













```

class ResamplingBenchmarks : public juce::UnitTest
{
public:
    ResamplingBenchmarks()
        : juce::UnitTest ("Resampling Benchmarks", "tracktion_benchmarks")
    {
    }

    void runTest() override
    {
        runResamplingRendering ("lagrange",      ResamplingQuality::lagrange);
        runResamplingRendering ("sincFast",      ResamplingQuality::sincFast);
        runResamplingRendering ("sincMedium",    ResamplingQuality::sincMedium);
        runResamplingRendering ("sincBest",      ResamplingQuality::sincBest);
    }

private:
    //=====
    //=====
    void runResamplingRendering (juce::String qualityName,
                                ResamplingQuality quality)
    {
        auto& engine = *Engine::getEngines()[0];
        auto edit = Edit::createSingleTrackEdit (engine);
        edit->ensureNumberOfAudioTracks (1);
        auto t = getAudioTracks (*edit)[0];

        const auto durationOfFile = 30s;
        auto sinFile = getSinFile<juce::WavAudioFormat> (fileSampleRate, durationOfFile, 2, 220.0f);
        const auto timeRange = TimeRange (0s, TimePosition (durationOfFile));
        auto waveClip = t->insertWaveClip (sinFile->getFile().getFileName(), sinFile->getFile(),
                                           {{ timeRange }}, false);

        waveClip->setUsesProxy (false);
        waveClip->setResamplingQuality (quality);

        {
            ScopedBenchmark sb (createBenchmarkDescription ("Resampling", "WaveNode quality", "30s sin wave, 96KHz to 44.1KHz, " + qualityName.toStdString()));
            Renderer::measureStatistics ("Rendering resampling",
                                       *edit, timeRange,
                                       toBitSet ({ t }),
                                       256, playbackSampleRate);
        }
    }
};

```

```

static ResamplingBenchmarks resamplingBenchmarks;

```

Create a 30s sin clip



```

class ResamplingBenchmarks : public juce::UnitTest
{
public:
    ResamplingBenchmarks()
        : juce::UnitTest ("Resampling Benchmarks", "tracktion_benchmarks")
    {
    }

    void runTest() override
    {
        runResamplingRendering ("lagrange",    ResamplingQuality::lagrange);
        runResamplingRendering ("sincFast",    ResamplingQuality::sincFast);
        runResamplingRendering ("sincMedium",  ResamplingQuality::sincMedium);
        runResamplingRendering ("sincBest",    ResamplingQuality::sincBest);
    }

private:
    //=====
    //=====
    void runResamplingRendering (juce::String qualityName,
                                ResamplingQuality quality)
    {

```

Create a 30s sin clip

```

        waveClip->setUsesProxy (false);
        waveClip->setResamplingQuality (quality);

        {
            ScopedBenchmark sb (createBenchmarkDescription ("Resampling", "WaveNode quality", "30s sin wave, 96KHz to 44.1Khz, " + qualityName.toStdString()));
            Renderer::measureStatistics ("Rendering resampling",
                                        *edit, timeRange,
                                        toBitSet ({ t }),
                                        256, playbackSampleRate);
        }
    }
};

static ResamplingBenchmarks resamplingBenchmarks;

```

# Tracktion Benchmarks

Categories: 

All

 Name: WaveNode quality Description:

All

 Platform: Linux

Start Date: 

dd/mm/yyyy

 End Date: 

dd/mm/yyyy

 Time 

Cycles

Show total

Show min/max

Show variance

Normalise results

Update Results

