Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Andrzej Karczyński
Nr albumu: 189403

Badanie algorytmów syntezy i przetwarzania dźwięku

Praca magisterska
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem
dr Janiny Mincer-Daszkiewicz
Instytut Informatyki

Sierpień 2005

Oświadczenie kierującego pracą

Oświadczam, że niniejsza praca została przygotowana pod moim kierunkiem i stwierdzam, że spełnia ona warunki do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora (autorów) pracy

Streszczenie

W pracy opisuję prototyp silnika syntezy dźwięku. Silnik umożliwia w łatwy sposób pisanie i łączenie ze sobą algorytmów przetwarzania dźwięku. Prototyp implementuje interfejs VST, dzięki czemu może działać jako wtyczka w macierzystych programach muzycznych. Napisane przeze mnie moduły umożliwiają stworzenie wirtualnego instrumentu muzycznego.

Słowa kluczowe

syntezator, synteza dźwięku, cyfrowe przetwarzanie sygnałów, filtr, modulacja amplitudy, modulacja częstotliwości, modelowanie fizyczne, aliasing

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

H. Information Systems
H.5 INFORMATION INTERFACES AND PRESENTATION
H.5.5 Sound and Music Computing

Spis treści

1. Wprowadzenie 7
1.1. Syntezatory dźwięku 7
1.2. Syntezatory i komputery 8
1.3. Prototyp silnika do syntezy dźwięku 9
1.4. Struktura pracy 9

2. Właściwości dźwięku 11
2.1. Czym jest dźwięk? 11
2.2. Dźwięk instrumentów 11
2.2.1. Instrumenty melodyczne 12
2.2.2. Instrumenty perkusyjne 12
2.3. Dźwięk w komputerze 12
2.4. Problemy 12
2.4.1. Aliasing 12
2.4.2. Efekt Gibbsa 16

3. Istniejące systemy syntezy dźwięku 19
3.1. Csound 19
3.2. Synthedit 20
3.3. Podsumowanie 21

4. Algorytmy syntezy 23
4.1. Tor syntezy 23
4.2. Generowanie sygnału 24
4.2.1. Oscylatory 25
4.2.2. Tablica fal 27
4.2.3. Szum i chaos 28
4.2.4. Modelowanie fizyczne 29
4.2.5. Generator konturu 30
4.3. Modulacja sygnału 31
4.3.1. Modulacja amplitudy 31
4.3.2. Modulacja częstotliwości 32
4.4. Przekształcenia liniowe 34
4.4.1. Filtry 34
4.4.2. Opóźnienie 36
4.4.3. Splot FFT 37
4.5. Przekształcenia nieliniowe 38
4.6. Zapobieganie efektowi aliasingu 39

5. Prototyp silnika syntezy dźwięku . . . 41
5.1. Założenia projektowe . . . 41
5.1.1. Wtyczka do aplikacji muzycznych . . . 41
5.1.2. Rozszerzalność . . . 42
5.1.3. Polifonia . . . 42
5.1.4. Szybkość . . . 42
5.1.5. Jakość . . . 42
5.2. Architektura . . . 43
5.2.1. Instrument . . . 43
5.2.2. Proces zarządzający polifonią . . . 45
5.2.3. Proces . . . 46
5.2.4. Bufor . . . 47
5.3. Procesy . . . 47
5.3.1. SineGenerator . . . 47
5.3.2. FastSineGenerator . . . 47
5.3.3. SawtoothGenerator . . . 47
5.3.4. AdditiveSawtoothGenerator . . . 47
5.3.5. WavetableSawtooth . . . 48
5.3.6. ADSREnvelopeGenerator . . . 48
5.3.7. KarplusStrongPluckedString . . . 48
5.3.8. DrawbarOrgan . . . 48
5.3.9. MultiGenerator . . . 49
5.3.10. Delay . . . 49
5.3.11. Reverb . . . 50
5.3.12. SinglePoleLPFilter . . . 50
5.3.13. FourStageLPFilter . . . 50
5.3.14. Peak4poleLPFilter . . . 50
5.3.15. Distortion . . . 51
5.3.16. ChebyshevPolynomialDistortion . . . 51
5.3.17. FFTConvolution . . . 52
5.3.18. Panning . . . 52
5.4. Możliwe rozszerzenia systemu . . . 52
5.4.1. Rozszerzenie parametrów . . . 52
5.4.2. Zapis ustawień . . . 52
5.4.3. Automatyzacja parametrów . . . 53
5.4.4. Interfejs użytkownika . . . 53
5.4.5. Tworzenie schematów syntezy . . . 53

6. Opis przykładów . . . 55
6.1. Sposób uruchamiania przykładów . . . 55
6.1.1. SAVIHost . . . 55
6.1.2. MiniHost . . . 55
6.2. Przykład 1 . . . 55
6.3. Przykład 2 . . . 56
6.4. Przykład 3 . . . 58
6.5. Przykład 4 . . . 59
6.6. Przykład 5 . . . 59
6.7. Przykład 6 . . . 60
6.8. Przykład 7 . . . 60

6.9. Przykład 8 61

7. Podsumowanie 63

A. Zawartość płyty 65

Bibliografia 67

Rozdział 1

Wprowadzenie

Przedmiotem pracy są wirtualne instrumenty, w których do wytworzenia dźwięku w celach muzycznych służą programy komputerowe. Modyfikując algorytmy tych programów konstruuje się różne instrumenty. Przedstawione techniki mają szerokie zastosowanie praktycznie we wszystkich gatunkach muzyki rozrywkowej.

Konwencjonalne instrumenty zajmują dużo miejsca, w szczególności dlatego, że aranżacje utworów muzycznych wymagają kilku, a często kilkunastu instrumentów. Dodatkowo gra na instrumentach jest rzeczą trudną, muzycy muszą doskonalić swoją technikę przez całe życie, a i tak z reguły potrafią się dobrze posługiwać tylko jednym instrumentem.

Syntezatory dźwięku ograniczają te trudności. Wszystkie instrumenty tak powstałe mogą być sterowane jedną klawiaturą muzyczną, dobrze znaną muzykowi. Dodatkowo instrumenty te mogą być sterowane przez komputer i muzyk może się skupić na kompozycji.

W ramach pracy powstał prototyp silnika wirtualnego instrumentu muzycznego, służącego do badań algorytmów syntezy dźwięku i łatwego tworzenia nowych instrumentów.

1.1. Syntezatory dźwięku

Klasyczne instrumenty wytwarzają dźwięk w sposób akustyczny. Dźwięk może powstawać poprzez drgania elementów mechanicznych. Przykładem mogą być instrumenty strunowe, w których drgania struny przenoszą się na drgania powietrza. Podobny proces występuje w instrumentach perkusyjnych, w których za wytwarzanie dźwięku odpowiedzialna jest membrana (w różnych rodzajach bębnów) lub ciała stałe, na przykład metalowe powierzchnie talerzy lub cymbałów. Trochę odmienne działanie mają instrumenty dęte. W nich dźwięk powstaje bezpośrednio z drgań słupa powietrza w ich wnętrzu. W procesie kształtowania dźwięku instrumentów akustycznych dość dużą rolę odgrywają też efekty rezonansu materiałów, z których są zbudowane i zamkniętego wewnątrz powietrza.

Syntezator dźwięku jest elektronicznym instrumentem muzycznym. Bezpośrednim wynikiem działania syntezatorów jest prąd zmienny, powstały przez działanie układów elektrycznych i elektronicznych. Do powstania dźwięku potrzebny jest jeszcze system nagłaśniający, który przekształca prąd w drgania powietrza w głośniku.

Właśnie w tym odmiennym charakterze kształtowania fali dźwiękowej tkwi cały potencjał syntezatorów. Układy elektryczne zdolne są do produkowania zupełnie innych fal, niespotykanych w naturalnym świecie. Dodatkowym atutem jest możliwość parametryzowania układów, dzięki czemu każdy użytkownik może ukształtować dźwięk według własnych potrzeb i zachować indywidualny charakter. Wykonawca może zmieniać parametry w czasie rzeczywistym, podczas grania, dzięki czemu syntezator płynnie zmienia brzmienie z jednego instrumentu na

inny.

Naturalną cechą układów elektrycznych jest także modularyzacja. Można wyodrębnić funkcjonalne części i łączyć je w dowolny sposób. Daje to nieograniczone możliwości brzmieniowe.

Okazało się, że rezultatem syntezy nie muszą być dźwięki rodem z science-fiction. Syntezatory są często wykorzystywane do naśladowania rzeczywistych instrumentów. W jednym urządzeniu możemy stworzyć coś w rodzaju trąbki, fletu, pianina, czy całej sekcji smyczkowej. Mimo, że wprawne ucho na pewno nie da się oszukać, brzmienia te na stałe weszły do naszej kultury.

Współcześnie w syntezie powstają także komputerowe modele fizycznych instrumentów, które obliczeniowo starają się oddać subtelności gry na swoich akustycznych odpowiednikach. Szczególnie udane wydają się być modele instrumentów strunowych. Wprawdzie nie zawsze wiernie oddają one rzeczywistość, lecz mimo to wiele wnoszą do bogactwa brzmień nam dostępnych.

1.2. Syntezatory i komputery

Do niedawna syntezatory istniały tylko jako urządzenia analogowe, tj. elektryczne. Miało to wiele wad. Układy elektryczne są drogie i zawodne. Zajmują dużo miejsca i nie są tak bardzo elastyczne, jak byśmy chcieli. Większość takich instrumentów była monofoniczna, czyli potrafiła grać jeden dźwięk jednocześnie. Skonstruowanie maszyny polifonicznej, grającej naraz wiele dźwięków, stwarzało problemy techniczne. Ponieważ dźwięki grane są jednocześnie, to muszą być syntezowane równolegle, zatem sprowadza się do zbudowania kilku takich samych układów syntezy, co zwielokrotnia koszty. Pierwsze modularne syntezatory analogowe, w których poszczególne moduły można było łączyć ze sobą w dowolny sposób, posiadały dużą liczbę przewodów, które użytkownik musiał przełączać, aby uzyskać żądany efekt. Wyglądało to jak stara centrala telefoniczna.

Współczesne syntezatory zbudowane są w technologii cyfrowej. Mały rozmiar, większe możliwości, ponieważ można w nich umieścić więcej, bardziej zróżnicowanych modułów, to tylko część ich zalet. Odznaczają się dużą polifonią, potrafią wydobywać wiele dźwięków jednocześnie. I to wszystko mniejszym kosztem.

Okazało się, że wśród wad syntezatorów analogowych były też zalety. Układy elektryczne, takie jak na przykład oscylatory, nie były zbyt stabilne i nie utrzymywały stałej amplitudy i częstotliwości pracy. Dawało to dodatkową głębię dźwiękowi, w przeciwieństwie do układów elektronicznych, które były zbyt idealne i grały sztucznie. Na szczęście tego typu niedociągnięcia można emulować dodając do algorytmów niewielki element losowości.

Kolejnym, naturalnym krokiem w rozwoju technologii wytwarzania dźwięku jest zastosowanie komputerów. W jednym urządzeniu możemy zgromadzić wiele instrumentów muzycznych i nie jesteśmy w żaden sposób ograniczeni. Ponieważ w środowisku komputerów instrument jest programem, zawsze można go poprawić lub wymienić na inny. Rozwojowi technologii sprzyja rozwój standardów muzycznych w komputerach. Twórcy oprogramowania nie muszą się martwić o obsługę klawiatur muzycznych. Zapewnia im to standard MIDI (zob. [MIDI]). Do obsługi kart muzycznych służy wieloplatformowy standard ASIO (zob. [ASIO]), który cechują bardzo niskie opóźnienia w zamianie sygnału cyfrowego na dźwięk.

Komputery są używane w profesjonalnych studiach muzycznych do rejestracji i przetwarzania dźwięków. Dla programów zajmujących się tym (będę je nazywał macierzystymi programami muzycznymi) stworzono standard wtyczek VST (zob. [VST]). Wtyczki mogą zajmować się dowolnym procesem wytwarzania lub przetwarzania dźwięku w programie. Daje

to nam możliwość zintegrowania całego środowiska pracy nad materiałem muzycznym w jednym programie i zestawie wtyczek. Interfejs VST jest dosyć prosty, dzięki czemu programista wtyczek zajmuje się tylko najważniejszymi sprawami, a cała praca spada na program macierzysty. Dlatego właśnie wtyczki stały się idealnym środowiskiem do tworzenia wirtualnych syntezatorów.

Niestety komputery nie są stabilnym środowiskiem, więc jeszcze nie wszędzie znajdują zastosowanie. W warunkach scenicznych, wykonywania na żywo, zawsze może coś pójść nie tak. Problemem może być opóźnienie komputera w przetwarzaniu dźwięku. Przy generowaniu dźwięku w czasie rzeczywistym dąży się do zminimalizowania opóźnienia. Z kolei krótszy czas przetwarzania może powodować przerwy w sygnale wyjściowym. Oprócz tego komputer może nie znieść trudów tras koncertowych.

Zaczęły się już pojawiać wolnostojące urządzenia służące tylko do uruchamiania wtyczek VST i moim zdaniem, w takim czy innym stopniu, wywrą wpływ na przemysł muzyczny. Na razie syntezatory programowe sprawdzają się głównie w warunkach domowych i studyjnych.

1.3. Prototyp silnika do syntezy dźwięku

W ramach pracy skonstruowałem prototyp silnika do syntezy dźwięku. Celem napisania silnika było ułatwienie implementacji i testowania algorytmów przetwarzających dźwięki, a także umożliwienie w łatwy sposób wykorzystania ich w praktyce. Twórca algorytmu musi dodać nowy moduł do systemu i udostępnić jego parametry. Później wystarczy tylko zbudować schemat z istniejących modułów i można już sprawdzić instrument w praktyce. System zajmuje się współpracą z programem muzycznym, odbiera komunikaty dotyczące klawiszy w klawiaturze muzycznej, zarządza parametrami, potrafi stworzyć polifonię poprzez duplikowanie odpowiednich modułów. Testowane algorytmy zostały napisane jako moduły do nowego silnika.

1.4. Struktura pracy

Praca składa się z następujących rozdziałów:

Rozdział 2. Właściwości dźwięku. W tym rozdziale opiszę fizyczne właściwości dźwięku.

Rozdział 3. Istniejące systemy syntezy dźwięku. W tym rozdziale przedstawię przykłady istniejących systemów syntezy dźwięku.

Rozdział 4. Algorytmy syntezy. W tym rozdziale przedstawię matematyczne podstawy niektórych algorytmów przetwarzania dźwięku.

Rozdział 5. Prototyp silnika syntezy dźwięku. W tym rozdziale przedstawię prototyp silnika syntezy dźwięku, jego architekturę i zaimplementowane moduły.

Rozdział 6. Opis przykładów. W tym rozdziale opiszę dołączone przykłady działania silnika syntezy dźwięku.

Rozdział 7. Podsumowanie. Ten rozdział zawiera podsumowanie pracy.

Rozdział 2

Właściwości dźwięku

W rozdziale tym przedstawię podstawowe informacje na temat dźwięku. Opiszę sposób powstawania dźwięku w instrumentach muzycznych i reprezentację dźwięku w komputerze. Z uwagi na inny charakter cyfrowego przetwarzania dźwięku, pojawiają się niepożądane efekty, z którymi trzeba sobie radzić tworząc wirtualny instrument muzyczny. Wpływ *aliasingu* i *efektu Gibbsa* na konkretne algorytmy jest opisany w rozdziale 4.

2.1. Czym jest dźwięk?

Dźwięk to zmiana ciśnienia lub zaburzenie, które rozchodzi się w ośrodku sprężystym. Ośrodkiem sprężystym może być ciało stałe, ciecz lub gaz. Najczęściej człowiek postrzega dźwięk rozchodzący się w powietrzu.

Najczęściej wizualizuje się dźwięk jako wykres zniekształcenia (zmian ciśnienia) w czasie.

Częstotliwość jest liczbą oscylacji określonego punktu w czasie jednej sekundy.

Amplituda to maksymalne odkształcenie punktu. Często potocznie mówi się też o amplitudzie jako chwilowym odkształceniu punktu, wtedy wykres amplitudy w czasie będzie oznaczać to samo co wykres odkształcenia w czasie.

Dźwięki, które są sinusoidami o stałej częstotliwości, postrzegane są jako czyste tony. Jak wiadomo z analizy Fouriera każdą funkcję okresową, jaką jest dźwięk, można przedstawić jako sumę sinusów i cosinusów. Czyli w zasadzie można każdy dźwięk syntezować jako sumę sinusów. Z tego między innymi faktu korzysta przedstawiona dalej synteza addytywna. Rozkład na sinusy przydaje się także w analizie dźwięku przez człowieka. Otrzymamy wtedy wykres amplitudy sinusów względem częstotliwości. Jest to bardzo podobne do tego, jak człowiek postrzega dźwięk. Efektem ubocznym rozkładu na sinusy są wartości fazy poszczególnych sinusów. Jednakże faza dźwięku ma mniejsze znaczenie dla jego postrzegania. Człowiek z reguły słyszy dźwięki, które są wielokrotnie odbite od ścian lub innych przeszkód, przez co informacja o fazie docierająca do człowieka i tak jest gubiona. Faza natomiast może mieć znaczenie w przetwarzaniu dźwięku. Na przykład, gdy dodamy takie same dźwięki o przeciwnej fazie, to zostaną one wytłumione.

2.2. Dźwięk instrumentów

W stworzeniu wirtualnego instrumentu muzycznego przydaje się znajomość konwencjonalnych akustycznych instrumentów. W punkcie tym przedstawię charakterystykę dźwięków produkowanych przez dwie klasy instrumentów: melodyczne i perkusyjne.

2.2.1. Instrumenty melodyczne

Instrumenty melodyczne stanowią najważniejszy element zainteresowania w moim systemie. Są one najważniejszym ogniwem w muzyce kultury zachodniej. Dlatego też większość algorytmów generujących dźwięk będzie dotyczyła tych właśnie instrumentów.

Większość instrumentów melodycznych tworzy falę o strukturze harmonicznej. Dźwięk składa się z serii częstotliwości, będących wielokrotnością jednej częstotliwości, nazywanej częstotliwością bazową dźwięku. Wynika to z budowy instrumentów. W instrumentach strunowych struna ograniczona jest z dwóch stron, co ogranicza powstawanie innych częstotliwości podczas drgania. Podobnie jest w przypadku instrumentów dętych, gdzie fala powstaje w słupie powietrza, ograniczonym korpusem instrumentu.

2.2.2. Instrumenty perkusyjne

Instrumenty perkusyjne nie poddają się tak łatwo analizie. Wynika to między innymi z faktu, że instrumentem perkusyjnym może być w zasadzie wszystko, co potrafimy usłyszeć. Można wykorzystać do tego celu tupnięcie w podłogę, klaskanie, uderzenie młota w kowadło, pluśnięcie wody, czy jakikolwiek szum.

Z uwagi na dwuwymiarowy charakter drgań instrumentów perkusyjnych (np. talerz perkusyjny, bęben), powstała fala zawiera wielokrotnie więcej składowych częstotliwości niż w instrumentach melodycznych. Dodatkowo częstotliwości te powiązane są skomplikowanymi wzorami matematycznymi, co znacznie utrudnia wierną syntezę tych instrumentów.

2.3. Dźwięk w komputerze

Dźwięk jest funkcją ciągłą o charakterze okresowym i poprzez swoją złożoność nie może być tak reprezentowany w komputerze. Wartości funkcji reprezentujemy w komputerze w skończonej dokładności. Dodatkowo możemy reprezentować skończoną liczbę wartości. Mierzenie poziomu sygnału w określonych momentach czasu nazywamy *próbkowaniem*, a przejście do skończonej reprezentacji *kwantyzacją*, jak pokazuje rysunek 2.1. Proces ten zachodzi w urządzeniu ADC (konwerter analogowo-cyfrowy). Analogiczny proces to rekonstrukcja sygnału analogowego z reprezentacji cyfrowej w urządzeniu DAC (konwerter cyfrowo-analogowy).

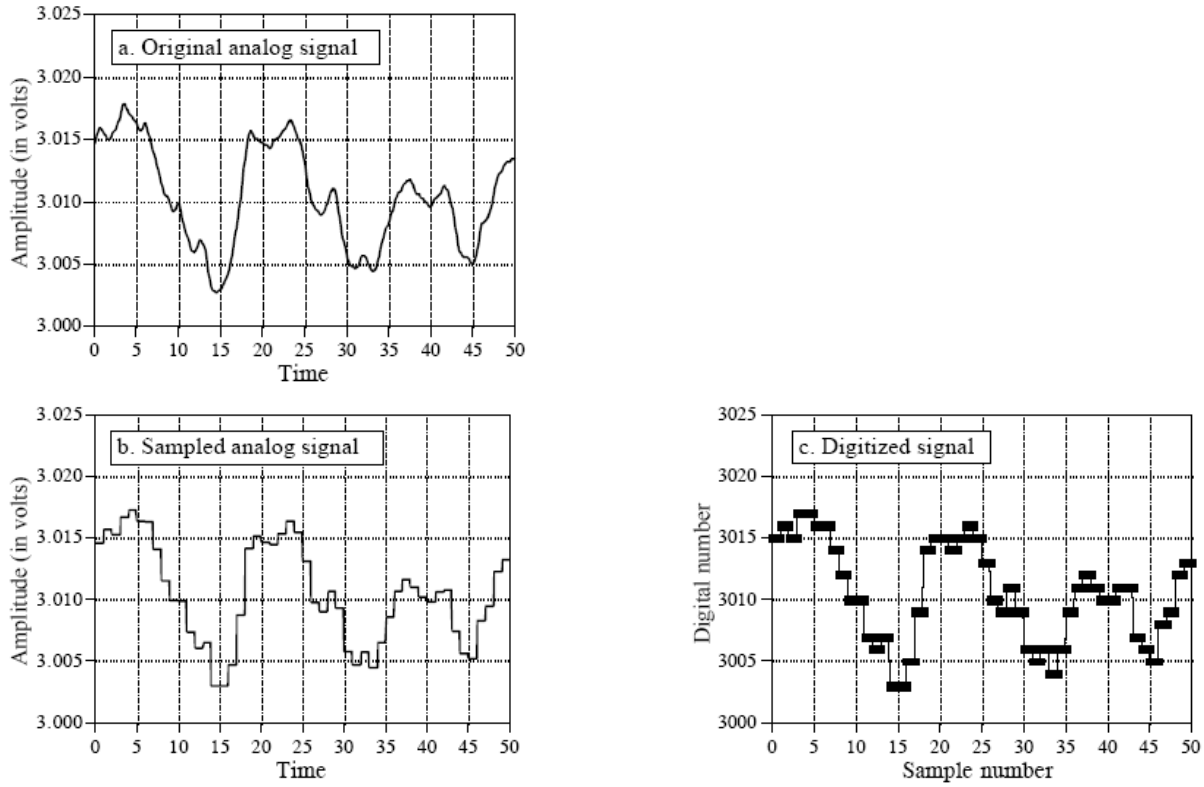
Czasem reprezentację cyfrową chcemy obejrzeć jako wykres względem częstotliwości, podobnie jak w przypadku sygnału ciągłego. Służy do tego dyskretna transformacja Fouriera (rysunek 2.2).

2.4. Problemy

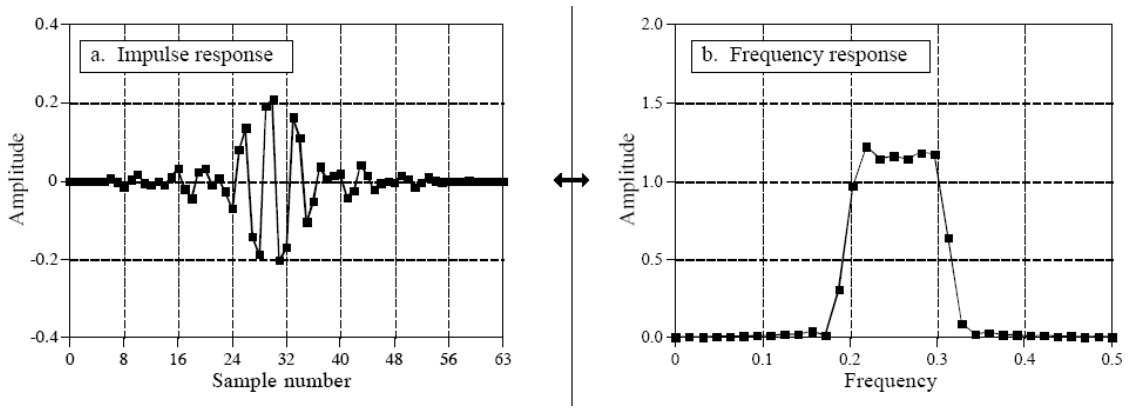
Reprezentacja sygnału w komputerze niesie za sobą szereg problemów. Niedokładności w reprezentacji liczb w komputerze powodują powstawanie szumu w dźwięku. Na szczęście łatwo to zniwelować, wybierając wystarczająco dużą reprezentację. Reprezentacja 16-bitowa daje już w zasadzie niesłyszalną dla człowieka różnicę pomiędzy głośnością sygnału a szumów, dla sygnałów o rozsądnym poziomie głośności. Dużo gorzej jest w przypadku efektów powodowanych przez próbkowanie.

2.4.1. Aliasing

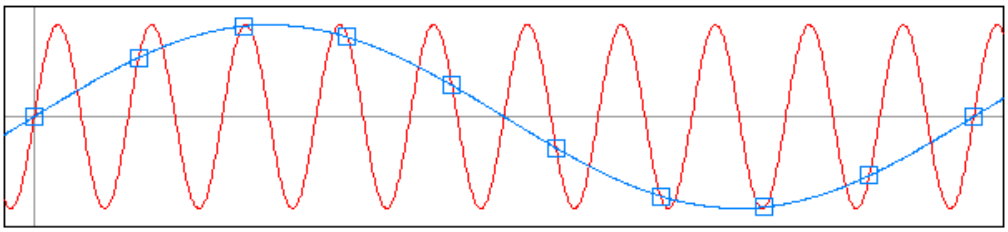
Aliasing jest efektem powodującym, że dwa różne ciągłe sygnały stają się nierozróżnialne (są swoimi aliasami), gdy są próbkowane cyfrowo. Przyglądając się rysunkowi 2.2 można



Rysunek 2.1: Sygnał oryginalny jest najpierw próbkowany, a później kwantyzowany (rysunek z [Smi99])



Rysunek 2.2: Wykres względem czasu i częstotliwości dla sygnału cyfrowego (rysunek z [Smi99])



Rysunek 2.3: Aliasing powstający w czasie próbkowania sinusoidy (rysunek z [Aliasing])

zauważyć, że sygnał w domenie częstotliwości określony jest dla wartości 0 — 0.5. Liczby te oznaczają ułamki rzeczywistej częstotliwości próbkowania. Na przykład, gdy próbkowaliśmy z częstotliwością 44100 Hz, otrzymamy przedział 0 — 22050 Hz. Wartość 22050 Hz jest zwana w tym przypadku częstotliwością Nyquista.

Kryterium Nyquista mówi, że częstotliwość próbkowania musi być większa niż dwukrotna szerokość pasma sygnału. Szerokość pasma oznaczamy jako różnicę między maksymalnymi i minimalnymi częstotliwościami składowych sinusoid. Gdy warunek ten jest zachowany, możemy z powodzeniem zrekonstruować wejściowy sygnał z jego reprezentacji cyfrowej. Warto zauważyć, że dla dźwięku kryterium to stosuje się dla pasma zaczynającego się 0, ponieważ w tym właśnie paśmie znajduje się dźwięk słyszalny dla człowieka.

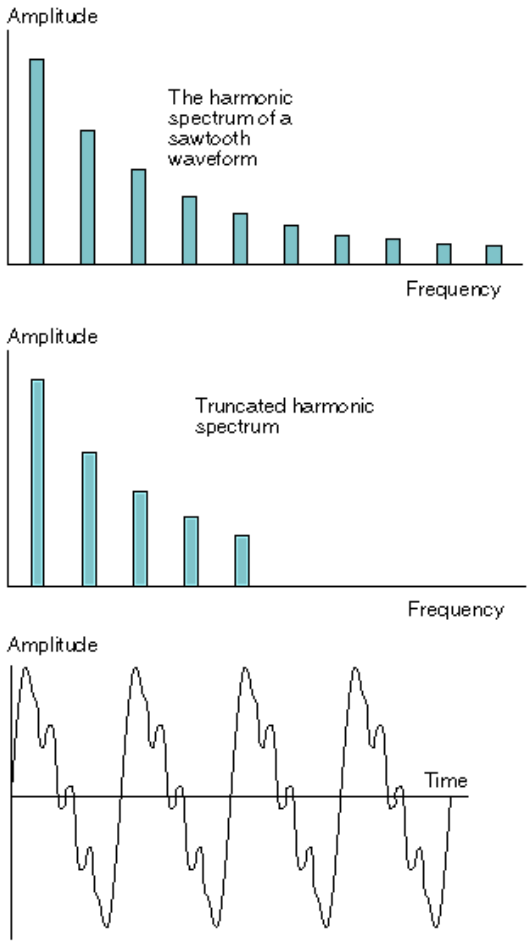
Efekt aliasingu pokazuje wykres 2.3. W tym przypadku próbujemy próbkować sinusoidę oznaczoną kolorem czerwonym. Niestety częstotliwość próbkowania jest dużo mniejsza od częstotliwości sinusoidy. Punkty pomiarów wypadają w miejscach oznaczonych kwadratami. Jak łatwo zauważyć przez punkty te możemy przeprowadzić wiele sinusoid. Podczas procesu rekonstrukcji sygnału wybieramy sinusoidę o najmniejszej częstotliwości — niebieska krzywa. Nie jest to krzywa wejściowa.

Zwiększając częstotliwość sygnału od 0 Hz będzie tak samo zwiększać częstotliwość rekonstruowanego sygnału, aż do połowy częstotliwości próbkowania. Od połowy częstotliwości próbkowania do całej częstotliwości próbkowania, częstotliwość rekonstruowanego sygnału będzie się zmniejszać. Łatwo to zaobserwować w praktyce. Wystarczy grać kolejno coraz wyższe nuty i słuchać powstającego dźwięku. Jeśli słychać w nim składowe harmoniczne, które zmniejszają częstotliwość wraz z kolejnymi nutami, to najprawdopodobniej zachodzi aliasing.

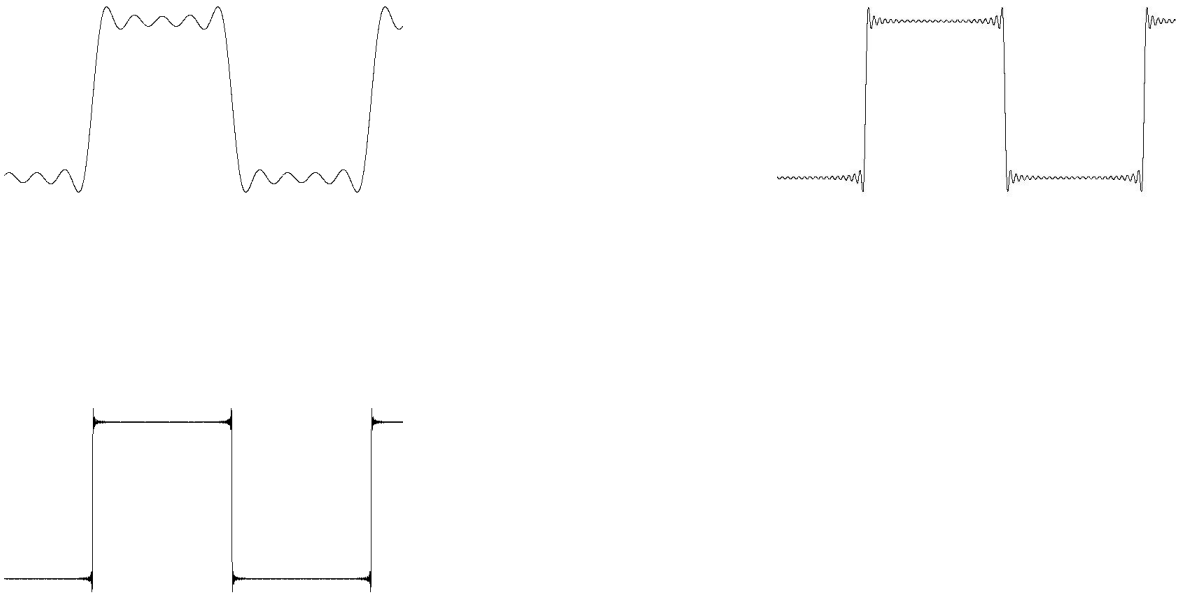
W konwerterach analogowo — cyfrowych przed próbkowaniem występuje filtr dolnoprzepustowy, przepuszczający tylko niskie częstotliwości. W ten sposób możemy zapewnić, że w nagrywanym dźwięku nie będzie aliasingu. Górne częstotliwości nie są dla nas interesujące, ponieważ występują poza pasmem słyszalnym dla człowieka. Dużo uważniej trzeba postępować w trakcie syntezy. Nie można w żadnym momencie dopuścić do wystąpienia zbyt wysokich częstotliwości, ponieważ zostaną one w próbce jako niepożądane niskie częstotliwości. Znajdujących się w próbce niskich częstotliwości, pochodzących z aliasingu, nie da się wyeliminować filtrem dolnoprzepustowym, ponieważ filtrowalibyśmy całe pasmo, w którym znajduje się też interesujący nas sygnał.

W syntezie, w czasie generowania sygnału, przykłada się dużą wagę do częstotliwości jakie powstają. Niestety aliasing może powstać także w dalszym przetwarzaniu dźwięku, przy zastosowaniu przekształceń nieliniowych, które dodają górne częstotliwości.

Filtrowanie sygnału filtrem dolnoprzepustowym obrazuje rysunek 2.4. Oryginalny sygnał, fala piłowa, miał nieskończenie wiele składowych sinusów. Filtr zadziałał w ten sposób, że



Rysunek 2.4: Filtrowanie sygnału filtrem dolnoprzepustowym (rysunek z [Rei1])



Rysunek 2.5: Kolejne przybliżenia fali kwadratowej dla 5 składowych, 25, 125 (rysunek z [Gibbs])

wyciął sinusy od pewnego momentu. Dolny obrazek pokazuje powstały w ten sposób sygnał w funkcji czasu. Wykres ten znacznie różni się od piły, ale dźwiękowo nie zauważymy różnicy, ponieważ wycięliśmy tylko niesłyszalne częstotliwości.

2.4.2. Efekt Gibbsa

Funkcje, które w rozkładzie na sinusy mają nieskończone rozwinięcie rodzą też inne problemy. Funkcje te są często wykorzystywane w klasycznych algorytmach syntezy, ponieważ reprezentują fale takie, jak piła czy fala kwadratowa, które były obecne w syntezie od początku. Obcięcie górnych składowych częstotliwości niweluje problem aliasingu, ale przyjrzyjmy się uważniej co tak naprawdę przynosi.

Weźmy funkcję fali kwadratowej, określoną nieskończonym rozwinięciem Fouriera:

$$\sin(x) + \frac{1}{3}\sin(\frac{x}{3}) + \frac{1}{5}\sin(\frac{x}{5}) + ...$$

Na rysunku 2.5 pokazano kilka przybliżeń tej funkcji odpowiednio dla 5-składowych, 25, 125. Jak łatwo zaobserwować, wielkości przewyższeń w funkcji nie zmniejszają się wraz ze wzrostem liczby składowych przybliżenia. Można powiedzieć, że błąd przybliżenia zostaje redukowany w szerokości i energii (energii sygnału w danym miejscu), ale zbiega do stałej wysokości.

Błędu przybliżeń funkcji zawierających skok nie da się zniwelować dłuższymi przybliżeniami. Udowodniono, że w ten sposób zachowują się wszystkie takie funkcje.

Dla syntezy dźwięku ma to duże znaczenie, ponieważ amplitudę sygnału musimy ograniczyć do zadanego przedziału (zwykle -1..1 dla liczb zmiennoprzecinkowych). Przekroczenie maksymalnej amplitudy powoduje zniekształcenie sygnału, co z kolei może prowadzić do dodania wyższych częstotliwości i aliasingu. Nawet, gdy nie powstanie aliasing, słyszalne jest wyraźne pogorszenie jakości dźwięku.

Z powyższych powodów wartości amplitudy funkcji trzeba utrzymywać na odpowiednio niskim poziomie, aby nie wystąpił efekt Gibbsa. Ponieważ nie zawsze da się dokładnie oszacować przewyższenie sygnału i w dalszych procesach syntezy sygnał może być wzmocniony, więc część odpowiedzialności za utrzymania niskiej głośności spada na użytkownika instrumentu.

Rozdział 3

Istniejące systemy syntezy dźwięku

Z uwagi na spore zainteresowanie komputerową syntezą dźwięków, powstały systemy ułatwiające użytkownikom tworzenie własnych instrumentów. W rozdziale tym przedstawię na przykładach dwa popularne podejścia do tego problemu.

3.1. Csound

Csound (zob. [Csound]) jest jednym z pierwszych języków programowania, powstałych specjalnie do syntezy dźwięku.

Język został skonstruowany w taki sposób, aby możliwie łatwo można było budować nowe instrumenty. Z drugiej strony trudniejszym zadaniem jest sterowanie parametrami utworzonego instrumentu. Oto przykład programu z dokumentacji systemu *Csound*:

```
          instr 3                       ; p3=duration of note
k1        linen     p4, p6, p3, p7      ; p4=amp
a1        oscil     k1, p5, 1           ; p5=freq
          out       a1                  ; p6=attack time
          endin
```

Mimo podobieństwa składni do assemblera, język *Csound* jest językiem wysokiego poziomu. Poszczególne instrukcje oznaczają całe moduły syntezy. Na przykład *linen* jest prostym generatorem konturu, a *oscil* — oscylatorem. Zmienne są parametrami modułów lub połączeniami między nimi. I tak zmienna *k1* jest wyjściem modułu *linen* i pierwszym wejściem *oscil*.

Oprócz programów definiujących instrumenty, *Csound* zawiera programy definiujące sterowanie tymi instrumentami. W programach sterujących możemy definiować zmiany parametrów w czasie, dzięki czemu może powstać muzyka.

Sterowanie instrumentami może odbywać się za pomocą zdarzeń MIDI, a od niedawna można eksportować instrumenty w formacie wtyczek do programów macierzystych.

Csound posiada ponad 450 gotowych do użycia modułów. Nie zawsze jednak istniejące moduły wystarczają. Do badania nowych algorytmów można napisać własne moduły w językach C lub C++.

Zalety systemu Csound:

- Szybkie definiowanie instrumentów.
- Wiele dostępnych modułów.

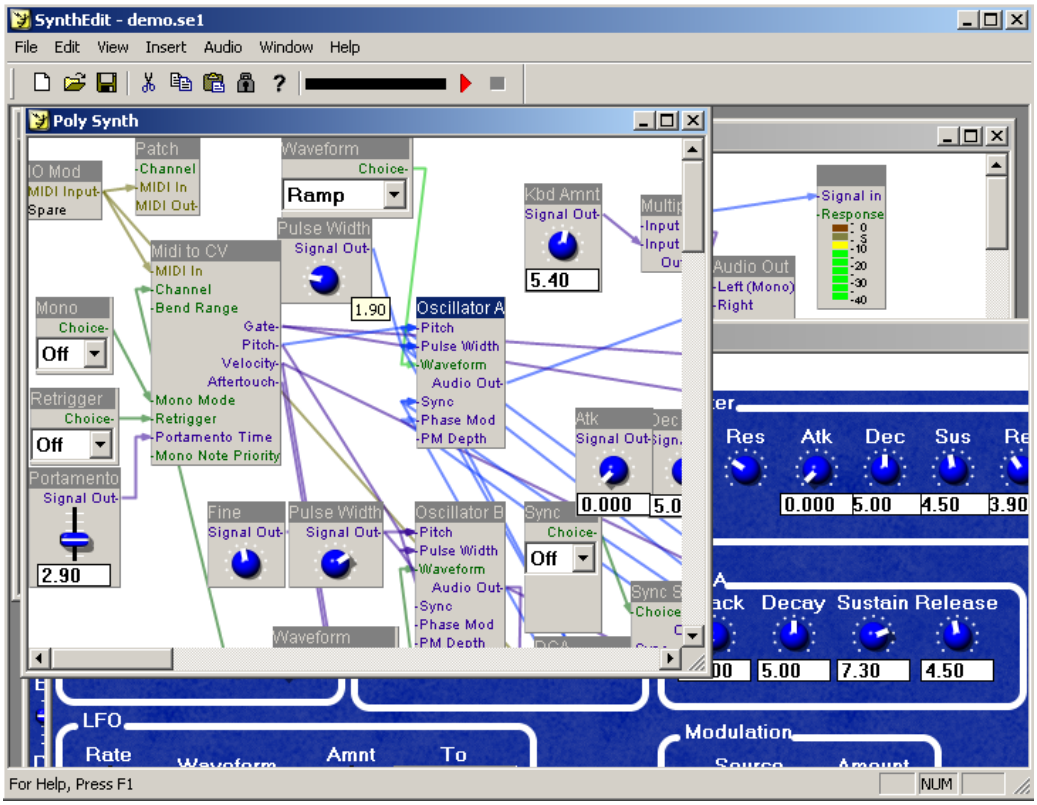
- Możliwość pisania nowych modułów w C lub C++.

Wady systemu Csound:

- Duża wielkość systemu.
- Spore wymagania pamięciowe i szybkościowe.
- Skomplikowane tworzenie nowych modułów.
- Utrudnione korzystanie z programów macierzystych.

3.2. Synthedit

Synthedit (zob. [Synthedit]) jest programem typu *shareware* i prezentuje graficzne podejście do syntezy dźwięku. System posiada wiele gotowych modułów, które wizualnie dodaje się do intrumentu i łączy ze sobą, tworząc sieć (rysunek 3.1).



Rysunek 3.1: System syntezy dźwięku *Synthedit*

Ten sposób budowania instrumentów jest w miarę intuicyjny, ale przy większej złożoności projektów staje się niepraktyczny. Powiązania między modułami mogą być bardzo skomplikowane i często łatwiej je wyrazić za pomocą języka programowania.

Synthedit oferuje możliwość eksportu instrumentu jako wtyczki VST. Oprócz tego możliwe jest skonstruowanie w łatwy sposób graficznego interfejsu użytkownika wtyczki.

Dla zaawansowanych użytkowników jest interfejs programistyczny w C++ do pisania nowych modułów.

Zalety systemu Synthedit:

- Łatwe, wizualne definiowanie instrumentów.
- Wiele dostępnych modułów.
- Możliwość pisania nowych modułów w C++.
- Wizualne konstruowanie interfejsu użytkownika.
- Eksport instrumentu jako wtyczki VST.

Wady systemu Synthedit:

- Program komercyjny.
- Wizualna definicja instrumentów może być niepraktyczna.
- Własne moduły trzeba oddzielnie kompilować.

3.3. Podsumowanie

Przedstawione systemy różnią się znacznie. *Csound* jest przeznaczony dla bardziej zaawansowanych użytkowników, z uwagi na konieczność nauczenia się nowego języka. Mimo to język ten ma bardzo zwięzłą postać, dzięki czemu nie musimy uczyć się tworzenia nawet bardzo skomplikowanych procesów syntezy dźwięku. Z kolei w *Synthedit* nie musimy długo uczyć się tworzenia instrumentów, dzięki intuicyjnemu interfejsowi użytkownika.

Tamte systemy mają wiele wad. Przede wszystkim mało praktyczne wydaje się oddzielenie implementacji modułów od budowy instrumentu. Z uwagi na to głównym ich przeznaczeniem jest konstruowanie schematów instrumentów z gotowych modułów. Przydałby się system, w którym dopisywanie nowych algorytmów było równie łatwe. Dlatego, w ramach projektu magisterskiego, podjąłem się tego zadania.

Rozdział 4

Algorytmy syntezy

Istotną rzeczą w komputerowej syntezie dźwięku (w klasycznej analogowej jest podobnie) jest modularność. Powstające algorytmy nie działają w oderwaniu od innych. Nie spotyka się przypadku, żeby w syntezie instrumentu brał udział tylko jeden moduł. Pojedynczy moduł musiałby być bardzo złożony, aby przynieść zadowalające rezultaty. Trudno byłoby go sparametryzować tak, aby tworzył nieograniczoną gamę różnorodnych dźwięków.

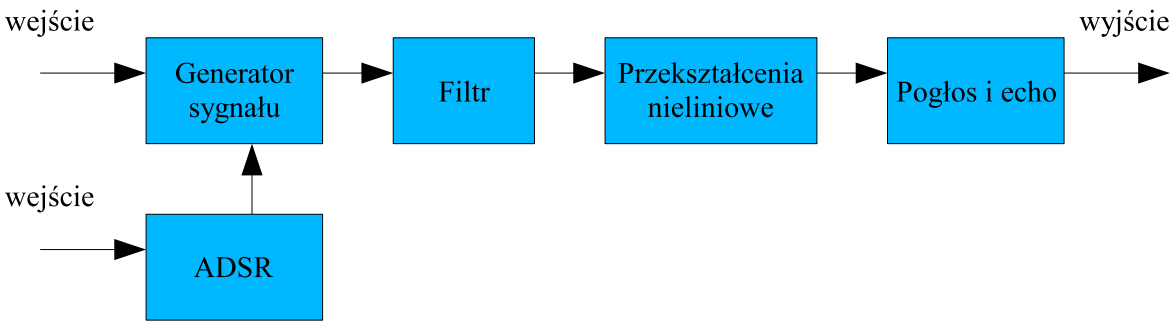
Podstawową cechą wszystkich modułów jest więc możliwość łączenia ich w dowolny sposób i tworzenia dowolnych konfiguracji. Nawet skomplikowane techniki modelowania fizycznych instrumentów wykorzystują do obróbki powstałych w ten sposób sygnałów moduły z innych sposobów syntezy.

4.1. Tor syntezy

Rysunek 4.1 pokazuje tor syntezy spotykany w większości syntezatorów. Poszczególne fragmenty toru często są dużo bardziej rozbudowane i mogą być wymieniane na inne elementy.

Wejście

Na wejściu syntezator dostaje instrukcje sterowania. Wychwytywane są wszystkie zdarzenia związane z klawiaturą muzyka. Gdy zostanie naciśnięty klawisz, przesyłana jest informacja o tonie dźwięku, który ma być zagrany i o jego amplitudzie. Gdy klawisz zostanie puszczony, przesyłana jest informacja o zakończeniu grania któregoś z aktywnych tonów. Klawiatury sterujące mogą wysyłać dodatkowe informacje, w zależności od posiadanego sprzętu. Mogą być przesyłane informacje o zmianach parametrów poszczególnych modułów.



Rysunek 4.1: Prosty schemat klasycznego toru syntezy

Generator sygnału

Generator sygnału jest elementem produkującym sygnał dźwiękowy. W tym miejscu mogą występować dwa takie elementy lub więcej. Mogą być podłączane równolegle i ich sygnały sumowane. Mogą być też podłączane szeregowo, to znaczy jeden steruje pracą drugiego. Więcej na temat tego typu zestawienia znajduje się w podrozdziale *modulacja sygnału* (4.3).

ADSR

ADSR jest najbardziej rozpowszechnionym generatorem konturu. Kontur jest specjalnym typem generatora, który nie działa okresowo, ale jest zależny od sygnałów z klawiatury. Służy do sterowania zmianami amplitudy dźwięku w czasie.

Filtr

Filtr jest modułem tłumiącym lub wzmacniającym określone częstotliwości w sygnale. Wzmacniając i tłumiąc składowe harmoniczne dźwięku w znacznym stopniu możemy kształtować jego charakter.

Przekształcenia nieliniowe

Przekształcenia nieliniowe powodują dodanie nowych częstotliwości do sygnału, wzbogacając dźwięk.

Pogłos i echo

Efekty należące do tej klasy odpowiedzialne są za umiejscowienie dźwięku w warunkach naturalnych. Symulujemy w ten sposób zachowanie dźwięku w takich środowiskach akustycznych jak pokój, czy sala koncertowa.

Wyjście

Wyjście zawiera wygenerowany wcześniej sygnał. Może on być odtworzony za pośrednictwem programu macierzystego lub zapisany na dysk.

Jak widać poszczególne elementy klasycznego toru syntezy odpowiedzialne są za różne cechy kształtowanego dźwięku. Pomimo wielkiego skomplikowania torów większości syntezatorów, podstawa na ogół zostaje taka sama.

4.2. Generowanie sygnału

Generatory sygnałów często są najważniejszymi elementami całego systemu. Tylko tutaj powstaje sygnał. Pozostałe elementy w zasadzie tylko go przetwarzają. Zgodnie z zasadą „śmieci na wejściu — śmieci na wyjściu" trzeba szczególnie zadbać o jakość generowanego sygnału. Mamy bardzo różne rodzaje generatorów, produkujących sygnały o różnym skomplikowaniu. Niektóre generowane sygnały mogą być nieciekawe i wymagają później solidnej obróbki w kolejnych modułach. Niektóre sygnały są bardzo urozmaicone i mogłyby w zasadzie być wyjściem syntezatora.

4.2.1. Oscylatory

Oscylatory są stosunkowo najprostszą metodą produkowania sygnału. Generowane funkcje zostały na ogół zaczerpnięte ze swoich analogowych odpowiedników.

Sinus

Generator sinusa wydaje się być najprostszą funkcją. Produkuje on pojedynczy ton, który nie brzmi zbyt ciekawie. Z reguły po syntezatorach oczekuje się bardziej skomplikowanych efektów. Dodatkowo pojedynczy ton słabo poddaje się obróbce. Filtrowanie takiego sygnału powoduje tylko zmianę jego amplitudy. Sinus jest natomiast bardzo często używany do modulacji innych sygnałów (zob. p. 4.3). Oprócz tego możemy połączyć równolegle kilka lub kilkanaście sinusów, produkujących różne składowe harmoniczne, co daje nam bardzo dużo możliwości kształtowania dźwięku. W końcu każdy dźwięk możemy przedstawić jako sumę sinusów.

Podstawową funkcją w programowaniu do wyznaczania wartości sinusa jest *sinf*. Korzysta ona z rozwinięcia serii Taylora do kilku pierwszych składników:

$$\sin x = x \quad \frac{x^3}{3!} + \frac{x^5}{5!} \quad \frac{x^7}{7!} + ...$$

Kiedy zależy nam na szybkości (a w syntezie czasu rzeczywistego nam zależy), możemy skorzystać z kilku możliwych przybliżeń.

W przedstawionych algorytmach (zob. [Trig]) s_n oznacza $\sin(2\pi n/N)$, a c_n oznacza $\cos(2\pi n/N)$.

Algorytm 1

$$\begin{aligned} s_0 &= 0 \\ c_0 &= 1 \\ s_{n+1} &= s_n + d \quad c_n \\ c_{n+1} &= c_n \quad d \quad s_n \end{aligned}$$

gdzie $d = 2n/N$

Jest to najszybsza aproksymacja sinusa i cosinusa. Niestety wprowadza dosyć duży błąd, proporcjonalny do $1/N$. Dla $N = 256$ maksymalny błąd sinusa wynosi $0,61$.

Algorytm 2

$$\begin{aligned} s_0 &= 0 \\ c_0 &= 1 \\ s_{n+1} &= w_i\, c_n + w_r\, s_n \\ c_{n+1} &= w_r\, c_n \quad w_i \quad s_n \end{aligned}$$

gdzie $w_r = \cos(2\pi/N)$ i $w_i = \sin(2\pi/N)$

Algorytm 2 generuje matematycznie dokładne wartości, ale jest podatny na błędy zmienno—przecinkowe. Błąd wynosi $O(\epsilon N)$, gdzie ϵ jest dokładnością reprezentacji liczb.

Algorytm 3

$$\begin{aligned} s_0 &= 0 \\ c_0 &= 1 \\ s_{n+1} &= s_n + (\beta c_n \quad \alpha s_n) \\ c_{n+1} &= c_n \quad (\alpha c_n + \beta s_n) \end{aligned}$$

gdzie $\alpha = 2\sin^2(\pi/N)$ i $\beta = \sin(2\pi/N)$

Ten algorytm jest ulepszoną wersją algorytmu 2. Średni błąd obliczeń wynosi $O(\epsilon\sqrt{N})$, choć pesymistyczny błąd jest taki sam, jak w algorytmie 2.

Zwróćmy uwagę, że algorytmy 2 i 3 nie sprawdzają się w przypadku częstych zmian częstotliwości, gdyż dla każdej częstotliwości trzeba obliczyć odpowiednie współczynniki (w_r i w_i dla algorytmu 1, α i β dla algorytmu 2), co sprowadza się do policzenia wartości sinusów i cosinusów.

Piła

Fala piłowa ma prostą strukturę względem czasu. W najprostszej implementacji wystarczy w każdej próbce zwiększać wartość próbki o stałą wartość, zależną od częstotliwości. Gdy wartość przekroczy 1, odejmuje się od niej 2. Okazuje się jednak, że ma ona skomplikowaną strukturę harmoniczną. W rozkładzie na sinusy otrzymujemy następujący wzór:

$$f(x) = \frac{\sin x}{x} + \frac{\sin 2x}{2x} + \frac{\sin 3x}{3x} + ... \tag{4.1}$$

Jak widać mamy nieskończoną sumę składowych harmonicznych, których amplitudy stosunkowo wolno maleją ($\frac{1}{n}$). To bogactwo harmoniczne daje nieograniczone możliwości kształtowania dźwięku. Fala ta była od początku wykorzystywana w analogowych syntezatorach dźwięku i jest używana do dzisiaj w syntezatorach cyfrowych. Duże możliwości daje dalsze kształtowanie sygnału zestawem filtrów, które mają za zadanie wzmocnić i wytłumić odpowiednie składowe harmoniczne, dając dużą kontrolę nad końcowym efektem.

Innym zastosowaniem fali piłowej jest synteza sekcji smyczkowej. Przypominam, że czysty dźwięk skrzypiec daje właśnie falę piłową. Sumując kilka takich fal, o nieznacznie odchylonych częstotliwościach, otrzymujemy bardzo ciekawe i trochę nawet realistyczne brzmienie. Nic dziwnego, że ta technika stała się podstawą w budowie syntezatora smyczków (ang. string synth).

Niestety w domenie cyfrowej piła nastręcza wiele trudności. Nieskończony ciąg wolno malejących składowych harmonicznych może prowadzić do wyraźnie słyszalnego aliasingu. Jednym ze sposobów ominięcia tego problemu jest synteza sygnału wprost ze wzoru 4.1. Obliczamy wartość próbki licząc sumę sinusów aż do osiągnięcia połowy częstotliwości próbkowania. Dla wysokich częstotliwości nie będzie z tym problemu. Biorąc 5000 Hz wystarczy policzyć 4 składniki, bo ostatni będzie miał częstotliwość 20000 Hz, co jest na granicy ludzkiego słuchu. Problem stanowią niskie częstotliwości. Dla 100 Hz otrzymamy 200 składowych harmonicznych, a nie jest to dolna granica wykorzystywana w muzyce. Ewentualnie możemy ograniczyć liczbę składowych do stałej liczby, np. dziesięciu. Niestety otrzymamy wtedy dźwięk „przytłumiony". Należy także pamiętać o ograniczeniu maksymalnej amplitudy dźwięku, ponieważ obcięcie wyższych składowych harmonicznych powoduje efekt Gibbsa (zob. p. 2.4.2).

W komercyjnych syntezatorach, do syntezy fali piłowej często wykorzystuje się tablicę fal. W czasie syntezy korzysta się z wcześniej wyliczonych wartości, zob. p. 4.2.2.

Fala kwadratowa

Fala kwadratowa przyjmuje na przemian tylko dwie wartości: wartość amplitudy i minus wartość amplitudy. Z uwagi na wyraźne skoki łatwo się domyślić, że ma ona nieskończony ciąg składowych harmonicznych:

$$f(x) = \frac{\sin x}{x} + \frac{\sin 3x}{3x} + \frac{\sin 5x}{5x} + ...$$

Obecne są tylko nieparzyste składowe harmoniczne, więc otrzymamy mniej bogaty, czyli bardziej delikatny dźwięk w porównaniu do fali piłowej. Fala kwadratowa bywa często wykorzystywana do „udawania" instrumentów dętych.

Wszystkie trudności z syntezy fali piłowej przenoszą się na falę kwadratową. Jesteśmy jednak w trochę lepszej sytuacji, ponieważ mamy dwukrotnie mniej istotnych składowych.

4.2.2. Tablica fal

Pojęcie tablicy fal (ang. wavetable) łączy się współcześnie z dwoma rodzajami tablic próbek. Z pierwszym mamy do czynienia w *samplerach*, a z drugim w *syntezatorach*.

Tablice w samplerach

Samplery generalnie są mniej ciekawymi urządzeniami w porównaniu do syntezatorów. Ich rola sprowadza się do odtwarzania wcześniej nagranych próbek dźwiękowych. Mamy więc tablicę wszystkich nagranych próbek i odtwarzamy jedną z nich w zależności od potrzeb. Każda próbka ma zapisaną częstotliwość dźwięku, z jaką była nagrana oraz przypisana jest do określonej strefy klawiatury (jednego lub większej liczby klawiszy) i określonej głośności.

Przypuśćmy, że naciskamy klawisz C4 z siłą 80. Sampler wyszukuje próbkę skojarzoną z klawiszem C4 i amplitudą 80 i zaczyna ją odgrywać. Jeśli trzymamy długo naciśnięty klawisz, to część próbki może stanowić pętlę, która jest powtarzana w razie potrzeby. Kiedy puścimy klawisz, zostanie odegrana część odpowiedzialna za zakończenie dźwięku.

W przypadku, gdy odgrywana próbka była nagrana w innej częstotliwości (np. gramy C4, a mamy nagrane D4), sampler musi zmienić częstotliwość odtwarzanego dźwięku. Istnieje kilka metod zmiany częstotliwości próbkowania. Najprostsza to odgrywanie próbki w innej prędkości. Wadą tej techniki jest zmiana czasu trwania próbki. Bardziej zaawansowane techniki transformują próbkę do domeny częstotliwości (szybka transformacja Fouriera), tam dokonują przekształcenia i znów transformują próbkę do domeny czasu (odwrotna szybka transformacja Fouriera).

Współczesne biblioteki próbek dla jednego instrumentu mają często osobne próbki dla każdego klawisza (w zakresie danego instrumentu), dla kilku głośności grania i dla różnych artykulacji. Część próbek może być odpowiedzialna za granie wolne, część za szybkie, *staccato*, *legato* itd. Pamięć potrzebna do zmagazynowania takiej tablicy sięga dziesiątków gigabajtów. Dlatego rozwinęły się techniki zapamiętywania początków próbek i dogrywania reszty w razie potrzeby. Mimo, że tak wielka ilość potrzebnej pamięci może się wydawać niepraktyczna, wciąż jest to najlepszy sposób na symulowanie prawdziwych instrumentów.

Tablice w syntezatorach

W syntezatorach tablice próbek służą w zasadzie tylko przyspieszeniu obliczeń. Chodzi o to, żeby można było odtwarzać ciekawe fale, znane z syntezatorów analogowych, w jak najmniejszej złożoności obliczeniowej, ale z zachowaniem ograniczenia pasma częstotliwości tak, żeby

nie występował aliasing. Oczywiście implementacja tablicy próbek niesie za sobą poboczną zaletę — dodawanie własnych fal użytkownika.

Tablica próbek dla jednej fali ma określoną liczbę rekordów, każdy przypisany innej częstotliwości. Częstotliwości mogą być dobrane na kilka sposobów:

- wykładniczo — tak człowiek postrzega dźwięk, oszczędzamy dużo miejsca w porównaniu z pozostałymi sposobami,
- częstotliwości nut w muzyce (klawiszy w klawiaturze muzycznej) — są to najczęściej używane częstotliwości, ale trzeba pamiętać, że z reguły możemy delikatnie rozstrajać oscylatory lub robić płynne przejścia pomiędzy tonami, czyli nie wyłapiemy wszystkich potrzebnych fal,
- według liczby składowych harmonicznych — dla każdej liczby składowych harmonicznych bierzemy największą częstotliwość, w której nie zachodzi aliasing.

Dla każdej częstotliwości mamy próbkę o tej samej długości N (np. $N = 512$). To jest zasadnicza różnica w porównaniu z tablicą w samplerach. Tam długość próbki zależała od długości nagrania. Dla każdej częstotliwości wyliczamy tylko jeden okres danej fali o maksymalnej liczbie składowych harmonicznych, nie powodujących aliasingu. Robimy to w ten sposób, żeby wszystko zajęło nam dokładnie N pozycji. Zwróćmy uwagę, że tylko liczba składowych harmonicznych zależy od częstotliwości próbkowania. W czasie bezpośredniego odtwarzania wszystkie te próbki dałyby ten sam ton (z różną liczbą składowych). Oprócz tego funkcja sinus dla wszystkich częstotliwości będzie miała taką samą postać. Sinus ma tylko jedną składową harmoniczną, więc w zakresie częstotliwości słyszalnych dla człowieka nie powoduje aliasingu.

W czasie odtwarzania danej fali z częstotliwością f, wybieramy próbkę zbudowaną dla częstotliwości - f. Odtwarzając próbkę po kolei, cyklicznie, otrzymamy falę okresową o częstotliwości s/N, gdzie s to częstotliwość próbkowania. Żeby otrzymać częstotliwość fali f, musimy więc ją odtwarzać przesuwając się o $N\frac{f}{s}$ pozycji. Będąc w pozycji niecałkowitej możemy brać wartość z zaokrąglenia pozycji. Lepszym sposobem jest zastosowanie interpolacji. Najczęściej wykorzystywaną interpolacją jest interpolacja liniowa, choć czasem stosuje się interpolacje wyższych stopni.

Zastosowanie interpolowanej syntezy tablicy fal nie gwarantuje niewystąpienia aliasingu, jeśli przy odtwarzaniu będziemy zmieniać pozycję o więcej niż jedną wartość. Oznacza to, że dobór długości próbki N ma bardzo duże znaczenie tak dla jakości generowanych fal, jak i zajętości pamięci.

4.2.3. Szum i chaos

Wiele zjawisk zachodzących w przyrodzie jest bliskie losowości. Pomijając naturę procesów subatomowych, mimo ciągłego rozwoju nauki nie da się w pełni zamodelować otaczającego nas świata. Jest zbyt wiele zmiennych i zbyt wiele niewiadomych, by stworzyć zwięzły opis. Dlatego często istotny czynnik w modelach stanowi losowość. Zauważmy, że sumowanie wielu zmiennych losowych daje rozkład bliski normalnemu, tak bardzo powszechny w otaczającym nas świecie.

Podobnie jest w przypadku dźwięku. Gdy złożoność i nieznane czynniki nie pozwalają w pełni oddać jego natury przy pomocy dostępnych nam środków, często wykorzystujemy w tym celu losowość. Modułem w syntezie dźwięku spełniającym tę rolę jest generator szumu. Generator szumu to generator sygnału, dający na wyjściu ciąg wartości losowych. Generatory

takie często były wykorzystywane w syntezie analogowej do tworzenia instrumentów perkusyjnych, w których mnogość składowych częstotliwości i ich brak związku pomiędzy sobą, nie pozwalały stworzyć lepszego modelu. Sygnał losowy, odpowiednio przefiltrowany, dawał efekty bliskie oczekiwanym.

Losowość w syntezie dźwięku wykorzystuje się na wiele innych sposobów. Przykładem mogą być losowe wahania częstotliwości w oscylatorach — tak jak się to dzieje w oscylatorach analogowych, czy modele fizyczne instrumentów (zob. p. 4.2.4).

4.2.4. Modelowanie fizyczne

Odrębną dziedziną w syntezie dźwięku jest modelowanie fizyczne. W klasycznej syntezie, gdy chcemy naśladować prawdziwe instrumenty, analizujemy ich brzmienie i staramy się je zasymulować, wykorzystując do tego celu dostępne nam środki wyrazu. Stosujemy generatory różnych fal, filtry, efekty, modulację. Moduły syntezatorów zupełnie się różnią od elementów wytwarzających dźwięk klasycznych instrumentów.

W modelowaniu fizycznym najważniejszą sprawą nie jest analiza dźwięku, który się wydobywa z instrumentu, tylko analiza pracy samego instrumentu, chodzi nam o sposób w jaki dźwięk się tworzy.

Algorytm Karplus-Stronga

Algorytm Karplus-Stronga (zob. [Bur]) symuluje pracę struny szarpanej. Bazuje na syntezie tablicy fal. Kroki działania algorytmu:

1. Weź tablicę długości równej okresowi dźwięku dla zadanej częstotliwości f. Długość tablicy wynosi s/f, gdzie s to częstotliwość próbkowania.

2. Wypełnij tablicę wartościami losowymi z przedziału $-1..1$.

3. Odtwarzaj tablicę cyklicznie.

4. Po odtworzeniu każdej wartości uśrednij dwie sąsiednie wartości.

W momencie szarpnięcia struny, zaczyna się ona poruszać w bardzo niekontrolowany sposób. Jest dużo składowych o dużej energii, czyli wysokiej częstotliwości. Ten chaotyczny charakter struny symulowany jest przez wypełnienie tablicy losowymi wartościami. Z biegiem czasu ruch struny uspokaja się, składowe wysoko-energetyczne wytłumiają się i zostają składowe o niskiej częstotliwości. Ten proces symulowany jest poprzez uśrednianie wartości. Zwróćmy uwagę, że te proste działanie, niewątpliwie zmniejszające energię układu, jest po prostu filtrem dolnoprzepustowym. Mimo całej losowości algorytmu, wzięcie tablicy cyklicznej o zadanej długości zapewnia nas, że produkowany dźwięk będzie zawierał tylko interesujące nas składowe harmoniczne. Ton bazowy ma dokładnie okres tablicy.

Mimo całej swojej prostoty zarówno pod względem algorytmu, jak i złożoności obliczeń, algorytm Karplus-Stronga daje bardzo realistyczne rezultaty. Niewątpliwie otrzymany dźwięk brzmi jak szarpany instrument strunowy, przypominający trochę klawesyn. Losowość algorytmu ma też dodatkową zaletę. Każde uruchomienie syntezy (szarpnięcie struny) różni się trochę od poprzedniego, dzięki czemu sprawia wrażenie bardziej naturalnego.

Inne algorytmy modelowania fizycznego

Modelowanie fizyczne jest dosyć młodą dziedziną syntezy dźwięku. Wynika to po części ze słabej mocy obliczeniowej dawnych komputerów i skomplikowanej analizy pracy instrumentów.

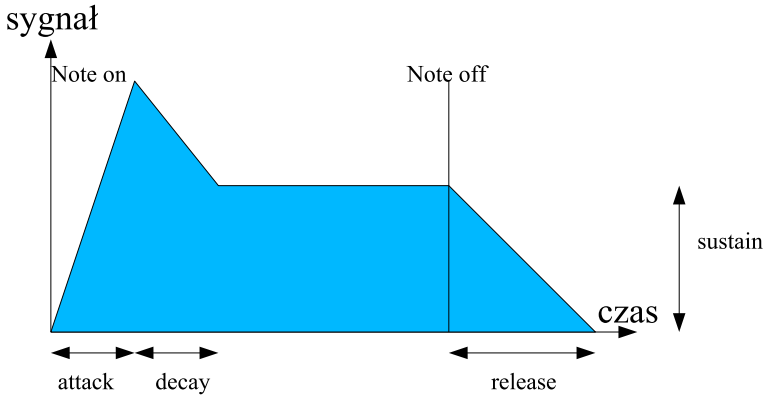
Szczególnie twórczym tematem modelowania okazało się symulowanie pracy instrumentów strunowych, głównie szarpanych. Nowoczesne algorytmy dużo wiernie oddają prawdę od algorytmu Karplus-Stronga. Modeluje się faktyczne wychylenie struny w określonym punkcie (często parametryzowanym) i zachowanie się układu po puszczeniu struny. Parametryzowane jest też miejsce, w którym badamy wartości odchylenia struny, czyli umiejscowienie pudła rezonansowego w instrumentach klasycznych lub przetworników w gitarze elektrycznej.

Pewne postępy poczyniono też w modelowaniu instrumentów dętych.

4.2.5. Generator konturu

Generator konturu jest szczególnym rodzajem generatora. Tworzy on sygnał ciągły, stopniowo zmieniający się w czasie. Ponieważ sygnał nie jest okresowy, nie ma on właściwości muzycznych. Generator konturu służy do sterowania pracą innych modułów.

Najbardziej rozpowszechnionym typem tego urządzenia jest *ADSR Envelope* (zob. [ADSR]). Nazwa pochodzi z angielskich słów *attack*, *delay*, *sustain*, *release*.



Rysunek 4.2: Działanie konturu ADSR

Działanie konturu ADSR pokazuje rysunek 4.2. Tworzony sygnał zależy od zdarzeń klawiatury muzycznej. Przechwytywane są zdarzenia naciśniecia klawisza (*note on*) i puszczenia klawisza (*note off*).

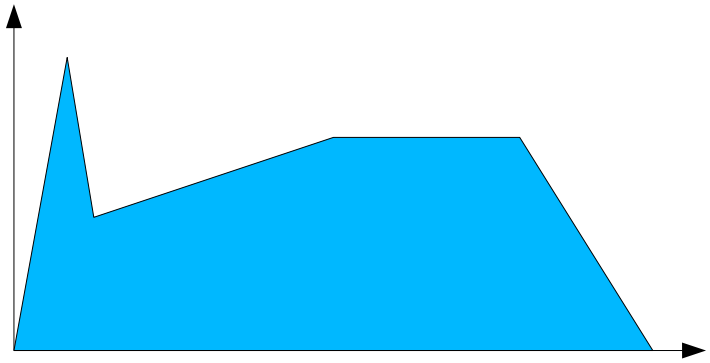
W momencie rozpoczęcia nuty sygnał zaczyna rosnąć od wartości 0, aż do osiągnięcia maksymalnej głośności sygnału. Czas trwania narastania sterowany jest parametrem *attack*. Po osiągnięciu górnego pułapu, sygnał zaczyna opadać w czasie *decay*, aż do osiągnięcia wartości *sustain*. Wartość ta jest utrzymywana przez cały czas grania dźwięku, aż do puszczenia klawisza. Wtedy sygnał zaczyna opadać do 0 w czasie, określonym parametrem *release*.

Jak widać długość generowanego sygnału zależy od czasu trwania nuty, natomiast częstotliwość nuty nie ma tu znaczenia.

Najczęstszym sposobem wykorzystania konturu ADSR jest sterowanie artykulacją dźwięku. Kontur podłącza się do amplitudy generatora sygnału dźwiękowego, dzięki czemu możemy zasymulować różne sposoby grania instrumentów. Dla przykładu instrumenty smyczkowe będą miały długi czas ataku, wysoki poziom podtrzymywania dźwięku i długi czas wygaszania

dźwięku. Dla odróżnienia pianina mają charakter bardziej perkusyjny, czyli należy ustawić krótki czas ataku i niski poziom podtrzymywania.

Niewątpliwą zaletą powyższego generatora jest jego prosta, intuicyjna budowa, dzięki czemu znalazł wiele zastosowań. Niestety nie zawsze to wystarcza. Czasem początkową fazę zmian amplitudy dźwięku instrumentu potrzebujemy określić na większej liczbie parametrów. Dobrym przykładem są tu instrumenty dęte blaszane. Ich przybliżoną artykulację pokazuje rysunek 4.3. Po krótkim okresie ataku, amplituda szybko maleje do określonego poziomu, a później stopniowo rośnie, osiągając poziom *sustain*. Dla takich nietypowych zastosowań powstało wiele innych konturów, o dużo większej liczbie parametrów.



Rysunek 4.3: Przybliżona artykulacja instrumentów dętych blaszanych

Aby jeszcze bardziej zwiększyć możliwości artykulacji w generatorach konturów często zamiast linii prostych stosuje się łuki, o regulowanym stopniu zakrzywienia.

Oprócz sterowania amplitudą dźwięku, możemy sterować innymi modułami toru syntezy, otrzymując inne interesujące efekty. Podłączenie konturu do częstotliwości tłumienia filtrów daje nam możliwość zmiany amplitudy niektórych składowych harmonicznych w trakcie trwania dźwięku.

4.3. Modulacja sygnału

Generowanie klasycznych fal syntezy analogowej czasem może nie wystarczać do osiągnięcia zamierzonych celów. Jednym ze sposobów na wzbogacenie mało interesującego sygnału jest jego modulacja.

4.3.1. Modulacja amplitudy

Modulacja amplitudy występuje w przypadku, gdy sterujemy wartością amplitudy generatora sygnału (zwanego *nośnikiem*), używając w tym celu innego generatora sygnału okresowego (zwanego *modulatorem*). Proces sprowadza się do dodania wartości sygnału modulatora do amplitudy nośnika. Częstotliwość pracy modulatora jest z reguły dużo niższa od częstotliwości pracy nośnika, często nawet w paśmie niesłyszalnym dla człowieka.

Do modulacji często używa się sinusa. Dla niskich częstotliwości daje to efekt stopniowych, okresowych zmian w głośności dźwięku, pulsowanie dźwięku, w muzyce określane terminem *tremolo*. Najbardziej efektownym przykładem efektu *tremolo* jest jego stosowanie w instrumentach smyczkowych. Polega on na szybkim poruszaniu smyczkiem w obie strony i graniu

pojedynczego dźwięku. Modulacja amplitudy jest więc kolejnym, ciekawym sposobem artykulacji dźwięku.

Zaobserwujmy co się stanie z sygnałem wyjściowym, gdy częstotliwość modulatora jest wysoka (zob. [Rei11]). Weźmy dwa przykładowe sygnały:

$$A_1 = a_1 \cos(\omega_1 t) \tag{4.2}$$
$$A_2 = a_2 \cos(\omega_2 t) \tag{4.3}$$

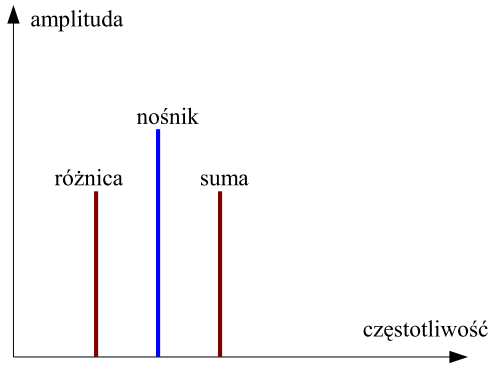
Rozważam zachowanie się cosinusów, gdyż każdy sygnał można przedstawić jako sumę cosinusów. Sygnał A_1 jest nośnikiem, a A_2 modulatorem. Stosując modulację amplitudy do nośnika otrzymamy:

$$A_1 = (a_1 + A_2) \cos(\omega_1 t) \tag{4.4}$$

Sygnał A_2 jest dany wzorem 4.3, więc możemy go wstawić do 4.4, otrzymując:

$$A_1 = (a_1 + a_2 \cos(\omega_2 t)) \cos(\omega_1 t) \tag{4.5}$$
$$A_1 = a_1 \cos(\omega_1 t) + a_2 \cos(\omega_2 t) \cos(\omega_1 t) \tag{4.6}$$
$$A_1 = a_1 \cos(\omega_1 t) + \frac{1}{2} [a_2 \cos((\omega_1 + \omega_2)t)] + \frac{1}{2} [a_2 \cos((\omega_1 \quad \omega_2)t)] \tag{4.7}$$

Ze wzoru 4.7 wynika, że w sygnale wynikowym mamy 3 składowe częstotliwości: częstotliwość nośnika, sumę i różnicę częstotliwości nośnika i modulatora. Wykres częstotliwości w wynikowym sygnale przedstawia rysunek 4.4.



Rysunek 4.4: Wykres częstotliwości wynikowego sygnału modulacji amplitudy

Zauważmy, że składowe harmoniczne wyniku modulacji amplitudy nie mają zależności harmonicznej, tzn. nie są wielokrotnością jednej częstotliwości. Stosując bardziej rozbudowane sygnały od sinusów i cosinusów, możemy otrzymać wiele niepowiązanych składowych w sygnale wyjściowym (każda składowa częstotliwości modulatora moduluje każdą składową częstotliwości nośnika). Jest to podstawą modulacji dzwoniącej (ang. ring modulation), często używanej do symulowania talerzy perkusyjnych.

4.3.2. Modulacja częstotliwości

Modulacja częstotliwości to sterowanie zmianą częstotliwości nośnika z pomocą sygnału modulatora. Jeśli mamy dwa sygnały:

$$A_1 = a_1 \cos(\omega_1 t) \tag{4.8}$$
$$A_2 = a_2 \cos(\omega_2 t) \tag{4.9}$$

to sygnał po modulacji częstotliwości będzie miał postać:

$$A_1 \quad = \quad a_1 \cos((\omega_1 + A_2)t) \tag{4.10}$$

Używając do modulacji sinusoidy o niskiej częstotliwości uzyskamy wolne, stopniowe zmiany częstotliwości nośnika. W muzyce efekt ten występuje pod nazwą *vibrato*.

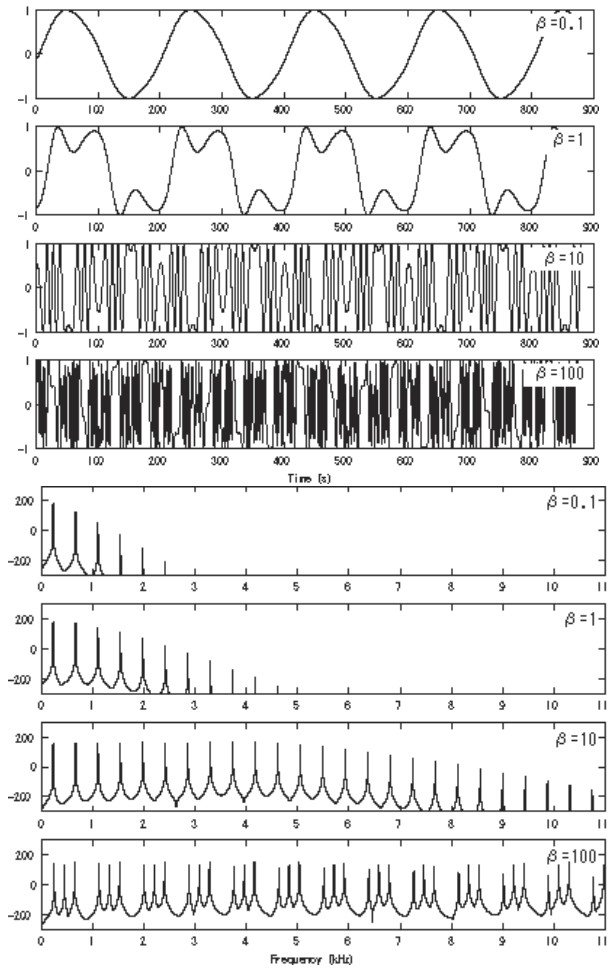
Sytuacja komplikuje się w przypadku modulowania sygnałem o wysokiej częstotliwości. W przeciwieństwie do modulacji amplitudy, nie uzyskamy 2 dodatkowych częstotliwości, lecz nieskończenie wiele:

$$\omega = \omega_1 \quad n\omega_2 \qquad \text{dla } n = 0, 1, 2, ... \tag{4.11}$$

Amplitudy poszczególnych częstotliwości wyrażone są dosyć zawiłymi równaniami matematycznymi. Wynik modulacji zależy od indeksu modulacji (β), wyrażonego wzorem:

$$\beta = \frac{\Delta\omega_1}{\omega_2} \tag{4.12}$$

Ze wzoru 4.12 wynika, że β zależy od $\Delta\omega_1$, czyli zmiany częstotliwości nośnika. Z kolei zmiana częstotliwości nośnika zależy od amplitudy modulatora. Czyli wpływ na wynik modulacji ma nie tylko częstotliwość, ale też amplituda.



Rysunek 4.5: Zależność wyniku modulacji częstotliwości od parametru β, rysunek z [Freq]

Rysunek 4.5 pokazuje wynik modulacji nośnika o częstotliwości 220 Hz poprzez modulator o częstotliwości 440 Hz względem czasu i częstotliwości, dla różnych wartości parametru β.

Modulacja częstotliwości była często używana w pierwszych syntezatorach cyfrowych. Używając modulatora o częstotliwości harmonicznie związanej z częstotliwością nośnika można uzyskać bogate ciągi harmoniczne, przydatne do syntezowania instrumentów melodycznych. Z kolei używając do modulacji fal o częstotliwościach niezwiązanych harmonicznie uzyskamy wyniki o strukturze nieharmonicznej, często wykorzystywane w syntezie instrumentów perkusyjnych.

Główną wadą syntezy poprzez modulację częstotliwości są jej nieintuicyjne rezultaty. Nie wszyscy również lubią brzmienia w ten sposób uzyskiwane. Nie zmienia to jednak faktu, że jest to interesujące narzędzie do syntezy dźwięku.

4.4. Przekształcenia liniowe

Ciekawą klasą modułów przetwarzania dźwięku są przekształcenia liniowe. Przekształcenia liniowe mają szereg własności, z których w syntezie dźwięku najistotniejszymi są:

- Proces można podzielić na podprocesy, przetwarzać szeregowo lub równolegle i nie ma to wpływu na końcowy rezultat. Z własności tej wynika naturalna modularność systemów przetwarzania dźwięku.
- Po przekształceniu liniowym nie pojawiają się nowe składowe częstotliwości. Dzięki temu w procesie liniowym nie pojawi się efekt aliasingu.

Działanie procesu liniowego możemy zbadać doświadczalnie wprowadzając na wejściu impuls i zapisując sygnał wynikowy procesu. Impuls jest sygnałem wyrażonym przez funkcję δ:

$$\delta(n) = \begin{cases} 1 & \text{dla } n = 0 \\ 0 & \text{dla } n > 0 \end{cases}$$

Wynikiem impulsu jest odpowiedź impulsowa, która w pełni wystarcza za opis całego układu. Wiedząc jak proces zachowuje się dla impulsu o wartości 1, wiemy jak będzie się zachowywał dla dowolnego impulsu. Wystarczy pomnożyć odpowiedź impulsową przez stałą wartość. Równie łatwo otrzymać wynik działania dla całego sygnału. Wystarczy policzyć odpowiedzi dla poszczególnych wartości (odpowiedź impulsowa dla przesuniętej wartości też jest przesunięta) i dodać wyniki do siebie. Działanie te jest zwane *splotem*:

$$y[n] = x[n] \quad h[n] = \sum_{j=0}^{M-1} h[j]x[i \quad j]$$

gdzie x jest sygnałem wejściowym, h odpowiedzią impulsową układu, a y sygnałem wyjściowym. Długość odpowiedzi impulsowej wynosi M. Pojedyncza wartość wyjściowa zależy od M wartości wejściowych.

Odpowiedź impulsowa jest dobrym opisem każdego układu liniowego. Z uwagi na dużą złożoność obliczeniową splotu, w praktyce używa się lepszych metod obliczania wyników konkretnych procesów.

4.4.1. Filtry

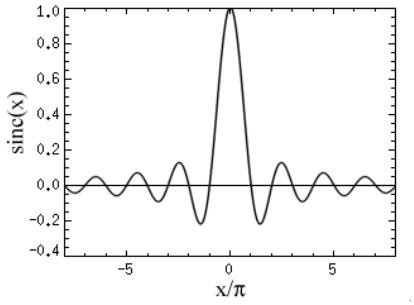
Filtry służą do wzmacniania lub tłumienia wybranych częstotliwości dźwięku. Te wybiórcze działanie okazuje się bardzo przydatne w syntezie. Do podstawowych zadań filtrów należą:

- Kształtowanie charakteru barwy dźwięku. Jest to podstawa tzw. *subtractive synthesis*. Wychodząc od bogatych fal dźwiękowych klasycznej syntezy, takich jak piła czy fala kwadratowa, zmniejsza się amplitudy niektórych składowych harmonicznych, tworząc unikatowe brzmienia.

- Wyciszanie niechcianych częstotliwości. Czasem w trakcie syntezy powstają niepożądane składowe częstotliwości. Jeśli znajdują się one poza pasmem syntezowanego dźwięku, to można je w ten sposób wyeliminować. Podobne zastosowanie występuje podczas redukcji aliasingu (por. 4.6).

Istnieje kilka podstawowych typów filtrów. Za najbardziej elementarny uważa się filtr dolnoprzepustowy, inaczej LPF (ang. low pass filter). Filtr ten ma za zadanie przepuszczać tylko dolne częstotliwości, tłumiąc częstotliwości powyżej pewnego progu (ang. cutoff frequency). Idealny filtr dolnoprzepustowy powinien przepuszczać wszystkie częstotliwości do progu, nie zmieniając ich i całkowicie tłumić częstotliwości powyżej progu. Okazuje się, że odpowiedzią impulsową takiego filtra jest funkcja sinc (rysunek 4.6):

$$sinc(x) = \begin{cases} \frac{\sin x}{x} & \text{dla } x \neq 0 \\ 1 & \text{dla } x = 0 \end{cases}$$

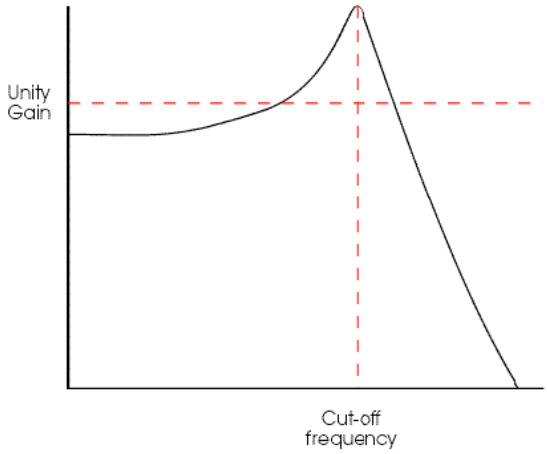


Rysunek 4.6: Wykres funkcji sinc, rysunek z [Sinc]

Niestety funkcja ta jest określona od -1 do 1, więc przeliczenie jej jest niemożliwe. Dlatego trzeba stosować funkcje, które tylko w przybliżeniu zachowują się jak idealny filtr dolnoprzepustowy. Jednym ze sposobów ich zbudowania jest stosowanie funkcji nieskończonej dla skończonego przedziału. Obcięcia nieskończonej funkcji można dokonać za pomocą różnych funkcji *okna*.

Analogicznym filtrem jest filtr górnoprzepustowy (ang. high pass filter), przepuszczający tylko górne częstotliwości. Filtry górnoprzepustowe często konstruuje się przekształcając filtr dolnoprzepustowy. Z kolei składając te dwa rodzaje filtrów w różny sposób otrzymamy filtr przepuszczający określone pasmo (ang. band pass filter) i filtr przepuszczający wszystko oprócz określonego pasma (ang. band reject filter).

Te podstawowe typy filtrów szczególnie są przydatne do skutecznego pozbywania się pewnych częstotliwości. W procesie kształtowania dźwięku często nie tyle jest ważna optymalność filtrów, co efekty, jakie można za ich pomocą uzyskać. Projektuje się więc filtry o ciekawszych charakterystykach, np. popularny w syntezie typ filtrów — dolnoprzepustowy z rezonansem (rysunek 4.7). Warto zaznaczyć, że wszystkie parametry filtra są modyfikowalne, czyli częstotliwość progu, wielkość rezonansu, a często nawet szybkość opadania.



Rysunek 4.7: Filtr dolnoprzepustowy z rezonansem, rysunek z [Rei6]

Filtry IIR

Filtry bazujące na odpowiedzi impulsowej, zwane FIR (ang. finite impulse response), w syntezie czasu rzeczywistego nie sprawdzają się, z uwagi na długi czas działania. Dlatego powstały filtry IIR (ang. infinite impulse response), o możliwie nieskończonej odpowiedzi impulsowej, bazujące na rekursji.

Obliczenie wartości n-tej próbki na wyjściu przedstawia poniższe równanie:

$$\begin{aligned} y[n] = a_0 x[n] \quad &+ \quad a_1 x[n \quad 1] + a_2 x[n \quad 2] + a_3 x[n \quad 3] + ... \\ &+ \quad b_1 y[n \quad 1] + b_2 y[n \quad 2] + b_3 y[n \quad 3] + ... \end{aligned}$$

gdzie y oznacza wyjście filtra, a x jego wejście. Odwołujemy się zarówno do poprzednich wartości z wyjścia, jak i z wejścia. Liczba współczynników a i b może być potencjalnie duża, ale zwykle nie jest większa niż 4. Właśnie w tym tkwi siła i szybkość filtrów rekurencyjnych. Zadowalające wyniki uzyskuje się już przy kilku mnożeniach i dodawaniach na obrót pętli.

Niebanalny jest natomiast dobór współczynników tak, aby uzyskach żądane rezultaty. Na szczęście można skorzystać z doświadczeń w projektowaniu filtrów analogowych. Filtry analogowe projektuje się z reguły w przestrzeni Laplace'a. Przejście do filtrów cyfrowych polega na przeprowadzeniu transformacji z do przestrzeni z, w której już jednoznacznie można wyznaczyć współczynniki (zob. [Smi99]).

Filtry IIR wydają się niezastąpione w przetwarzaniu sygnałów w czasie rzeczywistym dzięki swojej niezwykłej szybkości i niewiele gorszej efektywności, w porównaniu do filtrów FIR. Należy jednak uważać, gdyż błędy zmiennoprzecinkowe i rekurencyjny charakter tych procesów mogą doprowadzić do ich niestabilności.

4.4.2. Opóźnienie

Kolejnym przekształceniem liniowym wykorzystywanym w syntezie jest opóźnienie. W przeciwieństwie do filtrów nie modyfikuje ono częstotliwości sygnału, tylko jego czas. Najprostszy sposób wykorzystania opóźnienia zaobserwujemy w efekcie echa. Załóżmy, że tworzymy echo sygnału wejściowego w odległości N próbek (odległość w tym przypadku oznacza czas opóźnienia), o amplitudzie A. Oto kroki algorytmu:

1. Weź tablicę cykliczną o długości co najmniej N.
2. Dodaj do próbki wejściowej wartość z aktualnej pozycji w tablicy cyklicznej.
3. Przekaż wynik na wyjście.
4. Przeskaluj wynik o współczynnik A.
5. Wstaw wynik na pozycję aktualną $+N$.
6. Zwiększ aktualną pozycję o 1.

Punkty od 2 do 6 wykonywane są dla każdej wartości z wejścia. Zauważmy, że każda wartość z wejścia powraca na wyjście po N próbkach i z odpowiednią amplitudą, czyli w ten sposób, o który nam chodziło. Algorytm działa w czasie stałym względem jednej próbki i zajętości pamięci liniowej względem opóźnienia.

Pogłos

Opóźnienie można także zastosować w efektach pogłosu. Pogłos występuje w przyrodzie w bardzo wielu miejscach. Może on być naturalną charakterystyką akustyczną pomieszczenia, ale też efektem pudła rezonansowego skrzypiec. W najprostszym przypadku proces ten modelowany jest przez kilka pętli opóźniających, działających jednocześnie. W algorytmie echa wartość wyjściową będziemy wstawiać do tablicy w różnych odległościach i z różnymi współczynnikami skalowania. Dość skomplikowane środowiska akustyczne możemy symulować już przy użyciu czterech opóźnień. Zauważmy, że każde opóźnienie wróci na wyjście w różnym czasie i znów zostanie poddane kolejnym opóźnieniom. Odpowiada to mniej więcej wielokrotnym odbiciom w prawdziwych środowiskach akustycznych.

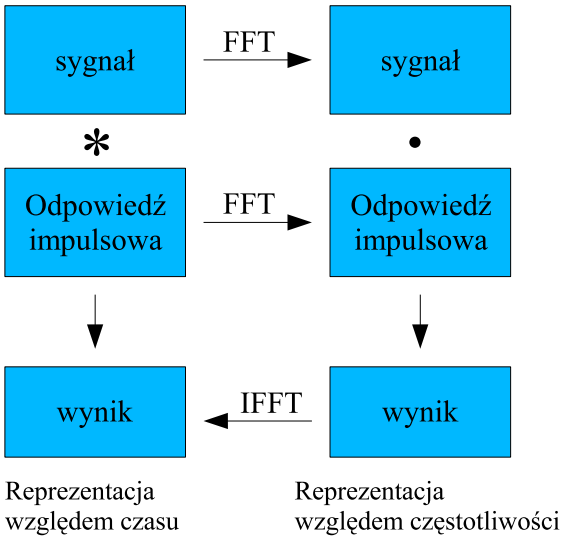
Problemem tej implementacji pogłosu jest brak kontroli nad charakterystyką otrzymanego środowiska. Parametryzacja głównie sprowadza się do określenia czasu tłumienia dźwięku. Kłopotliwe może również być właściwe dobranie czasów opóźnień tak, aby nie były ze sobą powiązane, nawet po wielokrotnej iteracji algorytmu, gdyż może to prowadzić do „dzwoniących artefaktów".

4.4.3. Splot FFT

Splot FFT jest szybką metodą obliczania splotu dwóch sygnałów. Wykorzystuje fakt, że w reprezentacji funkcji względem częstotliwości splot odpowiada operacji mnożenia. Proces ten przedstawia rysunek 4.8.

Sygnał i odpowiedź impulsowa układu liniowego przekształcane są do postaci częstotliwości poprzez dyskretną transformację Fouriera (FFT). Wyniki tych operacji są przez siebie mnożone (mnożenie liczb zespolonych). Tak powstały sygnał konwertowany jest do postaci czasowej w odwrotnej transformacji Fouriera ($IFFT$). Trzeba pamiętać, że wynik operacji splotu zajmuje $N + M$ miejsca, gdzie N i M to odpowiednio wielkość sygnału i odpowiedzi impulsowej. Dlatego zarówno sygnał, jak i odpowiedź impulsowa, musi być na wejściu powiększona do rozmiaru $N + M$ poprzez dodanie zer na koniec.

W powyższej metodzie musimy znać pełną postać sygnału. W przetwarzaniu dźwięku często mamy dostęp tylko do części sygnału. Możemy sobie z tym poradzić przetwarzając sygnał blokowo. Dla każdego bloku sygnału wejściowego robimy splot FFT z odpowiedzią impulsową układu. Ponieważ wynik tej operacji jest dłuższy od długości bloku, koniec sygnału wynikowego musimy dodać do wyników kolejnych kroków. Złożoność tej operacji to



Rysunek 4.8: Splot FFT

$O\left((N+M)\,log\,(N+M)\right)$ dla pojedynczego bloku, czyli $O\ \frac{N+M}{N} log(N+M)$ dla jednej wartości. Dlatego technikę tę stosuje się dla długich bloków lub krótkich odpowiedzi impulsowych.

W syntezie czasu rzeczywistego stosuje się nawet bloki długości 128, gdyż od tej wartości zależy opóźnienie powstającego dźwięku. Stosowanie odpowiedzi impulsowej sali koncertowej, sięgającej ponad 100000 wydaje się więc niepraktyczne.

Współcześnie często stosuje się splot FFT w połączeniu z odpowiedziami impulsowymi naturalnych środowisk w fazach produkcji muzyki, w których opóźnienia nie mają wielkiego znaczenia, np. w trakcie miksowania nagranego materiału.

4.5. Przekształcenia nieliniowe

Główną cechą przekształceń nieliniowych jest pojawianie się nowych częstotliwości w przetwarzanym sygnale. Pomimo wszystkich problemów, jakie często występują (nowe częstotliwości mogą powodować aliasing), przekształcenia nieliniowe przydają się, aby wzbogacić wejściowy sygnał.

Często spotykanym efektem jest przesterowanie sygnału. Wykorzystują go między innymi gitarzyści do wzmacniania brzmienia swoich gitar elektrycznych. Działanie efektu przedstawia wzór:

$$f(x) = \begin{cases} c & \text{dla } ax > c \\ -c & \text{dla } ax < -c \\ ax & \text{wpp.} \end{cases}$$

gdzie a to wmocnienie, a c wielkość graniczna. Sygnał jest najpierw wzmacniany przez wartość a, a później ograniczamy zakres wartości do przedziału $[-c, c]$. Powoduje to pojawienie się w sygnale wyjściowym dużej liczby nowych częstotliwości.

Żeby zmniejszyć liczbę dodatkowych częstotliwości, często stosuje się mniej rygorystyczne ograniczenia sygnału wyjściowego. Do tego celu doskonale nadają się wielomiany.

4.6. Zapobieganie efektowi aliasingu

Ponieważ aliasing jest efektem, który powoduje największe pogorszenie jakości dźwięku, stosuje się szereg metod, aby go ograniczyć:

- Filtrowanie sygnału wejściowego

  Sygnał wejściowy procesu przepuszcza się przez filtr dolnoprzepustowy. Dzięki temu w sygnale wyjściowym pojawi się mniej zbyt wysokich częstotliwości. Wadą tego rozwiązania jest zubożenie sygnału poprzez wycięcie również częstotliwości, nie powodujących aliasingu.

- Filtrowanie sygnału wyjściowego

  Podejście wprost nie jest skuteczne, ponieważ nie da się zlikwidować aliasingu, który już się pojawił. Dlatego zwiększa się częstotliwość próbkowania sygnału wejściowego dwukrotnie. Taki sygnał będzie miał teraz dwukrotnie większe pasmo do wykorzystania, a sam będzie zajmował maksymalnie połowę tego pasma. Po przetworzeniu sygnału stosuje się filtr dolnoprzepustowy, wycinający górną połowę pasma i zmienia się częstotliwość próbkowania na wejściową.

Rozdział 5

Prototyp silnika syntezy dźwięku

W rozdziale tym przedstawię prototyp silnika syntezy dźwięku, który posłużył mi do testowania wybranych algorytmów.

5.1. Założenia projektowe

5.1.1. Wtyczka do aplikacji muzycznych

Podstawowym założeniem projektu jest możliwość wykorzystania gotowego produktu w praktyce, do nagrywania muzyki. Ponieważ chciałem skoncentrować się na tworzeniu algorytmów, więc zrealizowałem system jako wtyczkę VST do macierzystych programów muzycznych. Istnieje wiele takich programów. Do testów wykorzystałem dwa darmowe: MiniHost (zob. [MiniHost]) i SAVIHost (zob. [SAVIHost]). Stworzenie wtyczki do programów macierzystych ma następujące zalety:

- Wieloplatformowość. Nie trzeba przejmować się różnicami w architekturach systemów operacyjnych.
- Obsługa klawiatur muzycznych MIDI. Klawiatury sterujące dają możliwość faktycznego grania na wirtualnym instrumencie.
- Wbudowany sekwencer. Sekwencer umożliwia nagranie zdarzeń MIDI i późniejsze ich odtworzenie. Można również edytować muzykę w zapisie nutowym lub innej notacji. Dla wtyczki nie ma różnicy, czy dźwięki grane są na żywo, czy odtwarzane za pomocą sekwencera.
- Nagrywanie audio. Istnieje możliwość nagrania wyjścia wirtualnego instrumentu w celu zmiksowania go z innym materiałem muzycznym, czy nagrania płyty CD.
- Zintegrowane środowisko muzyczne. W jednym programie mamy wszystko — komponowanie w zapisie nutowym, nagrywanie prawdziwych instrumentów audio, nagrywanie wielu wtyczek wirtualnych instrumentów.
- Interfejs użytkownika. Wiele wtyczek posiada swój własny interfejs do edycji parametrów, ale nie jest to wymagane. Dzięki ustandaryzowanej wymianie wartości parametrów między wtyczką a programem, program macierzysty potrafi sam stworzyć interfejs użytkownika dla wtyczek, które go nie posiadają.

- Automatyzacja parametrów. Wtyczki mają możliwość zmiany wartości parametrów w czasie rzeczywistym. Dzięki programowi macierzystemu zmiany te można nagrywać, edytować i później automatycznie odtwarzać.

Istnieje kilka standardów wtyczek do programów macierzystych. Zdecydowałem się użyć standardu VST, rozwijanego przez firmę Seinberg ([VST]). Jego podstawowymi cechami są:

- Najbardziej popularny standard wtyczek.
- Obsługiwany przez wszystkie macierzyste programy muzyczne.
- Dostępny dla wielu platform: Windows, Linux, Macintosh.
- Prosty interfejs programisty.

5.1.2. Rozszerzalność

Prototyp miał być systemem testowym. Jednak chciałem umożliwić dodawanie nowych algorytmów w prosty sposób i łączenie ich z już istniejącymi. Silnik systemu zajmuje się całą logiką potrzebną do współdziałania komponentów.

5.1.3. Polifonia

Algorytmy syntezy i przetwarzania dźwięku są z reguły monofoniczne. Oznacza to, że generują sygnał pojedynczego dźwięku. Stworzenie instrumentu grającego wiele dźwięków równocześnie polega w gruncie rzeczy na zduplikowaniu tych procesów i połączeniu ich równolegle tak, że każdy gra inny dźwięk. Żeby zadanie te nie było uciążliwe dla programisty, silnik powinien to robić automatycznie.

5.1.4. Szybkość

Bardzo istotne w syntezie dźwięku jest obciążenie procesora. Często w programach macierzystych używa się wielu wtyczek jednocześnie, zatem czas zabierany przez pojedynczą wtyczkę powinien być minimalny. Dlatego wszystkie procesy czasochłonne silnika, takie jak przydział pamięci, określenie kolejności procesów, duplikacja procesów, są przesunięte do momentu inicjacji instrumentu. Poszczególne procesy powinny być zoptymalizowane pod względem szybkości działania.

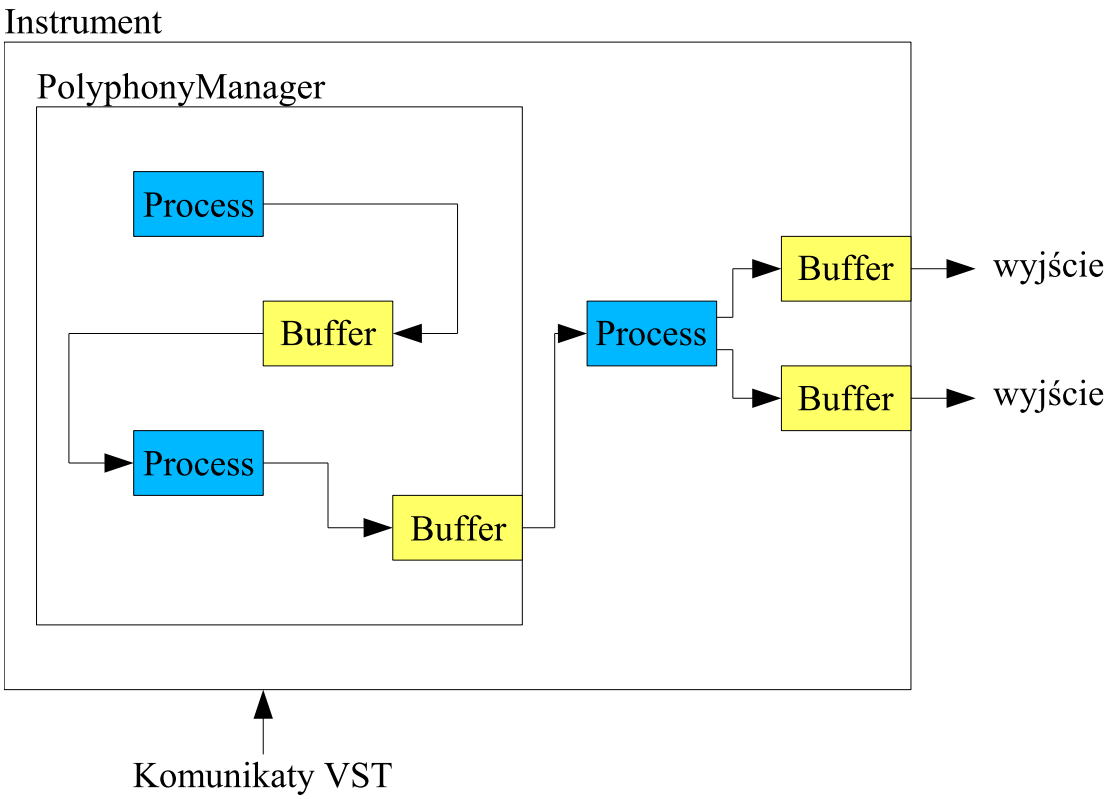
Oprócz tego bardzo ważną rzeczą jest opóźnienie przetwarzania dźwięku. Opóźnienie te jest czasem pomiędzy naciśnięciem klawisza na klawiaturze muzycznej, a słyszalnym efektem powodowanym przez te naciśnięcie. Niestety wtyczki nie mają całkowitego wpływu na ten czas reakcji. Dużo zależy od szybkości przetwarzania karty muzycznej. Rozmiar bufora karty muzycznej jest dolnym ograniczeniem na opóźnienie dźwięku. Ważne jest, aby wtyczka nie wprowadzała dodatkowych opóźnień poprzez wewnętrzne buforowanie dźwięku.

5.1.5. Jakość

Kluczowym czynnikiem wpływającym na jakość dźwięku jest jakość zaimplementowanych algorytmów. Algorytmy powinny być zoptymalizowane pod względem jakości. Powinno się stosować podwójną precyzję obliczeń w miejscach, w których ma to słyszalny wpływ na jakość. Dużą wagę należy przyłożyć do uniknięcia niepożądanych efektów, takich jak aliasing.

5.2. Architektura

Zajmę się teraz ogólną architekturą silnika syntezy dźwięku. Ilustruje ją rysunek 5.1.



Rysunek 5.1: Architektura silnika syntezy dźwięku

System został napisany w języku *C++*, z uwagi na wymagania szybkościowe i zgodność ze standardem VST.

Podstawowym pojęciem w systemie jest schemat syntezy. Podczas inicjacji instrumentu dodawane są do niego poszczególne procesy i bufory, tworząc sieć. Podczas syntezy instrument, posługując się schematem wywołuje poszczególne procesy, tworząc w ten sposób dźwięk. W dalszej części rozdziału opiszę role poszczególnych rodzajów komponentów: instrumentu, procesów, procesu zarządzającego polifonią i buforów.

5.2.1. Instrument

Instrument jest głównym elementem systemu. Implementuje on interfejs VST. Przestawię teraz podstawowe zadania klasy *Instrument*:

- Dodawanie procesów do schematu syntezy.
- Dodawanie buforów do schematu syntezy.
- Przekazywanie parametrów pomiędzy programem macierzystym a odpowiednim procesem.
- Przekazywanie zdarzeń VST do odpowiednich procesów.

- Ustalenie kolejności procesów w trakcie syntezy.
- Wykonanie procesu syntezy.

Instrument posiada kilka wyjść do programu macierzystego. Reprezentują je specjalne bufory, których pamięć nie jest alokowana przez wtyczkę. Wskaźnik na miejsce w pamięci ustawiany jest podczas syntezy i pochodzi z programu macierzystego.

Przekazywanie parametrów

Według standardu VST, instrument powinien przekazywać liczbę dostępnych parametrów i odpowiednio wartości, nazwy dla poszczególnych parametrów.

Instrument w prototypie silnika nie posiada scentralizowanej wiedzy o parametrach syntezy. Za to każdy z procesów udostępnia wiedzę o swoich parametrach. W momencie zapytania o parametr, instrument wyszukuje odpowiedni proces na podstawie numeru parametru i pozycji procesu na liście procesów. Zapytanie te jest przekazywane do znalezionego procesu.

Dzięki temu procesy mogą być pisane niezależnie, każdy zna tylko swoje parametry.

Przekazywanie zdarzeń VST

Zdarzenia VST to zdarzenia MIDI (pochodzące m.in. z klawiatury muzycznej) rozszerzone o automatyzację parametrów. Procesy rejestrują chęć odbierania zdarzeń w instrumencie poprzez metodę *addKeyListener*. W momencie wpłynięcia zdarzenia, zamieniane jest ono do wewnętrznej postaci i przekazywane wszystkim zainteresowanym procesom. Bardziej zaawansowaną obsługę zdarzeń posiada moduł odpowiedzialny za polifonię (*PolyphonyManager*).

Ustalenie kolejności procesów

Schemat syntezy jest siecią procesów i buforów. O sieci tej zakładam, że jest to graf skierowany acykliczny, czyli nie zawiera pętli. Oczywiście przetwarzanie sygnału może odbywać się w pętli, ale będzie ją tworzył pojedynczy proces. Przykładem może być pętla opóźniająca, czyli echo.

Ustalenie kolejności wykonania procesów polega na nadaniu im priorytetu, będącego liczbą naturalną. Większy priorytet oznacza, że proces musi zostać wykonany wcześniej w trakcie syntezie. Priorytet liczony jest jako maksymalna odległość w sensie liczby buforów od wyjścia instrumentu.

Przydzielanie priorytetów procesom odbywa się poprzez przeszukiwanie grafu wgłąb, począwszy od buforów wyjścia. Odpowiedzialna jest za to metoda *computePriority* w klasach *Instrument*, *Process* i *Buffer*.

Po przydzieleniu procesów są one sortowane na liście według algorytmu *quick sort*. Jest to standardowy algorytm biblioteki STL.

Wykonanie procesu syntezy

Proces syntezy jest wykonywany krokami. W każdym kroku instrument ma za zadanie zapisać sygnał do wszystkich buforów wyjściowych, dostarczanych przez program macierzysty. Liczba buforów odpowiada liczbie wyjść instrumentu i na ogół wynosi 2 dla instrumentów stereofonicznych. Długość buforów jest bezpośrednio związana z opóźnieniem generowanego dźwięku.

Na początku wskaźniki buforów programu macierzystego zostają przypisane do wewnętrznych buforów wyjścia silnika syntezy. Następnie czyszczone są bufory wewnętrzne, po czym następuje wywołanie przetwarzania dla wszystkich procesów po kolei.

5.2.2. Proces zarządzający polifonią

Proces zarządzający polifonią (*PolyphonyManager*) jest specjalnym typem procesu, bo łączy w sobie funkcjonalność zwykłego procesu i całego instrumentu. Jako że jest to zwykły proces, nie jest on obowiązkowy w schemacie syntezy, a nawet może być jego kilka instancji, choć z reguły jest jedna.

Głównym zadaniem tego procesu jest stworzenie polifonii na podstawie schematu syntezy oraz przekazywanie odpowiednich zdarzeń odpowiednim duplikatom schematu. Oprócz tego zawiera część funkcjonalności instrumentu, tzn. dodawanie procesów i buforów, określenie kolejności przetwarzania, zarządzanie parametrami i zdarzeniami.

Podczas tworzenia instancji zarządcy polifonią należy określić liczbę buforów wejściowych i wyjściowych. Bufory te są jedynym sposobem przekazywania sygnału między procesami wewnątrz zarządcy a tymi na zewnątrz. Dodatkowo bufory te nie są duplikowane. Oznacza to, że bufor wyjściowy będzie miał zmiksowany sygnał ze wszystkich duplikatów schematów syntezy.

Tworzenie duplikatów schematów syntezy

Żeby uzyskać polifonię, dla każdego głosu musi powstać jeden duplikat zbudowanego schematu syntezy. W tym celu, po ustaleniu kolejności przetwarzania poszczególnych procesów, tworzona jest tablica procesów rozmiaru będącego iloczynem wielkości polifonii przez liczbę procesów w schemacie. Można na nią spojrzeć jak na wiele tablic w ten sposób, że dla każdego głosu mamy tablicę z pełnym schematem syntezy. Podobnie tworzona jest tablica z buforami. Przejście od listy procesów do tablicy ma tu takie znaczenie, że teraz każdy proces będziemy identyfikować przez jego pozycję w tablicy.

Dla każdego bufora ze schematu znajdowane są wszystkie procesy wejściowe i wyjściowe. Każde z tych połączeń kopiowane jest według pozycji w tablicy do wszystkich głosów polifonii.

Na koniec sprawdzane są połączenia z buforami wejściowymi i wyjściowymi zarządcy polifonii i dołączane są do tych buforów procesy z kolejnych głosów.

Cały ten algorytm zawiera się w metodzie *setPolyphony*. Po wyjściu z metody zostanie tablica ze zduplikowanymi schematami procesów i buforów.

Zarządzanie parametrami

Na zewnątrz zarządcy polifonii ciągle chcemy widzieć tylko jeden schemat. Równoległe przetwarzanie dźwięku powinno mieć znaczenie tylko dla wewnętrznej reprezentacji. Dlatego do otoczenia przekazywane są parametry procesów tylko z oryginalnego schematu syntezy. Zakładamy, że analogiczne procesy z kolejnych głosów mają tak samo ustawione parametry.

W momencie żądania zmiany któregoś parametru, wyszukiwany jest na podstawie numeru parametru proces z oryginalnego schematu syntezy. Mając jego pozycję w tablicy procesów znajduję wszystkie analogiczne procesy we wszystkich głosach. Wszystkim znalezionym procesom ustawiana jest ta sama wartość parametrów.

Zarządzanie zdarzeniami muzycznymi

Zarządzanie zdarzeniami muzycznymi jest nieco inne. Interesują nas tutaj zdarzenia początek nuty i koniec nuty. Kluczowym parametrem tych zdarzeń jest numer nuty, czyli wysokość dźwięku. Istnieje kilka strategii rozdzielania tych zdarzeń. Postanowiłem kierować się następującymi założeniami:

- Może zdarzyć się sytuacja, że muzyk gra więcej dźwięków niż wartość polifonii. Oczywiście nie mając wystarczającej liczby głosów nie jesteśmy w stanie wszystkich dźwięków zagrać, ale algorytm mimo to powinien działać.
- Koniec nuty musi trafić do tego samego głosu, w którym był odpowiadający mu początek nuty.
- Dla początku nuty najlepiej wziąć głos, który miał najdawniej koniec nuty. Zwróćmy uwagę, że dla głosów, które dostały koniec nuty ma znaczenie, który wybierzemy. Zdarzenie końca nuty nie oznacza, że głos przestał grać, często mamy do czynienia z dość długimi zakończeniami dźwięków.
- Gdy nie mamy wolnego głosu, czyli takiego, który miał koniec nuty, wybieramy głos, który miał najdawniej początek nuty.

W ten sposób jesteśmy pewni, że zagramy wszystkie początki dźwięków. Jeśli coś utniemy, to będzie to dźwięk, który teoretycznie się zakończył lub najdawniej zaczął. Na ogół taki dźwięk ma w danej chwili najmniejsze znaczenie.

Do implementacji powyższych założeń posłużyły mi 3 struktury danych: tablica nuty granej przez każdy głos, kolejka głosów, które otrzymały koniec nuty i kolejka głosów, które otrzymały początek nuty.

W momencie otrzymania końca nuty, wyszukuję w tablicy głos, który ją gra. Jeśli taki głos został znaleziony, to wyjmuje go z kolejki początków nut, wkładam na koniec kolejki końców nut i wysyłam mu dane zdarzenie.

W momencie otrzymania początku nuty sprawdzam, czy jest jakiś głos w kolejce końców nut. Jeśli tak — wyjmuję pierwszy głos z tej kolejki, w przeciwnym wypadku wyjmuję pierwszy głos z kolejki początków nut. Wybrany głos trafia na koniec kolejki początków nut i zostaje mu wysłane dane zdarzenie.

5.2.3. Proces

Proces (*Process*) jest podstawowym elementem w systemie do implementacji algorytmów syntezy i przetwarzania dźwięków. Proces:

- zna częstotliwość próbkowania i wielkość bufora programu macierzystego,
- może udostępniać dowolną liczbę parametrów sterujących,
- może posiadać dowolną liczbę buforów wejściowych i wyjściowych,
- może reagować na zdarzenia początku nuty i końca nuty.

Zadaniem procesu w momencie syntezy jest wypełnienie wszystkich buforów wyjściowych sygnałem. Sygnały wejściowe mogą być wykorzystane w dowolny sposób. Na przykład w układzie wzmacniającym sygnał wyjściowy będzie po prostu sygnałem wejściowym pomnożonym przez liczbę. Z kolei w generatorze fali okresowej sygnał wejściowy może posłużyć do sterowania modulacją częstotliwości lub amplitudy.

Przykłady zaimplementowanych procesów znajdują się w p. 5.3.

5.2.4. Bufor

Bufor (*Buffer*) jest elementem służącym do przekazywania sygnałów pomiędzy procesami i do programu macierzystego. Rozmiar wszystkich buforów w silniku jest taki sam, jak deklarowany rozmiar buforu w programie macierzystym. Na ogół jest to ten sam rozmiar, co w buforze sterownika karty muzycznej i od niego zależy opóźnienie odtwarzania dźwięku.

W celu ograniczenia liczby buforów i czasu przetwarzania, bufor może mieć dowolną liczbę procesów wejściowych i wyjściowych. Każdy proces wejściowy dodaje własne wartości do wartości w buforze, dzięki czemu sygnały procesów wejściowych są automatycznie miksowane ze sobą. Ma to niestety jednak wadę, ponieważ przed rozpoczęciem przetwarzania bufor trzeba wyczyścić. Wiele procesów wyjściowych nie stwarza problemów, gdyż one tylko z bufora czytają.

5.3. Procesy

W części tej przedstawię zaimplementowane procesy, tj. moduły, które mogą być użyte do stworzenia syntezatora dźwięku.

5.3.1. SineGenerator

SineGenerator jest generatorem sygnału sinusa za pomocą funkcji *sinf*. Reaguje na zdarzenia klawiatury muzycznej. Posiada 3 opcjonalne bufory wejściowe do sterowania:

- 0 — sterowanie amplitudą sygnału. Przydatne do podłączenia generatora konturu.
- 1 — sterowanie zmianą częstotliwości, czyli jej modulacja.
- 2 — sterowanie zmianą amplitudy, czyli jej modulacja.

5.3.2. FastSineGenerator

FastSineGenerator jest szybszym generatorem sinusa za pomocą algorytmu 3 z p. 4.2.1 (zob. [Trig]). Proces ma tylko jeden bufor wejściowy, do sterowania amplitudą.

5.3.3. SawtoothGenerator

SawtoothGenerator jest prostym generatorem fali piłowej.

5.3.4. AdditiveSawtoothGenerator

AdditiveSawtoothGenerator generuje falę piłową za pomocą składowych harmonicznych. W konstruktorze klasy można określić maksymalną liczbę składowych harmonicznych, używanych do generowania fali.

Algorytm dodaje kolejne składowe harmoniczne do osiągnięcia ich maksymalnej liczby lub przekroczenia połowy częstotliwości próbkowania. Dzięki temu podczas generowania sygnału nie zachodzi aliasing.

5.3.5. WavetableSawtooth

WavetableSawtooth jest generatorem fali piłowej za pomocą tablicy fal. Tablica fal generowana jest dla 128 dźwięków standardu *MIDI*. W ten sposób dla wszystkich nut tworzone są optymalne fale. Dla częstotliwości pośrednich optymalność może nie być zachowana, ale są to sytuacje rzadsze. Możliwe jest podanie długości generowanych fal.

Aby proces inicjowania przebiegał możliwie szybko, generowanie zaczyna się od fali o najmniejszej liczbie składowych harmonicznych, czyli od dźwięku najwyższego. Kolejne, niższe dźwięki będą się różniły tylko dodatkowymi składowymi harmonicznymi. Dlatego każda następna nuta tworzona jest na podstawie poprzedniej z dołożonymi brakującymi składowymi harmonicznymi. Można tak zrobić, ponieważ dla każdego dźwięku generowany jest tylko jeden okres fali o stałej długości, nie zależącej od częstotliwości.

Przy odtwarzaniu fali w przypadku pozycji niecałkowitej wykorzystuje się interpolację liniową.

5.3.6. ADSREnvelopeGenerator

Generator konturu ADSR zależy od zdarzeń muzycznych: początek nuty i koniec nuty. Posiada cztery podstawowe parametry: czas ataku dźwięku, czas zanikania dźwięku, amplituda podtrzymywania dźwięku i czas wyciszania dźwięku.

Generator może znajdować się w pięciu stanach. Trzy stany stosowane są po wystąpieniu początku dźwięku. Stan czwarty odpowiada wyciszaniu dźwięku po napotkaniu końca nuty. Stan piąty to cisza w oczekiwaniu na kolejną nutę.

Zauważmy, że początek kolejnej nuty może wystąpić przed kompletnym wyciszeniem dźwięku. Pokazuje to rysunek 5.2. W pierwszym przypadku po napotkaniu początku nuty, kontur jest zerowany i generowany od początku. W moim procesie zastosowałem strategię z drugiego przypadku. Po napotkaniu początku nuty sygnał generowany jest od tego samego miejsca, w którym był zakończony. Krzywizna sygnału jest zachowana, w związku z tym może to potencjalnie skrócić czas ataku.

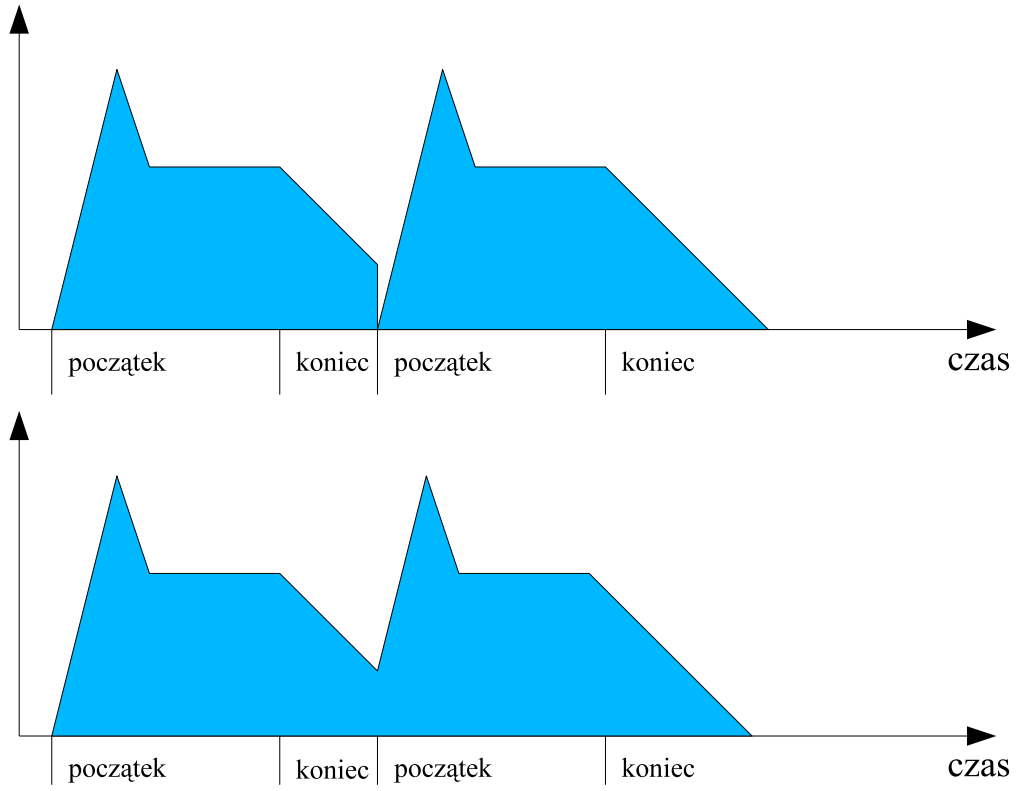
5.3.7. KarplusStrongPluckedString

KarplusStrongPluckedString implementuje algorytm struny szarpanej Karplus-Stronga, opisany w p. 4.2.4. Algorytm bazuje na cyklicznie odtwarzanej tablicy z losową falą. Tablica tworzona jest w momencie inicjacji dla maksymalnej długości fali, czyli minimalnej częstotliwości. Algorytm reaguje na zdarzenie początku nuty. Po otrzymaniu tego zdarzenia ustawiana jest aktualna długość tablicy na długość okresu dźwięku danej nuty. W ramach optymalizacji, zamiast wypełnienia tablicy wartościami, proces ustawia flagę początku nuty. Generowanie wartości losowych występuje dopiero podczas pierwszego okresu fali.

Wartości losowe tworzone są przez złożenie sześciu wartości generatora liczb pseudolosowych. Dzięki temu uzyskuję rozkład bardziej zbliżony do normalnego. Do generowania liczb pseudolosowych używam algorytmu *Mersenne twister* z biblioteki [Fog].

5.3.8. DrawbarOrgan

DrawbarOrgan jest przykładem syntezy addytywnej w postaci organów, popularnego instrumentu muzyki rozrywkowej. Dźwięk w organach powstaje z połączenia dziewięciu składowych harmonicznych. Każda z tych składowych ma ustawialną amplitudę. W klasycznych instrumentach tego typu mamy do dyspozycji następujące składowe: pierwszą, trzecią, dru-



Rysunek 5.2: Dwie strategie zachowania konturu ADSR w przypadku wystąpienia początku nuty przed kompletnym wyciszeniem dźwięku

gą, czwartą, szóstą, ósmą, dziesiątą, dwunastą i szesnastą. Trzecia składowa z nie do końca jasnych przyczyn występuje przed drugą.

Implementacja jest modułem złożonym. Do generowania poszczególnych składowych wykorzystuje szybkie generatory sinusów *FastSineGenerator*. Jako parametry udostępnia amplitudy poszczególnych składowych i głośność ogólną.

5.3.9. MultiGenerator

MultiGenerator jest przykładem rodzaju syntezy addytywnej, używanego m.in. w syntezatorach sekcji smyczkowych. W tym przypadku wynikiem syntezy jest suma wielu bogatych harmonicznie fal składowych. Każda z fal ma lekko zmienioną częstotliwość i ustawienie w panoramie stereofonicznej. Powoduje to efekt podobny do orkiestry, w której gra wiele takich samych instrumentów, ale każdy jest w innym miejscu i żaden nie jest idealnie nastrojony.

W module jako falę składową zastosowałem generator fali piłowej. W konstruktorze można określić liczbę głosów, jakiej chcemy użyć. Każdy głos dostaje dwa parametry: rozstrojenie od częstotliwości bazowej i ustawienie w panoramie stereofonicznej.

5.3.10. Delay

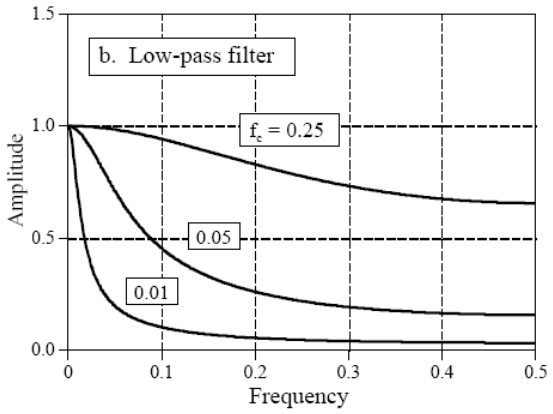
Delay jest prostą pętlą opóźniającą, tworzącą efekt echa. Zawiera jeden wewnętrzny bufor cykliczny, pamiętający stare wartości sygnału. Efektem można sterować za pomocą dwóch parametrów: czas opóźnienia echa i relatywna głośność echa względem sygnału.

5.3.11. Reverb

Reverb jest prostym, stereofonicznym efektem pogłosu pomieszczenia. Implementacja posiada dwa wewnętrzne bufory, oddzielne dla obu kanałów wyjściowych. Dla każdego kanału tworzone są trzy echa sygnału, o opóźnieniach niezależnych względem siebie. Lewe i prawe echa wzajemnie się przeplatają w ten sposób, że echem echa lewego jest echo prawe, a echem echa prawego jest echo lewe. W ten sposób otrzymujemy bardzo dużo małych ech sygnału, co symuluje wielokrotne odbicia od ścian pomieszczenia.

5.3.12. SinglePoleLPFilter

SinglePoleLPFilter jest najprostszym rekurencyjnym filtrem dolnoprzepustowym, pamiętającym tylko jedną wartość wstecz. Na podstawie częstotliwości filtrowania wyliczane są tylko dwa współczynniki, więc nie gwarantuje on zbyt dobrej efektywności. Rysunek 5.3 pokazuje działanie filtru dla różnych częstotliwości filtrowania.



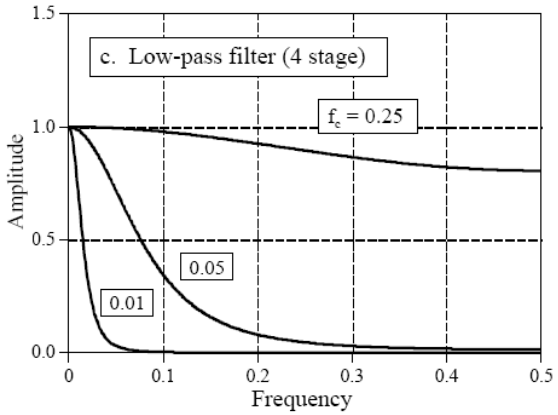
Rysunek 5.3: Działanie filtru *SinglePole* dla różnych częstotliwości filtrowania, rysunek z [Smi99]

5.3.13. FourStageLPFilter

FourStageLPFilter jest filtrem dolnoprzepustowym odpowiadającym poczwórnemu złożeniu szeregowemu najprostszych filtrów. Składanie szeregowe filtrów może być realizowane wprost, tzn. przepuszczamy sygnał przez cztery filtry lub, tak jak jest w tym przypadku, poprzez wyliczenie współczynników pojedynczego filtra, powstałego ze złożenia tych filtrów. Więcej na ten temat można znaleźć w [Smi99]. Działanie tego filtra prezentuje rysunek 5.4.

5.3.14. Peak4poleLPFilter

Peak4poleLPFilter jest rekurencyjnym filtrem dolnoprzepustowym z rezonansem. Rezonans oznacza możliwość wzmocnienia zakresu częstotliwości przed częstotliwością filtrowania. Filtr odwołuje się do czterech wcześniejszych wartości sygnału. Sposób obliczenia współczynników został zaczerpnięty z [musicDSP]. Parametry filtru to częstotliwość filtrowania i poziom rezonansu.



Rysunek 5.4: Działanie filtru *FourStage* dla różnych częstotliwości filtrowania, rysunek z [Smi99]

5.3.15. Distortion

Distortion jest prostym efektem przesterowania sygnału. Jako że jest to przekształcenie nieliniowe, powstaje nieskończenie wiele nowych składowych częstotliwości. Procesem sterują dwa parametry: wzmocnienie i odcięcie. Wzmocnienie powoduje zwiększenie amplitudy sygnału, nawet o 20 dB. Odcięcie powoduje ograniczenie wartości sygnału. Wszystkie wartości powyżej odcięcia dostają wartość odcięcia.

5.3.16. ChebyshevPolynomialDistortion

Przesterowanie za pomocą wielomianów Czebyszewa jest przykładem przekształcenia nieliniowego o ograniczonej liczbie nowych składowych częstotliwości. Sygnał wyjściowy tworzony jest za pomocą rekurencyjnej postaci funkcji Czebyszewa:

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_{n+1}(x) &= 2xT_n(x) \quad T_{n-1}(x) \end{aligned}$$

Do sygnału wyjściowego wchodzi n pierwszych wartości wielomianów. Wartość wielomianu zerowego jest pomijana. Wartości pozostałych wielomianów skalowane są przez wartości odpowiednich amplitud, które są parametrami procesu. W konstruktorze procesu można sprecyzować liczbę generowanych wielomianów. Złożoność algorytmu jest liniowa względem liczby wielomianów.

Wielomiany Czebyszewa można zdefiniować także w postaci trygonometrycznej:

$$T_n(\cos(\theta)) = \cos(n\theta)$$

Z powyższej postaci wynika, że dla pojedynczej składowej częstotliwości, n-ty wielomian tworzy składową o częstotliwości n-krotnej. Wynika z tego, że wielomiany tworzą tylko składowe harmoniczne, ograniczone stopniem wielomianu. Są więc dobrym sposobem na ograniczenie wpływu aliasingu na sygnał wyjściowy i większą kontrolę nad sygnałem wyjściowym, ponieważ każdy wielomian ma regulowaną amplitudę.

5.3.17. FFTConvolution

Splot FFT jest bardzo ogólnym procesem. Można w ten sposób zamodelować dowolne przekształcenia liniowe, zarówno filtry, jak i środowiska akustyczne. Implementacja działa w blokach o długości bloków przetwarzania, otrzymanej z programu macierzystego. Dla małych długości bloków algorytm ma dużą złożoność, więc może być niestabilny.

Do przeprowadzenia splotu potrzebna jest próbka dźwiękowa z odpowiedzią impulsową układu. Proces czyta pliki typu *wav*, zarówno monofoniczne, jak i stereofoniczne. W zależności od rodzaju pliku produkowany jest jeden lub dwa sygnały wyjściowe. Pliki stereo przydatne są w symulowaniu pogłosów pomieszczeń, zaś mono do tworzenia filtrów lub symulacji brzmienia pudła rezonansowego bądź zestawów nagłaśniających.

Do dyskretnej transformacji Fouriera algorytm używa biblioteki [fftw].

5.3.18. Panning

Panning jest prostym procesem pomocniczym. Jego zadaniem jest stworzenie z sygnału mono sygnału stereo. Za pomocą parametru *pan* można określić w jakich proporcjach sygnał wejściowy jest rozbijany na bufory wyjściowe.

5.4. Możliwe rozszerzenia systemu

Prototyp silnika syntezy dźwięku nie jest systemem kompletnym. Możliwe są zarówno rozszerzenia związane z działaniem samego silnika, jak i rozszerzenia zaimplementowanych modułów. Ponieważ liczba algorytmów procesów do zaimplementowania w syntezie dźwięku jest w zasadzie nieograniczona, w części tej omówię rozszerzenia związane z samym silnikiem syntezy.

5.4.1. Rozszerzenie parametrów

W zaimplementowanym systemie każdy moduł ma stałą liczbę niezależnych parametrów. Takie założenie dobrze pasuje do prototypu. Łatwo jest w ten sposób implementować i testować nowe algorytmy. Jednak w końcowym instrumencie muzycznym często przydają się parametry nie związane z konkretnym modułem, związane z wieloma modułami lub zależne od siebie. Niektóre parametry modułów chciałoby się ukryć lub sterować nimi pośrednio.

W prototypie istnieją przykłady tworzenia takiej warstwy pośredniej. Są nimi procesy: *DrawbarOrgan* i *MultiGenerator*. Są to procesy złożone, składające się z innych procesów i na własne potrzeby wprowadzają nowe parametry, za pomocą których sterują danymi procesami.

Bardziej ogólnym sposobem obejścia powyższych problemów jest stworzenie dodatkowej warstwy meta-parametrów. Warstwa ta pośredniczyłaby pomiędzy użytkownikiem a parametrami procesów.

5.4.2. Zapis ustawień

Jedną z możliwości standardu VST jest możliwość zapisu ustawień parametrów, czyli tak zwanych programów. W wirtualnych instrumentach muzycznych jest to funkcja bardzo przydatna, gdyż pozwala łatwo przywoływać ulubione ustawienia, a program macierzysty może pamiętać ustawienia wszystkich wtyczek danego projektu. Dzięki temu otwierając projekt od razu można rozpocząć pracę w miejscu, w którym się skończyło.

5.4.3. Automatyzacja parametrów

Czasem przydatną funkcją standardu VST jest automatyzacja parametrów. Można zapisać zmiany parametrów w trakcie grania na instrumencie wirtualnym, a później je odtwarzać. Niestety odtwarzanie zmian parametrów działa poprzez mechanizm zdarzeń. Zdarzenia mają stempel czasowy i nie da się automatycznie wprowadzać zmian w poszczególnych procesach w odpowiednim czasie. Proces sam wie, w którym miejscu bufora przetwarzania aktualnie się znajduje i w odpowiednim momencie musi nanieść zmiany.

Ponieważ nie jest to funkcjonalność kluczowa, zaimplementowane procesy nie reagują na zdarzenia zmian parametrów.

5.4.4. Interfejs użytkownika

Standard VST zapewnia stworzenie interfejsu użytkownika dla parametrów wtyczki przez program macierzysty. Jednak aby oddać indywidualny charakter instrumentu, często tworzy się specjalny interfejs dla wtyczki. Umożliwia nam to dowolne rozmieszczenie parametrów, stosowanie własnych kontrolek, np. pokręteł, suwaków i przełączników.

Tylko z własnego interfejsu użytkownika możemy umożliwić wybór pliku. Taka funkcja szczególnie jest przydatna w procesach posługujących się plikami, takich jak splot FFT. Bez własnego interfejsu nazwę pliku trzeba zaszyć w kodzie instrumentu lub w pliku konfiguracyjnym.

Do tworzenia interfejsów użytkownika istnieje oddzielny standard VSTGUI. Głównymi zaletami tego standardu jest prostota i przenośność.

5.4.5. Tworzenie schematów syntezy

W implementacji schematy syntezy budowane są na sposób programistyczny. Nie jest to specjalnie trudne i daje pełnię możliwości. Czasem mogłaby się przydać możliwość skonstruowania takiego schematu w pliku konfiguracyjnym. W ten sposób silnik, po przeczytaniu pliku, generowałby schemat automatycznie, bez konieczności kompilowania całego instrumentu.

Jest to jednak funkcjonalność poboczna, gdyż głównym założeniem było konstruowanie wolnostojących wtyczek VST, bez konieczności posługiwania się dodatkowymi plikami konfiguracyjnymi.

Rozdział 6

Opis przykładów

W rozdziale tym przedstawię przykłady napisane w moim silniku syntezy dźwięku.

6.1. Sposób uruchamiania przykładów

Przykłady znajdują się na płycie CD w postaci skompilowanych wtyczek VST. Są to pliki `ex1.dll`, ..., `ex8.dll`. Do ich uruchomienia można użyć jednego z dwóch dołączonych macierzystych programów muzycznych. Oba mają możliwość grania za pomocą myszy na ekranie komputera, za pomocą klawiatury komputerowej lub połączonej za pośrednictwem portu MIDI klawiatury muzycznej. Używając myszy można sterować parametrami wtyczek, przesuwając suwaki na ekranie.

6.1.1. SAVIHost

SAVIHost (zob. [SAVIHost]) został stworzony z myślą o uruchamianiu dokładnie jednej wtyczki. Każdemu przykładowi odpowiada plik wykonywalny o tej samej nazwie, uruchamiający program SAVIHost właśnie z tą wtyczką.

6.1.2. MiniHost

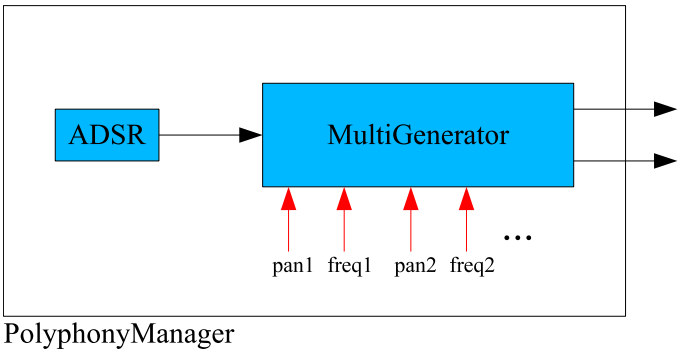
MiniHost (zob. [MiniHost]) znajduje się w oddzielnym katalogu na płycie. Za jego pomocą można uruchomić wszystkie przykłady. Wybór wtyczki dokonujemy poprzez menu *VST* i pozycję *Load VST Plugin...*. W wyświetlonym oknie dialogowym należy wybrać odpowiedni plik z wtyczką.

MiniHost wymaga sterowników karty dźwiękowej ASIO. Jeśli dana karta dźwiękowa nie posiada takiego sterownika, to można użyć *Asio4All* (zob. [Asio4All]) — uniwersalnego sterownika ASIO, znajdującego się również na płycie.

6.2. Przykład 1

Przykład pokazuje syntezę addytywną kilku fal piłowych, o lekko zróżnicowanych częstotliwościach.

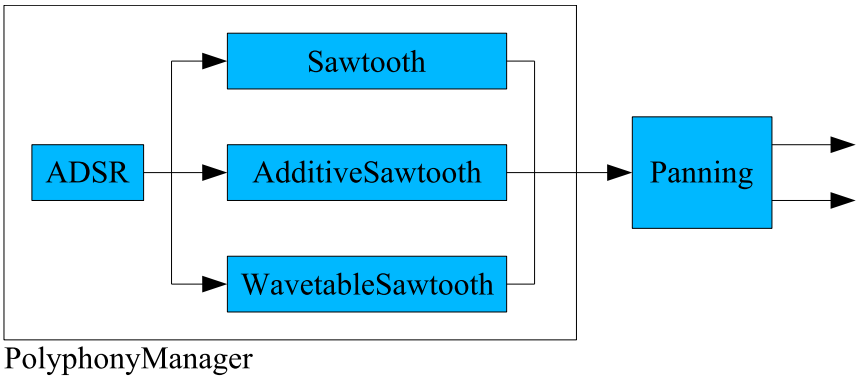
Schemat syntezy obrazuje rysunek 6.1. Za kształtowanie amplitudy w czasie odpowiada generator konturu ADSR. *MultiGenerator* generuje szereg fal piłowych. Dla każdej fali są dwa parametry: ustawienie w panoramie stereofonicznej (*pan*) i zmiana częstotliwości względem częstotliwości bazowej dźwięku (*freq mult*).



Rysunek 6.1: Przykład 1

6.3. Przykład 2

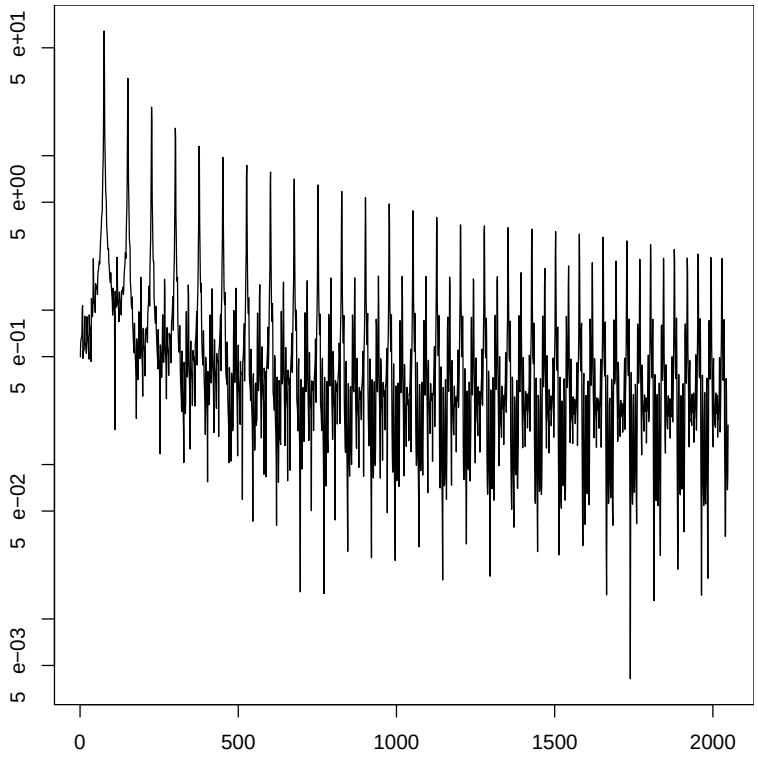
W przykładzie porównane są sposoby generowania fali piłowej.



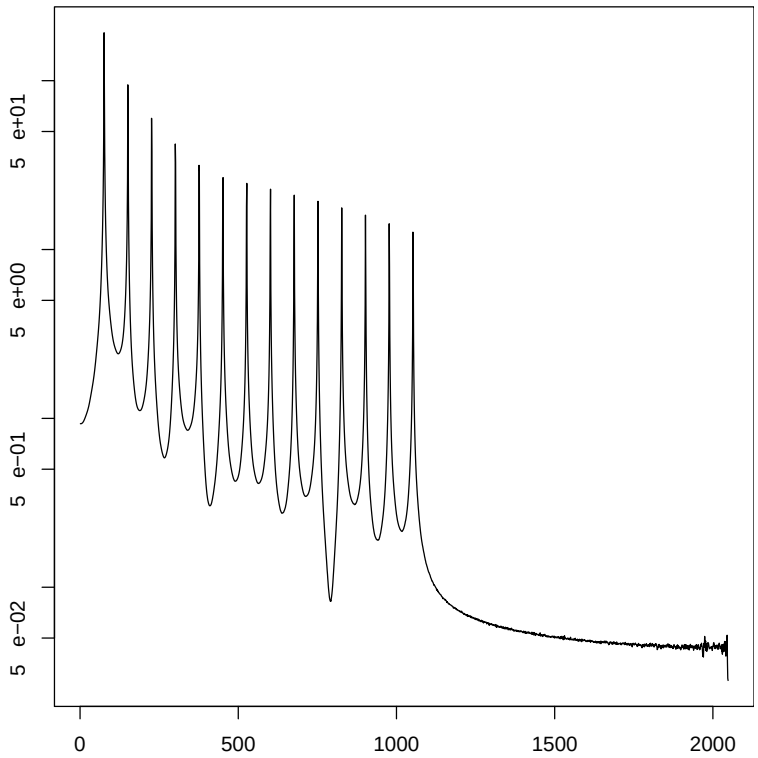
Rysunek 6.2: Przykład 2

W schemacie syntezy (rysunek 6.2) znajdują się 3 generatory fali piłowej, połączone równolegle. Aby usłyszeć działanie konkretnego generatora należy ustawić jego amplitudę (*amp*) na wartość niezerową, a amplitudy pozostałych generatorów na wartość 0. Parametry generatorów występują w następującej kolejności:

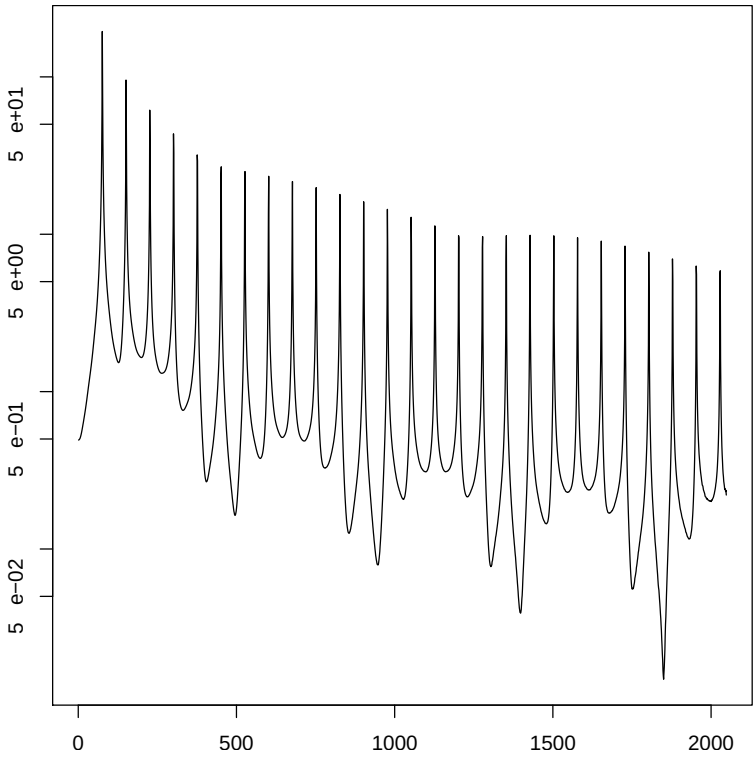
1. Prosty generator piły.
2. Generator addytywny. Fala powstaje jako suma sinusów do 14 składowych harmonicznych.
3. Generator z użyciem tablicy fal. Wartości fal są wyliczane podczas inicjacji w ten sposób, że zawierają wszystkie składowe harmoniczne, mieszczące się w paśmie dźwięku (połowa częstotliwości próbkowania).



Rysunek 6.3: SawtoothGenerator



Rysunek 6.4: AdditiveSawtoothGenerator



Rysunek 6.5: WavetableSawtoothGenerator

Wykresy 6.3, 6.4, 6.5 pokazują fale produkowane przez te generatory względem częstotliwości. Oś pionowa ma skalę logarytmiczną, żeby lepiej było widać kolejne składowe harmoniczne.

Prosta fala piłowa ma najbogatszy wykres. Kolejne składowe harmoniczne mają coraz mniejsze amplitudy. Po dojściu do częstotliwości maksymalnej składowe się nie kończą. Jednak w sygnale nie może być wyższych częstotliwości, więc zachodzi aliasing. Na rysunku wyraźnie widać, jak kolejne składowe o coraz mniejszych amplitudach zostały zawinięte i ich częstotliwości zaczęły maleć, począwszy od maksymalnej częstotliwości. Na tym się proces nie kończy, wyższe częstotliwości zostają znowu „odbite" od częstotliwości 0 itd.

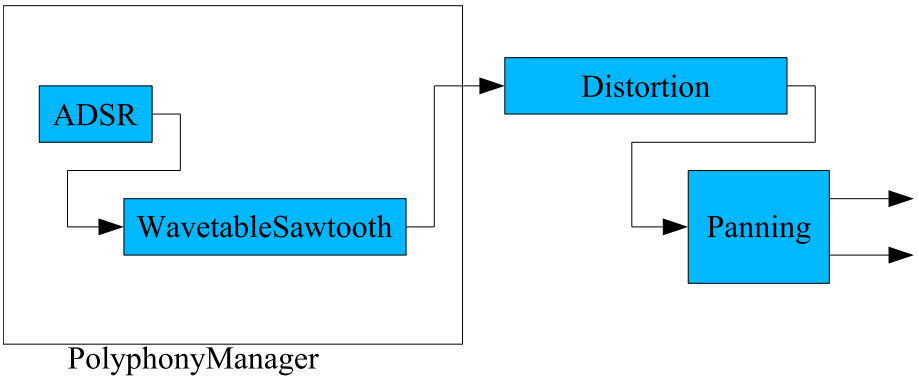
Addytywna fala piłowa ma z kolei najuboższy wykres. Wyraźnie widać 14 tworzących ją składowych harmonicznych. Nie ma prawa zachodzić tu aliasing, gdyż nawet dla wyższych częstotliwości tworzone są tylko składowe mieszczące się w paśmie.

Fala tworzona za pomocą tablicy fal ma idealną postać do syntezy dźwięku. Wykres zawiera wszystkie składowe harmoniczne, mieszczące się w paśmie dźwięku i tylko te. Ostatnia składowa wyraźnie wypada w okolicach maksymalnej częstotliwości i nie zachodzi aliasing.

6.4. Przykład 3

Przykład 3 pokazuje działanie efektu przesterowania dźwięku (zob. p. 5.3.15).

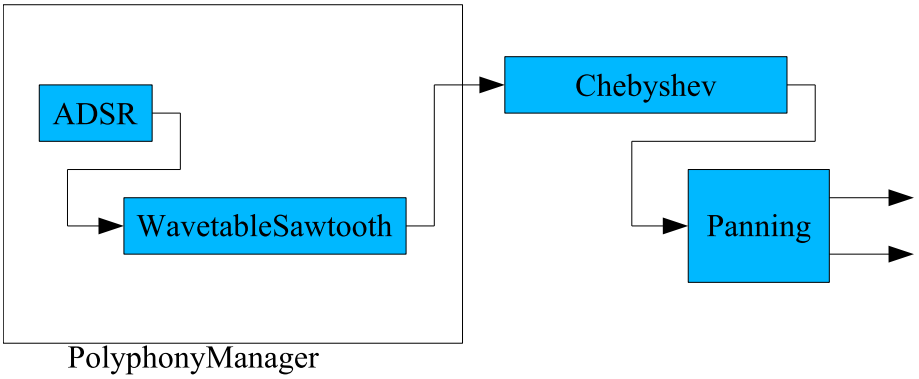
Schemat syntezy obrazuje rysunek 6.6. Wpływem efektu przesterowania na dźwięk steruje głównie parametr wzmocnienia (gain). Zwiększając wartość tego parametru powodujemy wzrost liczby wysokich składowych harmonicznych.



Rysunek 6.6: Przykład 3

6.5. Przykład 4

Przykład 4 pokazuje bardziej skomplikowane przekształcenie nieliniowe, oparte na wielomianach Czebyszewa (zob. p. 5.3.16).

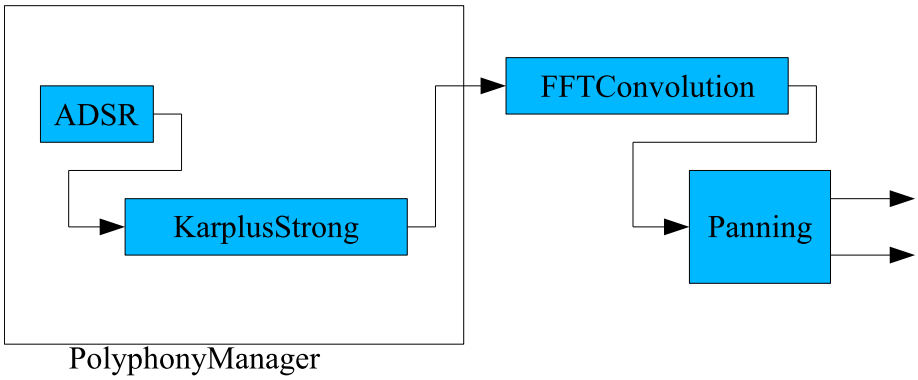


Rysunek 6.7: Przykład 4

Schemat syntezy obrazuje rysunek 6.7. Użytkownik ma możliwość sterowania amplitudami wyników kolejnych wielomianów za pomocą parametrów gain 1 — 5.

6.6. Przykład 5

Przykład 5 pokazuje działanie syntezatora struny szarpanej Karplus-Stronga (zob. p. 5.3.7) i splotu FFT.

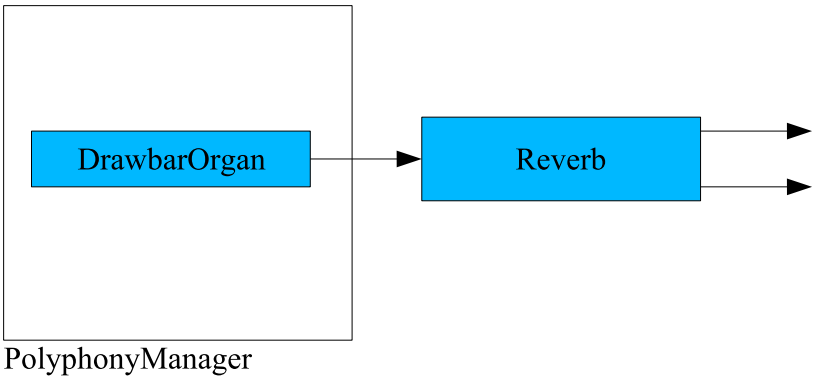


Rysunek 6.8: Przykład 5

Schemat syntezy obrazuje rysunek 6.8. Sygnał powstający w generatorze struny jest splatany z próbką dźwiękową odpowiedzi impulsowej wzmacniacza gitarowego.

6.7. Przykład 6

Przykład 6 symuluje organy jako syntezę addytywną sinusów (zob. p. 5.3.8).



Rysunek 6.9: Przykład 6

Schemat syntezy pokazuje rysunek 6.9. Brzmieniem organ steruje dziewięć parametrów, odpowiadających dziewięciu składowym harmonicznym. Dodatkowo na końcu syntezatora dodałem prosty pogłos (zob. p. 5.3.11).

6.8. Przykład 7

Przykład 7 pokazuje wpływ filtrowania na generowany sygnał.

Schemat syntezy obrazuje rysunek 6.10. Sygnał tworzy generator piły. Później jest on kształtowany przez filtr dolnoprzepustowy z rezonansem (zob. p. 5.3.14). W filtrze można ustawić częstotliwość rezonansu (*peak freq*), będącą bezpośrednio przed częstotliwością filtrowania i poziom rezonansu (*peak g*). Ciekawe efekty można uzyskać przy dużym poziomie rezonansu i częstotliwości filtrowania bliskiej częstotliwości bazowej dźwięku.

Rysunek 6.10: Przykład 7

Dodatkowo podłączony został efekt echa (zob. p. 5.3.10). Jako parametry udostępnia głośność sygnału powtarzanego (decay) i opóźnienie sygnału (delay).

6.9. Przykład 8

Przykład pokazuje modulację amplitudy i częstotliwości sinusa.

Rysunek 6.11: Przykład 8

Schemat syntezy obrazuje rysunek 6.11. Sygnał tworzony jest za pomocą trzech generatorów sinusa. Każdy generator ma ustawialną częstotliwość sygnału (freq) oraz amplitudę sygnału (amp). Na liście wystąpują parametry w następującej kolejności:

1. Parametry generatora sygnału sużącego do modulacji amplitudy.
2. Parametry generatora sygnału sużącego do modulacji częstotliwości.
3. Parametry sygnału nośnika.

Za pomocą tego przykładu można porównać różnice w modulacji sygnału za pomocą niskich i wysokich częstotliwości. Dla wysokich częstotliwości wyraźnie pojawiają się dodatkowe składowe harmoniczne. Wpływ na uzyskiwane efekty mają również amplitudy modulatorów.