თბილისის სახელმწიფო უნივერსიტეტის ზუსტ და საბუნებისმეტყველო მეცნიერებათა ფაკულტეტი

კონსტანტინე კუტალია

ყველა-ერთში გიტარის სტუდია Web Audio-ის გამოყენებით

ინფორმაციული ტექნოლოგიების სამაგისტრო პროგრამა

ინფორმაციული ტექნოლოგიების მაგისტრი

სამაგისტრო ნაშრომის ხელმძღვანელი - ლელა მირცხულავა, დოქტორი, ასოცირებული პროფესორი

თბილისი

2021

სარჩევი

Annotation

Electric guitar is one of the most popular instrument. You have heard it on countless pop, jazz and rock records. Those iconic sounds are generated by electric devices known as amplifiers. Without them electric guitar can be hardly heard even in the room. First generations of guitar amplifiers used vacuum tubes to amplify signals. In 1970s when solid state technology became widespread transistor-based guitar amplifiers were popularized, yet many players would still prefer vacuum tube sounds as they sound warmer and fuller. At the end of 1990s digital simulations of both kinds of amplifiers appeared in a form of Line 6 POD guitar effects processor. It spawned a great interest due to its versatility, low cost, lack of need of maintenance and many factors but its sound quality was generally considered worse to analog counterparts. During 2000s many iterations of digital simulations appeared on various platforms. By the end of 2010s digital solution began to approach new levels thanks to products like Kemper Profiling Amp which could automatically generate new realistic amplifier simulations using user-supplied audio signals. It found home in the studios of top producers. Just few years ago first web browser-driven amplifier simulations appeared which used Web Audio API. They would try to make electric guitar playing through a computer a bit easier since native applications require setup and are platform-dependent. Since then not much has been done building upon this idea whereas Web Audio API has been improved. This paper aims to work on the aforementioned issue and offers a solution in a form of web-based native-like application. By itself it is a demonstration to use the latest achievements of web-technologies of both UI development and audio processing in an unified manner.

ანოტაცია

ელექტრო გიტარა ერთ-ერთი ყველაზე პოპულარული ინსტრუმენტია. მათი ხმა გსმენიათ ურიცხვ პოპ, ჯაზ თუ როკ ჩანაწერზე. ეს ლეგენდარული ხმები წარმოიქმნება ხმის გამაძლიერებლის სახელით ცნობილი ელექტრო მოწყობილობით. მათ გარეშე ელექტრო გიტარის გაგონება თითქიმს შეუძლებელი იქნებოდა. გიტარის გამაძლიერებლების პირველი თაობები სიგნალის გასაძლიერებლად იყენებდნენ ვაკუუმის ნათურებს. 1970-იანებში, როცა ტრანზისტორული ტექნოლოგია ფართოდ დამკვიდრდა, ტრანზისტორის ხმის გამაძლიერებლებიც პოპულარული გახდა, თუმცა ბევრი გიტარისტი კვლავ ნათურების სავსე და თბილ ჟღერადობას ამჯობინებდა. 1990-იანების ბოლოს ორივე სახის ციფრული სიმულაციები გამოჩნდა Line 6 POD გიტარის ეფექტების პროცესორის სახით. ამან თავისი მოხერხებულობის, სიიაფის, მოვლის აუცილებლობის არარსებობის და სხვა ფაქტორების გამო დიდი ინტერესი გააღვივა, თუმცა მისი ხმა, ზოგადად, ანალოგურზე უარესად მიიჩნეოდა. 2000-იანებში ციფრულ სიმულაციათა უამრავი სახესხვაობა გამოჩნდა სხვადასხვა პლატფორმაზე. 2010-იანების ბოლოს ციფრულმა სიმულაციებმა ახალ დონეს მიაღწიეს, ეს კი ისეთი პროდუქტების დამსახურებაა, როგორიცაა Kemper Profiling Amp, რომელსაც ავტომატურად შეუძლია ახალი გამაძლიერებლის სიმულაციის წარმოქმნა მომხმარებლის მიერ მიწოდებული ხმოვანი სიგნალების მიხედვით. მან თავი მრავალი საუკეთესო პროდიუსერის სტუდიაში დაიმკვიდრა. მხოლოდ რამდენიმე წლის წინ პირველი ვებ-ბრაუზერში ჩაშენებული გამაძლიერებლის სიმულაცია გამოჩნდა, რომლებიც Web Audio API-ის იყენებდნენ. ისინი ცდილობდნენ კომპიუტერით ელექტრო გიტარის დაკვრა ცოტათი უფრო მოსახერხებელი გაეხადათ, რადგან ადგილობრივ აპლიკაციებს სჭირდებათ დაყენება და პლატფორმაზე დამოკიდებულები არიან. ამის შემდეგ ამ იდეის განსახორციელებლად ბევრი რამ არ გაკეთებულა, ამასობაში კი Web Audio API დაიხვეწა. ეს ნაშრომიც სწორედ ზემოთ აღწერილ საკითხზე მუშაობას ისახავს იმზნად. ძირითადი შედეგები პასუხობს ამ მიზანს ადგილობრივთან მიახლოებული ვებ აპლიკაციის შემოთავაზების სახით. თავისთავად სიახლეს წარმოადგენს ვებ-სივრცეში დანერგილი უახლესი ტექნიკური მიღწევების, როგორც UI-ში, ისე ხმის პროცესინგში, ერთიანად გამოყენება.

შესავალი

სურათი1. Line 6 POD - ჰროდუქტი, რომელმაც გიტარის სამყარო სამუდამოდ შეცვალა. გამოვიდა 1998 წელს, მომხმარებელს სთავაზობდა 10-ზე მეტი გამაძლიერებლის ციფრულ სიმულაციასა და უამრავ ეფექტს, რაც იმხნად წარმოუდგენელი იყო.

ისტორიულად, ელექტრო გიტარის ძირითადი დანიშნულება ხმის გაძლიერება იყო, რათა ასობით და ათასობით მსმენელს მისი გაეგოთ, განსაკუთრებით ცოცხალი ბენდში დაკვრისას. შესაბამისად, ძველი ხმის გამაძლიერებლები საკმაოდ ხმამაღალი იყო. საქვეყნოდ ცნობილია The Beatles-ის მიერ გამოყენებული VOX AC 15 და VOX AC 30 15 და 30-ვატიანი გამაძლიერებლები. დროთა განმავლობაში გაიზარდა ხმის სიმძლავრის მოთხოვნებიც და 70-იან წლებში უკვე ნახავდით Marshall-ის 100-ვატიან გამაძლიერებლებს, რომელთაც იყენებდნენ ბენდები: AC/DC, The Who, Van Halen და ა.შ. არაა გასაკვირი, რომ ამ ეპოქის მრავალ მუსიკოსს დღეს სმენა თითქმის დაკარგული ან დაზიანებული აქვს. ტექნიკის განვითარებასთან ერთად დაიხვეწა PA (public address) სისტემები, რაც გულისხმობს სასცენო ხმის უზრუნველყოფას. შესაბამისად, მძვინვარე ხმის მქონე და ფიზიკურად არარენტაბელურ გამაძლიერებლებზე მოთხოვნამ ბოლო

წლებში მნიშვნელოვნად იკლო. 2000-იანებში პოპულარულობა მოიპოვა ხელჩანთის ზომის 15-20-ვატიანმა გამაძლიერებლებმა, 2010-იანებში კი უკვე მრავალი ცნობილი მუსიკოსი ციფრულ აპარატურაზე გადაერთო, რაც თავისთავად განაპირობა კომპიუტერული ტექნიკის წინსვლამ. ხმის ციფრული დამუშავება ამ შემთხვევაში გულისხმობს საჭირო აუდიო პროცესინგის ანალოგურიდან ციფრულში გარდაქმნას. კერძოდ, ამ ტექნიკით შესაძლებელია ისეთი ფენომენის მიმსგავსება, როდესაც ანალოგური გამაძლიერებელი სრულ დატვირთვაზეა და გიტარის ჟღერადობას თავისებურ ელფერს ჰმატებს (ე.წ. power amplifier distortion). გიტარისტთათვის მნიშვნელოვანია ამ ეფექტის ციფრული წარმოქმნის შესაძლებლობა დაბალ ხმაზეც კი. გარდა ამისა, ციფრულ ტექნიკას სხვა უპირატესობანიც გააჩნია: იგი დაფუძნებულია ტრანზისტორულ სქემებზე, რაც გაცილებით იაფფასიანი და მდგრადია, ვიდრე მრავალ (განსაკუთრებით ვინტაჟურ) გამაძლიერებელში გამოყენებული ვაკუუმის ნათურები. ამ უკანასკნელის ხმაზე გავლენის ციფრულად წარმოქმნა კი დიდი ამოცანაა.

ამგვარად, დღესდღეობით ელექტრო გიტარა არა აუცილებელი სასცენო დანადგარია, არამედ სასიამოვნო და მრავალფეროვანი ჰანგების მომგვრელი ინსტრუმენტი. ვინაიდან, ელექტრო გიტარას თავად თითქმის არანაირი აკუსტიკური ჟღერადობა გააჩნია, ბაზარზე სულ უფროდაუფრო იზრდება მოთხოვნა პორტატულ გამაძლიერებელ/პროცესორებზე (ანუ, შესაძლებელია ხმის დამუშავება მოხდეს ერთ მოწყობილობაზე, გაძლიერება/გამოშვება კი - მეორეზე).

ამ ნაშრომში მხოლოდ განხილული იქნება ხმის პროცესინგის ტექნიკური უზრუნველყოფის ნაწილი. გაძლიერება და მსმენელამდე მიტანა საინჟინრო საკითხია და მისი ინტეგრაცია საკმაოდ მარტივია. მაგალითად, კომპიუტერში ჩაშენებული დინამიკებით და ა.შ.

კონკრეტული ელექტრო ინსტრუმენტებისთვის შემუშავებული ხმის ეფექტების პროცესორები გამოთვლითი ძალის მიხედვით საკმაოდ ძვირადღირებულია დღესდღეობით გავრცელებულ x64-არქიტექტურის პროცესორებთან შედარებით. ანუ, მარტივად რომ ვთქვათ, პერსონალური კომპიუტერი მრავალჯერ სწრაფია მუსიკალური ეფექტების პროცესორებთან შედარებით. მაგალითისთვის, ბაზარზე წამყვან პროდუქტს Neural DSP Quad Cortex-ს აქვს სპეციალიზირებული 4-ბირთვიანი 2GHz სიხშირის პროცესორი. მისი ღირებულება კი შეადგენს წარმოუდგენელ $1849-ს! შედარებისთვის, ამ ტექსტის აკრეფისას ჩემი საშუალო დონის პროცესორი Ryzen 5 3600 მუშაობს 4Ghz-ზე 6

ბირთვით, მთლიანი მოწყობილობის ფასი კი რამდენჯერმე ნაკლებია Quad Cortex-ისაზე. ცხადია, არსებობს პროგრამული თუ ინჟინრული ოპტიმიზაციის საკითხები (Quad Cortex

სურათი 2. Neural DSP Quad Cortex - ამ დროისთვის ყველაზე ძლიერი გიტარის ეფექტების პროცესორი საკუთარი არქიტექტურის 4-ბირთვიანი 2GHz პროცესორით, ოპერაციული სისტემით და 10-ზე მეტი შემავალ/გამომომავალით (Input/Output)

საკუთარ Linux-ზე დაფუძნებულ ოპერაციულ სისტემას იყენებს), გააჩნია ჩაშენებული მრავალი შემავალი ჯეკი, რაც უზრუნველყოფს არაერთი ინსტრუმენტის ერთდროულად შეერთებას და ა.შ. მთავარი კითხვა კი მაინც შემდეგია: შესაძლებელია თუ არა ანალოგური აპარატურის მსგავსი ხმის მიღება კომპიუტერით თავის აუტკიებლად?

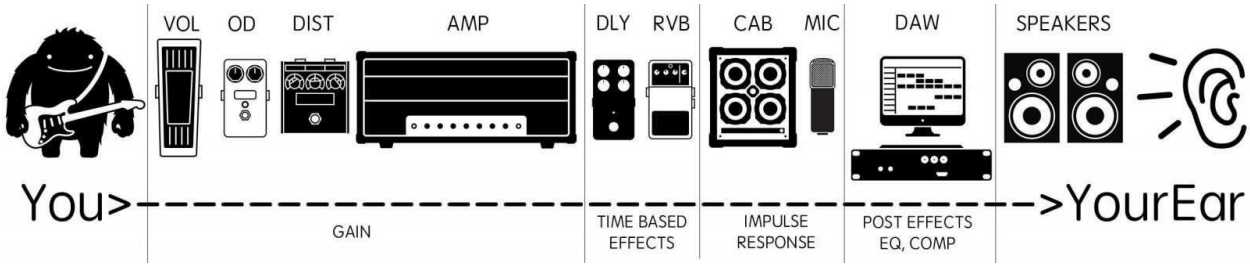
ეს უკიდურესი კრიტიკულად მნიშვნელოვანია, რადგან როცა საქმე ტექნოლოგიებს ეხება, გიტარისტები საკმაოდ კონსერვატიულად განწყობილი ხალხია. ერთმა უსიამოვნებამ მათ შეიძლება კომპიუტერებისადმი გული სამუდამოდ აუცრუვოს. ბაზარზე არსებობს მრავალი პროფესიონალური დონის აპლიკაცია, რომელიც ანალოგურთან მაქსიმალურად, ყურისათვის თითქმის განურჩევლად ახლო ხმას იძლევა. მათ ნაკლოვანებებიც აქვთ: მაღალი ფასი და არაპორტაბელურობა. კერძოდ, ისინი ადგილობრივი (Native) აპლიკაციებია. რაც სასურველ ოპერაციულ სისტემაზე დაყენებასა და გაშვებას გულისხმობს. არსებობს უფასო პროგრამებიც, თუმცა არაერთი მათგანი არაა standalone ვერსიის სახით, ანუ მათ ჩასართავად საჭიროა მასპინძელ (host) პროგრამაში

ჩანართის (plugin) სახით გაშვება. ასეთ პროგრამებს ძირითადად DAW (Digital Audio Workstation) ეწოდებათ. მათთან თავსებადობის აუცილებლობა ქმნის დამატებითი სტანდარტიზაციის (VST2, VST3 და ა.შ.) და მისი მიყოლის აუცილებლობას და ეს ყველაფერი არაა. ჩვეულებრივ, ელექტრო გიტარა დაკვრისას შეერთებულია შემდეგ მოწყობილობათა ჯაჭვში: ხმის გამაძლიერებელი, დინამიკები და სურვილისამებრ, ეფექტები. ეს ნიშნავს, რომ დამაჯერებელი ხმის მისაღებად დამკვრელს ყველა ამ კომპონენტის სიმულაციაში ჩართვა სჭირდება. არსებობს ამ საკითხის როგორც ცალკეული, ისევე ერთიანი მოგვარებები. ერთიანი აპლიკაციები თუ პლაგინები (DAW-ში ჩასართავი) მოიცავენ ყველა ამ კომპონენტს, თუმცა ბევრი მათგანი ფასიანია, იკავებენ ასობით მეგაბაიტს. ეს იმიტომ, რომ უმეტესად ისინი შეიცავენ რამდენიმე გამაძლიერებლისა თუ ეფექტის სიმულაციას. განსაზღვრულ მომენტში გიტარისტს შეიძლება მხოლოდ ერთი დამაკმაყოფილებელი ხმა სჭირდებოდეს, ამხელა პროგრამული უზრუნველყოფასთან გამკლავების შემდეგ კი, როგორც ითქვა, შესაძლოა საერთოდ გაუქრეს გიტარაზე დაკვრის სურვილი. შესაბამისად, ბრაუზერში ჩაშენებული ყველა-ერთში სახის სიმულაცია წაადგება მოსახერხებლობის პრობლემის გადაჭრას. ასევე, ამ პროექტის ღია კოდის ფორმატი (open source) მას საგანმანათლებლო ხასიათსაც შესძენს და ამ ტექნოლოგიური სფეროს დემოკრატიზაციას შეუყწობს ხელს.

დღესდღეობით ბრაუზერში ჩაშენებული გიტარის გამაძლიერებლის თუ ეფექტთა სიმულაციათა რაოდენობა რამდენიმე ერთეულს არ უნდა სცდებოდეს. უმრავლესობა ელემენტარული დონისაა, მაგალითად, ეს სიმულაცია დამაჯერებლად მარტივი ვიზუალური ინტერფეისით და მრავალფეროვანი ეფექტებით წარმოგვიდგება, თუმცა ძირეული ხმის გამაძლიერებლის სიმულაცია საკმაოდ შეზღუდულია, გათვლილია უფრო ისეთ მსუბუქი სტილის მუსიკაზე, როგორებიცაა პოპი ან ბლუზი. შედარებით მეტი ეფექტი და გამაძლიერებელი გვხვდება ამ ვებ-გვერდზე. აქ უკვე შესაძლებელია სასურველ ეფექტთა კომბინაციის და პარამეტრების დამახსოვრება. მიუხედავად 9 გამაძლიერებლის სიმულაციისა, ყველა მათგანს ერთნაირი ტონალობის კონტროლი აქვს (tonestack), რაც ხმაში მცირე განსხვავებებში გამოიხატება და მთლიანი ვებ-აპლიკაციის ჟღერადობაც ჩამორჩება ადგილობრივი (native) აპლიკაციების ხარისხს.

საკითხთან მიმართებაში ყველაზე საფუძვლიანია Michael Buffa-სა და Jerome Lebrun-ის ნაშრომი Real time tube guitar amplifier simulation using WebAudio (1). აღნიშნული სიმულაცია იყენებს ე.წ. თეთრი ყუთის (white box) მიდგომას, რაც გულისხმობს ფიზიკური სქემის დეტალურ ციფრულ მოდელირებას. კონკრეტულ სიმულაციაში

გვხვდება თეთრი ყუთის მოდელის ალტერნატიული ვარიანტი, რომელშიც სამაგალითო მოწყობილობის მისაბაძად გამოყენებულია ტალღების ციფრული ფილტრები. მოდელში კომპონენტთა ზოგიერთი ურთიერთქმედება სიმარტივისთვის გათვალისწინებული არაა. ასეთია, მაგალითად, გამაძლიერებლის დამოკიდებულება სპიკერების წინაღობაზე (speaker impendance). წრედის ლოგიკური ნაწილები (ფილტრები, ნათურები და ა.შ.) პროგრამაში წარმოდგენილია ისეთი მაღალი დონის WebAudio კვანძებით, როგორებიცაა Wave Shaper და Biquad Filter. აღნიშნული მეთოდით შესაძლებელია დამაკმაყოფილებელი ხარისხის ჟღერადობის მიღება, რაც მოითხოვს საუკეთესო პარამეტრების ხელით პოვნას და შრომატევადი პროცესია. ავტორების ასეთი გადაწყვეტილება ნაწილობრივ გამოწვეული იყო WebAudio API-ის იმჟამინდელი განუვითარებლობით. კერძოდ, აუდიო ნიმუშების (sample) ხელით დამუშავება მოუხერხებელი იყო Script Processor node-ის მიერ გამოწვეული ხმოვანი გვერდითი ეფექტებისა და დამატებითი დაგვიანების (latency) გარეშე. რაც შეეხება დაგვიანებას, აღსანიშნავია, რომ ბოლო წლებში Windows-ზე გაცილებით შემცირებული დაგვიანებისა, იგი კვლავ შესამჩნევია სწრაფი გიტარისტისთვის. macOS სისტემაზე ეს პრობლემა არ შეინიშნება. Mac პლატფორმა მუსიკალურ წრეში საკმაოდ პოპულარულია. ამიტომ, მივიჩნევ, რომ პროექტზე მუშაობა მაინც სასარგებლო იქნება, ამავდროულად, ვიმედოვნებ, რომ უახლოეს ხანებში Windows სისტემაზეც დაგვიანება დასაშვებ საზღვარს მიუახლოვდება (23მწმ-მდე).



სურათი 1. სიგნალის გზა, რომელსაც იგი გადის გიტარიდან ჩვენს ყურებამდე

ზემოთ განხილულ სიმულაციის პირადი ხმარებიდან გამოიკვეთა რამდენიმე სფერო, რომლის გაუმჯობესებაც შეიძლება. პირველი, თვალშისაცემია რიგითი გიტარისტისთვის ზედმეტად გადატვირთული ვიზუალური ინტერფეისი. საიტი იმდენად ბევრ კონტროლს შეიცავს, დამაკმაყოფილებელი ტონალობის მისაღებად წუთებია საჭირო. ამას ართულებს სპიკერის სიმულაციათა ნაკლებობა. როგორც პირველ ნახატშია ნაჩვენები, გამაძლიერებლისა და ეფექტების შემდეგ სიგნალი ხის კონსტრუქციაში (კაბინეტი) ჩაშენებულ სპიკერებში გადის, ამ უკანასკნელიდან გამოსულ ხმას კი მიკროფონი იწერს

და დასამუშავებლად კომპიუტერში აგზავნის. კაბინეტის კონსტრუქციასა და მასში ჩაშენებული სპიკერ(ებ)ს ხმაზე გადამწყვეტი მნიშვნელობა აქვს, იმდენად, რომ პროდიუსერები მათ არჩევას და სხვადასხვა მიკროფონით სხვადასხვა პოზიციიდან ხმის აღებას საათებს, დღეებს თუ წლებს ახარჯავენ. ეს საკითხი ზოგისთვის იმდენად აქტუალური იყო, რომ დაამზადებინეს სპეციალური მიკროფონის რობოტი, რომელიც მიკროფონს მცირე მანძილზე სამ განზომილებაში პულტის მეშვეობით მართავს.

სურათი 2. მიკროფონის რობოტი

ბევრი ეფექტი თუ გამაძლიერებლის კომპონენტის მახასიათებელი უკვე სიმულირებულია, იქნება ეს უშუალოდ WebAudio-ში თუ WebAssembly-თვის გადმოთარგმნილ C/C++ კოდში. WebAssembly არის ასემბლის მსგავსი ინსტრუქციების ფორმატი, რომლის წაკითხვაც თანამედროვე ბრაუზერების ძრავებს შეუძლიათ. WebAssembly-ის კოდის WebAudio სივრცეში ჩასმისთვის გამოიყენება ახლახანს დამატებული AudioWorklet, რომელმაც ნელი და მოუხერხებელი ScriptProcessorNode შეცვალა. AudioWorklet-ის მეშვეობით შესაძლებელია დაუმუშავებელ აუდიო ბაფერებზე (buffer) წვდომა და ზემოქმედება რეალურ დროში, განცალკევებულ ნაკადში (thread). ეს, ფაქტობრივად, ვებ ბრაუზერში როგორც კომერციული დონის აუდიო პროდუქტების:

სინთეზატორების, ეფექტების და ციფრული აუდიო სადუგრების შექმნას ხდის შესაძლებელს, ისე ადგილობრივი პროდუქტების გადაწერას. ამისთვის შესაძლებელია ისეთი პროგრამული უზრუნველყოფის გამოყენება, როგორიცაა FAUST, Emscripten (C/C++ კოდის Javascript-ში გადასათარგმნად), WebAudioModules და ა.შ. ვებ პლატფორმა უზრუნველყოფს მოდულარობის მაღალ დონეს, როდესაც პლაგინთა დამატება მხოლოდ რამდენიმეხაზიანი კოდითაც შესაძლებელია. AudioWorklet-ების მეშვეობით სიგნალის დეტალიზირებული მანიპულირება არამარტო დამატებითი ეფექტებისთვის, არამედ გამაძლიერებლის წრედის გარკვეულ კომპონენტთა სიმულაციისთვის უფრო მეტ სიზუსტეს გვაძლევს. ამ მიდგომი რეალიზაციისთვის ვიყენებ Faust კომპილატორს, უფრო სწორად კი, კომპილატორის კომპილატორს, რომელსაც შეუძლია Faust მუსიკალური პროგრამირების ენის კომპილატორის სასურველი პარამეტრებით ვებ ბრაუზერში გენერაცია და JIT (just in time), ანუ მოთხოვნისამებრ კომპილაცია.

სურათი 3. KPP_Distruction ადგილობრივი აპლიკაცია და მისი შესატყვისი ლოკალურ ვებსაიტის პროტოტიპი. სლაიდერები ავტომატურადაა გენერირებული. შესაძლებელია როგორც აუდიო ჩანაწერთან, ისე ცოცხალი გიტარის სიგნალთან დაკავშირება

ეს ინოვაციური მიდგომაა, რომლის წარმატებულ პრაქტიკულ გამოყენებას არსებულ ვებ-სივრცეში ვერ წავაწყდი (გარდა თავად ოფიციალური IDE-ისა, რომელიც სამუშაო სივრცე უფროა, ვიდრე მზა პროდუქტი). FAUST ონლაინ კომპილატორის გამოყენებით საშუალება მეძლევა ვებ-სივრცეში გადავიყვანო არსებული ადგილობრივი აპლიკაციები

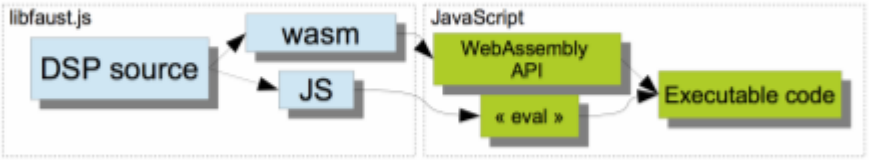
მცირედი შესწორებებით და მოვარგო სპეციალიზირებული სამომხმარებლო ინტერფეისი, რაც ჯამში უზრუნველყოფს ხმის მაღალ ხარისხსა და მომხმარებლის გამოცდილებას (UX). აღნიშნული კომპილატორი აგენერირებს თანამედროვე AudioWorklet სტანდარტში იმპლემენტირებულ WebAssembly კოდს, რაც ბრაუზერისთვის native-თან მიახლოვებული სისწრაფის გარანტია. ასევე, ვინაიდან კოდი მოთხოვნისამებრ (ძირითადად, ჩატვირთვისას) კომპილირდება, კომპილაციის დრო კი თანამედროვე კომპიუტერებზე უმნიშვნელოა, აპლიკაციის დეველოპმენტი და მოხმარება ერთ სივრცეში ექცევა. აპლიკაციის გასაშვებად და ასაკინძად (build/bundle) ვიყენებ ასევე თანამედროვე Create React App-ს, რომელიც Node JS-ზე მუშაობს. ამ სახით შესაძლებელია არამარტო გამარტივებული დეველოპმენტი მოდულარული package-ების გამოყენებით, არამედ სამომავლოდ აპლიკაციის აბსტრაგიზაცია და ტიპიზაციის (პროექტში ვიყენებ Javascript-ის ქვეტიპ Typescript-ს) საშუალებასაც იძლევა, რათა სხვა დეველოპერებმა გამოიყენონ, როგორც პაკეტი (package).

ლიტერატურის მიმოხილვა

აღნიშნულ საკითხზე ყველაზე ვრცლად ნამუშევარი აქვს Michel Buffa-ს Web Audio Conference-ისა (2) და პროექტ WASABI-ის ფარგლებში. იგი და მისი თანაავტორები ნაშრომში განიხილავენ სპეციფიური დომეინის (აუდიო) დაპროგრამების ენა FAUST-ის კოდის WebAssembly-ში გადატანასა და ამუშავებას. კერძოდ, დედააზრია დაბალი დონის WebAssembly კოდის ბრაუზერში ახლადშემოღებული AudioWorklet-ების გამოყენებით ცალკე ნაკადად გაშვება და აუდიო სიგნალის ნამდვილ დროში დამუშავება. FAUST (Functional Audio Stream) დაპროგრამების ენის უპირატესობა სხვა ისეთ მსგავს დაპროგრამების ენებთან შედარებით, როგორიცაა Max/MSP, PureData, Supercollider, არის ის, რომ მისი პროგრამები არა ინტერპრეტირებული, არამედ სრულად კომპილირებადია. შესაბამისად, საჭირო არქიტექტურული ფაილების გამოყენებით შესაძლებელია მისი, როგორც C/C++-ის მაღალი დონის ალტერნატივად ხმარება. არქიტექტურულ ფაილებში იგულისხმება ის კოდი, რომელიც პროგრამებს დააერთებდა ოპერაციული სისტემის სპეციფიურ კონტროლერებთან (OSC, MIDI, GUI) და აუდიო დრაივერებთან. არსებობს FAUST კომპილატორის ორი სახე: მთავარი (ოფიციალური master branch), რომელსაც კოდი გადაჰყავს მხოლოდ C++-ში და Faust2, რომელსაც რამდენიმე ენაში თარგმნა შეუძლია შუალედური FIR-ის წარმოდგენის მეშვეობით (FAUST Imperative Representation). FIR ზოგადად აღწერს sample-ებზე ჩატარებულ გამოთვლებს. იგი შეიცავს ცვლადებისა და მასივების წაკითხვისა და ჩაწერისთვის და არითმეტიკული ოპერაციების ჩასატარებლად საჭირო პრიმიტივებს, განმარტავს საჭირო კონტროლის სტრუქტურებს (for და while ციკლები, if სტრუქტურა და ა.შ.) კომპილატორის შიდა "სიგნალების ენა" გადაკომპილირდება FIR-ში. FIR-ის C, C++, Java, Javascript, asm.js, WebAssembly თუ LLVM IR-ში გადასათარგმნად არაერთი ბექენდი შეიქმნა. შესაბამისად, FAUST კომპილატორი შეიფუთა ჩაშენებად ბიბლიოთეკად და ეწოდა libfaust, გამოქვეყნდა რა factory/instance მოდელზე დაფუძნებულ შესაბამის API-თან ერთად. ჩემს ნაშრომშიც სწორედ ეს ტექნოლოგია გამოიყენება, კერძოდ, NodeJS-ის package, რომელშიც უკვე ჩაშენებულია დამატებითი Javascript-ის "წებოვანი" კოდი, ე.ი. დამხმარე ფუნქციები DSP კოდის აღწერის JSON სახით მისაღებად. ეს metadata მეტად გამოსადეგია სამომხმარებლო ინტერფეისში კონტროლების - სლაიდერების, ღილაკების და ა.შ. ჩასაშენებლად. განხილული ნაშრომიდან ასევე გამოყენებულია დენორმალიზირებული ათწილადების მართვის რეჟიმი - A Flush To Zero. ეს გულისხმობს ისეთი ათწილადი რიცხვების განულებას,

რომლებიც არ ემორჩილებიან დადგენილ სტანდარტებს. ეს ხდება ხოლმე საკმარისად დაბალ სიგნალებზე მუშაობისას თუ რეკურსიული ფილტრების გამოყენებისას. მათზე გამოთვლების ჩატარება განსაკუთრებით ძვირადღირებულია Intel-ის პროცესორებზე. ჩვეულებრივ, ეს რეჟიმი hardware დონეზევე ჩართულია, თუმცა არის გამონაკლისებიც WebAssembly-ში, ამიტომ, ყოველი შემთხვევისთვის, ამ რეჟიმის ჩართვით ყველა რეკურსიულ მარყუჟში (loop) იმპლემენტირდება აღნიშნული software სტრატეგია.

როგორც ცნობილია, C++ კოდის ვებ სივრცეში გადასატანად გამოიყენება Emscripten კომპილატორი. ზემოთ აღნიშნული ნაშრომიც სწორედ ამაზე მიუთითებს: თავად FAUST კომპილატორი გადათარგმნილია სუფთა JavaScript-სა და WebAssembly-ში - libfaust.js ბიბლიოთეკასა და libfaust.wasm ფაილში. Wasm-ს (WebAssembly) საწყისი DSP კოდი გადააქვს ბაიტების მასივში გადანაწილებულ მოდულსა და JavaScript-ის დამხმარე ფუნქციებში, რომლებიც წარმოდგენილნი არიან სტრიქონებად.



სურათი 4. libfaust.js + wasm-ის დინამიური კომპილაცია

აქედან კი უკვე შესაძლებელია DSP "ეგზემპლარის" შექმნა და Web Audio კვანძებთან (node) მიბმა, რომლის კონტროლი შესაძლებელია დაგენერირებული JavaScript ფუნქციების მეშვეობით.

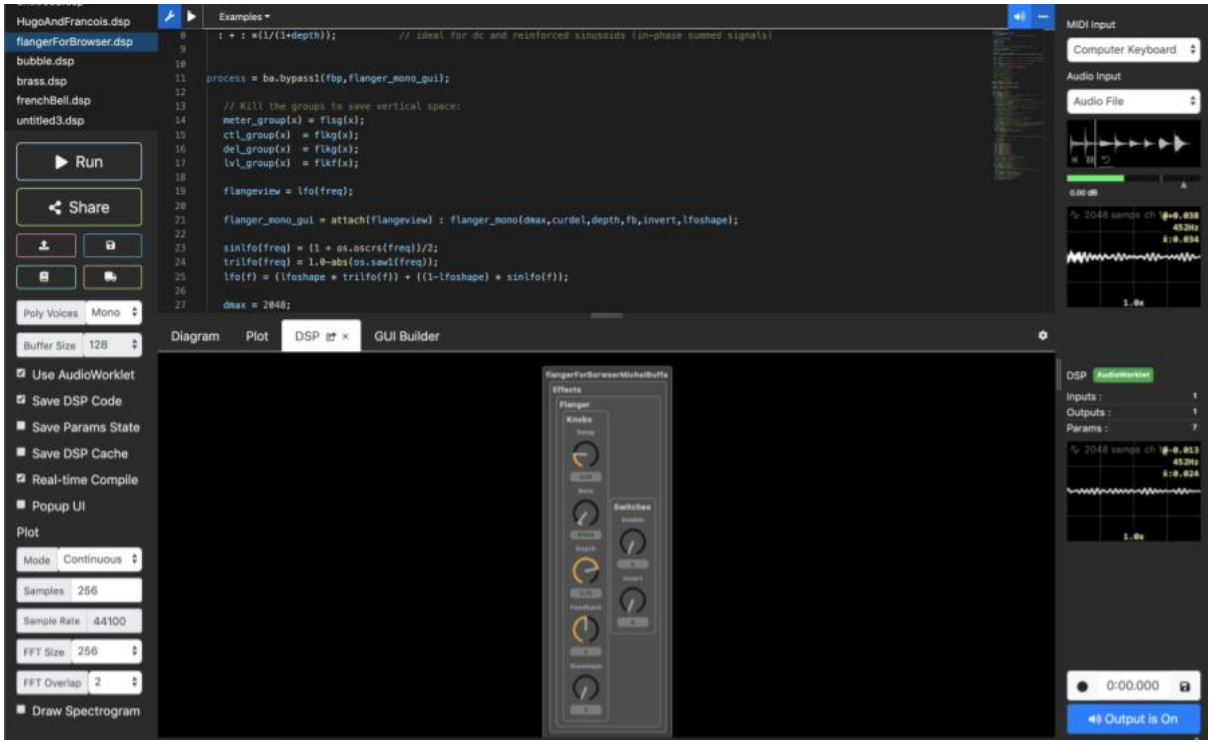
faust.createDSPFactory(dsp_code, argv, cb);

faust.createDSPInstance(fact, context, bs, cb);

შემდეგ მოყვანილია გამოყენების მაგალითები და სხვადასხვა ბრაუზერსა თუ სისტემაზე ჩატარებული წარმადობის (performance) ტესტები. დასკვნის სახით კი ნათქვამია, რომ საზოგადოება ელოდება ისეთი დეტალების გაუმჯობესებას, როგორიცაა გაუმჯობესებული დენორმალიზირებული რიცხვების მართვა და AudioWorklet-ების იმპლმენეტაციას, რაც მას შემდეგ (2017) მართლაც მოხდა: Node Package Manager-ზე განთავსებული libfaust ბიბლიოთეკა მხარს უჭერს როგორც ScriptProcessorNode-ს, ასევე, AudioWorklet კვანძებს, რაც უზრუნველყოფს ძირითადი გამოთვლების ცალკე ნაკადში გადატანას. ასევე, დღევანდელი ცნობილი ბრაუზერების აბსოლუტურ უმრავლესობას აქვს ამ ტექნოლოგიის მხარდაჭერა.

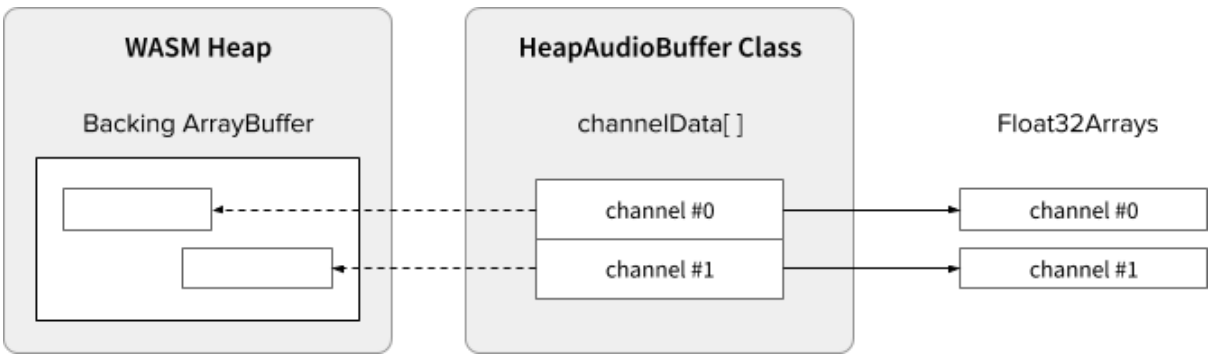
ამავე ნაშრომის ავტორმა 2020 წელს წარმოადგინა (3) FAUST დაპროგრამების ენის ვებში ჩაშენებული IDE (Integrated Development Environment), რომელიც იყენებს არამხოლოდ WebAssembly-ს, არამედ სარგებლობს W3C (World Wide Web Consortium)-დან წამოსული სტანდარტებით (PWA – Progressive Web Application, WebMIDI, WebAudio, WebComponents და ა.შ.). მისი საჭიროება ახსნილია მრავალი C/C++ (აუდიო) დეველოპერის DSP (Digital Signal Processing)-ის პროტოტიპირებისთვის ისეთ DSL-ებზე (Domain Specific Language) გადართვა, როგორიცაა FAUST. ამისთვის კი თანამედროვე ვებ პლატფორმა ხელსაყრელია, ვინაიდან აუდიო აპლიკაციების არამარტო სუფთა JavaScript-ის მაღალი დონის ხელსაწყოებით (მაგ.: WebAudio API) არის შესაძლებელი, არამედ C/C++ ან სხვა სპეციფიური DSL ენების WebAssembly-ში გადაკომპილირებითაც. FAUST Online IDE-ის საშუალებით შესაძლებელია მრავალ ფაილზე მუშაობა, IntelliSense-ის მსგავსი მახასიათებლების: სინტაქსის მითითებისა და ავტომატური შევსების გამოყენება. მოყვება რა FAUST კომპილატორის WebAssembly ვერსია, დეველოპერებს პირდაპირ ბრაუზერში შეუძლიათ კოდის წერა, კომპილაცია და გაშვება. ასევე, აღსანიშნავია მრავალი აუდიო შემავალი სიგნალი (ფიზიკური აუდიო მოწყობილობები, MIDI ინტერფეისები თუ კომპიუტერის კლავიატურა), სხვადასხვანაირი სიგნალის ვიზუალიზაცია (სპექტოგრამა, ოსცილოსკოპი და ა.შ.), DSP კოდის სატესტოდ ავტომატურად გენერირებული GUI (FAUST ენას GUI-ის აბსტრაქტულად აღსაწერად გააჩნია პრიმიტივები). მაგალითად, `vslider("label",init,min,max,step)` დაგვიხატავს სლაიდერს სახელით label, მინიმალური მნიშვნელობით min, მაქსიმალური მნიშვნელობით max და გადაწევისას ბიჯით step. ვებისთვის libfaust აგენერირებს შესაბამის JSON (JavaScript Object Notation) აღმწერ (descriptor) ფაილს, სადაც აღწერილია ყველა ეს მნიშვნელობა და მისამართები კონტროლის საშუალებისთვის.

აღნიშნულ IDE-ს ასევე გააჩნია ჩაშენებული GUI Editor. ჯამში 70-ზე მეტი პლაგინი თუ მასპინძელი (host) აპლიკაცია აეწყო. ჩემს ნაშრომშიც დეველოპმენტის პერიოდში გამოვიყენე აღნიშნული IDE, კერძოდ FAUST სინტაქსის უკეთ გასაცნობად. თუმცა GUI-ის დაპროგრამებისა და მართვის მეტი სადავეებისთვის გამოვიყენე Create React App-ით შექმნილი აპლიკაცია, რაც უზრუნველყოფს debugging-ის, დეველოპმენტის (fast refresh) და მოდულარობას (node package manager).

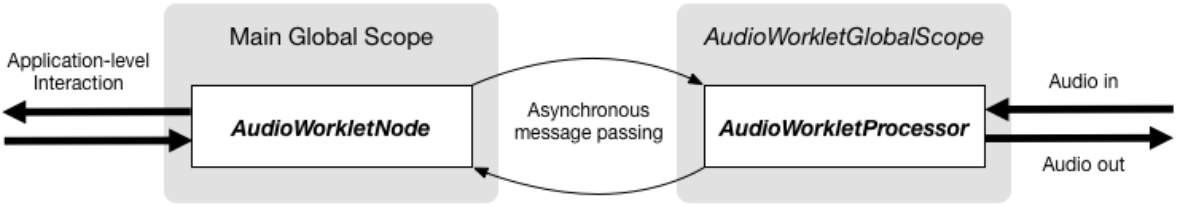


სურათი 5. FAUST Online IDE კომპილირებული კოდის სატესტად გენერირებულ default GUI-თან ერთად

“სცენის მიღმა” libfaust საკმაოდ მძიმე ოპერაციებს ასრულებს, მაგალითად, მეხსიერების გადანაწილებას WASM-ის გროვასა (heap) და აუდიო მონაცემების მასივს შორის. WebAssembly-ის კოდი მხოლოდ WASM-ის გროვისათვის გამოყოფილ მეხსიერებაზე მუშაობს. მისი JavaScript-თვის ინტეგრაციისათვის საჭიროა შუალედური ბაფერი, რომელიც მეხსიერებას მთავარ (main) და დამამუშავებელ (rendering) ნაკადებს შორის გაცვლის. Google-ის ოფიციალური [ბლოგის მიხედვით](#) (4), იყო შემოთავაზება, რომ WASM-ის გროვა პირდაპირ Audio Worklet სისტემას შეერთებოდა. ეს თავიდან აგვაცილებდა ზედმეტ კლონირებას JavaScript-ისა და WASM-ის მეხსიერებებს შორის.



სურათი 6. Google-ის ბლოგში განხილული მოდელი (HeapAudioBuffer კლასი ამჟამად ამოღებულია)



სურათი 7. ასინქრონული კავშირი AudioWorkletNode-სა და შესაბამის AudioWorkletProcessor-ს შორის. ისინი განცალკევებულ ნაკადებში ეშვებიან

შეიძლება ითქვას რომ, უკვე ბევრი რამაა გაკეთებული აუდიო პლაგინების ვებ სივრცეში გადმოსატანად და მათზე სამუშაოდ. შესაბამისად, ჩემი ნამუშევრის ძირითადი მიმართულება თანამედროვე ტექნოლოგიურ სიახლეების დანერგვასთან ერთად მოქნილი ელექტრო გიტარის ვირტუალური სტუდიის შექმნაა, რაც ყველაზე მეტად გამოადგებოდათ სახლის სტუდიებში მომუშავე პროდიუსერებს (რომელთაც შეზღუდული სივრცე გააჩნიათ აურაცხელი ფიზიკური მოწყობილობებისთვის), უბრალოდ გიტარის მოყვარულებს თუ ტექნოლოგიების ენთუზიასტებს.

სურათი 8. ლეგენდარული Marshall-ის სრული ზვინი (Full Stack) - ღირს ათასობით დოლარი და იწონის 100კგ-მდე

სურათი 9. იმავეს სიმულაცია ტრანზისტორულ სქემაზე დაფუძნებულ პედალში. რამდენჯერმე ჰატარა და იაფი, თუმცა ხმის ხარისხი საგრძნობლად ჩამოუვარდება

მეთოდები

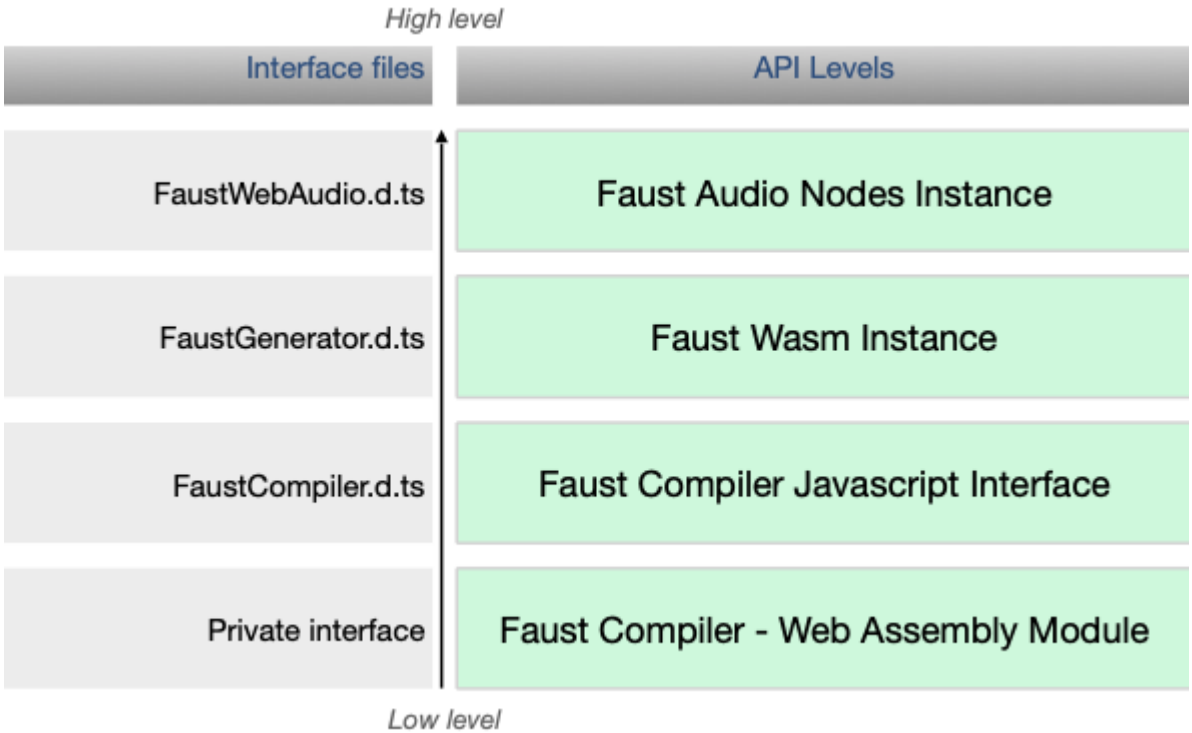
განსხვავებით ზემოთ განხილული ნაშრომების ავტორის ერთ-ერთი ნამუშევრისა, ინტერფეისის გასამარტივებლად არ გამოვაჩინე ფიზიკურ გამაძლიერებელზე არარსებული კონტროლები და ვამჯობინე მხოლოდ ისეთების დატოვება, როგორებიცაა ექუალიზაცია და ხმის კონტროლი. დანარჩენი გამაძლიერებლის სქემასთან დაკავშირებული პარამეტრები კი წინასწარაა გაწერილი ან შერჩეულია და შენახული გარკვეულ ფაილებში, რომელთაც preset-ები ეწოდებათ. გარდა გამაძლიერებლის მართვისა, როგორც უკვე აღვნიშნეთ, უმნიშვნელოვანესია სპიკერების ცვლის გამარტივებული მექანიზმი. ამისი კარგი განხორციელება წარმოდგენილია Overloud TH-U-სა და Neural DSP-ის პლაგინებში. მომხმარებელს თაგვის მოძრაობით შეუძლია მიკროფონების მოდელების, პოზიციებისა და ხმის დონის ცვლილება, როგორც ეს ნაჩვენებია სურ. 3-ზე. ტექნიკურად, კაბინეტის, მასში მოთავსებული სპიკერისა და მიკროფონის პოზიცია ერთიანი სახით ინახება აუდიო ფაილად. გავრცელებული ფორმატია .wav, რომლის წაკითხვაც ბრაუზერთა უმრავლესობას შეუძლია. ანუ, ეს წარმოადგენს გარკვეულ დიაპაზონში (გავრცელებულია 44.1Khz / 2 = 22.05Khz-იანი დიაპაზონი) სიხშირეებსა და მათ ამპლიტუდას შორის წრფივ დამოკიდებულებას. სიგნალისთვის მისი შესაბამება შესაძლებელია ოპერაციით, რასაც WebAudio-ში convolver node ასრულებს. ეს ერთგვარი მატრიცული ნამრავლივითაა. ასეთ ფაილს Impulse Response (IR) ეწოდება. მისი ზომა რამდენიმე ათეულ კილობაიტს არ აღემატება ხოლმე. ამგვარად შესაძლებელია ასეულობით პოზიციის საიტში ჩატვირთვა, რომელთა ჯამური ზომაც მხოლოდ რამდენიმე მეგაბაიტამდეა და გაშვებისას მნიშვნელოვან რესურსებს არ მოითხოვს. ეს გვაძლევს მიკროფონის რობოტის მსგავსი მექანიზმის მოწყობის საშუალებას. ცხადია, პოზიციები წინასწარ განსაზღვრული და, ამგვარად, დისკრეტიზირებულია, თუმცა საკმარისად ბევრი IR გადასვლის მსუბუქ შეგრძნებას ტოვებს. ყველა შემთხვევაში, ეს უკეთესია, ვიდრე IR-ების არაინტუიტურ ჩამოსაშლელ სიაში მოთავსება და სასურველი კომბინაციების სახელების მიხედვით ძიება.

სურათი 10. Neural DSP: Fortin Nameless კაბინეტის სექცია

გამაძლიერებლის ფიზიკურ აპარატთან მიმართებით ასეთ სახის პროექტისთვის უპირატესობა მივანიჭე ე.წ. ნაცრისფერი ყუთის მოდელირების მიდგომის სახესხვაობის გამოყენებას. ამ დროს პროგრამისტმა ცოტა რამ თუ იცის სქემაზე, ადარებს რა შესავალ-გამოსავალის აზომვებს და განმეორებადი ოპტიმიზაციით ციფრული მოდელი უახლოვდება სამაგალითო (reference) მოწყობილობას. მიუხედავად მისი ნაკლოვანებებისა (ზუსტი პარამეტრების მისაღებად დიდი ადამიანური რესურსის გაწევა, აღსრულებისას მეტი გამოთვლითი რესურსი), იგი ვებ-პლატფორმაზე დამაკმაყოფილებელ არჩევნად მიმაჩნია. Native აპლიკაციებში მისი საპირწონეა შავი ყუთის (black box) მიდგომა, როდესაც მოცემულია მხოლოდ შესავალი და გამოსავალი სიგნალები, პროგრამისტს კი უწევს მთლიანი პროცესის თავად შემუშავება. ამ მიდგომის პიონერია Kemper Profiling Amp, რომელსაც მცირე დროში შეუძლია მიწოდებული სუფთა და გამაძლიერებლიდან მიღებული სიგნალის მიხედვით ახალი სიმულაციის ავტომატურად შექმნა. მოგვიანებით იგი გამოჩნდა პერსონალური კომპიუტერების პროგრამულ უზრუნველყოფაშიც. განსაკუთრებული მოწონება დაიმსახურა Neural DSP-მ, რომელიც ნეირონულ ქსელებს იყენებს რეალისტური ჟღერადობისთვის. პროექტში მოსახერხებელია თეთრი და ნაცრისფერი მიდგომების შეხამება მათი თეორიული საფუძვლების შედარებითი სიმარტივის გამო, გამოთვლითი რესურსები კი თანამედროვე

კომპიუტერებში ერთი გამაძლიერებლის რეალურ დროში სიმულაციისთვის პრობლემას წარმოადგენს. ეს ნამუშევარი სქემის ელექტრული კომპონენტების მოდელირებისთვის ძირითადად იყენებს WebAudio-ის Wave Shaper კვანძსა და Biquad ფილტრს. ეს მიდგომა წარმატებით გამოიყენება ისეთ კომერციულ პროდუქტშიც, როგორიცაა ცნობილი Line 6 POD XT. თუმცა, ვინაიდან ბრაუზერებში უკვე დანერგილია AudioWorklet-ები, ვამჯობინე native აპლიკაციებში უკვე არსებული ალგორითმების გამოყენება, რადგან:

- AudioWorklet-ები მრავალნაკადიანობას და შესაბამისად, სისწრაფეს გვმატებს (თანამედროვე პროცესორთა აბსოლუტურ უმრავლესობას 2 ფიზიკური ბირთვი მაინც გააჩნია)
- აღარაა საჭირო ფიზიკურ მოდელს მიახლოვებული ხმის მისაბაძად მარტივი ფუნქციების გამოყენებით საუკეთესო კომბინაციების ხელით ძებნა (ჩვეული პროცესი white/grey box მოდელინგის შემთხვევაში)
- რაც ყველაზე მთავარია - სიზუსტე. AudioWorklet-ებით გაშვებულ WebAssembly-ში გადაკომპილირებულ FAUST პროგრამული ენის კოდში წვდომა გვაქვს სემპლების დონეზეც კი.
-



სურათი 11. libfaust ბიბლიოთეკის API-ის ორგანიზაცია.
მარცხნივ მოცემულია TypeScript-ის ინტერფეისის ფაილები სტატიკური ტიპიზაციის მხარდასაჭერად

გარდა დახვეწილი გამაძლიერებლის სიმულაციისა და მოქნილი სპიკერების არჩევის მექანიზმისა, საბოლოო ნამუშევრის სრულყოფისთვის დავამატე ისეთი პოპულარული ეფექტებია, როგორებიცაა ექო, overdrive, noise gate და ა.შ. ისინი გარკვეულ ჟანრთა განუყრელი ნაწილებია. მაგალითად, ბლუზ მუსიკაში ხშირად ხმარობენ overdrive პედალ-ეფექტს, რაც გულისხმობს მსუბუქად გადატვირთული (overdriven) გიტარის ხმას. ეს ფიზიკურად მიიღწევა ტალღის გლუვი წაჭრის (soft clipping) მეშვეობით. ასევე, noise gate მნიშვნელოვანია მძიმე მეტალ მუსიკის გიტარის ჟღერადობიდან დამატებითი ფონური ზუზუნის მოსაცილებლად, რასაც ელექტრო გიტარა შენობის ელექტროგაყვანილობიდან (საქართველოში ცვლადი დენი 220 ვოლტზე თავისებურია 50Hz სიხშირით) თუ სხვადასხვა ელექტრომაგნიტური გამოსხივების მქონე მოწყობილობიდან იკრებს, სიგნალის მაღალი გადატვირთვის (high gain) პირობებში კი ხმის გამაძლიერებელი ამ სიხშირეებს ერთი-ორად გამაღიზიანებელს ხდის.

საბედნიეროდ, ეფექტთა უმრავლესობა უკვე არსებობს უფასო (ღია ლიცენზიით) ადგილობრივი აპლიკაციების სახით. უპირატესობა მივანიჭე ოლეგ კაპიტონოვის პლაგინთა კრებულს, რომლის კოდი ყველასათვის წვდომადია, გამოირჩევა კოდის სიმარტივით და სხვა არაერთ უფასო პროგრამასთან შედარებით უკეთესი ხმის ხარისხით. აღნიშნული პლაგინებიდან ეფექტ-პედალთა სიმულაციების ვებ სივრცეში გადმოტანა საკმაოდ მარტივია libfaust ბიბლიოთეკის მეშვეობით (Figure11), რომელიც FAUST კოდს ბრაუზერში, ფაქტობრივად, დაუყოვნებლივ აკომპილირებს WebAssembly კოდში, შესაბამისი JavaScript დამხმარე, მფუთავი ფუნქციების გამოყენებით. პირველ რიგში, კლიენტის მხარეს ვიწერთ (ან დეველოპმენტის შემთხვევაში, დისკზე პირდაპირ ვწვდებით) DSP კოდს. აქამდე ან ამის შემდეგ კი libfaust ბიბლიოთეკით ვიღებთ და ვიმახსოვრებთ FAUST ბიბლიოთეკასა და factory ობიექტს, რომელიც შუალედურ რგოლს ასრულებს DSP კოდის კომპილაციასა და არსებულ აუდიო კონტექსტთან დაკავშირებისას. იგი შეიძლება იყოს როგორც მონოფონიური, ისე პოლიფონიური. საბოლოოდ კი მისი მეშვეობით ვაკომპილირებთ კოდს გარკვეული პარამეტრებით და ვიღებთ მზა AudioWorkletNode-ს (ან, სურვილისამებრ ძველ ScriptProcessorNode-ს), რომლის ჩაბმა უკვე პირდაპირ შესაძლებელია ხმოვან ჯაჭვში:

```
    factory.compileNode(context, 'Faust', compiler, text, '-
ftz 2', false, 128).then(node => {
```

```
streamSource.connect(node as AudioNode).connect(convolver).connect(audioContex
t.destination);
    });
```

სადაც compiler კომპილატორის ობიექტს აღნიშნავს, text – DSP კოდს სტრიქონის ფორმატში, '-ftz 2' კი Flush To Zero პარამეტრია კომპილატორისთვის, რიცხვი 128 - ბაფერის ზომა. streamSource შესაძლოა იყოს როგორც მიკროფონიდან, ისე რომელიმე შეერთებული ინსტრუმენტიდან (ჩვენს შემთხვევაში, ელექტრო გიტარიდან) წამოსული სიგნალის ნაკადი, ან თუნდაც ნებისმიერი ბრაუზერის მიერ მხარდაჭრილი ფორმატის აუდიო ჩანაწერი. ეს მოსახერხებელია დეველოპმენტის დროს, რათა წინასწარ დაკრული სუფთა სიგნალი მრავალჯერ გავამეოროთ და შევამჩნიოთ როგორ აისახება პროგრამის ხმა მასზე. Convolver კვანძი ზემოთ განხილული სპიკერის სიმულაციაა. იგი მხარდაჭერილი Web Audio API-შივე და იქმნება შემდეგნაირად:

```
    fetch('/ir/1on-preshigh.wav') // IR ფაილი
            .then(response => response.arrayBuffer()) // აუდიო ფაილის ორობი
თში გადაყვანა და ბაფერთა მასივში გადანაწილება
            .then(buffer => {
              audioContext.decodeAudioData(buffer, decoded => {
                convolver.buffer = decoded; // კონტექსტის მიხედვით დეშიფრირე
ბული IR-ის convolver-თან მიბმა
            });
```

როგორც ვხედავთ, ოპერაციათა დიდი ნაწილი ასინქრონულია, რაც განაპირობებს IR-ის (სპიკერთა მგრძნობელობის წრფივი "რუკა" - Impulse Response) შეცვლისას პროგრამის შეუფერხებლობას.

ამის შემდეგ GUI-ის მოწყობა ტრივიალური საკითხია: ადრე ნახსენები ავტომატურად დაგენერირებული DSP-ის კონტროლების აღმწერი ობიექტის მეშვეობით შეგვიძლია ღილაკებისა თუ სალიდერების დაპროგრამება:

სურათი 12. სლაიდერთათვის მითითებულია DSP-ში გაწერილი საწყისი მნიშვნელობა, დასახელება, უმცირესი და უდიდესი დასაშვები მნიშვნელობები და ა.შ.

KPP Plugin Pack-ის ყველაზე საინტერესო პროდუქტი მაინც გამაძლიერებლის სიმულაცია tubeAmp არის. იგი შეიცავს: პრე-გამაძლიერებელს (preamplifier, გამოიყენება ხმის ძირითადი დამუშავებისთვის), ტონალობის კონტროლთა ნაკრებს (tonestack, ხმის სიძლიერის, ბასების, მაღალი სიხშირეების და ა.შ. ურთიერთდამოკიდებული სქემა), მთავარი გამაძლიერებლისა (power amplifier, ახდენენ სიგნალის მეტ-ნაკლებ შეფერვას) და კაბინეტის (სადაც სპიკერებია ჩამონტაჟებული) სიმულაციას. აქედან კაბიენტის სიმულაციის საკითხი დამოუკიდებლადაცაა გადაჭრილი და არ საჭიროებს პლაგინში ჩაშენებულისვე გამოყენებას.

რაც შეეხებათ დანარჩენ კომპონენტებს, კოდის ნაწილი დაწერილია არამარტო FAUST-ში, ასევე, C++-ში. ამისი ძირითადი მიზეზია DSP-თვის მნიშვნელოვანი პარამეტრების დაშაბლონება და ცალკე ფაილებში, პროფილებში გატანა. ამას კი დამატებითი უზრუნველყოფა სჭირდება: მეხსიერების მართვა, ოპერაციულ სისტემასთან და სხვა ინფრასტრუქტურებთან, ამ შემთხვევაში, LV2 და LADSPA-თან. ასევე, C++-შია UI-იც, თუმცა მას უგულებელვყოფთ, რადგან ვებში საკუთარ იმპლემენტაციას ვახდენ.

მაშასადამე, საჭირო გახდა პროფილების სისტემის ვებ სივრცეში გადმოტანა. პროფილები წარმოდგენილია .tapf ფაილების სახით. ესაა დაუმუშავებელი (raw) ორობითი კოდი, რომელიც C++-ში იკითხება fread ფუნქციის გამოყენებით სასურველი ჩასაწერი ცვლადის სტრუქტურის ზომის ზომით. მაგალითად:

```
   // Load preamp IR data to temp buffer
            if (fread(&preamp_impheader, sizeof(st_impulse_header), 1, profile
_file) != 1)
            {
              return NULL;
            }
```

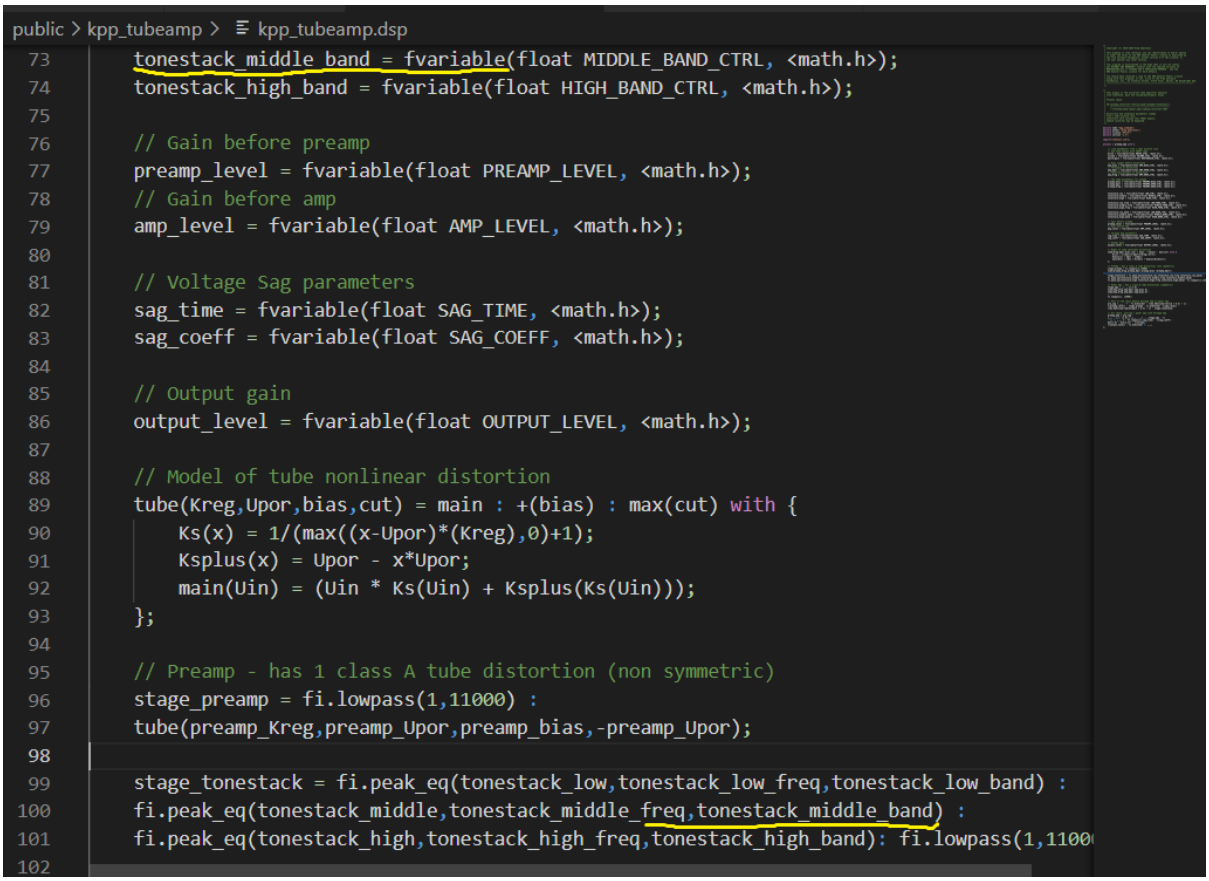
სადაც st_impulse_header არის:

```
// Header structure of
// impulse response data
// in *.tapf profile file

typedef struct {
    int sample_rate;
    int channel;
    int sample_count;
}st_impulse_header;

#endif
```

ბრაუზერის JavaScript-ში ამ ოპერაციების შემცვლელია Fetch API (ფაილების წამოღებისათვის) და ArrayBuffer (ორობითი მონაცემების განაწილებისათვის). ამის შემდეგ გამაძლიერებლის პროფილის პარამეტრები ხელით გაგზავნადია AudioWorkletProcessor-ში. ამისათვის საწყისი FAUST ენაში დაწერილ .dsp ფაილში უცხო fvariable ცვლადები უნდა ჩავანაცვლოთ თუნდაც სლაიდერებით ან რაიმე სახის კონტროლით, რადგან FAUST კომპილაციის ვებ რეჟიმში უცხო მონაცემები fvariable სახით არაა დაშვებული და “ისვრის” შეცდომას.

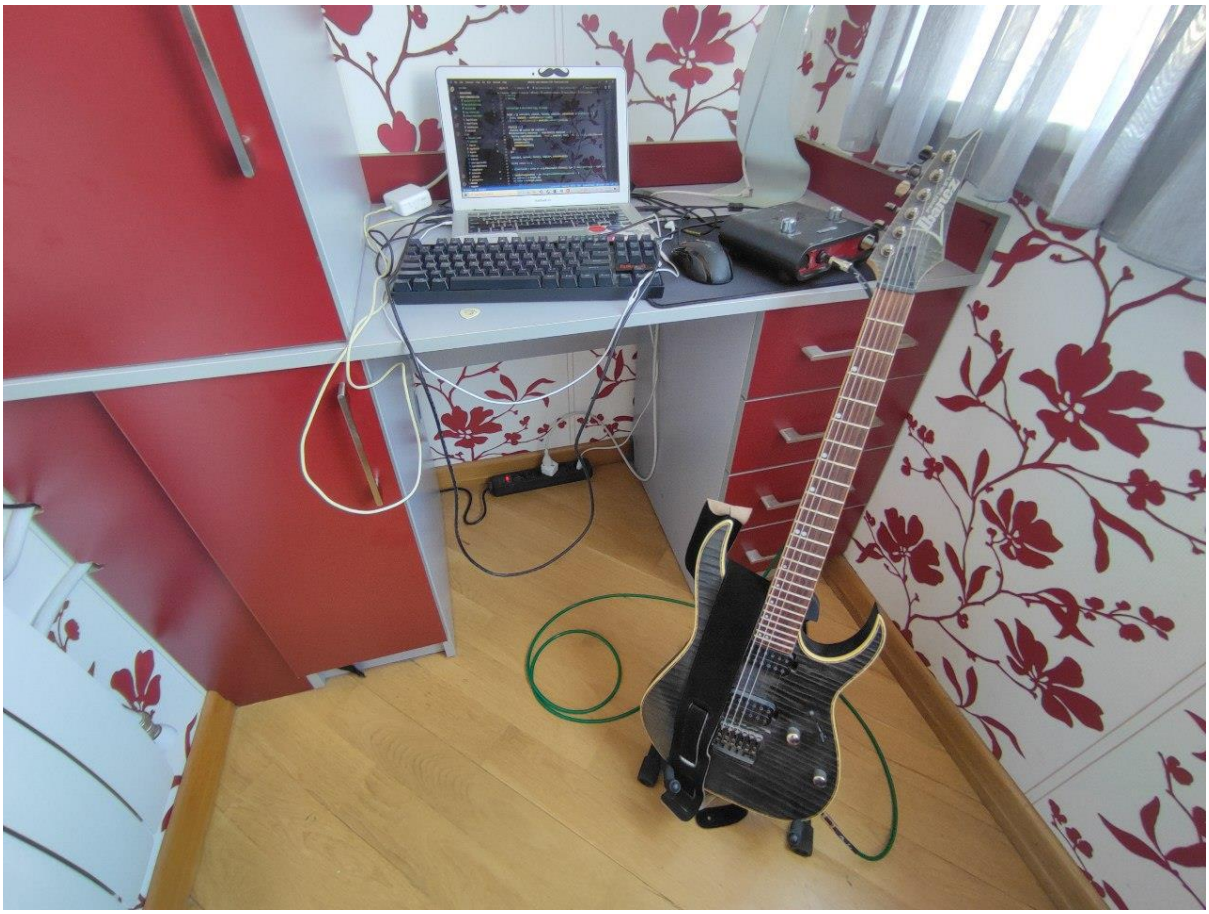


სურათი 13. კოდის ნაწილი kpp_tubeamp.dsp-დან. უშუალოდ DSP-ის კოდი 100-იოდე ხაზია. მთავარი პარამეტრები და კოეფიციენტები გარე პროფილის ფაილიდან მოდის და იწერება fvariable (foreign variable - უცხო ცვლადი)-ის სახით

კვლევის შედეგები და მათი განხილვა

ასეთნაირად ვიღებთ უახლეს ტექნოლოგიებზე მორგებულ ვებ სივრცეში წარმოდგენილ ელექტრო გიტარის ციფრულ სტუდიას. მისი ძირითადი სიძლიერეა მარტივი დიზაინი, ადგილობრივთან მაქსიმალურად მიახლოვებული სიჩქარე (სხვაობა შეუმჩნეველია რამდენიმე ეგზემპლარის შემთხვევაში) და ხმის ხარისხი.

კვლევის შედეგად დაგროვდა DSL ტიპის დაპროგრამების ენებთან მუშაობის და მათი კოდის ვებ სივრცეში გადმოპორტირების ცოდნა, განსაკუთრებით მაშინ, როცა კოდი განაწილებული მრავალი პლატფორმისათვის და ენათა შორის: C++, FAUST, JavaScript, HTML. KPP Plugin Pack თავდაპირველად შექმნილია Linux-თვის, დეველოპმენტისთვის კი ვიყენებ Windows 10 სისტემასა და Google Chrome 91 ვერსიას. საბოლოო ტესტირებისთვის ვიყენებ Line 6 TonePort UX1 აუდიო ინტერფეისს. მისი სპეციალიზირებული პროცესორი უზრუნველყოფს ელექტრო გიტარის ანალოგური სიგნალის გაციფრულებას და USB პორტით კომპიუტერისათვის მიწოდებას დაბალი დაგვიანებით და მაღალი ხარისხით (44.1-48KHz სიხშირული დიაპაზონი, 16-24 bit depth).



სურათი 14. სამუშაო გარემო მუსიკოსთა და პროგრამისტთათვის

რაც მთავარია, ხმის ხარისხი დამაკმაყოფილებელია და მოიცავს მრავალ ჟანრს, პოპიდან მძიმე მეტალის ჩათვლით. ამიტომ მომხმარებელს არ სჭირდება მრავალი პროგრამული უზრუნველყოფის თავად დაყენება საკუთარ სისტემაზე, არამედ ყველაფერი უკვე ერთ სივრცეშია.

ტექნიკურად, Create React App-ს შეუძლია პროექტის build ვერსიის შექმნა. ამ სახით სერვერზე გადატანილი სტატიკური ფაილები უზრუნველყოფს საიტის სრულად კლიენტის მხარეს მუშაობას, რაც დეველოპერისთვის პროექტის შენახვას საკმაოდ იაფს ხდის - დაკომპილირებული React JS-ის JSX სინტაქსი, სკრიპტ და მედია ფაილები (IR-ები, სურათები GUI-თვის) ჯამში მხოლოდ რამდენიმე ათეულ მეგაბაიტს იწონის.

დასკვნა

შესრულებული სამუშაო

შეგვიძლია მოკლედ შევაჯამოთ შესრულებული სამუშაო:

კვლევის საწყის ეტაპზე გამოჩნდა საკითხთან დაკავშირებული ტექნოლოგიური ტრენდები, არსებული მოგვარებები და მოსალოდნელი განახლებები ვებ-სივრცეში. კერძოდ, ბოლო წლებში განვითარდა ადგილობრივი აპლიკაციების ბრაუზერებში დანერგვის არაერთი შესაძლებლობა: Emscripten, WebAssembly. აუდიო დამუშავების კონტექსტში ყველაზე საინტერესოა ScriptProcessorNode და AudioWorklet API-ები. უპირატესობა მიენიჭა AudioWorklet-ს, რადგან იგი პირველისაგან განსხვავებით წარმოადგენს დამოუკიდებელ ნაკადს, რომელიც მთავარ ნაკადთან გზავნილებით ურთიერთობს. ანუ, ფაქტობრივად, იგი სპეციალიზირებული Web Worker-ია, რაც ვებ დეველოპერებისათვის უფრო ნაცნობი ფენომენია. AudioWorklet-ში შესაძლებელია WebAssembly-ის მოდულის ჩატვირთვა:

```
audioContext.audioWorklet.addModule()
```

თავად WebAssembly წარმოადგენს ზოგიერთი ენის კომპილატორის კომპილაციის სამიზნეს, მათ შორის, FAUST-ისას.

სწორედ FAUST კომპილატორის ონლაინ ბიბლიოთეკა შეირჩა ამ პროექტისთვის. ირონიულად, იგი თავადაა C++-დან გადაკომპილირებული WebAssembly-ში. ბრაუზერში ხმარებასა და WebAudio კვანძებთან თავსებადობას უზრუნველყოფს თანდაყოლილი JavaScript დამხმარე ფუნქციები.

შემდეგ ამოირჩა საუკეთესო პროდუქტები ვებში გადასაპორტირებლად: KPP Plugins Pack. მასში შედის თითქმის ყველა საბაზისო ეფექტი: overdrive (Bluedream), distortion (Distruction), fuzz, noise gate (Deadgate), ორი დამატებითი ეფექტი: octaver და Single2Humbucker და რაც ყველაზე მთავარია, tubeAmp – ვაკუუმის ნათურებზე დაფუძნებული გამაძლიერებლის მოდელი ე.წ. ნაცრისფერი ყუთის მიდგომით, რაც გულისხმობს ფიზიკური სქემის ნაწილობრივ მოდელირებას. DSP ალგორითმებისათვის საჭირო კოეფიციენტები დაშაბლონებულია და გატანილი .tapf ორობითი მონაცემების მატარებელ ფაილებში. ეს შესაძლებელს ხდის ერთ ღილაკზე დაჭერით შეიცვალოს მთლიანი გამაძლიერებლის ვირტუალური სქემა. არსებობს მრავალი მოყოლებული (default) და არაოფიციალური პროფილი ჟანრებისდა მიხედვით. პროფილების ჩატვირთვა უზრუნველყოფილია ბრაუზერში JavaScript-ის გამოყენებით.

დაბოლოს, საბოლოო პროდუქტს აქვს დინამიკების სიმულაციის შეცვლის შესაძლებლობა IR-ების ინტუიტური არჩევის მეშვეობით.

მოცემული გამაძლიერებლის სიმულაციის გამოყენება მიზანშეწონილია როგორც არატექნიკური მუსიკოსებისათვის, რომლებიც დამატებითი ინსტალაციის გარეშე ეძებენ სასურველ ხმას, ასევე ტექნოლოგიების ენთუზიასტთათვის, აუდიო ალგორითმების დამპროგრამელთა და ვებ დეველოპერთათვის.

მომავლის პერსპექტივა

უფრო მრავალმხრივად შეიძლება აღნიშნული პროექტის სამომავლოდ განვრცობა:

- მეტი ეფექტის დამატება და GUI-ის გაუმჯობესება. განსაკუთრებით: delay, reverb, chorus და მოდულაციის სხვა ეფექტები.
- tubeAmp პროფილების მომხმარებლის მიერ შენახვა და ჩატვირთვა, ამ პროცესის ე.წ. ღრუბლოვანად გადაქცევა. ამჟამად პროფილები იქმნება და ინახება ხელით ადგილობრივი აპლიკაცია tubeAmp-Designer-ის მეშვეობით. ამჟამად პროფილების შენახვის ოფიციალური მეთოდია საიტის მეშვეობით.
- FAUST პლაგინების გარდა სხვა ენებზე შექმნილი პროგრამების ინტეგრაცია (JUCE, C/C++ და ა.შ.).
- პროექტის მოდულარიზაცია, web-aware გახდომა, რაც გულისხმობს მის მარტივ ჩაშენებადობას, NPM სისტემაში პაკეტად გაშვება და ა.შ.

საგულისხმოა პროექტის პრაქტიკული ღირებულება ამ (ახალ) სფეროში ჩართვით დაინტერესებული დეველოპერებისათვის და მისი, როგორც ტექნიკური გიდის, გამოყენება.

შენიშვნები

არის ერთი გასათვალისწინებელი შემთხვევაც, რაც ვახსენეთ პროექტის შესავალ ნაწილში: დღემდე Windows-ის პლატფორმაზე ბრაუზერები არაა მორგებული ნამდვილ დროში აუდიო პროცესინგისათვის. ხშირად დაგვიანება აჭარბებს 20მწმს, რაც თითქმის შეუძლებელს ხდის სწრაფი მუსიკალური პარტიების დაკვრას. ეს პრობლემა ფუნდამენტურ დონეზე დგას Android სისტემაშიც. თუმცა ამის გამოსწორება დროის საკითხია. შესაბამისად, Mac კომპიუტერები ამჟამად ერთადერთი რეალური ვარიანტია მუსიკოსთათვის, Windows/Linux სისტემების მომხმარებლებს კი დეველოპმენტისთვის შეუძლიათ ჩაშენებული აუდიო ჩანაწერები გამოიყენონ.

გამოყენებული ლიტერატურა

(1) Michel Buffa, Jerome Lebrun. Real time tube guitar amplifier simulation using WebAudio. Web Audio Conference 2017 – Collaborative Audio #WAC2017, Queen Mary University of London, Aug 2017, London, United Kingdom. ffhal-01589229 https://hal.univ-cotedazur.fr/hal-01589229/document

(2) Stéphane Letz, Yann Orlarey, Dominique Fober. Compiling Faust audio DSP code to WebAssembly. Web Audio Conference, Aug 2017, London, United Kingdom. ffhal-03162898f https://hal.archives-ouvertes.fr/hal-03162898/document

(3) Michel Buffa, Jerome Lebrun, Shihong Ren, Stéphane Letz, Yann Orlarey, et al.. Emerging W3C APIs opened up commercial opportunities for computer music applications. The Web Conference 2020 - DevTrack, Apr 2020, Taipei, Taiwan. ffhal-02557901 https://hal.inria.fr/hal-02557901/document

(4) Hongchan Choi. Audio Worklet Design Pattern. https://developers.google.com/web/updates/2018/06/audio-worklet-design-pattern

დანართები

- https://lucaong.github.io/guitarstack/ virtual web-based guitar stack simulation
- https://js-rocks.web.app/stage Electric guitar effects and cabinet emulator made with Web Audio API
- https://mainline.i3s.unice.fr/AmpSimFA/ Guitar Amp Simulation using WebAudio
- https://wasabi.i3s.unice.fr/dynamicPedalboard/ Dynamic Pedalboard
- https://faust.grame.fr/ FAUST – Functional Audio Stream
- http://webaudiomodules.org/ Web Audio Modules
- https://faustide.grame.fr/ FAUST Online IDE
- https://github.com/olegkapitonov/Kapitonov-Plugins-Pack Kapitonov Plugins Pack
- https://www.npmjs.com/package/@grame/libfaust Faust Web Audio Library
- https://github.com/olegkapitonov/tubeAmp-Designer tubeAmp Designer
- https://kpp-tubeamp.com/library tubeAmp Profile Library