实验三， 数字人展示补充材料

主要的工作：

1. 模型简化
2. 创建Mesh with morph targets
3. **Lip Sync技术，**phoneme，threejs编程

# 模型简化

**Addons for Model Optimization**

* [**DecalMachine**](https://blendermarket.com/products/DECALmachine)**:**  
  ARKit applications often require low-poly models for performance reasons. DecalMachine helps optimize models while retaining high-quality details by baking them into decals.
* [**Simplygon Blender Addon (Third Party)**](https://www.simplygon.com/)**:**  
  Simplygon is a powerful tool for automatic model optimization, reducing polygon count while preserving visual fidelity. It's great for preparing models for ARKit projects.
* [**MESHmachine**](https://blendermarket.com/products/MESHmachine)**:**  
  This addon is useful for refining topology and fixing issues in imported meshes. It’s especially helpful when working with models exported from ARKit or scanned models.

**Addons for Texture Baking and UV Mapping**

* [**UVPackmaster**](https://blendermarket.com/products/uvpackmaster)**:**  
  Texture optimization is critical in AR applications. UVPackmaster is a powerful UV mapping addon that helps you create efficient UV layouts, which is essential for ARKit models.
* [**Baketool**](https://blendermarket.com/products/baketool)**:**  
  This addon streamlines the process of baking textures, which is useful for ARKit models that require baked lighting or texture maps.
* [**Substance Painter Live Link**](https://github.com/MatthiasHaberkorn/Blender-Substance-Painter-Live-Link)**:**  
  If you're texturing ARKit models in Substance Painter, this live link addon allows real-time synchronization between Blender and Substance Painter.

# Morph targets创建

Morph targets are a way of animating 3D models by blending between predefined shapes (targets) using a set of vertex positions. For lip sync, you'd need to have a set of target shapes that correspond to different phonemes (speech sounds).

* **Mesh with morph targets**: Your 3D face model should be created with multiple morph targets that represent different facial expressions or phonemes.
* **Target vertices**: Each morph target consists of a set of target vertex positions, and the final mesh shape is computed by blending between these targets based on some weighting factor.
* ARKit (Apple’s augmented reality framework) supports a set of **blend shapes** (morph targets) that allow facial expressions to be tracked and animated in real-time on iOS devices. These blend shapes are specifically designed to capture the nuances of human facial expressions and can be used to animate the face model based on camera input or predefined animations (like lip sync from audio).

主要过程

* **Create the base model**: Create a neutral expression base model (resting face).
* **Add morph targets**: For each phoneme (like "A", "E", "O", "M", "P", etc.), create a variation of the face with that phoneme shape.
* **Export model**: Export your model with these morph targets into a format that Three.js can read, such as GLTF or FBX. GLTF is recommended as it's efficient and well-supported in Three.js.
* **ARKit Blend Shape List**
* Here’s the full list of ARKit blend shapes to create in Blender:

| **Blend Shape Name** | **Description** |
| --- | --- |
| **browDown\_L** | Lowering left brow. |
| **browDown\_R** | Lowering right brow. |
| **browInnerUp** | Raising inner brow. |
| **browOuterUp\_L** | Raising outer left brow. |
| **browOuterUp\_R** | Raising outer right brow. |
| **cheekPuff** | Puffing cheeks. |
| **cheekSquint\_L** | Squinting left cheek. |
| **cheekSquint\_R** | Squinting right cheek. |
| **eyeBlink\_L** | Blinking left eye. |
| **eyeBlink\_R** | Blinking right eye. |
| **eyeLookDown\_L** | Looking down with left eye. |
| **eyeLookDown\_R** | Looking down with right eye. |
| **eyeLookIn\_L** | Looking inward with left eye. |
| **eyeLookIn\_R** | Looking inward with right eye. |
| **eyeLookOut\_L** | Looking outward with left eye. |
| **eyeLookOut\_R** | Looking outward with right eye. |
| **eyeLookUp\_L** | Looking up with left eye. |
| **eyeLookUp\_R** | Looking up with right eye. |
| **eyeSquint\_L** | Squinting left eye. |
| **eyeSquint\_R** | Squinting right eye. |
| **eyeWide\_L** | Widening left eye. |
| **eyeWide\_R** | Widening right eye. |
| **jawForward** | Moving jaw forward. |
| **jawLeft** | Moving jaw left. |
| **jawRight** | Moving jaw right. |
| **jawOpen** | Opening jaw. |
| **mouthClose** | Closing mouth. |
| **mouthDimple\_L** | Dimpling left side of mouth. |
| **mouthDimple\_R** | Dimpling right side of mouth. |
| **mouthFrown\_L** | Frowning left corner of mouth. |
| **mouthFrown\_R** | Frowning right corner of mouth. |
| **mouthFunnel** | Funnel shape with mouth. |
| **mouthLeft** | Pulling mouth left. |
| **mouthRight** | Pulling mouth right. |
| **mouthLowerDown\_L** | Lowering left side of lower lip. |
| **mouthLowerDown\_R** | Lowering right side of lower lip. |
| **mouthPress\_L** | Pressing lips on left side. |
| **mouthPress\_R** | Pressing lips on right side. |
| **mouthPucker** | Puckering lips. |
| **mouthRollLower** | Rolling lower lip. |
| **mouthRollUpper** | Rolling upper lip. |
| **mouthShrugLower** | Lowering lower lip. |
| **mouthShrugUpper** | Raising upper lip. |
| **mouthSmile\_L** | Smiling with left corner. |
| **mouthSmile\_R** | Smiling with right corner. |
| **mouthStretch\_L** | Stretching left corner. |
| **mouthStretch\_R** | Stretching right corner. |
| **mouthUpperUp\_L** | Raising left side of upper lip. |
| **mouthUpperUp\_R** | Raising right side of upper lip. |
| **noseSneer\_L** | Sneering left side of nose. |
| **noseSneer\_R** | Sneering right side of nose. |
| **tongueOut** | Sticking out the tongue. |

**Rigging/Face Tracking Addons**

* **Blender's Built-in Shape Key System:**  
  For ARKit, creating shape keys that match Apple's 52 facial blendshapes is critical. Blender's shape key system can be used to create these expressions, and you can use addons to streamline the process.
* [**Auto-Rig Pro**](https://blendermarket.com/products/auto-rig-pro)**:**  
  Auto-Rig Pro is a powerful rigging tool that can help you rig humanoid models for animation and AR applications. It supports custom face rigs and can be used to retarget ARKit face blendshapes.

**Features:**

* + Easy rigging for characters.
  + Face rigging that can be customized for ARKit blendshapes.
  + Includes export tools for GLTF and USDZ (formats used in ARKit).
* [**FaceIt Addon**](https://blendermarket.com/products/faceit)**:**  
  FaceIt is a Blender addon designed specifically for facial rigging and animation. It supports ARKit blendshapes and automates the process of creating them for AR applications.

**Features:**

* + Automatic generation of ARKit-compatible blendshapes.
  + Directly exports facial rigs for ARKit workflows.
  + Works well with high-resolution face models.

# Lip sync技术

Blender + Rhubarb Lipsync 插件的使用<https://www.youtube.com/watch?v=vZVtUEEssxQ>

库的位置 https://github.com/DanielSWolf/rhubarb-lip-sync

**3. Audio Analysis for Lip Sync**

To perform lip sync, you need to analyze the input audio (the speech or dialogue) and map it to the corresponding phoneme shapes at each point in time.

* **Phoneme Mapping**: Using the transcription, you can extract phonemes using libraries such as:
  + **CMU Sphinx** (Pocketsphinx): A lightweight speech recognition library that can output phonemes directly from speech.
  + **Web Speech API**: Some browsers provide a speech recognition API that can output phonemes and timestamps.
* **Audio to Phoneme Mapping**: Alternatively, you can directly analyze the audio waveform to detect the phoneme transitions using libraries like:
  + **TensorFlow.js**: You can use machine learning models for phoneme recognition from audio input.
  + **PyDub or librosa** (for pre-processing the audio file in a server-side app, then sending phoneme timings to the Three.js frontend).

**4. Animating Lip Sync in Three.js**

Once you have the phoneme information from the audio, you'll need to map that to the morph targets in Three.js. Here's how to approach it:

**Step-by-Step Process:**

1. **Load your 3D Model with Morph Targets**:
   * Load your 3D model using THREE.GLTFLoader (for GLTF models). Ensure that the model has morph targets.

Example:

javascript

复制代码

const loader = new THREE.GLTFLoader();

loader.load('path/to/model.glb', function (gltf) {

const mesh = gltf.scene.children[0];

mesh.morphTargetInfluences = [];

mesh.morphTargetDictionary = gltf.scene.morphTargetDictionary;

scene.add(mesh);

});

1. **Map Phonemes to Morph Targets**:
   * Create a mapping between the phonemes and the corresponding morph target indices. For example:

javascript

复制代码

const phonemeMapping = {

"A": 0, // index of the 'A' shape key

"E": 1, // index of the 'E' shape key

"O": 2, // index of the 'O' shape key

// Add more phoneme to target index mappings

};

1. **Animate Morph Targets**:
   * As you play back the audio, adjust the morph target influences based on the phoneme you want to display at each moment.
   * Use THREE.AnimationMixer or requestAnimationFrame to drive the morph target influence over time.

Example:

javascript

复制代码

function updateLipSync(phoneme, weight) {

// Get the corresponding morph target index

const targetIndex = phonemeMapping[phoneme];

if (targetIndex !== undefined) {

mesh.morphTargetInfluences[targetIndex] = weight;

}

}

1. **Syncing with Audio**:

• Pre-recorded Phoneme Timings: Use a tool like CMU Sphinx or an API like Google Speech-to-Text to extract phoneme timings from the audio.

• Map Phonemes to Blend Shapes: Create a mapping between phonemes (such as “A”, “E”, “O”, “M”, etc.) and the ARKit blend shapes (like mouthFunnel, jawOpen, etc.).

* + Use the Web Audio API to analyze the audio in real-time or use pre-computed phoneme timings to trigger the right morph target influences at the right time.

Example (Web Audio API for syncing):

javascript

复制代码

const audioContext = new (window.AudioContext || window.webkitAudioContext)();

const analyser = audioContext.createAnalyser();

const source = audioContext.createBufferSource();

source.buffer = audioBuffer; // Load an audio buffer from file or stream

source.connect(analyser);

analyser.connect(audioContext.destination);

source.start();

// Use an analyser to analyze the audio and adjust lip sync

function syncWithAudio() {

let buffer = new Float32Array(analyser.frequencyBinCount);

analyser.getFloatFrequencyData(buffer);

// Based on the audio data, trigger appropriate phonemes or mouth shapes

updateLipSync(currentPhoneme, influenceWeight);

}

**5. Libraries & Tools**

* **Three.js**: The core library for rendering and working with morph targets.
* **Web Audio API**: For analyzing and processing audio data in the browser.
* **GLTFLoader**: For loading GLTF models with morph targets in Three.js.
* **TensorFlow.js**: If you're implementing machine learning models for phoneme detection directly in the browser.
* **Tone.js**: If you need additional audio handling, like synthesizing sounds or manipulating audio tracks.

**Example Mapping for Lip Sync:**

javascript

复制代码

const phonemeToBlendShape = {

"A": "mouthOpen", // ARKit blend shape for mouth opening

"E": "mouthSmileLeft", // ARKit blend shape for smile

"O": "mouthFunnel", // ARKit blend shape for "O" shape

// More phoneme-to-blend shape mappings here

};

* **Animate the Lip Sync**: Use the phoneme timing to gradually update the morph target influences over time (similar to how I described in the previous response).

javascript

复制代码

function updateLipSync(phoneme, weight) {

const targetBlendShape = phonemeToBlendShape[phoneme];

if (targetBlendShape) {

const shapeIndex = mesh.morphTargetDictionary[targetBlendShape];

if (shapeIndex !== undefined) {

mesh.morphTargetInfluences[shapeIndex] = weight;

}

}

}

# **Ready Player Me**

1. Use ARKit blend shapes (from Ready Player Me models).
2. Extract phonemes from the audio.
3. Map phonemes to the appropriate blend shapes.
4. Animate the blend shapes over time to match the speech.

This approach allows for high-quality, real-time facial animations with minimal setup.

### ****Creating and Uploading Your Custom Face Model to Ready Player Me****

As of the latest updates, **Ready Player Me** doesn’t directly allow users to upload custom 3D models (e.g., a face model) directly into their platform. Instead, they focus on avatar customization. However, Ready Player Me supports **importing models from other 3D programs** (like Blender or Maya) through their SDKs, but these models would need to fit specific parameters (i.e., humanoid avatars with ARKit-compatible blend shapes).

To upload and use your own custom face model, you would need to:

1. **Prepare Your Custom Face Model**:
   * Use a 3D software like **Blender** to create or modify a face model with the necessary facial blend shapes for lip sync.
   * Ensure your model uses **ARKit-compatible blend shapes** such as:
     + jawOpen, mouthSmileLeft, mouthFunnel, eyeBlinkLeft, etc.
   * Export this model as **GLTF** or **FBX** file formats.
2. **Create ARKit-Compatible Blend Shapes in Blender**: If your face model doesn’t yet have ARKit-compatible blend shapes, you can add them in **Blender** using shape keys (morph targets). There are tutorials available online to guide you through adding blend shapes for lip sync or ARKit compatibility. Here's an overview of the steps:
   * **Import Your Face Model** into Blender.
   * **Create Shape Keys** for the blend shapes required for lip sync (e.g., mouthSmileLeft, mouthOpen, eyeBlinkLeft).
   * **Shape Key Editing**: Adjust the vertex positions for each shape key to match the appropriate facial expressions.
   * **Export the Model** in **GLTF** or **FBX** format.

Once you've created the blend shapes, you can export the model in a format that **Ready Player Me** can use (GLTF or FBX).

1. **Integrate with Ready Player Me SDK** (Optional): If you're looking to build a custom solution that integrates your own model with Ready Player Me's avatars, you can use their **SDK** to load custom models and possibly integrate them into their avatar system. The SDK provides support for creating custom avatars and adding ARKit-compatible features.
2. **Upload to Ready Player Me**: You can upload your model using the **Ready Player Me SDK** in a more advanced setting (for example, for games or other applications). To do this, you would:
   * Set up an account with Ready Player Me.
   * Follow the documentation on how to use their SDK for creating custom avatars. This allows you to integrate your custom model into Ready Player Me's ecosystem, but you may need to adjust things for ARKit compatibility.

Visit the official documentation for Ready Player Me SDK here: Ready Player Me SDK Documentation.

###### 