TRƯỜNG ĐẠI HỌC XÂY DỰNG HÀ NỘI
KHOA CÔNG NGHỆ THÔNG TIN

LẬP TRÌNH ÂM THANH

CHƯƠNG 11

Ứng dụng Web Audio phần 1

Chương 11 · 5 tiết · Mã học phần 531833 · 75 tiết · Học kỳ 5

Hà Nội – 2026

Sau chương này người học phải làm được

- Dựng một đồ thị node hoàn chỉnh; chỉ ra mỗi khối tương ứng mục nào đã học.
- Chứng minh bằng số: node dựng sẵn và mô-đun tự viết cho cùng kết quả.
- Phân biệt ba đồng hồ trong trang web; biết đồng hồ nào dùng để đặt sự kiện phát tiếng.
- Giải thích vì sao node nguồn chỉ dùng một lần; viết đúng vòng đời của nó.
- Tính và đo độ trễ đường vòng, ngân sách tính toán trên máy của người học.
- Viết AudioWorkletProcessor và kiểm chéo với bản Python cùng thuật toán.
- Chọn đúng a-rate hay k-rate; đổi tham số không gây tiếng tách.
- Cài bộ lập lịch cửa sổ nhìn trước; gỡ bảy lỗi hay gặp bằng cách đo.

MỤC 11.1

Vị trí chương học trong học phần

- Từ chương 5 đến chương 10, chương trình tự quyết định khi nào gọi hàm xử lý.
- Từ chương này, trình duyệt gọi mã người học cứ 2,90 ms một lần.
- Mỗi lần gọi phải trả đúng 128 mẫu; trả muộn một lần thì người nghe thấy khoảng lặng.

11.2	Đồ thị node	Mô hình lập trình: nối khối, không viết vòng lặp.
11.3	Ngữ cảnh và đồng hồ	Một đồng hồ duy nhất, tiến từng khối 128 mẫu.
11.4	Nguồn phát	Dao động, bộ đệm, hằng số. Đều chỉ dùng một lần.
11.5	Thời gian thực	Trễ, ngân sách tính toán, và cái giá khi vượt.
11.6	AudioWorklet	Mang mã của chúng ta vào luồng âm thanh.
11.7	Lập lịch sự kiện	Đường bao tham số và bộ lập lịch cửa sổ nhìn trước.
11.8	Đo và gỡ lỗi	Analyser, và bảy lỗi hay gặp.

Bảy mục, và không mục nào thêm một phép xử lý mới. Cái mới nằm ở chỗ chạy KHI NÀO, và ai gọi.

Mọi con số trong chương này là số đo trên đồ thị node chạy thật.

Hai kiểu chạy, hai cách viết cùng một chuỗi xử lý

THEO LÔ — chương 5 đến 10

Đọc cả tệp, xử lý, ghi ra tệp. Chạy bao lâu cũng được. Sai thì sửa rồi chạy lại. Không ai chờ.

→

THỜI GIAN THỰC — chương 11, 12

Cứ 2,90 ms phải trả xong 128 mẫu. Trễ một lần là nghe thấy. Không có "chạy lại".

Cùng những phép xử lý ấy, nhưng bị đặt vào một ràng buộc thời gian không thương lượng được.

(a) Hai kiểu chạy

PYTHON — tự cầm vòng lặp

```
x = doc_wav("a.wav")
y = ap_biquad(x, b, a)
y = nen_dong(y)[0]
ghi_wav("b.wav", y)
```

→

WEB AUDIO — chỉ nối khối

```
src.connect(filter)
filter.connect(comp)
comp.connect(ctx.destination)
src.start(0)
```

Đoạn mã bên phải không có vòng lặp nào. Vòng lặp nằm trong trình duyệt, chạy ở một luồng khác, và gọi mã của chúng ta.

(b) Cùng một chuỗi xử lý, hai cách viết

Hình 11.2 — Hai kiểu chạy, và hai cách viết cùng một chuỗi xử lý

Chương này không thêm phép xử lý nào

GainNode, DelayNode, BiquadFilterNode, DynamicsCompressorNode, ConvolverNode, StereoPannerNode, PannerNode chính là danh sách những thứ đã tự viết từ chương 5 đến chương 10. Mục 11.2.4 đo lại: lệch trung bình 0,001 dB.

Mã viết bằng JavaScript, không còn Python

Web Audio là giao diện lập trình của trình duyệt, và trình duyệt chỉ chạy JavaScript. Ba khác biệt cần nhớ: câu lệnh kết thúc bằng dấu chấm phẩy, khối lệnh gói bằng ngoặc nhọn; mảng âm thanh là Float32Array 32 bit; luỹ thừa dùng Math.pow.

MỤC 11.2

Đồ thị node: mô hình lập trình của Web Audio

- Không viết vòng lặp, chỉ nối khối.
- Mỗi node dựng sẵn tương ứng một mục đã học.
- Nhiều nhánh nối vào cùng đích thì tự cộng, và đồ thị không tự bảo vệ.
- Phép kiểm trọng tâm: biquad của trình duyệt khớp công thức chương 9.

Cùng một chuỗi xử lý, viết bằng Web Audio

```
const ctx = new AudioContext();

const src = ctx.createBufferSource();
src.buffer = buf;                    // AudioBuffer đã nạp sẵn

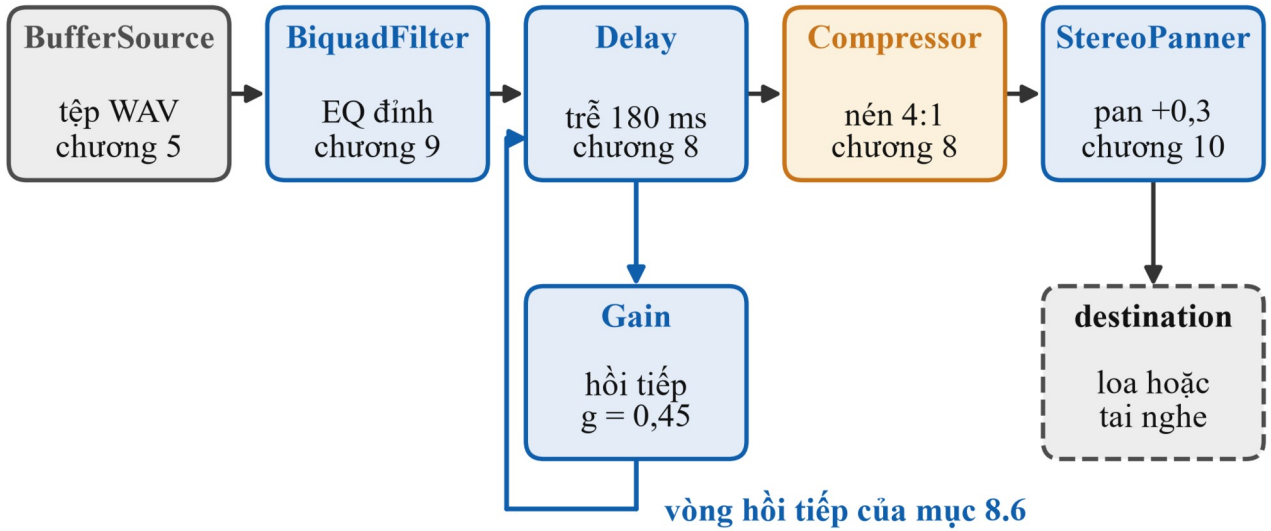
const eq = ctx.createBiquadFilter();
eq.type = 'peaking';
eq.frequency.value = 900;
eq.Q.value = 1.2;
eq.gain.value = -8;

const comp = ctx.createDynamicsCompressor();
comp.threshold.value = -18;
comp.ratio.value = 4;

src.connect(eq).connect(comp).connect(ctx.destination);
src.start(0);
```

Hàm connect trả về chính node đích, nên nối được thành chuỗi.

Một đồ thị node gồm sáu khối của cả học phần



ĐỌC ĐỒ THỊ NÀY RA MỘT CÂU

Sáu khối xử lý là sáu thứ đã học ở các chương 5 đến 10. Cả sáu có sẵn trong trình duyệt, viết bằng C++, chạy ngoài luồng JavaScript. Cái mới của chương này không phải khối, mà là cách nối và cách ra lệnh cho chúng.

Hình 11.3 — Một đồ thị node gồm sáu khối của cả học phần

Mỗi node dựng sẵn là một mục đã học

Node	Làm gì	Học ở (mục)	Còn thiếu gì
GainNode	Nhân một hệ số	5.4 — gain theo dB	Không thiếu gì
DelayNode	Trễ tới 180 giây	6.4, 8.5 — trễ, lọc lược	Không nội suy bậc cao
BiquadFilterNode	Bảy kiểu lọc bậc hai	9.3 — công thức RBJ	Không đặt thẳng được hệ số
IIRFilterNode	Lọc bất kỳ theo b và a	9.3 — sai phân	Hệ số cố định, không tự động hoá
DynamicsCompressorNode	Bộ nén động	8.3 — nén và giới hạn	Không có bộ giới hạn riêng
ConvolverNode	Tích chập đáp ứng xung	9.8 — vang tích chập	Tự chuẩn hoá, hay gây bất ngờ
StereoPannerNode	Pan hai kênh	10.4 — giữ công suất	Chỉ một luật, không chọn được
PannerNode	Đặt nguồn trong không gian	10.6, 10.7 — ITD, ILD	Bảng HRTF của trình duyệt
AnalyserNode	FFT theo khung	4.5 — phổ và cửa sổ	Chỉ để nhìn, không xử lý
AudioWorkletNode	Mã của chính mình	Mọi thứ viết bằng Python	Đây là chỗ bù mọi chỗ thiếu ở trên

Đọc cột thứ ba theo chiều dọc: đó gần như là mục lục của mười chương vừa qua.

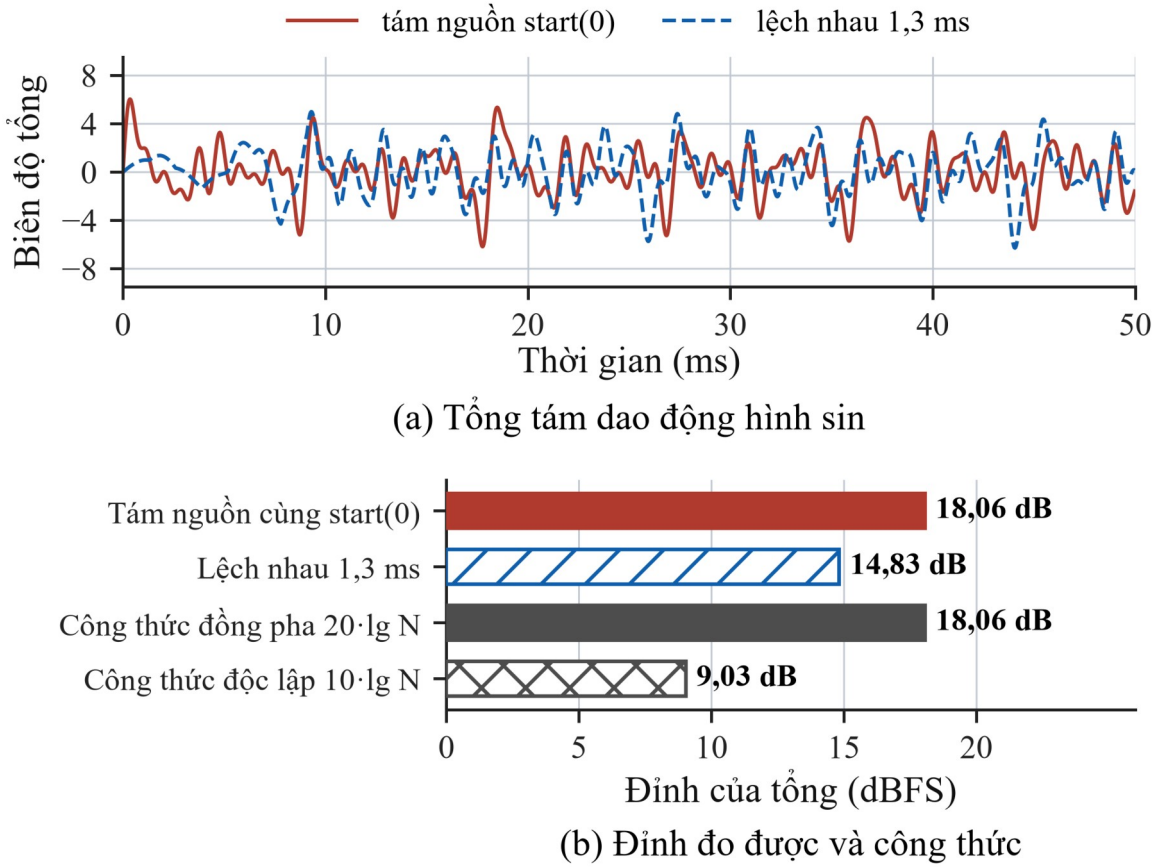
Hình 11.4 — Bảng đối chiếu node dựng sẵn với mục đã học

Chỗ node dựng sẵn thiếu so với bản tự viết

- BiquadFilterNode không cho đặt thẳng năm hệ số.
- IIRFilterNode không đổi được hệ số sau khi tạo, nên không làm được wah của mục 9.5.
- StereoPannerNode chỉ có luật giữ công suất của mục 10.4.3; luật thoả hiệp phải dựng bằng hai GainNode.
- Không có bộ giới hạn riêng: tỉ số tối đa 20, không có lookahead của 8.4, không bảo đảm được trần.
- Đó là lý do cần AudioWorklet ở mục 11.6.

Tám nguồn cùng gọi start(0)

- Tám sin ở tám tần số, mỗi sin nối vào một GainNode hệ số 1.
- Trực giác: nguồn khác tần số là độc lập, đỉnh tăng 9,03 dB.
- Đo thật: +18,06 dBFS (decibel so với mức toàn thang, Full Scale), đúng công thức đồng pha của mục 10.2.3.
- Trực giác sai trong cả hai trường hợp đã đo.



Hình 11.5 — Tám nguồn cùng gọi start(0), và con số ấy khớp công thức nào

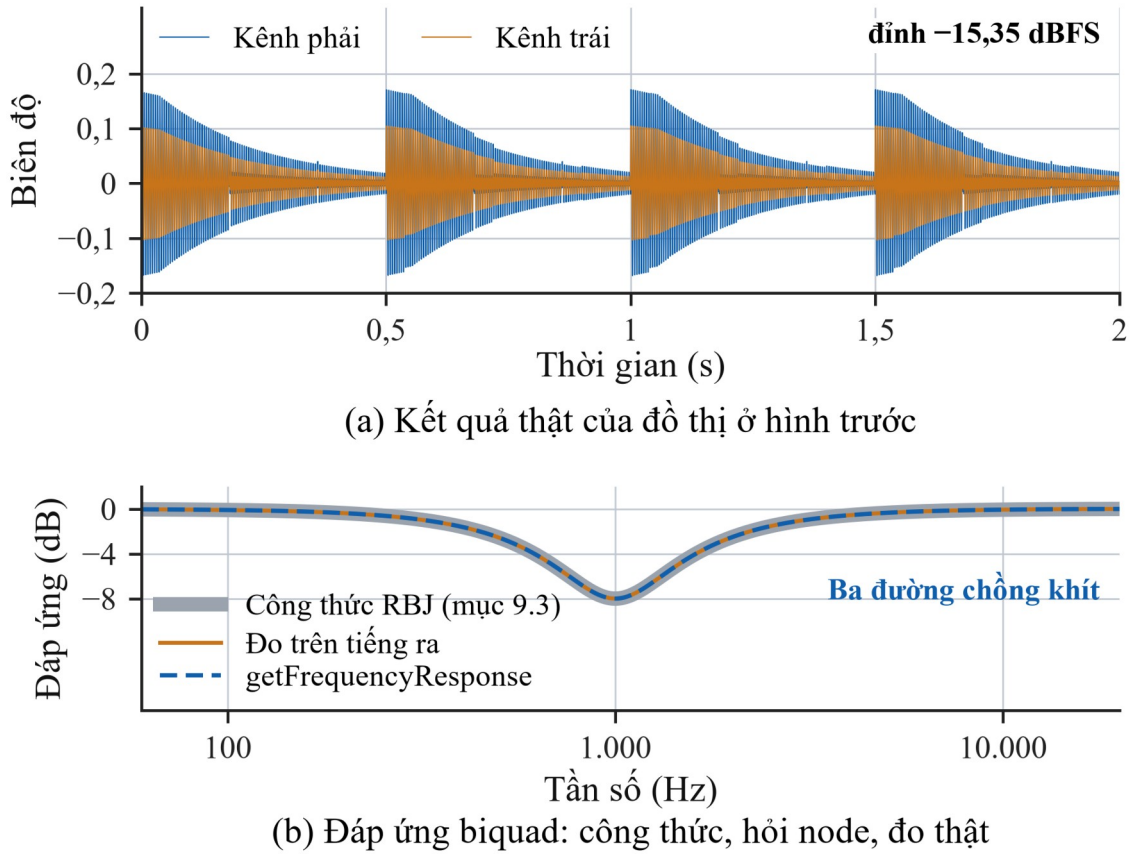
Đo lại trên đồ thị chạy thật

Cách làm	Đỉnh đo được	Khớp công thức nào
Tám nguồn cùng gọi start(0)	+18,06 dBFS	Đồng pha 10.2.3: 20·log10(8)
Tám nguồn khởi động lệch 1,3 ms	+14,83 dBFS	Giữa hai công thức, vẫn cao hơn độc lập nhiều
Công thức độc lập (để so sánh)	+9,03 dBFS	Con số trực giác đưa ra; sai trong cả hai trường hợp

Tám tần số khác nhau vẫn đồng pha

Tất cả bắt đầu cùng thời điểm với pha ban đầu bằng 0, nên đỉnh chồng lên nhau. Hệ số tương quan trên đoạn dài gần 0, nhưng đỉnh chỉ cần một mẫu mà tám sóng cùng lớn. Cách chắc chắn: chừa headroom theo công thức đồng pha.

Ba đường đáp ứng của biquad chồng khít nhau



Hình 11.6 — Kết quả thật của đồ thị, và ba đường đáp ứng chồng khít nhau

Ba nguồn số và độ khớp

Nguồn số	Lấy ở đâu	Lệch so với công thức
Công thức RBJ	biquad_he_so và dap_ung_biquad_db, mục 9.3	Mốc để so
getFrequencyResponse	Hỏi thẳng node của trình duyệt	0,0000 dB
Đo trên tiếng ra	Nhiễu trắng qua node, lấy hiệu hai phổ	Trung bình 0,001 dB; lớn nhất 0,055 dB

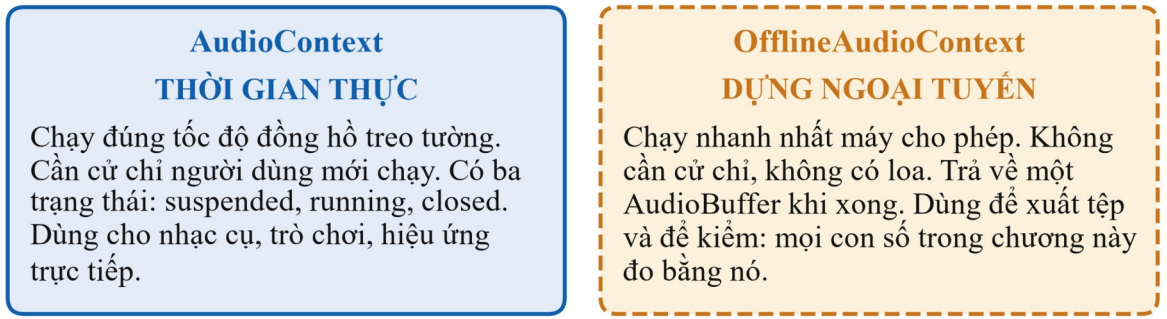
Mô-đun tự viết và node dựng sẵn là cùng một thứ.

MỤC 11.3

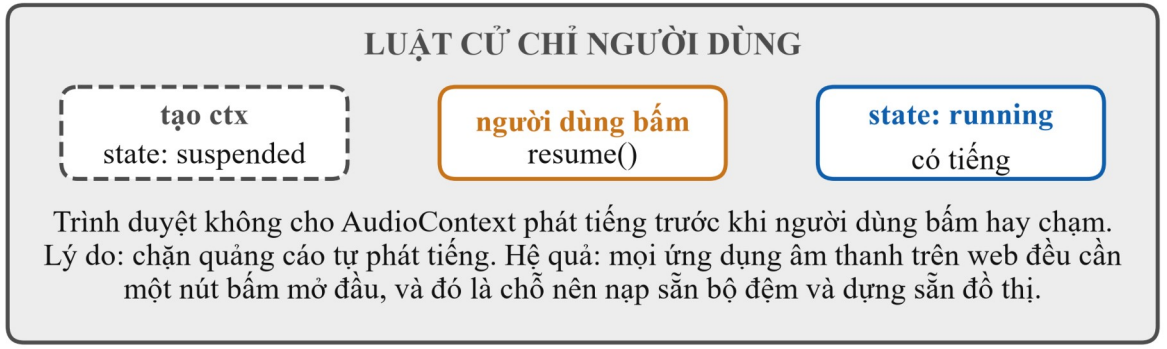
Ngữ cảnh, đồng hồ và vòng đời node

- Hai loại ngữ cảnh: thời gian thực và ngoại tuyến.
- Đồng hồ ngữ cảnh nhảy từng khối nhưng lịch chính xác tới mẫu.
- Node nguồn chỉ phát một lần: dựng mới cho mỗi tiếng.

Hai loại ngữ cảnh và luật cử chỉ người dùng



(a) Hai loại ngữ cảnh



(b) Luật cử chỉ người dùng

Hình 11.7 — Hai loại ngữ cảnh và luật cử chỉ người dùng

Vì sao phải có một nút bấm mở đầu

- AudioContext được tạo ở trạng thái suspended cho tới khi người dùng bấm hoặc chạm.
- Lý do: chặn quảng cáo tự phát tiếng.
- Nút bấm là chỗ tốt nhất để nạp bộ đệm, nạp module worklet, dựng sẵn đồ thị.
- Quên resume thì không có tiếng và không có lỗi; in ctx.state ra là biết.

Thời lượng một khối

$$t_q = \frac{N_q}{f_s} = \frac{128}{44100} \approx 2{,}90 \text{ ms}$$

(11.1)

- Nq là kích thước khối dựng của Web Audio, cố định 128 mẫu trong chuẩn.
- fs là tần số lấy mẫu của ngữ cảnh.
- Ở 44,1 kHz, mỗi khối kéo dài 2,90 ms.
- Hàm xử lý bị gọi 344,53 lần mỗi giây.

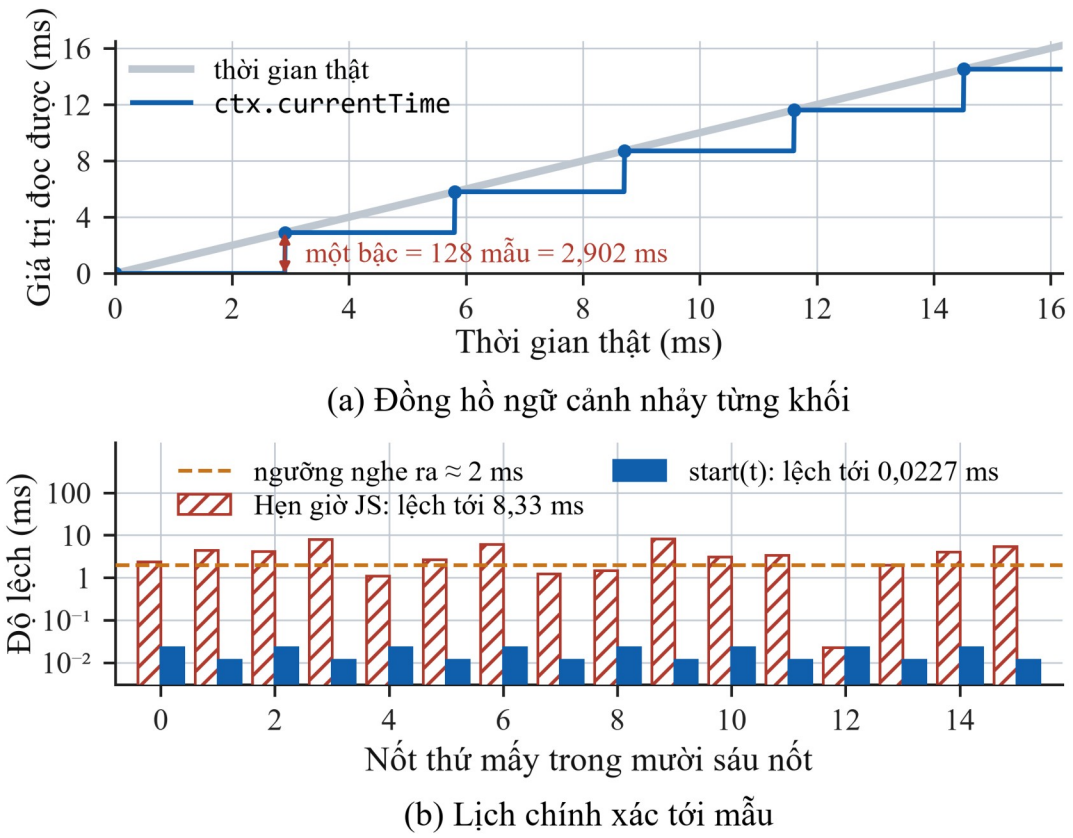
Đồng hồ nhảy từng khối

$$t_k = k\frac{N_q}{f_s}, \qquad k = 0, 1, 2, \ldots$$

(11.2)

- ctx.currentTime tính bằng giây kể từ khi ngữ cảnh được tạo.
- Đây là đồng hồ duy nhất dùng để đặt thời điểm phát tiếng.
- Giá trị đọc từ JavaScript chỉ đổi mỗi 128 mẫu.

Đồng hồ nhảy từng khối, lịch chính xác tới mẫu



Hình 11.8 — Đồng hồ nhảy từng khối, nhưng lịch thì chính xác tới mẫu

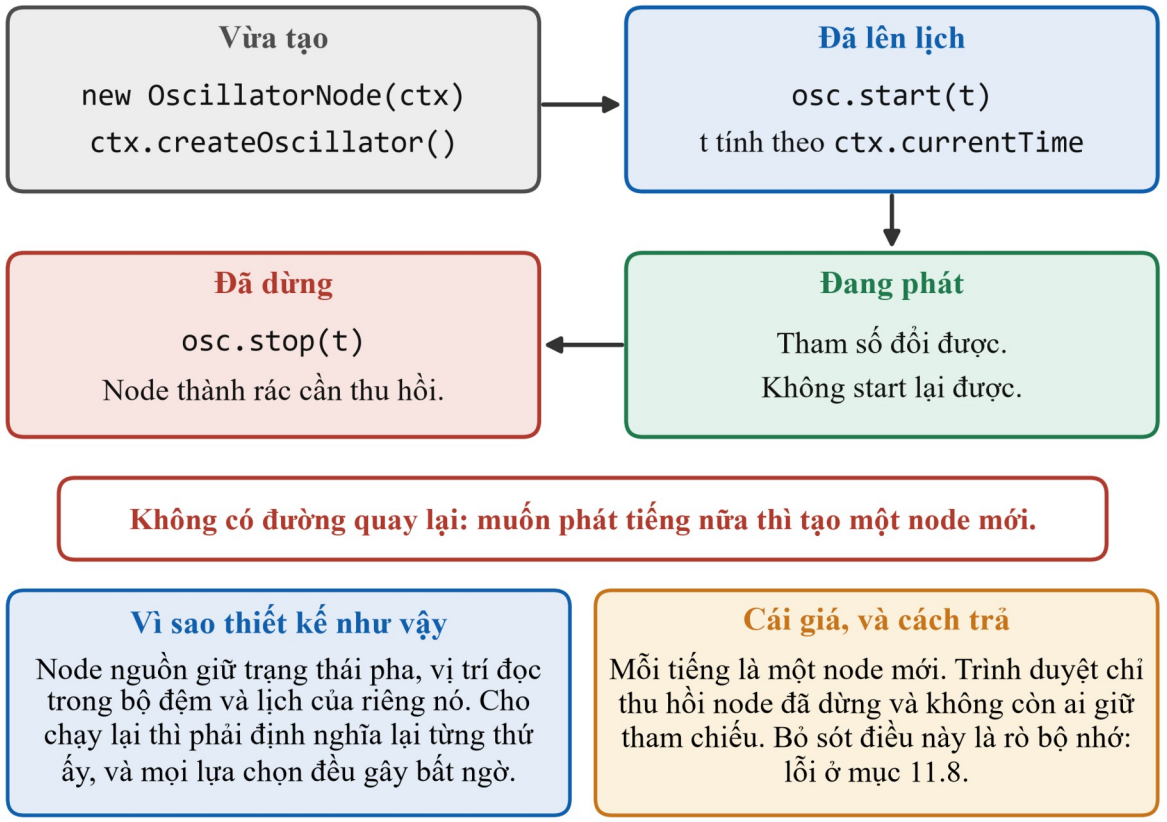
Đọc currentTime, nhưng đặt lịch bằng start(t)

- Đo trên đầu ra worklet: currentTime đổi mỗi bước 2,902 ms, không dùng để đặt nốt vào đúng nhịp.
- Thời điểm truyền cho start(t) được thực hiện chính xác tới mẫu.
- Mười sáu nốt móc kép ở 120 nhịp/phút lệch nhiều nhất 0,0227 ms, đúng một mẫu ở 44,1 kHz.
- Cách đúng: đọc currentTime, cộng một khoảng, truyền tổng cho start.

Ba đồng hồ trong một trang web

Đồng hồ	Nguồn	Dùng để làm gì
ctx.currentTime	Thạch anh của card âm thanh	Mọi thời điểm phát tiếng; đồng hồ thật
performance.now()	Bộ đếm của hệ điều hành	Đo hiệu năng, vẽ giao diện; trôi so với đồng hồ trên
setTimeout, setInterval	Hàng đợi sự kiện của trình duyệt	Chỉ đánh thức bộ lập lịch; sai vài ms, dừng khi tab ở nền

Bốn trạng thái của một node nguồn



Hình 11.9 — Bốn trạng thái của một node nguồn

Cách đúng: mỗi tiếng là một node mới

```
function danhTrong(t) {
  const s = ctx.createBufferSource();
  s.buffer = mauTrong;              // dùng lại BỘ ĐỆM, không dùng lại node
  const g = ctx.createGain();
  g.gain.setValueAtTime(0.8, t);
  s.connect(g).connect(bus);
  s.start(t);
  s.onended = () => { s.disconnect(); g.disconnect(); };
}
```

Dùng lại bộ đệm, không dùng lại node.

AudioBuffer và AudioBufferSourceNode

- AudioBuffer là dữ liệu: mảng mẫu nạp một lần, dùng lại cho hàng nghìn tiếng.
- AudioBufferSourceNode là con trỏ đọc kèm lịch riêng: nhẹ, tạo mới mỗi tiếng, bỏ sau khi dùng.
- Nạp lại bộ đệm mỗi tiếng thì tốn bộ nhớ và thời gian.
- Dùng lại node thì báo lỗi ở tiếng thứ hai.
- Giữ mọi node trong một mảng gây rò bộ nhớ; phải disconnect khi onended.

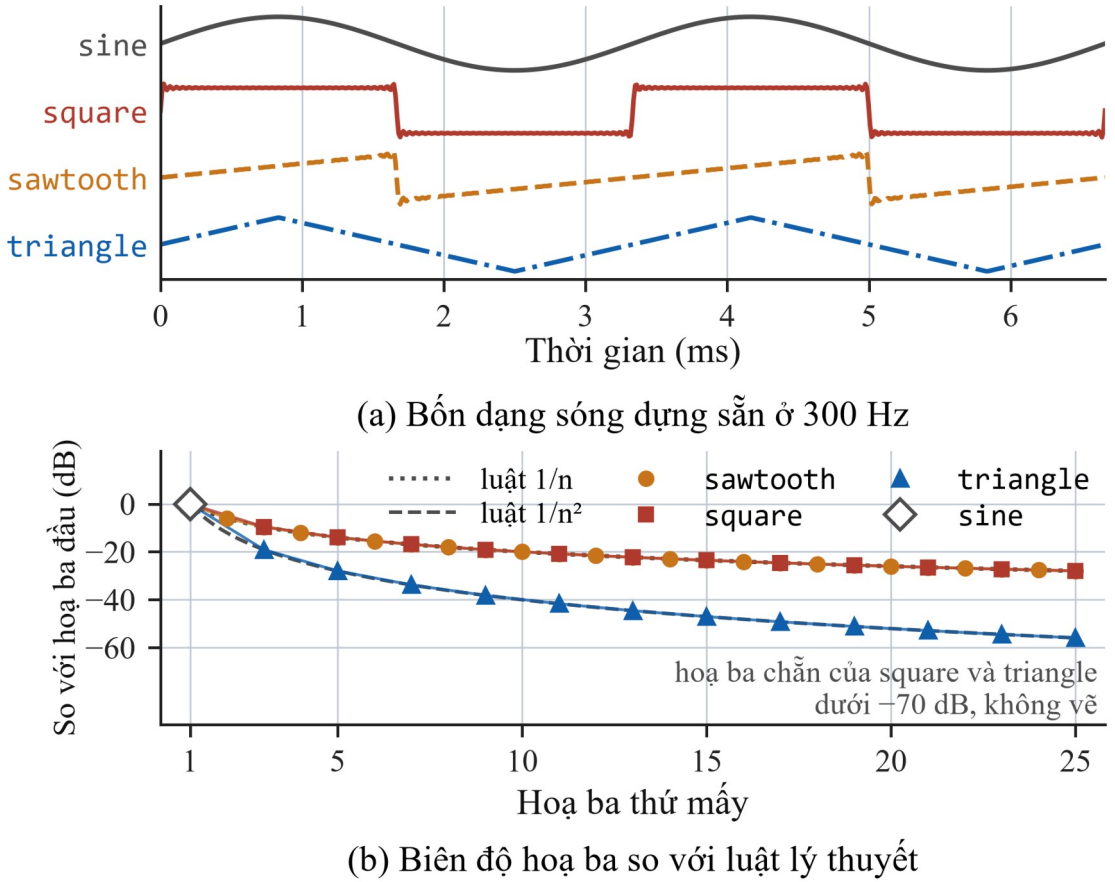
MỤC 11.4

Nguồn phát

- Ba loại nguồn: dao động, bộ đệm, hằng số.
- Dao động của trình duyệt là băng giới hạn, ít chồng phổ hơn bản tự tính.
- playbackRate và detune là hai núm của cùng một tỉ lệ đọc.

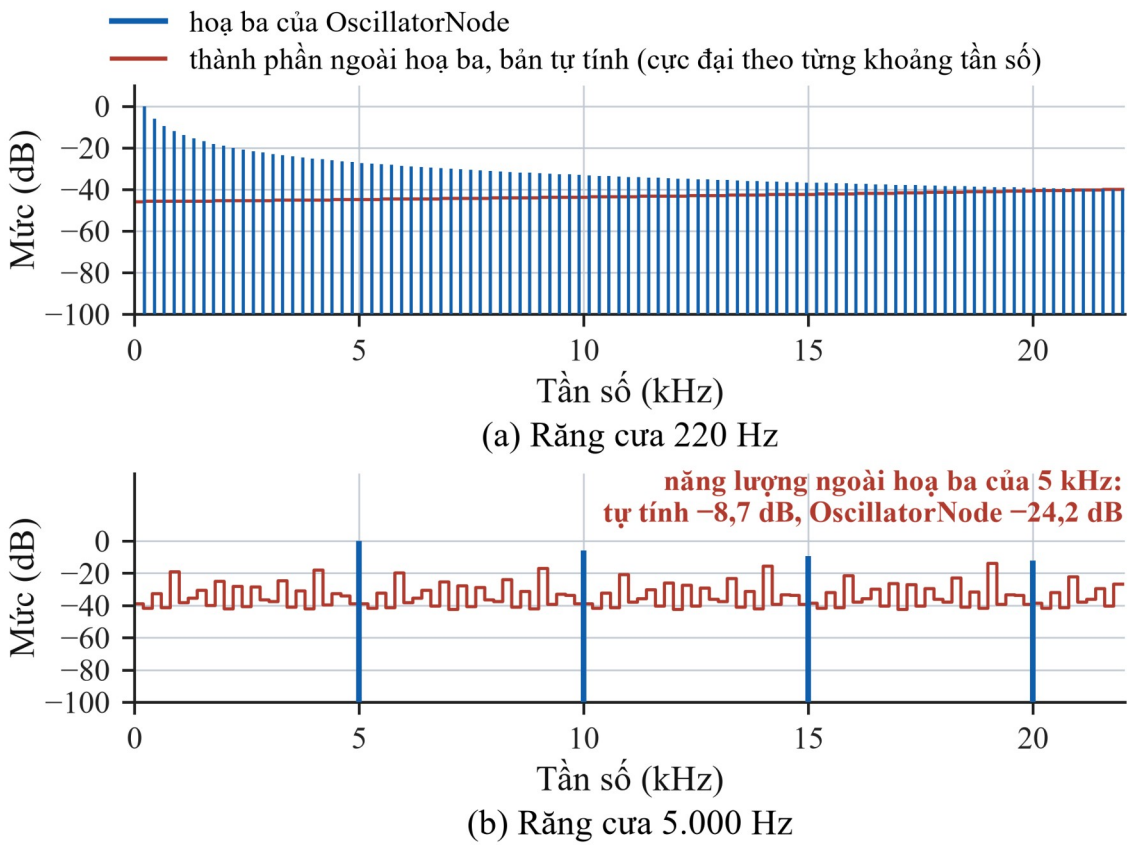
Bốn dạng sóng dựng sẵn

- Hình dạng giống bốn hàm đã viết ở mục 2.5.
- Biên độ hoạ ba bám đúng lý thuyết: răng cưa theo 1/n, sóng vuông bỏ hoạ ba chẵn, tam giác theo $1/n^2$.
- Nhưng được dựng từ bảng sóng đã cắt mọi hoạ ba vượt Nyquist, rồi nội suy.
- Khác biệt chỉ lộ ở tần số cao.



Hình 11.10 — Bốn dạng sóng dựng sẵn, và biên độ các hoạ ba của chúng

Răng cưa 220 Hz và 5 kHz: tự tính so với OscillatorNode



Hình 11.11 — Răng cưa 220 Hz và 5 kHz: tự tính so với OscillatorNode

Năng lượng nằm ngoài mọi hoạ ba của tần số đang chơi

Cách sinh răng cưa 5 kHz	Năng lượng lạc	Tính ra phần trăm
Tự tính bằng công thức (mục 2.5)	−8,7 dB	Khoảng 13,5 % tổng năng lượng
OscillatorNode của trình duyệt	−24,2 dB	Khoảng 0,38 %, nhỏ hơn ba mươi lần

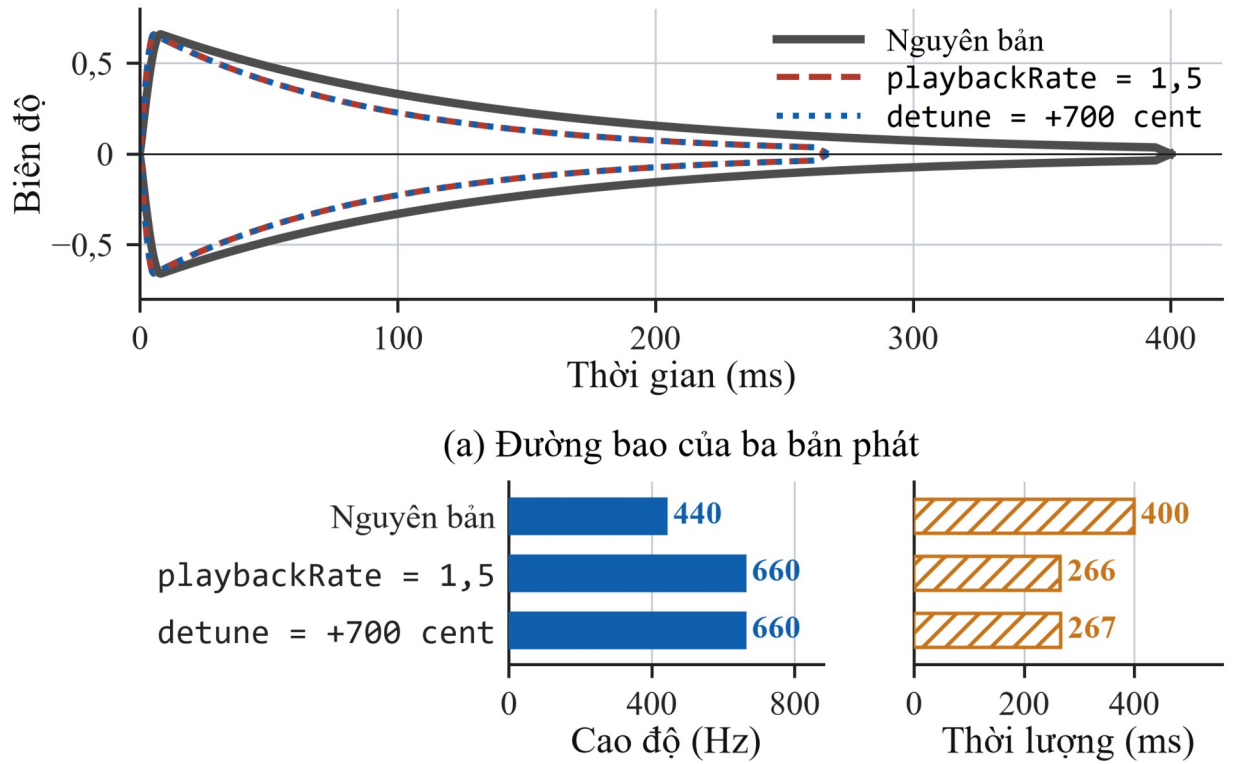
Hai núm, một đại lượng

$$r = \text{playbackRate} \times 2^{d/1200}\ ; \qquad T_{\text{ra}} = T_{\text{gốc}} / r$$

(11.3)

- Cả hai tác động vào cùng một tỉ lệ tốc độ đọc r.
- playbackRate tính theo tỉ lệ thẳng; detune tính theo cent (mục 2.6).
- Hai núm nhân với nhau chứ không cộng.

Cùng một bộ đệm, ba cách phát, hai đại lượng đo được



Hình 11.12 — Cùng một bộ đệm, ba cách phát, và hai đại lượng đo được

Đo lại trên ba bản phát cùng một bộ đệm

Cách phát	Tỉ lệ r	Cao độ đo được	Thời lượng đo được
Nguyên bản	1,000	440 Hz	400 ms
playbackRate = 1,5	1,500	660 Hz	266 ms
detune = +700 cent	1,498	660 Hz	267 ms

700 cent là một quãng năm đúng, cho tỉ lệ 1,498, gần bằng 1,5.

Không tách được cao độ khỏi thời lượng

BufferSourceNode chỉ đọc bộ đệm nhanh hay chậm hơn: đúng phép đổi tần số lấy mẫu của mục 3.7, đổi cả hai thứ cùng lúc. Đổi tiếng bước chân bằng detune là cách rẻ, nhưng chỉ trong khoảng hẹp chừng cộng trừ 200 cent.

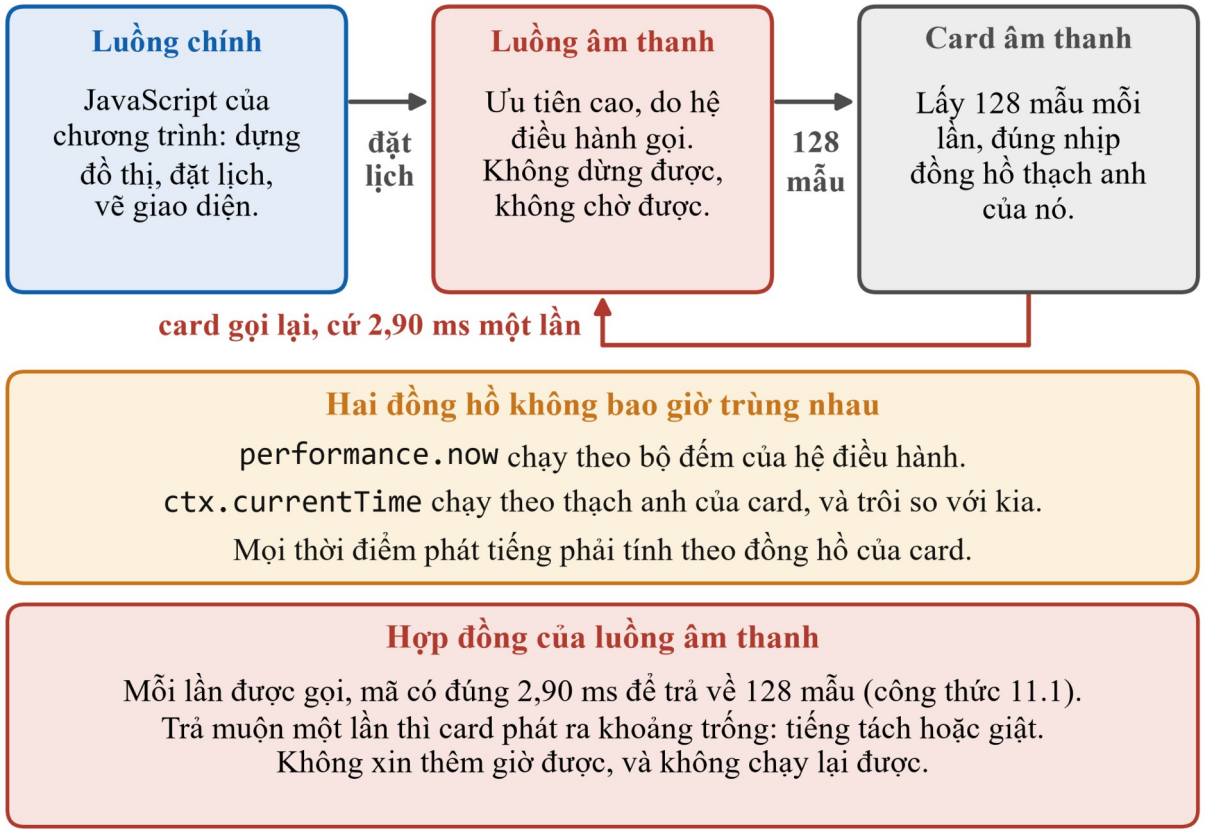
MỤC 11.5

Xử lý thời gian thực: trễ và ngân sách

- Ba bên trong một vòng lặp: card âm thanh, luồng âm thanh, luồng chính.
- Độ trễ đường vòng cộng từ ba chỗ trễ.
- Ngân sách tính toán cho biết chạy nổi bao nhiêu giọng.
- Vượt ngân sách để lại dấu vết đo được trong phổ.

Ba bên trong một vòng lặp thời gian thực

- Card âm thanh chạy theo thạch anh riêng và không chờ ai.
- Mỗi khối, card đòi đủ số mẫu tiếp theo.
- Chương trình JavaScript nằm ở luồng chính, ngoài vòng lặp này.
- Hai đường giao tiếp duy nhất: AudioParam và cổng tin nhắn.



Hình 11.13 — Ba bên trong một vòng lặp thời gian thực

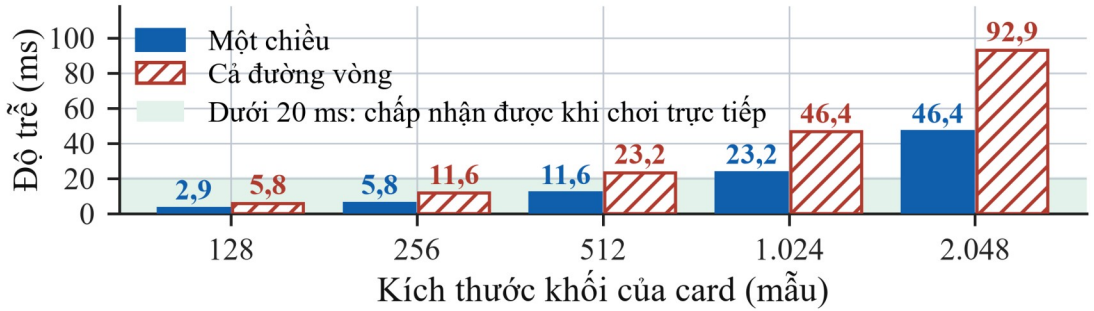
Độ trễ đường vòng

$$t_{\text{vòng}} = \frac{L_{\text{vào}} + L_{\text{ra}}}{f_s} + t_{\text{thiếtbị}}$$

(11.4)

- L_vào và L_ra là kích thước khối của card ở hai chiều.
- t_thiết bị gồm bộ chuyển đổi, đường truyền không dây, đệm của hệ điều hành.
- Khối 128 mẫu của Web Audio không xuất hiện: nhỏ hơn hẳn và bị hai số kia che khuất.

Độ trễ theo kích thước khối, và ba chỗ trễ



(a) Độ trễ theo kích thước khối

Khối vào
Từ micro tới trình duyệt: bằng kích thước khối của card, không đổi được từ JavaScript.

+

Xử lý
128 mẫu mỗi lần gọi, cố định trong chuẩn Web Audio: 2,90 ms.

+

Khối ra
Từ trình duyệt tới loa: bằng kích thước khối của card, cộng đệm của hệ điều hành.

(b) Ba chỗ trễ cộng lại, theo công thức (11.4)

`ctx.baseLatency` và `ctx.outputLatency` báo hai số đầu: nên độ trễ đo được chứ không phải đoán.

Hình 11.14 — Độ trễ theo kích thước khối, và ba chỗ trễ

Độ trễ theo kích thước khối của card, ở 44,1 kHz

Khối	Một chiều	Cả đường vòng	Nhận xét
128 mẫu	2,9 ms	5,8 ms	Lý tưởng để chơi đàn trực tiếp; tốn CPU nhất
256 mẫu	5,8 ms	11,6 ms	Vẫn dưới ngưỡng nhận ra
512 mẫu	11,6 ms	23,2 ms	Bắt đầu nhận ra khi gõ phím; dùng được cho trò chơi
1.024 mẫu	23,2 ms	46,4 ms	Chỉ hợp cho phát lại
2.048 mẫu	46,4 ms	92,9 ms	Không dùng cho tương tác

Tải trọng ρ

$$\rho = \frac{t_{\text{xử lý}}}{t_q}\ ; \qquad \rho < 1$$

(11.5)

- ρ là tỉ số giữa thời gian thật sự dùng và thời gian có được cho một khối.
- ρ vượt quá 1: trả kết quả muộn, card phát ra khoảng trống.

Số giọng chạy nổi

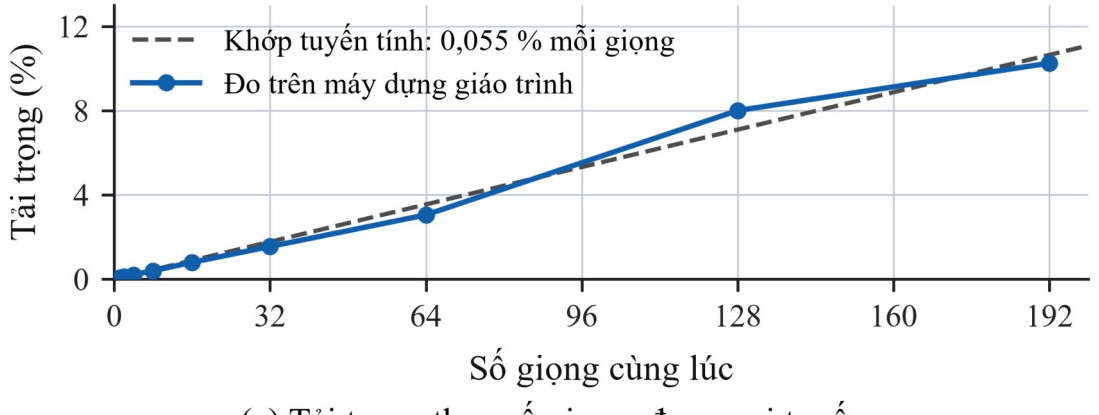
$$N_{\max} = \left\lfloor \frac{\rho_{\max} t_q}{t_{\text{giọng}}} \right\rfloor$$

(11.6)

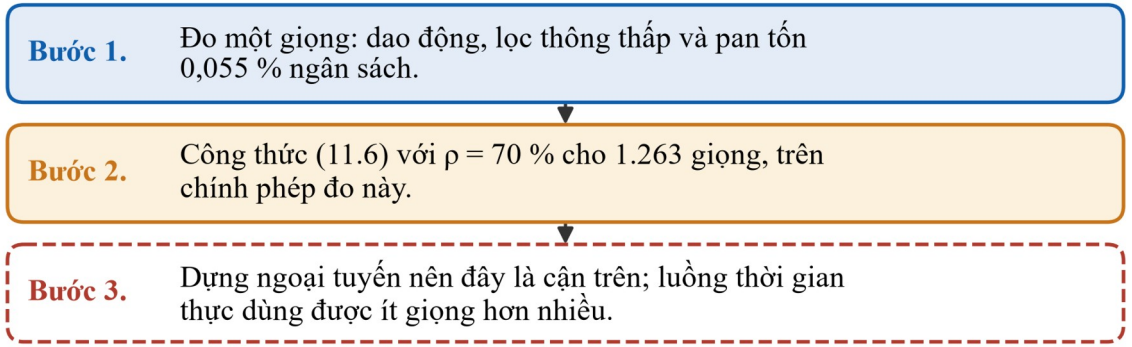
- Suy ra từ tải trọng của một giọng và mức tải được phép.
- Cách làm: đo một giọng trên máy đích, chia cho ngân sách, rồi chừa 30 %.
- Đúng cách chừa headroom của mục 10.2, chỉ khác đơn vị.

Đường cong gần như thẳng qua gốc

- Điều đáng tin là hình dạng đường cong, không phải con số cuối.
- Mỗi giọng thêm vào tốn đúng chừng ấy, nên dự đoán được.
- Phép đo dựng ngoại tuyến trên máy dựng giáo trình, nên con số là cận trên.
- Trong trình duyệt thật, số giọng dùng được thấp hơn nhiều.



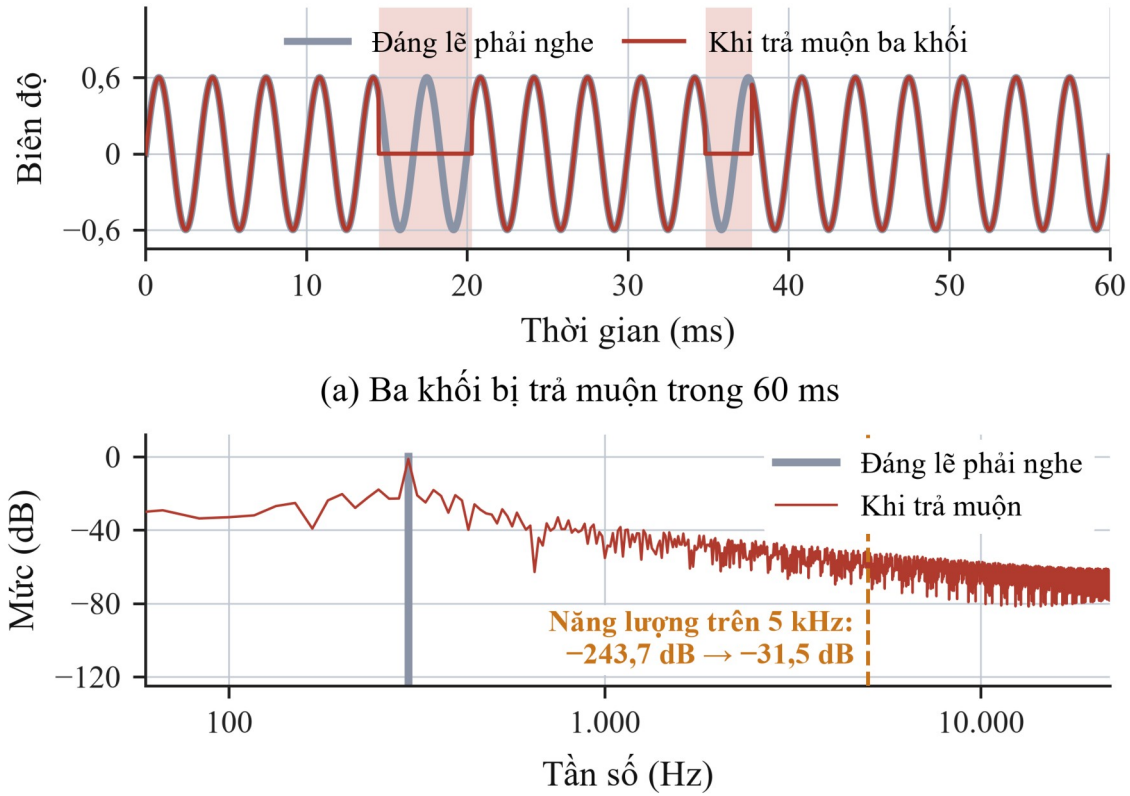
(a) Tải trọng theo số giọng, đo ngoại tuyến



(b) Cách suy ra số giọng chạy nổi

Hình 11.15 — Tải trọng theo số giọng, và cách suy ra số giọng chạy nổi

Ba khối bị trả muộn, và dấu vết trong phổ



Hình 11.16 — Ba khối bị trả muộn, và dấu vết của chúng trong phổ

Sóng sin 300 Hz mất ba khối trong 60 ms

Đại lượng	Bình thường	Khi mất ba khối
Phần năng lượng trên 5 kHz	−243,7 dB	−31,5 dB
Nghe ra sao	Một nốt sạch	Một nốt kèm ba tiếng tách

Mép khoảng trống là bước nhảy đột ngột, đúng thứ gây tiếng tách ở mục 6.6.

Khi tải trọng quá cao

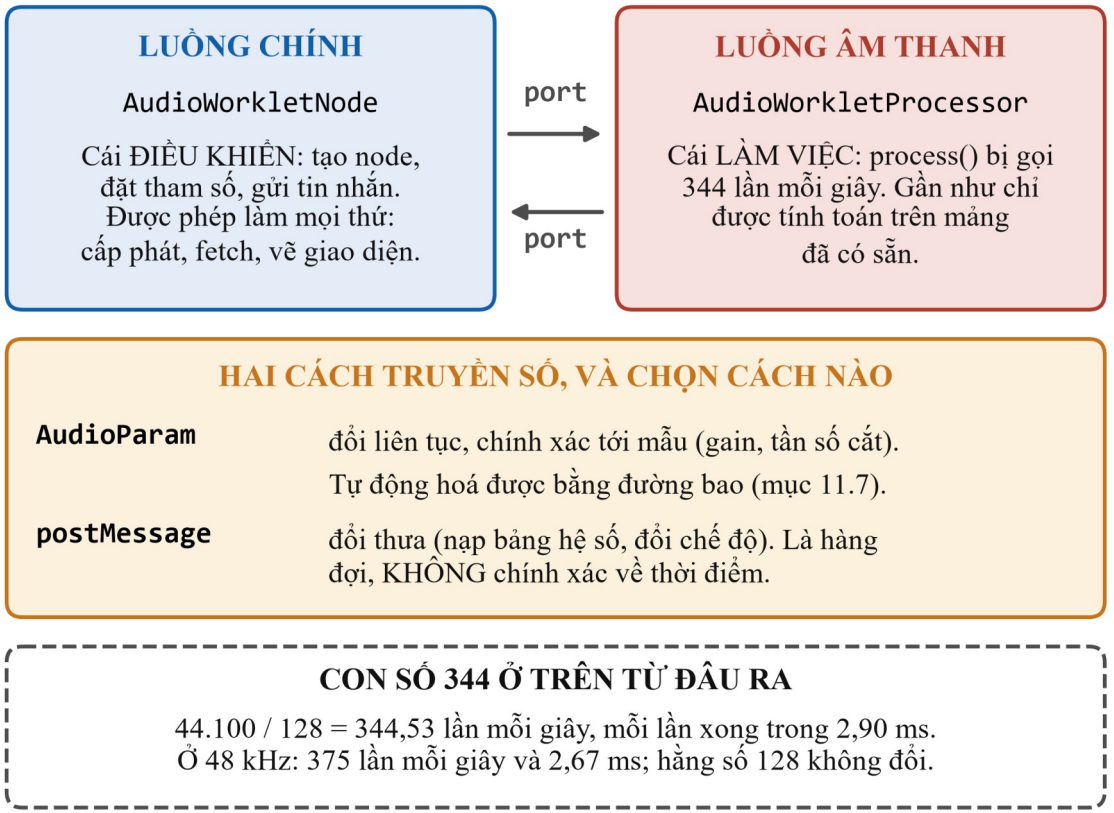
- Giảm số giọng: dừng giọng đã tắt gần hết, gộp các giọng giống nhau.
- Giảm chất lượng: hạ bậc bộ lọc, bỏ bớt tầng vang, đổi ConvolverNode sang bộ vang thuật toán của mục 9.7.
- Tăng kích thước khối của card: rẻ nhất, nhưng độ trễ tăng gấp đôi mỗi lần.
- Đây là đánh đổi trung tâm của cả mô-đun.

MỤC 11.6

AudioWorklet: mang mã riêng vào

- Một worklet gồm hai nửa ở hai luồng khác nhau.
- Kiểm chéo: bộ nén chương 8 chạy trong trình duyệt, lệch dưới 152 dB.
- Tham số a-rate và k-rate; bảy điều cấm trong process.

Hai luồng, và hai cách truyền số qua ranh giới



Hình 11.17 — Hai luồng, và hai cách truyền số qua ranh giới

Hàm process của luồng âm thanh

$$y[n] = f(x[n],\ \mathbf{p}[n]),\qquad n = 0, \ldots, 127$$

(11.7)

- AudioWorkletNode ở luồng chính điều khiển; AudioWorkletProcessor ở luồng âm thanh làm việc.
- process nhận một khối 128 mẫu cùng giá trị tham số và phải ghi đủ 128 mẫu ra.
- Bị gọi 344,53 lần mỗi giây ở 44,1 kHz, mỗi lần có 2,90 ms.

Khung của một AudioWorkletProcessor

```
// tệp riêng, ví dụ wk-nen.js
class Nen extends AudioWorkletProcessor {
  static get parameterDescriptors() {
    return [
      { name: 'nguong', defaultValue: -18,
        automationRate: 'k-rate' },
      { name: 'tiso', defaultValue: 4, automationRate: 'k-rate' },
    ];
  }
  constructor() { super(); this.z = 0; }   // trạng thái giữa các khối
  process(inputs, outputs, params) {
    const a = inputs[0], b = outputs[0];
    if (!a || !a[0]) { for (const ch of b) ch.fill(0); return true; }
    // ... vòng lặp 128 mẫu ở đây ...
    return true;                            // còn sống thì trả true
  }
}
registerProcessor('nen', Nen);
```

Ba điều dễ sai ở khung worklet

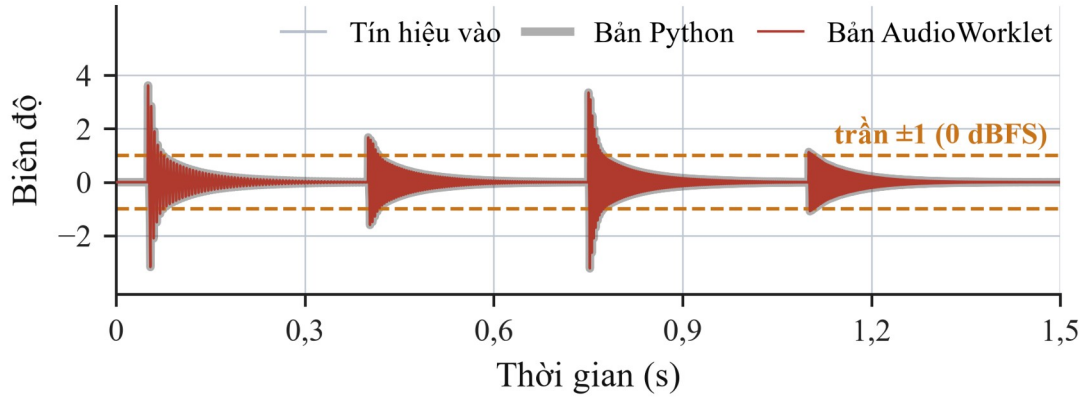
- addModule là bất đồng bộ; quên await thì lỗi "processor chưa đăng ký".
- Tệp worklet chạy trong phạm vi riêng: không thấy biến toàn cục, không có window hay document.
- Trả về false là xin bị huỷ, node im vĩnh viễn.
- Chỉ trả false khi chắc chắn không bao giờ phát tiếng nữa.

Bộ nén của mục 8.3, viết bằng JavaScript

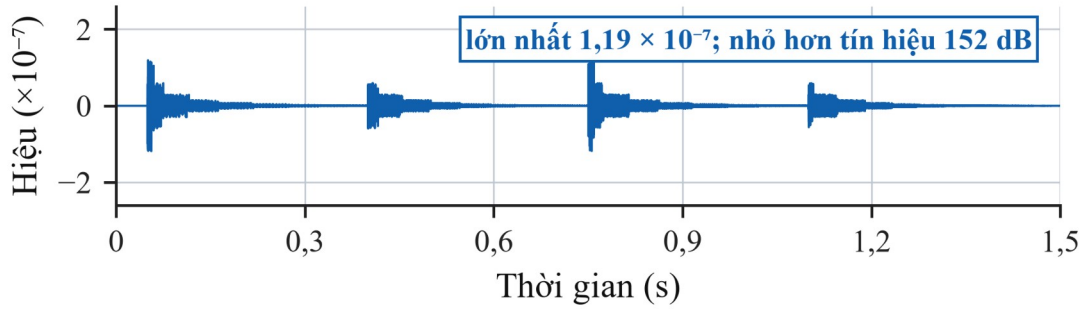
```
process(i, o, p) {
  const a = i[0], b = o[0];
  const T = p.nguong[0], R = p.tiso[0];
  const at = Math.exp(-1 / (p.tancong_ms[0] / 1000 * sampleRate));
  const re = Math.exp(-1 / (p.nha_ms[0] / 1000 * sampleRate));
  const bu = -(T * (1 - 1 / R));          // gain bù tự động
  for (let n = 0; n < b[0].length; n++) {
    const L = 20 * Math.log10(Math.max(Math.abs(a[0][n]), 1e-9));
    const gr = L > T ? (T + (L - T) / R) - L : 0;
    const k = gr < this.z ? at : re;       // đi xuống là tấn công
    this.z = k * this.z + (1 - k) * gr;
    const g = Math.pow(10, (this.z + bu) / 20);
    for (let c = 0; c < b.length; c++) b[c][n] = a[c][n] * g;
  }
  return true;
}
```

Cùng cách dò mức, cùng đường đặc tuyến tĩnh, cùng cách làm trơn trên gain.

Bản Python và bản AudioWorklet, và hiệu của chúng



(a) Hai bản cài đặt vẽ chồng lên nhau



(b) Hiệu của hai bản

Hình 11.18 — Bản Python và bản AudioWorklet, và hiệu của chúng

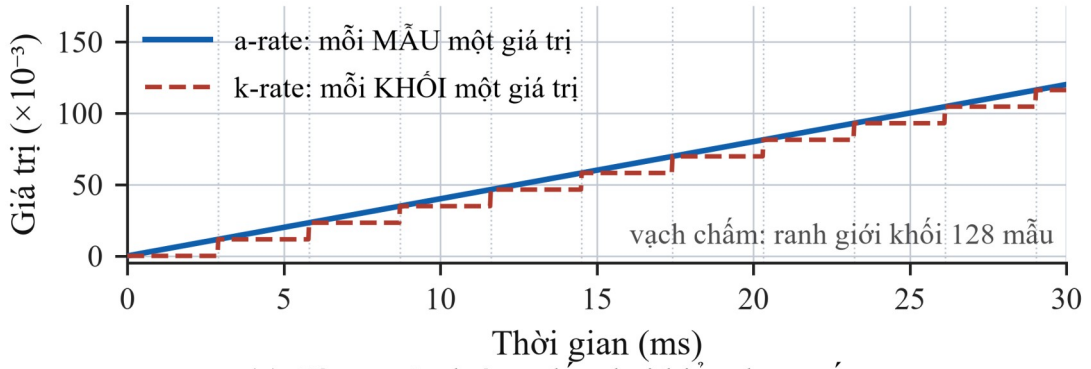
Kết quả phép kiểm chéo

Đại lượng	Giá trị
Lệch lớn nhất giữa hai bản	$1{,}19 \times 10^{-7}$
So với mức tín hiệu	Nhỏ hơn 152 dB
Kết luận	Bằng sai số làm tròn của số thực 32 bit: hai bản là một

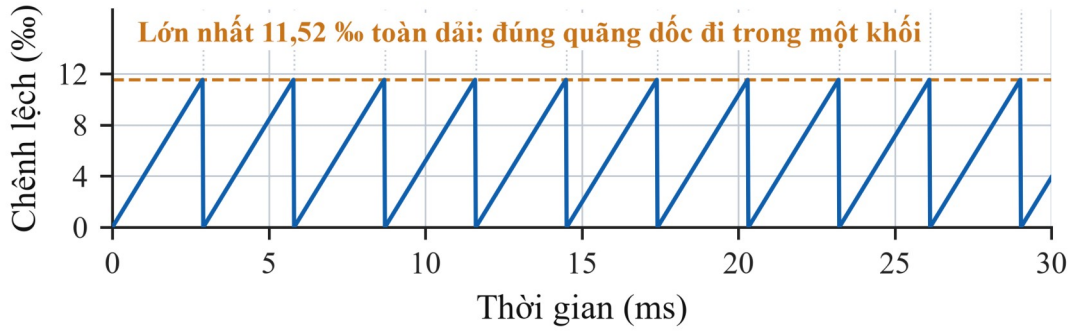
Không phải học lại xử lý tín hiệu

Mọi thuật toán đã viết bằng Python đều chạy được trong trình duyệt. Chỉ phải học cách đưa vào: chia theo khối 128 mẫu, giữ trạng thái trong đối tượng, không làm gì bị cấm trong luồng âm thanh. Gain bù tự động đẩy đỉnh vượt 0 dBFS, nên vẫn phải nối bộ giới hạn của mục 8.4 phía sau.

Cùng một đường dốc, hai kiểu tham số



(a) Cùng một đường dốc, hai kiểu tham số



(b) Chênh lệch k-rate so với a-rate

Hình 11.19 — Cùng một đường dốc, hai kiểu tham số, và chênh lệch giữa chúng

Hai chế độ tham số

Chế độ	Trả về mỗi khối	Chọn cho	Cái giá
a-rate	128 giá trị, mỗi mẫu một giá trị	Gain, tần số cắt; thứ tai nghe ra đứt quãng	Tốn hơn; vòng lặp phải đọc mảng
k-rate	Một giá trị cho cả khối	Ngưỡng, tỉ số, chế độ; thứ đổi thưa	Chậm hơn a-rate một khối, 2,90 ms

Chênh lệch lớn nhất giữa hai chế độ: 11,52 phần nghìn toàn dải, bằng quãng đường dốc đi trong 128 mẫu.

Đọc tham số cho đúng cả hai chế độ

```
const h = params.hesoGain;          // có thể dài 1 hoặc 128
for (let n = 0; n < b[0].length; n++) {
  const g = h.length > 1 ? h[n] : h[0];    // MẪU CHUẨN, luôn viết thế này
  b[0][n] = a[0][n] * g;
}
```

Mảng có thể dài 1 ngay cả với a-rate, khi giá trị không đổi trong khối; không kiểm độ dài sẽ chạy sai trong im lặng.

Bảy điều cấm trong luồng âm thanh

ĐỪNG LÀM	VÌ SAO	LÀM THAY BẰNG
Cấp phát mảng mới	Có thể kích bộ dọn rác, và nó dừng luồng	Cấp sẵn trong constructor, tái dùng
`console.log`	Mỗi dòng là một lần vượt ranh giới luồng	Gom số liệu, gửi về bằng port
Vòng lặp không chặn trên	Một khối quá 2,90 ms là ĐỨT TIẾNG	Chia việc ra nhiều khối, giữ trạng thái
`await, Promise, fetch`	Không chờ được; hàm phải trả về ngay	Nạp trước ở luồng chính, gửi vào bằng port
Đọc ngày giờ của hệ thống	Đó không phải đồng hồ âm thanh (mục 11.3)	Dùng biến currentTime trong worklet
Trả về false quá sớm	Trả false là xin bị huỷ; node im vĩnh viễn	Trả true chừng nào còn phát ra tiếng
Chia cho một số có thể bằng 0	NaN lan ra cả đồ thị rồi im hẳn	Kẹp mẫu số, như 1e-9 ở mục 7.2

Sáu dòng đầu là vấn đề hiệu năng; dòng cuối là vấn đề đúng sai, và là lỗi khó tìm nhất trong bảy dòng.

Hình 11.20 — Bảy điều cấm trong luồng âm thanh

Vì sao một số NaN là lỗi khó tìm nhất

Một phép chia cho 0 hoặc loga của số âm là ra NaN. NaN lan qua mọi node phía sau trong một khối, cả đồ thị im hẳn mà không có thông báo lỗi. Phòng: kẹp mẫu số và đối số loga bằng 1e-9. Tìm: dựng lại bằng OfflineAudioContext và soát mảng kết quả.

MỤC 11.7

Lập lịch sự kiện và đường bao tham số

- Mọi AudioParam là một đường theo thời gian: ba kiểu đường.
- Đổi tham số không kêu tách: đường mũ êm hơn đường thẳng.
- Lịch theo nhịp và bộ lập lịch cửa sổ nhìn trước.

linearRampToValueAtTime

$$v(t) = v_0 + (v_1 - v_0)\frac{t - t_0}{t_1 - t_0}$$

(11.8)

- Đặt thẳng vào thuộc tính value là cách thô sơ nhất, và gây tiếng tách.
- Cách đúng là mô tả cả một đường: giá trị này, tại thời điểm kia, đi theo kiểu nào.
- Đường thẳng đi đều từ điểm này tới điểm kia: fade tuyến tính của mục 6.6.

exponentialRampToValueAtTime

$$v(t) = v_0\left(\frac{v_1}{v_0}\right)^u, \qquad u = \frac{t - t_0}{t_1 - t_0}, \qquad v_0 v_1 \neq 0$$

(11.9)

- Đi theo tỉ lệ chứ không theo hiệu, nên nghe đều trong tai (mục 1.5).
- Cả hai đầu phải khác 0: điều kiện cuối của công thức là bắt buộc.

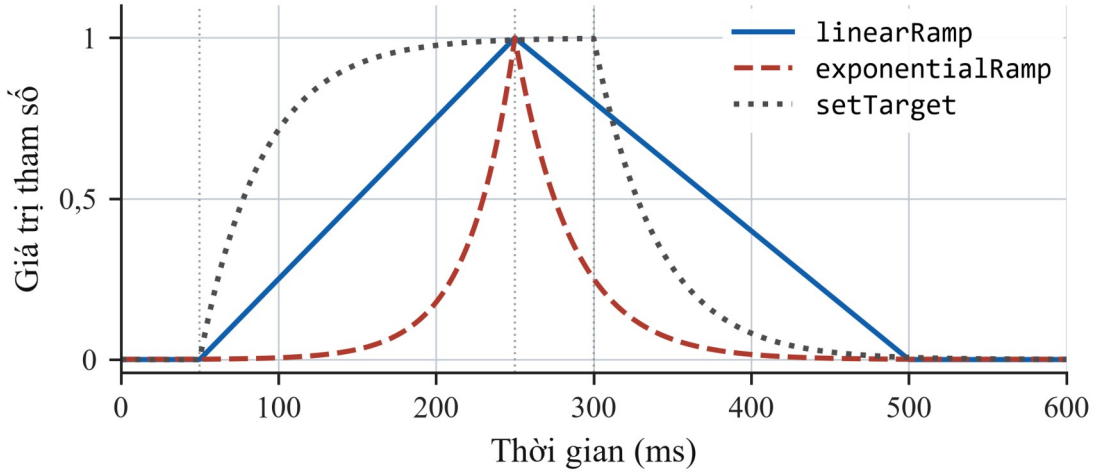
setTargetAtTime

$$v(t) = v_1 + (v_0 - v_1)\, e^{-(t - t_0)/\tau}$$

(11.10)

- Không có đích cuối, chỉ có một hằng số thời gian.
- Đây là bộ làm trơn một cực của mục 8.2, hiện ra dưới dạng một lời gọi hàm.

Ba kiểu đường bao đo trên đầu ra thật, và cách chọn



(a) Ba kiểu đường bao tính theo công thức (11.8) đến (11.10)

`linearRamp` (11.8): đi thẳng; dùng cho pan và đại lượng tuyến tính.

`exponentialRamp` (11.9): đi theo tỉ lệ nên nghe đều; không qua 0.

`setTarget` (11.10): tiến dần về đích; sau 3τ mới đạt 95 %.

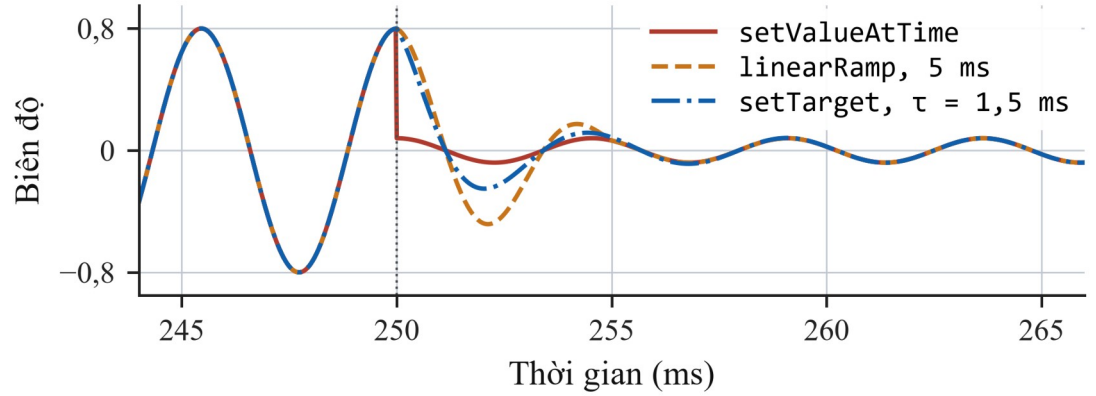
(b) Cách chọn kiểu đường bao

Hình 11.21 — Ba kiểu đường bao đo trên đầu ra thật, và cách chọn

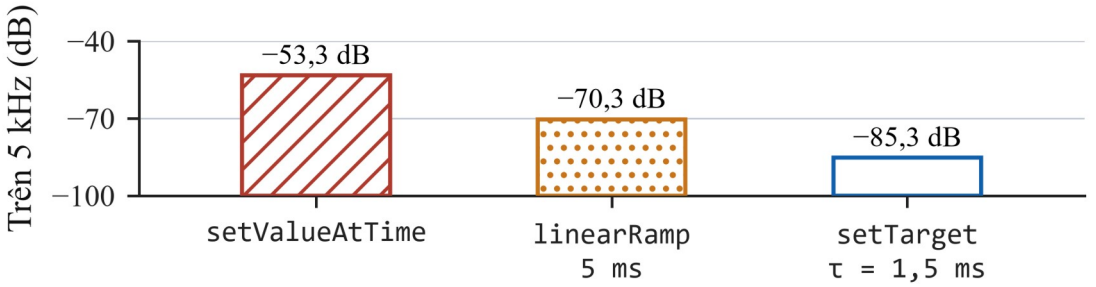
setTargetAtTime không bao giờ chạm tới đích

Sau một hằng số thời gian đi được 63 % quãng đường, sau ba hằng số thì 95 %. Dùng nó để tắt hẳn thì tiếng không bao giờ tắt và node nguồn không bao giờ tự dừng. Cách đúng: dùng cho chỗ chỉ cần về gần, rồi đặt thẳng về 0 sau ba tới năm hằng số thời gian.

Hạ gain bằng ba cách khác nhau, và tiếng tách đo được



(a) Hạ gain từ 0,8 xuống 0,08 tại mốc 250 ms (sóng sin 220 Hz)



(b) Năng lượng trên 5 kHz: càng thấp càng ít tiếng tách

Hình 11.22 — Hạ gain bằng ba cách khác nhau, và tiếng tách đo được

Ba cách hạ gain từ 0,8 xuống 0,08 tại mốc 250 ms

Cách gọi	Năng lượng trên 5 kHz	Nhận xét
setValueAtTime, đặt thẳng	−53,3 dB	Bước nhảy đúng ở mẫu ấy: tiếng tách
linearRampToValueAtTime, 5 ms	−70,3 dB	Tốt hơn 17 dB; còn hai chỗ gãy
setTargetAtTime, τ = 1,5 ms	−85,3 dB	Tốt hơn 32 dB dù chuyển ngắn hơn; không có chỗ gãy

Êm hay không nằm ở đạo hàm, không ở độ dài

Đường thẳng có hai chỗ đạo hàm nhảy bậc; đường mũ có đạo hàm liên tục ở mọi điểm. Vì vậy τ = 1,5 ms êm hơn hẳn đường thẳng 5 ms. Cùng nguyên lý với fade hình cosin êm hơn fade tuyến tính cùng độ dài ở mục 6.6.

Ba cách gọi, và điểm dễ sai của đường mũ

```
// LUÔN đặt một mốc trước khi vẽ đường, nếu không đường sẽ bắt đầu
// từ sự kiện gần nhất TRƯỚC đó — có thể là tận thời điểm 0
g.gain.setValueAtTime(g.gain.value, t);

g.gain.linearRampToValueAtTime(0.08, t + 0.005);      // cách 1
g.gain.setTargetAtTime(0.08, t, 0.0015);              // cách 2

// Cách 3 — đường mũ — KHÔNG đi về 0 được:
// g.gain.exponentialRampToValueAtTime(0, t + 0.005);  →  LỖI
g.gain.exponentialRampToValueAtTime(0.0001, t + 0.005);
g.gain.setValueAtTime(0, t + 0.006);                  // rồi tắt hẳn
```

Quên đặt mốc thì đường dốc bắt đầu từ sự kiện gần nhất trước đó, có thể là thời điểm 0.

Khoảng cách giữa các nốt

$$t_k = t_0 + k\frac{60}{\mathrm{BPM}\cdot d}$$

(11.11)

- BPM (Beats Per Minute – số nhịp mỗi phút) là số nhịp mỗi phút; d là số nốt trong một nhịp: 1 là nốt đen, 2 là móc đơn, 4 là móc kép.
- Ở 120 nhịp/phút, nốt móc kép cách nhau 125 ms.
- Mọi thời điểm phải tính theo ctx.currentTime, không theo đồng hồ JavaScript.
- Hẹn giờ của JavaScript lệch tới 8,33 ms; start(t) chính xác tới một mẫu.

Điều kiện chọn cửa sổ nhìn trước

$$T_{\text{nhìn}} > T_{\text{gọi}} + J_{\max} \tag{11.12}$$

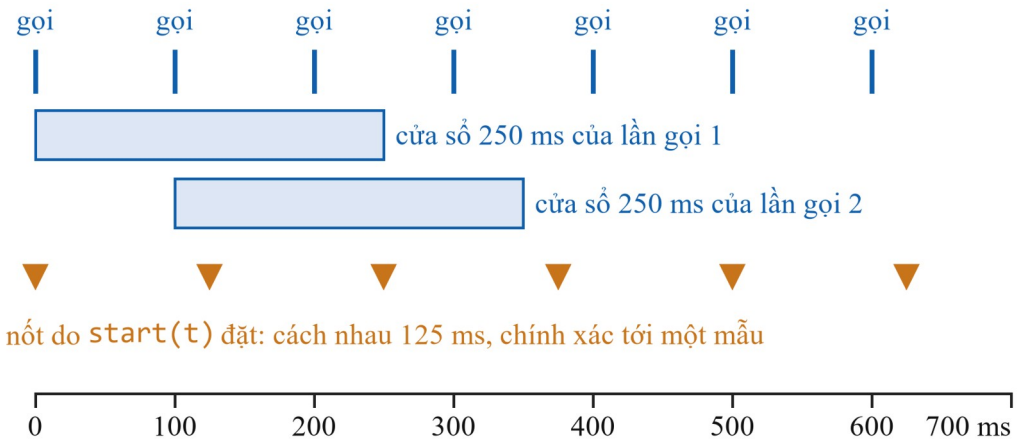
- Mâu thuẫn: nhạc chạy vô hạn thì không đặt lịch một lúc được, mà hẹn giờ JavaScript lại không chính xác.
- Lời giải: hẹn giờ làm việc thô, việc đặt đúng thời điểm giao cho đồng hồ âm thanh.
- Cửa sổ phải rộng hơn chu kỳ gọi cộng độ trễ lớn nhất của hẹn giờ.

Bộ lập lịch cửa sổ nhìn trước

```
let tKe = 0;                    // thời điểm nốt kế tiếp
let stt = 0;                    // số thứ tự nốt
const NHIN = 0.25;              // cửa sổ nhìn trước, giây
const GOI = 100;                // chu kỳ gọi, mili giây

function datLich() {
  while (tKe < ctx.currentTime + NHIN) {
    danhTrong(tKe, stt % 4 === 0);      // tạo node mới, start(tKe)
    tKe += 60 / bpm / 4;                // công thức (11.11)
    stt += 1;
  }
}
setInterval(datLich, GOI);
```

Bộ lập lịch cửa sổ nhìn trước và kết quả đo được



(a) Hẹn giờ gọi mỗi 100 ms; mỗi lần đặt lịch các nốt trong 250 ms tới

Vì sao cần

Hẹn giờ của JavaScript lệch vài ms và dừng khi tab ở nền.

Cách làm

Cứ 100 ms gọi một lần, đặt mọi nốt trong 250 ms tới bằng `start(t)`

Số đo

Mười sáu nốt móc kép: lệch lớn nhất 0,0227 ms, đúng một mẫu.

(b) Điều kiện (11.12): cửa sổ > chu kỳ gọi + độ trễ lớn nhất

Hình 11.23 — Bộ lập lịch cửa sổ nhìn trước và kết quả đo được

Hai cách đặt lịch, đo trên cùng mười sáu nốt

Cách đặt lịch	Lệch lớn nhất	Lệch chuẩn	Nghe ra sao
Cửa sổ nhìn trước, start(t)	0,0227 ms	0,0057 ms	Đều tuyệt đối
Đặt thẳng bằng hẹn giờ JavaScript	8,33 ms	3,88 ms	Không đều; vượt ngưỡng chừng 2 ms

Cách thứ nhất lệch nhỏ hơn cách thứ hai hơn ba trăm lần.

Cửa sổ nhìn trước và chu kỳ gọi

- Cửa sổ 200 đến 300 ms dùng được cho gần như mọi trường hợp.
- Rộng quá: đổi nhịp hay đổi tiếng chậm phản ứng, vì nốt đã đặt không rút lại dễ.
- Hẹp quá: thỉnh thoảng sót một nốt, chỉ khi máy bận nên khó tái hiện.
- Chu kỳ gọi lấy chừng một phần ba đến một nửa cửa sổ.

MỤC 11.8

Đo và gỡ lỗi trong trình duyệt

- AnalyserNode: mắt nhìn vào đồ thị mà không đổi tiếng ra.
- Bảy lỗi hay gặp, sáu trong bảy kiểm bằng số.
- Quy trình gỡ lỗi ba bước biến nghe ngóng thành đo đạc.

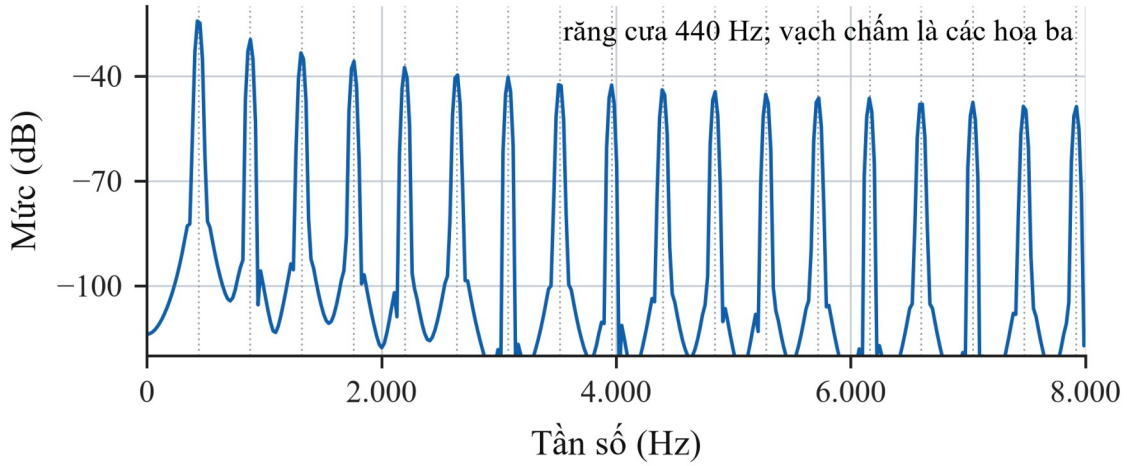
Hai công thức của mục 4.6

$$\Delta f = \frac{f_s}{N_{\text{fft}}}; \qquad T_{\text{cửasổ}} = \frac{N_{\text{fft}}}{f_s}$$

(11.13)

- Đây là mục 4.6 nguyên vẹn, chỉ đổi tên biến.
- fftSize lớn thì mịn về tần số nhưng thô về thời gian, và ngược lại.
- Đánh đổi này không có cách đi vòng.

Phổ lấy thẳng từ AnalyserNode, và ba con số phải đặt đúng



(a) Phổ lấy từ `getFloatFrequencyData`

`fftSize` = 2048: 1.024 vạch, cách nhau 21,5 Hz, cửa sổ 46,4 ms (11.13).

`smoothingTimeConstant` = 0,8: trung bình trượt; chỉ đổi cái nhìn thấy.

`minDecibels`, `maxDecibels` = −100, −30 dB: chỉ dùng cho dữ liệu dạng byte.

(b) Ba thuộc tính cấu hình

Hình 11.24 — Phổ lấy thẳng từ AnalyserNode, và ba con số phải đặt đúng

AnalyserNode không nằm trên đường tín hiệu

Cắm nó vào giữa chuỗi hay không cắm thì tiếng ra vẫn thế, và không cần nối tới destination mới chạy. Vì vậy đây là công cụ gỡ lỗi an toàn: cắm bao nhiêu cái cũng không đổi kết quả, chỉ tốn chút CPU.

Bảy lỗi hay gặp và cách kiểm từng lỗi

Triệu chứng	Nguyên nhân thường gặp	Cách kiểm
Không có tiếng gì cả	Ngữ cảnh còn ở trạng thái suspended	In `ctx.state`; gọi `resume()` khi bấm
Nghe được một lần rồi thôi	Gọi `start()` lần thứ hai trên cùng một node	Mục 11.3.3: tạo node mới cho mỗi tiếng
Tiếng tách ở mỗi lần đổi	Đặt thẳng gain bằng `setValueAtTime`	Đo năng lượng cao tần (hình 11.22)
Nhịp không đều	Đặt lịch bằng `setTimeout` thay vì `start(t)`	Đo thời điểm onset (hình 11.23)
Càng chạy càng chậm	Node cũ chưa được thu hồi vì còn tham chiếu	Đếm node sống; gỡ sau `onended`
Méo tiếng khi thêm giọng	Cộng nhiều nhánh mà không chừa headroom	Đo đỉnh; áp công thức đồng pha (mục 10.2.3)
Im hẳn giữa chừng	Một mẫu thành NaN và lan ra cả đồ thị	Kẹp mẫu số; dựng ngoại tuyến rồi soát

Hình 11.25 — Bảy lỗi hay gặp và cách kiểm từng lỗi

Ba bước biến nghe ngóng thành đo đạc

- Bước 1: dựng lại đúng đồ thị bằng OfflineAudioContext; lỗi biến mất thì là lỗi thời gian thực (ngân sách hoặc lịch).
- Bước 2: lỗi còn thì ghi AudioBuffer ra tệp; tìm NaN, mẫu vượt trần, đo phổ, đo thời điểm onset.
- Bước 3: so với bản Python cùng thuật toán; hai bản lệch ở đâu thì lỗi ở đó.
- Sáu trong bảy cách kiểm dùng công cụ đã viết ở các chương trước.

Làm ngay sau khi đọc xong chương

- Bài 1, đồ thị đầu tiên: dựng chuỗi bốn node của mục 11.2.1 với tệp WAV riêng; đổi ba tham số bộ lọc, đối chiếu đáp ứng ở mục 9.3.
- Bài 2, đo độ trễ và ngân sách: in baseLatency, outputLatency; thêm dần giọng tới khi đứt tiếng, so với công thức (11.6).
- Bài 3, bộ lập lịch: cài bộ lập lịch ở mục 11.7.4, thay bằng setTimeout, rồi chuyển tab để nghe lỗi.

Bảy bài tập của chương 11

#	Đề bài	Đáp số
1	Ở 48 kHz, khối 128 mẫu dài bao lâu; process() gọi bao nhiêu lần mỗi giây?	2,667 ms; 375 lần
2	Khối 256 mẫu ở 48 kHz: độ trễ cả đường vòng, bỏ qua phần thiết bị?	10,67 ms
3	Worklet tốn 0,9 ms mỗi khối ở 44,1 kHz: ρ; giữ tải dưới 70 % chạy nổi mấy bản?	ρ = 0,310 (31,0 %); 2 bản
4	playbackRate = 2, detune = −1200 cent: r; mẫu 3 giây phát dài bao lâu?	r = 1,000; vẫn 3 giây
5	linearRamp từ 0,2 lên 0,9 trong 40 ms: giá trị tại mốc 25 ms?	0,6375
6	setTargetAtTime từ 1,0 về 0,1 với τ = 8 ms: bao lâu còn cách đích 5 %?	23,97 ms
7	BPM = 96, móc kép: hai nốt cách bao lâu; cửa sổ 200 ms chứa mấy nốt?	156,25 ms; 2 nốt

Sáu câu phải trả lời được sau chương 11

1. Web Audio thêm phép xử lý nào mới?
Không thêm phép nào. Biquad của trình duyệt khớp công thức mục 9.3 tới 0,001 dB.

2. Đồng hồ nào dùng để đặt nốt?
`ctx.currentTime` , và chỉ qua `start(t)`: chính xác tới một mẫu; hẹn giờ JavaScript lệch 8,33 ms.

3. Mỗi lần được gọi có bao nhiêu thời gian?
128 mẫu, tức 2,90 ms ở 44,1 kHz (công thức 11.1). Trả muộn là đứt tiếng.

4. Đưa mã Python của chúng ta vào bằng cách nào?
`AudioWorklet` . Bộ nén của mục 8.3 chép sang JavaScript, lệch dưới sai số float32.

5. Đổi một tham số thế nào cho không kêu tách?
Không đặt thẳng. Ba kiểu đường bao hạ năng lượng trên 5 kHz từ −53,3 xuống −85,3 dB.

6. Đồ thị node có tự bảo vệ khỏi vượt trần không?
Không. Tám nguồn cùng `start(0)` cho đỉnh +18,06 dBFS, đúng công thức đồng pha mục 10.2.3.

Hình 11.26 — Sáu câu phải trả lời được sau chương 11

Những điều cần nhớ

1. Chương không thêm phép xử lý nào: node dựng sẵn và mô-đun tự viết khớp nhau tới 0,001 dB.
2. Cái mới là quyền điều khiển: trình duyệt gọi mã cứ 2,90 ms một lần, phải trả đúng 128 mẫu.
3. ctx.currentTime là đồng hồ thật; start(t) chính xác tới một mẫu, hẹn giờ JavaScript lệch tới 8,33 ms.
4. Node nguồn dùng một lần: dùng lại bộ đệm, dựng node mới cho mỗi tiếng.
5. Độ trễ và ngân sách tính ra được và đo được; thuật toán Python chạy trong AudioWorklet lệch dưới 152 dB.
6. Đường mũ và đường tiến dần đổi tham số không kêu tách; lịch dùng cửa sổ nhìn trước.
7. Gỡ lỗi bằng đo: dựng ngoại tuyến, soát bằng số, so với bản Python.