`{` and `}`).
`visemes`	*string*	Constrained according to `viseme_type`.	Each character represents the viseme for one frame of animation. The exception is the `|` character which delineates segments, and does not count for a frame.
`viseme_type`	*string*	"blair", "toonboom", or "rms"	The type of viseme encoding used by the `visemes` element. If unspecified, the value is supplied externally, e.g. a known constant value in game code or a general settings file.
`events`	*string*	Segment delimiter count must match `visemes`.	Events that can be correlated to segments to trigger emotions, gestures, and other animations that should be synchronized with audio. The syntax of the events is arbitrary and determined by what the game wishes to support, but this document offers the "Character Event Syntax" as a suggested starting point.

Using Only the Lipz Elements Important to Your Project

All of the elements in the file format are optional. If you are authoring Lipz files for your project, here is a checklist of questions to decide which elements to use. Let's assume your project will be performing lip synch based on Lipz files, because it becomes hard to recommend use of Lipz otherwise.

- **Subtitles** - Will your game display subtitles? If yes...
    - **Short Audio** - Will your audio clips be short enough that one audio file would correspond to one displayed subtitle?
- **External** - Do you prefer to keep project-wide common settings out of individual Lipz files?
- **Events** - Do you want to time character animation events to specific points in dialogue audio playback? If yes...
    - **Event Context** - Do you want to include text in the Lipz file so you can see context needed for editing events?

If you know the answers to the above questions, you can find your recommended elements to use from the table below.

Element	Use If Answered
`fps`	"No" to **External**
`text_` *ll* `-` *cc*	"Yes" to **Subtitles** and **Localized Subtitles**. Or "Yes" to **Events** and **Event Context**.
`visemes`	Use this element in all cases.
`viseme_type`	"No" to **External**
`events`	"Yes" to **Events**

Use segmenting (inclusion of `|` segment delimiter character) according to table below.

Feature	Use If Answered
segmenting	"Yes" to **Subtitles** and "No" to **Short Audio** Or "Yes" to **Events** and "No" to **Short Audio**.

The rationale is that segmenting is useful for either subtitles or tying character animation events to dalogue audio, but if the audio files are split into small enough clips, then segmenting with a Lipz file is no longer useful.

Segmenting

Segmenting is optional. It can be used to correlate ranges of frames to text or events. With segmenting, the following is possible:

- Subtitles displayed on a screen can be timed to correspond to the playback of matching dialogue from an audio file.
- Algorithms can be devised for displaying subtitles in an automated and localization-optimized way, e.g. "display however many segments fit on screen" will handle large differences in translation text length better than just "display segment X on screen".
- Animated character emotions and gestures can be timed to correspond to an exact moment of playing dialogue.

Segments are always delineated with the `|` character. When used within the `visemes` element, they indicate the exact frame of a segment's boundary. In the example below, the first segment lasts 20 frames, and is then followed by the second segment.

```
{
    "visemes" : "125643356734563-----|9723399421---"
}
```

If text is specified that also contains segmenting characters (`|`) then the segments defined in the text should be interpreted as corresponding to the segments defined in the viseme. This also implies timing values from the visemes correspond to the text. In the example below, the game code that loads the Lipz file will know that the "It was you!" text begins on the 20th frame. The convention of stripping leading and trailing white space from segments of the `text` element before displaying allows the Lipz file to visually align segments of `text` element inside the Lipz file for easier editing.

```
{
    "visemes" :    "125643356734563-----|9723399421---",
    "text_en-us" : "Let me think...     |It was you!"
}
```

Imagine a character on the screen who thinks carefully for a moment, ("Let me think...") and then screams an unexpected accusation ("It was you!"). For dramatic effect, it may be better for the game to show the second segment of text only after the corresponding dialogue has been played. Segmenting gives you this level of control.

Adding to this example, it would be extra-dramatic to have the character point angrily at the accused at the moment the second segment begins. For this, we can add an `events` element.

```
{
    "visemes" :    "125643356734563-----|9723399421---",
    "text_en-us" : "Let me think...     |It was you!",
    "events" :     "                    |points butler"
}
```

For the event to do anything in the game, you'd need to write handling that looked for "points butler" in a currently playing segment to trigger a pointing-at-butler animation.

If the game implements a scripting container, you might alternatively include script to execute. A LUA example is shown below.

```
{
    "visemes" :    "125643356734563-----|9723399421---",
    "text_en-us" : "Let me think...     |It was you!",
    "events" :     "                    |pointAt(butler)"
}
```

Knowing which Viseme to Show

The game code will perform the tasks of loading an audio file that contains dialogue. At the same time it does this, it should also load the Lipz file that corresponds to the audio file so that visemes and other information will be available later when the audio file is played.

When the game code plays an audio file, there should be some facility of learning the current play position within the audio file. At any point where a speaking character's lips should be updated, a function like the one below can be called to learn the current viseme that should be used for display. The game code can then use whatever facility is provided for animation to set the character's lips to match the current viseme.

```
//Returns one character code of the current viseme corresponding to playing dialogue
 audio.
function getCurrentViseme(
        playPositionSecs, //Float of seconds elapsed since dialogue audio began play
ing.
        visemes,          //String loaded from "visemes" element of Lipz file.
        fps) {            //Frame rate used for interpreting length of each frame in
 "visemes".

    var unsegVisemes = visemes.replace(/\|/g, ''); //Removes any segment delimiter "
|" characters.
    var frameNo = Math.floor( playPositionSecs * fps );
    if (frameNo < 0 || frameNo >= visemes.length) { //Frame# is out of bounds.
        return '-'; //Return "resting mouth" viseme as a safe default.
    } else { //Frame# is valid.
        return unsegVisemes.charAt(frameNo);
    }
}
```

Character Event Syntax

A game may optionally implement support for some or all of the character events described in this section, or merely use it as a jumping off point for it's own unique character event syntax. Character events are passed to the game via the `events` element. The events that are defined here are focused on kinds of animation that should happen with timing that is synchronized to audio.

In the example below, a speaking character begins feeling sad, then his mood changes to irritated, and he finishes with a glance to another character he is speaking with. Each character event is specified at the beginning of a segment in the `text` element, and the timing of the segments is defined by the segment delimiters found in the `visemes` element.

```
{
    "visemes" :    "SOOSSAAMSAAASESLOOLEE--|SOOSESSOSOOSEES-------|SOOSOOOOO--",
    "text-en_us" : "Sometimes I get lonely.|Nothing to do here,   |you know?",
    "events" :     "feels sad              |feels irritated       |sees amy",
}
```

Tying emotions and actions to dialogue text is easily done without thinking about animation. In this way, a writer with a flair for visual drama can quickly do a lot of work that is usually performed by an animator.

Often, it is another character besides the currently speaking one that reacts to dialogue. In this case, the event will specify the name of a character that the event applies to.

```
{
    "visemes" :    "ASAWAASASAAASSS--------------|AAMEEEEEE----",
    "text-en_us" : "And the winner is...         |Amy!",
    "events" :     "amy feels nervous            |amy feels happy"
}
```

We may wish to specify multiple events occuring in one segment. This can be done with a "," separator. In the example below, a second contender's hopes are dashed.

```
{
    "visemes" :    "ASAWAASASAAASSS--------------|AAMEEEEEE----",
    "text-en_us" : "And the winner is...         |Amy!",
    "events" :     "amy feels nervous,rob feels nervous|amy feels happy,rob feels ha
ppy"
}
```

Note that it's not necessary to use whitespace inside of element values to line up segment delimiters (`|`). This is just a formatting nicety that is available.

"Feels" Command

Set an emotion for the character. Emotions are treated as persistent modes of the character.

Syntax: [**character**] feels **emotion**

variable	description
character	ID of a character in the scene. If omitted, the character for which the Lipz file is playing should be used.
emotion	ID of an emotion for character to express. See "Emotions".

Emotions

value	description
neutral	normal or default face for character
amused	
happy	
sad	
irritated	
angry	
suspicious	
confused	
afraid	
thinking	lost in thought, concentrating
evil	malicious glee

"Sees" Command

Character will look at a prop or another character for at least a moment. Game should be implemented to have characters automatically look at interesting things like people talking. This command will override the default target of attention temporarily.

Syntax: [**character**] sees { **prop** | **target-character** }

variable	description
character	ID of a character in the scene. If omitted, the character for which the Lipz file is playing should be used.
prop	ID of a prop in the scene.
target-character	ID of a character in the scene.

"Stares" Command

Character will look at a prop or another character and continue to look. Game should be implemented to have characters automatically look at interesting things like people talking. This command will override the default target of attention until the "stop staring" command is given.

Syntax: [**character**] stares { **prop** | **target-character** }

variable	description
character	ID of a character in the scene. If omitted, the character for which the Lipz file is playing should be used.
prop	ID of a prop in the scene.
target-character	ID of a character in the scene.

"Stops Staring" Command

Character will stop staring, returning to a normal state of attention.

Syntax: [**character**] stops staring

variable	description
character	ID of a character in the scene. If omitted, the character for which the Lipz file is playing should be used.

"Points" Command

Character points at a prop or another character and continue to point. The state persists until the "stop pointing" command is given.

Syntax: [**character**] points { **prop** | **target-character** }

variable	description
character	ID of a character in the scene. If omitted, the character for which the Lipz file is playing should be used.
prop	ID of a prop in the scene.
target-character	ID of a character in the scene.

"Stops Pointing" Command

Character will stop pointing. For this command to have effect, a previous "points" command for the character must be specified.

Syntax: [**character**] stops pointing

variable	description
character	ID of a character in the scene. If omitted, the character for which the Lipz file is playing should be used.

"Does" Command

Syntax: [**character**] does **action**

variable	description
character	ID of a character in the scene. If omitted, the character for which the Lipz file is playing should be used.
action	ID of a game-defined action.