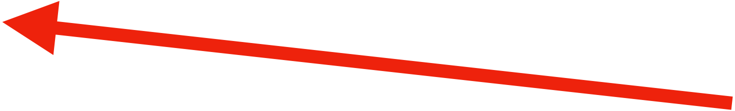
Double Buffering

Add a new bit "BIT_BUSY"



`BIT_BUSY` is set when realtime thread is in the middle of load/ store of `idx` variable

CAS loop to ensure that idx is only incremented when `BIT_BUSY` is not set

```cpp
using FrequencySpectrum = std::array<float, 512>;

enum { BIT_IDX = (1 << 0), BIT_NEWDATA = (1 << 1), BIT_BUSY = (1 << 2)};

std::array<FrequencySpectrum,2> mostRecentSpectrum;
std::atomic<int> idx = {0};

void processAudio (const float* buffer, size_t n) {
    auto freqSpec = calculateSpectrum (buffer, n);

    auto i = idx.fetch_or(BIT_BUSY) & BIT_IDX;
    mostRecentSpectrum[i] = freqSpec;
    idx.store ((i & BIT_IDX) | BIT_NEWDATA);
}

void updateSpectrumUIButtonClicked() {
    auto current = idx.load();

    if ((current & BIT_NEWDATA) != 0) {
        int newValue;
        do {
            current &= ~BIT_BUSY;
            newValue = (current ^ BIT_IDX) & BIT_IDX;
        } while (! idx.compare_exchange_weak (current, newValue));

        current = newValue;
    }

    displaySpectrum(mostRecentSpectrum[(current & BIT_IDX) ^ 1]);
}
```

79

Double Buffering

```
using FrequencySpectrum = std::array<float, 512>;

enum { BIT_IDX = (1 << 0), BIT_NEWDATA = (1 << 1), BIT_BUSY = (1 << 2)};

std::array<FrequencySpectrum,2> mostRecentSpectrum;
std::atomic<int> idx = {0};

void processAudio (const float* buffer, size_t n) {
    auto freqSpec = calculateSpectrum (buffer, n);

    auto i = idx.fetch_or(BIT_BUSY) & BIT_IDX;
    mostRecentSpectrum[i] = freqSpec;
    idx.store ((i & BIT_IDX) | BIT_NEWDATA);
}

void updateSpectrumUIButtonClicked() {
    auto current = idx.load();

    if ((current & BIT_NEWDATA) != 0) {
        int newValue;
        do {
            current &= ~BIT_BUSY;
            newValue = (current ^ BIT_IDX) & BIT_IDX;
        } while (! idx.compare_exchange_weak (current, newValue));

        current = newValue;
    }

    displaySpectrum(mostRecentSpectrum[(current & BIT_IDX) ^ 1]);
}
```

Add a new bit "BIT_BUSY"

BIT_BUSY is set when realtime thread is in the middle of load/ store of idx variable

CAS loop to ensure that idx is only incremented when BIT_BUSY is not set

farbot's RealtimeMutatable

```
using FrequencySpectrum = std::array<float, 512>;

RealtimeMutatable<FrequencySpectrum> mostRecentSpectrum;

void processAudio (const float* buffer, size_t n) {
    auto& freqSpec = mostRecentSpectrum.realtimeAcquire();

    freqSpec = calculateSpectrum (buffer, n);

    mostRecentSpectrum.realtimeRelease();
}

void updateSpectrumUIButtonClicked() {
    auto& recentSpectrum = mostRecentSpectrum.nonRealtimeAcquire();

    displaySpectrum(recentSpectrum);

    mostRecentSpectrum.nonRealtimeRelease();
}
```