VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

TVORBA GRAFICKÉ KNIHOVNY PRO ZÁSUVNÉ MODULY VST
CREATION OF THE GRAPHIC LIBRARY FOR VST PLUG-INS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. Filip Dufka

VEDOUCÍ PRÁCE
SUPERVISOR
RNDr. Lubor Přikryl

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Audio inženýrství**
Ústav telekomunikací

Student: Bc. Filip Dufka ***ID:*** 174447
Ročník: 2 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Tvorba grafické knihovny pro zásuvné moduly VST

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte možnosti zobrazování grafického uživatelského rozhraní pro zvukové zásuvné moduly s důrazem na 3D vykreslování. Zvolené metody implementujte v jazyce C++ do grafické knihovny pro technologii VST3, která bude zvládat vykreslování obrázků, 3D objektů, 3D grafů, a to pokročilejšími zobrazovacími metodami. Vyberte zvukový efekt, který implementujte v prostředí Matlab nebo Python.
Prostudujte metody a popište je z pohledu výpočetní náročnosti a realizujte několik základních prvků knihovny. Ke grafické knihovně vytvořte kompletní dokumentaci.
Ze zvukového efektu vytvořte zásuvný modul technologie VST3, který bude využívat vámi vytvořenou grafickou knihovnu, a rozšiřte ji o možnost uživatelské interakce.

DOPORUČENÁ LITERATURA:

[1] SYSEL, P.; SMÉKAL, Z. Číslicové filtry. Brno: Vysoké učení technické v Brně, 2012. s 1-148. ISBN 978-8-2-4-4454-6.

[2] WHITROW, R. J. OpenGL graphics through applications. New York: Springer, 2008. ISBN 978-1-84800-023-0.

Termín zadání: 1.2.2019 ***Termín odevzdání:*** 16.5.2019

Vedoucí práce: RNDr. Lubor Přikryl
Konzultant:

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

Abstrakt

Diplomová práce se zabývá grafickým uživatelským rozhraním zvukových zásuvných modulů. V úvodní části je popsána struktura zvukových zásuvných modulů a poté popsány aktuální techniky a praxe vykreslování editorů zvukových parametrů na příkladech grafických knihoven sloužících k těmto účelům. Popsána je jejich neefektivnost a paměťová náročnost. Ve čtvrté části jsou grafické vykreslovací možnosti porovnány s dynamičtějšími technologickými obory — grafickým a herním průmyslem. Podrobeny analýze jsou aktuální zobrazovací techniky pro aplikace běžící v reálném čase a posuzována je jejich možnost začlenění do editorů zvukových zásuvných modulů.

Další část je věnována řešení neefektivních technik zmíněných v první části. Navrženy jsou dvě metody s cílem dosáhnout realistického vzhledu bez zvýšení výkonové náročnosti. První metoda nabízí obecné řešení využitím 3D grafiky a grafických akcelerátorů s vykreslením pomocí „odloženého stínování“. Druhá představená metoda nahrazuje nutnost použití velkého množství (v řádech stovek) bitmapových obrázků pro realizaci otočení virtuálního potenciometru na operaci jednodušší — vyžadující pouze šesti obrázků pro vykreslení srovnatelného výsledku. Tato metoda byla v práci pojmenována jako „metoda normálových map“.

Součástí práce bylo vytvoření grafické knihovny RealBox, jejímž základem jsou navržené metody. Knihovna dovoluje uplatňovat jak prostorový, tak 2D přístup k uživatelskému rozhraní s vizí zrychlit a zjednodušit současné pracovní postupy. Ke knihovně byla vytvořena kompletní dokumentace popisující veškeré funkce, které knihovna nabízí.

Použitelnost knihovny je demonstrována na třech zvukových zásuvných modulech technologie VST3, které byly v rámci diplomové práce vytvořeny. Důraz byl kladen na vykreslovací možnosti a kvalitu editoru zvukových parametrů a na rychlost práce. Vypracovaná grafická knihovna tedy nabízí alternativu současným zastaralým technikám a základ pro další zlepšování grafických uživatelských rozhraní zvukových zásuvných modulů.

Klíčová slova

Digitální zpracování zvuku, grafické uživatelské rozhraní, odložené stínování, OpenGL, VST3, vykreslování v reálném čase, zvukové zásuvné moduly.

DUFKA, Filip. *Tvorba grafické knihovny pro zásuvné moduly VST*. Brno, 2019. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/118152. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce Lubor Přikryl.

Abstract

Master‘s thesis covers use of graphical user interface in audio plug-ins. In first part structure and rendering techniques of audio plug-ins graphical libraries are described. Their efficiency and their way of memory utilization is questioned. Next part puts these techniques in comparison with the state of the art methods used in computer graphics and gaming industry. Their possible use in audio graphical interafaces is analyzed.

In the following part, thesis finds solutions to uneffective methods in frequently used situations by presenting deferred shading into audio parameter editor with the goal of photorealistic rendering. Second introduced technique of „Knob Normal Maps“ reduces number of images needed for rendering of turning knob from hundereds to six with comparable results.

Goal of diploma thesis was to create graphical library. Graphical library with name RealBox was created, and introduced techniques are the core features. Library reduces work needed to achieve graphical user interfaces for 2D and 3D cases of use. Full class and method documentation for RealBox library was assembled.

Library was tested during creation of three VST plugins, with different approaches and emphasis on quick work and fine rendering. Graphical library offers new, faster way of creating audio plug-in interfaces.

Keywords

Digital audio processing, graphical user interface, deferred shading, OpenGL, VST3, real-time rendering, audio plug-ins.

Prohlášení

Prohlašuji, že svou diplomovou práci na téma „*Tvorba grafické knihovny pro zásuvné moduly VST*“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následku porušení ustanovení § 11 a následujících autorského zákona c. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změne některých zákonů (autorský zákon), ve znění pozdejších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne 16. května 2019.

. . . . . .
podpis autora

Poděkování

Děkuji vedoucímu diplomové práce RNDr. Luboru Přikrylovi za velmi podnětné připomínky, dlouhodobou motivaci a inspiraci. Dík také patří celé mé rodině a Elišce za bezmeznou trpělivost a obětavou pomoc.

V Brně dne 16. května 2019.

. . . . . .
podpis autora

Obsah

Seznam obrázků

1 Úvod

Digitální zpracování zvuku se vyznačuje modulárním způsobem práce. Zvukový signál prochází ze vstupu signálovými procesory, které upravují nebo analyzují jeho průběh na výstup tohoto řetězce. V digitálním prostředí je tento tok realizován nejčastěji v tzv. *Digital Audio Workstation* (DAW), který umožňuje míchat výstupy několika řetězců dohromady.

Tok zvukového signálu je ovlivňován softwarovými komponenty nazývanými *Zvukové zásuvné moduly* či *plug-iny*. V mnohých případech se jedná o složité výpočetní struktury, jejichž použití není triviální. Z toho důvodu je nutné, aby zvukové zásuvné moduly měly natolik dostatečně jednoduché a intuitivní ovládání, aby uživatel co nejrychleji pochopil funkci zvukového efektu a mohl ho plně využít.

Toho lze nejlépe dosáhnout nabídnutím uživateli vhodně strukturované, graficky přívětivé vizuální stránky zvukového pluginu. Dosavadní technologický vývoj se oprávněně soustředil hlavně na zvukový aspekt zvukového zpracování, s faktem, že grafická podoba vznikala často neefektivním a neobratným způsobem.

Cílem této práce je na tato neelegantní řešení poukázat a navrhnout nové způsoby, jak grafické uživatelské rozhraní řešit s inspirací v herním a grafickém průmyslu. Součástí práce je vytvoření grafické knihovny, která proces tvorby vizuální podoby zvukového zásuvného modulu vývojářům usnadní, a dovolí jim se dále soustředit na kvalitu zvukového zpracování.

Protože je cíl práce koncipován jako výchozí bod pro další vývojáře zabývající se tvorbou uživatelského prostředí, budou funkce a výhody této grafické knihovny pro názornost demonstrovány celkově na třech zvukových zásuvných modulech: NormalKnobs, RealStomp a ReproSpace.

2 Zvukové zásuvné moduly

Zvukové zásuvné moduly jsou dynamicky linkované knihovny, jejichž funkcionalita je vyvolávaná hostitelskou aplikací. V těchto knihovnách se nacházejí výpočetní funkce, které vytvářejí, upravují nebo analyzují zvukový signál.

Moduly mohou mít podobu zvukového efektu, jehož úkolem je změnit zvukový signál podle potřeb uživatele. Jedná se například o filtry, modulátory, dozvukové generátory, úpravu dynamiky a další.

Další možností, jak využít zvukových zásuvných modulů je tvorba virtuálních hudebních nástrojů. Ty vytvářejí zvukový signál na základě podnětu MIDI zpráv, které definují základní hudební vlastnosti (výška a trvání tónu).

Vnitřní strukturu zvukového zásuvného modulu můžeme obecně rozložit na dvě části: **procesní část** a **kontrolní část**, která bývá pro přehlednost ještě obohacena o **editor zvukových parametrů.** [1] Jejich popisem se budou zabývat následující kapitoly 2.2 a 2.3.

2.1 Virtual Studio Technology

Virtual Studio Technology (zkráceně *VST*) je jednou z dominantních technologií, které se zvukovými zásuvnými moduly zabývají. Je vyvíjena a spravována firmou Steinberg, a nabízí multiplatformní a dostupnou variantu pro vývojáře hostitelských aplikací i zvukových zásuvných modulů.

Alternativami k této technologii jsou například **AAX** (*Avid Audio Extension*) od společnosti Avid a **AudioUnit** od společnosti Apple. V této práci bude však popisována pouze struktura a využití VST, a to konkrétně verze VST3.

Hlavní předností VST3 je strukturace procesní části a kontrolní části v dva nezávislé celky, které spolu komunikují pouze skrz hostitelskou aplikaci. Tato zdánlivá komplikace dovoluje rozložit výkon počítače do těchto dvou celků s výsledkem zrychlení většiny operací a větší možnosti hostitelské aplikace vstoupit do chodu zásuvného modulu.

2.2 Procesní část

Souboru funkcí, která má na starosti úpravu zvuku, říkáme **procesní část**. Do procesní části vysílá hostitelská aplikace vzorky zvukového signálu hromadně po blocích — těm se také říká *zásobníky (buffery)*. Procesní část projde veškeré vzorky, které jsou reprezentovány číselnou hodnotou, a v pozměněné či nepozměněné formě je uloží do výstupního zásobníku hostitelské aplikace, aby mohly býti dále upravovány či přehrány.

Velikost zásobníku je definována uvnitř zvukového ovladače systému a je důležitým prvkem při zpracování signálu, protože dokud není ukončena operace zpracování celého bloku, nemůže být sekvence vzorků přehrána ani dále zpracována. Příliš malá velikost může způsobit zahlcení procesoru množstvím instrukcí (každé volání zabere jisté množství „režijního“ času), a naopak příliš dlouhý zásobník vytvoří zpoždění mezi vstupem a výstupem.

Požadavky na procesní část jsou kladeny při vytvoření zvukového zásuvného modulu, a někdy také při změně těchto požadavků. Kromě předání informace o velikosti zásobníku a vzorkovací frekvenci je zde také definován počet vstupních a výstupních kanálů. Procesní část by měla být připravena na veškeré kombinace kanálů, avšak v případě nesplnění podmínek vstupních a výstupních kanálů může procesní část odmítnout signál zpracovávat.

2.2.1 Módy zpracování

Podle rychlosti předávání vstupních dat můžeme definovat dva módy, na které by měla být procesní část připravena: zpracování v reálném čase a offline zpracování.

Při **zpracování v reálném čase** se počítá s faktem, že signál bude ihned po úpravě uživateli zpět přehrán. V ideálním případě je signál zpracován rychleji, než je přehráván, a tedy část času je procesní část nečinná.

Avšak vzniká zde již zmiňované zpoždění, které je určené výpočetní náročností zvukového řetězce a velikostí zásobníku. Aby tato latence nebyla slyšitelná pro uživatele (a hlavně pro

interpreta) který zvukovým řetězcem upravuje zvuk hraného nástroje, neměl by čas zpracování jednoho zásobníku skrz veškeré zásuvné moduly překročit hodnotu 20 ms.

Naopak **offline zpracování** je využíváno při ukládání a vypočítání hotové zvukové stopy. Zvukový signál je posílán do procesní části ve chvíli, kdy zpracoval předchozí zásobník, protože není nutné čekat na chvíli, kdy má být zásobník přehrán. [1]

2.2.2 Změna parametrů

To, jakým způsobem mají být jednotlivé vzorky zpracovány může být definováno dopředu vývojáři zvukového modulu. Častěji je však vyžadováno, aby parametry zpracování byly proměnlivé, a nejlépe uživatelem upravovatelné.

Z tohoto důvodu bývají vstupním argumentem u každého zpracování zásobníku kromě vstupních vzorků také události změny parametrů. V nejlepším zájmu procesní části je si tyto nové hodnoty parametrů uložit a popřípadě z nich přepočítat vnitřní proměnné. Protože změna parametru může být pozvolně měněna, je posílán kromě výsledné hodnoty parametru také zásobník průběhu změn tak, aby procesní část mohla s touto informací volně pracovat.

Parametry mohou být změněny dvojím způsobem. Buď je upraví hostitelská aplikace například pomocí automatizace, nebo tak učiní uživatel skrz kontrolní část (viz 2.3) a editor zvukových parametrů (viz 3).

Automatizací parametrů nazýváme operaci, která zaznamenává hodnoty parametrů v čase v hostitelské aplikaci, a poté je předává procesní i kontrolní části zvukového zásuvného modulu. Takto uložené hodnoty jsou ukládány do stop (častěji nazývané *automatizační obálky*), nebo jsou definovány matematickou funkcí — vždy podle možností hostitelské aplikace.

Důležitou informací je fakt, že hostitelská aplikace má přednostní právo upravovat parametry modulu, nicméně dovoluje reakci na uživatelské změny v několika režimech: *Read, Touch, Latch, Write, Trim* nebo *Relative*.

2.3 Kontrolní část

Kontrolní část zastřešuje veškerou činnost spojenou s úpravou proměnných parametrů zvukového modulu. Parametry jsou zde vytvářeny a předány hostitelské aplikaci, aby s nimi mohla libovolně pracovat. Veškeré parametry jsou v jádru zastoupeny číselnou hodnotou, která bývá nejčastěji číslem s pohyblivou desetinou čárkou.

Číslem s pohyblivou destinnou čárkou je parametr i v případě, že zastupuje například celé číslo nebo binární stav. Kromě toho bývá číslo normalizované do rozmezí [0,1]. Hostitelská aplikace v takovém případě není schopna jakkoliv interpretovat parametry pro uživatele. Interpretace je tedy další zodpovědností kontrolní části.

Pro každý z typů parametrů by uvnitř kontrolní části měla existovat definice toho, jak má být normalizovaná hodnota parametru interpretována pro vypsání textového řetězce na obrazovku nebo pro vnitřní zpracování uvnitř zvukového zásuvného modulu.

Kontrolní část definuje jméno parametru, základní hodnotu parametru, způsob vypsání textového řetězce, způsob převedení textového či číselného řetězce na normalizovanou hodnotu, omezení pro úpravu hodnoty a další. Je také záhodno, aby uživatel nemusel po každém spuštění zvukového zásuvného modulu znovu nastavovat hodnoty parametrů, a proto je úkolem kontrolní části ukládat a načítat stavy parametrů.

Hostitelská aplikace má na starosti uživateli nabídnout uživatelské prostředí, ve kterém může uživatel parametry měnit. K takové situaci málokdy dojde, protože kontrolní část může vytvářet vlastní rozložení uživatelského prostředí vytvořením **editoru zvukových parametrů**, a vývojáři zvukových zásuvných modulů této možnosti ve většině případů využívají. [1]

3 Editor zvukových parametrů

Editor zvukových parametrů dovoluje vývojáři přesně definovat rozložení a vizuální reprezentaci uživatelských prvků. Kdyby tato činnost byla plně svěřena hostitelské aplikaci, která o funkci plug-inu má minimální představu, byly by veškeré parametry reprezentovány číselnými posuvníky, a to i v případech, že se jedná binární hodnotu nebo parametr, který je v této podobě špatně představitelný.

Editor je volán kontrolní částí zvukového zásuvného modulu a umožňuje volnou komunikaci mezi těmito dvěma komponentami, na rozdíl od komunikace mezi procesní a kontrolní částí v případě technologie VST3.

Editor zvukových parametrů by měl dovolit uživateli upravovat parametry a vidět změny, které provedl. To činí pomocí uživatelských prvků, které jsou napojeny na vnitřní parametry kontrolní části.

Rozhodnutí, které uživatelské prvky vůbec nechat uživatele měnit, řeší designový obor **UX Design** (*User Experience Design*). Zvukový zásuvný modul může být ve své struktuře často velmi komplexní a pro uživatele matoucí. Proto je důležitým úkolem UX designera některé zvukové parametry buď spojit nebo zjednodušit do menší sady parametrů, které uživateli dovolí dostatečnou kontrolu a zároveň přehlednost.

Možné hodnoty parametru by v uživatelském rozhraní měly mít také vhodnou vizuální reprezentaci. Disciplína nazývaná **UI Design** (*User Interface Desgin*) se tedy snaží vytvořit nejen vhodný vzhled uživatelského prvku, aby jeho pohyb či úprava respektovala funkčnost prvku, ale také vhodné rozložení těchto prvků v editačním okně tak, aby uživatelské prvky byly seskupeny do logických celků.

UX a UI Design spolu vytváří část zvukového modulu, která je viditelná pro uživatele. Ten na základě objektivních a subjektivních faktorů rozhoduje, zda si daný zvukový zásuvný modul koupí a bude ho efektivně používat. Mezi objektivní faktory samozřejmě patří funkčnost zvukového efektu, zda plní svojí primární funkci. Rozhodujícím prvkem pro výběr mezi efekty stejného druhu je ale už subjektivní měřítko, kterým je hodnocen výsledný zvuk. Často je porovnáván charakter výstupního zvuku a vhodnost použití pro různé situace.

Dalším faktorem je dostatečná kontrola uživatele nad procesní částí zvukového efektu — tedy pole působnosti UX designu. Dostatečné množství parametrů dovoluje uživateli nabýt dojmu, že nad zvukovým procesorem má plnou kontrolu; v případě malého či příliš vysokého počtu parametrů tento dojem může lehce ztratit.

Jedním ze subjektivních faktorů pro výběr zvukového zásuvného modulu je přívětivost nebo působivost jeho uživatelského prostředí. V případě, že má zvukový efekt zastaralý či nemoderní vzhled, je přirozeným úsudkem uživatele, že i jeho funkčnost nevyužívá současných technologií. V jiných situacích je naopak záhodno, aby uživatel měl pocit, že využívá starších analogových technologií, byť se nachází v digitálním prostředí (viz kapitola 3.3).

V obou případech je úkolem designera navrhnout vhodné uživatelské rozhraní, aby těchto dojmů uživatel nabýval, a zároveň mu byla poskytnuta veškerá čitelnost chodu zvukového procesoru. V současnosti je běžným standardem, že zvukové zásuvné moduly dosahují vzhledu uživatelského rozhraní užitím neefektivních a kompromisních prostředků.

Cílem této diplomové práce je proto popsat dosavadní techniky, a navrhnout nové — paměťově a výkonově úspornější cesty pro vykreslování parametrových editorů zvukových zásuvných modulů.

3.1 Dostupné technologie pro vykreslování uživatelských rozhraní

Navrhované přístupy technik vykreslování uživatelských rozhraní jsou podmíněny dostatečnou softwarovou podporou, která dovolí jak programátorům, tak grafikům využívat možnosti doby i v rámci editačních oken zvukových zásuvných modulů.

V cestě stojí nejen vykreslovací možnosti pro uživatelské prvky, které by bez problémů mohly zvládnout specializované grafické či herní enginy, ale taky rozhraní zásuvných modulů, které určují, jak spolu bude editační okno a procesní část zvukového procesoru komunikovat.

V případě, kdy je vývojářem definováno uživatelské prostředí v podobě editoru zvukových parametrů, odkáže hostitelská aplikace na ukazatel okna přidělený tomuto zásuvnému modulu. Zde je asi jediný rozdíl mezi vytvářením okna standalone aplikace a okna zásuvného modulu. V případě samostatné aplikace je ukazatel na okno přidělen aplikaci přímo operačním systémem, v případě zásuvného modulu zprostředkovaně skrz hostitelskou aplikaci.

3.1.1 Knihovna VSTGUI

Třídy a funkce pro zobrazování uživatelských zobrazení parametrů není přímou součástí technologie VST3, ale jsou obsaženy v knihovně **VSTGUI**, která je distrubuovaná společně s VST3 jako součást vývojářského balíku.

Knihovna VSTGUI je vývojářským nástrojovým balíkem pro tvorbu uživatelských rozhraní zvukových zásuvných modulů nejen technologie VST, ale také i AAX a AudioUnit. Knihovna podporuje veškeré techniky, které budou zmíněné v kapitole „Techniky vykreslení

uživatelských prvků“ na stránce 15 a dovoluje tyto prostředky prezentovat na platformách Windows a MacOS.

Uvnitř této knihovny je třída *COpenGLView*, která, jak už název napovídá, dovoluje vytvoření OpenGL kontextu - základního komunikačního prostředku s grafickým akcelerátorem. To vytváří základ pro využití aktuálnějších zobrazovacích prostředků, nikoliv však hotové řešení připravené pro rychlou implementaci. [1]

3.1.2 Aplikační rámec JUCE

Pokročilejší možností je meziplatformní framework **JUCE** (*Jules' Utility Class Extensions*), který je určený pro vytváření hudebních modulů a hudebních aplikací. Vyznačuje se svojí filozofií, kdy napsaný kód je možné cílit jak na Windows, tak Max OS X a Linuxové platformy, a zvukové moduly je možné „'zabalit"' do formy VST pluginu, AAX pluginu nebo AudioUnit pluginu. Toho je docíleno vyšší mírou abstrakce, kdy je o nižší úroveň kódu (tu platformě závislou) postaráno frameworkem JUCE.

JUCE nabízí veliké množství tříd pro správu například síťových protokolů, kryptografie, vícevláknového přístupu k aplikaci, ale hlavně správu grafických uživatelských prvků a balíků pro úpravu zvuku, předpřipravených pro rychlé použití. Neobsahuje však taktéž žádné prostředky pro pokročilejší vykreslení.

Framework JUCE je vhodné používat s předpřipravenými nástroji — aplikacemi, které tvorbu pluginů a standalone aplikací urychlí. Jedná se o editor grafického uživatelského rozhraní a o generátor souborů připravených pro distribuci. jen

3.1.3 Knihovna WDL

Další alternativou je open source C++ knihovna **WDL**. Její vývoj má na starosti společnost Cockos Inc., která stojí za DAW Reaper. Knihovna WDL se snaží být jednoduchou, ale pomocnou knihovnou, která má pomoci při vytváření hudebních efektů a zvukových zásuvných modulů. Účel knihovny je být dostatečně nápomocná, ale ne tak, aby vývojář kvůli ní byl nucen používat množství dalších pomocných tříd.

Knihovna sama o sobě obsahuje pomocné třídy pro správu paměti, polí, textových řetězců, podporu načítání zdrojových souborů, ale hlavně zvukové funkce, jako například FFT řešení, konvoluční funkce, sinusové generátory, převaděče z a do decibelového relativního měřítka.

Knihovna WDL se sestává z několika menších knihoven, každé specifikované na něco jiného. Například knihovna pro úpravu zvukového signálu obsahuje spoustu dalších tříd a funkcí pro správu zásobníků, úpravu výšky signálu či kódování a dekódování zvukového signálu. To vše se snahou být co nejmenší možnou knihovnou, která výslednému zvukovému efektu nepřidá na velikosti.

V oblasti grafického uživatelského rozhraní má také co nabídnout prostřednictvím knihoven LICE a Plush. Knihovna LICE dovoluje načítání a ukládání bitmapových obrázků skoro všech možných formátů, jejich zobrazování a operace s nimi a s jejich barevnými modely.

Naopak knihovna Plush2 je softwarovým 3D vykreslovacím enginem, který dovoluje vykreslování scén, správu kamer, textur a materiálů, import 3D souborů a mnoho dalšího.

Nicméně podpora grafických knihoven u tohoto balíku, se zdá, že se zastavila v roce 2014 a množství dokumentace je minimální.[8]

3.2 Techniky vykreslení uživatelských prvků

Vhodné vykreslení editačního rozhraní je definováno především typem parametrů, které zvukový zásuvný modul obsahuje. Vzhled uživatelského prvku se odvíjí od charakteru zastupovaného parametru a také zažitých zvyklostí zvukařů či hudebníků, kteří s tímto parametrem pracují. Uživatelské prvky lze dělit dle přímého vlivu uživatele na dvě kategorie: **pasivní** a **aktivní**.

Způsob vykreslení uživatelských prvků je definován implementací grafických knihoven, a je často závislý na potřebách vývojářů, grafickém stylu a potřebné rychlosti zobrazení. Níže budou popsány způsoby použité v grafických knihovnách a editorech zvukových zásuvných modulů.

3.2.1 Pasivní uživatelské prvky

Pasivní uživatelské prvky slouží uživateli pouze k indikaci. Jejich stav a hodnota reflektuje hodnoty proměnných nebo stav zvukového zásuvného modulu. Hlavním rozdílem oproti aktivním uživatelským prvkům je nemožnost s nimi jakkoliv interagovat. Mezi hlavní zástupce patří například *texty, diody, indikátory úrovní či grafy.*

A. Texty

Jedním z nejzákladnějších způsobů, jak prezentovat uživateli informace o stavu zvukového efektu, je vypsání textového řetězce. Ten je ve VSTGUI vykreslen vždy základní grafickou knihovnou pro danou platformu. Například pro Windows využívá VSTGUI grafické knihovny Direct2D, pokud je přítomna. Pokud není, je použita starší knihovna GDI+.

Podobně je řešena podpora na operačním systému Linux, kde je využita knihovna Cairo. Na operačních systémech Mac se využívá součástí Cocoa a Carbon. [1] K těmto platformově závislým funkcím je ve VSTGUI vytvořena abstraktní vrstva, která zabaluje veškeré závislé funkce do jedné třídy spravující grafický kontext editoru parametrů.

V pozadí však knihovny fungují jinak, například starší GDI+ na rozdíl od Direct2D nevyužívá hardwarové akcelerace při vykreslování základních grafických prvků a textů. [16]

B. Diody

Diody jsou nejjednodušším možným prvkem pro vykreslení, protože mohou nabývat pouze dvou stavů - vypnuto, zapnuto. Toho je dosahováno dvojicí bitmapových obrázků, popřípadě změnou barvy u jednoduchého objektu (např. kruhu či čtverce).

Příkladem použití diody může být indikátor aktivity zvukového efektu (vypnut, zapnut), případně indikátor přesáhnutí určité prahové úrovně signálu (přesáhnuto, nepřesáhnuto).

C. Indikátory úrovní

Indikátory úrovně slouží k podrobnějšímu znázornění parametru, jehož hodnota se pohybuje v jasně daném rozmezí hodnot. Nejčastější využitím je realizace znázornění hlasitosti zvukového signálu, kde řídícím parametrem je efektivní hodnota hlasitosti

Indikátor bývá realizován pomocí jednoduchého objektu čtverce, kterému je při změně vlastnosti změněna výška. V situacích, kdy je volen realistický styl zobrazení, se musí přikračovat ke kompromisům.

Prvním řešením je umístění několika diod nad sebe, každá s jinou prahovou hladinou zapnutí a vypnutí, díky čemuž dojde k vytvoření sloupce, který reaguje na změnu parametru. Nevýhodou je nutnost vytváření množství těchto prvků a také definování rozhodovacích úrovní pro veškeré diody.

Druhou stejně málo elegantní variantou je použití sady obrázků, kde každý obrázek zachycuje hladinu parametru v určitém stavu. Podle aktuální hodnoty parametru se pak jednotlivé obrázky přepínají a lze tak dosáhnout působivějších a realističtějších zobrazení, avšak za ztráty důležité části paměti. Pro tyto případy je proto nutné vytvořit kolem stovek obrázků pro veškeré možné hodnoty, proto není známo, že by se tato technika někde využívala.

Třetí možnou a nejčastěji používanou technikou je generace bitmapového obrázku, který obsahuje „pohyblivou“ část indikátoru. Oblast této bitmapy je z jedné strany omezována v poměru k maximální možné délce v daném směru, takže dochází k ořezu tohoto obrázku. Pod obrázkem se pak nachází nepohyblivá část obrázku — pozadí.

D. Grafy

Zvukové aplikace vyžadují ke vhodné komunikaci směrem k uživateli také zobrazení závislostí dvou proměnných. Tuto závislost je možné si představit například jako vztah hlasitosti k času, nebo přenosové funkce ke kmitočtu.

Vykreslení grafu je prováděno nejčastěji základními operacemi pro vykreslení čáry. Vykreslení je provedeno pro každý ze vzorků nebo segmentů potřebného zobrazení často instrukcí centrální procesní jednotky. V závislosti na rozlišení se může jednat o náročnou operaci, která je zjednodušena zmenšením frekvence vykreslování.

Většina grafických knihoven pro správu editoru zvukových parametrů obsahuje aspoň základní nástroje pro vykreslení čar do uživatelského prostředí, pokročilejší zobrazení ale nutí vývojáře k vytvoření vlastního rozšíření této grafické knihovny, což není vždy (podle struktury knihovny) triviálním úkolem.

3.2.2 Aktivní uživatelské prvky

Aktivní uživatelské prvky jsou ty, které uživatel může přímo ovlivnit svým uživatelským vstupem. Jedná se převážně o *tlačítka*, *posuvníky* a *otočné knoflíky*. Aktivní prvky musí reagovat na pozici kurzoru myši a uživateli nejlépe napovědět, že se jeho kurzor nachází nad správným prvkem (*hover*), ale také musí reagovat na stav myši - konkrétně kde a které tlačítko myši bylo zmáčknuto (*click*) a zdali není tlačítko drženo nějakou dobu (*drag*).

Požadavkem pro aktivní uživatelské prvky je okamžitá reakce na uživatelský vstup. Pokud by se vizuální stav uživatelského prvku změnil až po jeho editaci a ne zároveň s ní, uživatel může být uživatel přesvědčen, že nad zvukovým efektem nemá dostatečnou kontrolu.

A. Tlačítka

Nejsnadnějším prvkem pro implementaci jsou **tlačítka**. Zde platí stejný princip jako u diody - tedy že nejčastěji se jedná o dva obrázky, které mezi sebou střídají stav viditelnosti. Kromě těchto dvou obrázků se do zdrojových souborů zahrnuje i stav tlačítka při najetí kurzoru, a také případ, kdy je tlačítko zakázané, tedy jeho hodnotu nelze změnit.

Při stisku tlačítka je ještě vhodné indikovat uživateli, že tlačítko bylo zmáčknuto. V jednoduchých implementacích toho může být dosaženo krátkou změnou barvy tlačítka, ve složitějších se jedná o krátkou animaci změny polohy tlačítka.

B. Posuvníky

Dalším typem aktivních uživatelských prvků je **posuvník**, v anglické literatuře nazývaný také jako *slider*. Posuvníkem může být, stejně jako u indikátoru úrovně vybuzení, objekt obdélníku, který podle polohy stisknutého tlačítka myši mění svojí velikost v definované oblasti. Definovaná oblast omezuje polohu posuvníku pro jeho minimální a maximální hodnotu, a proto je nutné tuto oblast také vykreslovat. Nejlépe pomocí dalšího obdelníku v pozadí s jinou, méně sytou barvou.

Některé posuvníky nevyužívají pro znázornění aktivní hodnoty změnu velikosti aktivní oblasti, ale změnu pozice. Zde opět figuruje důležitá část pozadí, po kterém se pohybuje aktivní část uživatelského prvku - posuvník. Každou z těchto částí lze v praxi zrealizovat použitím pouze jednoho bitmapového obrázku.

C. Otočné knoflíky

Nejsložitějším typem pro implementaci je **otočný knoflík**, jenž má kopírovat funkci a podobu otočného potenciometru. Byť by stejnou funkci knoflíku zastal adekvátně i posuvník, otočný knoflík má výhodu v malé velikosti, a je tedy možné jich do editačního okna vložit více.

Technik pro zobrazení je několik, ale vždy jsou závislé na stylu zobrazení editačního okna. Otočný knoflík lze rozdělit na dvě oblasti: knoflík a pozadí knoflíku. Pozadí knoflíku je převážně určeno k vykreslení škály hodnot parametru, jichž parametr otočného knoflíku může nabývat. Pohyblivou část — knoflík — lze implementovat několika metodami:

C-A Metoda otočné bitmapy

Jednou z technik je vykreslení obrázku otočného knoflíku a jeho následné otáčení. Byť tato metoda může využít pouze jednu použitou texturu, není možné tento způsob využít v případech, kdy je požadován realistický vzhled vykreslení, protože při rotaci dojde i k otočení zdroje světla.

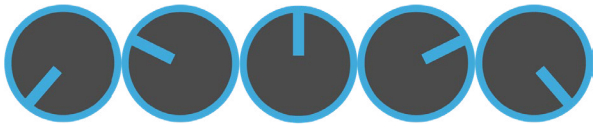
C-B Metoda otočného ukazatele

Z tohoto důvodu je možné použít **metodu otočného ukazatele**. Zde jsou využity minimálně dvě bitmapy, jedna pro kulatou část, která reaguje na osvětlení scény, není však pohyblivá. Aktivní částí je textura ukazatele, kterou je možno otáčet a která nereaguje na osvětlovací podmínky. Při rotaci dochází k dojmu, že se točí obě tyto části.

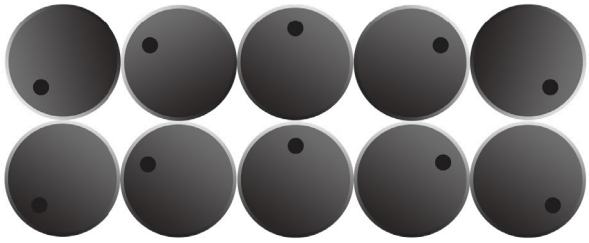
C-C Metoda animovaného knoflíku

Zmíněný způsob ale zásadně omezuje možnosti vzhledu otočného knoflíku, proto je nejčastěji realizována nejméně paměťově šetrná varianta „*animovaného knobu*“. V tomto případě je pro každou hodnotu otočného knoflíku přiřazen jeden bitmapový obrázek. Díky tomu jsou zachovány veškeré osvětlovací podmínky, ale také zvýšeny požadavky na grafického designera, který musí vytvořit až stovky obrázků pro všechny polohy.

Z obrázku výše je jasné, že nelze zvolit jednu z těchto bitmap a otáčet je dle libosti, ze stejného důvodu jako v předchozím názorném případě. Taktéž nelze použít metodu otočného ukazatele, neb knoflík obsahuje hrbolky na stranách a samotný ukazatel reaguje na osvětlení. Zde na obrázku je 25 poloh otočného knoflíku, to by však pro zobrazování plynulého pohybu nestačilo — obrázků by bylo potřeba několikanásobně více.



Obrázek 1: Metoda otočné bitmapy.



Obrázek 2: Porovnání metody otočné bitmapy a otočného ukazatele.
Metoda otočné mapy (nahoře) při otáčení pohybuje i se stínováním,
což produkuje nepřijatelné výsledky.
Metoda otočného ukazatele (dole) tento problém řeší otáčením
pouze ukazatelem.

Obrázek 3: Metoda animovaného knoflíku.
Na obrázku je znázorněn pouze zlomek potřebných poloh knoflíku.
Použitím takto malého množství poloh (25) by produkovalo skokové
změny zobrazení otočného knoflíku.

3.3 Styly vykreslování uživatelského rozhraní

Na použitých demonstračních obrázcích otočného knoflíku z minulé kapitoly je jasně vidět, že v závislosti zvolení grafického stylu zvukového zásuvného modulu se i technika zobrazení jednotlivých uživatelských prvků mění a je nutné v prvotních fázích projektu nutné zohlednit časové dispozice pro vytváření vzhledu zvukového zásuvného modulu.

Vizuální styly lze klasifikovat do několika skupin. Každá z těchto skupin se liší jak výsledným dojmem uživatele, tak postupem spolupráce grafika a programátora, tak i požadovanými technologiemi pro vykreslení.

3.3.1 Jednoduchý 2D styl

Jednoduchý 2D styl se vyznačuje absencí světelných zdrojů. Uživatelské prvky tak lze řešit jednoduchými objekty (obdelník nebo kruh) s plnou vyplněnou barvou. V častých případech se opět jedná o použití množství bitmapových obrázků, které i přesto, že je nutné jich vygenerovat velké množství, nezaberou tolik času jako další styly níže.

Tento styl se používá převážně u novějších zvukových efektů, které nemají žádnou původní hmatatelnou podobu, a umožňuje poměrně rychlé změny v rozložení uživatelských prvků z důvodu nezávislosti prvků na sobě a také na pozadí. Skupina těchto zvukových efektů se také vyznačuje častým použitím 2D grafů, které s vizuálním stylem ladí, a vytváří komplexní obrazový dojem.

Moderním a elegantním řešením pro vykreslení tohoto stylu by bylo použití vektorové grafiky, která dovoluje neomezené přiblížení bez ztráty kvality, a navíc s benefitem ušetřené paměti zvukového zásuvného modulu. Z dnešních technologických řešení však vektorový přístup nabízí pouze JUCE (viz kapitola 3.1.2 na stránce 14).

3.3.2 Stínovaný 2D styl

Stínovaný 2D styl se snaží částečně imitovat světelné prvky a zdroje, svým vzhledem ale přiznává svůj grafický původ - 2D grafický program bitmapové (Adobe Photoshop, GIMP) nebo vektorové povahy (Adobe Illustrator, InkSpace). Styl dovoluje nastínit dojem prostorovosti použitím množství lineárních barevných přechodů s pevně daným úhlem natočení. Pokud jsou tyto přechody vhodně umístěny na pomyslné hrany a plochy, může být dosaženo až téměř nerozeznatelného vzhledu od skutečnosti.

Tento styl se vyznačuje častým použitím metody otočného ukazatele a vývojářům zvukového pluginu nabízí poměrně rychlý pracovní proces. Uživatelské prvky nejsou, stejně jako v předchozím případě, vázány samy na sebe ani na pozadí, a lze je libovolně přemisťovat. Právě kvůli značným výhodám byl tento styl v minulosti často používán. Tomu tak není v současnosti, a to možná právě z důvodu „přeúžívání“, či postoupení moderních trendů „zpátky“ k realistickému zobrazení.

3.3.3 Realistické zobrazení

Na realistické zobrazení může uživatel narazit v katalozích zvukových efektů nejčastěji. Tento vizuální styl sází na dobrou pověst analogových předloh zvukových efektů. Charakter zvuku těchto zařízení vykazoval ne vždy deterministické chování a přidával zvuku další harmonické a barevné složky. Této zdánlivé nevýhody zvukoví inženýři využívali k uměleckým záměrům přidat do nahrávky zajímavé zvukové prvky.

V digitální doméně se těchto charakteristických prvků dosahuje nejčastěji simulacemi původních analogových zařízení, a tyto simulace mají značnou oblibu u uživatelů, neboť dovolují dosáhnout stejného zvuku za méně peněz bez nutnosti vlastnit rozměrnou krabici zvukového procesoru, která je v horších případech na trhu téměř nedostupná.

Tyto zvukové simulace vyžadují ke svému zvukově barevnému charakteru i adekvátní vizuální reprezentaci. Pokud se jedná o kompletní rekonstrukci efektu do digitální podoby, je možné využít fotografií zařízení, nutné je však vytvořit veliké množství záběrů, protože jedinou možnou metodou pro zobrazení všech možných pozic otočných potenciometrů je *metoda animovaného knoflíku*. Problém nastává v případě, kdy je nutné k aktuální sadě uživatelských prvků přidat další či pozměnit rozložení.

Imitace původního či odvozeného analogového vzhledu je obecně nazývána jako **skeumorfismus** [22]. Tato grafická interpretace je v některých případech kritizována za zbytečně ilustrativní a dekorativní. Naopak zastánci argumentují důležitostí se zorientovat v digitálním prostředí v závislosti na zažitých zvyklostech z reálného světa. [23]

A. Současný přístup

Častěji je možno se k realistickému zobrazení přiblížit pomocí 3D grafických programů. Zde grafický designer musí modelačními nástroji vytvořit povrch uživatelských prvků a pozadí. Poté je možno tyto prvky osvítit virtuálními světly a simulovat materiálové chování v případě osvitu směrem k virtuální kameře.

Takto připravené scéně schází už pouze animace otočných potenciometrů pro všechny možné případy natočení. Pohled kamery je nutno zvolit tak, aby se jednotlivé uživatelské prvky nepřekrývaly, protože pak by nebylo možné obrázek rozstříhat na více segmentů.

Je však nutno zmínit důležitý poznatek, že v případě použití virtuální kamery s perspektivními vlastnostmi není možné už v dalších fázích vývoje upravit rozložení prvků. Každý uživatelský prvek má svoji pevně danou pozici v prostoru editoru zvukových parametrů.

Pro lepší ilustraci je vhodné vytvořit modelovou situaci zvukového zásuvného modulu s 11 otočnými knoflíky a třemi tlačítky. Za předpokladu, že otočný knoflík vyžaduje pro plynulý pohyb minimálně 256 obrázků s jednotlivými polohami, vychází počet bitmapového materiálu pouze na otočné prvky na 2 816 obrázků. K tomu je potřeba přičíst 4 obrázky na každé tlačítko pro stavy vypnuto, zapnuto, zakázáno a přejeto myší, a také obrázek pozadí. Výsledný počet se tedy vyšplhá na hodnotu 2 829. Zde není brána v potaz velikost zdrojových materiálů a může být zřejmé, že bez obrazové komprese by mohlo dojít u velikosti zvukových zásuvných modulů k obrovským hodnotám.

Z tohoto důvodu se využívá nejčastěji přední ortogonální pohled. Tento pohled nejen že dovoluje přemisťovat prvky celým rozložením, ale také umožňuje definovat jedinou podobu otočného knoflíku pro veškeré další použití, čímž dojde k radikální úspoře místa.

Předchozí modelový případ se tedy zredukuje na 256 obrázků pro všechny otočné knoflíky, 4 obrázky pro všechny stavy tlačítka a jeden obrázek pro pozadí s celkovou hodnotou 261 obrázků pro celý zásuvný modul za předpokladu, že je pro všechny případy použit stejný typ virtuálního otočného potenciometru.

Ortogonální zobrazení však má tu nevýhodu, že jeho použitím grafické prostředí ztrácí veškerou obrazovou dynamiku. Tomu se lze částečně vyhnout aplikováním perspektivního pohledu pouze pro pozadí, uživatelské prvky však tímto spravit nelze.

B. Vhodné přístupy do budoucnosti

Techniky shrnuté v předchozí kapitole mají dva společné rysy: obrovskou neefektivitu a vysoký podíl kompromisů ve vizuální stránce. Dnešní doba však rozvojem počítačově herního průmyslu dovoluje tyto způsoby práce nejen obohatit, ale také nahradit. Počítačové hry jsou totiž společně s filmovým průmyslem hlavním hybatelem počítačové grafiky, a to konkrétně vykreslování v reálném čase, což je přínosem pro všechna odvětí, která se počítačové grafiky aspoň částečně dotýkají.

Ruku v ruce s rozvojem počítačové grafiky se rozvíjí hardwarová základna pro grafické výpočty - grafické akcelerátory. **Grafické procesní jednotky** (GPU) mají na starosti odtížit výkon pro vykreslování grafických uživatelských rozhraní a grafických výpočtů z centrální procesní jednotky, na které mimochodem běží i zpracování zvuku.

Proto zní logicky využívat této procesní jednotky k vykreslování grafického uživatelského rozhraní, opak je však dominantní tendencí ve světě zvukového zpracování. Cílem této diplomové práce je přiblížit dostatečně pokročilé techniky užité při vykreslování počítačových her k účelům zobrazování uživatelského rozhraní zvukových zásuvných modulů.

Jedním ze způsobů jak zlepšit zmíněný proces práce je pracovat rovnou s 3D modely, které reprezentují tvary uživatelských prvků, až v editačním okně zásuvného modulu. Zde lze vypočítávat vliv světla a odrazů na povrch tělesa a provádět základní pohybové operace dle libosti - rotace, translace a změna měřítka. Také změna barvy či podsvícení by zaznamenala možnost zredukovat generaci několika obrázků na změnu jednoho parametru určující vykreslovací vlastnosti objektů.

Tímto způsobem je možno původních 261 obrázků modelové situace z minulé podkapitoly zredukovat na 3 modely: model otočného knoflíku, model tlačítka a model pozadí. Je zde nutno definovat jejich materiálové vrstvy, což jsou opět bitmapy, jejich množství ani velikost ale zdaleka nedosahují hranice stanovené zmíněným příkladem.

Tyto 3D modely je nutné vykreslit v dostatečné kvalitě tak, aby posun k tomuto způsobu práce dával smysl v porovnání s předvypočítanými stavy realistického stylu. To bude zmíněno v kapitole 5.4.1 na stránce 38.

Tato diplomová práce také navrhuje způsob, který je jakýmsi mezistupněm mezi původním 2D vykreslováním několika obrázků a plným využitím 3D vykreslovacích možností. Jedná se o využití triplanárních normálových map a metody zachyceného materiálu. O způsobu, jak funguje tento způsob a jak ji lze realizovat, je možné se dočíst v kapitole 6.1 na stránce 41.

3.3.4 Shrnutí dostupných technologií pro grafickou knihovnu

Z popsaných knihoven se zdá jako nejvhodnější využít základní technologie VST3 a její podknihovnu VSTGUI. Je to z toho důvodu, že je aktivně vyvíjena a obsahuje dostatečné množství dokumentace, takže je méně pravděpodobné, že knihovna závislá na této technologii se stane později zastaralou. Také je tato technologie vhodná k spolupráci s ostatními knihovnami třetích stran.

Framework JUCE je taktéž aktivně spravovaným projektem, který udržuje programátorské a technologické trendy, ale jelikož se jedná o abstraktně pojatou knihovnu nastavenou na platformně závislých základech, je očekáváno, že práce s nižší úrovní by byla práce tímto faktem zpomalena.

Vešekré knihovny ve svých útrobách obsahují pomocné třídy pro správu OpenGL kontextu a i základní práci s tímto kontextem, žádná z nich však nenabízí dostatečně pokročilé

funkce (například podpora vyšších verzí OpenGL), které jsou podstatou této diplomové práce. Všechny knihovny ve své podstatě nabízejí stejný výchozí bod pro vytváření grafické knihovny pro zvukové zásuvné moduly.

4 Techniky moderní počítačové grafiky

V předchozí kapitole byly popsány techniky počítačové grafiky, které se používají k vykreslování editoru zvukových parametrů. Tato kapitola si klade za cíl popsat techniky, které se používají v herních a grafických aplikacích (průmyslech), rychle se rozvíjejících odvětvích, s myšlenkou, že by některé poznatky mohly být do budoucna aplikovány i do prostředí zvukových zásuvných modulů.

Naprostá většina aktuálních technik využívá grafické výpočetní jednotky, které jsou uzpůsobeny k vykreslování grafiky a nacházejí se dnes už téměř ve všech zařízeních. Slouží převážně k rychlejšímu vykreslování uživatelského prostředí operačního systému, vykreslování vektorových objektů běžného života, jako jsou například texty a mapy, ale jejich vývoj a optimalizace je převážně řízena herním průmyslem, který využívá převážně 3D grafické zobrazení a realistické zobrazení virtuálního světa.

Největší optimalizace je dosažena pomocí několikastupňového procesu, jemuž se říká *proudové zpracování geometrických dat* (v angličtině *rendering pipeline)*. Jedná se o návaznost několika programů, jež běží v grafické výpočetní jednotce, jimž se říká *shadery*.

Komplexita těchto shaderů by měla být co nejmenší, avšak počítá se s tím, že těchto shaderů může běžet zároveň velké množství. Každý takovýto program má na starosti výpočet jedné části vykreslovacího řetězce, například pozici bodu vzhledem ke kameře nebo barvu jednotlivého pixelu. To dovoluje masivní paralelizaci výpočtu geometrických transformací a barevných operací, které jsou potřeba k vykreslování 3D scény.

Komunikaci s hardwarem grafického akcelerátoru a správu správného chodu proudového zpracování geometrických dat zastřešují grafické knihovny a rozhraní. Příkladem může být rozhraní **Direct3D** a nebo aplikační prostředí **OpenGL** .

Podle funkce v proudovém zpracování se programy dělí v opengl na vertex shadery, geometry shadery, fragment shadery a také compute shadery. Základní vykreslovací řetězec obsahuje vždy vertex shader a fragment shader. [4]

4.1 Geometrické transformace

Grafické aplikace pro reprezentaci geometrických dat využívají homogenních souřadnic – každý prostorový bod je vyjádřen čtyřprvkovým vektorem značící jeho souřadnice v orgonální bázi (x,y,z) a váhovací koeficient w.

Přidání čtvrté souřadnice dovoluje využít maticového počtu s maticemi 4x4 k provedení většiny geometrických operací: posunu, rotace, změnu měřítka, orografické a perspektivní projekce. Každou z operací lze zapsat do maticového tvaru a jejich efekty spojovat do jedné

transformační matice jejich společným násobením, kde pořadí násobení ovlivňuje pořadí operací.

Váhovací koeficient je roven nule, pokud souřadnice zastupují směrový vektor. Poté na něj nejsou aplikovány operace posunu. [14] Většinou se při převodu z kartézských souřadnic volí hodnota $w = 1$, pro plné využití vlastností homogenních souřadnic. Při perspektivní projekci je právě tato hodnota pozměněna a při zpětném převodu do kartézských souřadnic je nutné vektor upravit tak, aby byla splněna podmínka $w = 1$.

Grafické akcelerátory jsou těmto matematickým operacím přizpůsobeny a dovolují využití maticového počtu pro každou instanci použití shaderu (například pro každý ze zpracovávaných bodů). Transformace probíhá ještě rychleji, pokud je transformační matice předvypočítaná pro každou sadů bodů (objekt) dopředu, například na začátku vykreslovací smyčky.

4.2 Základní principy stínování

Snahou počítačové grafiky několika posledních desítek let je se vykresleným obrazem co nejvíce přiblížit skutečnosti. Skutečný svět však dovoluje spatřovat obraz díky miliónům fotonů, které se šíří ze zdroje světla a mnohonásobně se odrážejí od překážek až do oka pozorovatele. V digitálním světě se však ani zdaleka nelze přiblížit počtu fotonů a odrazů, takže je nutné přistupovat k vykreslování na obrazovku počítače s jistou mírou aproximace a zjednodušení.

Současná počítačová grafika je tedy takovým souborem triků využívajících nedokonalost lidského oka a mozku. Jedním z hlavních principů počítačové grafiky je zjednodušení překážek reálného světa pouze na jejich povrch zanedbávajíc objem. Tento povrch je navíc ještě několikanásobně zjednodušen segmentací povrchu na trojúhelníky či čtyřúhelníky, obecně **polygony**.

Plocha každého polygonu je definována sadou bodů a hran, které jej omezují. Některé z těchto bodů a hran jsou sdíleny mezi dalšími polygony a dohromady tvoří takzvanou polygonovou síť — v angličtině **mesh**. Čím je tato síť hustší, tím více dovoluje uchovat povrchový detail objektu, avšak na úkor výkonu, protože každý z bodů polygonové sítě a každý polygon musí projít transformací vertex shaderu z 3D koordinace na 2D polohu na obrazovce.

Hustota polygonové sítě může být zredukována několika možnými způsoby. Prvním z nich je použití normálových a deformačních map pro zobrazení povrchových nerovností v místech, kde by množství těchto nerovností znamenalo obrovské zhuštění polygonové sítě.

Pro zjednodušení cesty fotonu od zdroje světla do oka pozorovatele se nejčastěji zanedbává fakt, že světelný paprsek ve většině případů projde cestou odrazů přes mnoho překážek. V nezákladnější formě 3D vykreslování se bere v potaz pouze následující: zdroj světla, pozice virtuální kamery (pozorovatele) a natočení polygonů překážky. Z těchto informací lze pomocí *Snellova zákona* a *Blinnova* nebo *Phongova osvětlovacího modelu* určit míru osvícení ve dvou složkách — **difuzní** a **zrcadlové**. [17]

Difuzní odraz nebo také *odraz Lambertovského povrchu* počítá s jistou mírou podpovrchového i povrchového rozptylu světla, takže není důležité, z jakého úhlu se na povrch po-

zorovatel podívá. Pomyslný světelný paprsek svoji příchozí energii rozloží rovnoměrně všemi směry a jedná se nejčastěji o složku, která určuje barvu povrchu.

Druhým typem základního odrazu je **zrcadlový odraz**, kterému se také jinak říká spekulární odraz, který se snaží napodobit vlastnosti ideálně vyleštěného povrchu — například kovů, skla nebo plastu.

Kombinací těchto dvou složek se donedávna vypočítával vzhled většiny 3D scén v počítačových hrách a ve filmech. A i přesto, že základ těchto složek vychází z fyzikálních principů, není výstupem dostatečně přesvědčivý obraz. Hlavní nevýhodou tohoto přístupu není dodržování zákona zachování energie, jelikož některé části obrazu se mohou zdát až moc tmavé a některé zase moc světlé, a to i po aplikaci korekční gammy.

Tento pohled na 3D model je však v posledních pár letech nahrazován dalšími přístupy, které se snaží jak zohledňovat principy zachování energie, ale také další fyzikální jevy, mezi které patří například fresnelův jev.

Hlavním přístupem tohoto nového typu je způsob vykreslování s názvem **Physically Based Rendering,** jež byl veřejnosti představen na konferenci Siggraph 2012 Brentem Burley ze společnosti Walt Disney. Jedná se o rozšíření předchozích vykreslovacích technik a popsání jejich funkčnosti na celovečerním animovaném filmu Wreck-It Ralph. [10]

Hlavní zásluhou na tomto rozšíření má firma Epic Games, která na základě této přednášky uplatnila toto poznání ve směru real-time vykreslování, a určila tak nový standard pro většinu grafických programů.

4.3 Deferred Shading

Deferred Rendering nebo také *odložené stínování* je způsob, jak vykreslovat scénu ve dvou průbězích. V prvním průběhu se vypočítá nejdříve umístění geometrie, její natočení vzhledem ke kameře a také vliv perspektivy. Tyto informace o geometrii jsou uloženy do takzvaného geometrického zásobníku (**G-Bufferu**). V druhém průběhu probíhá osvětlování scény na základě právě tohoto G-Bufferu společně s dalšími postprocesovými efekty, jako například odrazy nebo hloubkové rozostření scény.

To dovoluje naprosté oddělení světelné složky od geometrické složky, tím se výpočetní náročnost pro komplexní polygonové sítě se tím radikálně snižuje. Tento způsob je převzat z kompozitního způsobu práce z filmového průmyslu, kde je výsledný obraz postupně seskládán a osvětlení scény je „odloženo“ do nejposlednějšího momentu.

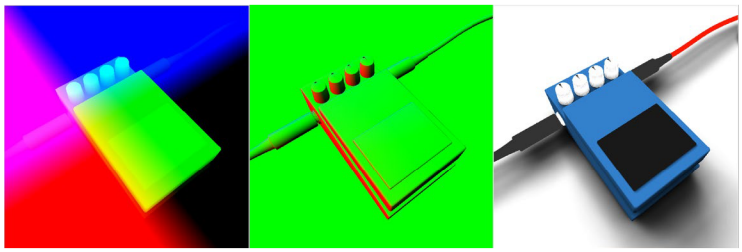
Jako první komerční demonstrací tohoto způsobu práce je konzolová hra Shrek vydaná v roce 2004, a od té doby se pomalu stala technika běžnou součástí real-time vykreslování, kdy většina dnešních počítačových i konzolových her využívá právě možnosti velkého počtu světelných zdrojů, obrazově orientovaných odrazů a dalších obrazových efektů.

G-Buffer může obsahovat téměř jakékoliv informace o geometrii objektu, nicméně nejčastější je podoba podobná návrhu Saita and Takahashiho z roku 1990 [9]. Tedy jsou zde

uloženy pozice jednotlivých pixelů v globálních souřadnicích (X, Y, Z), natočení normál v globálních souřadnicích (X, Y, Z), index objektu a někdy i vzdálenost od kamery. Veškeré hodnoty jsou uloženy pro každý pixel obrazu. Zde princip zaznamenané geometrické informace byl využit k zobrazování výškové mapy pomocí vrstevnic, tedy stylizovaného, nikoliv realistického vykreslení, avšak pro rozšíření stačí přidat předvypočítané rozsahy jednotlivých světel, která lze nakumulovat do jedné barevné textury.

G-Buffer je v terminologii opengl takzvaným framebuffer objektem, do kterého je obraz ukládán místo samotného okna aplikace, a protože se jedná o více složek, lze jej popsat také označením *multiple render targets*; v praxi se nejčastěji jedná o tříkanálové (RGB) textury, které jsou ve vykreslovacím řetězci posouvány dále do druhého stupně.

Ve druhém stupni je s daty G-bufferu pracováno na sestavení výsledného obrazu. Data o natočení a pozici mohou být použita k výpočtu osvětlení jednotlivými světly, které jsou uloženy jako další podzásobník g-bufferu. Oblasti, které byly definovány v předchozím stupni, jsou teď vyhodnoceny, jestli jejich natočení opovídá Snellovu zákonu, a vypočítá se jejich barva, jestli je použit klasický osvětlovací model nebo fyzikálně založené vykreslování.



Obrázek 4: GBuffer - Ukázka prvního průchodu odsunutým stínování.
Zleva: globální souřadnice, natočení normál, albedo mapa (barva).

4.4 Technologie Material Capture

Pro účely stilizovaného a rychlého zobrazení byla vyvinuta technika osvícené koule (*Lit Sphere*) [20], jejím úkolem bylo replikovat vykreslovací styly klasických malířů a kreslířů do digitálního prostředí. Její princip spočívá v přichystání osvětlovacího modelu na povrch koule, která slouží jako vhledávací tabulkou pro vykreslení povrchu 3D objetku.

Tento způsob byl od dob představení techniky volně přejmenován na **material capture** a byl adaptován k využití převážně ve modelačních programech. Jedná se hlavně o grafické programy sculptovacího charakteru, kde designer pracuje s modelem obdobne jako sochař. Mezi příklady sculptovacích nástrojů patří například ZBrush od firmy Pixologic nebo program Mudbox od firmy Autodesk.

Osvětlovací technika provedena tímto způsobem je závislá pouze na normálách vztažených ke kameře, což má za následek posun osvětlení s posunem kamery.

Jak již bylo zmíněno, veškeré podmínky osvětlení jsou zapečeny do textury (*material capture texture* - zkráceně *matcap texture*), a není s nimi možno dále manipulovat. Tento problém se snaží vyřešit [21] dekompozicí matcap textury na několik upravitelných vrstev.

Obrázek 5: Technologie Material Capture.
Konvice nahoře je vstupním obrazem, ze kterého jsou vypočítány veškeré ostatní s pomocí osvětlených koulí — vyhledávacích tabulek.

4.5 Interakční systémy

Většina aplikací využívající grafické vykreslovací systémy vyžaduje k chodu programu uživatelské instrukce. Instrukce mohou být předávány stiskem kláves, pohybem a kliknutím myši, tlačítky herního ovladače, vstupem z dotykové obrazovky nebo speciálního vstupního zařízení. Tento vstup musí být zaznamenáván a adekvátně zpracováván. Pro účely jednoduchosti budou další kapitoly omezeny pouze na interakci realizovanou myší počítače.

Informaci o pozici kurzoru může předat aplikaci operační systém knihovny spravující okna či uživatelské rozhraní. Pro běh programu s 2D uživatelským rozhraním to představuje pohodlný postup při zjišťování, jestli se kurzor nachází v určité obdélníkové oblasti (pro případ obdélníkových uživatelských prvků) nebo v určité vzdálenosti od středu uživatelského prvku (pro kruhové prvky).

Po splnění podmínek vhodné pozice a indikace stisku správného tlačítka myši se vyšle asynchronní zpráva o této akci. V takovém případě hovoříme o *event-driven systému,* který je preferován při programování uživatelského rozhraní.

Preferován je z důvodu, že pro chod programu je zbytečné kontrolovat pravidelně v časových intervalech, jaké změny byly na vstupu provedeny. Ale výhodnější je předat systému funkce zpětného volání (v angličtně *callbacks*), které budou zavolány v určité situaci.

Tento systém je uplatnitelný i v případě, že prostor okna aplikace nezobrazuje 2D prostředí, ale 3D scénu, ve které se mohou překrývat objekty, mohou být zkresleny perspektivní

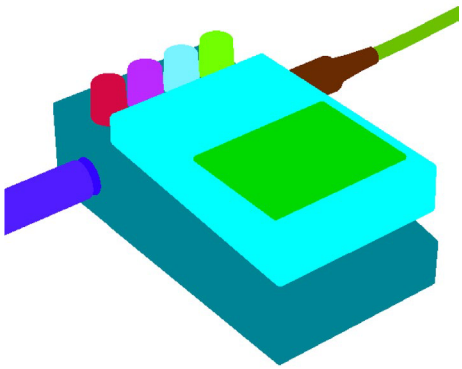
projekcí nebo na ně může být pohlíženo z jiného úhlu. V takovém případě je nutné vyřešit správný výběr objektů a dostatečně intuitivní reakci programu v případě uživatelské akce (klik, táhnutí, dvojklik atd.).

4.5.1 Výběr objektů ve scéně

V rozhraní OpenGL jsou pro výběr objektů z pohledu kamery dvě varianty. První z pozice kamery vyšle orientovaný paprsek ve směru, kde se nachází kurzor. Tento paprsek je pak porovnáván s každým objektem ve scéně, zdali nedojde ke kolizi. V případě kolize je možno získat nejen odkaz na objekt, ale také přesné místo kolize.

Operace má jednoduché řešení, pokud mají objekty analytický popis (např. koule, kvádr, atd.). Veškeré ostatní objekty kolizi zaznamenávají pouze na úrovni polygonů, což může být výpočetně náročné. V praxi se tedy využívá sekvenčního testu pro objekty, který začne analytickým objektem, který objekt obaluje. Tím je buď koule nebo kvádr. Test není příliš přesný pro malé záhyby, ale určí, jestli paprsek vůbec mířil příslušným směrem. Jakmile je kolizní test úspěšný, je objekt podroben přesnějším testům.

Druhá varianta spočívá ve využití principu G-Bufferu. Do tohoto zásobníku je vložena ještě jedna obrazová textura, která zaznamenává indexy jednotlivých objektů z aktuálního pohledu. V rámci této práce tuto texturu budeme označovat jako **indexový objektový zásobník.** Při pohybu myši je pak z pozice kurzoru na 2D textuře vyčtena barva (viz Obrázek 6), a tedy určen objekt, který se nachází pod kurzorem myši. Tato metoda se vyznačuje větší paměťovou náročností, ale dovoluje být aplikována na objekt libovolné komplexity.



Obrázek 6: Možné zobrazení indexového objektového zásobníku.

4.5.2 Správa translace a rotace

Objekty 3D scény se v interaktivních grafických aplikacích mohou pohybovat, rotovat, a v některých případech i zvětšovat a zmenšovat. Toho je docíleno při geometrické transformaci, která probíhá pro každý z bodů polygonové sítě. Každý z bodů je z kartézských souřadnic převeden do homogenního tvaru, maticovým násobením transformován do cílené pozice a poté převeden opět na kartézské souřadnice.

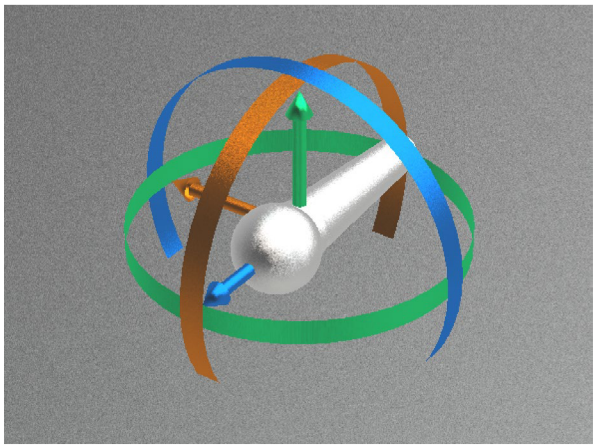
Tomuto tématu se věnuje kapitola 4.1 na stránce 23, nicméně zmíněná pravidla platí pouze pro stanovení statické rotace objektu a přestávají platit v případě, kdy je objekt animován — nelze zde použít lineární interpolace mezi eulerovými úhly. Pokud by lineární interpolace byla použita na rotační úhly, docházelo by k nepředpokládanému chování. Tento pohyb lze popsat jako nestabilní „viklavý“ pohyb. Také se může stát, že rotační osy přestanou být na sobě nezávislé a při rotaci bude ztracena jedna osa volnosti (tzv. *gimbal lock)*. [19]

Z tohoto důvodu je rozumnější zvolit sice komplikovanější zato determinističtější matematický aparát zvaný **kvaternion** (v angličtině *quaternion*). Ten dovoluje reprezentovat orientaci v podobě čtyř reálných čísel jako projekci čtyřrozměrné jednotkové koule (hyperkoule) do trojrozměrného prostoru. [18] Pokud je zvolen tento způsob výpočtu, je možné lineárně interpolovat úhly (tentokrát už v podobě kvaternionů), natáčet objekty tímto směrem bez rizika ztráty osy volnosti a v případě potřeby přepočítat zpět do eulerových úhlů.

Nutnost přepočítávat a zobrazovat orientaci v eulerových úhlech spočívá v její pochopitelnosti a možnosti představy tří na sobě nezávislých os a jejich úhlů. Pro nečíselnou editaci úhlů bývají vyobrazena k objektu také trojrozměrná manipulační táhla (v angličtině *gizmos*). V případě rotace se nejčastěji jedná o na sobě kolmé barevné prstence, které táhnutím dovolují upravit rotaci objektu změnou eulerových úhlů (přepočítaných na kvaternion a zpátky).

Tohoto zobrazení je nejčastěji využíváno v grafických editorech zaměřených na 3D grafiku. Obecně platí nepsané pravidlo, že jednotlivé osy jsou barevně — odlišeny červeně pro osu x, zeleně pro osu y a modře pro osu z. Naopak panuje neshoda napříč grafickými programy v rozhodnutí, kam mají jednotlivé osy mířit.

Ať jsou osy namířeny kamkoliv, je možné objekt po těchto osách posouvat, a to opět pomocí manipulačních táhel, nejčastěji v podobě šipek. Pokud se objekt posouvá po osách, které byly už jednou natočeny, jedná se o *translaci lokální* — objekt se posouvá vzhledem ke svému natočení dopředu, nahoru a doleva. Pokud není při posunu na rotaci brán zřetel, jedná se o *translaci globální*.



Obrázek 7: Translační a rotační manipulační táhla pro 3D scénu.

4.6 Vykreslení textu

Vykreslení textu je s OpenGL prováděno vytvořením bitmapové textury při incializaci, která obsahuje veškeré znaky v různých velikostech. Z této textury (někdy nazývaná *font atlas*) jsou mapována jednotlivá písmena a umisťována na obdélníkové polygonové objekty scény. Podporu tohoto způsobu vykreslování nabízí knihovna *freetype [24]*. Nevýhodou tohoto způsobu je paměťové omezení, které nedovoluje hladké vykreslení znaků s větší velikostí.

Druhý, modernější, způsob pracuje s vektorovou povahou fontových souborů. Vykreslováním vyplněných kvadratických a Beziérových křivek lze dosáhnout libovolně škálovatelnou a transformovatelnou podobu znaků s minimálním dopadem na kvalitu a paměťovou náročnost. Nevýhodou však je zvýšení výpočetní náročnosti na grafický akcelerátor. [11], [12]

5 Knihovna RealBox

5.1 Filozofie knihovny RealBox

Knihovna **RealBox** byla v rámci této diplomové práce vytvořena jako reakce na techniky popsané v kapitole 3.2. Vybrány byly hlavně ty postupy, které jsou nejméně efektivní a ztěžují celkový postup práce na tvorbě uživatelského rozhraní zvukového zásuvného modulu. K těmto přístupům byla knihovnou RealBox navržena lepší alternativa, ale veškeré starší, ale dostačující metody byly ponechány na knihovně VSTGUI, kterou RealBox rozšiřuje.

Rozšíření je provedeno pouze v jednom místě, děděním ze třídy `COpenGLView` do třídy `RealBoxView`. Tím je vytvořena možnost do budoucna osamostatnit knihovnu RealBox jako samostatný nezávislý celek. Veškeré další třídy už nevyužívají žádných funkcí ani tříd knihovny VSTGUI. Více ke struktuře knihovny bude popsáno v kapitole 5.3 na stránce 31.

RealBox je statickou knihovnou napsanou v jazyce C++, což dovoluje její funkce při tvorbě zvukového zásuvného modulu zkompilovat do jediného souboru bez nutnosti dalších návazností. Výsledkem zvukového modulu pak bude jediný soubor dynamické knihovny s příponou *.vst*.

Knihovna RealBox se snaží vytvářet a využívat nízkoúrovňové funkce pro nejrychlejší možný výkon, a pokud možno bez zbytečných omezení tak, aby výsledkem byla kvalitní podoba grafického uživatelského rozhraní. Zároveň je ale účelem této knihovny abstrahovat nízkoúrovňový kód do podoby, které bude rozumět i teorie neznalý vývojář.

Hlavním těžištěm knihovny je využívání grafického rozhraní OpenGL, které je i ve vyšších verzích běžnou součástí softwarové výbavy dnešních operačních systémů nebo bývá aktualizováno grafickými ovladači grafických karet a akcelerátorů. Knihovna využívá nástroje maximálně verze 4.0, která byla vydána už roku 2010. Tím je zvýšená pravděpodonost, že uživatelské prostředí bude fungovat na většině zařízení.

OpenGL je využíváno plně kvůli svému proudovému zpracování geometrických dat a většina vykreslovacích funkcí je realizována v shaderech. Knihovna obsahuje dvojici připravených řešení, ale dovoluje vlastní řešení s poměrně jednoduchou implementací.

Knihovna vznikala společně s ukázkovými příklady a byla rozšířena pokaždé, když aktuální postup byl příliš neefektivní. Veškeré hotové metody byly silně ovlivněny způsobem práce naznačeným v kapitole 4 a demonstrují sílu aktuálního stavu počítačové grafiky.

5.2 Využité knihovny třetích stran

Aby knihovny mohly svojí činností co nejlépe využívat pokročilejších funkcí, je záhodno používat na menší kroky k tomuto cíli již hotových řešení. Jedním z těchto řešení je knihovna **glew**, která má na starosti načítání veškerých dalších funkcí OpenGL, které se nenacházejí ve verzi OpenGL 1.1. K tomuto načítání dochází z dynamicky linkované knihovny, která je přístupná nejčastěji díky aktualizacím ovladače grafické karty a nachází se v adresářích operačního systému.

Další důležitou knihovnou pro správu obsahu grafické scény je matematická knihovna **glm**, která se stará o správnou práci s trojprvkovými vektory pro pozici, čtyřprvkovými vektory pro barvu (RGBA) a také s maticemi potřebné pro veškeré transformace (translace, rotace, měřítko, perspektiva a další).

Knihovna glm je konstruována jako hlavičková knihovna, veškeré definice jsou přítomny v hlavičkových souborech této knihovny, což dovoluje nejrychlejší možnou implementaci této knihovny. Knihovna glm také obsahuje velké množství matematických funkcí, které dovolují aplikovat matematické poznání v oblasti počítačové grafiky bez nutnosti vlastní implementace.

Knihovny samozřejmě využívají knihovnu **vstgui**, ze které čerpá pouze pár základních tříd pro správu editačního okna. A také třídu pro vykreslení OpenGL kontextu. Tento přístup dovoluje využívat staré vykreslovací techniky společně s novými, touto prací představenými technikami.

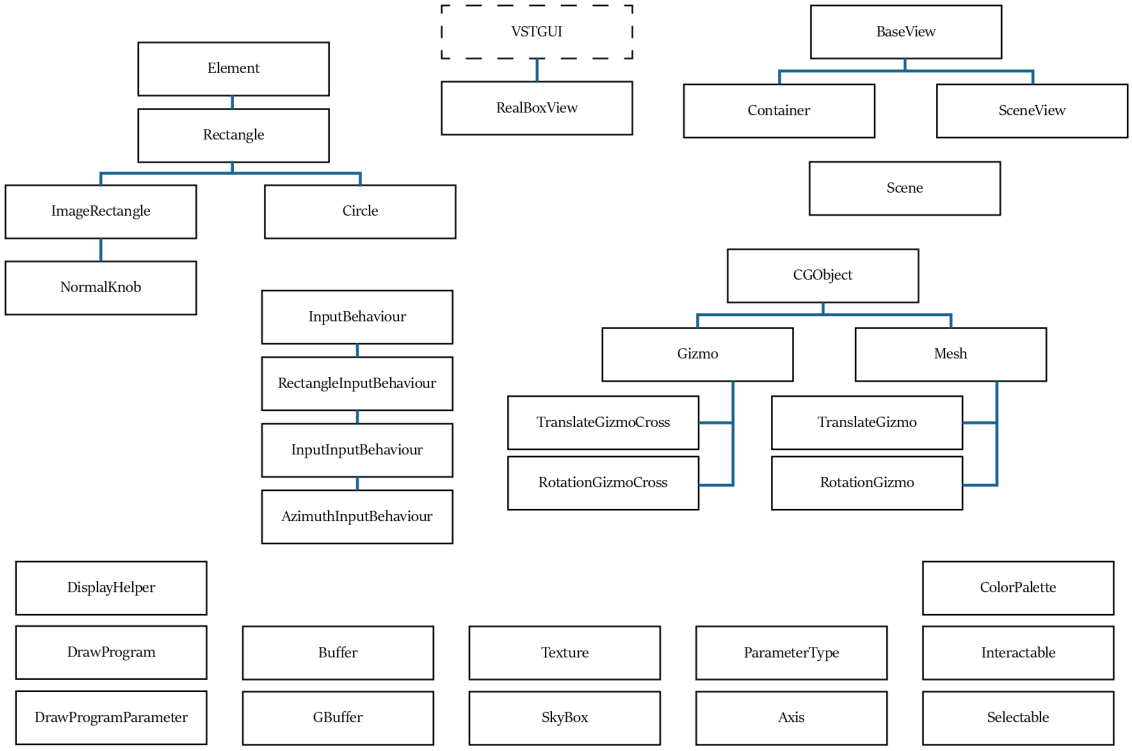
Poslední využitou externí knihovnou je **stb**. Ta dovoluje načítání bitmapových dokumentů přímo z paměti, tedy konkrétněji z resource balíku zvukového zásuvného modulu. Tato knihovna je důležitá v tom ohledu, že dovoluje načítání komprimovaných obrazových formátů, jako například png a jpeg.

5.3 Struktura knihovny RealBox

V následující kapitole bude popsána pouze část knihovny RealBox. Detailní popis struktury tříd a jejich funkcí je k nalezení v dokumentaci, která je přílohou této diplomové práce. Obecně lze rozdělit strukturu knihovny na tři části: třídy pro tvorbu 2D uživatelského rozhraní, tvorbu 3D uživatelského rozhraní a pomocné funkce.

Jak již bylo naznačeno, knihovna vychází ze třídy `COpenGLView` knihovny **vstgui.** Tato třída při svém vzniku vytváří OpenGL kontext, což je sada objektů určených ke kreslení do obrazového zásobníku grafické karty. Tento kontext je podmínkou pro jakoukoliv komunikaci s grafickou kartou. Veškerým operacím, které je potřeba provést, jako například volání vykreslovacích funkcí, musí předcházet volání funkce `getPlatformOpenGLView()->makeContextCurrent()` třídy `COpenGLView`.

Třída `COpenGLView` a všichni její potomci dědí ze třídy `CView`, takže mohou být libovolně umístěni do okna editoru — stejně jako jakékoliv další ovládací prvky knihovny VSTGUI. Prvek získá data o své velikosti, poloze, a případně jsou zavolány její funkce informující o poloze či akci myši.



Obrázek 8: Struktura tříd knihovny RealBox.

5.3.1 Jádro knihovny RealBox

Tuto práci má na starosti třída `RealBoxView`, která slouží jako prostředník mezi knihovnou VSTGUI a RealBox. Její funkcí je kromě správy veškerých dat, které získá ze své rodičovské třídy vytvořit objekt knihovny RealBox, který dědí z rozhraní `BaseView`.

`BaseView` může nabírat dvou podob. První podoba slouží k zobrazování 2D uživatelského prostředí a jmenuje se `Container`. Druhým možným potomkem je třída `SceneView`, která by měla hostit objekt typu `Scene`, který v sobě ukrývá veškeré informace o 3D scéně, která má být do části okna vykreslena.

Obě tyto podoby je vhodné vytvořit přichystanými funkcemi `createContainer` a `createSceneView`, z nichž druhá vyžaduje jako argument objekt typu `Scene`. Třídám `Container` a `SceneView` pro komunikaci s kontrolní částí je potřeba zadat odkaz na kontrolní část, aby mohly ohlásit případnou změnu hodnoty parametru funkcí `appendController`.

Komunikace druhým směrem, v případě, že kontrolní část změní hodnotu parametru, probíhá propagací prostřednictvím funkcí `update`, které vyžadují argumenty s číslem parametru a s novou normalizovanou hodnotou. Veškeré aktivní a pasivní prvky či objekty ve 3D scéně se novým hodnotám mohou přizpůsobit.

5.3.2 Třídy pro tvorbu 2D uživatelského prostředí

Základní třídou pro správu 2D uživatelských prvků je již zmiňovaná třída `Container`. Jejím účelem je hostit, spravovat a vykreslovat objekty typu `Element`. Při vložení nového prvku jsou tyto objekty vloženy do zásobníku a při první příležitosti inicializovány pro grafickou kartu. `Container` prvkům předává nové hodnoty a také prvky ničí při zavření editačního okna.

Každý `Element` v sobě obsahuje informaci o své aktuální hodnotě, o parametru, ke kterému přísluší, o své pozici v rámci třídy `Container`, a také způsob, jakým reagovat na uživatelské vstupy. Způsob je definován třídou `InputBehaviour`, která může být definována pro různé typy uživatelské interakce.

`InputBehaviour` nejprve zjišťuje, jestli se myš vůbec nachází nad prvkem. Pro tyto účely jsou v knihovně RealBox připraveny dvě třídy, které z `InputBehaviour` dědí. Jedná se o třídu `RectangleInputBehaviour`, která svým potomkům předává informaci, jestli se myš nachází nad aktuálním prvkem, prostřednictvím vnitřní proměnné `hovered`.

Ze třídy `RectangleInputBehaviour` dědí další ze zástupců správy místa kurzoru — `CircleInputBehaviour`, která v případě, že je podmínka pro obdelníkovou oblast splněna, zahájí zkoumání skrz vzdálenostní funkce a taktéž výsledek uloží do proměnné `hovered`.

S výsledkem těchto testů je už pak snadné operovat, například pro zvýraznění aktuálního uživatelského prvku, nebo akcí při stisku myši. Zmíněné dvě třídy s akcí myši už dále nic neprovádějí, pouze ukládají pro své případné potomky proměnnou `clicked`, která značí, že do prostoru uživatelského prvku bylo kliknuto.

Potomkem třídy `CircleInputBehaviour` je například třída `AzimuthInputBehaviour`, která registruje při tažení myši změnu úhlu od středu prvku a určuje novou hodnotu parametru. Hodnota se nachází v rozmezí [0,1] a s prvkem je možno libovolně otáčet. Daný prvek (třídy `Element`) může na tuto změnu reagovat jakkoliv, například otočením své vizuální reprezentace.

Důležité je si všimnout, že tímto způsobem je odloučeno chování prvku od jeho vizuální reprezentace, což dovoluje lepší modulárnost a lepší práci s třídami uživatelského prostředí. To je jedním z rozdílů, který odlišuje knihovnu RealBox od ostatních grafických knihoven.

Pro vizuální reprezentace grafických prvků je v knihovně RealBox připraveno několik tříd, například třída `Rectangle`, která dovoluje vykreslit oblast jednou barvou, nebo může být upravena tak, aby vykreslovala jakoukoliv další grafiku definovanou vývojářem. Toto vykreslení plně využívá proudového zpracování geometrických dat.

I pro tento jednoduchý úkon vybarvení oblasti jednou barvou se využívá vlastností shaderů, které jsou dohromady skládány do *„grafických programů“*. Grafický program v tomto kontextu představuje jednu větev proudového zpracování geometrických dat a obsahuje aspoň jeden *vertex shader* a jeden *fragment shader*. Ty jsou definovány jako textové řetězce, které se pomocí OpenGL při otevření okna zkompilují.

Jakmile OpenGL tyto řetězce zkompiluje, mohou být zahozeny, protože se nacházejí jako instrukce uvnitř grafické paměti. Předpokládá se, že tyto instrukce budou opakovaně využívány, a proto v knihovně RealBox je na tuto situaci připravena třída `DrawProgram`.

`DrawProgram` je jednou z nejkritičtějších tříd knihovny RealBox. Dovoluje totiž jednoduchou správu grafických programů, jejich shaderů a parametrů, které shadery ovlivňují. Parametry (v angličtině také jako *uniforms*) se liší od zvukových parametrů, ale mohou jimi býti ovládány. Docíleno toho je možno funkcí `assignParameter`, která do objektu třídy `DrawProgramParameter` uloží odkaz na proměnnou základních typů.

Parametry jsou pak do grafického akcelerátoru posílány opět prostřednictvím OpenGL při každém volání funkce `use` třídy `DrawProgram`. Grafický program pak hledá korespondenci každého z parametrů a jeho jména s místem v grafické paměti. Shadery pak pracují s aktualizovanými hodnotami.

Například vstupními parametry pro třídu `Rectangle` jsou jeho pozice, velikost (oba parametry v pixelech), barva a také transformační matice, která dovoluje správné vykreslení do přiděleného prostoru. Transformační matici je možné upravit, aby dovolovala zvětšování, otáčení, či zkosení.

Fragment shader vykresluje do oblasti definované při vykreslování grafického programu. Pro veškeré třídy typu `Rectangle` se jedná dva trojúhelníky sestavené tak, aby dohromady vytvářely obdelník. Protože jsou pozice bodů obdélníku využívány často, je využíváno statických funkcí pomocné třídy `Buffer`.

Shadery použité ve třídě `Rectangle` mohou být samozřejmě nahrazeny uživatelskými shadery, které mohou ovlivňovat hraniční body, nebo častěji způsob, jakým je prostor vybarvován.

Příkladem takovéto úpravy je například třída `Circle`, která do prostoru obdélníku dovoluje vykreslit kruh v závislosti na parametru poloměru, nebo třída `ImageRectangle`, která do této oblasti vykresluje bitmapový obrázek.

Při tom je využíváno objektu typu `Texture`, který spravuje odkaz na místo v paměti grafické karty, kde je tento bitmapový obrázek uložen. S tímto objektem je pak možné dále pracovat vykreslením obrázku, či jeho úpravou pomocí shaderů.

Pomyslným vrcholem všech zmíněných tříd může být třída s názvem `NormalKnob`, která dědí ze třídy `ImageRectangle`. Jejím úkolem je vykreslení otočného knoflíku, ovšem za pomocí minimálně dvou a maximálně pětice textur. Třída využívá metody navržené speciálně pro knihovnu RealBox a popisuje ji podrobněji kapitola „5.4.2 Otočný knoflík s využitím Material Capture“ na stránce 38.

5.3.3 Třídy pro tvorbu 3D scény

Rozdílný přístup pro tvorbu uživatelského rozhraní nabízí již zmíněná třída `Scene`, která je postoupena jako argument pro funkci `createSceneView` třídy `RealBoxView`. Má na

starosti vykreslování 3D scény s pomocí virtuální kamery. Jak seznam ukazatelů na 3D objekty, tak ukazatel na kameru je uložen do proměnných `mainCamera` a `meshes`.

Počítá se s tím, že vykreslovací funkce `draw` třídy `Scene` bude volána každou šedesátinu sekundy prostřednictvím její vyšší třídy, ale není to nutnou podmínkou, a záleží na implementaci této třídy vývojářem.

Třída typu `Mesh` je strukturou připravenou k uchovávání informací o viditelné polygonové síti, jejích texturách, případně barvě. Nejdůležitější součástí této třídy jsou odkazy na místa v paměti, kde se nacházejí zásobníky bodů, texturových souřadnic k těmto bodům a normály bodů. Aby vůbec tato data byla nahrána do paměti grafického procesoru, musí být nahrána z disku, konkrétně z dynamicky linkované knihovny zvukového zásuvného modulu. To je prováděno funkcí statické pomocné třídy `DisplayHelper::LoadOBJ`.

`Mesh` je specializovaným typem třídy `CGObject`, který v sobě uchovává veškeré nutné transformační funkce nutné k posunu a rotaci objektu. Základní proměnné jsou uloženy ve třech trojprvkových vektorech: `position`, `scale` a `eulerRotation`, které obsahují reprezentaci středu objektu v prostoru. Tyto tři proměnné jsou ještě obohaceny o informaci natočení v podobě kvaternionu do proměnné `rotation`, která je primární zdroj dat o natočení objektu, a kdykoliv je tato hodnota změněna, do `eulerRotation` je vypočítána a konvertována nová hodnota.

Pro tyto proměnné jsou zde připravené výpočetní funkce, které připravují transformační matici potřebnou pro vykreslení scény. Transformační matice je sestavena jako výsledek maticového násobení škálovací, rotační a translační matice, v tomto pořadí.

Objekty třídy `CGObject` lze řadit do hierarchické struktury usazením objektu do proměnné `parent`. V takovém případě je transformační matice vypočítávána sekvenčně až do té doby, dokud nedorazí do kořenového objektu (s hodnotou `NULL`). Vše je připraveno ve funkci `getModelMatrix`.

Z rotační matice je možné získat i další informace o natočení tělesa, například užitečnou informací je vektor mířící směrem dopředu, nahoru a doprava od objektu. Tyto hodnoty lze lehce získat, protože se jedná o třetí, druhý a první sloupec rotační matice (v tomto pořadí).

Další užitečné funkce třída `CGObject` nabízí v oblasti lokální rotace. Je zde k nalezení funkce `rotateLocalX`, která využívá vlastností kvaternionů pro (díky knihovně glm) snadné nasměrování objektu o určitou změnu úhlu v dané ose. Samozřejmostí jsou variace této funkce pro ostatní lokální ortogonální osy.

Funkce pro přímou i krokovou změnu trojice parametrů `position`, `scale` a `eulerRotation` jsou samozřejmostí.

Pro správné vykreslení objektů dědící z třídy `CGObject` je nutné pracovat i s konceptem virtuální kamery. Z tohoto důvodu je součástí knihovny RealBox také třída `Camera`. Ta spravuje podobné informace o poloze, až na fakt, že jejím klíčovým argumentem pro rotaci není trojice nezávislých eulerových úhlů, ani kvaternion, ale směr pozorování v podobě proměnné

`target`. Z této pozice už je pak je výpočet rotační matice jednoduchou operací, pokud bereme v potaz, že vektor mířící od kamery má mít vrchní hranu obrazu vždy natočenou vzhůru.

Protože ne vždy se podaří zaplnit pohled na scénu tak, aby byla plně zaplněna objekty, je vhodné počítat i s variantou, kdy na pozadí objektů bude vykreslena plná barva. Toho lze docílit běžnou funkcí OpenGL `glClear` před vykreslením všech objektů, a nebo lze s pozadím pracovat pokročileji, například vykreslením na pozadí objekt, který bude vždy ostatními objekty překryt.

Toho je v praxi docíleno vymazáním hloubkového bufferu, který určuje pořadí objektů při vykreslení, následně po zobrazení pozadí. Textura pozadí však musí mít vhodný formát, aby vyplnila veškeré možné zorné úhly. Je možné použít jednu texturu, která vznikla sférickým mapováním prostoru. Metoda však roztahuje vrchní a spodní póly obrazového materiálu, a proto RealBox využívá šesti čtvercových textur složených do kostky.

Pro pohodlnou práci s touto sadou bitmapových obrázků nabízí knihovna RealBox třídu `SkyBox`. Její využití je v ukázkových případech mířeno spíše na vhodná data pro simulaci odrazu prostředí (*enviroment mapping*), dalších využití však najde více.

Poslední třídou je třída `GBuffer`. Účelem třídy je spravovat proudové zpracování geometrických dat, přes dvoje volání. První vykreslení přes tento objekt (přes metodu `firstPassRender`) zapíše veškeré důležité informace do zásobníků textur (*GBufferu*), které budou využity při druhém vykreslovacím průběhu (`secondPassRender`), který už výsledná data seřazuje pouze na obdélník podobně jak třída `Rectangle`.

Mezi informace předvypočítané pro druhé vykreslení patří: globální pozice pixelů, globální normálová mapa, barevná informace společně s informací o hrubosti a poslední informace o indexu objektu pro možnou detekci objektů po najetí kurzoru.

Použití této třídy je koncipováno tak, aby vývojář poskytl fragment shader pro druhou fázi zpracování odloženého stínování (nebo použil ten, který se nachází v ukázkových případech) a využil objekt `GBuffer` k vykreslení scény. Více konfigurace ani úpravy není potřeba.

5.3.4 Pomocné třídy

Většina tříd pro vykreslení jak 2D, tak 3D obsahu je závislá na vstupních datech, která definuje vývojář zásuvného modulu. Může se jednat o textury nebo 3D polygonové sítě, které mají být do editoru zvukových parametrů vykresleny a jsou se zvukovým modulem přenášeny v podobě takzvaných zdrojů (anglicky *resources*), které jsou do paměti čerpány z dynamické knihovny zvukového zásuvného modulu za běhu programu.

Data je nejlépe přenášet v jejich komprimované podobě, avšak pro práci s nimi je potřeba je převést do původní nekomprimované podoby. K tomuto účelu slouží v knihovně RealBox statická třída `DisplayHelper`, která obsahuje funkce `LoadImg` a `LoadCubeMapTexture`. Taty funkce jsou schopny vytáhnout ze zdrojů zásuvného modulu bitmapový soubor a načíst jej do paměti grafické karty, připravené k použití.

Aktuálně RealBox prostřednictvím těchto funkcí nabízí import bitmapových obrázků formátu *JPEG* a *PNG*.

Podobně funguje funkce `LoadOBJ`, která už nepracuje s komprimovaným souborovým typem, ale nejzákladnějším možným zápisem geometrických dat — *Wavefront .obj* formátem. Z tohoto jsou vyčteny základní informace o polohách bodů, jejich skládání do polygonů, informace o normálách i o texturových souřadnicích.

O těchto datech se předpokládá, že je vygeneroval jeden z grafických programů, který dovoluje export do tohoto formátu. Důležité je zmínit, že aktuálně knihovna RealBox přijímá pouze trojúhelníkové polygony.

Třída `DisplayHelper` nabízí možnosti geometrická data nejen importovat, ale také vytvářet. Důkazem mohou být funkce pro tvorbu krychle, kužele i válce.

Načítat nemusí být nutné jen viditelná data, ale také zdrojový kód definující chod shaderů a grafických programů. Příprava těchto textových řetězců probíhá právě ve funkcích `LoadShader`, `CreateShader` a `CreateProgram`. Výstupy těchto funkcí jsou pak ukazatele na tyto programy v grafickém akcelerátoru, a na jejich správnosti závisí celá knihovna RealBox.

Pro možnosti interakce se zde nachází funkce pro geometrické vyjádření paprsku, který směřuje z kamery směrem ke kurzoru myši v trojrozměrné reprezentaci. Tuto funkcionalitu je možné najít v metodě `getMouseRayDirection`.

Pro další geometrickou kontrolu slouží funkce `getIntersectRayPlane`, která ze vstupních parametrů vlastností roviny a vlastností paprsku vypočítá místo jejich střetnutí.

Pro pomoc s interakcí v prostoru pro otáčení a posouvání je do knihovny RealBox zakomponovány třídy `TranslateGizmo` a `RotateGizmo`. Tyto třídy dědí z třídy `Mesh`, a ve výsledku se jedná o úzké válce, které ilustrují polohu a možnosti translačního a rotačního pohybu osy.

Pro reprezentaci všech možných transformací jsou manipulační táhla seskupována do trojic, v knihovně RealBox nazvaných `TranslateGizmoCross` a `RotateGizmoCross`. Tyto třídy jsou vkládány do scény funkcí `appendToScene`. Akce vykonané uživatelem musí však už být definovány v každé scéně zvlášť.

5.4 Technologie knihovny RealBox

Knihovna RealBox pro geometrickou práci používá transformační matice a kvaterniony, jak bylo popsáno v předchozích kapitolách. To je standardem pro práci s 3D grafikou obecně, knihovna diplomové práce je využívá i pro 2D grafiku, protože dovoluje jednoduše ovlivňovat natočení uživatelských prvků. To stejné platí pro využití shaderů, které jsou pro vylepšování efektivity tvorby stěžejní.

5.4.1 Vykreslovací systém 3D scény

Centrem vykreslovacího systému knihovny je technologie odloženého stínování (*deferred shading*) popsaná v kapitole 4.3 a její implementace zmíněná při popisu třídy `GBuffer`. Tento způsob práce dovoluje větší možnosti při snaze přiblížit se obrazem realistickému vzhledu.

V ukázkových případech je odložené stínování využito pouze povrchně k simulaci odrazů scény s pomocí metody odraženého prostředí (*enviroment map*) a možností výběru myší samostatných objektů. Odložené stínování ale dovoluje mnohem pokročilejší používané způsoby. Mezi příklady patří simulace odrazů na základě předvypočítaného obrazu (*screen space reflections*), simulace okluze skrytých záhybů (*screen space ambient occlusion*) a mnoho dalších. Právě na implementaci těchto metod je knihovna RealBox připravena.

Protože je pořád nutné myslet na výkon zvukového zásuvného modulu, nelze tyto metody implementovat všechny najednou, a je nutné volit kompromisy. Jedním z kompromisů, který byl vybrán u ukázkových příkladů, je zanedbání vrhání stínů objekty. Efekt vrženého stínu je u pluginu RealStomp realizován už předpřipravenou texturou, která je aplikována na objekty a hlavně na podlahu. Způsobu šlo využít, protože se celý objekt kromě rotace otočných knoflíků nepohyboval.

5.4.2 Otočný knoflík s využitím Material Capture

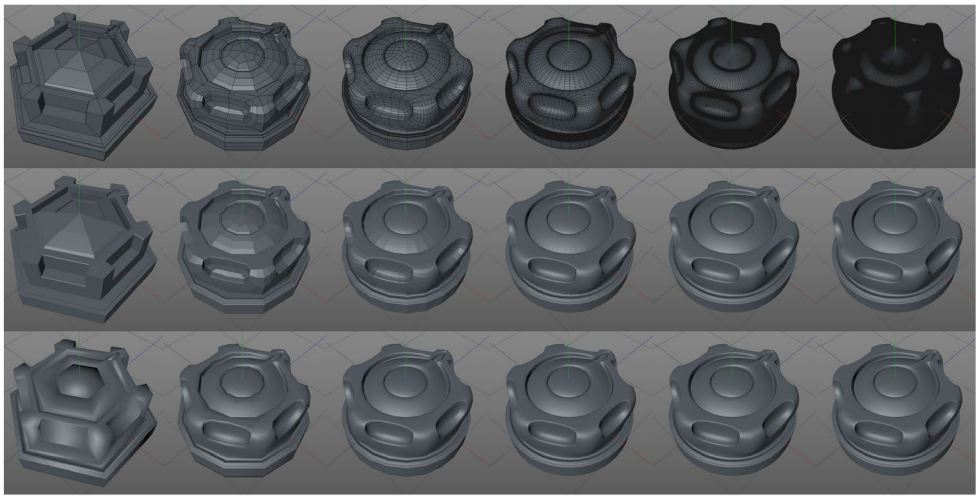
Knihovna RealBox slouží kromě nástroje pro vykreslení 3D objektů do scény hlavně jako demonstrace nových možných způsobů pro řešení již dlouholetých problémů. Jedním z klíčových problémů je tvorba realisticky vypadajících otočných knoflíků s pomocí „metody animovaného knoflíku“ popsané v sekci C-C kapitoly 3.2.2.

Přijatelného vzhledu otočného knoflíku lze dosáhnout množstvím obrázků zachycených ve všech možných polohách, což je značně neefektivní a paměťově neekonomické.

Možnou alternativou je využití knihovny RealBox pro vykreslení 3D objektu otočného knoflíku a jednouduchou geometrickou transformací tento objekt v prostředí otáčet. Avšak v případech, kdy tento model má mít zaoblené tvary, dosahuje počet jeho polygonů až do řádu desetitisíců. Grafik zodpovědný za modelaci tohoto objektu může tento počet zmenšit úpravou topologie polygonové sítě objektu tak, aby hustota polygonů na zaoblených částech byla vyšší a na částech rovných nižší.

Dalším možným trikem jak zmenšit počet polygonů, a tedy i paměťovou náročnost, je smazat polygony, u kterých je nulová nebo nízká pravděpodobnost, že je uživatel uvidí. Nejčastěji se jedná o vnitřní nebo zadní strany uživatelských prvků.

Po aplikaci obou zmíněných postupů může být dosaženo výrazných paměťových úspor, ale i přesto je nutné využívat například Phongova stínování - principu, kdy jsou normály jednotlivých bodů polygonů interpolovány na základě natočení samotných polygonů. Efekt Phongova stínování je zobrazen na obrázku níže (Obrázek 9).



Obrázek 9: Porovnání polygonových hustot otočného knoflíku.
Nahoře: zobrazení polygonů; Uprostřed: konstatní stínování;
Dole: Phongovo stínování.

V případech, kdy ani Phongovo stínování nepomůže dosahovat dojmu oblých vyhlazených hran, a přesto je počet polygonů příliš vysoký na to, aby se 3D model načetl kdykoliv chce uživatel otevřít editační okno zásuvného modulu, je potřeba přistoupit k jiné metodě.

V této práci je navržena nová metoda — **metoda normálových map**, která využívá technik popsaných v kapitole 4.4, k výsledku, kde pro veškeré možné polohy bude možné využít pouze dvou obrázků — obrázku materiálu a obrázku triplanární normálové mapy.

Zdrojová data pro tento styl práce musí být opět vytvořena v grafickém editoru, postup exportu se však liší. Místo vykreslení animace rotace do např. 255 snímků, kde každý snímek zaznamenává rotaci knoflíku o určitý úhel, mohou být provedeny pouze dva výpočty.

Z 3D reprezentace otočného potenciometru je potřebné získat barevnou informaci o jejích normálách, což je výpočet poměrně jednoduchý, důležité je však normalizovat hodnoty do podoby, která bude přenositelná bitmapovým obrázkem. Hodnoty barevných kanálů tedy nesmí klesnout pod hodnotu 0.

V této hotové vykreslené normálové mapě se nachází informace o natočení jednotlivých pixelů, což dovoluje následné umístění otočného potenciometru do širšího prostředí za pomocí skyboxů či sférických map. Co však ale tato mapa nerespektuje, je bližší prostředí — konkrétně stíny. Jedinou možností pro vytvoření předvypočítané dynamiky podobné dynamice stínů je využívat statistický model *ambientní okluze*.

Druhým prvkem, který je potřeba získat, je obrázek koule, která je umístěna v libovolném prostředí. Povrch koule odráží většinu svého okolí a reaguje na světelné podmínky. Se znalostí umístění normál této koule může obrázek fungovat jako vyhledávací tabulka hodnot pro barvu materiálu včetně veškerých odrazů a povrchových charakteristik.

Očekává se, že otočné potenciometry nebudou pouze z jednoho materiálu, z tohoto důvodu se předpokládá i vytvoření **indexové mapy**, která určuje, na jaké části textury má být aplikován jaký materiál. Knihovna RealBox aktuálně podporuje možnost tří materiálů, které jsou definovány černobílou texturou, a to konkrétně černá pro první materiál, 50% šedá pro druhý materiál a bílá pro třetí materiál. V případě jiných hodnot dojde ke kombinaci mezilehlých materiálů v daném poměru.

Využití metody normálových map pro zobrazování ovládacích prvků je implemetováno pouze pro otočný potenciometr formou třídy `NormalKnob`. Jeho konstruktor je volán s názvem normálové mapy, který se nachází v seznamu zdrojových souborů, pozicí v rámci objektu třídy `Container`, a také velikost prvku, v jakém se má vykreslit do okna.

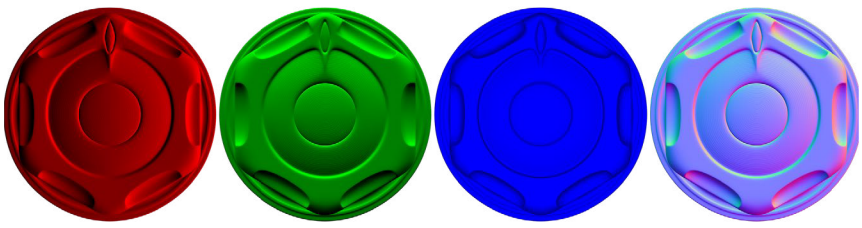
Třída `NormalKnob` dědí parametry a funkce ze třídy `ImageRectangle`, která zvládá v OpenGL kontextu vykreslovat libovolný obrázek. Třída `NormalKnob` předchozí třídu rozšiřuje o možnost tento obrázek okolo středu otočit, a hlavně využívat možnosti definovat barvu z matcap textury v závislosti na natočení.

Aby třída `NormalKnob` a veškerá její logika mohla správně pracovat, je možné naplnit zásobník dalších textur funkcemi `armMatCap1`, `armMatCap2`, `armMatCap3`, `armMaterialMask`, `armAmbientOcclusion` na nově vzniklém objektu této třídy. Všechny tyto funkce jsou volány s argumentem odkazu na seznam zdrojů.

Veškeré textury jsou načteny do paměti GPU až v době inicializace, která proběhne v momentu, kdy se vytvoří editační okno a OpenGL kontext, buď v době inicializace, nebo při dalším volání vykreslovací smyčky.

Tento způsob práce lze také nazvat odsunutým stínování, jen pouze v tomto případě je první průběh předvypočítán ještě dříve, než je editor vůbec otevřen. V třídě `NormalKnob` se tedy pracuje obdobně jako v případě druhého průchodu odloženého stínování. K nalezení je tu pouze jedna dvojice vertex a fragment shaderů.

Do vertex shaderu jsou posílány informace o natočení, které proběhne vynásobení pozic hraničního obdelníku s rotační maticí. Uvnitř fragment shaderu dochází nejdříve k přepočítání barev z normálové mapy, které se doteď nacházely v rozsahu 0-1 pro možnou editaci v grafických programech, na rozsah -1 až 1. Směr normály je následně otočen o záporný úhel, o který byla otočena textura normálové mapy, takže je zachována informace o globálním natočení normál.



Obrázek 10: Triplanární normálové mapy
pro tři směry a jejich spojená podoba.

Trojrozměrný vektor normály je pak využit jako klíč k textuře **matcap**, která slouží jako vyhledávací tabulka, jejíž klíč je trojrozměrný a měl by být dvourozměrný, musí nutně dojít k transformaci. Zde je využito faktu, že matcap textura byla zachycena z ortogonálního frontálního pohledu, a tedy může být osa Z vynechána. Předtím je nutné klíč transformovat opět na rozsah 0-1, jelikož souřadnice pro vyčítání z textur pracují v glsl v tomto rozsahu.

Využitím metody normálových map je možno zeefektivnit způsob práce při potřebě změnit vlastnosti materiálu a barvy objektu. Toho může být docíleno jednoduchou úpravou matcap textury v bitmapovém editoru (např. Adobe Photoshop) nebo použitím jednoho z veřejně dostupných matcap zdrojů.

6 Tvorba ukázkových zásuvných modulů

Tvorba grafické knihovny pro zásuvné zvukové moduly by byla zbytečná, kdyby třídy a funkce nebyly podrobeny testu a demonstrovány na příkladech možného využití. Pro názornou ukázku práce s knihovnou RealBox byly vytvořeny tři zvukové zásuvné moduly.

Veškeré zvukové efekty jsou vytvořeny jako samostatné projekty v programu Visual Studio 2019 a využívají knihoven vst3, včetně vstgui. Na tyto statické knihovny v podobě nezkompilovaných projektů jsou navázány závisle, takže před jejich zlinkováním dojde k jejich kompilaci. Stejným způsobem je nejvhodnější využívat knihovnu RealBox. Pro každý z projektů je nutné definovat cesty, ze kterých mají být hlavičky externích knihoven čerpány.

Knihovna RealBox definuje hlavičkovou cestu ve složce include. Aby mohly být argumenty funkcím knihovny RealBox správně předány, je vhodné v projektech využívat knihovnu **glm** a taktéž je důležité importovat do projektů knihovnu **glew** a **vstgui**.

Aby bylo možné z dynamické knihovny vůbec vytvořit VST plugin, měla by se odkazovat na knihovnu **vst**. Kromě toho musí definovat vstupní bod pro hostitelskou aplikaci. Tato procedura je poměrně složitá, kvůli způsobu jakým vst zvukový plugin vytváří, proto je pro tvorbu zvukového zásuvného modulu vhodnější kopírovat funkční ukázkové projekty a upravovat jejich funkcionalitu.

Pro import souborů (obrázků, 3D modelů a shaderů) je nutné zapisovat jejich cesty a použité indexy do souboru s příponou .rs — díky tomu bude kompilátor vědět, že soubory má zahrnout do jednoho souboru dynamické knihovny.

6.1 Tvorba modulu RealStomp

Na obrázku (Obrázek 11) je vidět polygonová síť kytarového pedálového efektu. 3D model byl vymodelován v grafickém programu Cinema 4D se snahou využít co nejméně polygonů, ale s dostatečnou hustotou polygonů, aby zaoblené hrany mohly odrážet virtuální studiové okolí.

3D model byl částečně otexturován a stínové mapy pro podložky byly dopředu zapečeny. Zapečení stínů uloží vliv stínů do textury a stíny už není nutné při vykreslování vypočítávat.

Některé objekty ale nemají pevně stanovenou texturu, a tedy ani barvu. Tu je možno definovat přímo při vykreslování.

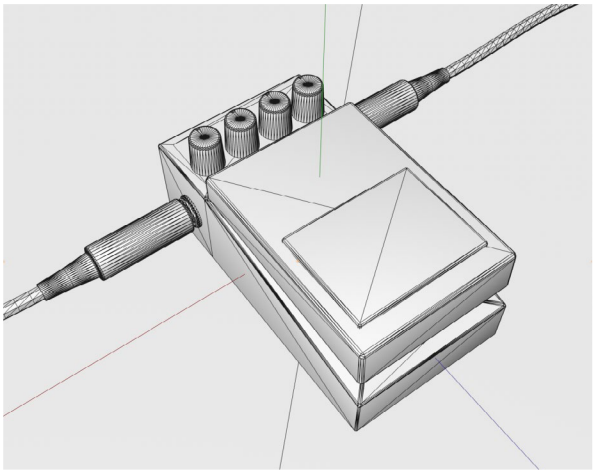
Pro možné načítání textur a 3D modelů do zásuvného modulu je využito třídy `DisplayHelper` a její funkce `LoadShader`, `LoadOBJ`, `LoadImg`. Poté, co jsou textury a 3D modely načteny, jsou připraveny k vykreslení. Z jejich pozic a natočení se získává jejich transformační matice, která se přeposílá dále do grafické karty. Taktéž třída `Mesh` v sobě uchovává odkaz na zásobník bodů a trojúhelníkové sítě stejně jako textury potřebné pro vykreslení. Veškeré tyto informace jsou při volání vykreslovací smyčky předány objektu třídy `GBuffer`, která ve dvou průbězích spustí vykreslení.

První průběh vykreslení zohlední pozici kamery a pozici všech objektů ve scéně, aby vytvořil tři nové obrázky: globální pozici veškerých pixelů ve scéně, normálovou mapu všech pixelů ve scéně a albedo mapu všech pixelů ve scéně. Rozsah hodnot poziční mapy je v rozmezí hodnot číselného typu `float`. Rozsah hodnot vektoru normálové mapy se pohybuje v hodnotách jednotkové koule. Mezivýstupy tohoto prvního průběhu zobrazuje Obrázek 4.

Druhý průběh vykreslení pracuje pouze s těmito třemi bitmapovými daty z obrázku (Obrázek 4). Na jejich základě vypočítává možné odrazy, které se mohou nacházet ve scéně. Protože je výpočetně náročné do odrazu zahrnout barvu dalších objektů ve scéně, používá se předvypočítaná barva okolí v objektu `Skybox`.

To samo by ale pro simulaci realistického vzhledu nestačilo. Výsledek by byl až příliš lesklý, a proto je nutno do vykreslovacího modelu započítat i faktor *„hrubosti materiálu“*. Ten je určen u každého z objektů a může být definován v prvním průběhu speciální texturou. Tento parametr udává, jak moc se světlo rozptýlí po dopadu na tento materiál.

Toto rozptýlení je ovlivněno množstvím odražených vzorků, které jsou v tomto případě na hodnotě padesát. Vzorky využívají vektoru odrazu, který se poměrně k hrubosti pixelu náhodně pozmění, a hodnota barvy je vyčtena z skyboxu tohoto pozměněného vektoru. Součet výsledných barev je poté sečten a podělen počtem vzorků.

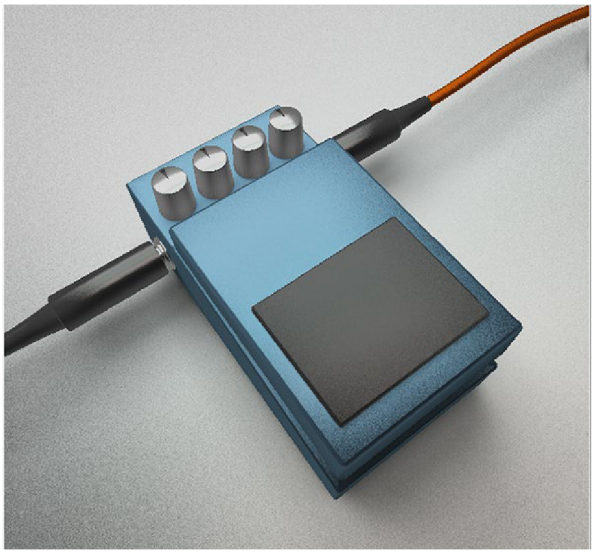


Obrázek 11: Polygonová síť stomp kytarového efektu RealStomp.

Odraz objektu je pak pouhým vynásobením spojen s albedo mapou. Hotový obraz projde ještě barevnou korekcí, aby výsledek nepůsobil příliš tmavě. Barevně upravený výstup druhého vykreslovacího průběhu je vidět na obrázku (Obrázek 12).

Pro možnou interakci uživatele se zvukovým modulem byla přidána možnost tahem myši pozměnit pozici kamery. Kamera opisuje kruh kolem kytarového efektu a demonstruje sílu vykreslování uživatelského prostředí s 3D vykreslením. Tohoto prostorového efektu by se dříve používanými technikami dosahovalo těžko.

Otočné knoflíky je možné vybírat klikem a prstencem značícím rotační osu je možné knoflík otočit. Takto otočený knoflík je možno dále interpretovat ve zvukový parametr přeposlaný do kontrolní části zásuvného modulu.



Obrázek 12: Výsledná podoba modulu RealStomp.

6.2 Tvorba modulu ReproSpace

Podobných principů využívá zvukový zásuvný modul **ReproSpace**. Ten se snaží simulovat vztah mikrofonu a reproduktoru v prostoru. S každým prvkem se dá libovolně otáčet a pohybovat, a v závislosti na vyzařovacím úhlu reproduktoru a směrové charakteristice mikrofonu je určován decibelový zisk.

Oba objekty byly vymodelovány a otexturovány v programu Cinema 4D, s vužitím pomocných funkcí třídy `DisplayHelper` importovány a převedeny na objetky třídy `Mesh`. Takto vytvořené objekty byly zapsány do zásobníku třídy `ReproSpaceScene`. K nim je připojen jeden objekt třídy `TranslateGizmoCross` a jeden objekt třídy `RotateGizmoCross`.

Uvnitř této třídy dojde při inicializaci k nastavení kamery a je zavolán konstruktor třídy `SkyBox`. Hned poté následuje vytvoření objektu třídy `GBuffer`, aby bylo využito odloženého stínování. Vykreslovací smyčka probíhá poměrně jednoduše. Každý z polygonových objektů je poslán do obou fází grafického programu a vykreslen na obrazovku.

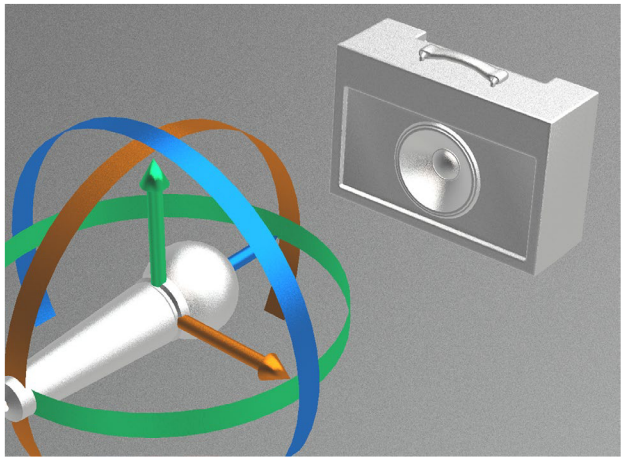
Nejsložitějšími instrukcemi jsou ve scéně reakce na uživatelský vstup. Každou vykreslovací smyčkou dojde k získání indexového čísla objektů v oblasti kurzoru myši, a v případě kliku se označí tento objekt za vybraný. V té situaci se přesune dvojice manipulačních táhel na místo označeného objektu a stanou se objektem závislým (vybraný objekt je jejich rodičem).

Pokud se kurzor myši nachází nad manipulačním táhlem, a je zaznamenána akce táhnutí myši, vytvoří se pro rotační táhlo v místě středu objektu rovina, jejíž normála je orientována směrem natočené rotační osy. Tato rovina je protínána paprskem vycházející z kamery a směřující směrem kurzoru myši. Místo protnutí je zaznamenáno, stejně jako změny této polohy. Z úhlů těchto změn vůči středu je vypočítána hodnota, která je pak přeposlána do lokální rotační funkce objektu. Tím je možno objekty rotovat podle jejich vlastních os.

V případě, že interagovaným táhlem je `TranslateGizmo`, prodlouží se směr osy do souvislé linky, na které je hledán nejbližší bod od paprsku směřující z kamery a kurzoru myši. Posunem bodu po této přímce je posouván i objekt v dané lokální ose.

Při každém posunu a rotaci je vypočítána i vzájemná orientace dvou předních vektorů obou objektů. K tomuto účelu slouží v třídě `CGObject` statická funkce `getFacing`, jejímiž argumenty jsou jakékoliv dva objekty této třídy.

Funkce `getFacing` vypočítává skalární součin dopředného vektoru obou objektů k sobě, tyto hodnoty normalizuje z hodnot [-1,1] do hodnot [0,1] a tyto normalizované hodnoty jsou mezi sebou vynásobeny.



Obrázek 13: Výsledná podoba modulu ReproSpace.

Výsledek je pak přeposlán směrem ke kontrolní části, která je předá přes hostitelskou aplikaci procesní části. Procesní část tuto hodnotu použije jako zisk pro příchozí singál.

Součástí zvukového zásuvného modulu není možnost automatizovat a parametrizovat polohy a rotace obou objektů. Globální souřadnice by nebyly problémem, avšak rotace reprezentovaná eulerovými úhly by nedostatečně reprezentovala automatizační možnosti. Kdykoliv by došlo v rámci automatizace k interpolaci úhlů, docházelo by k nedeterministickému chování. Variantou by bylo jako parametr použít kvaternion, nicméně tím by popis rotace ztratil snadno pochopitelný tvar.

Celková scéna může být nahlížena z několika možných úhlů, díky táhnutí myši mimo interagovatelné objekty je kamerou posouvat po kružnici tak, aby se vždy dívala do středu souřadnic.

6.3 Tvorba modulu NormalKnobs

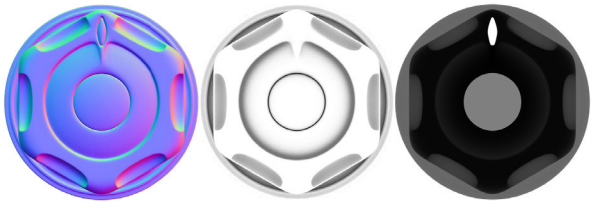
Zvukový zásuvný modul NormalKnobs je demonstrací několika technologických snah zlepšit tvorbu uživatelského prostředí. Zvukový zásuvný modul obsahuje pouze jeden otočný knoflík, který využívá vykreslovací metody normálové mapy zmíněné v kapitole 5.4.2.

Pro ilustraci byl v programu Cinema 4D vytvořen model otočného potenciometru, který obsahuje velké množství zakulacení a záhybů a nebylo by jej možné importovat jako polygonovou síť. K tomuto otočnému knoflíku je zachycena jeho normálová mapa a uložena do obrázku.

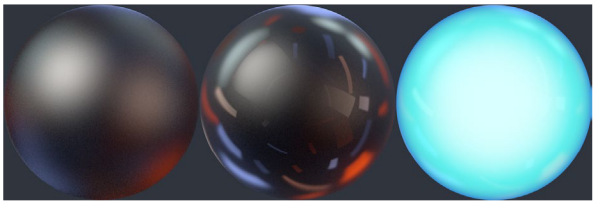
Tento obrázek je načten do zvukového modulu pomocí několikrát zmiňované funkce knihovny RealBox společně s dalšími texturami určujícími materiálovou masku, místa ambientní okluze a hlavně matcap textury, které budou určovat výsledný odraz knoflíku.

Veškeré tyto textury byly nahrány do objektu `NormalKnob` a zbytek práce je přenechán knihovně RealBox. Ta se postará o vhodné vykreslení veškerých vrstev včetně přepočítání normál na barvu otočného potenciometru.

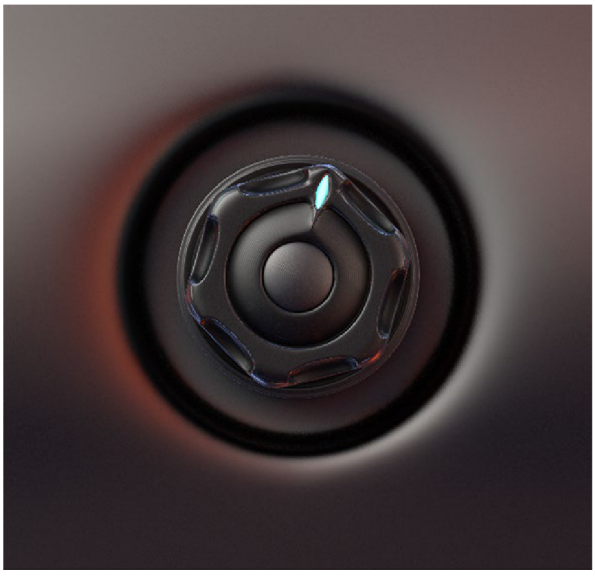
Objektu je přisouzena interakční vlastnost `AzimuthInputBehaviour`, která dovoluje knoflíkem otáčet dle libosti. Výsledná hodnota natočení v tomto zásuvném modulu neovlivňuje jakkoliv výsledný zisk zpracovávaného zvukového signálu. Příklad slouží k demonstraci využití minima bitmapových obrázků k získání přijatelného vizuálního výsledku.



Obrázek 14: Výsledné textury pro zásuvný modul NormalKnobs.
Zleva: normálová mapa, mapa ambientní okluze, materiálová maska.



Obrázek 15: MatCap textury pro zásuvný modul NormalKnobs.



Obrázek 16: Výsledná podoba modulu NormalKnobs

7 Závěr

V této práci byly shrnuty současné způsoby a nedokonalosti práce při vytváření grafických uživatelských rozhraní zvukových zásuvných modulů. Hlavní nevýhodou je zdlouhavý pracovní postup, který nedovoluje rychlé a efektivní změny, navíc disponují paměťově nešetrnými řešeními.

Současným trendům byla navržena inspirace průmyslem počítačových her a grafického průmyslu, který zvládá vykreslování interaktivní grafiky v reálném čase. Poznatky byly uplatněny do podoby grafické knihovny RealBox, která má přispět k rychlejší práci na grafických uživatelských rozhraních. Pro tyto účely byly navrženy dvě výpočetně nenáročné metody *využití odloženého stínování* a *metody normálové mapy*.

Metody by měly razantně zlepšit pohodlnost práce a dovolit ambicióznější vizuální reprezentace grafického uživatelského rozhraní.Pro knihovnu RealBox byla taktéž vytvořena kompletní dokumentace dopomáhající možným vývojářům k rychlé implemetaci knihovny do modulů využívajících technologii VST.

Reference

[1] VST3 SDK Documentation. *Steinberg Website* [online].
Hamburg: Steinberg Media Technologies, 2016 [cit. 2016-10-22].
Dostupné z: http://www.steinberg.net/en/company/developers.html

[2] SEGAL, Mark a Kurt AKELEY. *The OpenGL Graphics System: A Specification Version 1.0* [online].
Sunnyvale: Silicon Graphics, 1994 [cit. 2016-12-04].
Dostupné z: https://www.opengl.org/registry/doc/glspec10.pdf

[3] *History of OpenGL* [online]. Beaverton: Khronos Group, 2015 [cit. 2016-12-10].
Dostupné z: https://www.opengl.org/wiki/History_of_OpenGL

[4] *OpenGL Shading Language* [online].
Beaverton: Khronos Group, 2015 [cit. 2016-12-10].
Dostupné z: https://www.opengl.org/wiki/Related_toolkits_and_APIs

[5] *OpenGL: Related toolkits and APIs* [online]. Beaverton: Khronos Group, 2016 [cit. 2016-10-10].
Dostupné z: https://www.opengl.org/wiki/OpenGL_Shading_Language

[6] *OpenGL Loading Library* [online]. Beaverton: Khronos Group, 2016 [cit. 2016-10-10].
Dostupné z: https://www.opengl.org/wiki/OpenGL_Loading_Library

[7] OpenGLRenderer Class Reference *JUCE Documentation* [online].
Londýn: ROLI, 2016 [cit. 2016-12-03].
Dostupné z: https://www.juce.com/doc/classOpenGLRenderer

[8] WDL Overview *WDL Documentation* [online].
New York: Cockos Inc., 2018 [cit. 2018-12-03].
Dostupné z: https://www.cockos.com/wdl/

[9] SAITO, T., TAKAHASHI, T. Comprehensible Rendering of 3-D Shapes. In: *ACM SIGGRAPH Computer Graphics: Volume 24, Number 4, Aug. 1990*. New York: ACM, 1990, s. 197-206.

[10] BURLEY, B. *Physically Based Shading at Disney* [online]. Siggraph 2012.
Dostupné z: https://blog.selfshadow.com/publications/s2012-shading-course/

[11] LENGYEL, E., GPU-Centered Font Rendering Directly from Glyph Outlines, *Journal of Computer Graphics Techniques (JCGT)*, vol. 6, no. 2, 31-47, 2017
Dostupné z: http://jcgt.org/published/0006/02/02/

[12] LOOP, Charles a Jim BLINN. Rendering Vector Art on the GPU. *GPU gems 3*.
Upper Saddle River, NJ: Addison-Wesley, 2008. ISBN 978-0321515261.
Dostupné také z:
https://developer.nvidia.com/gpugems/GPUGems3/ gpugems3_ch25.html

[13] HUGHES, J. F., et al. *Computer graphics: principles and practice*. Third edition.
Upper Saddle River, New Jersey: Addison-Wesley, 2014. ISBN 978-0-321-39952-6.

[14] MÖLLER, T., et al. *Real-time rendering*. Fourth edition. Boca Raton: CRC Press, 2018. ISBN 978-1-138-62700-0.

[15] DUFKA, F. *Rozšírení grafických rozhraní zvukových zásuvných modulu o 3D grafiku*. Brno: Vysoké učení technické v Brne, Fakulta elektrotechniky a komunikacních technologií, 2017. 38 s. Vedoucí bakalářské práce Ing. Petr Frenštátský.

[16] KENNEDY, John a Michael SATRAN. Comparing Direct2D and GDI Hardware Acceleration *Microsoft Direct2D Documentation,*[online]. 2018, [cit. 2019-05-10]. Dostupné z: https://docs.microsoft.com/en-us/windows/desktop/Direct2D/comparing-direct2d-and-gdi

[17] ŽÁRA, Jiří, Bedřich BENEŠ, Jiří SOCHAR a Petr FELKEL. Moderní počítačová grafika. 2., přeprac. a rozš. vyd. Brno: Computer Press, 2004. ISBN 80-251-0454-0.

[18] SANDERSON, Grant a Ben EATER. Visualizing quaternions: An explorable video series. In: *Ben Eater - Educational Videos* [online]. 2018 [cit. 2019-05-11]. Dostupné z: https://eater.net/quaternions/

[19] SHOEMAKE, Ken. Animating Rotation with Quaternion Curves. In: *ACM SIGGRAPH Computer Graphics: Volume 19 Issue 3, Jul. 1985.* New York: ACM, 1985, s. 245-254.

[20] SLOAN, P-P. J., MARTIN, W., GOOCH, A. a GOOCH, B. The lit sphere: a model for capturing NPR shading from art. In: *GI ,01 Proceedings of Graphics Interface 2001.* Ottawa: Canadian Information Processing Society, 2001, s. 143-150. ISBN 0-9688808-0-0

[21] ZUBIAGA, C. J., MUÑOZ, A., BELCOUR, L., BOSCH C., BARLA, P. MatCap Decomposition for Dynamic Appearance Manipulation. In *Eurographics Symposium on Rendering 2015.* Darmstadt: 2015.

[22] Skeuomorph. In: Oxford Dictionary. [on-line]. [cit. 2019-05-15]. Dostupné z: http://www.oxforddictionaries.com/definition/english/skeuomorph

[23] OSWALD, David and Steffen KOLB. Flat Design vs. Skeuomorphism – Effects on Learnability and Image Attributions in Digital Product Interfaces. In: *Proceedings of the 16th International conference on Engineering and Product Design, Education and Human Technology Relations 2014*. University of Twente, Netherlands, 2014, s. 402-407. ISBN 978-1-904670-56-8

[24] TURNER, David, WILHELM, Robert and Werner LEMBERG. *FreeType Project* [software]. [cit. 2019-05-15] Dostupné z: https://www.freetype.org/

A Příloha - Knihovna RealBox

V přiloženém archivu (elektronická verze) popřípadě na přiloženém CD (tištená verze) se nachází adresář „Zdrojové Soubory“, který obsahuje plný zdrojový kód knihovny — tedy hlavičkové a zdrojové soubory, včetně souborů projektu Visual Studio 2017.

Také obsahuje knihovny třetích stran (složka „Zdrojové Soubory/ ThirdParties“,), vždy s přiloženou právní licencí buď v samostatném souboru nebo jako součástí hlavičkových souborů. Jedná se o licence MIT a public domain. Taktéž je ve stejném adresáři přiložen vývojářský balík VST3 SDK verze 3-6-10, na který se vztahuje GPL licence verze 3.

Zdrojové soubory s jejich závislostmi na knihovnách třetích stran lze zkompilovat do statické knihovny a do tří dynamických knihoven s příponou `.vst` obsahující tři ukázkové zásuvné moduly. Bude možná nuté změnit verzi Windows SDK a Platform Toolset projektů RealBox, NormalKnobs, RealStomp a ReproSpace podle softwarových možností.

Zásuvné moduly se vytvoří ve složce `C:/Program Files (x86)/Common Files/VST3/Filip Dufka/`, kde k nim bude mít většina DAW aplikací přístup.

V archivu se také nachází kompletní dokumentace knihovny v souboru `dokumentace.html`. Obsahuje popis pro vešekeré třídy knihovny s popsanými metodami a vnitřními proměnnými.