Středoškolská technika 2010

Setkání a prezentace prací středoškolských studentů na ČVUT

LOKALIZACE ZVUKOVÝCH ZDROJŮ V REÁLNÉM ČASE

Tomáš Popelka, Bronislav Robenek

Střední průmyslová škola elektrotechniky a informatiky, Ostrava, příspěvková organizace
Kratochvílova 7/1490, Ostrava - Moravská Ostrava

Středoškolská odborná činnost 2008/2009

Obor 18 – Informatika

LOKALIZACE ZVUKOVÝCH ZDROJŮ V REÁLNÉM ČASE

Autoři:
Bronislav Robenek, Tomáš Popelka
3. a 2. ročník

Konzultant práce:
Ing. Zdeněk Nálevka

SPŠ Elektrotechniky a Informatiky Ostrava
Kratochvílova 7
710 00 Ostrava

Ostrava, 2009

Prohlašujeme tímto, že jsme soutěžní práci vypracovali samostatně pod vedením Ing. Zdeňka Nálevky a uvedli jsme v seznamu literatury veškerou použitou literaturu a další informační zdroje včetně internetu.

V Ostravě dne

LOKALIZACE ZVUKOVÝCH ZDROJŮ V REÁLNÉM ČASE

B. ROBENEK, T. POPELKA

Střední průmyslová škola elektrotechniky a informatiky,
Ostrava, příspěvková organizace

2008/09

ABSTRAKT

Sluch, jeden z nenahraditelných přirozených smyslů zahrnuje několik vjemů, jedním z nich je schopnost lokalizovat zdroje zvuku kolem sebe. Cílem této práce bylo této schopnosti porozumět, navrhnout umělé řešení a následně jej zrealizovat.

Navrhovaný algoritmus pracuje na principu rozložení vstupních signálů na frekvenčně oddělené signály, které jsou použity k výpočtu TDOA mezi jednotlivými mikrofony pomocí křížové korelace. Takto získané body jsou následně dle souřadnic přiřazovány skutečným zdrojům zvuku.

KLÍČOVÁ SLOVA

TDOA Estimation, Multilateration, Fourier transform, Clusterization, ASIO, 3D Projection

OBSAH

Abstrakt **3**

Obsah **4**

Úvod **6**

Používané zkratky a definice **6**

Přirozená schopnost lokalizace zvukových zdrojů **6**

Obecný teoretický rozbor **7**

Stanovené zadání **8**

Hardwarová část **9**

Konstrukce mikrofonního pole 9

Varianta A *9*

Varianta B *10*

Zajištění přenosu signálu z mikrofonu do PC 12

Teoretická část **13**

Analýza problému 13

Vývoj lokalizačního algoritmu A 14

Separační algoritmus *14*

Lokalizační algoritmus *14*

Klasterizační algoritmus *15*

Vývoj lokalizačního algoritmu B 15

Softwarová část **16**

Vývoj nahrávací komponenty 17

SamplesBuffer 18

Varianta A *18*

Varianta B *19*

Implementace Lokalizačního algoritmu A 19

Implementace Lokalizačního algoritmu B 20

Tracker 21

Vývoj prezentační komponenty 22

Varianta A *22*

Varianta B *23*

Varianta C *23*

Finální Varianta B *24*

Závěr **25**

Další možnosti vývoje *25*

Reference a seznam použité literatura **26**

Konzultace 27
Přílohy 28

Úvod

Cílem této práce bylo studium a implementace přirozené schopnosti pasivně vyhledávat zdroje zvuku v okolí. Během vývoje tohoto projektu se vyskytly takové nároky na hardwarové vybavení, které nám znemožnily náš původní záměr zcela naplnit. Proto jsme se rozhodli přijmout jednodušší řešení stejného problému s nižšími nároky na hardware, které však neřešilo ozvěny. To vedlo ke zkreslení výsledků.

Používané zkratky a definice

ANN – Artifacial Neural Network – Umělá neuronová síť
ASIO - Analog Serial Input/Output – Nízko latenční ovladač zvukové karty firmy Steinberg
API - Application programming interface
FFTW – Fastest Fourier Transform in the West – Algoritmus pro výpočet FT
FT – Fourier Transform
GLU - OpenGL Utility Library
ILD – Interaural Level Difference
ITD – Interaural Time Difference
NN – Neural Nework – Přirozená neuronová síť
OS – Operační systém
SB – SamplesBuffer – Komponenta pro dočasné ukládání vzorků
SPŠEI - Střední průmyslová škola elektrotechniky a informatiky, Ostrava, příspěvková organizace
STFT – Short Time Fourier Transform
TDOA – Time Difference of Arrival
WMM – Windows Multi Media Extensions – API pro zvukové karty, uvedeno v roce 1991

Přirozená schopnost lokalizace zvukových zdrojů

Poté co je zvuková vlna projde vnějším uchem, kdy dojde k drobným cíleným úpravám signálu (změna spektra zvuku přicházejícího zezadu – *ušní boltec* a zesílení zvuku – *vnější zvukovod*), zvukové - tlakové vlny narážejí na ušní bubínek. Tím se rozvibrují součásti středního ucha (přenosové kůstky) a následně i vnitřního, kde vlnění dosáhne ústrojí *cochlea* [*Figure 1*]. Vnitřní strana ústrojí je pokrytá různě dlouhými vlasy (*stereocilia, [Figure 2]*), které jsou napojeny na nervové zakončení jednotlivých nervů. Množství energie přenesené na konkrétní nervové zakončení závisí na délce tyčinky. *Cochlea* funguje jako přirozená *Short Time Fourier Transform* [1][2], která transformuje signál z časové domény na v čase se měnící frekvenční spektrum. Každý vlas snímá konkrétní frekvenci (dle své délky - oscilátor), a čím větší energie je ve měřeném signálu právě na této frekvenci tím více tato tyčinka osciluje, čímž se budí nerv.

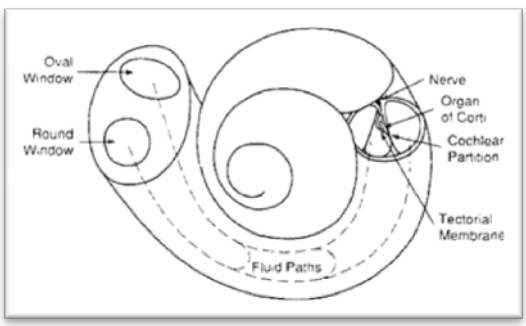


Figure 1: Cochlea

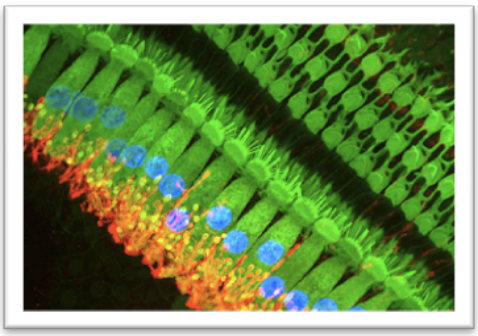


Figure 2: Stereocilia

Axony těchto nervů jsou připojeny na vstup NN, která pokud je dostatečně naučená tak paralelně zpracovává rozdíl amplitud (ILD) a časový posun (ITD) mezi levým a pravým uchem. [3] Na základě těchto informací určí NN vjem *azimuth* zdroje zvuku. Vjem *Elevation* je určován jen pomocí změny pozice uší – naklánění hlavy. Vjem vzdálenost od zdroje zvuku je získán ze zkušeností, kdy na základě dalších informací – vizuální kontakt / rozpoznání frekvenčního spektra – NN „odhadne“ jaká byla původní hlasitost a porovná ji s přijatým signálem. Díky danému útlumu zvuku na jednotku vzdálenosti dráhy, může NN doručit vjem vzdálenosti.

Pokud by na vstupu tohoto systému byly dva signály z mikrofonů s ideální kulovou charakteristikou, nebyla by takto naučená NN schopna rozlišit zda zvuk přichází zepředu nebo zezadu. Tento problém řeší nejen lidské tělo unikátním designem *ušního boltce,* který zvuk přicházející zezadu ztlumí a změní frekvenční spektrum. Naučená NN je na tyto rozdíly citlivá a tudíž na výstupu dostáváme přesnější informace.

Diky rozložení vstupního signálu na mnoho jednotlivých, frekvenčně oddělených signálů (v ústrojí *cochlea*) na výstupu NN získáváme pozici více zdrojů zvuků zároveň, protože NN jednotlivé signály zpracovává samostatně.

Audiolokalizační systém lidského mozku není konstruován k jednoznačnému datovému výstupu (tzn. množina známých bodů), ale spíše je určen pro orientaci v prostoru a hledání zdrojů zvuku s pomocí ostatních smyslů a zkušeností.

OBECNÝ TEORETICKÝ ROZBOR

Pokud bychom se rozhodli navrhnout tento umělý audiolokalizační systém přesně podle přirozené předlohy, museli bychom pro určení *azimuth* a *elevation* jen pomocí dvou mikrofonů naše mikrofonní pole naklánět a při určování vzdálenosti bychom museli použít obrazový, nebo jiný senzor pro rozpoznání objektů. Tyto informace by sloužily ke klasifikaci zdroje zvuku a následné uložení do databáze známých zdrojů. Také bychom museli odhadovat počáteční hlasitosti. Když už by jsme se rozhodli zpracovávat obraz, bylo by pro určení vzdálenosti jednodušší použit dvě kamery ve známých souřadnicích a algoritmus pro detekci hran, což je rozhodně podmětem další práce.

Kvůli těchto komplikací jsme se rozhodli použít 4 mikrofony, namísto přirozených dvou. Tyto mikrofony vhodně rozmístíme v prostoru, tak aby nebyly v rovině, tzn aby bylo možné vždy určit zda zvuk přichází zepředu či zezadu.

Pro zjištění přesné polohy neznámého vysílače signálu v 3D prostoru, potřebujeme znát minimálně 4 časy, které urazí signál od známých „přijímačů“ signálu. Tudíž vyjdeme ze soustavy rovnic [*Equation 1*] pro jednotlivé vzdálenosti založené na vzorci pro euklidovskou vzdálenost [4].

$$T_A = \frac{1}{v}\left(\sqrt{(x-x_A)^2+(y-y_A)^2+(z-z_A)^2}\right)$$
$$T_B = \frac{1}{v}\left(\sqrt{(x-x_B)^2+(y-y_B)^2+(z-z_B)^2}\right)$$
$$T_C = \frac{1}{v}\left(\sqrt{(x-x_C)^2+(y-y_C)^2+(z-z_C)^2}\right)$$
$$T_D = \frac{1}{v}\left(\sqrt{(x-x_D)^2+(y-y_D)^2+(z-z_D)^2}\right)$$

EQUATION 1 - MULTILATERATION

Kde T_A, T_B, T_C a T_D jsou časy přijetí signálu jednotlivými přijímači A[x_A, y_A, z_A], B[x_B, y_B, z_B], C[x_C, y_C, z_C] a D[x_D, y_D, z_D]. Proměnné x,y,z jsou souřadnicemi neznámého vysílače. v je rychlost šíření signálu.

Z tohoto rozboru vyplývá že naším cílem bude určení času, kdy byl signál zachycen jednotlivými přijímači budeme měřit tzv. TDOA.

Nejjednodušší metodou pro určení TDOA mezi dvěma přijímači je metoda statistická - výpočet bodu, ve kterém mají jejich signály největší křížovou korelaci [4]. Vzdálenost tohoto bodu od 0 pokládáme za TDOA mezi těmito přijímači.

STANOVENÉ ZADÁNÍ

1. Hardwarová část
    a. Konstrukce mikrofonního pole
    b. Konstrukce předzesilovače
    c. Zajištění přenosu signálu z mikrofonu do PC
2. Teoretická část
    a. Analýza problému
    b. Vývoj lokalizačního algoritmu A
    c. Vývoj lokalizačního algoritmu B
3. Softwarová část
    a. Vývoj nahrávací komponenty
    b. SamplesBuffer
    c. Implementace lokalizačního algoritmu A
    d. Implementace lokalizačního algoritmu B
    e. Tracker
    f. Vývoj prezentační komponenty

HARDWAROVÁ ČÁST

Tato část naší práce byla nezbytnou součástí realizace. Bylo zapotřebí zvážit pořizovací cenu a nároky na jednotlivé komponenty.

Rádi bychom zde poděkovali *SPŠEI* za zapůjčení zvukové karty a notebooku pro vývoj a prezentaci naší práce. Dále bychom rádi poděkovali firmám *Texas Instruments* a *Maxim* za zaslaní vzorků svých čipů, čímž nám usnadnily prototypování.

KONSTRUKCE MIKROFONNÍHO POLE

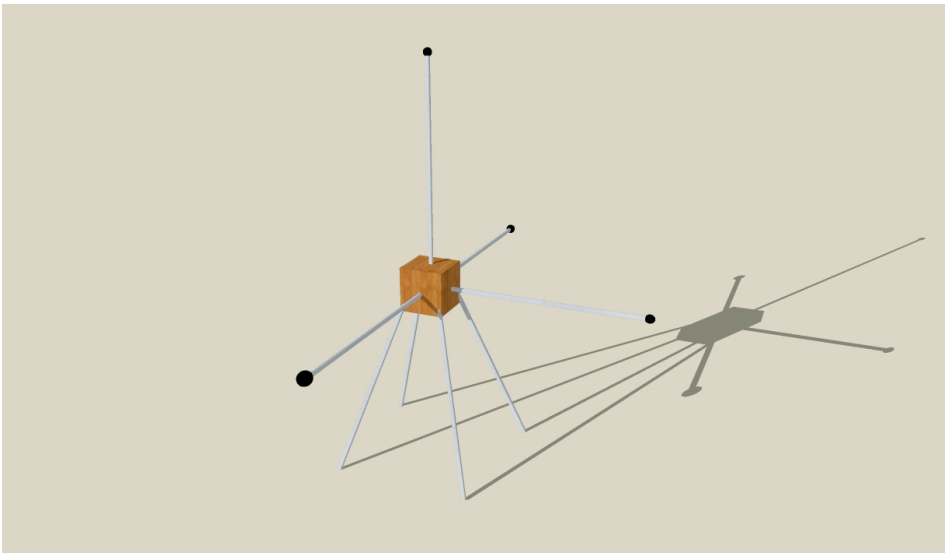
Nároky na mikrofonní pole byly tři. Umístění mikrofonů tak, aby nebyly v jedné rovině. Dále vzdálenost mezi mikrofony minimálně 1m kvůli přesnosti a také nároky na skladnost, případně přenositelnost.

Dalším důležitým rozhodnutím byl výběr mikrofonu. Na základě zkušeností jsme vybrali elektretový mikrofon s vnitřním FET zesilovačem *MCE 2500* (40-20000Hz, 2V/0,5mA, 2,2kOhm., prům.6x2,7mm). Napájení je realizováno po stejném vodiči jako signál, tudíž stačí vést k mikrofonu pouze 1 stíněný vodič.

VARIANTA A

Původním nápadem byla konstrukce s dřevěnou krychli s hranou cca 20cm a se čtyřmi metrovými kovovými tyčemi na držení mikrofonů + 4 tyče jako nohy konstrukce.

Do horní, levé, pravé a přední strany krychle by byly navrtány otvory, do kterých by se nosné tyče zasunuly. Dále by se navrtaly otvory do spodních rohů, do kterých by se zasunuly nohy naší konstrukce.



PRVOTNÍ NÁVRH KONSTRUKCE

VARIANTA B

Pro tuto variantu je použito 5 hliníkových dutých tyčí o délce 1m s průměrem 10mm, 1 ocelová tyč o délce 100mm s průměrem 12mm, upínací svorka a spojky na hadičky.

Na svorku je přivařena širší tyč – „bota“, do které se zasouvá držák konstrukce – „noha“. Spojky na hadičky jsou použity pro spojovací kříž, do kterého lze zasunout tyče pěti různými směry. Jedna hliníkovou tyč je zkrácena na 200mm a použita jako „noha konstrukce“. Zbývající tyče jsou po stranách navrtány a vnitřkem tyčí je vedena kabeláž. Pokud by byly problémy s EMI rušením daly by se tyto tyče použít jako přídavné stínění.

Do spojovacího kříže se shora, zleva, zprava a ze předu zasunou metrové tyče s mikrofony. Ze spodu je připojena „noha“, která se vloží „do boty“. Celá konstrukce je uchytitelná pomocí svorky na jakoukoli vodorovnou plochu, například k desce stolu.

FINÁLNÍ VARIANTA B

KONSTRUKCE PŘEDZESILOVAČE

Mezi hlavní nároky na funkci předzesilovače patří nízká šumovost, nízké zkreslení a přiměřené zesílení. Na vstup předzesilovače přivádíme výstup z mikrofonu, takže samotná konstrukce předzesilovače z části závisí na použitých mikrofonech.

Námi použité elektretové mikrofony vytvářejí výstupní signál na úrovni $U_{pp} = 40mV$. Výstup předzesilovače bude připojen na vstup zvukové karty [5]. Naše zvuková karta používá rozsah signálu $U_{pp} = 2V$. Z toho plyne potřebné napěťové zesílení 50x.

Při konstrukci jsme se rozhodli použít integrovaný přístrojový zesilovač firmy *Texas Instruments INA217* [6]. Jedná se o integrovaný obvod s nastavitelným zesílením a symetrickým napájením. Maximální proudový odběr je ±10 mA.

Celé řešení obsahující 4 předzesilovače a napájení mikrofonů jsme umístili na jeden plošný spoj o rozměrech 8x8cm. Pro jednodušší napájení jsme si připravili DC-DC měnič z 5V na ±12V. Tento zdroj je založen na integrovaném obvodu *Maxim MAX743*.

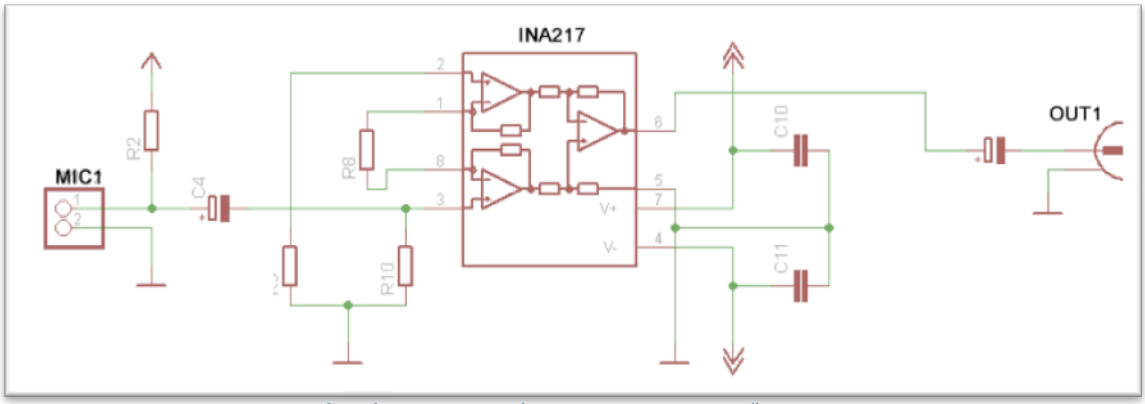
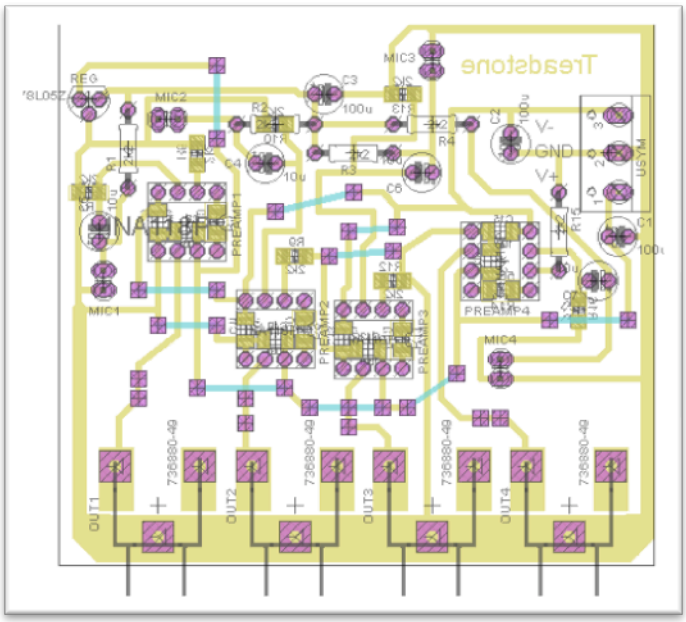


SCHÉMA ZAPOJENÍ JEDNOHO ZESILOVAČE



DPS ČTYŘ ZESILOVAČŮ

Zajištění přenosu signálu z mikrofonu do PC

Výstupní signál je z předzesilovače paralelně přiveden 4 žilovým audio-kabelem na vstupy externí zvukové karty *ESI MAYA 44 USB* [5], která je připojena k PC pomocí rozhraní USB.

Tato zvuková karta podporuje protokol ASIO, který nám umožní pohodlný přístup k nahraným vzorkům. Mimo jiné podporuje vzorkování na frekvenci 48kHz a má 18 bitový A/D převodník. Po všech stránkách se řadí ke zvukovým kartám určených pro amatérský mastering a našim potřebám plně postačí.

TEORETICKÁ ČÁST

ANALÝZA PROBLÉMU

Z přirozeného modelu a z obecného teoretického rozboru jasně vyplývá, že největším problémem bude rozpoznání času příjmu signálu – TDOA – u jednotlivých mikrofonů. Na vstupu procesu dostáváme pravidelné množství vzorků tak jak nám jej připraví ASIO ovladač. Používáme nahrávací buffer o velikosti INPUT_LENGHT = 1072 vzorků, což při vzorkovací frekvenci f_v = 48kHz odpovídá 22ms. Toto je mimo jiné perioda s jakou budou do naši aplikace přicházet nová data. A doba během které musí program stihnout tyto data uložit.

Další důležitou veličinou je obnovovací frekvence, neboli kolikrát za vteřinu dojde k přepočítání zdrojů zvuku. Pokud nebude lokalizační algoritmus pracovat s pouze INPUT_LENGHT vzorků, bude muset být zaveden vyrovnávací buffer, ten jsme nazvali *SamplesBuffer*, jeho hlavním úkolem je shromažďovat nová data do paměti. A následně je po potřebných blocích uvolňovat lokalizačnímu algoritmu. Pokud algoritmus nestihne zpracovat daný blok dat než dojde k dalšímu volání, dojde ke nespojitosti dat. Při návrhu takového bufferu je třeba brát v úvahu situaci, kdy bude lokalizační algoritmus potřebovat překrývající se bloky.

Cílem lokalizačních algoritmu je získat souřadnice zdrojů zvuku v nahrávaných datech. V principu oddělení jednotlivých zdrojů zvuku se algoritmy A a B liší. Společný mají pouze algoritmus pro získání souřadnic z vypočtených TDOA.

Pro výpočet souřadnic využijeme řešení soustav rovnic [Equation 1]. Řešení získáme pomocí *Bucher algorithm* [8], kterým získáme dva kořeny výše uvedené soustavy. Z těchto kořenů je vždy jen jeden skutečný. O rozpoznání správného řešení se stará dílčí algoritmus, který porovná výsledky s reálnými veličinami a předchozími výsledky a následně chybné řešení vyřadí.

Po včasném zpracování, jsou výsledná data předána další komponentě, která se stará o identifikaci jednotlivých zdrojů vůči jejich předchozímu stavu - *Tracker*. Výhodou identifikace je možnost sledování jednoho konkrétního zdroje zvuku a případná zpětná vazba od uživatele.

Vývoj lokalizačního algoritmu A

Tento algoritmus je silně inspirován přirozenou schopností audiolokace. Základní myšlenkou je rozdělení vstupních signálů na množství frekvenčně oddělených signálů a následného výpočtu TDOA mezi těmito dílčími signály, určení souřadnic těchto jednoduchých dílčích zdrojů. Z těchto dílčích zdrojů vypočteme reálné zdroje pomocí klasterizačního algoritmu.

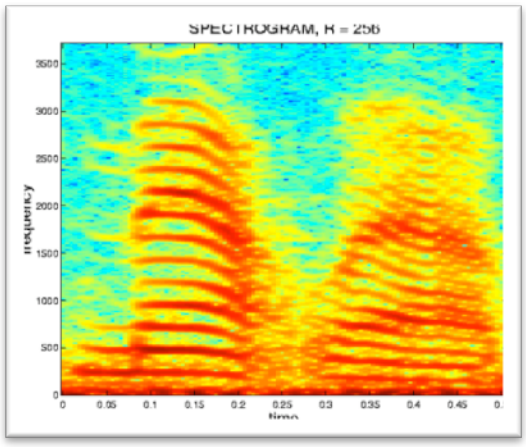
Separační algoritmus

Separování signálů nejjednodušeji dosáhneme STFT [3] . Toto řešení je jakousi umělou *cohlea (viz přirozená schopnost lokalizace)*. Získáme v čase se měnící frekvenční spektrum (v našem případě power spectrum), ve kterém si můžeme vybrat konkrétní frekvenci a její měnící se amplitudu vnímat jako samostatný signál.

Důležitým parametrem je zde PADDING, což je délka ve vzorcích o kolik se bude výstupní blok ze SB posouvat směrem doprava při každém dalším cyklu výpočtu FT. Také nám udává kolik FT je nutno stihnout spočítat než přijdou nová data. Tato konstanta také ovlivňuje přesnost výsledků protože místo 1 vzorku budeme získávat TDOA jako celočíselný násobek doby PADDING vzorků.

Druhou důležitou veličinou je délka vstupních dat pro FT, která ovlivňuje maximální frekvenci výstupní frekvenční domény dle Nyquist-Shannon teorému. To znamená že z 8192 vzorků získáme spektrum s maximální frekvencí 4096.

Na výstupu FT získáme pole komplexních čísel představujících frekvenční doménu. Power spectrum získáme pomocí výpočtu absolutní hodnoty tohoto čísla. Takto dostaneme měnící se power spectrum v aktuálním čase. Pro další zpracování potřebujeme tyto hodnoty dočasně ukládat, tím získáme STFT v požadované délce.



Ukázka STFT

Lokalizační algoritmus

Pro určení TDOA použijeme *křížovou korelaci* [4], jak již bylo řečeno v teoretickém rozboru hledáme posun při kterém mají dva vstupní signály největší *křížovou korelaci* [4]. Protože se jedná o velice oblíbený algoritmus dovolili jsme si jej převzít od Paul Bourke [7]. Výsledkem je časový posun mezi dvěma funkcemi. Vzhledem k PADDINGU je vypočtený posun celočíselným násobkem PADDING času.

Následně spočítáme souřadnice dílčího zdroje zvuku pomocí *Bucher algortihm* [8]. Tímto postupem získáme maximálně 4096 zdrojů zvuků (pokud bude vstupní šířka dat 8192 vzorků). Maximálně protože můžeme vytvořit interval povolených výkonů abychom odstranili zdánlivé zdroje zvuku. Těmito zdroji můžeme

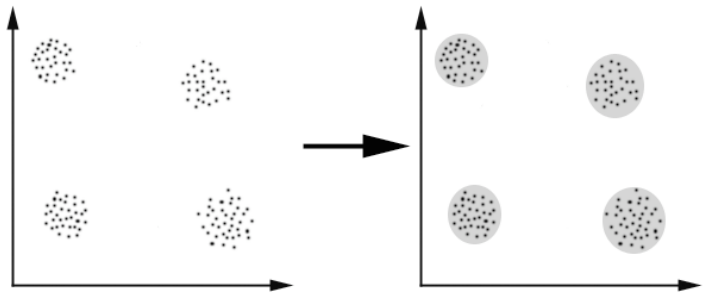
rozumět okolní hluk (nízký výkon), nebo složený zvuk z více reálných zdrojů na stejné frekvenci (vysoký výkon), jehož pozice se nám jeví mezi skutečnými zdroji.

KLASTERIZAČNÍ ALGORITMUS

Nyní když máme dílčí zdroje zvuku je našim cílem spočítat skutečné zdroje zvuku. K tomu bude sloužit klasterizační algoritmus, který vyhledá místa s největší hustotou dílčích zdrojů a vypočte jejich středy.

Existuje několik typů klasterizačních algoritmů [9], ale jejich nevýhodou je potřeba znát předem počet skutečných zdrojů ke kterým mají dílčí zdroje přiřazovat například algoritmus k-means [10]. Z tohoto důvodu jsme se rozhodli implementovat vlastní rekurzivní klasterizační algoritmus, založený na dělení prostoru.

Tento algoritmus rozdělí prostor na 4 kvádry a zjišťuje kolik dílčích zdrojů je v kterém kvádru. Pokud je tento počet pro daný kvádr menší než stanovená konstanta tyto dílčí zdroje vyřadí z dalšího zpracování. Pokud najde takový kvádr který obsahuje dostatečné množství bodů, rekurzivně jej rozdělí na čtyři menší kvádry a opakuje celý postup dokud se nedostane ke kvádru o velikosti srovnatelné s velikostí zdroje zvuku (konstanta). Střed takovéhoto kvádru označí za souřadnici skutečného zdroje zvuku. Abychom nezískávali spoustu zdrojů zvuků v jedné lokaci, rozhodli jsme se implementovat minimální vzdálenost zdrojů zvuku, nastavenou jako konstantu.



UKÁZKA KLASTERIZAČNÍHO ALGORTIMU

Poté co takto získáme množinu skutečných zdrojů zvuku, jsou tyto souřadnice předány *Trackeru,* který zajistí jejich identifikaci a následné předání Renderovací komponentě.

Takto postavený lokalizační systém by měl být značně stabilní a odolný, právě díky lokalizaci jednotlivých dílčích signálů a nikoliv celkového signálu. Pravděpodobně by byl tento algoritmus odolný i proti ozvěnám, bohužel neměli jsme možnost to vyzkoušet.

VÝVOJ LOKALIZAČNÍHO ALGORITMU B

Kvůli nedostatku výkonu si nemůžeme dovolit převádět signál na měnící se spektrum, proto je potřeba přijít s nějakým řešením, které by bylo schopno dosáhnout stejných / podobných výsledků.

Jedním z řešení by by byl výpočet *křížových korelací* [4] vstupních signálů a následné hledání vrcholů které budeme považovat za TDOA. Tyto vrcholy získáme pomocí dílčího algoritmu pro výpočet stacionárních bodů, nás budou zajímat především maxima.

Poté co získáme stacionární body, mělo by platit, že počet těchto bodů je roven počtu zdrojů. Je potřeba tyto body vhodně sobě navzájem přiřadit. Tak abychom získali trojice TDOA, které k sobě patří, toho dosáhneme vyhledáním takového maxima, které bude co nejpodobnější velikost.

Poté co takto získáme jednotlivé trojice TDOA potřebujeme spočítat souřadnice takovéhoto zdroje zvuku, toho dosáhneme opět pomocí *Bucher Algorithm* [8]. Výsledky lokalizace předáme *Trackeru* tak aby je připravil a následně odeslal *Rendereru*.

SOFTWAROVÁ ČÁST

Během vývoje softwaru došlo několikrát k změnám na úrovni celého projektu. Prvotně společně s Algoritmem A a WMM [11] začal vývoj na platformě *.NET* [12] v jazyce C# a DircetX. Po několika týdenní práci jsme došli k závěru, že WMM je nevhodné a začali jsme implementovat *ASIO* [13] pomocí hotového *managed* bindigu *BlueWave Interop ASIO* [14]. Když jsme začali se STFT zjistili jsme, že pro využití knihovny *FFTW* [15] je potřeba neustále kopírovat data mezi *managed* a *unmanaged* pamětí, což značně zpomalovalo celý proces daleko za hranice proveditelnosti.

Z tohoto důvodu jsme se rozhodli rozdělit program na dvě části – prezentační část, napsaná v *managed* kódu, a lokalizační jádro napsané v *Non-CLR C++,* pro threading jsme využili *Win32 API* [16]. Výsledkem je jeden spustitelný soubor a více dynamickými knihovnami [*Figure 6*].

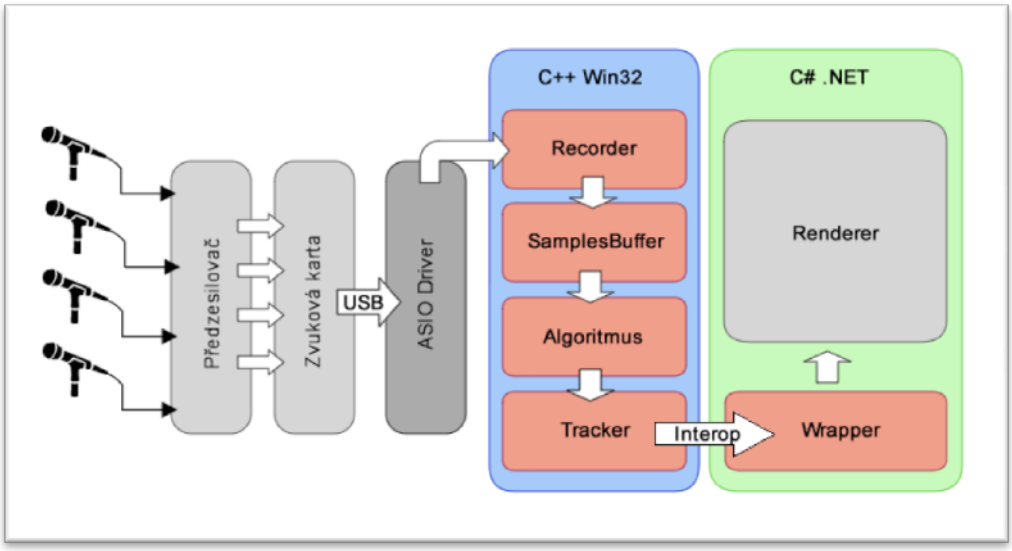


FIGURE 3 FINÁLNÍ VERZE SW NÁVRHU

Toto řešení se později ukázalo jako ideální, protože jsme mohli vypustit *BlueWave wrapper* pro ASIO a napsat si vlastní nízkoúrovňový *Recorder*, což znamenalo další úsporu výkonu. Jedním z důvodů také byla potřeba *unmanaged* přístupu k paměti, ta se sice z počátku zdála poměrně složitá, protože jsme v té době byli na poli klasického C++ nezkušení. Během implementaci *SamplesBufferu* se tato koncepce ukázala absolutně rozhodující.

V momentě když jsme začali testovat *Algoritmus A*, objevili jsme nepředpokládaný problém - nedostatečný výkon. Prvotní – chybné – časové kalkulace tuto situaci vylučovaly. Závěrem této situace jsme shledali fakt, že pro implementaci *Algoritmu A* je potřeba spočítat minimálně 15x více FT než stihl používaný PC . Testovací sestava obsahovala CPU *Intel Core 2 Duo 2Ghz.* Řešením by mohlo být využití paralelizace výpočtů, což by nám ale při dnešních 4 CPU jádrech příliš nepomohlo, příhodnější by bylo využít nyní začínající technologii - využití GPU, tzv. GPGPU [17]. Využitím grafické karty se stovkami unifikovanými jednotkami by zaručovalo dostatečný paralelní výkon. Bohužel pro testování jsme neměli příhodný hardware ani software.

Tato situace na nás měla zdrcující účinky. Jedno z řešení bylo ukončení projektu. Nicméně jsme se rozhodli pokračovat a zrealizovat řádově jednodušší *Algoritmus B* s nižšími nároky. Ten se během vývoje také několikrát změnil, ale popisovat každý z nich je zbytečné.

Poznámka: *Při čtení softwarové části, jsme se rozhodli nepopisovat zdrojový kód, ale pouze teoreticky ukázat na nutná rozhodnutí a řešení, které bylo třeba brát v úvahu. Celý zdrojový kód [Příloha 1] je příhodně komentovaný, a pro celkové pochopení doporučujeme vlastní implementaci.*

VÝVOJ NAHRÁVACÍ KOMPONENTY

Třída Recorder implementuje návrhový vzor Singelton, z toho důvodu, že v jednom čase může být otevřen pouze jediný ASIO ovladač a při komunikaci s ASIO ovladačem je potřeba statických *callback funkcí*, které takto získáme.

Samotná implementace komunikace s ASIO ovladačem byla vyvinuta s pomocí dokumentace z ASIO SDK [13] poskytovaného firmou *Steinberg*, která ASIO ovladače uvedla na trh.

Výhody ASIO vůči WMM [11] jsou hlavně ve nízké latenci vstupů a výstupů. Tento parametr pro nás nebyl rozhodující. Spíše pro nás byla výhodná možnost jednorázového zpracování vzorků až z 32 kanálů, zatímco WMM umožňuje z jednoho nahrávacího zařízení nahrávat pouze dva kanály (stereo). Tudíž by se v naší situaci musely otevřít 2 nahrávací zařízení pro 4 vstupní kanály.

V minulosti nám WMM přineslo problémy se synchronizací vzorků ze dvou zařízení. Synchronizace vzorků ze dvou zařízení v reálném čase se zdála zbytečně složitá.

Nevýhodou ASIO je nutnost hardwarové podpory u zvukové karty. Už delší dobu existuje projekt *ASIO4ALL* [18], který softwarově emuluje ASIO podporu libovolné zvukové karty, čímž tuto nevýhodu eliminuje. Dále zůstává pouze nevýhodně uzavřená licence od firmy *Steinberg*.

Závěrem této části bychom rádi vznesli jednoznačné doporučení:

„Pro jakékoliv synchronizačně náročné audio aplikace je ASIO ve většině případů mnohem výhodnější než WMM (pro OS Windows). WMM samo o sobě přináší více komplikací než užitku“

SAMPLESBUFFER

Jak již bylo uvedeno v Analýze problému, SamplesBuffer slouží k shromažďování nahrávaných vzorků do bloků které jsou následně předávány lokalizačnímu algoritmu.

VARIANTA A

Cílem tohoto bufferu je zajistit rychlý zápis a čtení kontinuálních dat. Pokud bychom měli nekonečnou paměť mohli bychom si dovolit nová data zapisovat stále na větší adresy. Protože tuto možnost nemáme a protože nepotřebujeme znát data starší než několik zápisů, můžeme si dovolit starší data umazávat a nahrazovat novýma.

Při realizaci bychom mohli využít klasické Double Buffering techniky, ale naším cílem je možná co největší kontinuita dat, tak abychom se mohli v poli plynule pohybovat vpřed a vždy očekávat navazující vzorky.

Vnitřní úložiště tvoří pole datového typu *fftw_complex* o velikosti *2*(OUTPUT_LENGHT+INPUT_LENGHT)* pro každý kanál [*Figure 3*], kde OUTPUT_LENGHT je délka výstupního segmentu pro FFT a INPUT LENGHT délka vstupního bloku vzorků. OUTPUT_LENGHT byl zvolen 8192 a INPUT_LENGHT vyplývá z nastavení komponenty *Recorder* - 1024.

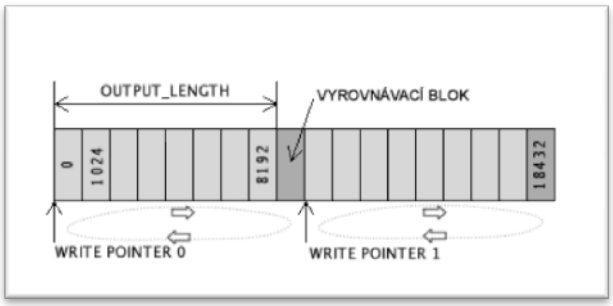


FIGURE 4 PRÁZDNÝ SB_A PRO 1 KANÁL

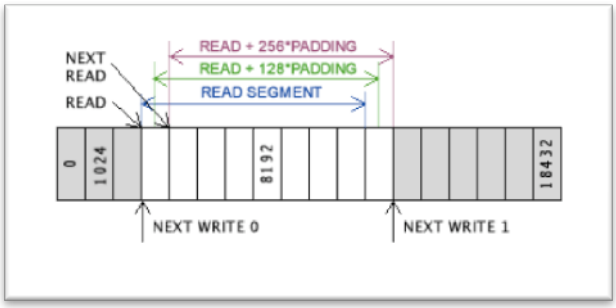


FIGURE 5 SB_A PŘED ZÁPISEM A PŘI ČTENÍ PRO PADDING = 4

Při přidávání nového bloku vzorků o velikosti INPUT_LENGHT je nejprve rozpoznáno zda je již naplněná první část bufferu, aneb je překonán počáteční – prázdný stav. Pokud plníme buffer poprvé vzorky se postupně kopírují jen do první části bufferu.

Pokud je již první část bufferu zaplněna tak jsou nová data zároveň zapisovaná do první i druhé části bufferu [*Figure 3*]. Vždy tak aby se nová data posunula v poli o INPUT_LENGHT doprava [*Figure 4*]. Pokud by mělo dojít k zápisu za hranice pole jsou zapisovací ukazatele znova nastaveny na začátek první a druhé části pole. Protože jsou data uložena dvakrát (První i druhá část bufferu), při čtení po přetočení data plynule navazují dál.

Když je čtecí ukazatel v na začátku druhé poloviny bufferu, a chtěl by se na další blok, tím pádem by konec výstupního bloku byl za hranicí pole, nastaví se čtecí ukazatel na začátek první části ve které jsou nové data, pouze na konci nám přibude INPUT_LENGHT nových dat.

Tento specifický druh zápisu nám po zapsání nových dat dává krátkou dobu mezi zápisy k přístupu k navazujícím segmentům o délce OUTPUT_LENGHT s maximálním překrytím INPUT_LENGHT směrem doprava. Tato doba je pro nás limitující a ve finální fázi znemožnila realizaci algoritmu A.

VARIANTA B

Algoritmus B nevyužívá překrývajících se výstupních segmentů, a proto je implementován pouze model double bufferingu, který využívá jeden buffer pro zápis a druhý pro čtení. Velikost bufferu jsme zvolili *4*INPUT_LENGHT* pro každý kanál. Datovým typem je *double*. Po naplnění psacího bufferu se role mění a z psacího se stává čtecí a naopak. Čerstvý čtecí buffer je předán algoritmu k zpracování.

Časově jsme zde omezeni dobou výměny bufferů, nicméně jsme implementovali sytém, který v případě že čtecí buffer ještě nebyl obsloužen zakáže výměnu bufferů a dochází k ztrátě dat.

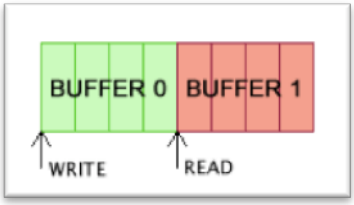


FIGURE 6 PRÁDNÝ SB_B

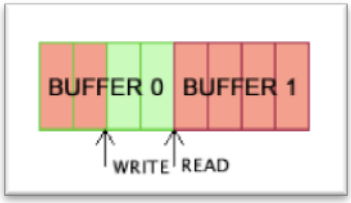


FIGURE 7 SB_B PŘED 3 ZÁPISEM

FIGURE 8 SB_B PO 4 ZÁPISU

IMPLEMENTACE LOKALIZAČNÍHO ALGORITMU A

Poté co je SB naplněn je potřeba spočítat v našem případě 256 FT pro data s PADDING=4. Ze kterých se následně odvodí výkonové spektrum jakožto absolutní hodnota výsledného komplexního čísla pro danou frekvenci.

Pro výpočet FT jsme po předběžném průzkumu usoudili, že nejvhodnější bude využít projekt FFTW [15]. Který v této oblasti patří k jedněm z nejrychlejších (dle benchmarků). Používá SSE2 instrukce a podporuje nastavení v závislosti na konkrétním procesoru. Před samotným zahájením výpočtů je vytvořen plán s určitou velikostí vstupního pole. Během tohoto tvoření, při použití FFTW_MEASURE, dojde k změření kritických výpočtů pro danou velikost a výsledky se použijí k rozhodnutí, která varianta výpočtů se využije.

V naší situaci jsme se rozhodli použít plán typu *fftw_plan_dtf_1d* , který slouží k výpočtu frekvenční domény z komplexního vstupu. My máme vstup reálný a proto komplexní část vstupních dat plníme 0. Toto také znamená, že pro maximální výkon je vhodné aby SB uchovával vzorky datového typu *fftw_complex*.

Pro získání výkonového spektra provedeme u komplexních výsledků výpočet absolutní hodnoty daného čísla, tím získáme výkon na dané frekvenci. Nyní máme na výběr co dále s tímto výsledkem uděláme. Jedním řešením je obdobný buffer jako u algoritmu B a následná křížová korelace. Druhou možností je přímá derivace a získání stacionárních bodů měnícího se spektra. Tyto body budou následně sloužit k určená TDOA mezi vstupy.

Vzhledem k nedostatečnému výkonu pro výpočet dostatečného množství FT v reálném čase jsme se se samotnou implementací dále nedostali, pouze bychom poznamenali, že jsme dokončili klasterizační algoritmus, který vykazoval, dle předběžných úsudků poměrně dobré výsledky.

Tento algoritmus byl implementován rekurzivní funkcí a instancemi třídy vector<Point>, kde Point je struktura uchovávající souřadnice bodu.

Implementace Lokalizačního algoritmu B

Během samotné implementace docházelo k častým změnám teorie, a proto i celého algoritmu B. Výsledkem je algoritmus popisovaný v teoretické části. Celý algoritmus je implementován v *Process* vláknu SB.

Jednou z důležitých optimalizací je vyjmutí výpočtu průměru (mean) vstupního signálu z algortimu pro křížovou korelaci, tím ušetříme poměrně znatelný výkon. Samotná křížový korelace nicméně zůstala stále pomalá.

Poté co provedeme křížovou korelaci, použijeme vlastní funkci *CriticalPoints* pro výpočet maxim této korelace. Základní úvahou je fakt, že v lokálním maximu funkce dochází ke změně derivace z kladné na zápornou. Přibližný výsledek derivace v daném bodě získáme jako rozdíl hodnot *y[i]-y[i+1],* protože jsme touto metodou získávali příliš mnoho stacionárních bodů došli jsme k výkonovému omezení, a experimentálně jsme došli k číselnému omezení maxima. (*Viz [Příloha 1], SB*).

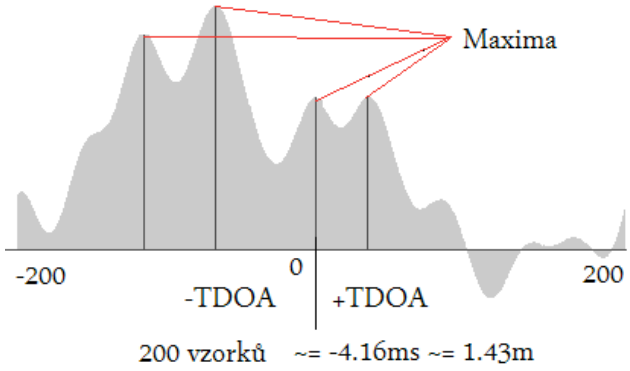


Figure 9 Skutečný výsledek algoritmu mezi kanálem 0 a 1

Poté co takto získáme maxima *[Figure 9]*, je potřeba je vůči sobě přiřadit a následně spočítat souřadnice. O vzájemné přiřazení se stará algoritmus, který hledá v ostatních kanálech takové maximum jehož velikost je co nerovnější velikosti maxima pro které hledáme pár. Tato metoda se ale neukázala jako ideální a proto doporučujeme vývoj jiného algoritmu, založeného na porovnávání reálnosti TDOA, tak aby nemohl být přiřazen nereálný pár.

Dalším problémem jsou ozvěny, které vytvářejí rázové vlny, jejichž křížová korelace má mnohem složitější průběh, tento fakt v podstatě zastavil vývoj algoritmu B.

Kvůli těmto komplikacím jsme pro prezentační potřeby zavedli lokalizaci nejhlasitějšího zdroje zvuku, pomocí určení TDOA jakožto největší křížové korelace.

Během vývoje jsme také došli k zajímavému zjištění ohledně úrovní jednotlivých signálů. Předpokládáme, že s elektretovými mikrofony je reálná i orientační lokalizace inspirovaná rozdílem amplitud – *ILD*.

Další komplikací je asi neúplně správné použití *Bucher Algorithm*. Po krátkých testech jsme chování výstupů shledali „divnými“. Pravděpodobně se jedná o problém se soustavami souřadnic, kdy originální algoritmus používá jinak orientované osy než zbytek projektu. A tudíž je výsledek lokalizace nestabilní a nehodnotitelný.

TRACKER

Podstatou Trackeru je tabulka aktuálních zdrojů zvuku. Ke každému zdroji si Tracker ukládá souřadnice, jedinečné ID a obnovovací koeficient (RK).

Když je zavolána funkce *AddSource(double x, double y, double z)* tracker projde tabulku zdrojů a pomocí vzorce pro eukleidovskou vzdálenost [*Equation 2*] v 3D prostoru určí vzdálenost mezi nově přidávaným zdrojem a aktuálně procházeným.

$$d = \sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2}$$

EQUATION 2

Kde x, y a z jsou souřadnice nového zdroje a x_i, y_i a z_i souřadnice aktuálně procházeného zdroje.

Pokud je jejich vzdálenost menší než kterou by zdroj zvuku urazil za dobu jednoho obnovení rychlostí v (nastaveno na 5*km/h)* tak v tabulce aktualizuje řádek aktuálního zdroje o nové souřadnice a nastaví jeho RK = 100. Pokud takto neupraví ani jeden řádek, je vytvořen nový s novými souřadnicemi a RK = 100.

Po každém obnovení dojde k průběhu tzv. CleanUp. Jedná se o algoritmus, který projde celou tabulku a všem zdrojům nastaví RK = RK-10. Následně ty zdroje, jejichž RK == 0 , z tabulky odstraní. Aktualizovaná tabulka je předána prezentační komponentě, která si uchovává pro zobrazovací účely svou vlastní kopii.

Samotná tabulka je realizována instancí třídy std::vector, tudíž je dynamicky zvětšována podle potřeby. Právě díky třídy vector je možné dynamicky mazat a přidávat nové „řádky“. Další výhodou tohoto řešení je, že samotná třída vector zaručuje lineární ukládání jednotlivých položek, tudíž toho můžeme využít při předávání do *managed* části při Interop komunikaci.

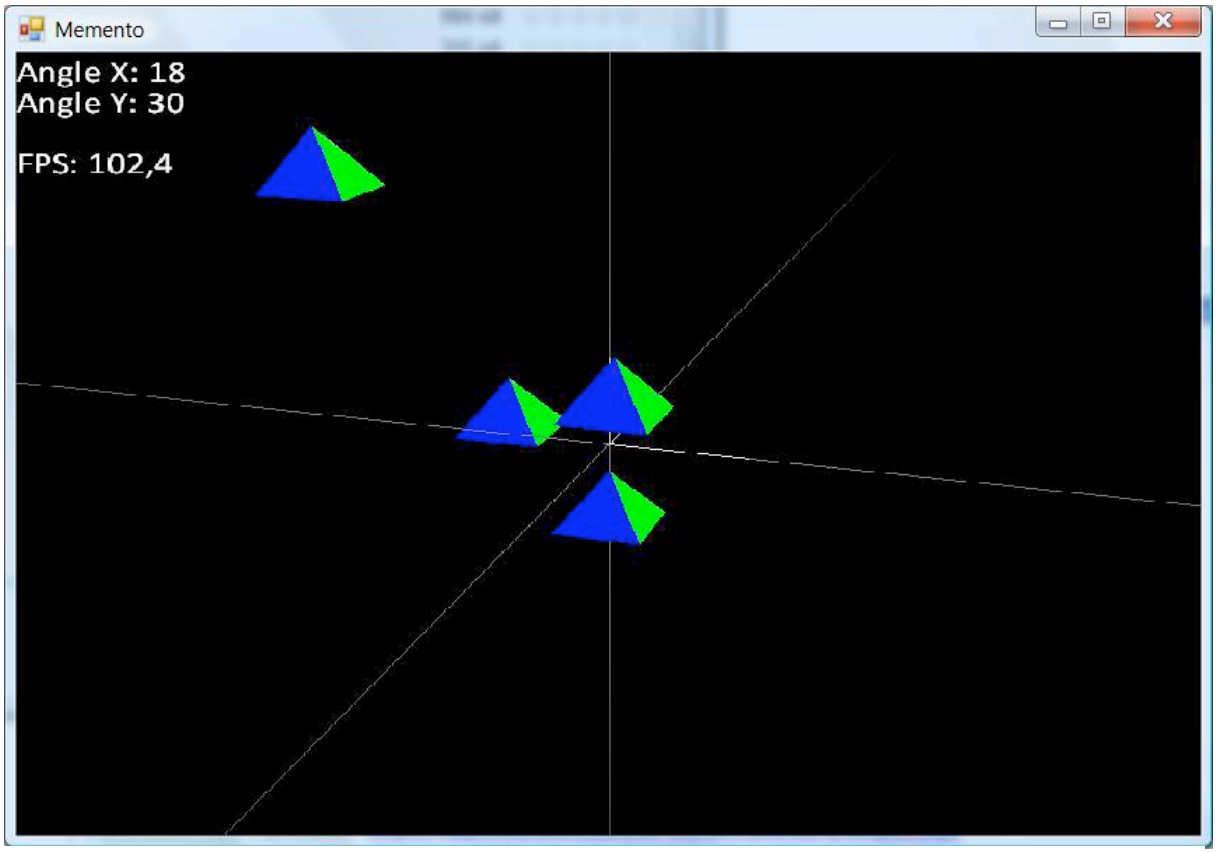
VÝVOJ PREZENTAČNÍ KOMPONENTY

Veškeré zpracované data se předají do prezentační komponenty, která na základě těchto dat vykreslí zdroje zvuku do našeho vesmíru. V orientaci ve vesmíru nám pomůže osový kříž, okolo kterého můžeme rotovat kamerou pomocí myši a s pomocí kolečka můžeme scénu přibližovat. Celá prezentační komponenta – Renderer – je naprogramována v C# - managed kódu.

VARIANTA A

Projekce probíhá v třídimenzionálním prostoru, proto jsme se rozhodli použít Direct3D [19] - konkrétně DirectX 9c. Novější verze - DirectX 10 pro nás nepřináší žádné zásadní změny, a vzhledem k nízké podpoře grafických karet v noteboocích, jsme se rozhodli zůstat u roky ověřené verze 9.

Kladné osy byly zvýrazněny tlustší čárou. Zdroje zvuků zde představovaly barevné pyramidky. V orientaci nám dále pomáhaly úhly, o které jsme byli otočeni vůči výchozí poloze.



RENDEROVÁNÍ POMOCÍ DIRECT3D 9C

Na vývoji této varianty jsme přestali pracovat, protože jsme přišli na to, že Direct3D je pro nás zbytečně složitý. Zatímco DirectX je komerční API pro hardware 3D akceleraci pod OS Windows, OpenGL je open source API, které poskytuje funkce pro renderování 2D a 3D grafiky, pokud je 3D akcelerace dostupná OpenGL ji využije. V každém případě nám vývoj této varianty přinesl cenné zkušenosti.

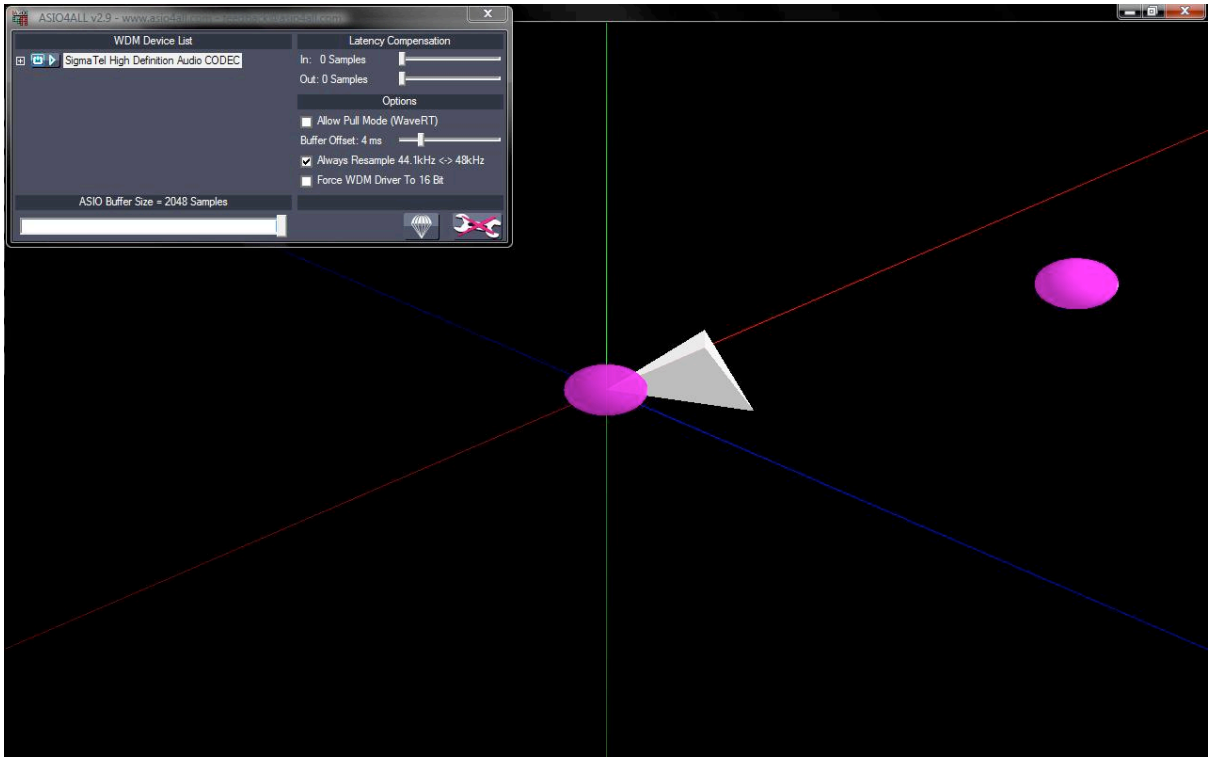
Pokud bychom se rozhodli Render kompletně postavit na DirectX, úplně bychom si uzavřeli cestu k portaci na jiný OS. Tím že *Renderer* zůstává na platformě .NET si sice také příliš nepomáháme, nicméně při případné portaci by většina algoritmů pro OpenGL zůstala zachována.

VARIANTA B

Vykreslovací jádro bylo přepsáno do OpenGL. Pro náš vývoj jsme si vybrali knihovnu Tao.OpenGL[20], které podporuje verzi GL 2.1 a GLU 1.3. Už během vývoje se ukázalo, že toto bude pro náš účel mnohem vhodnější a snazší než Varianta A, bez žádné pomocné knihovny jako je GLU.

Celé vykreslování funguje na principu Multibufferingu. Princip Multibufferingu spočívá v tom, že se scéna nekreslí ihned na obrazovku, ale nejprve se celá připraví v bufferu a ten se následně vymění s bufferem zobrazovaným na monitoru. Díky tomu je celé vykreslování o něco rychlejší a hlavně se nám nemůže stát, že by se scéna vykreslila neúplná nebo jinak poškozená, což by se v případě nepoužití této technologie stát mohlo.

My jsme se rozhodli pro Doublebuffering, který je pro nás dostačující. Double (dva) znamená, že máme dva buffery, které se střídají. Lze použít i větší množství (tzv. Triplebuffering, Quadbuffering, …).



RENDEROVÁNÍ POMOCÍ TAO.OPENGL

VARIANTA C

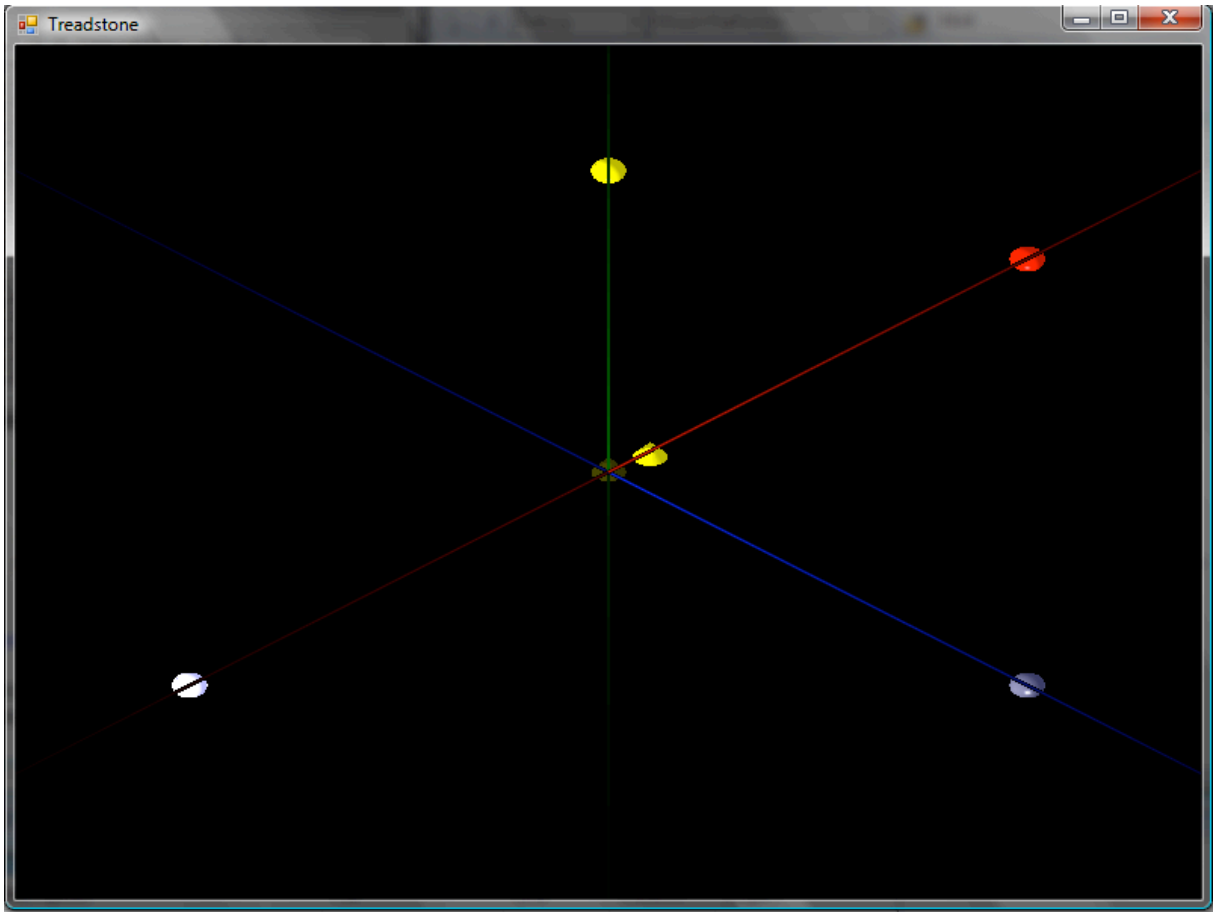
Při vývoji Varianty B jsme narazili na SharpGL[21], což je OpenGL vysokoúrovňová knihovna pro .NET. SharpGL by pro nás byla mnohem jednodušší než Tao.OpenGL, ale bohužel díky této jednoduchosti už nemáme tolik možností v nastavování. A pochopitelně je SharpGL řádově pomalejší.

Vývoj této Varianty C se zastavil prakticky okamžitě, protože už při vykreslování pouze několika jehlanů to více vytěžovalo procesor, než Varianta B. Z tohoto důvodu jsme se vrátili zpět k Variantě B.

FINÁLNÍ VARIANTA B

Finální úpravy Varianty B spočívaly ve snížení zatížení CPU. Nakonec se nám toto zatížení podařilo snížit na cca 4% jednoho jádra na Intel Core 2 Duo T8100 2,1GHz. S tímto vytížením jsme už spokojeni a prakticky jsme už vývojovou verzi prohlásili za finální.

Nakonec jsme dodělali ještě drobné, spíše kosmetické detaily jako možnost změny barvy vesmíru (klávesa B), změna barvy os (klávesa O), změna stylu os(klávesa S), zapínání a vypínání světla (klávesa L), zapnutí/vypnutí antialiasingu (klávesa A) a možnost navrátit se do startovní polohy (prostřední tlačítko myši).



FINÁLNÍ VERZE RENDEROVÁNÍ POMOCÍ TAO.OPENGL

Na osy jsou vykresleny pozice mikrofonů pro zachování měřítka, pro snazší orientaci mají tyto symboly stejnou barvu, jako orientační značky na mikrofonním poli.

Závěr

Po takto získaných zkušenostech je nám jasné, že lokalizace zvukových zdrojů není tak snadnou záležitostí jak jsme si zpočátku mysleli. Největším problémem je odlišit jednotlivé zdroje zvuku od ozvěn. Na tento problém jsme byli upozorněni Ing. Zdeňkem Nálevkou, bohužel jsme toto varování zanedbali a při náhradním návrhu a implementaci Algoritmu B v následné časové tísni jsme nebyli schopni toto řešit.

Jak se ukázalo, milióny let vývoje se nedají nahradit několika měsíční prací a očekávat stejně kvalitní výsledky. Tato práce nás nicméně inspirovala k úvaze o implementaci dalších, přírodou vytvořených mechanizmů, a dokonce soudíme, že je mnohem výhodnější vyhledat a následně uměle implementovat požadované, v přírodě se nacházející řešení, než vymýšlet nové čistě technické řešení. V našem případě by se jednalo o implementaci ANN místo křížové korelace.

Další možnosti vývoje

Toto je seznam témat, kterými by se dalo na tuto práci navázat a následně ji zpětně zdokonalit.

- Implementace Algoritmu A pomocí GPGPU
- Implementace ANN pro určení TDOA mezi dvěma signály
- Filtrace ozvěn
- Vlastní řešení převodu diskrétního signálu na měnící se power spectrum pomocí hledání periodicky opakujících se derivací
- Určení vzdálenosti od objektu pomocí dvou kamer ve známých souřadnicích a algoritmu pro detekci hran

REFERENCE A SEZNAM POUŽITÉ LITERATURA

[1] Wagenaar, Douglas J. Introduction to the Fourier Transform. 28 Feb. 2009. [online]
<http://www.med.harvard.edu/JPNM/physics/didactics/improc/intro/fourier1.html>

[2] Selesnick, Ivan. Short Time Fourier Transform. Connexions. 9 Aug. 2005. [online]
<http://cnx.org/content/m10570/2.4/>

[3] Wikipedia, The Free Encyclopedia. Sound Localization. 28 Feb. 2009. [online]
<http://en.wikipedia.org/w/index.php?title=Sound_localization&oldid=261938042>

[4] Wikipedia, The Free Encyclopedia. Multilateration. 28 Feb. 2009. [online]
<http://en.wikipedia.org/w/index.php?title=Multilateration&oldid=264957884>

[4] Wikipedia, The Free Encyclopedia. Cross-correlation. 28 Feb. 2009. [online]
<http://en.wikipedia.org/w/index.php?title=Cross-correlation&oldid=273252977>

[5] ESI. MAYA 44 USB. 28 Feb. 2009. [online]
<http://www.esi-audio.com/products/maya44usb/>

[6] Texas Instruments. INA217. 28 Feb. 2009. [online]
<http://focus.ti.com/docs/prod/folders/print/ina217.html>

[7] Bourke, Paul. Cross Correlation. Aug. 1996. [online]
<http://local.wasp.uwa.edu.au/~pbourke/miscellaneous/correlate/>

[8] Bucher, Ralph. Misra, D. A Synthesizable VHDL Model of the Exact Solution for Three-dimensional Hyperbolic Positioning System. VLSI Design, 2002 Vol. 15 (2), pp. 507–520.
<http://ralph.bucher.home.att.net/project.html>

[9] Matteucci, Matteo. A Tutorial on Clustering Algorithms. 1 Mar. 2009. [online]
<http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/>

[10] Wikipedia, The Free Encyclopedia. K-means algorithm. 1 Mar 2009. [online]
<http://en.wikipedia.org/w/index.php?title=K-means_algorithm&oldid=270855919>

[11] Microsoft Corporation. Windows Multimedia Start Page. MSDN. 28 Feb. 2009. [online]
<http://msdn.microsoft.com/en-us/library/ms713771(VS.85).aspx>

[12] Microsoft Corporation. .NET Framework Developer Center. MSDN. 28 Feb. 2009. [online]
<http://msdn.microsoft.com/en-us/netframework/default.aspx>

[13] Steinberg Media Technologies GmbH. Third-Party Developers. Steinberg. 28 Feb. 2009. [online]
<http://www.steinberg.net/en/company/3rd_party_developer/>

[14] Philpott, Rob. Low Latency Audio using ASIO Drivers in .NET. 28 Feb. 2009 [online]
<http://www.codeproject.com/KB/audio-video/Asio_Net.aspx>

[15] FFTW. FFTW Homepage. 3 Feb. 2009. [online]
<http://www.fftw.org/>

[16] Microsoft Corporation. Win32 and COM Development. MSDN. 28 Feb. 2009. [online]
<http://msdn.microsoft.com/en-us/library/aa139672.aspx>

[17] GPGPU. General-Purpose Computation Using Graphics Hardware. 28 Feb. 2009. [online] <http://www.gpgpu.org/>

[18] ASIO4ALL. ASIO4ALL - Universal ASIO Driver For WDM Audio. 28 Feb. 2009. [online] <http://www.asio4all.com/>

[19] Microsoft Corporation. DirectX: Advanced Graphics on Windows. 28 Feb. 2009. [online] <http://msdn.microsoft.com/en-us/directx/default.aspx>

[20] The Tao Framework. The Tao Framework. 28 Feb. 2009. [online] <http://www.taoframework.com>

[21] SharpGL. SharpGL. 28 Feb. 2009. [online] <http://www.sharpgl.com>

KONZULTACE

- Ing. Zdeněk Nálevka <z.nalevka@spseiostrava.cz>
    - 21. 11. 2008 - Harmonická analýza + Fourierova řada a transformace
    - 27. 11. 2008 - Zapojení elektretového mikrofonu a jeho napájení
    - Dec. 2008 – Kontrola zapojení přístrojového zesilovače
- Mgr. Radek Nowak <r.nowak@spseiostrava.cz>
    - 4. 12. 2008 – Klasterizační algoritmy
    - Jan. 2008 – Fourierova transformace – komplexní čísla
- Mrg. Vladimíra Ogořalková <v.ogoralkova@spseiostrava.cz>
    - 27. 3. 2009 – Řešení soustavy eukleidovských rovnic

PŘÍLOHY

1. Příloha 1 – DVD
    a. /Blackbriar/ - MS Visual Studio 2008 Solution
    b. /SamplesBuffer.cpp , /SamplesBuffer.h – Implementace SB A
    c. /Dokumentace.doc – Elektronická podoba dokumentace