Jak tworzyć muzykę na komputerze (I)

Jerzy Karczmarczuk
Zakład Informatyki, Uniwersytet w Caen, Francja

1. Wstęp

Jest to pierwsza część z dwuczęściowego artykułu poświęconego akustyce komputerowej. Nie będziemy omawiać aspektów muzycznych, nawet niewiele będzie o nutach. Interesuje nas *generowanie dźwięków*, sygnałów o charakterze muzycznym, oraz komputerowa symulacja instrumentów muzycznych (wraz z otoczeniem, np. generacja sztucznego pogłosu).

Ta dziedzina należy do fizyki (lub do matematyki stosowanej), co wcale nie oznacza, że zlekceważymy „ludzkie" aspekty dźwięku. Fizyczny sens dźwięków oprócz aspektów naukowych i kulturowych posiada także ewidentną stronę finansową. Strumień nowych patentów dotyczących syntetyzatorów[1], modułów do efektów specjalnych itp. trwa nieprzerwanie. Programów do generacji, „instrumentów wirtualnych" i programów muzycznych wspomagających kompozycję i symulujących orkiestry są już tysiące, choć pierwsze rozsądne syntezatory komercjalne oparte o modelowanie fizyczne powstały dopiero w połowie lat 90. Nasz tekst jest powierzchowny, jeśli chodzi o fizykę, aczkolwiek miejscami wprowadza pewne techniki programowania, które są rzadko uczone. Od Czytelnika oczekujemy trochę wiedzy o falach i drganiach, pewnego pojęcia o programowaniu i słuchu. Choć tzw. drewniane ucho nie przeszkodzi w zrozumieniu tekstu, przykłady akustyczne, które towarzyszą artykułowi, nie będą zbyt przekonujące... Muzyki nie można oddzielić ani od kultury, ani od fizjologii czy psychologii nawet jeśli interesuje nas głównie fizyka dźwięku. Pewne algorytmy kompresji (telefon cyfrowy, pliki MP3) są oparte na fizjologii słuchu, korzysta się z faktu, że nasza rozdzielczość widmowa i czasowa jest ograniczona.

Słyszalne drgania akustyczne mieszczą się z grubsza między częstotliwościami 20 Hz i 16 kHz. Normalna muzyka nie wykorzystuje całego widma, drgania poniżej 40 Hz (częstotliwość prądu zmiennego w sieci to 50 Hz) są „burczące" i rzadko występują samodzielnie, a ostre piski powyżej 4 kHz także są rzadkie. Wzorcową częstotliwością (klasyczny kamerton) jest A4: 440 Hz, klawisz 49, nieco na prawo od połowy klawiatury fortepianu. Częstotliwości podstawowe nut na klawiaturze wahają się od 27,5 Hz do 4186 Hz. (W tym tekście ignorujemy wszystkie stroje oprócz klasycznego, równomiernie temperowanego). Tony skrzypiec wahają się z grubsza od 196 Hz do 4,4 kHz. Normalny głos ludzki (mowa) jest dość niski.

[1] Urządzenia do generacji dźwięku słowniki nazywają „syntezatorami", co jest krótsze, ale chyba niespecjalnie ładne.

Dla dorosłego mężczyzny[2] od ok. 85 do 200 Hz i od 160 do ok. 260 Hz dla typowej kobiety, ale śpiew brzmi wyżej. Sopran mieści się między 240 a 1170 Hz, choć w dwóch znanych ariach Królowej Nocy z *Czarodziejskiego Fletu* Mozarta można usłyszeć nutę F6 – to jest prawie 1400 Hz i większość Czytelniczek *Fotonu* będzie z tym miała hmm... niejakie trudności.

Komputer domowy o częstotliwości zegara ponad 1,5 GHz potrafi programowo generować sygnały akustyczne i procesor będzie miał jeszcze dużo czasu, aby skomponować wiele składowych i dokonać skomplikowanego filtrowania, a nawet kompresji OGG (Vorbis) czy MP3 na bieżąco. Należy jednak wziąć pod uwagę szereg koniecznych aspektów.

1. Komputer to nie jest wieża HiFi! Głośniki w przeciętnym laptopie są bardzo słabe, i – typowo – dźwięków poniżej 200 Hz nie słychać. Polecamy kupno, za parędziesiąt złotych, porządniejszych wzmacnianych głośników, z widmem pozwalającym zejść do 80 Hz, albo niżej.
2. Czytelnik powinien dysponować oprogramowaniem do eksperymentów. Nie tylko odtwarzaczami muzyki, jak np. Winamp lub VLC pod Windows, czy XMMS lub Xine pod Linuxem. Dla nas ważne będzie oprogramowanie podstawowe, pozwalające sterować procesorem dźwiękowym z wnętrza programu. Tutaj wykorzystaliśmy język Python oraz moduł PyAudio. (Ten ostatni jest „blokujący”; gdy porcja danych jest przesyłana do procesora dźwiękowego i odtwarzana, program jest zawieszany. Są i inne biblioteki, ale trudniejsze w użyciu, a nas nie interesuje generacja dźwięku w czasie rzeczywistym (współbieżnie z programem, który „pisze” tę muzykę).
3. Języki interpretowane, wysokopoziomowe, jak Python, zwykle wystarczą do doświadczeń, ale *szybkie*, niskopoziomowe przetwarzanie sygnałów w czasie rzeczywistym jest w tych językach trudne, nawet jeśli biblioteki nie blokują programów w czasie pracy procesora dźwiękowego. Należy więc dobrze poznać język i nauczyć się optymalizować programy (np. w krytycznych fragmentach należy unikać pętli w Pythonie, o ile można je zastąpić iteracjami całych tablic). Jednak w tym artykule nie przejmujemy się optymalizacją, ważna dla nas jest przejrzystość algorytmiczna.

Polecamy skorzystać z naszego foldera **http://users.info.unicaen.fr/~karczma/Foton/Sounds/** na uniwersytecie w Caen (Normandia, Francja). Zawiera on ilustracje dźwiękowe do niniejszego tekstu w formacie `.ogg`, który jest czytany przez standardowe programy odtwarzające (VLC, Winamp), a także przez przeglądarkę Firefox. Folder zawiera m.in. 13-sekundowy plik `freqs.ogg`. Zawiera on próbki dźwięków od 330 do 30 Hz co 20 Hz. Amplituda wzrasta ze spadającą częstotliwością, jest równa $50/f$, ale akustycznie ma się wrażenie, że

[2] Są wyjątki… Bas operowy potrafi zejść niżej (czy Czytelnik zna arię Skołuby ze *Strasznego Dworu*? Nie? Zażyj tabaki!), a gdy znamienity fizyk krakowski, prof. Marian Mięsowicz, burczał pod nosem na swoim cotygodniowym seminarium, bo mu się referat średnio podobał, miałem wrażenie, że schodzi do częstotliwości infradźwięków.

dźwięk cichnie. Ta próbka posłuży Czytelnikowi do sprawdzenia jakości jego głośników, ostatnie próbki zapewne staną się niesłyszalne.

2. Trochę teorii

2.1. Próbkowanie

Dźwięk będzie dla nas ciągiem liczb reprezentującym dyskretną amplitudę sygnału. Nie zajmujemy się stereofonią, dźwięk ma tylko jeden kanał. W programach, ze względu na dokładność obliczeń używamy liczb zmiennoprzecinkowych, o wartościach rzędu jedności. Podczas kodowania plików **`.wav`**, – standardowy format pod Windows – należy zapisać je jako liczby całkowite na 15 bitach, więc przed konwersją pomnożyć np. przez 32 768. Zarówno programy, jak i sterowniki procesorów dźwięku muszą wiedzieć, z jaką częstotliwością dokonaliśmy próbkowania, tj. jakiemu interwałowi czasowemu odpowiada N próbek sygnału. To, co się dzieje z numerycznym sygnałem po przesłaniu go do bufora karty dźwiękowej, tj. konwersja cyfrowo-analogowa, nie należy już do naszego artykułu. Należy przestrzegać zasady wynikającej z tzw. Shannona: częstotliwość próbkowania (*sampling ratio*) winna być przynajmniej dwa razy większa od najwyższej częstotliwości, którą chcemy wiernie oddać (tzw. częstotliwości Nyquista), bez wprowadzania zniekształceń (pojawiania się częstotliwości „duchów"). W praktyce stosuje się często 44 100 Hz i taką częstotliwość, zwaną dalej SR, stosujemy w naszych programach. Oznacza to, że kilka minut muzyki to jest kilka lub kilkanaście milionów próbek, komputer sobie z tym poradzi, ale przetwarzanie tak dużych tablic przez program może trwać dość długo.

2.2. Nuty i liczby

Każda kolejna oktawa podwaja częstotliwość tonu. W stroju równomiernie temperowanym stosunek częstotliwości dwóch kolejnych półtonów wynosi $\sqrt[12]{2} = 1{,}059463$ i wszystkie tonacje (osobno durowe i molowe) mają taki sam charakter. Nuta C w tej samej oktawie, poniżej wzorcowego tonu A4 będzie miała częstotliwość 261,62 Hz. W popularnym formacie kodowania muzyki tonalnej: MIDI, także stosuje się równomierną, logarytmiczną skalę częstotliwości, i konwencjonalnie częstotliwościom przypisuje się liczby

$$p = 69 + 12 \cdot \log_2 \frac{f}{440\ \mathrm{Hz}}, \tag{2.1}$$

czyli $f = 440 \cdot (\sqrt[12]{2})^{p-69}$. Częstotliwości powinny być generowane dość dokładnie. Nie jest to zupełnie banalny problem, gdyż dźwięki muzyczne nie są monochromatyczne, zawierają nie tylko częstotliwość podstawową, ale także harmoniki, pod-harmoniki oraz częstotliwości poboczne, które wzbogacają barwę instrumentu. Dla instrumentów idiofonicznych, takich jak dzwony rurowe, gongi itp., wysokość dźwięku określana nutą jest dość umowna. Jak napisać najprostszy

program generujący dźwięk? Aby otrzymać monochromatyczną sinusoidę o częstotliwości **f** w Pythonie przy użyciu biblioteki **numpy**, kodujemy:

```
From numpy import *
SR=44100    # Sampling Ratio, częstość próbkowania.
trw=0.5     # Czas trwania w sekundach (przykładowo).
t=linspace(0,trw,SR*trw)
y=sin(2*pi*f*t)
```

gdzie **linspace** jest generatorem tablicy przedstawiającej czas, o wartościach od zera do czasu trwania. Trzecim parametrem jest liczba próbek. Tablicę **y** należy przesłać do urządzenia wyjściowego. **Ale nie będziemy tak programować**, ta technika generowania cyfrowej fali jest niefizyczna. Proste układy fizyczne nie mają pamięci i nie mierzą czasu. Rzeczywisty instrument nie ma zegarka. Wprowadzenie liczbowej osi czasowej jest formalnym artefaktem i tylko na pozór można powiedzieć sobie „no i co z tego". Zauważmy, że dla 440 Hz po minucie będziemy obliczać sinus niezbyt „fizycznej" liczby, przekraczającej 165 800. Nie tylko jest to mało sensowne, ale wprowadzimy także (niewielkie) błędy rachunkowe wynikające z redukcji trygonometrycznej argumentu i błędów zaokrągleń. Możemy w tekście zapisywać formułę dla monochromatycznej fali przy użyciu funkcji sinus, ale jej programowa generacja będzie używać odmiennych algorytmów nawet jeśli będzie to wolniejsze. Dla bardziej skomplikowanych dźwięków nie ma żadnych wzorów.

3. Proste generatory i przetworniki sygnałów

Głównym celem całego artykułu jest przedstawienie metod modelowania fizycznego *skomplikowanych* generatorów i przetwarzaczy dźwięku, ale o tym później. Trzeba zacząć od prostych oscylatorów.

3.1. Oscylatory i fale monochromatyczne

Okazuje się, że aby generować sinusoidę ekonomicznie i bez mierzenia czasu, bardzo pomocna będzie trójczłonowa relacja rekurencyjna wynikająca ze wzoru na sinus sumy argumentów. Niech $x = \omega t$, a $h = \omega\Delta t = \omega/\mathrm{SR}$, gdzie ω jest częstotliwością kołową: $\omega = 2\pi f$, a Δt jest odstępem czasowym między momentami próbkowania, tj. $\Delta t = 1/\mathrm{SR}$. Mamy:

$$\begin{aligned} \sin(x+h) &= \sin x \cos h \ + \cos x \sin h, \\ \sin(x-h) &= \sin x \cos h \ - \cos x \sin h. \end{aligned} \tag{3.1}$$

Stąd po dodaniu otrzymamy

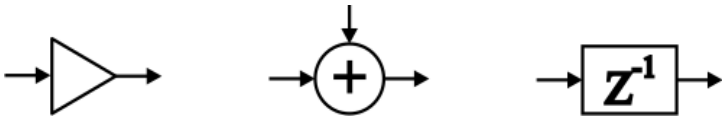
$$\sin(x+h) = 2\sin x \cos h \ - \sin(x-h). \tag{3.2}$$

Wystarczy obliczyć trzy wartości: $\sin(x_0)$, $\sin(\mathrm{x}_0 + h)$, oraz $\cos h$, czyli dwa warunki początkowe i parametr określający częstotliwość. Stąd potrafimy

obliczyć $\sin(x_0 + 2h)$ itd. Ogólnie $\sin(x_0 + nh) = y_n = c \cdot y_{n-1} - y_{n-2}$, gdzie $c = 2\cos h$. Rekurencja jest stabilna i po wielu dziesiątkach sekund (czyli setkach tysięcy próbek), odchylenie od rzeczywistej wartości nie przekroczy kilku procent (słuchowo jest to niezauważalne, gdyż jest to głównie przesunięcie fazy). Zaprogramować tę metodę można na wiele sposobów, np. obliczać w pętli `y[n]=c*y[n-1]-y[n-2]`, ale później pojawią się komplikacje, które sugerują inne schematy organizacji programu. Skonstruujmy jeszcze jeden model, będzie to program na numeryczne rozwiązanie równania różniczkowego oscylatora harmonicznego $\ddot{y}(t) = -\omega^2 y$. Można go sprowadzić do dwóch równań pierwszego rzędu, na parę (y, v): $\dot{y} = \omega v\,;\ \dot{v} = -\omega y$. Rozwiązać je można np zmodyfikowaną, stabilną metodą Eulera:

$$\begin{aligned} y_{n+1} &= y_n + h \cdot v_n; \\ v_{n+1} &= v_n - h \cdot y_{n+1}. \end{aligned} \tag{3.3}$$

Warunki początkowe, to $y_0 = \sin(\omega t_0)$, $v_0 = \cos(\omega t_0)$, np. 0 i 1 dla $t_0 = 0$. Ta metoda jest nieco mniej dokładna od poprzedniej. Program obliczający ten ciąg można napisać na wiele sposobów. Chcemy tutaj zaproponować pewien ogólny schemat kodowania ciągów dynamicznych, dość popularny w dziedzinie symulacji. Zamiast zaczynać od tablic i pętli, wyobrażamy sobie całą strukturę jako „urządzenie", które emituje *sygnał* numeryczny, próbkę po próbce. Pewne elementy tylko emitują, inne przetwarzają (mają wejście i wyjście), jeszcze inne kombinują kilka strumieni sygnałów. Wszystko można przedstawić diagramatycznie, oto kilka cegiełek do budowy większych „obwodów":

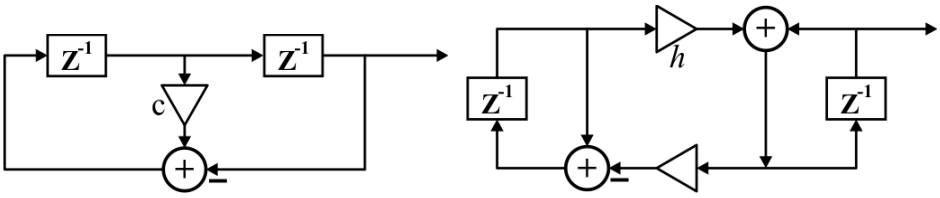


Rys. 1. Elementy graficzne przetworników sygnałów

Ogólny blok („pudełko") niekoniecznie jest zastosowaniem funkcji do każdego elementu: $y_n = f(x_n)$. Blok oznaczany jako Z^{-1} jest elementem opóźniającym, gdy próbka y_n doń wchodzi, na wyjściu dostaniemy wartość poprzednią, y_{n-1}. (Ten blok musi być sparametryzowany wartością początkową, emitowaną, gdy blok otrzymuje pierwszą próbkę). Trójkąt jest wzmacniaczem, mnoży sygnał przez stałą. Operatory binarne mogą być dowolne. W ten sposób, dwa diagramy, które emitują sinusoidy według dwóch powyższych algorytmów, będą miały postać na rys. 2. Instrukcja przypisująca blok pewnej zmiennej, np. `a=Blok(...)` pozwala traktować `a` jako strukturę przetwarzającą, ale także jako sygnał produkowany przez ten blok.

Pewne pakiety do programowania naukowego, np. Simulink, LabView, czy Xcos, pozwalają „kodować graficznie": zbliżone diagramy **są** programami na-

rysowanymi przez użytkownika, kompilator automatycznie przekształca te grafy w wykonywalne programy.

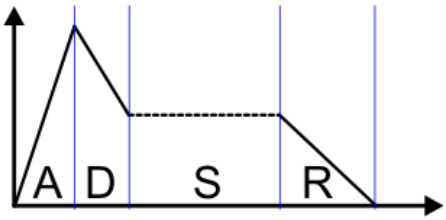


Rys. 2. Generatory fal monochromatycznych

Rozdział 4 poświęcimy tzw. technikom programowania strumieniowego. Będzie to koncentrat informacji dotyczącej kodowania diagramów jak te powyżej. W lewym wartości początkowe bloków opóźniających to zero i $\sin h$, a w prawym: zero i jeden. Które wartości odpowiadają którym blokom?

3.2. Modulowanie amplitudy (obwiednie) i częstotliwości

Bloki wzmacniające (trójkąty) mnożą amplitudę sygnału przez stałą. Jeśli zastąpimy ten blok przez operator mnożenia dwóch sygnałów, gdzie drugi ma znacznie wolniejszy przebieg, dostaniemy amplitudę pierwszego zmienną w czasie, co jest konieczne dla reprezentacji rzeczywistych dźwięków. Dla strun uderzanych, lub szarpanych jest to zasadnicze, (dla wysokich tonów fortepianu słychać głównie początkowe uderzenie...) bez obwiedni nie da się powiedzieć, jaki to instrument. Okresowa zmiana amplitudy dźwięku stanowi efekt *tremolo.*



Rys. 3. Obwiednia ADSR

Często w przypadku uderzanej struny generującej krótki dźwięk (zwłaszcza dla instrumentów klawiszowych), strukturę obwiedni redukuje się do tzw. schematu ADSR (*Attack – Decay – Sustain – Release*), narastanie, opadanie, utrzymanie i wybrzmiewanie. Prawdziwe obwiednie są bardziej skomplikowane, rys. 4 przedstawia początek tonu pianina.

Rys. 4. Obwiednia dźwięku pianina

Obwiednie instrumentów eksperymentalnych są często dowolne, nic nie ogranicza wyobraźni twórców (czasami wytrzymałość nerwowa słuchaczy...). W na-

szych generatorach stała mnożąca wpływa nie na amplitudę, lecz na częstotliwość. W ten sposób – zamieniając ten blok na operator mnożenia przez sygnał modulujący – możemy osiągnąć efekt zmiany wysokości tonu, *vibrato,* ale tylko dla monochromatycznej fali jest to takie proste (proszę posłuchać próbki **`monovib.ogg`**). Vibrato dla innych instrumentów, np. fletu, czy skrzypiec, wymaga zmian innych parametrów, a symulować je można przez dynamiczne „rozciąganie" i skracanie czasu, tj. nieliniowe powtórne próbkowanie, które zostanie omówione później.

4. Przetwarzanie strumieni danych

W tym rozdziale fizyki jest niewiele, omówimy tu pobieżnie pewną technikę programowania. Jest ona jednak niezwykle użyteczna w programach symulujących układy fizyczne i być może kiedyś do niej wrócimy. Nasze pierwsze doświadczenia polegały na wypełnieniu tablicy zawierającej amplitudę sygnału. Ale diagramy naszych oscylatorów, jak wspomnieliśmy, sugerują dynamikę czasową, zamiast mówić o tablicach danych będziemy operowali **strumieniami** – ciągami danych (próbek dźwięku) w czasie.

Przypuśćmy, że blok A ma dostarczać sygnał do bloku B, który go pośle dalej. Bloki w programie są to *grupy danych*: parametrów, odnośników do innych bloków (źródeł), akumulowanych próbek otrzymanych sygnałów, itp. Blok posiada szereg procedur, które można wywołać, np. **`A.funkcja(x)`**. Zorganizowanie kodu programu, który to zrealizuje może wykorzystać wiele różnych podejść i one wszystkie są wykorzystywane w przetwarzaniu sygnałów w różnych pakietach i bibliotekach. Przeciętny użytkownik na ogół nie wie jak te programy funkcjonują, ale nasz artykuł jest napisany dla ambitnych majsterkowiczów. Oto dwa alternatywne schematy organizacji.

1. Programowanie pasywne. Blok B wywołuje blok A, np. procedura czytająca dane z A wykonuje instrukcję **`x=A.next()`**, co przypisuje **`x`** nową próbkę. Jeśli A ma ją do dyspozycji (stała, albo wartość początkowa bloku opóźniającego), akcja się kończy, jeśli blok A przetwarza dane ze swoich źródeł, np. C, jego funkcja **`next`** musi wywołać **`C.next()`** i te wywołania się propagują aż do jakiegoś autonomicznego źródła. Ta metoda programowania jest „niefizyczna", proces przebiega jak gdyby „w tył w czasie", ale jest ekonomiczna i dobrze dostosowana do sporej klasy języków programowania. Tu prosto widać kombinację dwóch sygnałów, np. ich dodanie. Blok arytmetyczny po prostu żąda danych od dwóch źródeł. Program startuje od końca, od modułu, który ma otrzymać całkowity sygnał i go zapisać na dysku, zagrać, albo wyrysować.
2. Programowanie aktywne, albo „pchanie danych". Każdy blok winien wiedzieć, co ma się dziać *dalej*. Blok A (tj. procedura odpowiedzialna za kontynuację procesu) dysponujący próbką **`x`**, wykonuje **`B.send(x)`**. Następuje

wywołanie procedury czytającej dane, w bloku następnym. Tutaj proces jest bardziej „fizyczny", przypomina rzeczywisty przebieg sygnałów w obwodzie elektronicznym, lepiej widać co się dzieje, gdy źródło przesyła dane do różnych odbiorników (wystarczy kilka razy wywołać **send**), ale kombinacja sygnałów może być mniej czytelna i zaprogramowanie całości wcale nie jest łatwiejsze. Aktywacja następuje od źródeł.

Pierwszy schemat jest czytelniejszy, gdy jeden końcowy wynik pobiera informację z wielu źródeł, drugi, gdy źródło generuje kilka różnych strumieni wynikowych. Nie ma metody idealnej. W obu przypadkach będą kłopoty w przypadku sprzężeń zwrotnych (pętli; wszystkie instrumenty muzyczne je mają...), przy konieczności zapewnienia synchronizacji „równoległych sygnałów", przy warunkowym, decyzyjnym sterowaniu i alternatywnym wyborze źródeł bądź celów itd. Z tego względu techniki strumieniowe (*dataflow*) są rzadko uczone nawet na studiach informatycznych, mimo, że ich historia jest dość długa... Nie jest to głównym celem naszego artykułu.

Program generujący sinusoidę według relacji rekurencyjnej jest dłuższy niż metoda bezpośrednia (i znacznie wolniejszy), ale stanowi punkt wyjścia do programowania innych instrumentów. Przy użyciu naszego pakietu wykorzystującego technikę pasywną będzie on miał poniższy kształt. Pierwszym argumentem bloku opóźniającego jest wartość początkowa. Drugim (albo czasami jedynym) argumentem każdego bloku jest jego źródło. Procedura **ssplit** rozdziela strumień na dwa, a operator **a>>b** przypisuje blokowi **b** blok **a** jako jego źródło. Nie sądzimy że poniższy program zmęczy Czytelników.

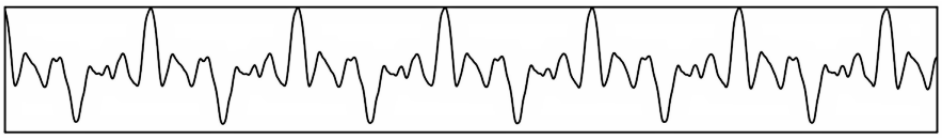
```
from conduits import *   # To jest nasz moduł
b=Delay(sin(h))          # Opóźnienie. Źródło jeszcze
nieznane
d,p = ssplit(b)       # Sklonowanie sygnału (2 kopie)
wynik,f = ssplit(Delay(0.0,d))
(2*cos(h)*p - f) >> b    # b dostaje swoje źródło
```

Program traktuje strumienie jako obiekty arytmetyczne, które można dodawać, mnożyć przez inne i przez liczby (wzmacnianie). Zmienna **wynik** zawiera generator wynikowy. Wywołanie **wynik.next()** dostarcza następnej wartości, wystarczy teraz w pętli wywołać go wiele razy i wynik posłać do wydruku albo do karty dźwiękowej. Czytelnik zechce zakodować drugi diagram.

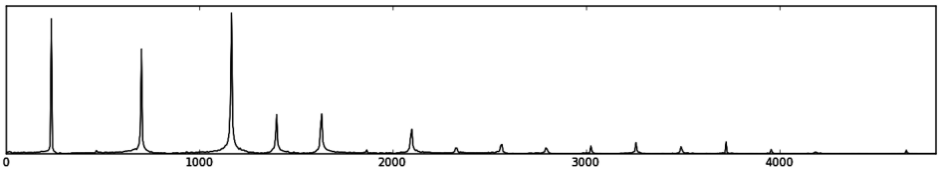
Uniwersalny pakiet przetwarzania strumieniowego winien dysponować innymi blokami, np. zastosowaniem dowolnej procedury do elementów strumienia (bez konieczności rozbijania jej na bloki), mechanizmy decyzyjne (warunkowe przesyłanie sygnału) oraz elementy interfejsu: pobieranie danych, wydruk, wykresy, wyjście audio itp. Nie będziemy się tym zajmować, choć pewne elementy będą nam później potrzebne, a w szczególności blok Z^{-L}: długa linia opóźniająca, symulująca przebieg fali przez ośrodek.

5. Synteza widmowa

Wracamy do metod generacji złożonych sygnałów dźwiękowych. Metoda zwana syntezą spektralną, lub techniką addytywną, polega na dodaniu do siebie pewnej liczby (kilku, kilkunastu...) składowych harmonicznych monochromatycznych, zgodnie z widmem, które zostało otrzymane z rzeczywistych instrumentów muzycznych. Rozkład częstotliwości instrumentu, jest jego najważniejszą charakterystyką, względnych faz różnych harmonik się nie słyszy. (Ale względne fazy zbliżonych częstotliwości mogą wpływać na dźwięk, w sumie i w różnicy sinusoid o *bardzo bliskich sobie* częstotliwościach, struktura dudnień będzie inna). Oto próbka amplitudy rzeczywistego dźwięku klarnetu (plik **`clarreal.ogg`**; fragment stacjonarny w czasie) o częstotliwości podstawowej 235 Hz i jego widmo (wartość bezwzględna jego transformaty Fouriera).



Rys. 5. Mała próbka dźwięku klarnetu



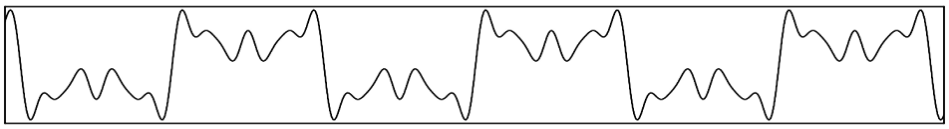
Rys. 6. Widmo klarnetu

widmo jest dość czyste, harmoniczne, a w zakresie niskich częstotliwości dominują składowe o *nieparzystych* wielokrotnościach częstotliwości podstawowej. Potem to się nieco psuje, ale poniższa formuła, z nieco odmiennymi amplitudami:

$$s(t) = \sin\omega t + 0.75\sin 3\omega t + 0.5\sin 5\omega t + 0.14\sin 7\omega t + 0.5\sin 9\omega t + 0.12\sin 11\omega t + 0.17\sin 13\omega t \tag{5.1}$$

daje sygnał dźwiękowy **`clarsim.ogg`**, który akustycznie jest bardzo podobny, mimo, że jego wykres jest dalece odmienny, gdyż względne fazy harmonik są inne niż w rzeczywistej próbce. Czytelnik może sobie wyobrazić ile prób i błędów jest konieczne, aby zrealizować bardziej skomplikowane instrumenty, np. saksofon, nawet jeśli dysponuje się rozkładami częstotliwości otrzymanymi przez analizę Fourierowską próbek. Ale i tu fizyka pomaga, np. sporo wiadomo o strukturze widma „masywnych" drgających obiektów, jak dzwony czy gongi.

Plik **gong0.ogg** symuluje w ten sposób dźwięk gongu (synteza zawiera minimalny wkład modulacji częstotliwości, co dodaje trochę dudnień).



Rys. 7. Synteza spektralna klarnetu

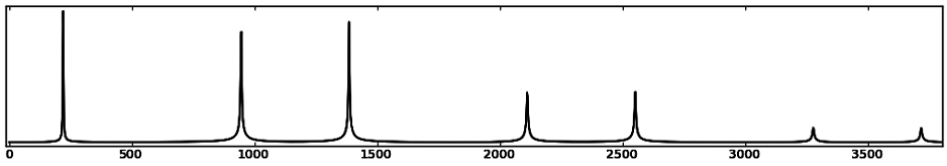
5.1. Synteza subtraktywna

Wypada tu wspomnieć o dwóch innych metodach, które jednak nie mogą teraz zostać szczegółowo omówione, gdyż wymagają znajomości technik filtrowania sygnału. W syntezie subtraktywnej generator początkowy produkuje falę dość skomplikowaną, np. złożenie fal trójkątnych lub prostokątnych. Następnie cała bateria filtrów modyfikuje widmo tej fali, osłabiając jedne a wzmacniając inne harmoniki, tak, by uzyskać dźwięk o pożądanej charakterystyce. Ta technika wymaga jeszcze więcej prób i dostrojeń niż synteza spektralna.

5.2. Metoda modulacji częstotliwości

Jest to prosta, ale wyrafinowana technika, w której monochromatyczną falę moduluje się *częstotliwościowo* innym sygnałem. W ten sposób można otrzymać efekt vibrato, ale tam sygnał modulujący ma częstotliwość rzędu 3–5 Hz. Jeśli jednak modulować sygnał podstawowy częstotliwością akustyczną, setkami Hz, to nie usłyszymy oscylacji częstotliwości i w ogóle z góry trudno sobie wyobrazić co usłyszymy, gdyż wynikowa fala będzie miała bardzo złożony kształt. Ta technika bywa wykorzystywana do dźwięków sztucznych, nieodpowiadających klasycznym instrumentom, lecz raczej syntezatorom eksperymentalnym, ale także do niektórych idiofonów.

W najprostszej wersji z dwoma generatorami, wystarczy wygenerować falę o kształcie $\sin(\omega_c t + d \cdot \sin(\omega_m t))$, gdzie ω_c jest częstotliwością „nośnika” (*carrier*), a ω_m – częstotliwością modulacyjną, która zwykle jest kilka razy większa od częstotliwości nośnika. Parametr d jest głębokością modulacji, która powinna być niewielka. Jeśli dodać zarówno do nośnika jak i do modulacji prostą obwiednię, wybrzmiewanie, to dostaniemy np. dźwięk w pliku **fmod0.ogg**, gdzie częstotliwość modulacji jest ok. 8 razy większa od częstotliwości nośnika. Widmo tego dźwięku (dla częstotliwości nośnika 220 Hz) przedstawiono na rys. 8.



Rys. 8. Modulacja częstotliwości; widmo „prostego” dźwięku

6. Modelowanie „fizyczne” instrumentów muzycznych

6.1. Fale i linie opóźniające

Niniejszy rozdział jest punktem wyjścia dla konstrukcji „prawdziwych” instrumentów. Zarówno drgająca struna, jak i słup powietrza w instrumencie dętym (aerofonie) jest ośrodkiem, przez który przebiega fala. W liniowym i jednowymiarowym przybliżeniu spełnia ono cząstkowe równanie różniczkowe

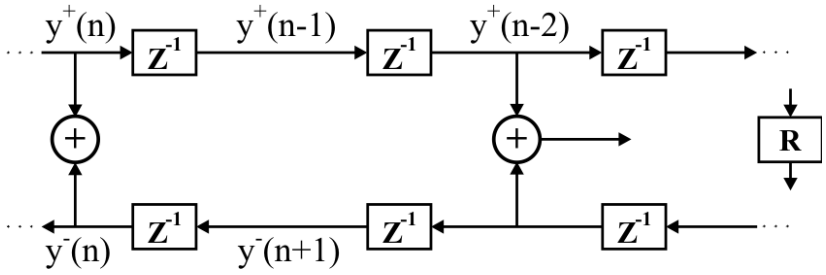
$$\frac{\partial^2 a}{\partial t^2} = c^2 \frac{\partial^2 a}{\partial x^2} \tag{6.1}$$

gdzie c jest prędkością fali, a a – amplitudą pewnej wielkości fizycznej: gęstości powietrza (lub ciśnienia), wychylenia podłużnego lub poprzecznego struny itp. Prędkość fali $c = \sqrt{K/\epsilon}$, gdzie K jest napięciem „struny”, a ϵ – liniową gęstością masy; w przypadku słupa powietrza, trzeba to przetłumaczyć na inne parametry. Ogólnym rozwiązaniem tego równania (d’Alembert, 1747), jest kombinacja fal biegnących w dwóch kierunkach: $y(x,t) = y^+(x-ct) + y^-(x+ct)$, gdzie y^+, y^- są dowolnymi funkcjami jednego argumentu. Kształt tych fal jest dowolny, czyli mogą być dowolnym złożeniem wielu fal monochromatycznych, ale warunki brzegowe dostarczą nam ograniczeń na dopuszczalne częstotliwości *w przypadku stacjonarnym*.

Przy brzegu: zaczepie struny, lub ścianie rury powietrznej (i przy wolnym ujściu do atmosfery), gdzie fala musi się odbić, amplituda, np. prędkość powietrza lub wychylenie struny winna się zerować, co dyskretyzuje dopuszczalne częstotliwości, możliwa jest pewna częstotliwość podstawowa odwrotnie proporcjonalna do długości ośrodka, oraz jej wielokrotności. Widmo częstotliwości jest dyskretne, proste. Przy odbiciu fali od zaczepu struny, lub od zamkniętej ściany rury, faza fali zmienia się na przeciwną (amplituda zmienia znak), a od „wolnego” końca – nie, faza „powraca”. Niestacjonarność, występowanie wytłumiania (zaczepy struny, palce gitarzysty itp. nie są sztywne), fakt, że np. flet ma inne otwory, itp., oraz inne zjawiska łagodzą te ograniczenia i widmo instrumentu jest bardziej skomplikowane. Jednak oscylujący element instrumentu można potraktować jako ***falowód*** o określonej długości. Przejście od fizyki do informatyki zaczyna się od dyskretyzacji czasu i przestrzeni. Przy odpowiednim doborze jednostek, reprezentantem będzie kombinacja wartości w tablicy **`y`** o indeksach **`y[n-m]`** i **`y[n+m]`**, gdzie jedna ze zmiennych oznacza odległość

przestrzenną (numer elementu), a druga – czas. W modelu przetwarzania strumieniowego linia opóźniająca to jest bufor – kolejka o określonej długości, z jednej strony dane wchodzą, z drugiej wychodzą. W każdym kroku czasowym wszystko się przesuwa od wejścia do wyjścia.

Ale każdy element struny traci energię na tarcie (zewnętrzne i wewnętrzne) i promieniuje energię akustyczną. Każdy zaczep i nieregularność wprowadzają zmiany widma (filtrują drgania) i wszystkie składowe fali się dodają. Możemy sobie wyobrazić strunę jako skomplikowany falowód, o strukturze przedstawionej na rys. 9. Opuściliśmy obecność wielu dodatkowych filtrów osłabiających dźwięk i wprowadzających dyspersję (zależność prędkości fali od częstotliwości), ale te filtry są wszędzie. Fizyka tego modelu jest trochę umowna.

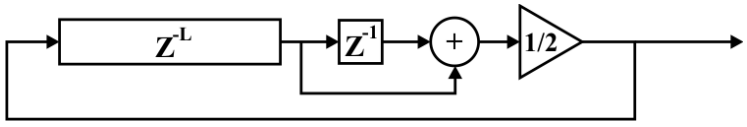


Rys. 9. Orientacyjny schemat falowodu – drgającej struny

W rzeczywistych układach dźwięk, który słyszymy jest tym fragmentem energii, który jest tracony, ale w modelu możemy wyprowadzić sygnał skądkolwiek. Blok R jest filtrem odpowiadającym za odbicie fali, z drugiej strony mamy także odbicie; diagram struny jest pętlą. *Podstawowym* uproszczeniem modelu jest zauważenie, że większość filtrów generuje liniowe kombinacje elementów sygnału wchodzącego. W tej sytuacji można je poprzestawiać i pogrupować, a pojedyncze opóźnienia skleić w jedną długą linię opóźniającą. Przejdźmy teraz do konkretnych, prostych przykładów.

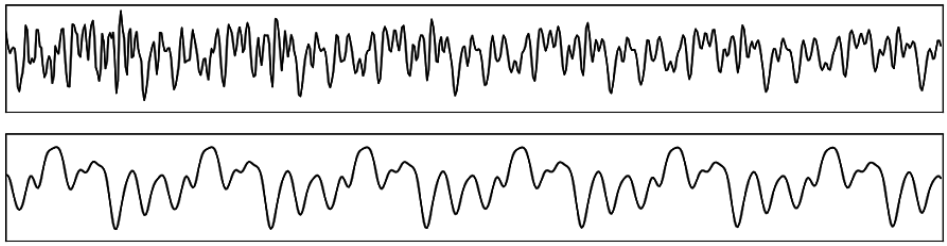
6.2. Prosta struna (Model Karplusa-Stronga)

Model struny sprowadzimy do jednej linii opóźniającej, oraz filtru, który pełni dwie funkcje: osłabia wyższe częstotliwości i wprowadza gaszącą obwiednię. Ten model jest przedstawiony na rys. 10. Filtrem jest średnia arytmetyczna dwóch kolejnych próbek sygnału, co wygładza sygnał. Długość linii to częstotliwość próbkowania podzielona przez częstotliwość tonu.

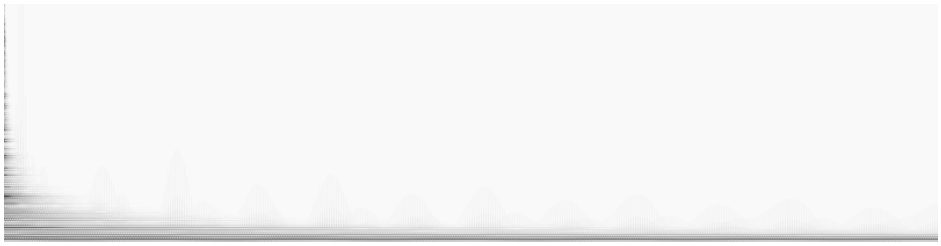


Rys. 10. Model Karplusa-Stronga

Linia opóźniająca zostaje początkowo wypełniona szumem, nieskorelowanymi liczbami losowymi. Biały szum „informatyczny” można wysłuchać w pliku **`wnoise.ogg`**. Zawiera on wszystkie częstotliwości (jego transformata Fouriera także wygląda jak szum), ale ponieważ wynik wygładzania jest z powrotem przesyłany do linii opóźniającej, kolejne iteracje szybko eliminują wysokie częstotliwości, i to co gaśnie najpóźniej, to składowe o okresie równym długości linii opóźniającej (i harmonik). Plik **`karpst0.ogg`** zawiera kilka próbek. Rys. 11 przedstawia dwie próbki amplitudy, od indeksu 200, oraz od indeksu 4000 dla częstotliwości 440 Hz (długość linii opóźniającej: 100). Dźwięk się „oczyszcza”.



Rys. 11. Dwie próbki amplitudy: wczesna i późniejsza, w modelu Karplusa-Stronga



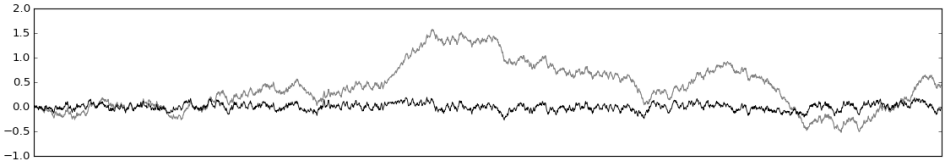
Rys. 12. Spektrogram struny Karplusa-Stronga

Na rys. 12 przedstawiamy widmo bieżące fragmentów o długości 1024 próbek (ok. 0,02 s) każdy, w funkcji czasu. Na osi poziomej mamy czas (do ok. 5 s), na osi pionowej – częstotliwość. Ciemniejsze obszary oznaczają większe natężenie. Widać, że w końcu zostają tylko niskie częstotliwości.

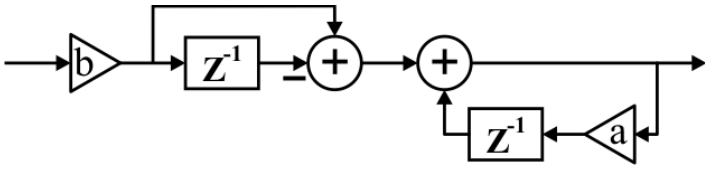
7. Filtry, wstępne wiadomości

Filtr jest to funkcją, która składa pewną liczbę próbek tak, aby zmienić jego widmo, wygasić jedne częstotliwości, wzmocnić inne. Konstruowanie różnic między sąsiednimi próbkami wzmacnia fluktuacje, natomiast dławi niskie częstotliwości (dwie identyczne próbki dają zero). Akumulacja lub średniowanie wygładza sygnał, eliminuje wysokie częstotliwości. Pierwszy model odpowiada w obwodzie elektrycznym kondensatorowi włączonemu szeregowo w obwód,

drugi – równolegle. Akumulacja jest *całkowaniem*, w najprostszej wersji wynik wyraża się formułą $y_n = y_{n-1} + x_n$, gdzie *x* jest sygnałem wejściowym. Oto model filtru, zwanego DC-blocker (DC: *direct current*, prąd stały), który usuwa bardzo niskie częstotliwości, natomiast zachowuje strukturę fluktuacji. Jest on potocznie używany w elektroakustyce, aby usunąć składowe stałe prądu przesyłanego do głośników, co zapobiega ich nasyceniu (słyszy się i tak tylko fluktuacje). Wynik filtrowania przedstawia rys. 13, a następny rysunek przedstawia sam filtr, jest on złożeniem obwodu różniczkującego z całkującym, ale to całkowanie jest ze stratami; stały wkład do amplitudy sygnału dość szybko zaniknie ze względu na to, że stała a jest mniejsza od jedności (na rys. 13: $a = 0.92; b = (1 + a)/2$).



Rys. 13. Działanie filtru DC-blocker



Rys. 14. Diagram filtru DC-blocker

Formuła dla tego filtru: $y(x)$ ma postać: $y_n = ay_{n-1} + b(x_n - x_{n-1})$. Najogólniejszy filtr liniowy można przedstawić w poniższej postaci:

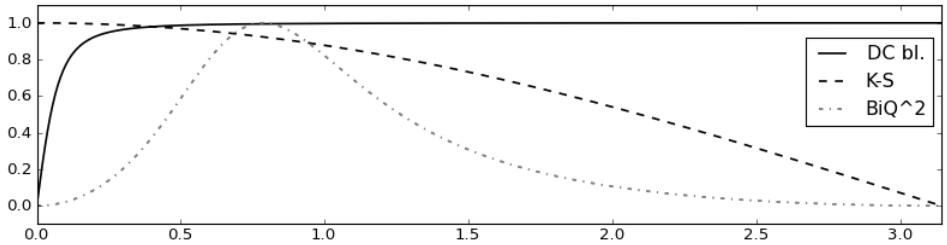
$$y_n = \sum_{k=0}^{N-1} b_k \cdot x_{n-k} - \sum_{k=1}^{M-1} a_k \cdot y_{n-k} \tag{7.1}$$

Oczywiście nie możemy tutaj rozwinąć teorii filtrów cyfrowych, ale dobrze wiedzieć, że wyrażenie

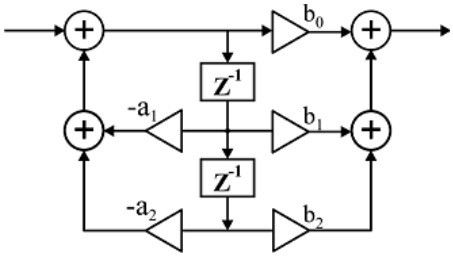
$$H(z) = \left|\frac{b_0 + b_1 z + b_2 z^2 + \cdots}{1 + a_1 z + a_2 z^2 + \cdots}\right|, \tag{7.2}$$

gdzie $z = \exp(if)$, przedstawia tzw. transmitancję, określającą osłabianie lub wzmacnianie amplitudy sygnału w funkcji częstotliwości. Umownie f zmienia się od zera do π, a górna granica odpowiada częstotliwości Nyquista. Rys. 15 przedstawia transmitancję filtrów $y_n = \frac{1}{2}(x_n + x_{n-1})$, DC-blockera z $a = 0{,}92$, oraz kwadratu tzw. pasmowego filtru bi-quad, z $H(z)$ mającego w liczniku

i mianowniku wielomiany stopnia 2. Są to popularne, proste filtry, które można składać (składanie filtrów jest wygodniejsze niż konstrukcja pojedynczych, ale skomplikowanych). W zależności od współczynników mogą one być także dolno-przepustowe, lub górno-przepustowe. Ich implementacja jest dość prosta, Czytelnik zechce sam narysować odpowiedni diagram. Wymieniając średnią arytmetyczną na bardziej skomplikowany filtr, dostaniemy inny dźwięk, np. łagodniejszy (gitara nylonowa), lub ostrzejszy (klawesyn, itp.) Istotną rolę będzie także odgrywać początkowe (lub późniejsze) wzbudzenie, zwykle łagodniejsze niż biały szum.



Rys. 15. Transmitacja trzech różnych filtrów



Rys. 16. Filtr bi-quad (uproszczony)

Bi-quad można zapisać jako $y_n = b_0x_n + b_1x_{n-1} + b_2x_{n-2} - a_1y_{n-1} - a_2y_{n-2}$, ale nie potrzeba go implementować w ten sposób, z czterema blokami opóźniającymi. Równoważną, a ekonomiczniejszą konstrukcją będzie $y_n = b_0w_n + b_1w_{n-1} + b_2w_{n-2}$, gdzie $w_n = x_n - a_1w_{n-1} - a_2w_{n-2}$.

8. Co dalej?

W drugiej części artykułu omówimy bardziej skomplikowane instrumenty strunowe, a także kilka modeli instrumentów dętych, oraz wprowadzimy filtry generujące pogłos, vibrato itp. W szczególności poznamy sposób symulowania dyspersji fal, ośrodka fizycznego, w którym prędkość zależy od częstotliwości, co jest niebagatelnym problemem w modelu linii opóźniających. Omówimy więc tzw. filtry pełnoprzepustowe, lub fazowe (*all-pass*), które nie obcinają widma, ale zniekształcają dźwięk wprowadzając dyspersję.

Obie części razem pokrywają prawdopodobnie 5% tej fascynującej tematyki, w której jest jeszcze sporo do zrobienia. Zapraszam do drugiej części artykułu.