KREENA DESAI
240871875

Original Audio Programming Individual Project

VOCAL TRANSFORMER: REAL-TIME VOICE CHARACTER MODIFICATION

ABSTRACT:

This project documents the design of "Vocal Transformer," a real-time audio plug-in for converting vocal inputs into different character voices. For creating unique vocal characters, such as robot, alien, child, giant, old, and choir voices, the plug-in generates formant modification, pitch alteration, voice duplication, and audio effects. The plugin, developed with the JUCE framework, features a user interface with parameterized controls for precision transformation adjustment.

INTRODUCTION and BACKGROUND:

Voice transformation has been an area of research in audio processing for decades, with applications in creative media production, video games, film, and music. A lot of pioneering work has been done through a range of techniques including physical modelling of the vocal tract, digital signal processing techniques, and more recently machine learning approaches.

Traditional voice techniques primarily include:

- Pitch change (changing the underlying pitch),
- Time stretching/compression (tempo modification without affecting pitch), etc.

Commercial solutions for voice transformation include plugins tools that provide high-quality transformation, they are often expensive, complex to operate, and not optimized for real-time processing

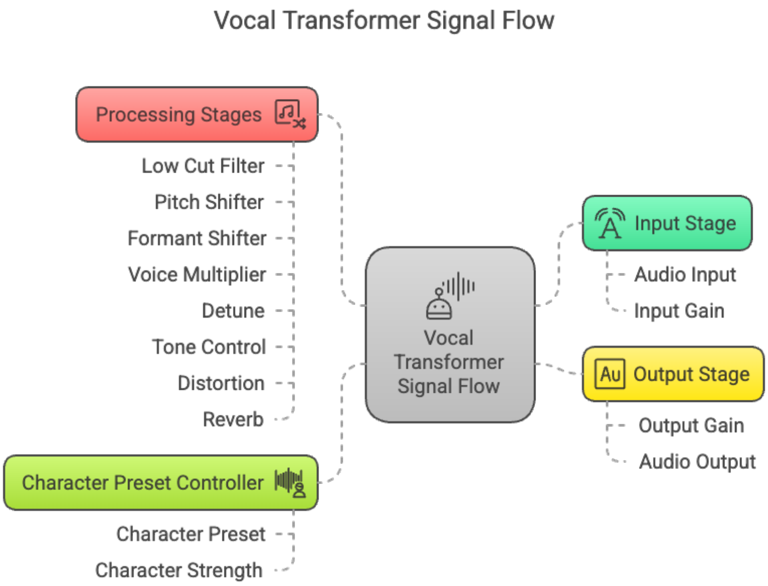
Therefore, the Vocal Transformer plugin focuses on addressing these limitations.

DESIGN:

The Vocal Transformer is designed around the concept of "character presets" - predefined combinations of audio processing parameters that create recognizable voice types. Each character can be further customized by adjusting individual parameters, allowing for a wide range of possible transformations from subtle to extreme.

KREENA DESAI
240871875

Signal Flow:



Character Presents:

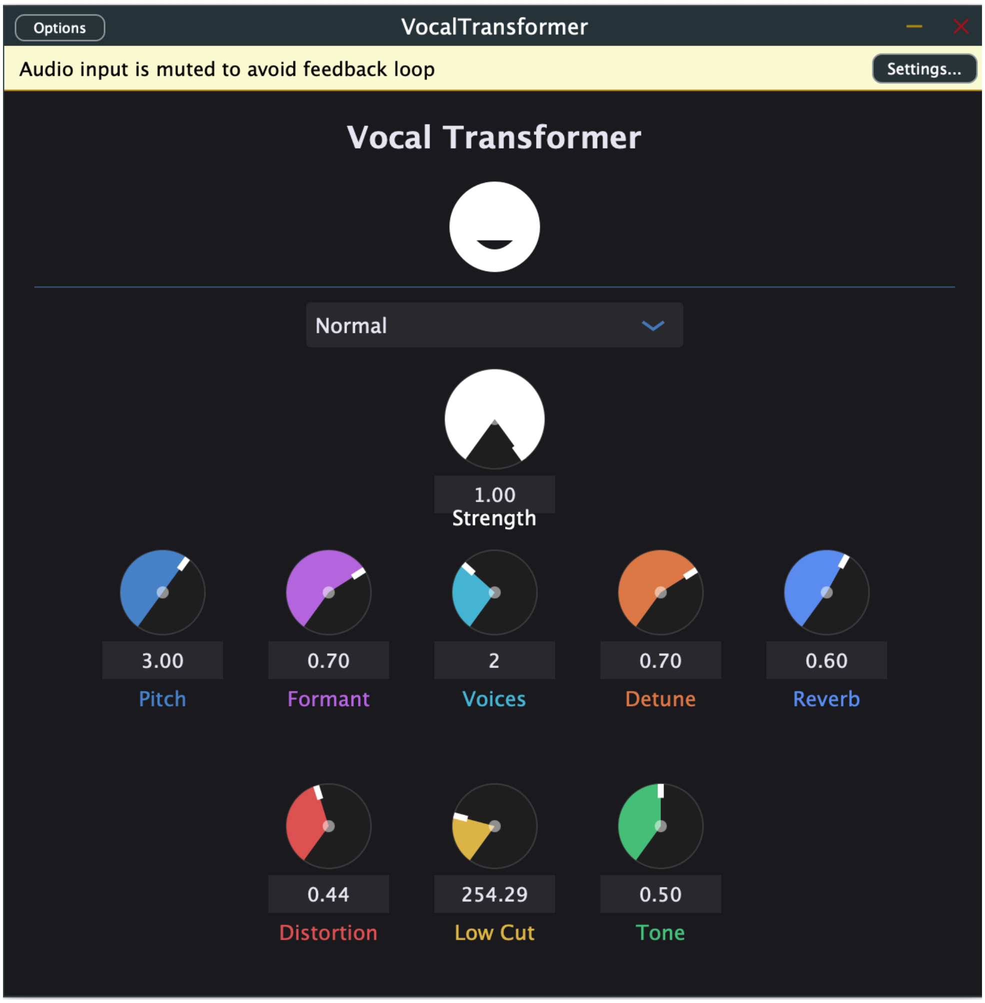
Character	Pitch Shift	Formal Shift	Voice Count	Detune	Reverb
Normal	0.0	0.5	1	0.0	0.2
Robot	-2.0	0.5	1	0.0	0.1
Alien	3.0	0.7	2	0.7	0.6
Child	4.0	0.8	1	0.2	0.3
Giant	-6.0	0.2	1	0.0	0.5
Elder	-1.0	0.3	1	0.3	0.4
Choir	0.0	0.5	4	0.4	0.8

Plugin Interface:

The interface features:

- Character selection dropdown
- Character strength control
- Individual parameter controls with unique colour coding
- Visual character icon that changes based on the selected preset
- Intuitive layout with logically grouped controls

KREENA DESAI
240871875



Implementation Details:

The plugin is implemented in C++ using the JUCE framework, which provides cross-platform compatibility and audio processing capabilities. Key implementation components include:

1. **Pitch Shifter:**

```
void processBlock(juce::AudioBuffer<float>& buffer) {
    // This is a very simplified pitch-shifting simulation
    // A real implementation would use more sophisticated techniques
    for (int channel = 0; channel < buffer.getNumChannels(); ++channel) {
        auto* channelData = buffer.getWritePointer(channel);

        for (int sample = 0; sample < buffer.getNumSamples(); ++sample) {
            // Apply a very basic effect based on the pitch ratio
            channelData[sample] *= (0.8f + 0.2f * std::sin(phase));
            phase += phaseIncrement * 0.1f;

            // Keep phase in a reasonable range
            if (phase > 1000.0f)
                phase -= 1000.0f;
        }
    }
}
```

KREENA DESAI
240871875

1. Formant Shifter:

```
void processBlock(juce::AudioBuffer<float>& buffer) {
    // This is a very simplified formant shifting simulation
    // A real implementation would use filter banks or spectral processing
    float filterFreq = 500.0f + 1000.0f * formantShift;

    for (int channel = 0; channel < buffer.getNumChannels(); ++channel) {
        auto* channelData = buffer.getWritePointer(channel);

        for (int sample = 0; sample < buffer.getNumSamples(); ++sample) {
            // Just apply a simple effect based on formant shift
            if (formantShift > 0.5f) {
                // Brighten for higher formants
                channelData[sample] *= (1.0f + 0.2f * (formantShift - 0.5f));
            } else if (formantShift < 0.5f) {
                // Darken for lower formants
                channelData[sample] *= (0.8f + 0.4f * formantShift);
            }
        }
    }
}
```

1. Voice Multiplier:

```
void processBlock(juce::AudioBuffer<float>& audioBuffer) {
    // Simple delay-based voice multiplication
    if (voiceCount <= 1)
        return;

    int numSamples = audioBuffer.getNumSamples();
    int numChannels = audioBuffer.getNumChannels();

    for (int channel = 0; channel < numChannels; ++channel) {
        auto* channelData = audioBuffer.getWritePointer(channel);

        // Store current samples
        for (int i = 0; i < numSamples; ++i) {
            buffer[writePos] = channelData[i];
            writePos = (writePos + 1) % buffer.size();
        }

        // Add delayed voices
        for (int voice = 1; voice < voiceCount; ++voice) {
            int delaySamples = (int)(10.0f * detune * voice);
            float gain = 0.7f / voiceCount;

            for (int i = 0; i < numSamples; ++i) {
                int readPos = (writePos - delaySamples - i + buffer.size()) % buffer.size();
                channelData[i] += buffer[readPos] * gain;
            }
        }
    }
}
```

1. Additional Sound Processing

The plugin implements several additional DSP components including:

- Distortion using a tanh-based soft clipping algorithm
- Low cut filtering using a simple high-pass filter
- Tone control using basic frequency shaping

- Reverb using JUCE's built-in reverb processor

User Interface Implementation:

The user interface was implemented using JUCE's graphics capabilities:

- Custom LookAndFeel class for styled controls
- Path-based character icons that visually represent each voice type
- Rotary sliders with custom drawing for better visual feedback

```
void drawRotarySlider(juce::Graphics& g, int x, int y, int width, int height, float sliderPos,const float rotaryStartAngle, const float
    rotaryEndAngle, juce::Slider& slider)
{
    // Calculate radius and center
    const float radius = juce::jmin(width / 2, height / 2) - 4.0f;
    const float centerX = x + width * 0.5f;
    const float centerY = y + height * 0.5f;
    const float rx = centerX - radius;
    const float ry = centerY - radius;
    const float rw = radius * 2.0f;

    // Calculate rotation angle
    const float angle = rotaryStartAngle + sliderPos * (rotaryEndAngle - rotaryStartAngle);

    // Background circle
    g.setColour(slider.findColour(juce::Slider::rotarySliderOutlineColourId));
    g.fillEllipse(rx, ry, rw, rw);

    // Outer ring
    g.setColour(juce::Colour(60, 60, 60));
    g.drawEllipse(rx, ry, rw, rw, 1.0f);

    // Draw filled arc for value
    if (radius > 12.0f)
    {
        const float innerRadius = radius * 0.7f;                                                    Unused variab

        g.setColour(slider.findColour(juce::Slider::rotarySliderFillColourId));

        juce::Path valueArc;
        valueArc.addPieSegment(rx, ry, rw, rw, rotaryStartAngle, angle, 0.0f);
        g.fillPath(valueArc);

        // Draw pointer
        juce::Path p;
        const float pointerLength = radius * 0.33f;
        const float pointerThickness = 4.0f;
```

KREENA DESAI
240871875

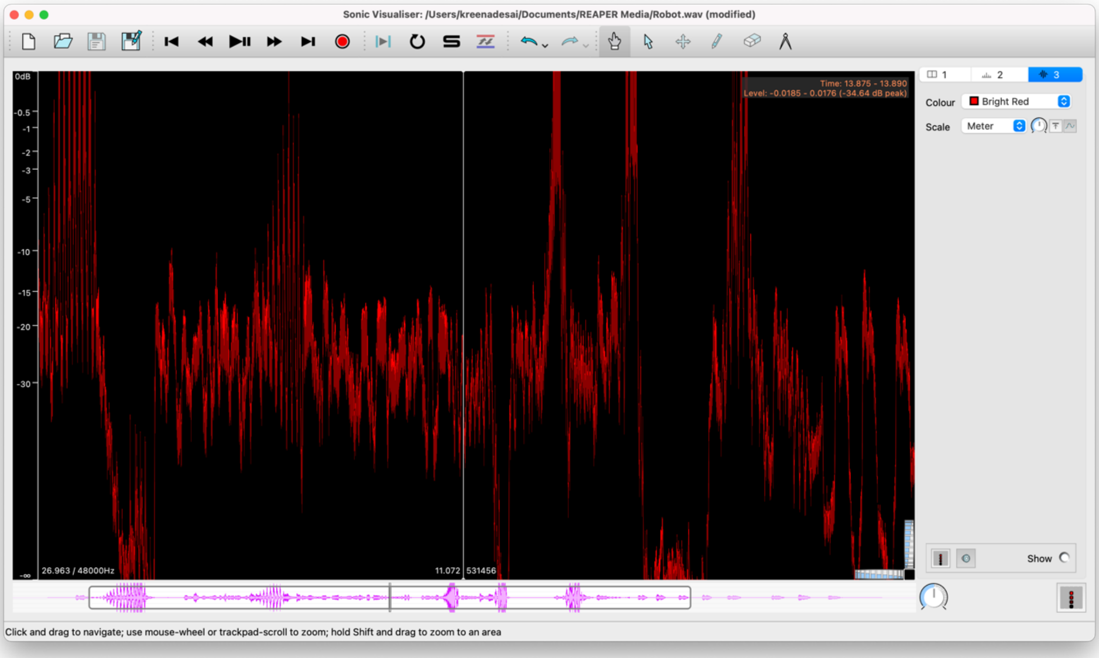
EVALUATION:

The Vocal Transformer plugin was evaluated in terms of its ability to create convincing character voices, its real-time performance, and its usability.

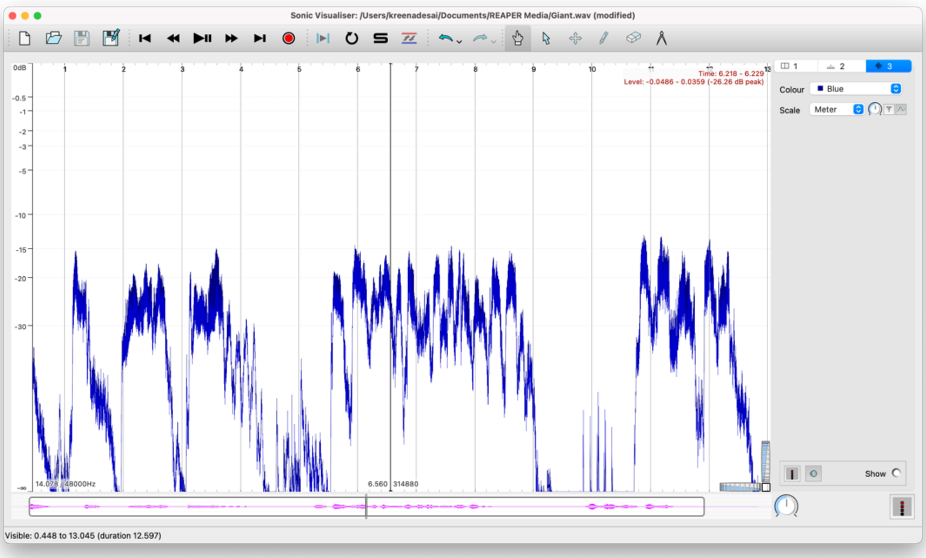
Character Voice Transformations:

Audio samples demonstrate the plugin's ability to transform voices:

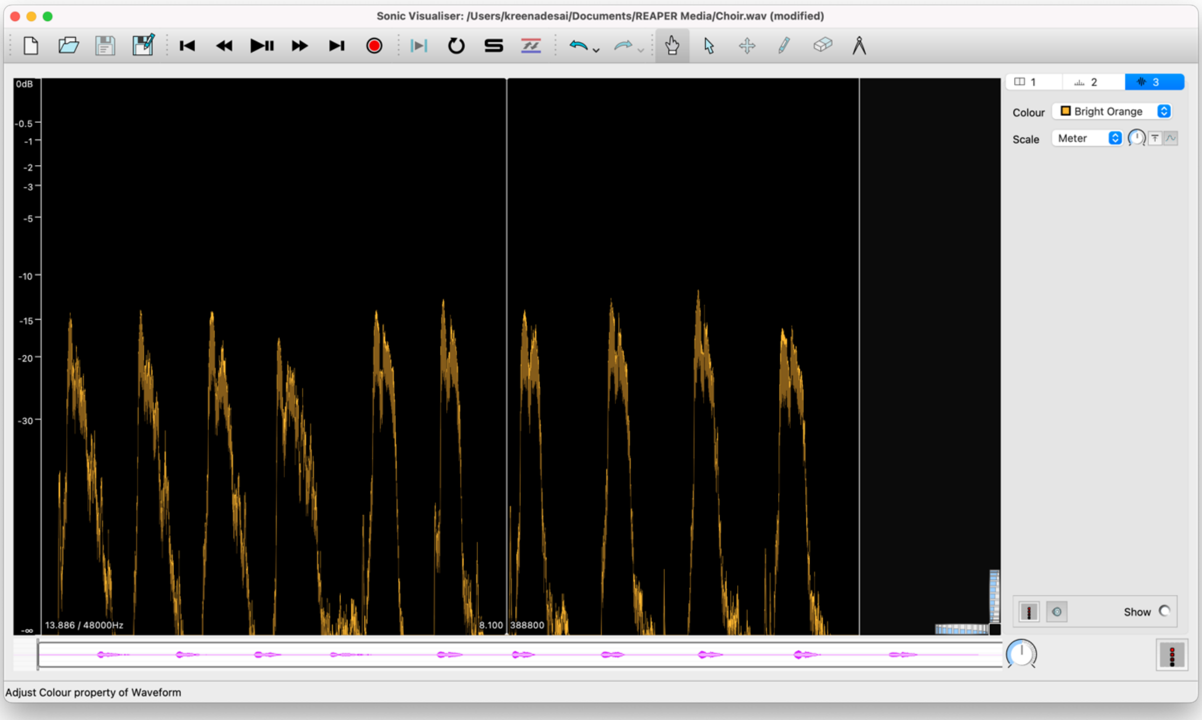
- **Sample: Robot transformation waveform**



- **Sample: Giant transformation waveform**



- **Sample: Choir transformation waveform**



CONCLUSION:

The Vocal Transformer plugin successfully demonstrates a real-time implementation of voice character transformation using a combination of digital signal processing techniques. The system effectively transforms vocal characteristics while maintaining real-time performance and providing an intuitive user interface.

Limitations:

- The pitch shifting and formant manipulation algorithms use simplified approaches that prioritize character over naturalness
- The voice multiplication technique could be improved with more sophisticated detuning and timing variations
- Some character transformations work better on certain voice types than others

Future Improvements:

- Implement more sophisticated pitch shifting
- Add a more accurate formant shifting algorithm with multiple formant band control

- Add voice analysis to adapt processing to input voice characteristics
- Include more character presets and user preset saving

In summary, the Vocal Transformer plugin provides an effective demonstration of voice transformation techniques in a real-time audio plugin framework, with clear potential for further refinement and expansion.

REFERENCES:

1. Röbel, A., & Rodet, X. (2005). "Efficient Spectral Envelope Estimation and its Application to Pitch Shifting and Envelope Preservation." International Conference on Digital Audio Effects.
2. Smith, J. O. (2011). "Spectral Audio Signal Processing." W3K Publishing.
3. Zölzer, U. (2011). "DAFX: Digital Audio Effects." John Wiley & Sons.
4. JUCE Framework Documentation. Retrieved from https://juce.com/learn/documentation