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

LẬP TRÌNH ÂM THANH

CHƯƠNG 12

Ứng dụng Web Audio phần 2

Chương 12 · 5 tiết · Mã học phần 531833 · 75 tiết · Học kỳ 5

Hà Nội – 2026

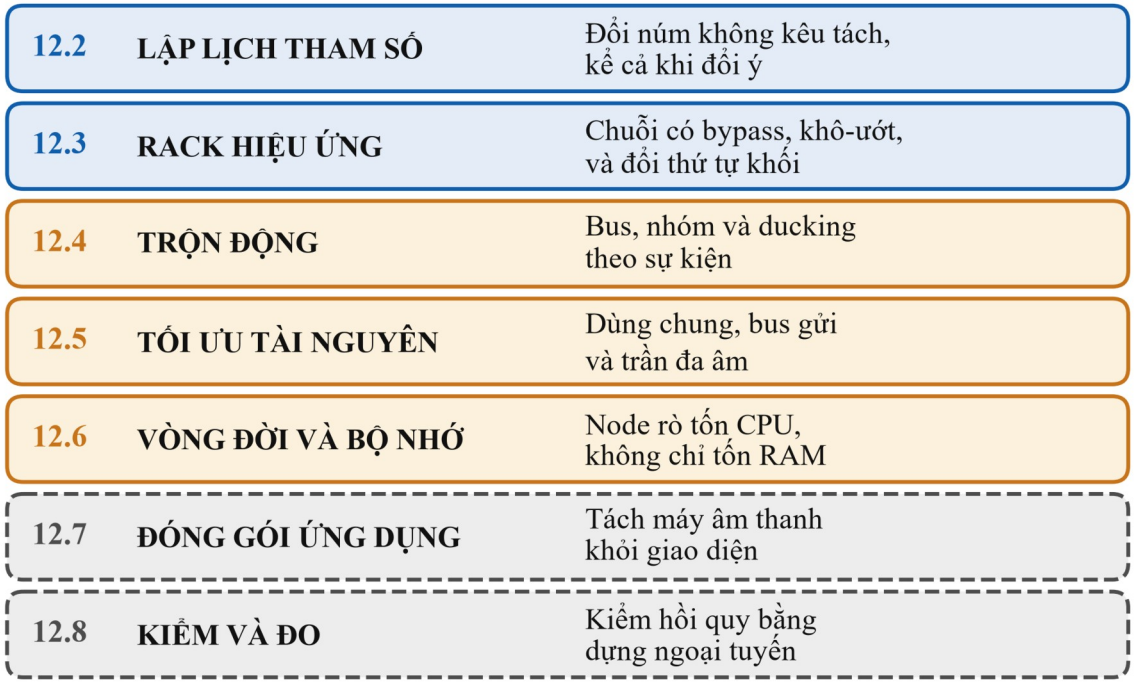
Sau chương này người học phải làm được

- Viết vào AudioParam bằng ba cách chuẩn; đo mức nhiễu răng cưa đã giảm.
- Huỷ và neo lại lịch cũ khi người dùng đổi ý; giải thích vì sao thiếu dòng neo thì sai.
- Dựng rack hiệu ứng có bypass thật; chứng minh đặt gain về 0 không tiết kiệm gì.
- Chọn luật trộn khô-ướt đúng; giải thích ảnh hưởng của thứ tự khe qua chỉ số đỉnh và độ to.
- Dựng bus và nhóm, cài ducking hai chiều; đo phần lời nổi lên trên tiếng ra.
- Chỉ ra khối nào gộp chung được; đặt trần đa âm có mờ ra và nêu cái giá phải trả.
- Viết đúng vòng đời sáu bước của một giọng; phát hiện node rò bằng cách đếm.
- Tách máy âm thanh khỏi giao diện; dựng phép kiểm hồi quy chạy tự động.

MỤC 12.1

Vị trí chương học trong học phần

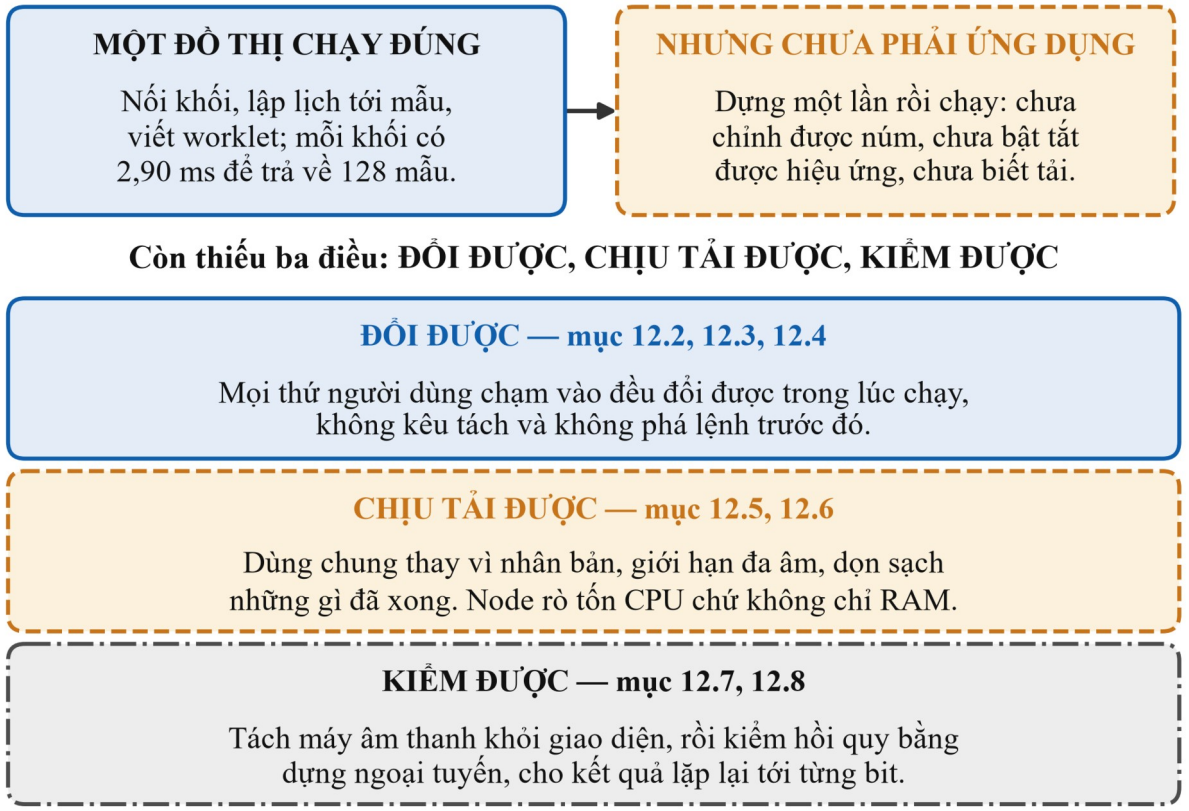
- Chương 11 dựng được đồ thị node chạy thật; chương 12 dựng thành ứng dụng giao được.
- Khoảng cách nằm ở những câu hỏi mà chương 11 chưa cần trả lời.
- Mọi câu trả lời trong chương đều là số đo trên tiếng ra.



Chương 11 dựng được một đồ thị chạy đúng. Chương này biến nó thành một ứng dụng: đổi được, chịu tải được và kiểm được.

Bốn khoảng cách cần lấp

- Viết vào tham số thế nào cho khỏi kêu tách.
- Tắt một hiệu ứng thì máy có rẻ đi thật không.
- Ba mươi giọng cùng lúc thì nên dùng chung gì.
- Tiếng đã tắt thì node của nó đi đâu.



Cả ba đều phải đo, vì không câu nào trả lời được bằng cách nghe.

Hình 12.2 — Bốn khoảng cách giữa một đồ thị chạy được và một ứng dụng giao được

Bảy mục và câu trả lời đo được

Mục	Câu hỏi	Trả lời đo được
12.2	Viết vào tham số thế nào cho khỏi kêu tách?	Ramp thay vì gán thẳng: sạch hơn 71 dB
12.3	Bypass bằng gain = 0 có rẻ không?	Không; ngắt nối rẻ 3,5 lần
12.3	Luật wet/dry nào giữ được mức?	Tuyến tính hõm 2,99 dB; đẳng công suất gợn 0,03
12.4	Ducking nâng lời lên bao nhiêu?	Từ dưới nhạc 2,62 dB lên trên nhạc 8,43 dB
12.5	Một bus vang chung rẻ hơn bao nhiêu?	Từ 3,2 lần (4 giọng) lên 16,2 lần (32 giọng)
12.6	Node phát xong mà không ngắt thì tốn gì?	Chậm hơn 1,77 lần; bộ nhớ thật không trả lại
12.8	Dựng lại cảnh có ra kết quả cũ không?	Đúng từng bit: sai lệch 0 trên 88.200 mẫu

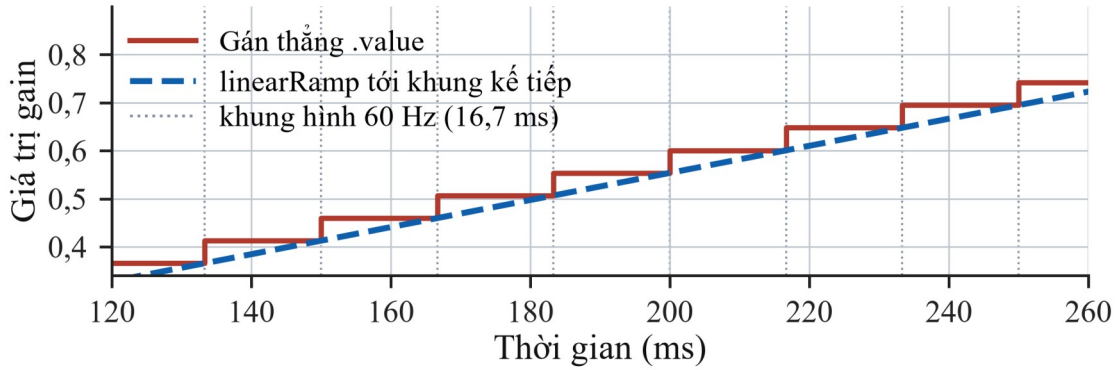
Học cách đo, đừng học thuộc con số

Con số của máy dựng giáo trình không phải con số của máy người dùng. Thứ đáng học là hình dạng kết quả: cái nào tăng tuyến tính, cái nào nằm ngang, cái nào không đổi. Hình dạng ấy đúng trên mọi máy.

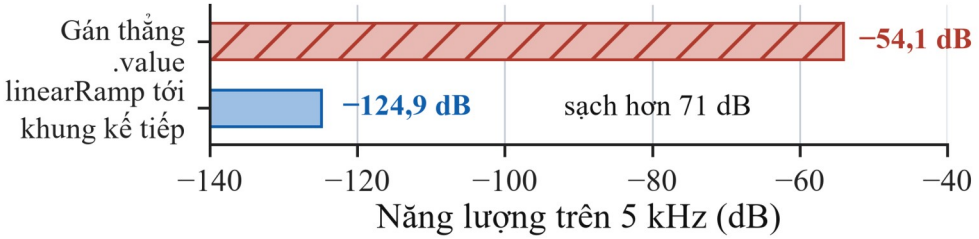
MỤC 12.2

Lập lịch tham số — đổi núm mà không kêu tách

- Nối thanh trượt HTML với AudioParam là việc đầu tiên ai cũng làm, và hay làm sai.
- Ba cách viết vào tham số, hai bước huỷ và neo, một khoảng an toàn.
- Bốn quy tắc gói gọn trong một hàm dat() duy nhất.



(a) Cùng một cử chỉ, hai cách viết vào tham số



(b) Phần năng lượng trên 5 kHz trong tiếng ra

Cách viết ai cũng nghĩ tới đầu tiên — và nó sai

```
slider.addEventListener("input", (e) => {
  gainNode.gain.value = Number(e.target.value);   // SAI
});
```

Giá trị đúng; sai ở chỗ tham số đứng yên giữa hai lần gán rồi nhảy một bước.

Ba nhịp thời gian cùng chạy trong một ứng dụng

Nhịp	Tần suất	Khoảng cách	Ai điều khiển
Khung hình giao diện	60 lần/giây	16,7 ms	Trình duyệt, theo màn hình
Khối dựng âm thanh	344,53 lần/giây	2,90 ms	Web Audio, cố định 128 mẫu
Mẫu	44.100 lần/giây	22,7 µs	Phần cứng

Giữa hai lần giao diện kịp viết vào tham số, Web Audio đã dựng xong gần sáu khối nhận cùng một giá trị.

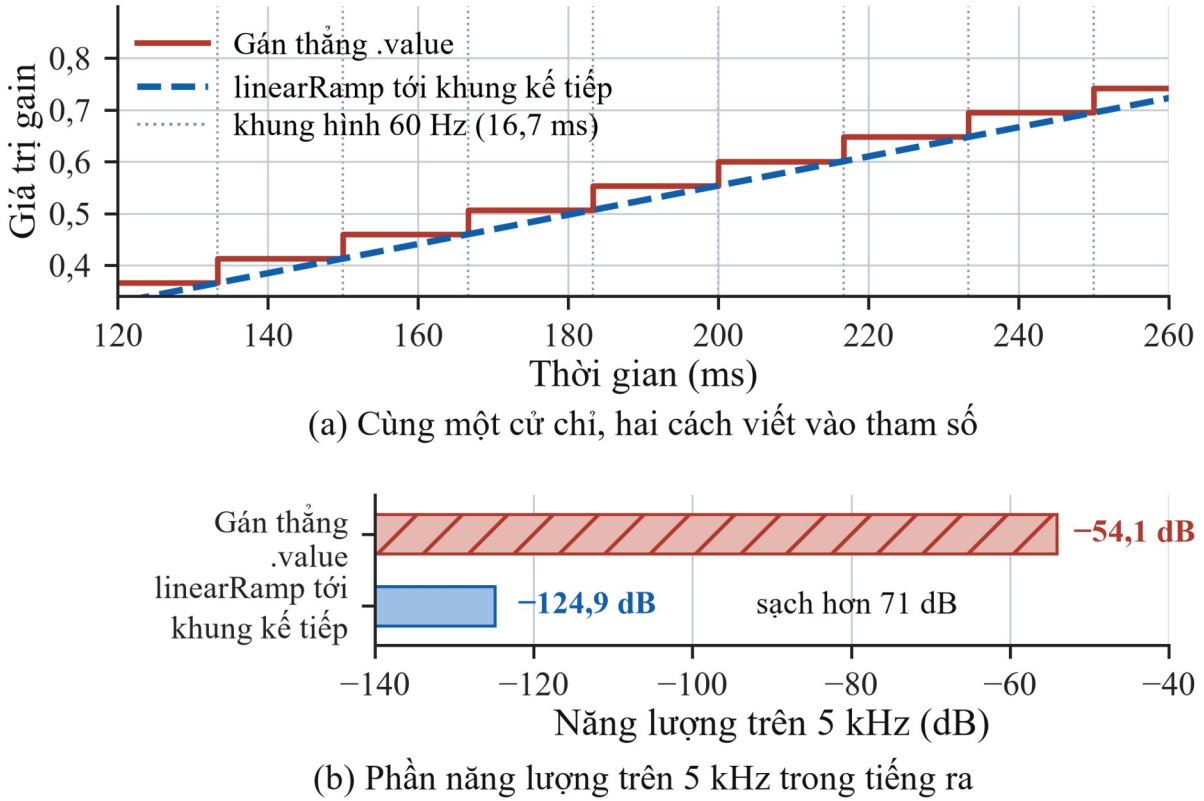
Mỗi bước nhảy là một chỗ gãy trong đường bao

$$\Delta v_k = \left| v(t_{k+1}) - v(t_k) \right|, \qquad t_{k+1} - t_k = \frac{1}{f_{\text{khung}}}$$

(12.1)

- Δv là độ lớn bước nhảy giữa hai khung hình liên tiếp; f là tần số khung hình.
- Chỗ gãy gây tiếng tách như ở mục 6.6, nhưng lặp lại 60 lần mỗi giây.
- Kết quả là tiếng rè liên tục, hay bị đổ nhầm cho bộ lọc.

Nhiễu răng cưa của phép gán thẳng và cách sửa bằng một dòng



Hình 12.3 — Nhiễu răng cưa của phép gán thẳng, và cách sửa bằng một dòng

Cách sửa: ramp tới khung hình kế tiếp

```
const KHUNG = 1 / 60;                    // 16,7 ms

slider.addEventListener("input", (e) => {
  const v = Number(e.target.value);
  const t = ctx.currentTime;
  const p = gainNode.gain;
  p.cancelScheduledValues(t);
  p.setValueAtTime(p.value, t);          // NEO
  p.linearRampToValueAtTime(v, t + KHUNG);
});
```

Năng lượng trên 5 kHz hạ từ −54,1 xuống −124,9 dB: sạch hơn 71 dB.

Vì sao ramp tới khung sau chứ không phải 5 ms

Ramp ngắn hơn khoảng cách hai khung thì tham số tới đích rồi đứng chờ, lại thành bậc thang. Ramp dài hơn thì mỗi lệnh mới cắt ngang lệnh cũ và cử chỉ bị trễ. Ramp đúng một khung cho đường liên tục mà vẫn bám tay.

Ba cách đưa tham số tới giá trị mới

Cách gọi	Hình dạng	Tới đích khi nào	Dùng cho
setValueAtTime	Bước nhảy thẳng đứng	Ngay lập tức	Tham số ngoài đường tín hiệu; neo
linearRampToValueAtTime	Đường thẳng	Đúng thời điểm đã hẹn	Núm điều khiển, mờ chéo, pan
exponentialRampToValueAtTime	Đường mũ	Đúng thời điểm đã hẹn	Tần số và cao độ
setTargetAtTime	Đường mũ tiệm cận	Không bao giờ tới hẳn	Ducking, tắt tiếng, đường bao trơn

Giới hạn của ba cách lập lịch

- **exponentialRamp:** không đi qua và không xuất phát từ 0; đích bằng 0 là lỗi.
- **Muốn tắt hẳn:** ramp xuống giá trị rất nhỏ rồi setValueAtTime(0) ngay sau.
- **setTargetAtTime:** nhận hằng số thời gian; sau một hằng số đi được 63 %, sau ba hằng số 95 %.
- **Cả ba:** tính từ giá trị đang có tại thời điểm hẹn, không phải lúc gọi hàm.

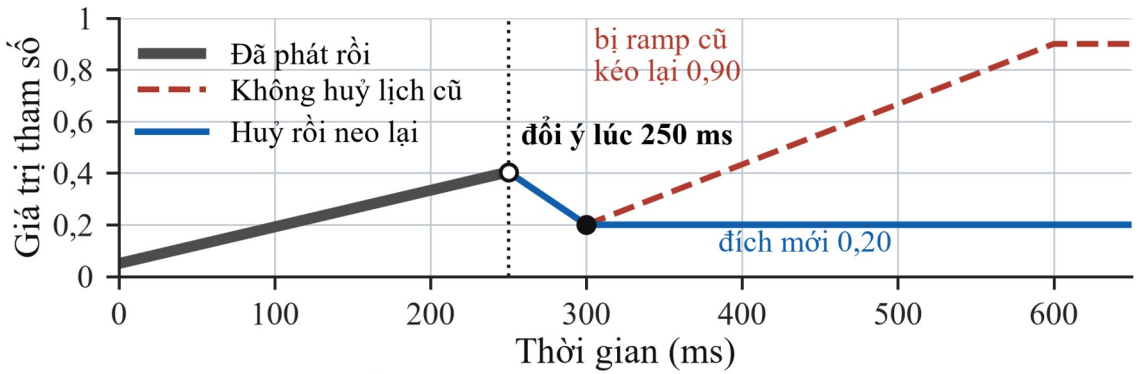
Mỗi loại tham số một cách đổi và một hằng số thời gian

THAM SỐ	CÁCH ĐỔI, HẰNG SỐ	VÌ SAO
Gain, âm lượng	linearRamp tới khung sau **16,7 ms**	Đổi nhanh thì kêu tách (mục 12.2.1)
Tắt hẳn một tiếng	setTarget rồi đặt thẳng 0 **τ = 5 ms**	Đường mũ không có chỗ gãy (mục 11.7.2)
Tần số cắt bộ lọc	linearRamp **20 – 50 ms**	Bước nhảy tần số dễ nghe hơn bước nhảy mức
Vị trí pan	linearRamp **30 – 80 ms**	Tai định vị chậm nên đổi chậm được (mục 10.6)
Ducking nhạc nền	setTarget hai chiều **30 xuống, 200 lên**	Hạ nhanh cho kịp lời, trả chậm cho êm (mục 12.4.3)
Ngưỡng, tỉ số bộ nén	Đặt thẳng cũng được —	Không nằm trên đường tín hiệu nên không sinh bước nhảy
Đổi cả một preset	Mờ chéo hai nhánh **20 – 50 ms**	Nhiều tham số đổi cùng lúc, không ramp từng cái được

Hình 12.4 — Mỗi loại tham số một cách đổi và một hằng số thời gian

Lệnh mới không thay thế lệnh cũ

- AudioParam giữ một danh sách sự kiện chứ không giữ một giá trị.
- Lệnh mới xếp vào sau; đường đi thật là kết quả của cả danh sách.
- Bản không huỷ chạm 0,2 rồi bị ramp cũ kéo ngược về 0,9.
- Bản huỷ đúng cách dừng ở 0,2 và giữ nguyên.



(a) Đổi ý giữa chừng một đường bao đang chạy

BƯỚC 1 — ĐỌC
```
const v = p.value;  // giá trị hiện tại
```

BƯỚC 2 — HUỶ
```
p.cancelScheduledValues(t);
```

BƯỚC 3 — NEO
```
p.setValueAtTime(v, t);
p.linearRampToValueAtTime(moi, t + fade);
```

Thiếu bước 2 thì lệnh cũ còn hiệu lực và kéo tham số ngược lại.
Thiếu bước 3 thì tham số nhảy về giá trị cũ.

(b) Ba dòng đi liền nhau mỗi lần lập lịch mới

Hình 12.5 — Quên huỷ lịch cũ, và cách huỷ rồi neo lại cho đúng

Ba dòng phải đi liền nhau mỗi lần lập lịch mới

```
// SAI — chỉ huỷ mà không neo lại
p.cancelScheduledValues(t);
p.linearRampToValueAtTime(v, t + 0.05);

// ĐÚNG — huỷ, NEO giá trị hiện tại, rồi mới ramp
p.cancelScheduledValues(t);
p.setValueAtTime(p.value, t);
p.linearRampToValueAtTime(v, t + 0.05);
```

Thiếu dòng neo, ramp mới không biết xuất phát từ đâu: tham số rơi về giá trị của sự kiện còn lại gần nhất.

cancelAndHoldAtTime: trên giấy gọn hơn, thực tế phải kiểm

Đặc tả có hàm huỷ-và-neo trong một lời gọi. Trên bản cài đặt dùng dựng giáo trình, nó cho kết quả giống cancelScheduledValues, tức không neo. Vì vậy bài dạy mẫu ba dòng; người học tự kiểm trên trình duyệt đích (bài tập 3).

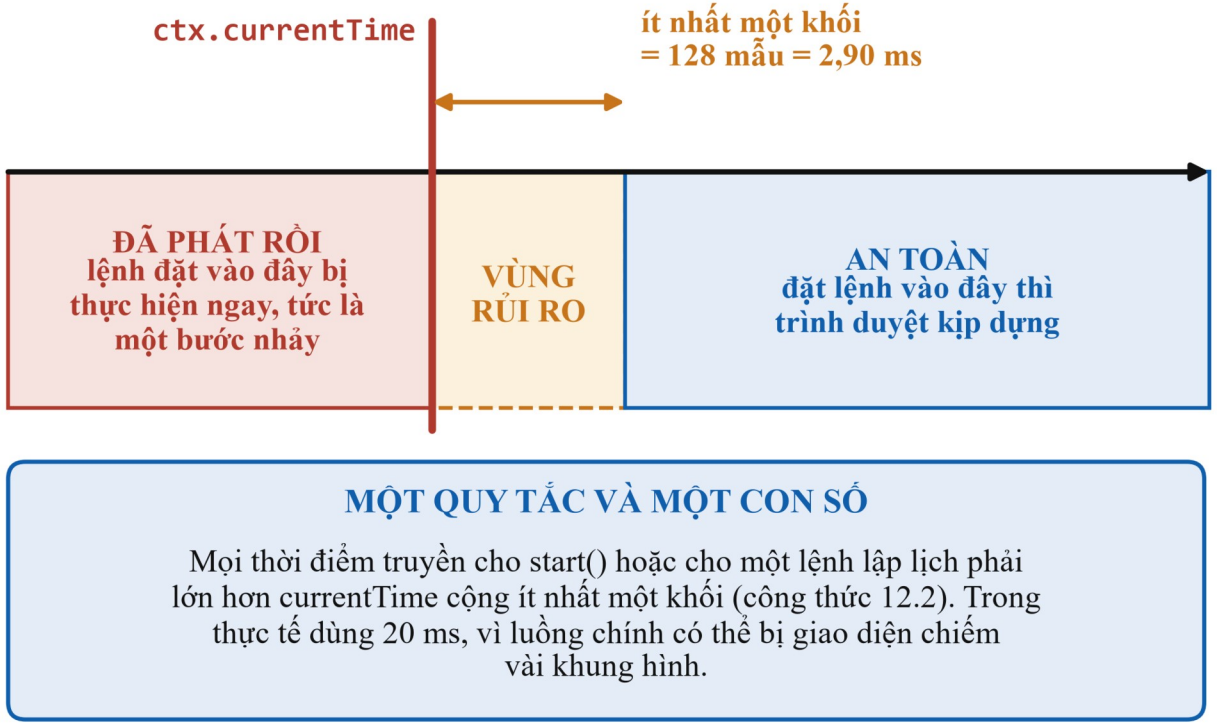
Đừng bao giờ hẹn vào đúng currentTime

$$t_{\text{lệnh}} \geq t_{\text{hiệntại}} + t_{\text{antoàn}}, \qquad t_{\text{antoàn}} \geq \frac{N_q}{f_s}$$

(12.2)

- currentTime là đầu khối đang dựng; khi mã chạy, khối ấy có thể đã dựng xong một phần.
- N = 128 mẫu là số mẫu một khối; f là tần số lấy mẫu.
- Ở 44.100 Hz, khoảng an toàn tối thiểu là 2,90 ms; thực tế dùng 20 ms.

Vì sao mọi lệnh lập lịch đều cộng thêm khoảng an toàn



Hình 12.6 — Vì sao mọi lệnh lập lịch đều cộng thêm một khoảng an toàn

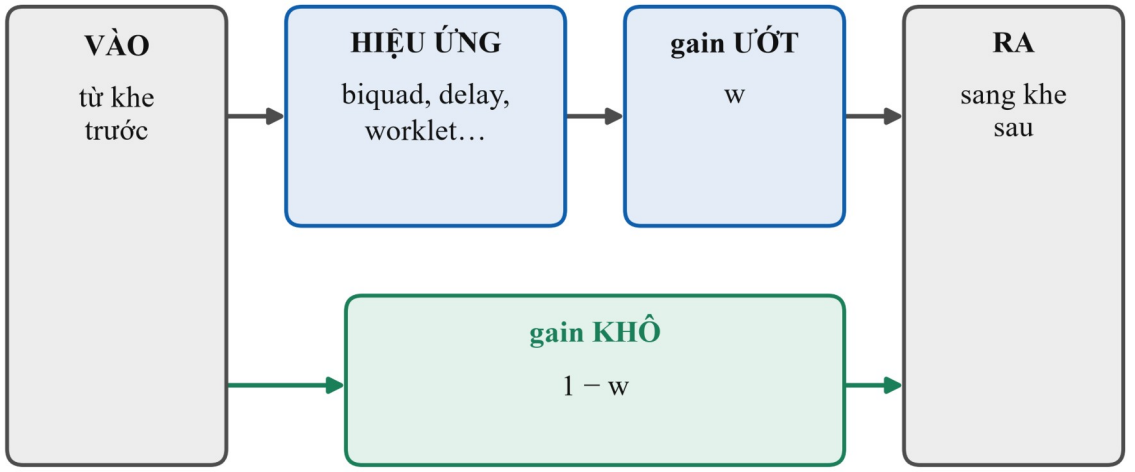
Không gán thẳng; luôn huỷ, neo, ramp

Không gán thẳng vào .value của tham số trên đường tín hiệu. Mỗi lần lập lịch: huỷ, neo, rồi ramp. Thời điểm hẹn luôn là currentTime cộng khoảng an toàn. Gói cả ba vào một hàm dat() và bắt mọi núm đi qua nó.

MỤC 12.3

Rack hiệu ứng — chuỗi khối bật tắt được

- Rack là chuỗi connect cộng ba thứ: bật tắt từng khe, trộn khô-ướt, đổi thứ tự.
- Bypass có hai cách; chỉ một cách nghe được.
- Luật trộn và thứ tự khe quyết định mức và đỉnh tiếng ra.



VÌ SAO KHÔNG CHỈ MỘT NODE HIỆU ỨNG

Hai gain không dùng để chỉnh âm lượng. Chúng là chỗ duy nhất bật tắt được khe mà không kêu tách: mờ chéo w từ 1 về 0 trong 20 ms thì khe tắt êm. Bật tắt bằng disconnect tạo bước nhảy; hình 12.8 đo được chênh 59 dB.

Chi phí của rack là tổng chi phí các khe đang bật

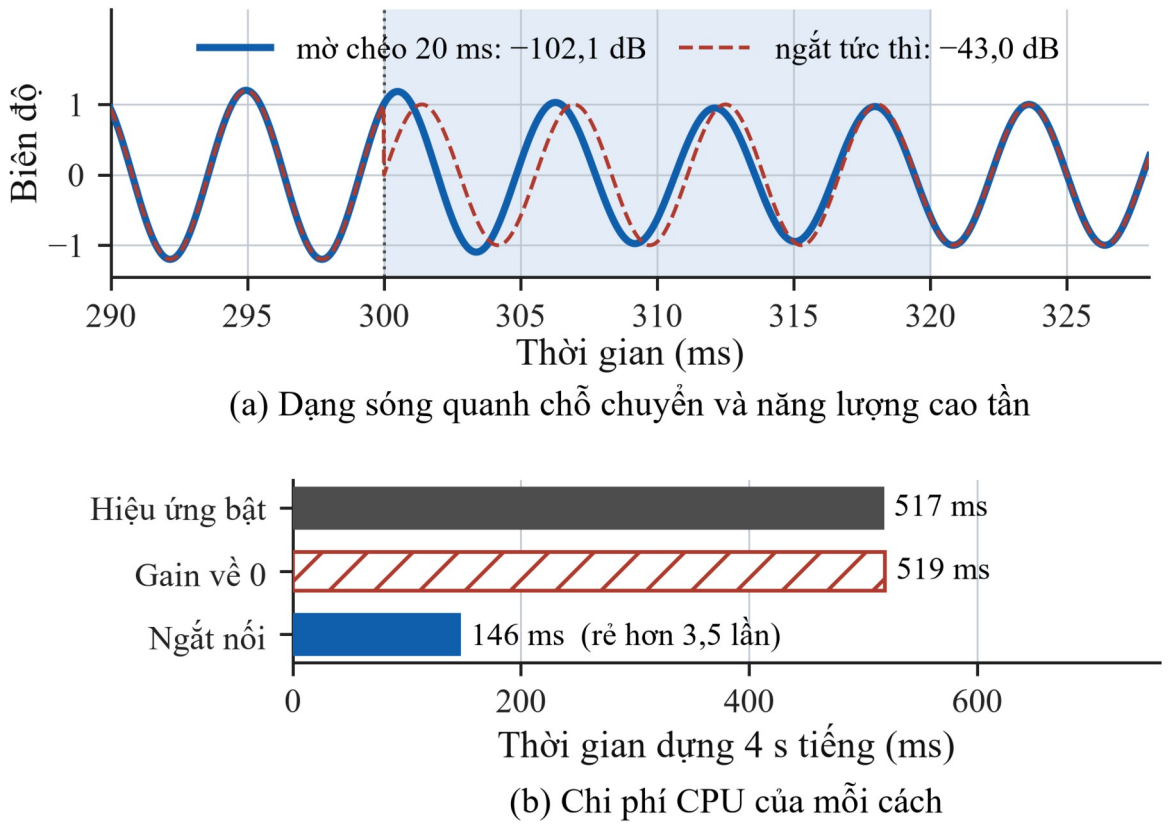
$$C_{\text{rack}} = \sum_{i=1}^{M} b_i c_i, \qquad b_i \in \{0, 1\}$$

(12.3)

- b là cờ bật tắt của khe thứ i; c là chi phí của khe ấy.
- Câu hỏi: khi b bằng 0, cái gì làm cho c biến mất khỏi tổng?

Hai cách bypass, chỉ một cách nghe được

- Ngắt nối: disconnect khối cần tắt rồi nối thẳng đầu vào sang đầu ra.
- Mờ chéo: dựng hai nhánh song song, chuyển giữa hai gain.
- Năng lượng cao tần quanh chỗ chuyển: −43,0 dB (ngắt tức thì), −102,1 dB (mờ chéo 20 ms).
- Chênh 59 dB: một tiếng bụp rõ so với không nghe thấy gì.



Hình 12.8 — Hai cách bypass, và cái giá tính bằng CPU của mỗi cách

Bypass đúng cách: mờ chéo trước, ngắt nối sau

```
batTat(khe, bat, ms = 20) {
  const t = this.ctx.currentTime + 0.02;
  const g = ms / 1000;
  for (const [p, dich] of [[khe.uotGain.gain, bat ? 1 : 0],
                           [khe.khoGain.gain, bat ? 0 : 1]]) {
    p.cancelScheduledValues(t);
    p.setValueAtTime(p.value, t);
    p.linearRampToValueAtTime(dich, t + g);
  }
  if (!bat) {
    setTimeout(() => khe.uot.disconnect(), ms + 60);
  } else if (!khe.dangNoi) {
    khe.uot.connect(khe.uotGain); khe.dangNoi = true;
  }
}
```

Đặt gain về 0 không tiết kiệm gì cả

Cùng đồ thị 24 giọng, dựng 4 giây tiếng: bật hiệu ứng 517 ms; gain về 0 là 519 ms; ngắt nối 146 ms. Web Audio vẫn chạy mọi node rồi mới nhân với 0. Chỉ ngắt nối mới rẻ, 3,5 lần.

Mức tổng hợp phụ thuộc hệ số tương quan

$$L = 10\log_{10}\left(g_k^2 + g_u^2 + 2\rho\, g_k g_u\right)$$

(12.4)

- Luật tuyến tính: nhánh khô nhân (1 − x), nhánh ướt nhân x.
- ρ là hệ số tương quan giữa hai nhánh; ρ = 1 thì giữ được mức.
- Nhánh ướt gần như không tương quan với nhánh khô: ρ về gần 0, mức tụt ở giữa.

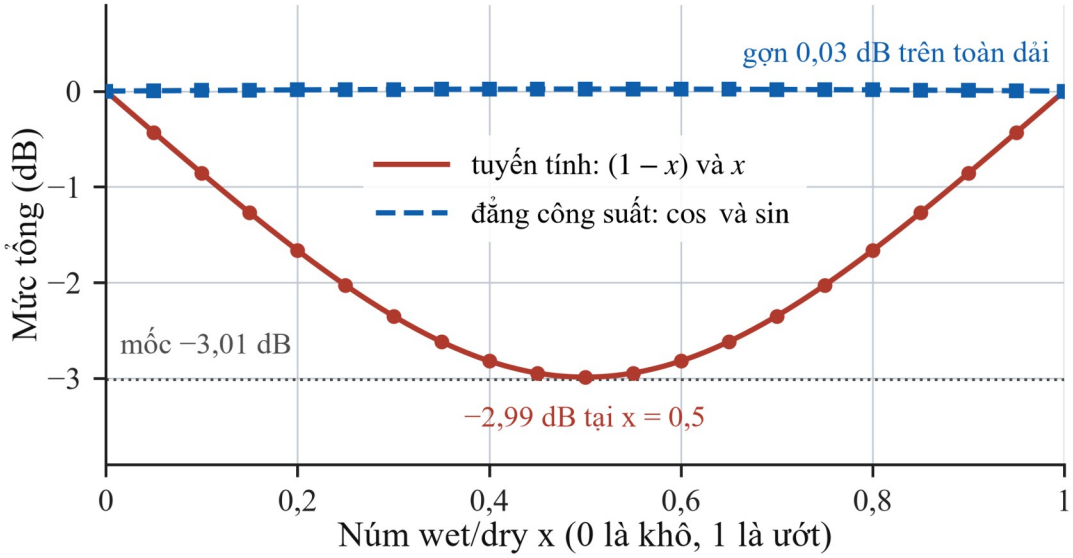
Dùng cosin và sin thay cho phép chia tuyến tính

$$g_k = \cos\frac{\pi x}{2}, \quad g_u = \sin\frac{\pi x}{2}; \qquad x \in [0,\ 1]$$

(12.5)

- Đo trên đồ thị chạy thật với một tiếng vang thật.
- Luật tuyến tính hõm 2,99 dB tại x = 0,5.
- Luật đẳng công suất chỉ gợn tổng cộng 0,03 dB trên toàn dải.

Hai luật trộn khô-ướt và mức đo trên toàn dải núm



(a) Mức tổng theo hai luật trộn

Nhánh ướt đã qua hiệu ứng nên gần như không tương quan với nhánh khô (ρ gần 0): mức tổng cộng theo công suất, đúng công thức (12.4).

Hình 12.9 — Hai luật trộn khô-ướt, và mức đo được trên toàn dải núm

Cộng hai thứ không tương quan thì cộng theo công suất

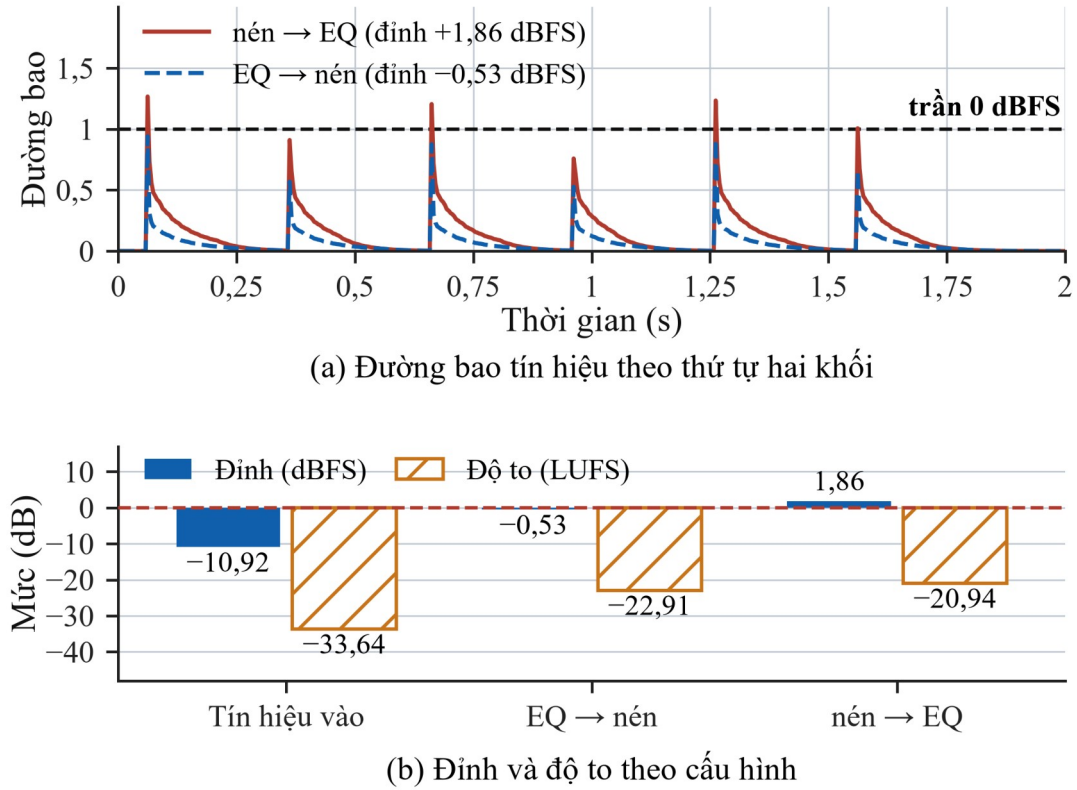
Mục 10.2: cộng hai nguồn không tương quan lên 3,01 dB, không phải 6,02. Mục 10.4: luật pan tuyến tính hụt 3,01 dB ở giữa. Mục này: luật wet/dry tuyến tính hõm 2,99 dB. Ba mục, cùng một phép toán.

Cùng hai khối, đổi thứ tự, hai con số đổi hẳn

Cấu hình	Đỉnh (dBFS)	Độ to (LUFS)	Nhận xét
Tín hiệu vào	−10,92	−33,64	Chưa qua khối nào
EQ → nén	−0,53	−22,91	Bộ nén thấy phần đã nâng và kiểm soát được
nén → EQ	+1,86	−20,94	Phần nâng đi thẳng ra, vượt trần 0 dBFS

Tiếng trống, EQ (equalizer – bộ cân bằng tần số) nâng 9 dB ở 900 Hz, bộ nén ngưỡng −24 với tỉ số 6.

Đổi thứ tự hai khối trong rack, hai con số đổi theo



Hình 12.10 — Đổi thứ tự hai khối trong rack, và hai con số đổi theo

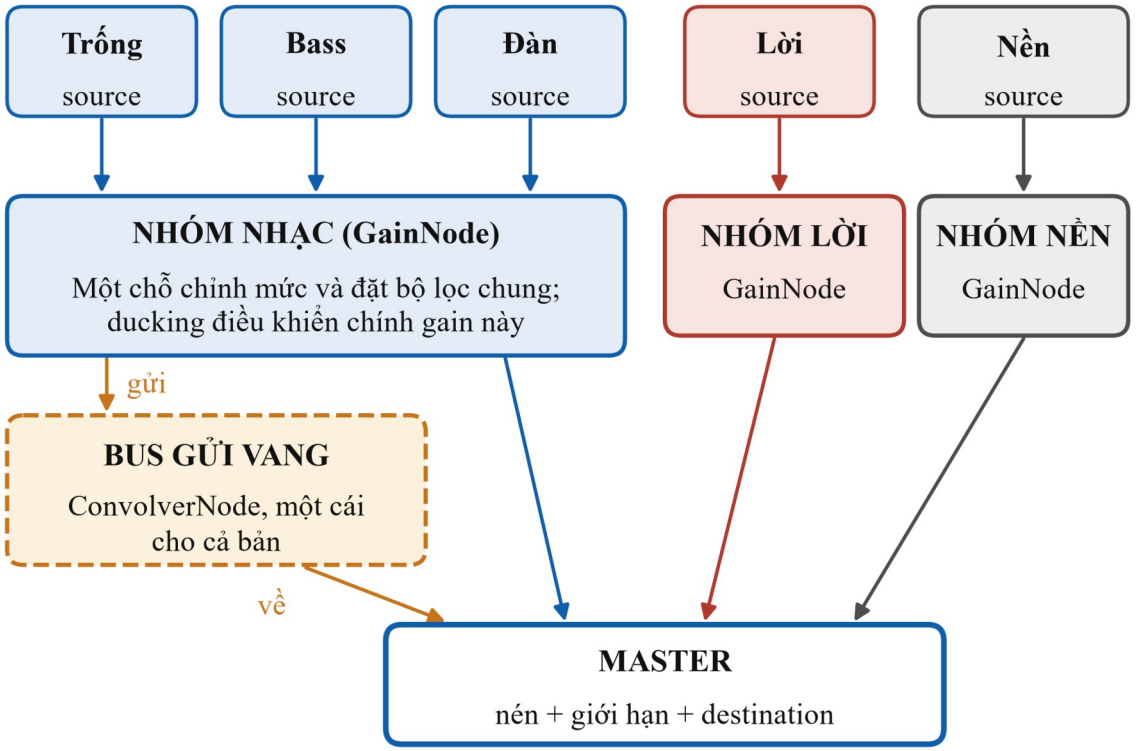
Thứ tự nên dùng khi chưa có lý do khác

- **Đầu tiên:** cổng và lọc bỏ tiếng ồn, kẻo bộ nén khuếch đại tiếng ồn nền.
- **Tiếp theo:** EQ cắt và EQ nâng, đều trước bộ nén.
- **Sau đó:** bộ nén, rồi hiệu ứng thời gian — để đuôi vang không bị nén theo.
- **Cuối cùng:** bộ giới hạn (mục 8.6), khối duy nhất có quyền nói câu cuối về đỉnh.

MỤC 12.4

Trộn động — khi bản trộn tự đổi theo sự kiện

- Nguồn sinh ra và chết đi liên tục, nên núm phải gắn vào tầng sống suốt phiên.
- Ducking hạ nhạc nền khi có lời.
- Mỗi tham số động phải có đúng một chủ.



Tầng 1, nguồn: sinh ra và chết đi liên tục. Tầng 2, nhóm: dựng một lần, sống suốt phiên. Tầng 3, bus gửi: nhánh song song cộng vào master.

Bộ nhóm tối thiểu cho ứng dụng trò chơi hoặc phát thanh

Nhóm	Chứa gì	Việc đặc biệt
Nhạc	Nhạc nền, nhạc chuyển cảnh	Là nhóm bị ducking hạ xuống
Lời	Thoại, thuyết minh	Kích hoạt ducking; thường không gửi vang
Hiệu ứng	Va chạm, bước chân, giao diện	Nhóm đông nhất; đặt trần đa âm
Môi trường	Gió, mưa, đám đông	Ít giọng nhưng dài; gửi vang nhiều nhất
Master	Bốn nhóm trên	Đặt bộ giới hạn cuối (mục 8.6)

Nhóm nằm trên đường tín hiệu, bus gửi nằm song song

Tắt nhóm là tắt cả loại tiếng ấy. Tắt bus gửi thì tiếng vẫn còn, chỉ mất phần vang. Lỗi điển hình: đặt convolver trên đường tín hiệu thay vì song song, rồi không tắt được tiếng khô.

Hạ mức nhóm nhạc theo tín hiệu điều khiển

$$g(t) = 1 - (1 - g_{\min})\, d(t), \qquad d(t) \in [0,\ 1]$$

(12.6)

- d là tín hiệu điều khiển từ 0 tới 1 theo mức có tiếng ở nhóm ưu tiên.
- g là mức thấp nhất mà nhóm bị hạ xuống.
- Với g = 0,28 như trong phép đo, nhóm nhạc bị hạ 11,1 dB khi có lời.

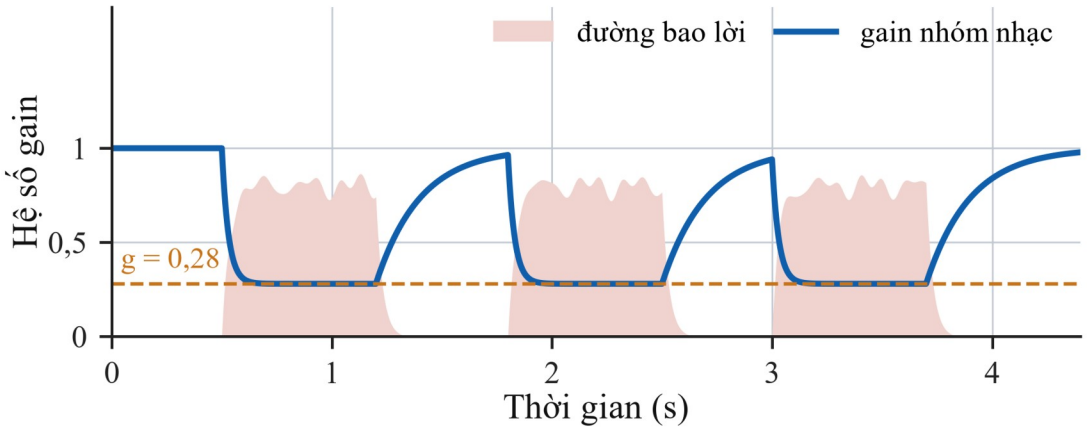
Chênh lệch lời–nhạc tăng đúng bằng lượng đã hạ

$$\Delta_{\text{lời}} = L_{\text{lời}} - L_{\text{nhạc}}\ ; \qquad \Delta_{\text{lời}}^{\text{duck}} = \Delta_{\text{lời}} - 20\log_{10} g_{\min}$$

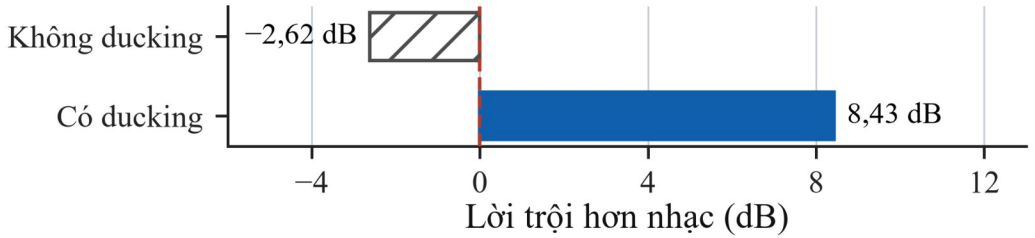
(12.7)

- Không ducking: lời thấp hơn nhạc 2,62 dB.
- Có ducking: lời trội hơn nhạc 8,43 dB.
- Đổi 11,06 dB, khớp với lượng hạ đã đặt; đo trên tiếng ra thật.

Đường điều khiển ducking và mức lời nổi lên trên tiếng ra



(a) Gain nhóm nhạc theo ba câu lời: hạ τ = 30 ms, trả τ = 200 ms



(b) Mức lời so với nhạc trên tiếng ra, đổi 11,06 dB

Hình 12.12 — Đường điều khiển ducking, và lời nổi lên bao nhiêu trên tiếng ra

Ducking hai chiều: hạ nhanh, trả chậm

```
const TAU_XUONG = 0.030;    // 30 ms — kịp trước âm tiết đầu
const TAU_LEN   = 0.200;    // 200 ms — trả chậm cho êm
const G_MIN     = 0.28;

duck(bat) {
  const p = this.nhacGain.gain;
  const t = this.ctx.currentTime + 0.02;
  p.cancelScheduledValues(t);
  p.setValueAtTime(p.value, t);          // NEO — bắt buộc
  p.setTargetAtTime(bat ? G_MIN : 1,
                    t, bat ? TAU_XUONG : TAU_LEN);
}
```

Hai hằng số thời gian không bao giờ bằng nhau

Hạ phải nhanh, 20 đến 40 ms, vì tai nghe ngay khi lời bị nhạc lấn. Trả phải chậm, 150 đến 400 ms, kẻo nhạc phồng lên giữa hai câu. Đặt hai hằng số bằng nhau thì nhạc nền thở theo từng câu.

Sự kiện nào đổi tham số nào, đổi ra sao, điểm dễ sai đi kèm

Sự kiện	Đổi tham số nào	Đổi ra sao	Điểm dễ sai
Lời bắt đầu	gain nhóm nhạc	`setTarget` `τ = 30 ms`	Bỏ sót việc trả lại khi lời im
Lời kết thúc	gain nhóm nhạc	`setTarget` `τ = 200 ms`	Trả nhanh quá: nhạc nghe như bơm
Vào phòng khác	mức gửi tới bus vang	`linearRamp` `300 ms`	Đổi buffer vang giữa chừng: kêu tách
Nhân vật lại gần	gain nguồn và pan	`linearRamp` `80 ms`	Ramp pan nhanh: tiếng nhảy trong đầu
Sức khoẻ xuống thấp	bộ lọc thấp qua trên nhóm	`ramp fc` `500 ms`	Đổi type bộ lọc lúc đang chạy
Tạm dừng trò chơi	gain master	`setTarget` `τ = 15 ms`	suspend() tức thì cũng kêu tách
Chuyển cảnh	mờ chéo hai nhánh	`400 – 800 ms`	Bỏ sót việc ngắt nhánh cũ sau khi mờ xong

Mọi điểm dễ sai trong bảng đều liên quan đến thời gian, không phải đến giá trị.

Hình 12.13 — Sự kiện nào đổi tham số nào, đổi ra sao, và điểm dễ sai đi kèm

Mọi tham số động đều phải có một chủ duy nhất

Ducking và núm của người dùng cùng ghi vào một AudioParam thì bên gọi sau thắng. Cách sửa: hai GainNode nối tiếp, mỗi cái một chủ, mức cuối là tích của hai. Mỗi GainNode thêm tốn khoảng 25 µs lúc dựng.

MỤC 12.5

Tối ưu tài nguyên — làm ít đi chứ không làm nhanh hơn

- Gộp khối tuyến tính: lọc tổng bằng tổng các bản đã lọc.
- Bus gửi vang là chỗ tiết kiệm lớn nhất của chương.
- Trần đa âm tiết kiệm CPU nhưng phải trả bằng đỉnh và độ thưa.

Cách riêng nhân chi phí hiệu ứng lên N lần

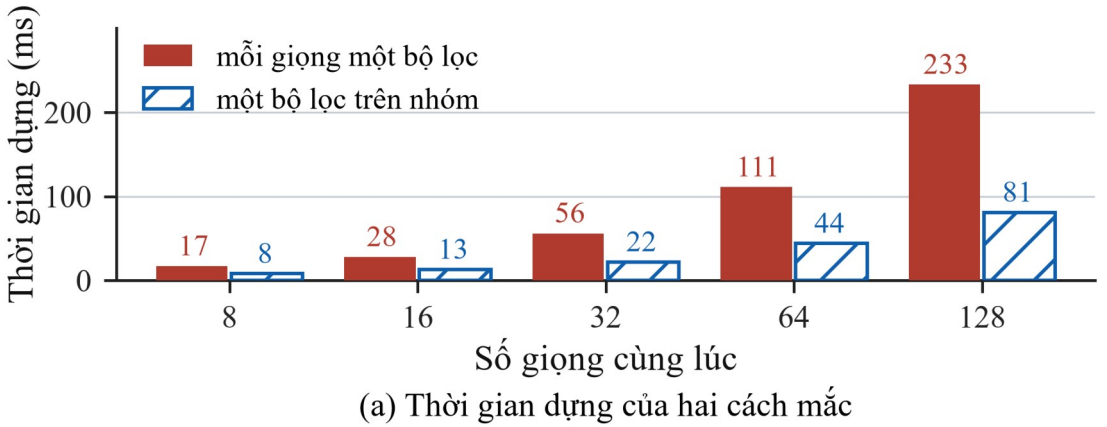
$$C_{\text{riêng}} = N(c_{\text{giọng}} + c_{\text{hiệuứng}}); \qquad C_{\text{chung}} = N c_{\text{giọng}} + c_{\text{hiệuứng}}$$

(12.8)

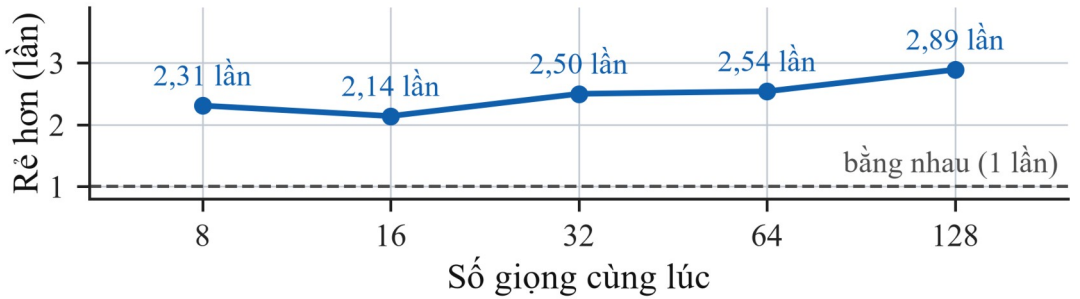
- Cách riêng: mỗi giọng một BiquadFilterNode.
- Cách chung: một bộ lọc đặt trên GainNode của nhóm, sau chỗ mọi giọng đã cộng.
- Bộ lọc tuyến tính, bất biến theo thời gian: hai cách cho cùng kết quả nghe được, khác giá.

Hai cách mắc bộ lọc và tỉ số giữa hai cách

- Với 8 giọng, cách chung rẻ hơn 2,31 lần.
- Với 128 giọng, rẻ hơn 2,89 lần.
- Tỉ số nhích lên theo số giọng: càng đông càng đáng gộp.



(a) Thời gian dựng của hai cách mắc



(b) Tỉ số giữa hai cách, tăng theo số giọng

Hình 12.14 — Cùng một tiếng, hai cách mắc bộ lọc, và tỉ số giữa hai cách

Khối nào gộp chung được, khối nào không

Khối	Gộp được?	Lý do
GainNode	Được	Cộng rồi nhân bằng nhân rồi cộng
BiquadFilterNode	Được	Lọc tổng bằng tổng các bản lọc
DelayNode	Được	Trễ tổng bằng tổng các bản trễ
ConvolverNode	Được; lãi nhất	Chập tổng bằng tổng các bản chập
DynamicsCompressorNode	Không	Nén tổng khác tổng các bản nén (mục 8.4)
WaveShaperNode	Không	Méo tổng khác tổng các bản méo
PannerNode	Không	Gộp là mất vị trí riêng (mục 10.7)

Quy tắc gọn: khối tuyến tính thì gộp được, khối phi tuyến thì không.

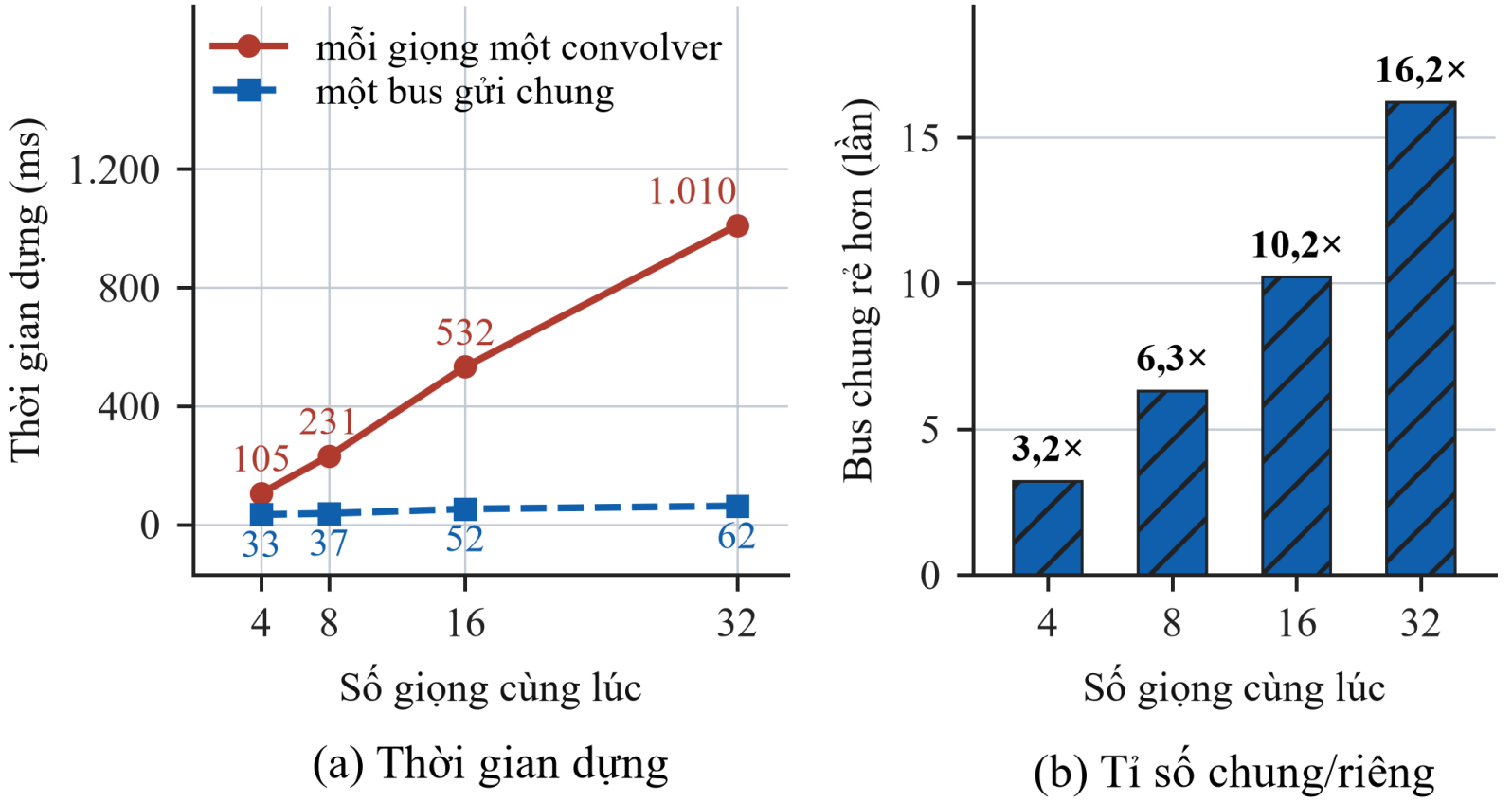
Chập tổng đã trộn bằng tổng các bản đã chập

$$y[n] = \sum_{i=1}^{N} x_i[n] + \left(h * \sum_{i=1}^{N} s_i x_i \right)[n]$$

(12.9)

- Phép chập là phép tính đắt nhất trong một đồ thị âm thanh thông thường.
- s là mức gửi của giọng thứ i: mỗi giọng có phần vang riêng.
- Phép chập chỉ chạy một lần.

Cái giá của việc mỗi giọng mang một phòng vang riêng



Hình 12.15 — Cái giá của việc mỗi giọng mang một phòng vang riêng

Vang riêng và vang chung — thời gian dựng đo được

Số giọng	Mỗi giọng một convolver	Một bus gửi chung	Chung rẻ hơn
4	105 ms	33 ms	3,2 lần
8	231 ms	37 ms	6,3 lần
16	532 ms	52 ms	10,2 lần
32	1.010 ms	62 ms	16,2 lần

Cột hai tăng gần tuyến tính; cột ba gần đứng yên: đúng hình dạng mà công thức (12.9) dự đoán.

Bus gửi vang — dựng một lần, mọi giọng dùng chung

```
// LÚC NẠP ỨNG DỤNG — đúng một lần
this.vang = this.ctx.createConvolver();
this.vang.buffer = await this.napIR("phong_vua.wav");
this.vangTra = this.ctx.createGain();
this.vangTra.gain.value = 0.8;
this.vang.connect(this.vangTra).connect(this.master);

// MỖI GIỌNG — chỉ thêm một GainNode nhỏ
if (gui > 0) {
  const sg = this.ctx.createGain();
  sg.gain.value = gui;              // hệ số s trong (12.9)
  g.connect(sg).connect(this.vang);
}
```

Số giọng sống là giá trị nhỏ hơn giữa hai vế

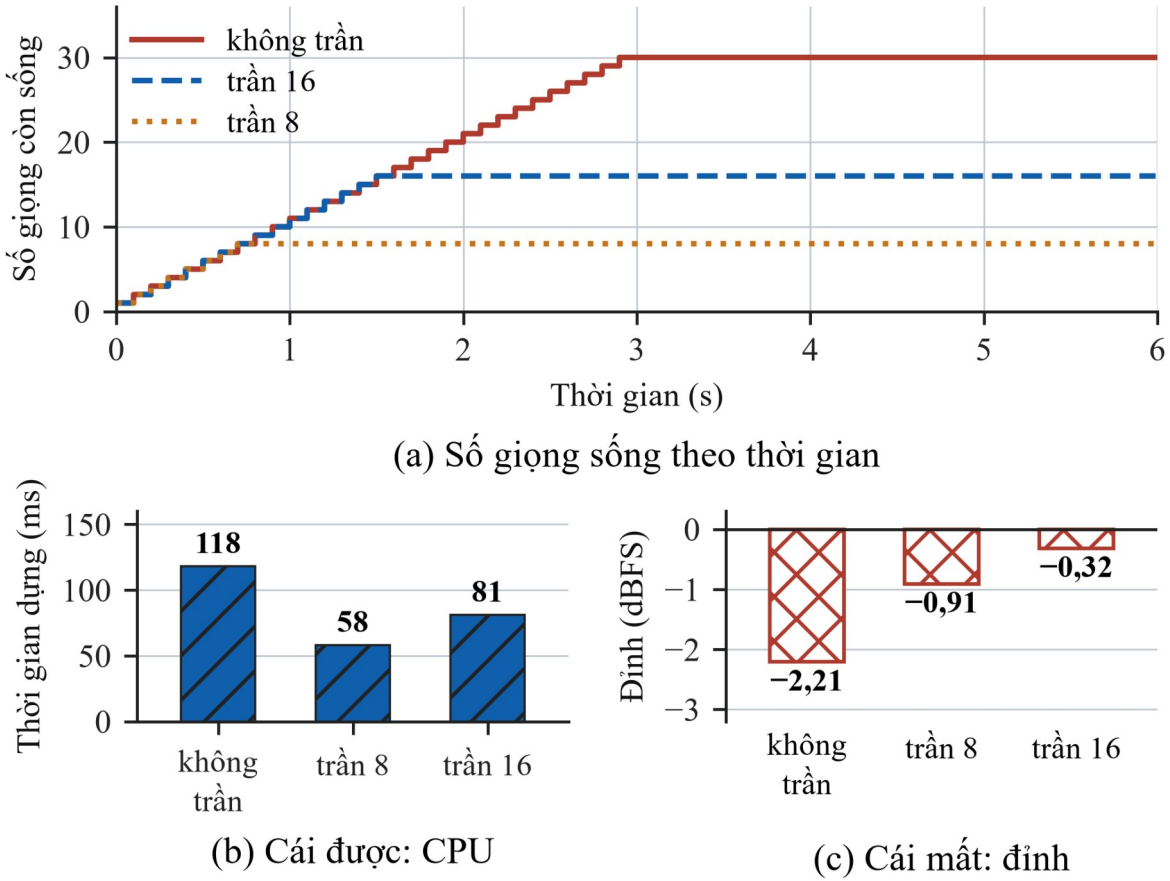
$$N_{\text{sóng}} = \min\left(N_{\text{trần}}, \left\lceil \frac{T_{\text{đuôi}}}{T_{\text{bước}}} \right\rceil\right)$$

(12.10)

- Vế thứ nhất là trần do người lập trình đặt ra.
- Vế thứ hai là tỉ số giữa độ dài đuôi và khoảng cách hai lần phát.
- Vế thứ hai là con số tự nhiên của bản nhạc.

Cái được tính bằng CPU, cái mất tính bằng dB

- Mẫu 150 nhịp/phút, sáu lớp: không trần dựng 118 ms.
- Trần 8 còn 58 ms; trần 16 còn 81 ms.
- Cái giá: đỉnh đi từ −2,21 lên −0,91 dBFS (decibel so với mức toàn thang, Full Scale).
- Tiếng nghe thưa hơn chứ không chỉ nhẹ máy hơn.



Hình 12.16 — Trần đa âm: cái được tính bằng CPU và cái mất tính bằng decibel

Trần đa âm có mờ ra, không cắt thẳng

```
phatCoTran(ten, opts) {
  if (this.song.size >= TRAN) {
    const cu = this.song.keys().next().value;   // giọng già nhất
    this.dungMuot(cu, 0.015);                   // mờ 15 ms rồi stop
  }
  return this.phat(ten, opts);
}

dungMuot(id, ms) {
  const { s, g } = this.song.get(id);
  const t = this.ctx.currentTime + 0.005;
  g.gain.cancelScheduledValues(t);
  g.gain.setValueAtTime(g.gain.value, t);
  g.gain.linearRampToValueAtTime(0, t + ms);
  s.stop(t + ms + 0.005);      // onended sẽ tự dọn
}
```

Cắt giọng già nhất, không bao giờ cắt giọng mới nhất

Giọng già thường đã ở phần đuôi, tai ít để ý. Cắt giọng nhỏ nhất đúng hơn nhưng tốn công theo dõi. Người dùng vừa bấm mà không nghe thấy gì là lỗi nghiêm trọng. Luôn mờ ra 10 đến 20 ms thay vì stop thẳng.

Sáu việc tối ưu xếp theo tỉ số được trên công bỏ ra

VIỆC	ĐO ĐƯỢC BAO NHIÊU	CÁI GIÁ PHẢI TRẢ
1. Bus gửi chung cho vang	16,2 lần ở 32 giọng	Mọi giọng dùng chung một phòng vang
2. Ngắt nối nhánh không dùng	3,5 lần so với đặt gain bằng 0	Phải mờ chéo trước khi ngắt (mục 12.3.2)
3. Lọc chung theo nhóm	2,89 lần ở 128 giọng	Mất khả năng chỉnh bộ lọc riêng từng giọng
4. Trần đa âm	2,0 lần khi trần bằng 8	Tiếng thưa đi; nghe rồi hãy chọn
5. Dựng sẵn đồ thị cố định	Mỗi node ≈ 25 µs (công thức 12.11)	Tốn bộ nhớ thường trực, ít khi đáng lo
6. Đổi thuật toán bộ lọc	Chưa đo, và thường không đáng	Tốn nhiều công nhất, được ít nhất

Xếp theo tỉ số được trên công bỏ ra, không theo thứ tự thường nghĩ tới trước.

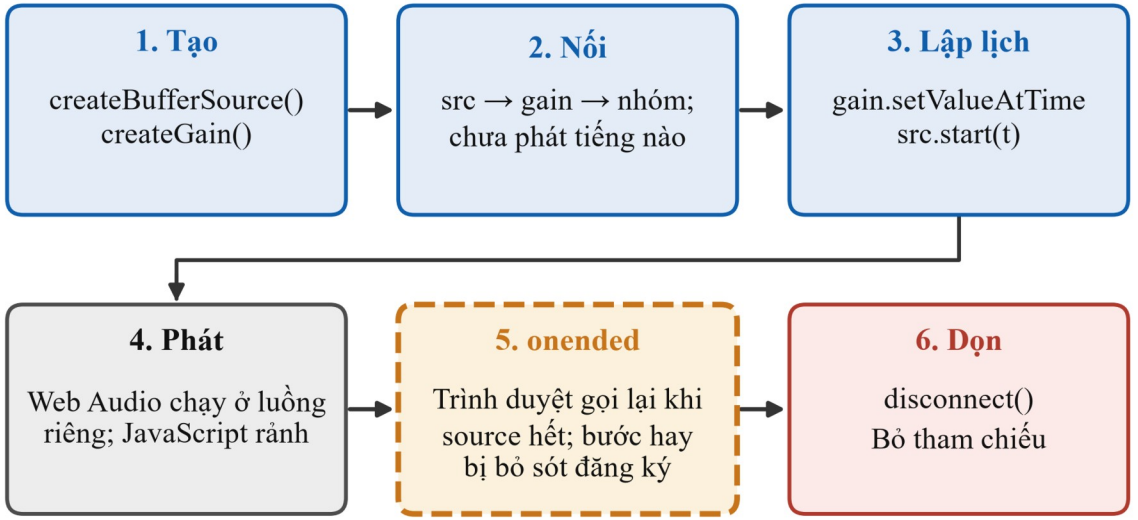
Dòng 1 và 2 cho tỉ số cao nhất và đo được ngay trong chương này.

Hình 12.17 — Sáu việc tối ưu xếp theo tỉ số được trên công bỏ ra

MỤC 12.6

Vòng đời và bộ nhớ — node đã chết đi đâu

- Mỗi tiếng phát ra là một vòng đời đầy đủ gồm sáu bước.
- Node bị bỏ sót vẫn tốn CPU và bộ nhớ.
- Chính việc dựng đồ thị cũng có giá.



Bỏ sót đăng ký onended thì bước 6 không bao giờ chạy: node vẫn nối vào đồ thị và trình duyệt vẫn chạy qua nó mỗi khối 128 mẫu, dù nó câm.

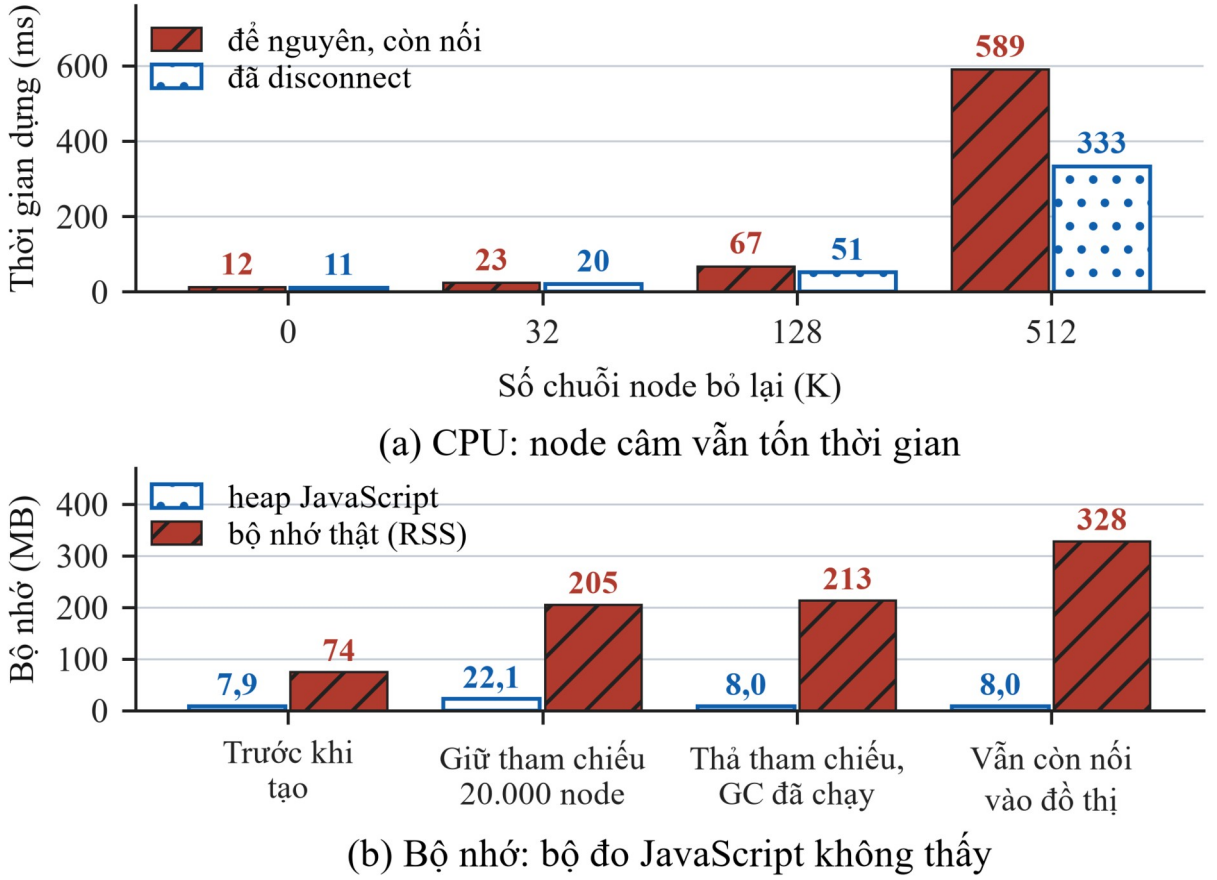
```
src.onended = () => {
  src.disconnect(); gain.disconnect(); song.delete(id);
};
```

Sáu bước, ai gọi, và chuyện gì sai nếu bỏ

Bước	Ai gọi	Bỏ thì sao
1. Tạo node	Mã của người lập trình	Không có tiếng
2. Nối vào đồ thị	Mã của người lập trình	Node chạy nhưng tiếng không tới đâu
3. Lập lịch tham số	Mã của người lập trình	Tiếng vào đột ngột, kêu tách
4. Phát	Web Audio, luồng riêng	Không có gì sai
5. onended	Trình duyệt gọi ngược lại	Bỏ đăng ký thì bước 6 không chạy
6. Ngắt nối, bỏ tham chiếu	Mã, trong bước 5	Rò node: chậm dần, tốn bộ nhớ

Node đã phát xong vẫn tốn CPU

- Node còn nối vào đồ thị thì trình duyệt còn chạy qua nó mỗi khối 128 mẫu.
- Im không có nghĩa là không tốn.
- Tỉ số dừng ở 1,77 lần: node câm vẫn rẻ hơn node đang kêu.
- Rò làm ứng dụng chậm dần chứ không chết ngay.



Hình 12.19 — Cái giá của node bị bỏ sót, đo bằng CPU và bằng bộ nhớ

Bỏ lại K chuỗi node đã phát xong — thời gian dựng

Số chuỗi bỏ lại	Còn nối vào đồ thị	Đã disconnect	Chậm hơn
0	12 ms	11 ms	1,05 lần
32	23 ms	20 ms	1,13 lần
128	67 ms	51 ms	1,31 lần
512	589 ms	333 ms	1,77 lần

Công cụ đo bộ nhớ của trình duyệt không thấy node rò

Tạo 20.000 GainNode: heap JavaScript lên 22,1 MB, thả tham chiếu rồi thu gom thì về 8,0 MB. Nhưng khi còn nối vào đồ thị, bộ nhớ thật leo thêm 115 MB trong khi heap đứng yên. Muốn chứng minh rò thì đếm Map các giọng sống: phải trở về rỗng.

Ngân sách một khối chia cho chi phí một node

$$N_{\text{node}}^{\max} = \frac{t_q}{t_{\text{node}}} = \frac{2{,}90\ \text{ms}}{25\ \mu\text{s}} \approx 116 \tag{12.11}$$

- Mỗi node mất khoảng 23 đến 28 µs để tạo và nối (64 node: 26,0 µs; 4.096 node: 27,8 µs).
- Kết quả khoảng 116 node: dựng nhiều hơn trong một khung là tiêu hết thời gian một khối.
- Hai luồng khác nhau nên không ngắt tiếng ngay, nhưng khung hình giật.
- Chia việc dựng ra nhiều khung, hoặc dựng sẵn bể node trừ BufferSourceNode.

Cái gì phải dọn, cái gì phải giữ, và dọn bằng cách nào

ĐỐI TƯỢNG	VÒNG ĐỜI	DỌN BẰNG CÁCH NÀO
BufferSourceNode	Một lần rồi kết thúc	onended → disconnect() → bỏ tham chiếu
GainNode của giọng	Theo giọng	Dọn cùng source trong chính onended ấy
GainNode của nhóm	Suốt phiên	Không dọn: dựng một lần lúc nạp
ConvolverNode của bus	Suốt phiên	Không dọn (mục 12.5.2)
AudioBuffer đã giải mã	Suốt phiên	Giữ trong Map; chỉ bỏ khi đổi màn chơi
AudioWorkletNode	Theo hiệu ứng	disconnect() và đóng port trước khi bỏ
AudioContext	Suốt phiên	close() khi rời trang; suspend() khi tạm dừng

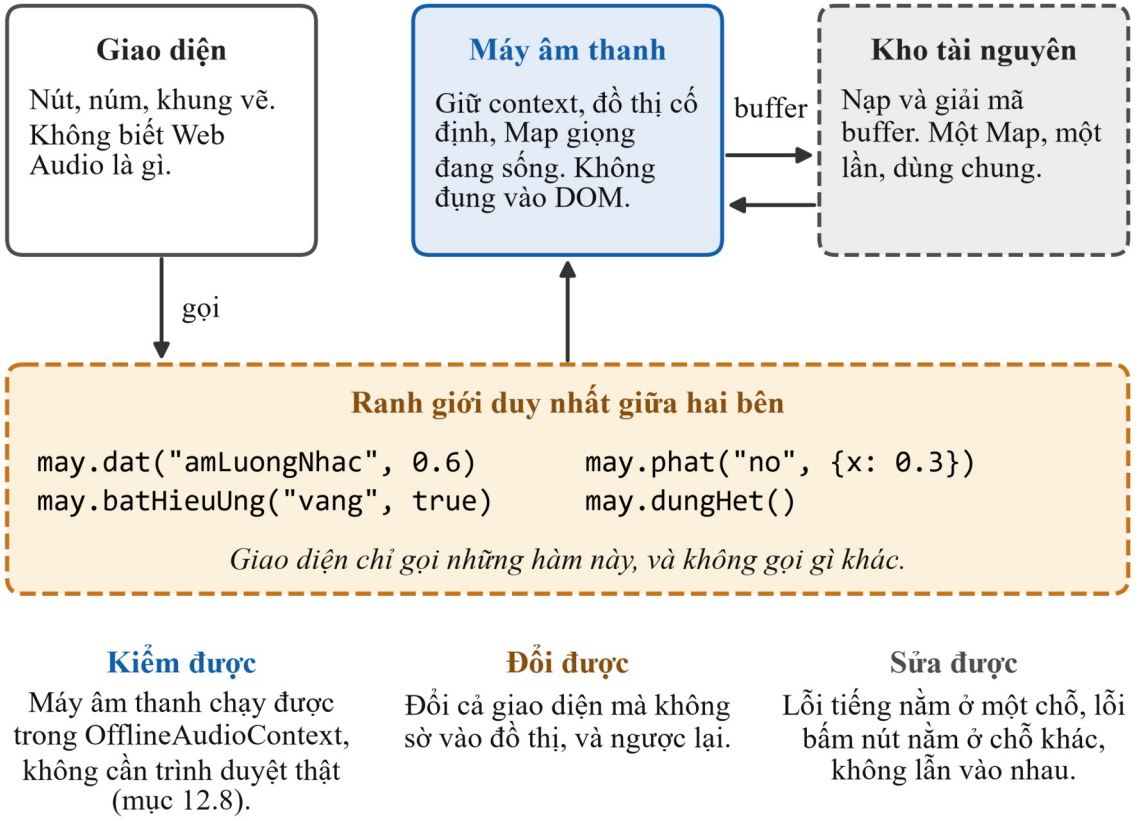
Hai dòng đỏ trên sinh ra hàng nghìn lần mỗi phút; các dòng còn lại sống lâu hơn một giọng. Nhầm hai nhóm ấy với nhau là gốc của hầu hết lỗi rò.

Hình 12.20 — Cái gì phải dọn, cái gì phải giữ, và dọn bằng cách nào

MỤC 12.7

Đóng gói ứng dụng — tách máy âm thanh khỏi giao diện

- Ba mô-đun, một ranh giới duy nhất.
- AudioContext chỉ mở được sau cử chỉ thật của người dùng.
- Hai mảnh mã gói cả chương: dựng một lần và sinh nhiều lần.



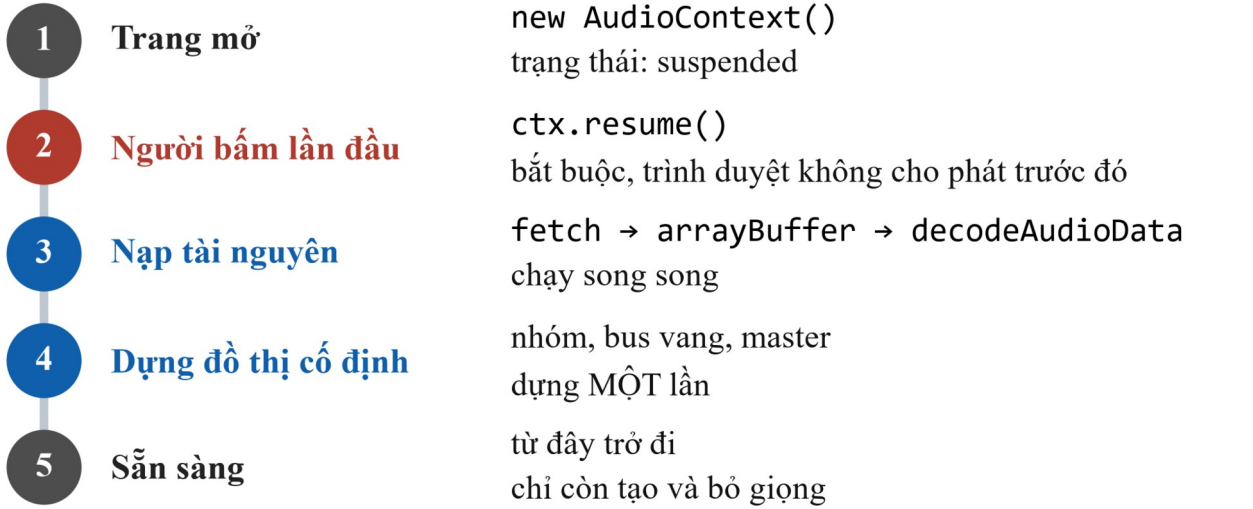
Ba mô-đun: việc của mỗi bên và thứ bị cấm

Mô-đun	Việc của nó	Bị cấm
Giao diện	Đọc trạng thái nút, núm; vẽ đồng hồ mức; gọi hàm của máy âm thanh	Biết tới AudioContext, node, currentTime
Máy âm thanh	Giữ context, dựng đồ thị cố định, quản lý giọng, lập lịch tham số	Đụng vào DOM; tự đọc tệp
Kho tài nguyên	Nạp, giải mã buffer; giữ Map dùng chung	Biết đồ thị mắc thế nào

Phép thử nghiêm nhất: chạy được máy âm thanh trong OfflineAudioContext, không có trình duyệt.

Năm mốc từ lúc mở trang tới lúc sẵn sàng phát

- AudioContext ở trạng thái suspended cho tới khi có cử chỉ thật.
- Làm sai thì bấm lần đầu im, lần hai mọi thứ đã hẹn bung ra cùng lúc.
- Nguyên nhân: các lệnh start hẹn vào một đồng hồ chưa chạy.



Điểm dễ sai số một của mọi ứng dụng Web Audio

Context tạo lúc trang mở luôn ở trạng thái suspended. Mọi lệnh start() gọi trước resume() sẽ xếp hàng và bung ra cùng một lúc khi người dùng bấm, nghe như một tiếng nổ. Phải kiểm ctx.state trước khi phát.

Nạp tài nguyên và dựng đồ thị có thể đổi thứ tự cho nhau, nhưng mốc người bấm lần đầu thì không: nó do trình duyệt quy định, không do chúng ta.

Hình 12.22 — Năm mốc từ lúc trang mở tới lúc ứng dụng sẵn sàng phát

Hai mảnh mã: dựng một lần và sinh ra hàng nghìn lần

```
class MayAmThanh {
 async moMay() {
  const c = new AudioContext();
  await c.resume();
  const m = c.createGain();
  const n = c.createGain();
  const r = c.createConvolver();
  n.connect(m); r.connect(m);
  m.connect(c.destination);
  this.ctx = c; this.nhac = n;
  this.vang = r;
  this.song = new Map();
 }
 dat(ten, v, ms = 30) {
  const p = this.param(ten);
  const t = this.ctx.currentTime
   + 0.02;
  p.cancelScheduledValues(t);
  p.setValueAtTime(p.value, t);
  p.linearRampToValueAtTime(
   v, t + ms / 1000);
 }
}
```

(a) Máy âm thanh: hàm dat()

```
phat(ten, muc = 1, gui = 0) {
 const c = this.ctx;
 const b = this.kho.get(ten);
 if (!b) return null;
 const s = c.createBufferSource();
 s.buffer = b;
 const g = c.createGain();
 g.gain.value = muc;
 s.connect(g).connect(this.nhac);
 if (gui > 0) {
  const k = c.createGain();
  k.gain.value = gui;
  g.connect(k).connect(this.vang);
 }
 const id = Symbol();
 s.onended = () => {
  s.disconnect(); g.disconnect();
  this.song.delete(id);
 };
 this.song.set(id, s);
 s.start(c.currentTime + 0.02);
 return id;
}
```

(b) Một giọng: hàm phat()

Hình 12.23 — Hai mảnh mã: thứ dựng một lần, và thứ sinh ra hàng nghìn lần

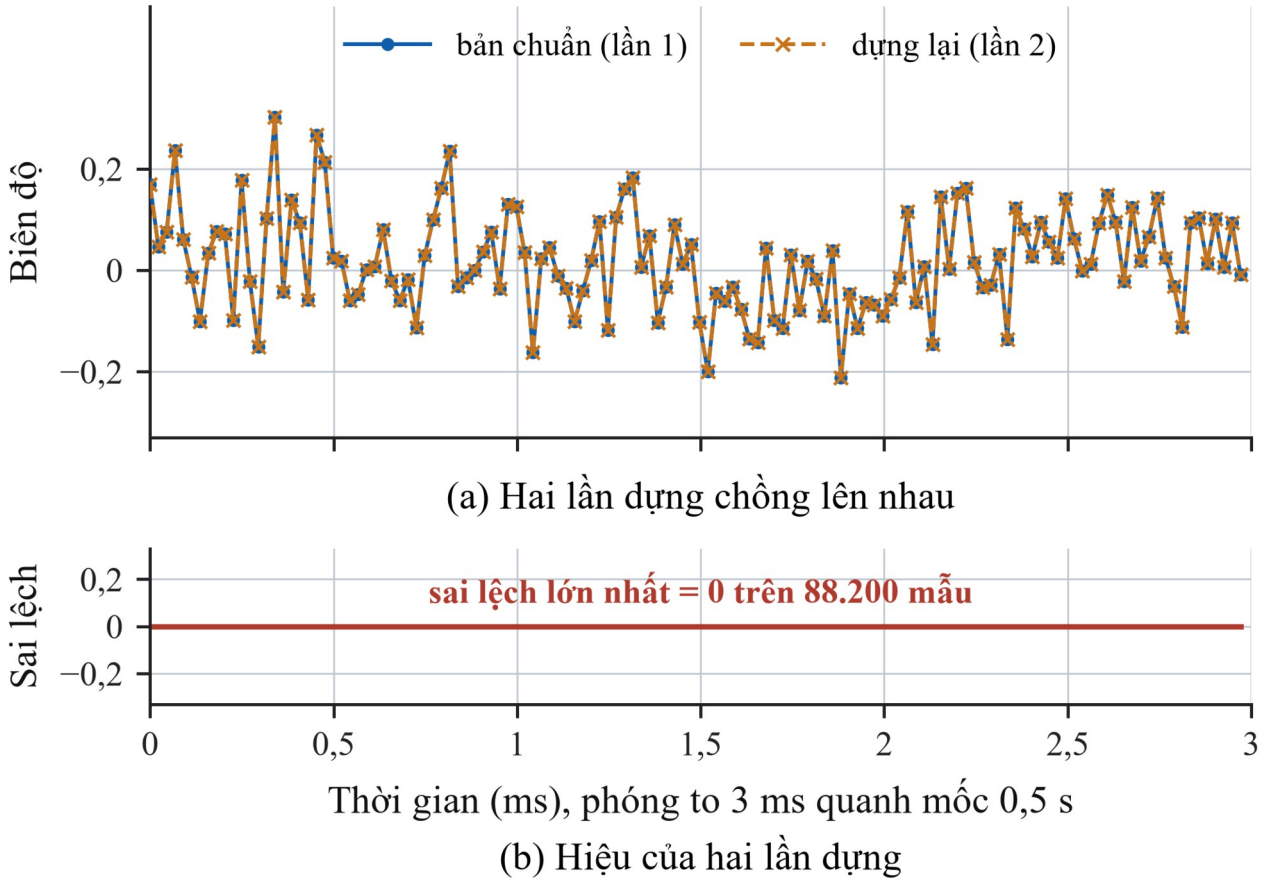
MỤC 12.8

Kiểm và đo — biết lần sửa vừa rồi không làm sai gì

- OfflineAudioContext cho kết quả lặp lại được, nên kiểm tự động khả thi.
- Ba mức kiểm từ chặt tới lỏng.
- Ba số đo trong lúc chạy và bảng kiểm tám dòng trước khi giao.

Dựng hai lần cùng một đồ thị cho cùng kết quả

- Kiểm âm thanh khó vì phụ thuộc đồng hồ, tải máy và thiết bị ra.
- OfflineAudioContext dựng nhanh hơn thời gian thực, không phụ thuộc thiết bị.
- Sai lệch lớn nhất bằng 0 trên 88.200 mẫu, từng bit một.



Hình 12.24 — Dựng hai lần cùng một đồ thị, và sai lệch giữa hai lần

Dựng lại, so với tệp chuẩn, đòi sai lệch nhỏ hơn ε

$$\max_n \left| y_{\text{mới}}[n] - y_{\text{chuẩn}}[n] \right| < \varepsilon \tag{12.12}$$

- ε ở đây lấy bằng đúng không.
- Không phải rất nhỏ mà bằng không: từng bit trùng nhau qua hai lần dựng.

Một phép kiểm hồi quy đầy đủ, chạy trong vòng kiểm tự động

```
import { OfflineAudioContext } from "node-web-audio-api";
import { readFileSync } from "node:fs";

async function dungCanh() {
  const c = new OfflineAudioContext(2, 44100 * 2, 44100);
  const may = new MayAmThanh(c);      // CÙNG mô-đun, khác context
  await may.moMay();
  may.phat("trong", { muc: 0.8, gui: 0.3 });
  may.dat("amLuongNhac", 0.6);
  return c.startRendering();
}

const b = await dungCanh();
const moi = b.getChannelData(0);
const chuan = new Float32Array(readFileSync("chuan.f32").buffer);

let lech = 0;
for (let i = 0; i < moi.length; i++)
  lech = Math.max(lech, Math.abs(moi[i] - chuan[i]));

if (lech > 0) { console.error("LỆCH", lech); process.exit(1); }
```

(rút gọn: bỏ lệnh ghi tệp)

Ba mức kiểm và khi nào dùng mức nào

Mức	Đòi hỏi	Dùng khi	Điểm dễ sai
Từng bit	$\varepsilon = 0$	Đổi tên biến, tách hàm, dọn mã	Đổi phiên bản trình duyệt cũng đỏ, và đỏ đúng
Theo ngưỡng	$\varepsilon = 10^{-6}$ tới 10^{-4}	Đổi thứ tự phép cộng, kiểu số, chuyển sang worklet	Ngưỡng quá lỏng thì lỗi thật lọt qua
Theo chỉ số	Đỉnh, LUFS, phổ lệch dưới ngưỡng	Đổi thuật toán có chủ ý	Hai tiếng khác hẳn vẫn có thể cùng LUFS

Lưu tệp chuẩn dạng float32 thô, và đừng nới ngưỡng

Mã hoá có tổn hao làm vô hiệu phép so từng bit. Lưu kèm tệp JSON ghi số kênh, tần số lấy mẫu và phiên bản thư viện. Khi cố ý đổi thuật toán, hãy nghe bản mới, xác nhận đúng rồi thay tệp chuẩn; nới ngưỡng làm phép kiểm mất giá trị.

Ba số đo trong lúc chạy, ngưỡng và việc phải làm

Số đo	Đọc từ đâu	Ngưỡng đáng lo	Việc phải làm
Tải luồng âm thanh	ctx.renderCapacity	Trên 70 % kéo dài	Giảm tải theo bảng mục 12.5.4
Số giọng đang sống	Kích thước Map của máy	Không về 0 sau khi im tiếng	Tìm chỗ thiếu onended
Số node tạo mỗi giây	Một biến đếm tự tăng	Trên 116 node trong một khung	Dựng sẵn và tái sử dụng

Ba số này nên hiện ngay trên màn hình bản dựng dành cho lập trình viên, cạnh số khung hình mỗi giây.

Tám dòng phải qua trước khi giao một ứng dụng Web Audio



Hình 12.25 — Tám dòng phải qua trước khi giao một ứng dụng Web Audio

Bảy bài tập cuối chương

- **Bài 1–2:** đo lại tỉ số bus vang; so ba cách bypass (gain 0, disconnect, mờ chéo rồi disconnect).
- **Bài 3–4:** kiểm cancelAndHoldAtTime trên trình duyệt của mình; cài luật wet/dry thứ ba và đo độ gợn.
- **Bài 5–6:** ducking theo đường bao so với 8,43 dB; tìm ba chỗ rò node bằng cách đếm Map.
- **Bài 7:** áp bảng kiểm tám dòng vào bài tập lớn của nhóm, nộp kèm ít nhất hai phép đo.

Chương 12 gói lại trong sáu con số



Trong Web Audio, thứ đắt nhất không phải là phép tính, mà là những node bị bỏ sót.

Chương 13 rẽ sang hướng khác hẳn: mô hình AI tạo sinh nhạc và giọng. Nhưng mọi thứ dựng ra ở đó rồi cũng phải chạy trong một đồ thị như đồ thị của chương này.

Hình 12.26 — Chương 12 gói lại trong sáu con số

Những điều cần nhớ

1. Mọi thứ đổi trên đường tín hiệu phải đổi có hình dạng: ramp thay vì bước nhảy, sạch hơn 71 dB.
2. Mỗi lần lập lịch: huỷ, neo, ramp; thời điểm hẹn luôn cộng khoảng an toàn.
3. Bypass thật cần mờ chéo rồi ngắt nối; gain về 0 không tiết kiệm gì (519 so với 517 ms).
4. Luật wet/dry đẳng công suất giữ mức (gợn 0,03 dB); ducking đưa lời từ −2,62 lên +8,43 dB.
5. Gộp khối tuyến tính và dùng bus gửi vang: rẻ hơn tới 16,2 lần; trần đa âm có cái giá là đỉnh.
6. Node bị bỏ sót làm chậm 1,77 lần; phát hiện bằng đếm Map; kiểm hồi quy đúng từng bit.