Možnosti využití SW nástroje Processing

Lukáš Kopl

Bakalářská práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Lukáš Kopl**
Osobní číslo:	**A22509**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Kombinovaná**
Téma práce:	**Možnosti využití SW nástroje Processing**
Téma práce anglicky:	**Possibilities of Using the Processing Software Tool**

Zásady pro vypracování

1. Zpracujte literární rešerši na oblast multimédií a vizualizace procesů s využitím volně dostupných SW nástrojů.
2. Popište vývojové prostředí SW nástroje Processing, použitý programovací jazyk a vybrané příkazy ze zadaných oblastí, včetně ukázek kódu a jeho výstup.
3. Daný SW nástroj použijte pro zadané oblasti multimédií a také v oblasti vizualizace procesů s propojením na vybranou embedded platformu, přičemž výsledky demonstrujte na ukázkových příkladech.
4. Navrhněte formu prezentace a zpracování získaných a vytvořených podkladů a proveďte její realizaci.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. ŽÁRA, Jiří. *Moderní počítačová grafika*. 2., přeprac. a rozš. vyd. Brno: Computer Press, 2004. ISBN 8025104540.
2. VODA, Zbyšek. *Průvodce světem Arduina*. Vydání druhé. Bučovice: Martin Stříž, 2017. ISBN 978-80-87106-93-8.
3. CHRISTENSEN, Mads G. *Introduction to audio processing*. New York, NY: Springer Science+Business Media, 2019. ISBN 978-3-030-11780-1.
4. TEKALP, A. Murat. *Digital video processing*. Upper Saddle River, NJ: Prentice Hall PTR, 1995. ISBN 0-13-190075-7.
5. SHIFFMAN, Daniel. *Learning Processing: a beginner's guide to programming images, animation, and interaction*. Boston: Morgan Kaufmann/Elsevier, 2008. ISBN 978-0-12-373602-4.
6. VANTOMME, Jan. *Processing 2: Creative Programming Cookbook*. Birmingham: Packt Publishing Ltd., 2012. ISBN 978--1-849517-94-2.
7. FRY, Ben, Casey REAS a Dan SHIFFMAN. *Processing* [online]. 2021 [cit. 2023-10-27]. Dostupné z: https://processing.org/.

Vedoucí bakalářské práce: **Ing. Pavel Navrátil, Ph.D.**
Ústav automatizace a řídicí techniky

Datum zadání bakalářské práce: **5. listopadu 2023**
Termín odevzdání bakalářské práce: **13. května 2024**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 5. ledna 2024

Jméno, příjmení: Lukáš Kopl

Název bakalářské práce: Možnosti využití SW nástroje Processing

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 10. 5. 2024

Lukáš Kopl v. r.
podpis studenta

ABSTRAKT

Bakalářská práce „Možnosti využití SW nástroje Processing“ má za cíl rozšířit povědomí o softwarovém nástroji Processing. V první teoretické části jsou popsány nástroje, se kterými se v průběhu práce zachází. Hlavním úkolem této části je podat ucelený návod na práci v daných prostředích a orientaci v jejich uživatelském rozhraní. Druhá teoretická sekce pojednává o principech moderní počítačové grafiky a matematických výpočtech, na kterých jsou založeny. Ve třetí teoretické části jsou vysvětleny pojmy týkající se obrazu a to, jak vzniká. Součástí této části je také objasnění práce s efekty ovlivňujícími obrázky a video. Čtvrtá teoretická kapitola rozšiřuje pojetí o zvuku a digitálním audiu. Jsou zde uvedeny základní termíny pro práci s audiem a jejich následné využití v praxi. Poslední teoretická část je věnována vybraným embedded platformám a systémům. Jsou zde nastíněny informace ohledně jejich stavby, vývoje a možností, kterými jsou dané systémy vybaveny. Praktická část je také rozdělena na pět korespondujících kapitol, ve kterých jsou popsány konkrétní případy související s danými teoretickými částmi. Jedná se o příklady složené z vytvořených skriptů v softwaru Processing či softwaru k daným embedded platformám. Součástí příkladů jsou také ukázky kódu, výpisy dat či jejich zobrazení pro uživatele.

Klíčová slova: Processing, Java, počítačová grafika, digitální audio, video, embedded platforma, Arduino

ABSTRACT

The aim of the bachelor thesis „Possibilities of Using the Processing Software tool“ is to increase the awareness of the Processing software tool. The first theoretical part describes the tools that are handled during the work. The main purpose of this part is to give a comprehensive guide to working in the given environments and orientation in their user interface. The second theoretical section discusses the principles of modern computer graphics and the mathematical calculations on which they are based. The third theoretical part explains the concepts related to images and how they are created. This section also includes an explanation of how to work with effects affecting images and video. The fourth theoretical chapter expands on the concepts of sound and digital audio. Basic terms for working with audio and their subsequent use in practice are introduced. The last theoretical

part is devoted to selected embedded platforms and systems. Information on their construction, development and the capabilities of these systems is outlined. The practical part is also divided into five corresponding chapters in which specific cases related to the theoretical parts are described. These are examples composed of scripts created in Processing software or software for the embedded platforms in question. The examples also include code samples, data dumps or their display for the user.

Keywords: Processing, Java, computer graphics, digital audio, video, embedded platform, Arduino

V této sekci bych rád poděkoval Ing. Pavlu Navrátilovi, Ph.D. za rady, pomoc a čas, který si vyčlenil a věnoval mi při tvorbě této bakalářské práce.

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Tato bakalářská práce se zabývá průzkumem možností práce s aplikací Processing. Během tvorby podkladů byly využity další programy, které dokáží s aplikací Processing interagovat, či jinak rozšířit práci.

V současné digitální éře, kdy se technologie a umění neustále ovlivňují, je tvorba audiovizuálního obsahu stále populárnější a dostupnější. Velkou zásluhu na tomto pokroku mají nově vzniklé inovativní nástroje. Jedním z těchto nástrojů je mimo jiné i programovací jazyk a vývojové prostředí Processing.

Cílem této bakalářské práce je prozkoumat a analyzovat využití aplikace Processing jako audiovizuálního nástroje. Zkoumáním jeho schopností lze dosáhnout dalšího pokroku v oblasti interaktivního umění a designu.

Bakalářská práce je rozdělena na část teoretickou a praktickou, které jsou dále rozděleny na několik dalších kapitol. Prvních pět kapitol je věnováno teoretické části a přiblížení kontextu práce čtenáři. Dalších pět kapitol je zaměřeno na předání výsledků praktické části práce.

V první kapitole jsou stručně představeny zvolené nástroje a jejich rozhraní. Druhá kapitola slouží k položení matematických základů, na kterých jsou mnohé principy z grafických a zvukových oblastí založeny. Další tři teoretické části poté pojednávají o vizuálních a zvukových základech, a dále také popisují zvolené embedded systémy.

Praktická část je také rozdělena na pět kapitol. V první kapitole praktické části jsou uvedeny postupy práce se zvolenými prostředími. Druhá kapitola uvádí příklady spojení s tvorbou a úpravou obrazu. Třetí kapitola využívá nástrojů aplikace Processing pro tvorbu ukázek spojených s audiem. Ve čtvrté kapitole dochází k tvorbě záznamu videa a úpravě vzniklých souborů. Finální kapitola praktické částí propojuje některé již zmíněné ukázky se zvolenými embedded systémy.

Věřím, že tato bakalářská práce přinese nový pohled na využití aplikace Processing jako nástroje pro audiovizuální tvorbu. Zároveň také doufám, že poskytne inspiraci pro umělce a vývojáře, kteří chtějí navázat na již položené základy vizuálního a zvukového myšlení v digitálním prostředí.

I. TEORETICKÁ ČÁST

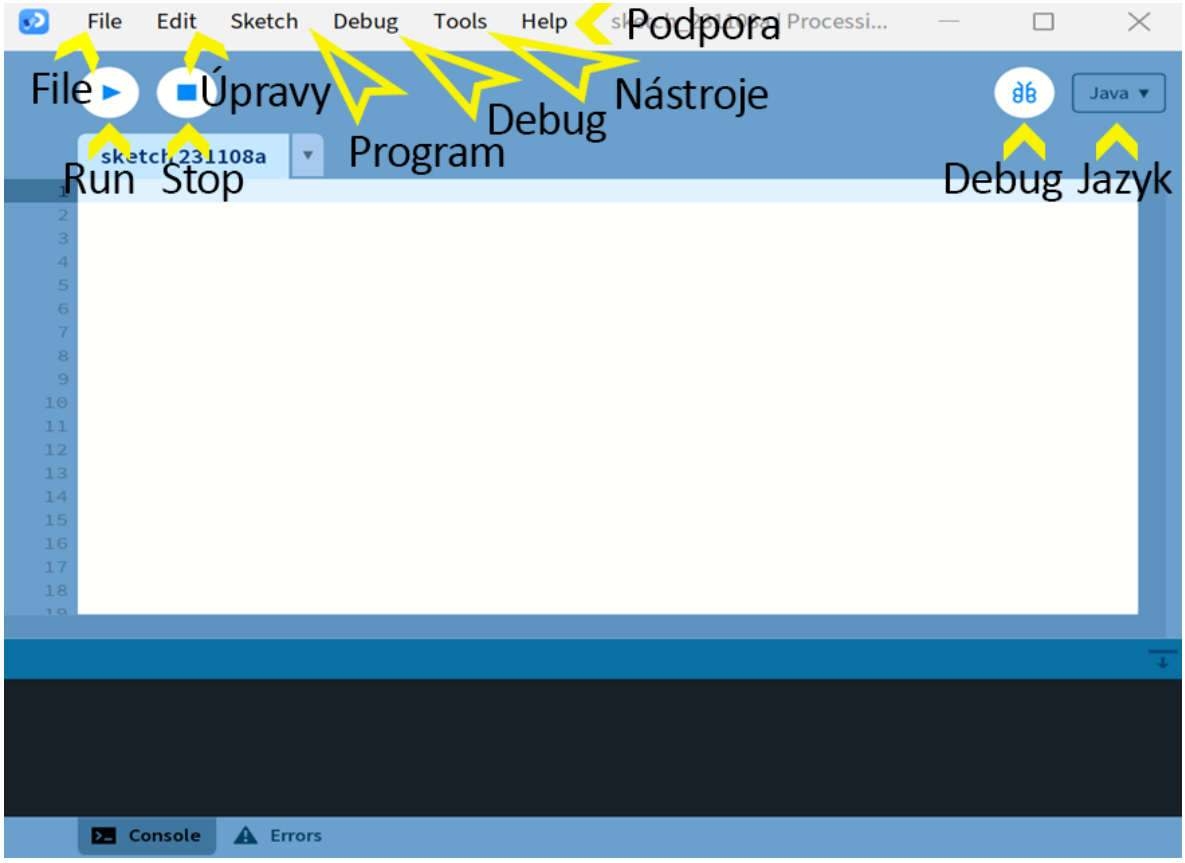
1 NÁSTROJE

V této kapitole budou postupně představeny nástroje, které byly využity při tvorbě této práce. Jsou zde obsaženy informace k základnímu porozumění uživatele ohledně daných softwarů, navigace v jejich rozhraních a také zaznamenán postup jejich instalace.

1.1 Processing

Processing je podle autorů flexibilní softwarový nástroj pro tvorbu skriptů. Účelem tohoto softwaru se stalo mimo jiné také vytvoření jednoduchého stylu či jazyku, který by měl sloužit lidem k porozumění a vyvinutí smyslu pro programování. Dominantním se stal hlavně v odvětvích spojených s vizuálním uměním a gramotností v oblasti vizuálních technologií.

Software disponuje jednoduchým vzhledem a veškeré funkce jsou rozloženy do několika kategorií. Tyto kategorie jsou seskupeny dle jejich funkcí a jsou přehledně seřazeny: [1]



Obrázek 1 Software Processing – rozhraní

- File – menu obsahuje základní funkce práce se soubory. Lze vytvářet nové soubory, otevírat již rozpracované soubory a následně je uložit. Také zde najdeme přístup k již předpřipraveným příkladům kódu k jednotlivým částem softwaru Processing.

- Edit – v této záložce najdeme možnosti pro programy jako např. Zpět a Znovu. Dále jsou zde obsaženy funkce kopírování a hledání v kódu.
- Sketch – nám umožňuje spustit vytvořený program a importovat nové knihovny.
- Debug – v tomto menu můžeme spustit debugger a přidávat body zastavení během průběhu ladění. Můžeme tak snadno odstranit chyby, které mohly v kódu vzniknout.
- Tools – obsahuje dodatečné často používané nástroje jako např. Výběr barvy či Tvorba fontu. Také lze importovat další užitečné nástroje, které nejsou součástí základní verze.
- Help – v této záložce najdeme informace ohledně softwaru Processing a odkazy na případné stránky s tutoriály a pomocnými informacemi.
- Výběr programovacího jazyka (Java v předchozí ukázce) – pomocí tohoto tlačítka můžeme zvolit jazyk programu.

Instalaci softwaru Processing provádíme navštívením příslušných webových stránek *https://processing.org/download*, kde po výběru dané platformy dojde ke stažení souboru s aktuální verzí programu. Instalace se liší pro každou platformu: [2]

- Na platformě Windows dojde ke stažení souboru s příponou *.zip*. Dvojitým kliknutím na daný soubor se nám otevře průzkumník, poté složku, která se nachází uvnitř našeho *.zip* souboru, označíme a přetáhneme na libovolné místo v naší paměti. V dané složce najdeme spustitelný soubor *processing.exe*, který dvojitým kliknutím spustí aplikaci.
- V prostředí Mac OS X použijeme podobný postup. Z webových stránek dojde ke stažení *.zip* souboru. Dvojitým stisknutím myši dojde k otevření daného souboru. V daném souboru označíme ikonu Processing a přetáhneme ji buď do složky aplikací nebo na plochu v závislosti na právech systému uživatele či jeho pohodlí. Pomocí této ikony poté aplikaci spustíme.
- Při instalaci verze softwaru pro Linux stáhneme soubor s příponou *.tar.gz*. Daný soubor by měl být stažen do domovského adresáře, v opačném případě ho do něj přesuneme. Poté otevřeme terminál a zadáme příkaz *tar xvfz processing-xxxx.tgz*. (xxxx jsou zástupné znaky, nahraďte je zbytkem názvu souboru, který je pojmenován po aktuální verzi.) Po rozbalení dojde k vytvoření nové složky s názvem *processing-xxxx*. (dle stažené verze) Nyní už jen přejdeme do daného adresáře pomocí příkazu *cd processing-xxxx* a spustíme aplikaci pomocí *./processing*.

- Software mimo jiné také umožňuje instalaci na minipočítač Raspberry Pi. V průběhu budeme potřebovat paměťovou kartu o velikosti minimálně 8GB. Z webových stránek stáhneme *.zip* soubor, který obsahuje aktuální Raspbian image s předinstalovaným softwarem Processing. Dle správců webových stránek k Raspberry Pi a Processing je také vhodné stáhnout z webových stránek *https://etcher.balena.io* instalační spustitelný soubor, který přidá program Etcher do našeho zařízení. Daný program spustíme, vybereme stažený *.zip* soubor s naším Raspbian image a naši načtenou paměťovou kartu. Pomocí tlačítka Flash! nám Etcher převede image do datové karty. Poté stačí kartu připojit do Raspberry Pi a v aplikacích načteného systému by měl být obsažen software Processing. (Na webových stránkách aplikace lze také stáhnout zabalenou verzi programu, kterou lze stáhnout do Raspberry Pi bez nutnosti reinstalace systému.)

Processing je také vizuální programovací jazyk, který takzvaně umožňuje vytvářet nákresy pomocí kódu. Přesto se nejedná o úplně samostatný programovací jazyk. Byl postaven na platformě Javy, což v praxi znamená, že se kód předpřipraví a převede přímo do kódu v Javě při začátku běhu programu. Třída PApplet z Javy je poté základem všech skriptů v Processingu. [3]

Kód v Processingu se skládá ze dvou hlavních částí *setup* a *draw*. Část setup se provede pouze jednou při spuštění programu. Tato část slouží převážně pro nastavení konfigurací a inicializaci jednotlivých komponentů programu či načítání dat. Druhá část *draw* se provede šedesátkrát za vteřinu. Slouží k provádění operací, výpočtu a kreslení.

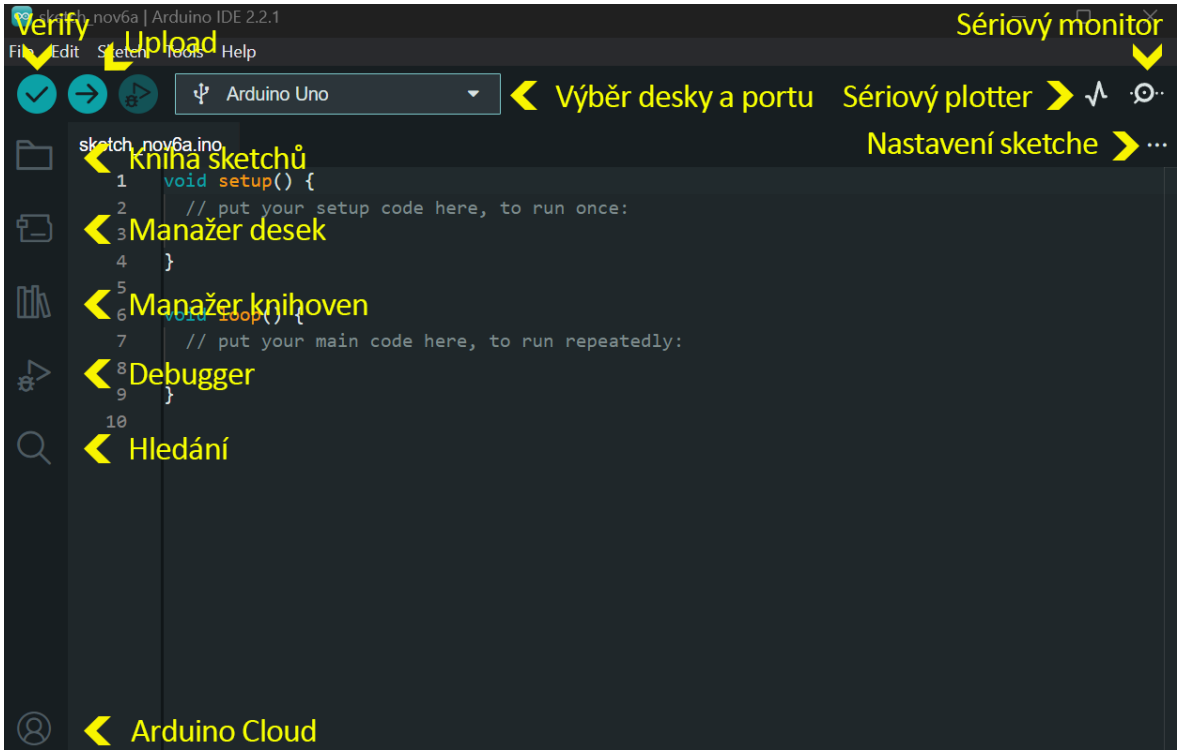
Processing je také postaven na souřadnicovém systému. Je nutné určit polohu každého objektu, který má být vykreslen na obrazovce. Pohybuje se v pixelech a počátek systému je v levém horním rohu. Každý z objektů má také svůj referenční bod, který se mezi objekty může lišit.

Interakce klávesnice a myši jsou v jazyku Processing také realizovatelné. Tyto interakce se implementují pomocí takzvaných eventů. Pokud dojde ke splnění podmínek eventů, provede se připojený kód. Mimo jiné se v prostředí také můžou zavést globální proměnné *mousePressed* a *keyPressed*.

1.2 Arduino IDE

Arduino IDE neboli Arduino Integrated Development Environment obsahuje textový editor pro psaní kódu, místo pro příjem zpráv, konzoli a mnohá další menu s dalšími funkcemi. Hlavním účelem je připojit hardware Arduina různých verzí, načíst na ně programy a udržovat komunikaci mezi zařízením a tímto vývojářským prostředím. V novější verzi tohoto prostředí byly přidány i další pokročilejší funkce včetně debuggingu, automatického doplňování kódu či přístup na online uložiště Arduino Cloud. [4]

Od IDE verze 2 software obsahuje přepracovaný vzhled a prostředí. Hlavním rozdílem je nová postranní lišta, která představuje možnost rychlého přístupu k nejpoužívanějším nástrojům. V následujícím obrázku jsou popsány hlavní body, které umožňují uživateli navigaci v programu.



Obrázek 2 Arduino IDE v. 2.2.1 – rozhraní

- Verify – provede kompilaci kódu a ověří správnost syntaxe.
- Upload – nahraje kód do připojeného a vybraného zařízení.
- Výběr desky a portu – zde se zobrazí detekované desky Arduino včetně jejich portů.
- Kniha sketchů – zde můžete nalézt své lokálně uložené programy. Lze nastavit synchronizaci s Arduino Cloud pro získání programů z online uložiště.

- Manažer desek – umožní průchod instalovatelnými balíčky Arduino a balíčky třetích stran. Některý hardware může vyžadovat instalaci dodatečného balíčku.
- Manažer knihoven – umožní průchod dostupnými knihovnami pro Arduino.
- Debugger – testuje a ladí program v reálném čase.
- Hledání – hledá zadaná klíčová slova v kódu.
- Arduino Cloud – interface pro přihlášení a nastavení Arduino Cloud.
- Sériový monitor – otevře nástroj sériový monitor, jako nové okno konzole.
- Sériový plotter – otevře nástroj sériový plotter, jako nové okno konzole.
- Nastavení sketche – umožňuje rychlou editaci základního nastavení programu.

Software Arduino IDE můžeme snadno nainstalovat stažením souboru příslušné verze z webové stránky *https://www.arduino.cc/en/software*. Další postup se poté liší dle platformy: [5]

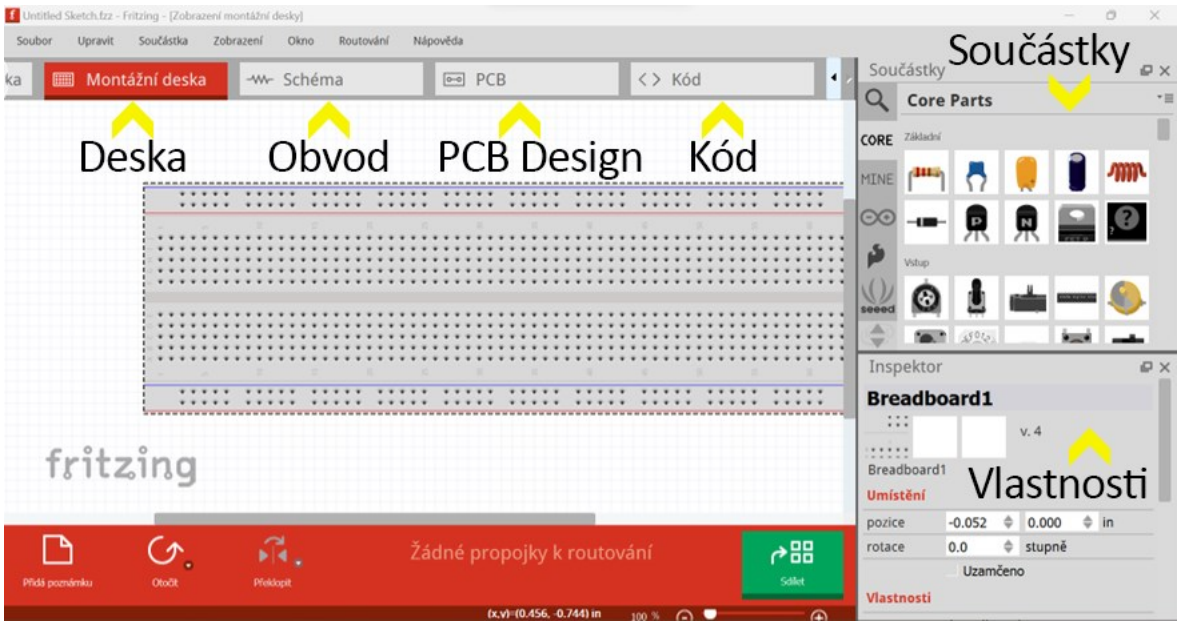
- Pro platformu Windows stáhneme z webových stránek instalační soubor pro danou verzi operačního systému. Stažený soubor dvojitým kliknutím spustíme a následujeme dané instrukce instalačního souboru. Potvrdíme podmínky, vybereme složku pro uložení instalace a potvrdíme tlačítkem *Install*. Poté už jen aplikaci spustíme přes nabídku Start či kliknutím na spustitelný soubor aplikace.
- Na platformě macOS stačí stáhnout soubor aplikace z webových stránek. Následně jej přesuneme do složky aplikací či případně na plochu. Dvojitým stisknutím ikony aplikace ji poté spustíme.
- Pro instalaci na systému Linux stáhneme AppImage soubor z webových stránek. U tohoto souboru změníme ve vlastnostech práva, čímž mu umožníme spustit soubor jako program. Následným dvojitým kliknutím poté soubor spustíme a budeme následovat další instrukce. (Pro otevření souboru AppImage je nutné mít FUSE nainstalovaný v systému.) Po dokončení aplikaci spustíme buď přes terminál či kliknutím na ikonu aplikace.

1.3 Fritzing

Fritzing je open-source software pro automatizaci elektronického designu. Ve zkratce slouží k vytvoření návrhů a náčrtů elektronických systémů jako např. integrované obvody či tištěné

obvodové desky. Obsahuje několik různých oken, ve kterých lze nejen designovat obvody s předem připravenými součástkami, ale také tvorbu vlastních komponentů a implementaci kódu. [6]

Během let nedošlo k velkým změnám ve vzhledu prostředí aplikace Fritzing. Software mimo jiné také osahuje český jazyk jako jednu z možností v nastavení aplikace.



Obrázek 3 Software Fritzing – rozhraní

Rozhraní aplikace lze rozdělil do šesti hlavních kategorií:

- Montážní deska – zde lze vybírat, propojovat a skládat součástky k vytvoření chtěného obvodu.
- Schéma – umožňuje vytvořit a upravit elektronický graf.
- PCB – nástroj pro tvorbu a design obvodových desek.
- Kód – umožňuje psát, modifikovat a načíst kód přímo do komponentů.
- Součástky – menu obsahující části hardwaru, se kterými lze pracovat. Při stažení aplikace obsahuje jen omezený počet součástek. Existuje možnost přidat či vytvořit nové části.
- Vlastnosti – obsahuje dodatečné informace a nastavení zvolené součástky.

Software Fritzing býval ještě do roku 2023 zdarma ke stažení na webových stránkách vývojářů: *https://fritzing.org/download/*. Během posledních pár měsíců autoři zavedli poplatek 8 € ke stažení již zkompilovaných souborů instalátoru. Jednou z možností zůstává

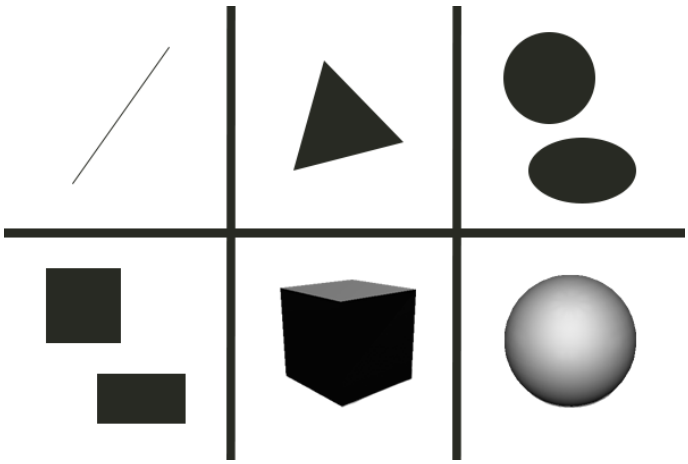
stáhnout starší již zkompilovanou verzi z doby, kdy byl software ještě kompletně zdarma. (např. verze 0.9.3b) Druhou možností je navštívit repozitář aplikace na stránkách GitHub. Zde se nachází celý kód aplikace a mimo jiné také postup jak si aplikaci pomocí kódu sestavit na vlastním zařízení. Na stránce *https://github.com/fritzing/fritzing-app/wiki/1.-Building-Fritzing* můžeme vidět importovaní aplikace do softwaru Qt Creator či vytvoření samostatné spustitelné verze. Po získání zkompilované verze se potom instalace liší dle různých platforem: [7]

- Na platformě Windows dojde ke stažení již vytvořené složky programu nebo instalačního souboru. V prvním případě pak již jen stačí danou složku uložit na vybrané místo v uložišti a danou aplikaci spustit dvojitým kliknutím na spustitelný soubor aplikace. V druhém případě otevřeme příslušný instalační soubor a budeme následovat instrukce. Během tohoto procesu nás daná instalace může požádat o udělení administrátorských práv za účelem přidání balíčku Visual C++ Redistributable od firmy Microsoft. Po dokončení instalace již jen stačí spustit aplikaci dvojitým kliknutím na ikonu aplikace.
- V prostředí macOS dojde ke stažení souboru s koncovkou *.dmg*. Tento soubor přesuneme do složky aplikací. Následně již stačí spustit software pomocí ikony ve složce s aplikacemi.
- Instalace na systému Linux probíhá stažením souboru typu AppImage z webových stránek. U tohoto souboru změníme ve vlastnostech práva, čímž mu umožníme spustit soubor jako program. Následným dvojitým kliknutím poté soubor spustíme a budeme následovat další instrukce. (Pro otevření souboru AppImage je nutné mít FUSE nainstalovaný v systému.) Po dokončení aplikaci spustíme buď přes terminál či kliknutím na ikonu aplikace.

2 MATEMATICKÉ ZÁKLADY

Matematika je nedílnou součástí velké škály oborů. Využití jejích principů je kritickou součástí práce nejen s grafikou, ale také se zvukem a videem. Účelem této kapitoly je přiblížení základů z matematiky, na kterých je řada příkladů založena.

2.1 Geometrické objekty



Obrázek 4 Geometrické objekty

2.1.1 Úsečka

Úsečka je část přímky ohraničena dvěma krajními body. Přímku, podle které je úsečka určena, lze zapsat pomocí rovnice $y = ax + b$. Úsečka se většinou zapisuje pomocí dvou svislých čar jako $|AB|$, kde A a B představují krajní body úsečky s jejich příslušnými souřadnicemi. [8]

2.1.2 Obdélník a čtverec

Obdélník a čtverec řadíme mezi čtyřúhelníky. Obdélník můžeme označit za rovnoběžník, což znamená, že má všechny vnitřní úhly pravé. Obvod tohoto objektu můžeme určit jako $O = 2 \cdot (a + b)$, kde a a b představují délky stran obdélníku. Obsah lze následně vypočítat pomocí vzorce $S = ab$.

Čtverec je poté speciální případ obdélníku, kdy jsou všechny strany stejně dlouhé. Obsah čtverce určíme jako $S = a^2$, kde a značí délku strany čtverce. [9]

2.1.3 Trojúhelník

Trojúhelník je geometrický útvar, který je tvořen ze tří vrcholů a tří stran. Strany jsou značeny pomocí malých písmen a jsou vždy naproti vrcholu s odpovídajícím velkým

písmenem. V trojúhelníku lze určit také výšky stran, které odpovídají vzdálenosti například bodu A od paty kolmice strany a vedené bodem A. Obvod trojúhelníku se určí rovnicí $O = a + b + c$, kde a, b a c značí strany. Obsah lze následně určit jako $\frac{1}{2}av_a$, je ovšem možné použit i jinou stranu a její příslušnou výšku. [10]

2.1.4 Kružnice a elipsa

Kružnice je množina bodů roviny, které mají od stejného středu v bodě $C = (c_1, c_2)$ stejnou vzdálenost r. Daný objekt složený z těchto bodů roviny lze následně popsat pomocí rovnice $\sqrt{(x - c_1)^2 + (y - c_2)^2} = r$.

Elipsa je také množina bodů roviny, ovšem s rozdílem přítomnosti hlavní a vedlejší poloosy. Tyto poloosy značí vzdálenost bodů roviny od středu elipsy a jsou vzájemně kolmé. Daný objekt se středem v bodě $C = (c_1, c_2)$ lze vyjádřit rovnicí $\frac{(x-c_1)^2}{a^2} + \frac{(y-c_2)^2}{b^2} = 1$. [11]

2.1.5 Kvádr a krychle

Za kvádr lze považovat kolmý hranol, který je tvořen obdélníkovou či čtvercovou podstavou. Rozměry kvádru jsou určeny pomocí délek tří hran a, b a c vycházejících ze stejného vrcholu. Stěnové úhlopříčky v navzájem opačných stěnách kvádru mají stejnou délku. Pomocí stěn kvádru lze sestavit síť kvádru, což představuje nákres stran podle příslušných sdílených hran v jedné rovině. Povrch tohoto objektu lze určit rovnicí $S = 2\,(ab + ac + bc)$. Objem lze poté vypočítat jako $V = abc$. [12]

Krychle je speciální případ kvádru, který má všechny hrany stejně dlouhé. Jedná se o pravidelný šestihran. Síť krychle bude složena ze šesti stejných čtverců v jedné rovině. Povrch krychle lze určit jako $S = 6a^2$. Objem lze následně vypočítat rovnicí $V = a^3$.

2.1.6 Koule

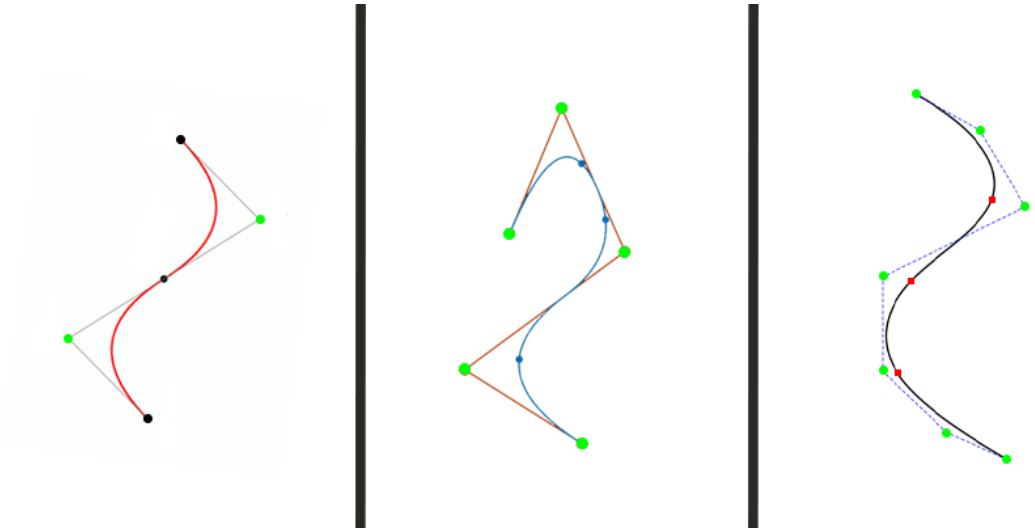
Koule popisuje množinu všech bodů, které mají menší nebo stejně velkou vzdálenost v trojrozměrném prostředí od středu v bodě $C = (c_1, c_2)$. Povrch koule tvoří kulovou plochu. Lze jej určit pomocí rovnice $4\pi r^2$, kde r značí poloměr koule. Objem lze poté vypočítat jako $V = \frac{4}{3}\pi r^2$. [12]

2.2 Křivky

Křivka je z matematického hlediska jednorozměrný objekt, případně také i zobrazení geometrické úsečky do daného matematického prostoru. Jako některé z jednoduchých případů křivky lze uvést například i kružnici či přímku. [14]

Křivku je možné definovat pomocí nějakého matematického prostoru, který definujeme jako M. V daném prostoru lze definovat interval reálných čísel $/$, který určí hranice zobrazení. Pod pojmem křivka k poté rozumíme spojité zobrazení z intervalu$/$ do prostoru M.

V matematice existuje velká řada křivek, které můžeme rozdělit podle dodatečných parametrů. Pokud u křivky derivace existuje v každém bodě, dá se o ní uvažovat jako o hladké či diferenciální. Hladká křivka je poté dále také regulární, pokud v žádném z jejích bodů není výsledná derivace nulová.



Obrázek 5 Křivky

2.2.1 Bézierova křivka

Bézierova křivka je parametrická křivka používaná ve dvoudimenzionálních grafických aplikacích. Křivka se skládá z pole minimálně dvou kontrolních bodů. Dráha, od které se Bézierova křivka odvíjí, začíná v prvním kontrolním bodě a končí v posledním kontrolním bodě, přičemž je řízena polohami mezilehlých kontrolních bodů. Určují se podle řídícího polygonu, který představuje lomenou čáru určenou polohovými vektory kontrolních bodů. [15]

V praxi se Bézierova křivka skládá pomocí De Casteljauova algoritmu. Jedná se o rekurzivní metodu založenou na lineární interpolaci mezi dvěma kontrolními body. Křivku lze rozdělit

na několik segmentů pomocí výpočtu kontrolních bodů mezi dvěma následnými kontrolními body. Mezi těmito body dojde ke vzniku nových kontrolních bodů podle výpočtu z předchozích kontrolních bodů. Zmíněný proces se opakuje dokud nezbývá jediný samostatný bod. [16]

Bézierovu křivku n-tého stupně lze vyjádřit vztahem

$$Q(t) = \sum_{i=0}^{n} p_i \, B_{i,n}(t), \;\; t \in 0..1,$$

kde p_i, i = 0,1,2,…n tvoří řídící body, $B_{i,n}(t) = c_{i,n} t^i (1-t)^{n-i}$ představují Bernsteinovy polynomy n-tého stupně a $c_{i,n} = \binom{n}{i} = \frac{n!}{(n-i)!i!}$ označuje binomické koeficienty. [17]

2.2.2 B-spline křivka

B-spline křivka popisuje křivku volného tvaru složenou z úseků Bézierových křivek stejného stupně. Tyto křivky jsou na sebe napojeny s nejvyšší možnou třídou spojitosti. Výhodou těchto křivek oproti zmíněným Bézierovým křivkám je lepší kopírování řídícího polygonu podle zvoleného stupně B-spline křivky. Jedná se o zjednodušení konceptu Bézierových křivek, pomocí kterého dojde k odstranění některých nevýhod Bézierových křivek. Mezi tyto nevýhody lze zařadit například možnost limitování lokální úpravy křivky podle změny polohy řídícího bodu či nevzdalování křivky od řídícího polygonu ani s počtem rostoucích řídících bodů. [18]

B-spline křivka je určena počtem kontrolních bodů, stupněm p (stupeň polynomů v parametrizaci) a uzlovým vektorem, který určuje napojení jednotlivých oblouků křivky.

Uvažujme vektor $T = \{t_0, t_1, \dots, \; t_m\}$, který bude představovat uzlový vektor. Vektor bude popisovat rostoucí sekvenci pro $t_i \in [0, 1]$. Dále také definujeme kontrolní body $P_0, \dots, P_n$. Stupeň křivky bude odpovídat rovnici $p = m - n - 1$. Pomocí bázových funkcí $N_{i,0}(t)$ a $N_{i,j}(t)$, kde $j = 1, 2, \dots, p$, poté vytvoříme rovnici křivky jako

$$C(t) = \sum_{i=0}^{n} P_i N_{i,p}(t).$$

2.2.3 NURBS

NURBS křivka umožňuje popsat řadu úhlů a zahnutí, mimo jiné také i celou kružnici. Určuje se pomocí řídících bodů, váhami daných řídících bodů, stupněm a také uzlovým vektorem. NURBS je složena z několika menších racionálních Bézierových křivek, což v základu znamená, že každá z těchto křivek je vytvořena na základě racionálně lomených funkcí. [19]

Řídící kontrolní body P_i a jejich váhy w_i, kde $i = 0, 1, \dots, n$ platí rovnice

$$P_i^W = [w_i P_{i,x}, w_i P_{i,y}, w_i],\ i = 0, \dots, n.$$

Tyto body v prostoru a uzlový vektor určují křivku v prostoru o dimenzi vyšším a projekcí dostaneme racionální parametrizaci rovinné NURBS křivky. Pomocí těchto bodů a vektoru parametrizace poté určíme parametrizaci NURBS křivky stupně p jako

$$C(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u) w_i P_i}{\sum_{i=0}^{n} N_{i,p}(u) w_i}.$$

Složením několika křivek v řadě lze také vytvořit NURBS povrch v prostoru.

2.3 Matice

Matici můžeme popsat jako tabulku o n sloupcích a m řádcích. V každé buňce této tabulky se nachází jiné číslo či výraz. Nemusí se jednat pouze o číselné hodnoty. Základní vzhled matice může vypadat například takto: $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$. Matice jsou hojně využívány v matematice pro řešení mnoha problémů, ale najdou své využití i v počítačové grafice či hudbě. [20]

2.3.1 Sčítání a odčítání

Sčítání a odčítaní matic je založeno na stejném principu jako operace u běžných čísel. Pokud uvažujeme dvě matice stejného typu (počet řádků a sloupců) můžeme je jednoduše sečíst či odečíst provedením dané operace u každé dvojice čísel matic na stejné pozici. Jinými slovy můžeme tyto operace zapsat například rovnicí $a_{ij} + b_{ij} = c_{ij}$ či obdobnou verzí s odčítáním.

$$\begin{bmatrix} 1 & 6 \\ 8 & 4 \end{bmatrix} + \begin{bmatrix} 5 & 4 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 6 & 10 \\ 10 & 7 \end{bmatrix}$$

2.3.2 Násobení

V násobení můžeme rozlišit dva způsoby, násobení matice číslem a násobení matice maticí. V prvním případě stačí jednoduše každý prvek matice vynásobit daným číslem a dojde ke vzniku vynásobené matice.

$$5 \begin{bmatrix} 4 & 7 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 20 & 35 \\ 10 & 5 \end{bmatrix}$$

V druhém případě násobení matice maticí jsou dané operace již složitější. Pro možnost násobení dvou matic musí být splněna podmínka, že počet sloupců první matice musí odpovídat počtu řádků druhé matice. Následný součin lze zapsat jako rovnice

$$(A \times B)_{ij} = \sum_{p=1}^{n} a_{ip} b_{pj}.$$

$$\begin{bmatrix} 2 & 3 \\ 3 & 2 \end{bmatrix} \begin{bmatrix} 1 & 5 \\ 8 & 4 \end{bmatrix} = \begin{bmatrix} 26 & 22 \\ 19 & 23 \end{bmatrix}$$

2.3.3 Inverze matice

Inverzní matice lze spočítat pouze na čtvercové matici, na obdélníkové matici není definována. Daná inverzní matice také existuje pouze, pokud je matice regulární. Uvažujme o transformované matici, pro určení transformace, která by původní transformovanou matici vrátila do počátečního stavu lze využít inverzní matici. [21]

V praxi to poté znamená, že pokud došlo k transformaci matice A do podoby B použitím matice S, lze použitím inverzní matice S^{-1} získat původní podobu matice A.

$$B = AS$$

$$A = BS^{-1}$$

3 OBRAZ A VIDEO

V této teoretické kapitole budou popsány principy práce s obrazem a termíny, které se při těchto operacích používají, případně také termíny týkající se objektů, které budou v praktické části vytvářeny.

3.1 Grafické operace

Obrázek 6 Grafické operace

3.1.1 Rozostření

Rozostření je proces, při kterém dojde ke zmenšení ostrosti obrazu a snížení úrovně detailů. Mezi nejběžnější použití rozostření patří odstranění šumu z obrazu či získání podrobnější části obrazu a snížení ostrosti okolního obrazu. [22]

V praxi můžeme obraz rozostřit použitím low-pass filtrů. Tyto filtry dosahují rozostření pomocí snížení odlišností mezi okolními pixely díky zprůměrování pixelů. Můžeme se setkat s několika druhy low-pass filtrů. Z nejběžnějších filtrů lze zmínit Mean a Gaussův filtr.

Mean filtr nahrazuje hodnotu každého pixelu průměrnou hodnotou jeho sousedních pixelů (včetně pixelu samotného). Gaussův filtr pracuje na principu dvoudimenzionální distribuce dle funkce bodového rozšíření. Matice, která se užívá pro tvorbu Gaussova filtru, může mít podobu:

$$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}.$$

3.1.2 Konvoluce

Konvoluci lze chápat jako matematickou operaci dvou funkcí, z nichž o jedné lze hovořit jako o zpracovávané funkci a o druhé jako o jádru. Výstupem konvoluce je odlišná funkce, kterou můžeme chápat jako upravenou původní funkci. Často se využívá u algoritmů, které zpracovávají dvourozměrný digitální obraz. Konvoluci lze vyjádřit pomocí rovnice:

$$f(x,y) = h(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} f(x-i, y-j)h(i,j).$$

Pokud pracujeme s diskrétní konvolucí, zadané jádro představuje tabulku (konvoluční masku). Daná maska je poté umístěna na zvolené místo v obrazu. Každý pixel, který tabulka překryla, je poté vynásoben příslušným koeficientem zadaným v konvoluční masce a následně je proveden součet daných hodnot. Výsledkem tohoto algoritmu vznikne nový pixel. [23]

Koeficienty uvedené v tabulce značí vliv hodnoty pixelu pod nimi. Maska se časem pohybuje po obraze a postupně dojde k nahrazení hodnot obrazu novými hodnotami součinů koeficientů masky s hodnotami obrazu. Pokud je konvoluce použita na celý obraz, dojde k rozostření obrazu.

3.1.3 Detekce hran

Detekce hran je technika zpracování obrazu používaná pro identifikaci bodů v digitálním obrazu s nespojitostmi, jednoduše řečeno, významnými změnami v jasu obrazu. Tyto body, ve kterých se hodnoty jasu vysoce liší, se nazývají hrany obrazu. V praxi se můžeme setkat s velkým množstvím metod detekce hran. Z nejpoužívanějších lze uvést metody Prewitt, Sobel, Laplacian a Canny. [24]

Metoda Prewitt pracuje na principu detekce horizontálních a vertikálních hran v obraze. Metoda Sobel hledá hrany v centru obrazu a napomáhá k lepšímu odstranění šumu v obraze. Metoda Laplacian je založena na matematických druhých derivacích. Metoda Canny poté probíhá v několika krocích a hledá hrany podle mnoha různých kritérií.

3.1.4 Histogram

Jedním z důležitých klíčů charakterizace obrazu je histogram. Jedná se o kvantifikaci množství a frekvencí barev obsažených v obrazu. Hodnota histogramu pro index i poté značí kolik pixelů v obraze má intenzitu i. Počet vektorů, ze kterých je histogram složen, záleží na původním obrazu. [25]

Vzhledem k statistické povaze histogramu jej lze vnímat i jako pravděpodobnost výskytu pixelů barvy v obraze. Histogram kvantifikuje poměry jasu v obraze, ovšem již nenese informace o jejich rozložení v ploše.

Histogram je jednou z významnějších pomůcek digitálních fotografií. Pomocí histogramů lze vyčíst, jak dobře technicky byl obraz zachycen. Mezi technicky dobré obrazy řadíme obrazy, které používají řadu intenzit.

Mezi operace s histogramy patří manipulace s barevnými složkami obrazu, dále také úpravy jasu a změny kontrastu.

3.2 Video

V dnešní době téměř většina digitálních systémů videa používá komponentní reprezentaci barev. Velká řada barevných videokamer poskytuje výstupy RGB, které jsou samostatně digitalizovány. Komponentní reprezentace zabraňuje vzniku chyb, které by mohly vzniknout při kompozitním kódování, za předpokladu, že vstupní signál RGB nebyl předtím již kompozitně kódován. V digitálním videu není zapotřebí žádných vměšování prázdných prostorů či synchronizačních pulzů, protože počítač ví, kolik pixelů je určeno na jeden řádek a kde začíná a končí. Proto jsou tyto prostory a pulzy při převodu analogového videa do digitálního odstraněny. [26]

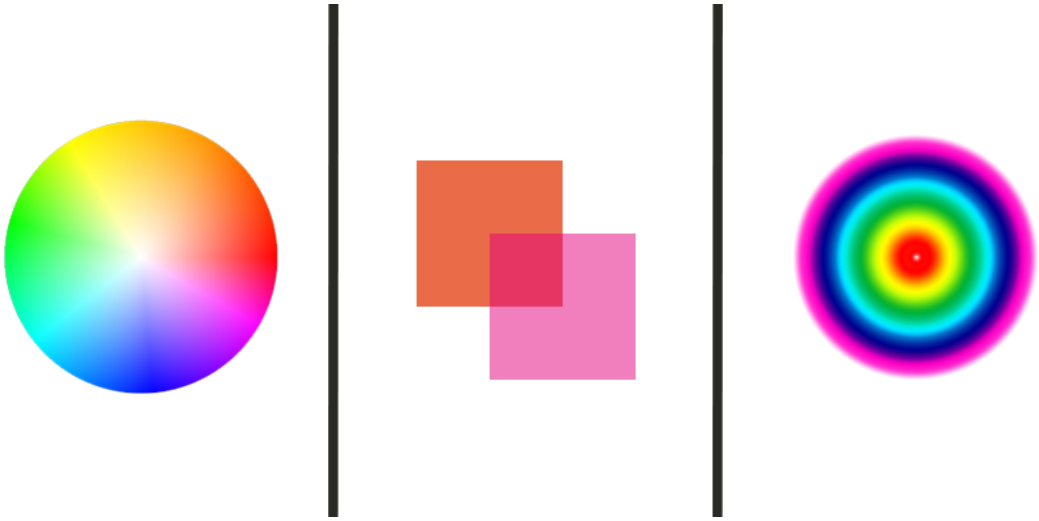
I když je vstupním videem kompozitní analogový signál, např. z videokazety, dochází nejprve k převodu na komponentní analogové video, které se následně individuálně digitalizuje. Také je možné tyto kompozitní signály digitalizovat pomocí převodníku s dostatečně vysokým taktem, aby došlo k zachování přirozeného vnímání barev videa.

Horizontální a vertikální rozlišení digitálního videa souvisí s počtem obrazových bodů na řádek a počtu řádků na snímek. Rozdíly mezi chybami u digitálního videa a analogového videa se velmi liší. V analogovém videu dojde k rozostření obrazu v příslušném směru. U digitálního videa dochází k pixelizaci v důsledku nedostatku rozlišení. Velikost daných chyb záleží na velikosti displeje a vzdálenosti sledování.

Jedním z největších problému s digitalizací videa byl v minulosti nedostatek místa a přenosové rychlosti. Díky moderním technologiím se tyto překážky povedlo překonat a v dnešní době je video celosvětově dostupné a rozšířené. Během let poté došlo k rapidnímu posunu v oblasti videa a standardů záznamu obrazu.

3.3 Barvy

Barva je založena na fyzikální veličině vlnové délky světla. V závislosti na vlnové délce lze poté určit viditelné spektrum barev a kategorizovat barvy do příslušných skupin. Barva dané látky je potom vlastnost, pomocí které dokáže látka odrážet světlo určitých vlnových délek. Vnímaní barvy je mnohdy ovlivněno velkou řadou faktorů. Existuje velká řada reprezentací barvy, mezi které se řadí například kolo barev či duhový model. [27]



Obrázek 7 Barvy

3.3.1 Odstín

Při použití reprezentace barev pomocí kola barev lze odstín vyjádřit jako úhel ve stupních od 0 do 360. Samotným odstínem je poté vyjádřena barva ve své čisté podstatě, tudíž neovlivněné světelností či směsi, pomocí které byla namíchána. Odstínem můžeme nazvat běžné barvy, které jsou pojmenovány obecnými názvy, například „červená“ či „modrá“.

3.3.2 Saturace

Saturace barvy, jinak řečeno její čistota, určuje, jak moc se barva odlišuje od šedého základu. Během této komparace nezáleží na světlosti šedé barvy, ale na faktu, jak moc se od zadané šedé barvy požadovaná barva liší.

3.3.3 Světlost

Světlost barvy popisuje vliv světla na vnímání barvy, jak jasná se barva jeví při pohledu oka. Udává se v procentech, přičemž 100 % znamená bílou barvu, tudíž maximální jas. Naopak 0 % značí černou barvu, tedy minimální jas.

3.3.4 Průhlednost

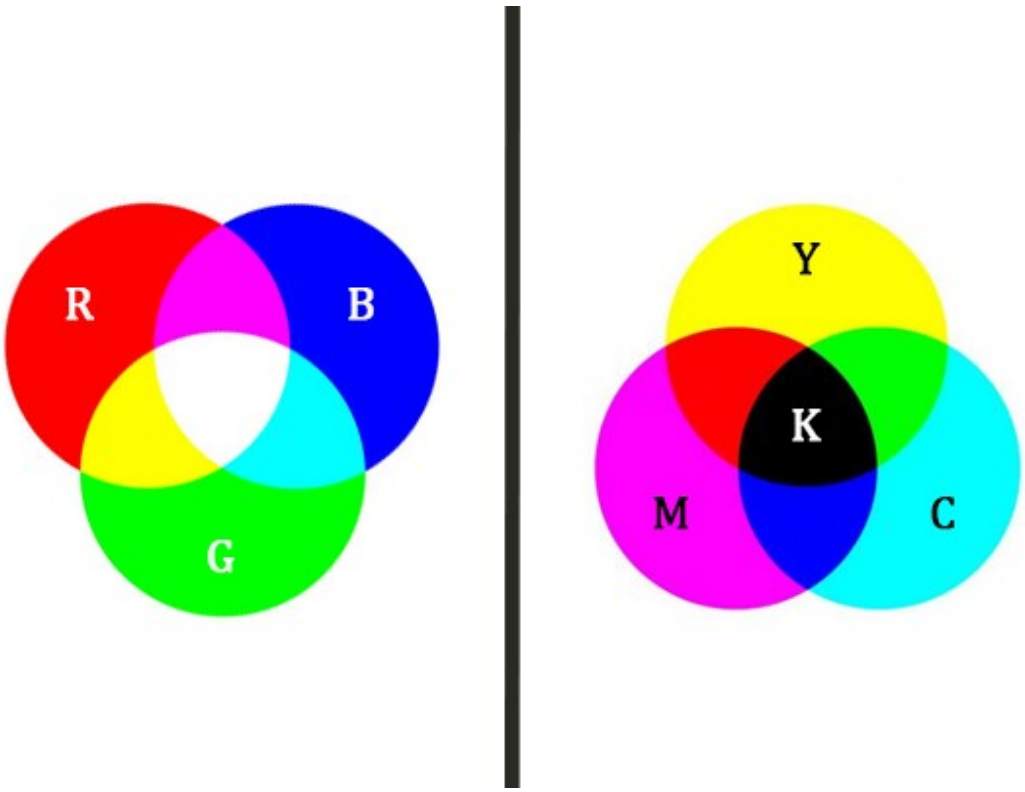
Průhlednost popisuje schopnost pohlcovat záření, nejčastěji světlo. Je určena pomocí jednoduchého čísla, které omezí působení určených druhů světla, což evokuje zmenšení světelnosti a barevné plnosti daného obrazu.

3.3.5 Gradient

Gradient je hladký přechod mezi jedním či několika odstíny jedné nebo více barev. Dosahují lepšího vzhledu a jsou oku více přitažlivé. V praxi se lze setkat s několika hlavními druhy gradientů jako například lineární gradient či radiální gradient. [28]

3.4 Barevné modely

Barvy užívané v počítačové grafice se z důvodu přehlednosti spojují do skupin několika základních barev. Tyto skupiny a kombinace lze poté označit za barevný model. Paleta základních barev se nemění, ovšem každý z modelů je popisuje odlišně. Model poté slouží k přiblížení způsobu, jak spojit dané základní barvy za účelem reálnosti. V současnosti se používá několik modelů, z nichž nejznámější jsou například RGB, CMYK či HSL. [29]



Obrázek 8 Barevné modely

3.4.1 RGB

Asi nejrozšířenějším modelem v grafice je model RGB. Mezi základní barvy tohoto modelu se řadí červená (red), zelená (green) a modrá (blue). Hlavní vlastností tohoto modelu je sčítání základních barev při jejich kombinování. Jedná se tedy o aditivní model. Vzhledem k tomu, že při míchání barev dochází ke zvětšování intenzity, je výsledná barva bílá. [30]

Barvy v tomto modelu se dají popsat v desítkové soustavě čísly od nuly do 255 nebo v šestnáctkové soustavě jako čísla od nuly do FF.

3.4.2 CMYK

Model CMYK je opakem modelu RGB. Jedná se o subtraktivní model. V daném modelu se tedy kombinované barvy odčítají. Po smíchání všech barev dojde ke vzniku černé barvy. Mezi základní barvy tohoto modelu patří azurová (cyan), purpurová (magenta) a žlutá (yellow). Vzhledem k častému využití tohoto modelu při tisku se také přidává kontrolní složka černé barvy (klíčová, key), která slouží k úspoře tonerů tiskáren.

3.5 Rastrová a vektorová grafika

Pro definování rastrové a vektorové grafiky na fundamentální úrovni je nutné znát některé základní pojmy. [31]

Pixel v počítačové grafice značí fyzický bod v obrazu. Jednoduše řečeno se jedná o nejmenší adresovatelný element obrázku reprezentovaném na obrazovce. Většina obrázků, které vidíme na obrazovce je zpracována pomocí rastrové grafiky. V tomto případě je obrázek vytvořen kolekcí pixelů, o které lze hovořit jako o bitmapě.

Bitmapa v počítačové grafice znamená mapování z nějaké domény na bity, tj. hodnoty, které jsou nula nebo jedna. Obecnější pojem pixmapa poté popisuje mapu pixelů, kde každý z bodů dokáže ukládat více než dvě barvy, tedy používat více než jeden bit na pixel.

Rastrové obrázky používají bitmapy jako uložiště informací. V praxi to také znamená, že větší soubory potřebují větší bitmapy pro uložení informací. K snížení místa na disku, které takovéto soubory potřebují, byly vyvinuty mnohé metody komprese. Obrázky ve formátu JPEG či GIF jsou nejznámější zkomprimované soubory. Změna měřítka u těchto souborů je jednoduchá, ale při zvětšení bitmapy dojde k částečnému rozostření obrázku. V těchto případech se poté volí použití vektorové grafiky.

Vektorová grafika využívá sekvenčních příkazů a matematických principů, které vytvářejí tvary a linie ve dvoudimenzionálním a trojdimenzionálním prostoru. Výsledný vektor složený z matematických křivek bude ostrý i po zvětšení či zmenšení. Finální soubor je složen ze skupiny příkazů, které popisují série bodů a jejich polohu. Výsledný soubor poté bude velikostně menší.

3.6 Komprese

Komprese obrazu je proces, při kterém dojde ke snížení velikosti grafického souboru bez snížení kvality obrazu pod přípustnou hranici. Pomocí komprese lze na místo na disku uložit větší množství obrázků a videí. Mezi hlavní druhy komprese lze zařadit ztrátovou a bezztrátovou kompresi. [32]

Ztrátová komprese redukuje velikost souboru permanentním odstraněním méně důležitých informací, například redundantních dat. Ztrátová komprese může výrazně snížit velikost souboru, ovšem může také snížit kvalitu obrazu, pokud dojde k přílišné kompresi. Ztrátová komprese je nevratná operace. Po jejím aplikování již nelze vrátit data do původního stavu. Používá se v případech, kdy je určitá ztráta kvality obrazu tolerovatelná.

Bezztrátová komprese snižuje velikost souboru bez odstranění některých nedůležitých informací. Výsledný soubor bude menší, ale zdaleka nedosahuje takového zmenšení jako ztrátová komprese. Tato operace je vratná, tudíž lze data vždy vrátit do původního stavu. Používá se v případech, kdy je kvalita obrazu důležitější než velikost dat na disku.

3.7 Grafické formáty

3.7.1 Rastrové formáty

BMP

Bitmap Image File je formát vyvinutý firmou Microsoft pro Windows. Formát neumožňuje žádnou kompresi, čímž si zachová vysokou kvalitu, ale také velikost dat. [33]

TIFF

Tagged Image File Format označuje bezztrátové obrazové soubory, které umožňují vysokou kvalitu obrazu i při menším snížení velikosti.

GIF

Graphics Interchange Format je aplikován u souborů často používaných na webu. Jsou limitované na 256 barev, umožňují průhlednost a animace.

JPEG

Joint Photographic Experts Group je ztrátový formát, což znamená, že byl obraz zmenšený, aby zabíral méně místa. Často se využívá na internetu a jedná se o populární formát pro digitální kamery.

PNG

Portable Network Graphics je bezztrátový formát. Podporuje až 16 milionů barev a v mnoha případech nahradily GIF.

3.7.2 Vektorové formáty

SVG

Scalable Vector Graphics je nejběžnější vektorový formát. Jedná se o oficiální formát na webu, který disponuje malou velikostí souboru a všemi výhodami vektorové grafiky. [34]

EPS

Encapsulated PostScript soubory lze označit za předchůdce SVG. Adobe vytvořilo formát EPS v době, kdy bylo nutné zachovat kvalitu pro umělecké účely. V dnešní době se jedná o zastaralejší formát.

PDF

Portable Document Format popisuje standartní formát většiny dokumentů a grafiky. Používá se pro finální verze dokumentů a obrazu, kdy je již lehce nelze editovat.

AI

Adobe Illustrator je formát specifický pro Adobe Illustrator. Jeho účelem je plně nahradit starší EPS soubory. Podpora těchto souborů platí pouze v prostředí Adobe aplikací, ale s omezenou funkcionalitou je lze otevřít i v podobných aplikacích.

3.7.3 Formáty videa

MP4

MPEG-4, známý jako MP4, je nejpoužívanější formát videí. Jedná se o standartní typ pro použití na webu díky zachování kvality i malé velikosti dat. Nevýhodou je složitý proces kódování. [35]

MOV

MOV je formát vytvořený firmou Apple pro podporu Quicktime přehrávače a slouží převážně pro editování videí. Umožňuje vysokou kvalitu obrazu za cenu velké velikosti souborů.

AVI

Audio Video Interleave je formát vytvořený firmou Microsoft. Používá se pro tvorbu videí a je také využíván pro některé nenáročné televizní aplikace.

WMV

Windows Media Video je formát vyvinutý firmou Microsoft pro použití v operačním systému Windows. Většinou není podporován v základním nastavení na zařízeních od firem Apple a Linux.

4 ZVUK

4.1 Základní terminologie

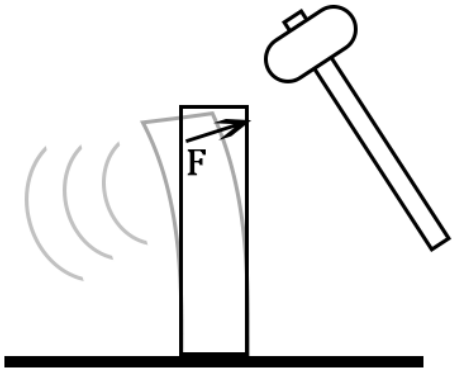
Zvuk je nedílnou součástí našich životů. Nejlépe lze definovat jako sérii vln, které vznikají v určitém bodě a skrze nějaké médium přejdou na jiné místo, kde je lze slyšet či změřit. Zmíněné vlny můžou procházet různými typy materiálů a mají vliv na okolní prostředí. [36]

4.1.1 Základní vibrace

Předtím než lze jakkoliv upravovat zvukový signál, je třeba pochopit, jak zvuk a audio funguje. K získání základního porozumění je vhodné užití příkladů založených na pohybu tyče či bloku materiálu. Vibrující bloky jsou používány k tvorbě hudebních not v několika nástrojích jako např. zvonkohra a xylofon. Lze předpokládat, že při úderu kladivem na jeden z konců bloku dojde k ohybu, který bude popsán pomocí funkce $x(t)$ v čase t. Vzhledem k tvrdosti bloku dojde ke vzniku návratové síly F. Protože síla F vytváří efekt pružiny, lze říci, že je proporcionální k ohybu bloku. Tento jev může být zapsán do rovnice

$$F = -kx(t),$$

kde k je fyzikální konstanta popisující vlastnosti materiálu, ze kterého je blok vyroben.

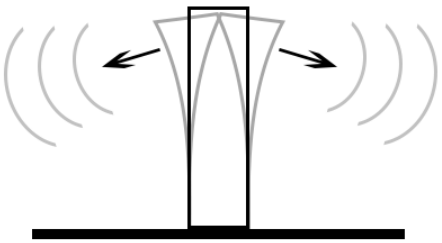


Obrázek 9 Znázornění úderu kladiva na blok [36]

Použitím druhého Newtonova zákona nahradíme sílu F za ma, kde m je hmotnost tělesa a a popisuje vektor zrychlení. Dosazením zápisu zrychlení a následným přesunem proměnné m na druhou stranu rovnice dojdeme k rovnici

$$\frac{d^2x(t)}{dt^2} = -\frac{k}{m}x(t).$$

Předešlé rovnici lze porozumět tak, že vzhledem k fyzikálním zákonům existuje spojitost mezi ohybem $x(t)$, fyzikální konstantou k a hmotností m. Rovnice $x(t)$, která by odpovídala těmto restrikcím by byla tvořena sinusoidou, tudíž by blok materiálu vibroval.



Obrázek 10 Znázornění vibrací bloku po úderu [36]

V některých případech by řešením mohla být také rovnice ve forma cosinu. Pohyb bloku tam a zpět dle sinusoidy způsobí, že se okolní vzduch rozpohybuje a začne vytvářet vlny, které putují od bloku. V závislostí na velikosti sinusoidy a její frekvenci lze poté danou vlnu slyšet.

4.1.2 Sinusoidy

Funkce, jejíž základní forma odpovídá funkci

$$x(t) = A\cos\ (\Omega t + \Phi)$$

by odpovídala finální funkci v předchozí kapitole a mohla by být řešením. Funkce *sinus* a *cosinus* jsou poté speciální případy této základní funkce, s tím, že při $\Phi = 0$ dojde k získání *cosinusu* a při $\Phi = -\pi/2$ dojde k získání *sinusu*. Tato funkce má také několik základních parametrů.

Frekvence Ω, která je měřena v radiánech za sekundu, určuje počet cyklů sinusoidy za sekundu. Zároveň také určuje růst křivky a má spojitost s frekvencí f v jednotce Hertz jako $\Omega = 2\pi f$.

Amplituda A o velikosti větší než 0 upravuje sinusoidu takovým způsobem, že místo průchodu mezi -1 a 1 křivka prochází mezi -A až A. V podstatě mění, jak hlasitá je sinusoida. Síla křivky je potom určena rovnicí $A^2/2$.

O fázi Φ lze uvažovat jako o posunu sinusoidy v čase. Teoreticky je možné ji zapsat jako $\Phi = (t - t_0)$. Stejný posun v čase přinese jiné výsledky pro jinou frekvenci křivky.

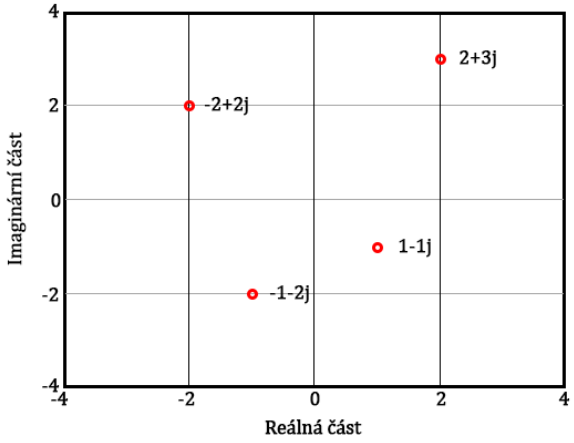
4.1.3 Komplexní čísla

Komplexní číslo je v podstatě dvojdimenzionální reprezentace formy

$$z = a + jb.$$

Číslo je složené ze dvou částí a a b, o kterých lze uvažovat jako o koordinátách grafu. Jsou nazývány kartézské souřadnice a jsou součástí kartézské reprezentace grafů. Proměnná j je takzvaná imaginární jednotka, avšak a nazýváme reálnou částí a b také nazýváme imaginární částí.

Často lze najít dvojdimenzionální koordinační systém, ve kterém jsou na ose *x* reálné části a na ose *y* imaginární části.



Obrázek 11 Graf komplexních čísel [36]

V základu se komplexní čísla chovají podobně jako reálná čísla. Hlavní rozdíl lze pozorovat u zacházení s imaginární jednotkou j. Hlavní rovnice pro práci s touto jednotkou jsou

$$j^2 = -1 \text{ a } j = \sqrt{-1}.$$

Použitím těchto převodů dojde ke ztížení běžně známých operací pro reálná čísla.

4.1.4 Fázor

Fázor je funkce založená na komplexních funkcích a Eulerově rovnici. Použitím Eulerovy rovnice lze zjednodušit operace s komplexními čísli. Z následných rovnic lze poté fázor definovat jako

$$z^t = e^{j\Omega t}.$$

Práce s fázory nám umožňuje oddělit fázi a amplitudu sinusoidy od časově závislé části s frekvencí. Násobením komplexními čísli lze nyní upravit velikost a argumenty komplexní funkce.

Funkce fázorů budou sloužit pro porozumění signálů, filtrů a audio efektů. Operacemi se sinusoidou z nich také můžeme získat funkci *sinus* či *cosinus*.

Operace	Notace	Obdélníková	Exponenciální
Komplexní číslo	z	$a + jb$	$re^{j\phi}$
Velikost ($\lvert\cdot\rvert$)	$\lvert z\rvert$	$\sqrt{a^2 + b^2}$	r
Úhel ($\angle\cdot$)	$\angle z$	$tan^{-1}(\frac{b}{a})$	ϕ
Konjugace (*)	z^*	$a - jb$	$re^{-\phi}$
Sčítání (+)	$z_1 + z_2$	$(a_1 + a_2) + j(b_1 + b_2)$	
Odčítání (-)	$z_1 - z_2$	$(a_1 - a_2) + j(b_1 - b_2)$	
Násobení ($\cdot$)	$z_1 \cdot z_2$		$r_1 r_2 e^{j\psi_1 + j\psi_2}$
Dělení (/)	z_1/z_2		$\frac{r_1}{r_2} e^{j\psi_1 - j\psi_2}$

Obrázek 12 Přehled operací s komplexními čísly [36]

4.2 Digitální audio signály

Jak již bylo zmíněno, zvuk lze vnímat jako vlny procházejí prostředím. Ovšem pro práci s využitím počítačů je tento systém prakticky neproveditelný. Vzhledem k tomu, že počítače mohou uchovat pouze konečný počet čísel, musíme přejít z analogového systému na digitální. Procesy spojené s tímto převodem se nazývají sampling, kvantování a rekonstrukce.

ADC je zařízení, nebo čip, který převádí signály z analogového do digitálního systému pomocí samplingu a kvantování, kdežto DAC převádí signály z digitálního do analogového systému pomocí rekonstrukce.

Tyto signály se převádí hlavně z důvodu flexibilnější práce, které by bez použití počítačů na analogovém hardwaru nebyly možné.

4.2.1 Sampling

Sampling neboli vzorkování je první z metod pro převod analogového signálu. Předpokládejme, že máme časově plynulou funkci $x(t)$ definovanou pro jakékoliv reálné t, která by měla být zpracována v počítači. Proces samplingu spočívá v měření hodnot této funkce v daných časových instancích t_n pro indexy $n = 0, 1, 2, ...$ k získání signálu

$$x_n = x(t_n).$$

Danou funkci poté nazýváme diskrétní funkcí času. Nejjednodušší metoda samplingu je uniform sampling. V tomto procesu měříme hodnoty stejně vzdálených bodů

$$t_n = T_s n, pro\ n = 0, 1, 2, ...,$$

kde T_s je čas v sekundách mezi po sobě jdoucími signály a je nazýván perioda samplingu. Lze pozorovat, že následný digitální signál je v podstatě jen seřazená sekvence čísel. Čím menší je T_s, tím přesněji bude původní signál nasamplován. Počet získaných vzorků za vteřinu lze popsat jako frekvence samplování, která je definována jako

$$f_s = \frac{1}{T_s}.$$

Tato frekvence je měřena v jednotkách Hertz. Většina audio nahrávek je vzorkována při frekvenci 44,1 a více kHz. Další důležitý pojem je digitální frekvence, která popisuje číslo radiánů na vzorek, a je definována rovnicí

$$\omega = 2\pi \frac{f}{f_s}.$$

Během převodu je doporučeno řídit se vzorkovacím teorémem, který je popsán autory jako např. Nyquist, Shannon či Whittaker. Teorém sděluje, že frekvence vzorkování by měla být alespoň dvakrát větší než nevyšší frekvence sinusoidy a lze jej zapsat rovnicí

$$f_s > 2f_{max}.$$

Pokud nebude teorém při vzorkování dodržen, může dojít k fenoménu známém jako aliasing. Nastane ohyb hodnot z vyššího spektra frekvence vzorkování, což vede k jejich objevování v nižší části spektra jako zrcadlení signálů. Tímto se původní signál nenávratně zkreslí.

4.2.2 Kvantování

Jak již bylo zmíněno, počítače pracují pouze s konečným počtem čísel. Tudíž je třeba zmapovat tato nekonečná čísla na systému, se kterým počítače dokáží pracovat. Tento proces se nazývá kvantování.

Pro provedení kvantování je třeba použít kvantizér. Předpokládejme, že měřený signál se nachází v určitém rozpětí mezi $-\alpha$ a α. Pokud máme k dispozici β bitů, lze reprezentovat 2^{β} různých hodnot. Pokud se časový interval rozdělí na stejně veliké části, pak tyto části mají velikost

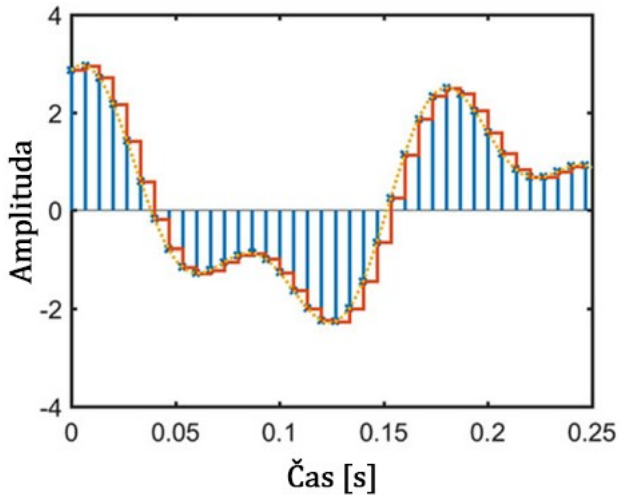
$$\Delta = \frac{\alpha}{2^{\beta-1}}.$$

Tento způsob je označován jako uniform kvantování a jedná se o nejběžnější formu kvantování pro audio. Symbol Δ popisuje vzdálenost kroku. Čím větší je počet bitů, tím nižší je vzdálenost kroku. Běžně se užívá 16, 24 či 32 bitů na vzorek. Je nutné si uvědomit, že pokud je vložený signál větší než specifikované rozpětí kvantování, dojde k useknutí signálu, což povede ke vzniku nepříjemných zvuků.

Během kvantování může dojít k nepřesnostem v převodu zvuku, jinými slovy přidáním různých hluků do požadovaného signálu. Poměr hladiny signálu k poměru hluku je označován jako poměr signálu-hluku (SNR). Čím větší je tento poměr, tím lépe se kvantování zdařilo a chyby ve výsledném signálu budou méně znatelné.

4.2.3 Rekonstrukce

Rekonstrukce je proces znovuvytvoření původní časově plynulé funkce z hodnot vzorkovaného a kvantovaného signálu. Přepokládejme, že byl dodržen vzorkovací teorém. Pomocí postupného posunování každé hodnoty dojde ke vzniku nové funkce, která by z většiny měla odpovídat původní funkci před digitalizací.



Obrázek 13 Znázornění procesu rekonstrukce [36]

4.3 Filtry

Filtry popisují mnohé způsoby práce s již vytvořeným digitálním signálem. V praxi se nejčastěji lze setkat s lineárně založenými filtry. Digitální filtry jsou tvořeny formou rovnic, které popisují časově posunuté, či upravené verze hodnot signálu. Tyto verze jsou škálovány podle reálného čísla, které nazýváme koeficient filtru.

4.3.1 Jednoduchý filtr

Digitální filtr pracuje se sekvencí čísel, která se mu předá. Pomocí předem definované rovnice poté filtr vytvoří novou modifikovanou sekvenci. Většina rovnic obsahuje násobení či přidání hodnot posunutých verzí signálu. Příkladem této rovnice je například

$$y_n = x_n + ax_{n-1},$$

kde y_n představuje modifikovanou hodnotu, x_n představuje původní hodnotu a další část rovnice představuje danou modifikaci.

Obrázek 14 Základní diagram filtru [36]

4.3.2 Analýza filtrů

Analýza filtrů spočívá v rozboru hodnot. Jedním ze základních principů, který je možné provést, je rozbor známé rovnice filtru. Tento způsob je nejjednodušší, protože se opírá o již známé modifikace, které filtr provedl.

Dalším způsobem je porovnání hodnot fázorů vstupu a výstupu filtru. Rozdíl mezi fázorem vstupu a výstupu lze nazvat pojmem přenosová funkce. Matematickými operacemi s touto přenosovou funkcí lze určit frekvence použité během vytvoření signálu či následném vzorkování a kvantování.

Jedna z dále využívaných metod analýzy je vložení jiné hodnoty signálu na vstup. Tato hodnota je často označována za impuls, který má většinu energie koncentrovanou v jedné časové instanci. Tento postup je v matematice pojmenován podle matematika Leopolda Kroneckera. Pokud o této funkci uvažujeme jako o signálu, jedná se o nekonečnou řadu čísel (0 a 1).

4.3.3 Filtry se zpětnou vazbou

V případě filtrů se zpětnou vazbou se pracuje nejen s posunutými verzemi hodnot na vstupu, ale také na výstupu. Tento způsob filtru umožňuje vytvoření složitějších a silnějších systémů filtrování. Často lze tyto filtry najít také pod označením IIR filtr. Jedna ze základních rovnic těchto filtrů může vypadat jako

$$y_n = x_n + ay_{n-1}.$$

4.3.4 Resonanční filtry

Náhlá změna ve frekvenci odpovědí z filtru se nazývá resonance. Účelem resonančního filtru je záměrně vyvolat tyto změny. S využitím filtru se zpětnou vazbou a umístěním hodnot filtru do blízkosti jednotkového kruhu signálu v úhlu odpovídající frekvenci lze vytvořit resonanci. Frekvenci v bodě změny lze nazvat středovou frekvencí a šířku změny lze popsat jako šířku pásma.

4.3.5 Vlastnosti filtrů

Filtry následují řadu vlastností, které lze aplikovat při jejich tvorbě či úpravě. První vlastností je homogenita, která říká, že pokud upravíme původní signál před aplikací filtru určitou hodnotou, výsledný signál bude pozměněn stejně velkou hodnotou.

Další vlastností je aditivnost, která popisuje chování filtrů při jejich sčítání. Pokud existují dva vstupy signálů, dostaneme stejný výsledek nehledě na to, zda je nejdříve sečteme před průchodem filtru nebo provedeme průchod filtrem a následně sečteme.

4.3.6 Druhy filtrů

V současnosti je velké množství filtrů, které se již tvoří v matematických softwarech jako např. MATLAB. Z důvodu lepší orientace lze filtry rozdělit do několika kategorií:

- Low-pass – označuje filtry, které nechají projít frekvence pod stanovenou frekvencí pro uříznutí a upravují či odstraňují frekvence nad požadovanou hranicí.
- High-pass – značí filtry, které nechají projít frekvence nad požadovanou frekvencí a odstraní frekvence pod požadovanou hranicí.
- Bandpass – popisuje filtry, které umožní nastavení rozmezí dvou hranic, kdy následné frekvence v požadovaném rozmezí zachová a zbytek odstraní.

- Stopband – určuje filtry, které jsou opakem Bandpassu. To znamená, že odstraní vše v uživateli zvoleném rozhraní a zachová vše ostatní.

4.4 Fouriérova transformace

Fouriérova transformace je jedna z nejvíce využívaných funkcí při práci s audiem. Pojmenovali ji podle Jeana-Baptista Josefa Fouriera, který položil základ tomu, co dnes známe jako Fouriérova transformace. Tato metoda umožňuje analyzovat a interpretovat složky zvukových signálů.

4.4.1 Transformace

Základní princip transformace spočívá v dekompozici signálu na fázory s jinými frekvencemi, kdy následné amplitudy těchto fázorů reflektují množství energie, která je reprezentována v daném signálu při dané frekvenci. Po užití této metody lze říci, že signál transformujeme z časové do frekvenční domény. Základní rovnici Fouriérovi transformace lze zapsat jako

$$X(\omega) = \sum_{n=0}^{N-1} x_n e^{-j\omega n}.$$

V této rovnici $X(\omega)$ označuje spektrum signálu x_n. Jedná se o vratnou funkci, to znamená, že s použitím inversní transformace lze dojít k původnímu spektru signálu. V případě použití časově plynulé frekvence se do inverzní Fouriérovi transformace přidává integrál.

4.4.2 FFT a Zero-padding

Z rovnice Fouriérovy transformace lze určit, že je nutné využití *N* komplexních násobení a adicí pro každý z *N* bodů frekvencí. Základní transformace tedy potřebuje N^2 operací k výpočtu. Vzhledem k této náročnosti byla vytvořena metoda FFT, založená na principu rozděl a panuj, který rekurzivně rozpracuje rovnice na menší části, čímž dojde ke zmenšení výpočetní náročnosti. Během let došlo k několika implementacím tohoto systému, z nichž se nyní používá Fast Fourier Transform in the West.

Při použití metody lze narazit na problém s lichým počtem vzorků, které je potřeba spočítat pomocí transformace. V tomto případě se použije zero-padding, který za signál přidá jednu či více nul a číslo převede do sudé formy.

4.4.3 Windowing

Windowing je operace, při které dojde k násobení vstupního signálu pomocí takzvané „window“ funkce, kterou označíme například jako w_n.

$$y_n = w_n x_n$$

Pomocí této metody můžeme převést nekonečný signál na řadu pozorovatelných „oken“. Tudíž lze pomocí této metody dosáhnout změny nekonečného signálu na konečný.

4.5 Efekty

Zvukové efekty mají velkou řadu využití v různých odvětvích, ať už ve filmové či hudební produkci. Už jen zvuk elektrické kytary závisí na kombinaci několika efektů, které si hráč určuje řadou potenciometrů a tlačítek. Velká spousta efektů je v základu jednoduchá a velmi podobná. Je nutno dodat, že efekty popsané v této kapitole mohou být vytvořeny různými způsoby, tudíž se nejedná o jediný způsob jejich implementace.

4.5.1 Echo

Echo je jeden z nejjednodušších efektů, který je často nazýván jako efekt zpoždění či ozvěny. Popisuje opakování zvuku v určitém intervalu za zvukem původním. Jednoduchý efekt echo lze implementovat např. jako funkce

$$y_n = x_n + bx_{n-D}.$$

V této rovnici písmeno D označuje časový interval mezi původním zvukem a ozvěnou, písmeno b označuje koeficient filtru, který určuje, jak hlasitá ozvěna bude. V praxi lze také implementovat postupně klesající ozvěnu, která bude přecházet ze slyšitelných do neslyšitelných hladin.

4.5.2 Vibrato

Jedná se o efekt, který lze v jeho jednoduché formě implementovat pomocí periodických variací na výšce tónů užívaných v hudbě. Tento efekt lze replikovat například rovnicí

$$y_n = x_{n-D(n)}.$$

Tento princip lze také označit jako modulace ve frekvencích signálu. Vzhledem k rozdílným variacím frekvencí dojde k vytvoření efektu vibrata.

4.5.3 Tremolo

Efekt tremola je založený na podobném principu jako vibráto, ovšem v případě tremola dochází k modulaci amplitudy čily síly signálu místo jeho frekvence. Časem dochází k vytvoření efektu měnící-se hlasitosti zvuku. Tento efekt lze matematicky zapsat násobením vstupního signálu pomocí takzvaného modulačního signálu.

$$y_n = x_n m_n$$

Ve speciálním případě, který můžeme označit jako prstencová modulace, dojde k prohození signálu, kdy se modulační signál stane nosičem a je modulován vstupním signálem. Výsledkem je specifický vtipný zvuk často používaný ve filmovém průmyslu.

4.5.4 Chorus

Zvukový efekt Chorus imituje zvuky několika zároveň hrajících nástrojů či zpívajících zpěváků. V řadě případu nemusí být zvuky přesně synchronizované a mohou se lišit v hlasitosti. K vytvoření tohoto efektu dojde postupným kopírováním původního signálu s různými zpožděními a také úpravou jejich sil. Z matematického hlediska lze tento efekt popsat například rovnicí

$$y_n = x_n + g x_{n-D(n)}.$$

V této funkci proměnná g značí parametr úprav síly a proměnná D označuje časový posun.

4.5.5 Reverb

Reverb je prostorový efekt, který má zvuku dodat hloubku pomocí adice dozvuku a řady ozvěn založených na zvukovém signálu. Jedná se o konvoluční metodu, při které dojde k artificiálnímu tvoření dozvuku za účelem napodobení přírodních ozvěn. Příkladem tohoto efektu může být například funkce

$$y_n = \sum_{m=0}^{M-1} h_m x_{n-m},$$

kde x_n značí vstupní signál, y_n popisuje výstupní signál, h_m označuje koeficient filtru a uvažujeme, že impulsní odpověď obsahuje M po sobě jdoucích vzorků.

4.5.6 Stereo Panning

Termín popisuje prostorový efekt, při kterém dochází k úpravě prostorového vnímání. Nejčastěji dochází k úpravě amplitudy na některé z rozdělených stop, čímž vznikne efekt orientace v prostoru. Ve většině případů se používá několik reproduktorů v různých úhlech.

5 EMBEDDED SYSTÉMY

Pomocí slovního spojení embedded systém lze popsat systém založený na mikroprocesoru, který je složen z počítačových součástek a obsahuje software určený k výkonu dedikované funkce. Může pracovat samostatně nebo být součástí většího systému. V okolí jádra systému se většinou nachází integrovaný obvod pro výpočet real-time operací.

Jejich složitost se pohybuje od jednoduchého mikroprocesoru až k sérii procesorů s připojenými periferiemi, často i se zobrazením výsledků přes komplexní grafické rozhraní.

5.1 Základní struktura embedded systému

Embedded systém řídí mikropočítač či digitální signálový procesor, integrované obvody pro aplikace a řadu dalších komponentů. Dané procesní systémy jsou poté propojeny s dalšími systémy, které obstarávají elektrické a mechanické propojení. [37]

Instrukce embedded systémů, o kterých lze hovořit jako o firmwaru, jsou uloženy v paměti určené ke pouze čtení jako např. na paměťovém čipu flash. S ostatním světem poté komunikuje přes periférie a propojení vstupních a výstupních zařízení.

5.1.1 Součástky

Základní struktura embedded systému obsahuje tyto součástky:

- Senzor – měří a převádí fyzickou hodnotu kvantity k elektrickému signálu, který poté může být přečten osobou či dalšími elektrickým nástrojem.
- A-D převodník – převádí analogový signál od senzoru do digitálního signálu.
- Procesor a integrované obvody – zpracují data a vypočítají výstup, který uloží v paměti.
- D-A převodník – převádí digitální signál do analogového signálu.
- Aktuátor – porovná výstup od D-A převodníku k uloženému výstupu a uloží schválený výstup.

5.1.2 Historie

První moderní real-time embedded systém byl naváděcí počítač k misi Apollo, vyvinutý v šedesátých letech Dr. Charlesem Stark Draperem na univerzitě v Massachusetts. Měl za účel automaticky sbírat data a provádět výpočty pro řídící a lunární modul.

V roce 1971 vydal Intel první komerčně dostupný mikroprocesor (Intel 4004). Daná jednotka stále vyžadovala podpůrné čipy a externí paměť. V roce 1978 poté Národní asociace strojírenských výrobců (NEMA) vydala standart pro programovatelné mikropočítače, čímž zlepšila jejich design. V osmdesátých letech se již začaly integrovat další součástky jako paměť či systém vstupu a výstupu.

Tyto mikropočítače a systémy se poté velmi rychle staly částí mnoha jiných zařízení, ať už telefonů, či silničních semaforů.

5.2 Arduino

Arduino je open-source elektronická platforma založená na hardwaru a softwaru, která je jednoduchá k použití. Tyto desky jsou schopny číst různé vstupy a přeměnit je na výstup jako například zapnutí monitoru či online publikace. Pomocí instrukcí poslaných do mikrokontroleru lze přesně nastavit, co má daná deska dělat. Instrukce jsou psány podle programovacího jazyka Arduina v přiloženém softwaru Arduino IDE. [38]

Během let se Arduino stalo součástí mnoha projektů od dennodenních objektů až ke komplexním vědeckým nástrojům. Práci s Arduinem se věnují tisíce lidí po celém světě od studentů až po profesionály. Na webu lze najít důkladně propracovanou stránku a knihovnu s velkou řadou informací, jak s Arduinem pracovat.

5.2.1 Historie

Původní model Arduina vznikl již v roce 2005 v italském městě Ivrea jako jednoduchý nástroj pro výuku studentů z tamního Institutu interaktivního designu. Tento výtvor se mezi studenty velmi rychle uchytil, proto se jej jeho tvůrci rozhodli rozšířit po celém světě. K největšímu rozvoji ovšem nedošlo vyšším prodejem desek, ale hlavně šířením návodů a schémat pro práci s tímto zařízením. [39]

Programovací část byla vytvořena na základech položených knihovnou jazyka Java s názvem Processing. Dále byl také přidán i editor s cílem zjednodušit studentům výuku programování. Během let došlo k několika spolupracím této desky s velkými firmami jako např. Intel. Vzhledem k povaze tohoto projektu jakožto open-source také došlo k vytvoření několika větví tohoto projektu pod jinými názvy a tvůrci.

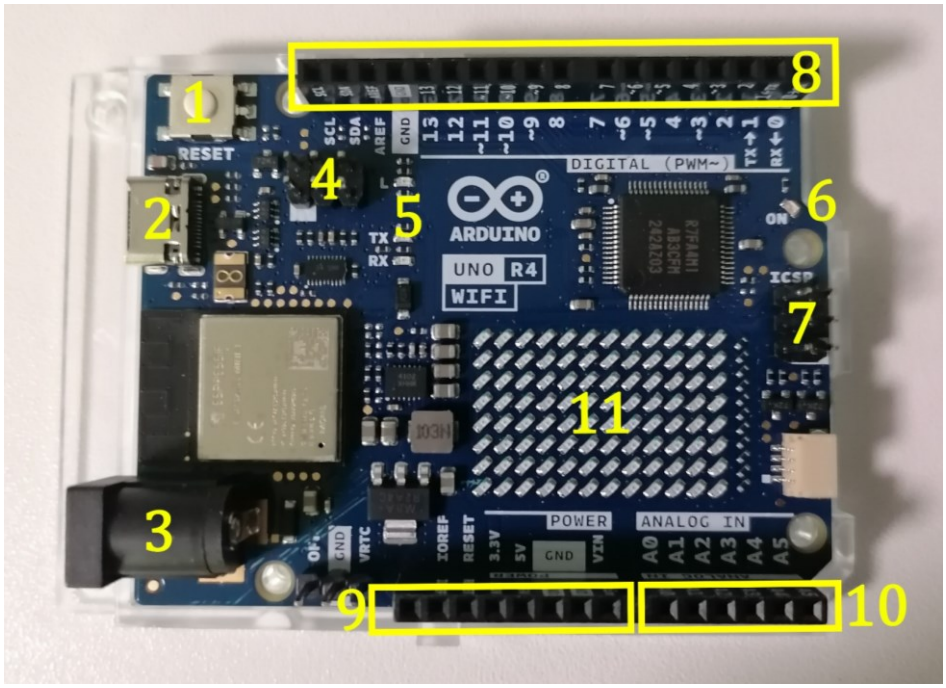
5.2.2 Typy desek

Během let došlo k tvorbě mnoha modelů a revizí již vydaných typů. Základem většiny desek je procesor od firmy Atmel, jenž je doplněn o řadu dalších elektrických komponentů. V elektronických obchodech se můžeme setkat s doplněním názvu jako např. Rev3 či R3, což značí číslo verze této desky. Mezi verzemi mohlo dojít ke menším či větším změnám v komponentech a rozložení dané desky. Mezi typy desek patří např.: Arduino Mini, Arduino Nano, Arduino Uno, Arduino Due, Arduino Tre a mnoho další.

V této práci jsou příklady prováděné na modelu desky Arduino Uno, z daného důvodu tento typ bude také důkladněji popsán. Jedná se o aktuálně nejpoužívanější typ desky. Jednou z lukrativních výhod tohoto modelu je existence USB rozhraní. Příjemné je také obsažení slotu pro SD kartu. Jiná verze modelu může také pracovat s Wi-Fi či Bluetooth přes bezdrátovou komunikaci.

5.2.3 Vybavení desky

Informace popsané v této kapitole úzce závisí na vybraném modelu desky, proto se tato kapitola týká práce s modelem Arduino Uno R4 WiFi. Tento model obsahuje řadu komponentů pro usnadnění práce se zařízením a také velké množství způsobů vstupu a výstupu.



Obrázek 15 Arduino Uno R4 WiFi s popisky

1) Číslo 1 značí resetovací tlačítko. Tlačítko vytvoří stejný efekt jako odpojení od napájení. Deska počká na nové instrukce a poté je začne vykonávat.
2) USB konektor typu C. Slouží pro připojení desky za účely programování. Lze přes něj desku i napájet.
3) Napájecí konektor. Lze pomocí něj napájet desku pokud nepoužijeme napájení pomocí USB konektoru.
4) ICSP hlavice pro externí programování USB-převodníku.
5) LED diody L, RX a TX. Dioda L indikuje připojení k portu 13, diody RX a TX poté indikují komunikaci přes sériovou linku.
6) Indikační LED dioda ON. Značí napájení desky.
7) ICSP hlavice pro externí programování čipu.
8) Digitální piny. Pomocí těchto pinů lze připojit další obvody a zařízení. Piny označené vlnovkou podporují pulzně šířkovou modulaci.
9) Piny Arduina spojené s napájením a uzemněním.
10) Analogové vstupy. Lze sem připojit vodiče, po kterých se bude posílat nějaká analogová hodnota.
11) LED matice. Pomocí systému řádků a sloupců lze zobrazit informace.

5.2.4 Wiring

Wiring je programovací jazyk, který je v současnosti využíván u desek Arduino. Programy se dělí na dvě časti: *setup* a *loop*. *Setup* je část programu, která proběhne jen jednou na začátku a poté se již neprovádí. Za *loop* se označuje část programu, která se bude provádět opakovaně ve smyčce.

Jazyk Wiring umožňuje práci s proměnnými a podporuje obecně známe datové typy jako integer či double. Je také doplněn o další typy, které pracují s piny a vstupy desky a lze přidat další datové typy pomocí dodatečně instalovaných knihoven. Podobně jako další známe jazyky umožňuje práci s cykly, tvorbu polí a mnohé další zažité funkce.

V tomto jazyku rozlišujeme typ vstupu a výstupu, přes který pin k němu přistupujeme a také, zda se jedná o analogový či digitální vstup či výstup. Mezi funkce, kterými tento jazyk disponuje, patří například *analogWrite* či *analogRead*.

5.2.5 Funkce

Desky Arduina jsou vybaveny řadou funkcí, které umožňují jednoduchou integraci různých veličin a operací. Mezi funkcemi pro práci s časem najdeme např. *delay*, *millis* či *micros*. Během funkce *delay* dojde k zastavení všech vnitřních činností na zadanou dobu. Funkce *millis* a *micros* vracejí hodnoty uložené ve vnitřním časovači procesoru, pouze s jinými jednotkami.

Můžeme zde také najít hojně používané matematické funkce pro zjištění maxima a minima, absolutní hodnoty, mocnin a odmocnin. Nechybí mimo jiné i nástroje pro práci s goniometrickými funkcemi.

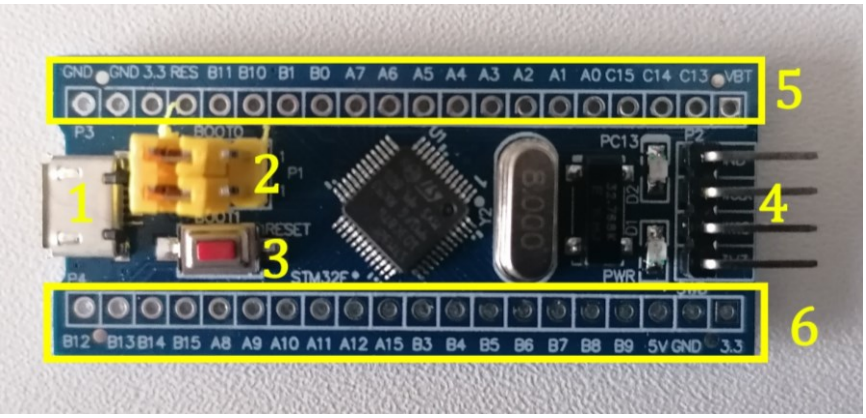
V prostředí Arduina je rovněž umožněno uživateli vytvořit a definovat vlastní funkce, které nemusí být v základu implementovány. Takové funkce se poté aktivují v hlavních částech kódu pomocí jejích pojmenování a případných proměnných.

5.2.6 Komunikace

Arduino umožňuje komunikaci mezi deskou a externím zařízením pro nahrání programu, ale také i vzájemné propojení dvou desek Arduino či podobných systémů. Nejčastěji dochází k přenosu po sběrnici I2C, ale většina modelů umožňuje i komunikaci pomocí protokolů UART a SPI.

5.3 STM32

STM32 je série 32 bitových mikrokontrolerů vyvíjených firmou STMicroelectronics. Jedná se o velkou řadu druhů a variací kontrolérů založených na architektuře ARM. V současnosti jsou tyto mikrokontrolery užívány v mnoha odvětvích od jednoduché tiskárny po komplexní obvodové desky v automobilech. [40]



Obrázek 16 STM32 F103C

5.3.1 Vybavení desky

1) Micro USB port pro napájení a připojení k počítači.
2) Jumpery bootu umožňují nastavit, zda se provede program uložený v desce, nebo se do desky uloží nový program.
3) Tlačítko reset.
4) Hlavice pro rozhraní k debugování.
5) Řada pinů, jsou zde obsaženy piny uzemnění, napětí a množství analogových vstupů či výstupů.
6) Řada pinů, jsou zde obsaženy piny uzemnění, napětí, propojení časovačů a kanály komunikací.

5.3.2 ARM

ARM je zkratka pro Advanced Risk Machine. Jedná se o jednu z nejpopulárnějších architektur použitou například v kamerách, mobilních telefonech a embedded systémech. Lze u nich vyzvednout hlavně nízkou konsumpci energie a lepší schopnosti výkonu. ARM je 32 bitová RISC architektura. V adresní sběrnici můžeme adresovat až 2^{32} lokací. RISC instrukce implikují, že hardware je komplexní a množství práce s procesy je prováděno v samotných komponentech. Tímto lze dosáhnout lehčích a jednodušších instrukcí pro ARM procesor.

5.3.3 Prostředí

Pro práci s STM32 je možné využít mnoho různých prostředí a softwarů jako STM32CubeIDE, CubeMX či Arduino IDE. Některé prostředí jsou již vybavena nástroji pro komunikaci s tímto typem desky. U ostatních jako např. Arduino IDE je nutno stáhnout a nainstalovat balíček pro práci s těmito deskami.

Následně lze napsat program a po správném nastavení jumperů bootu na desce a připojení desky k počítači můžeme daný program do desky nahrát.

5.4 Raspberry Pi

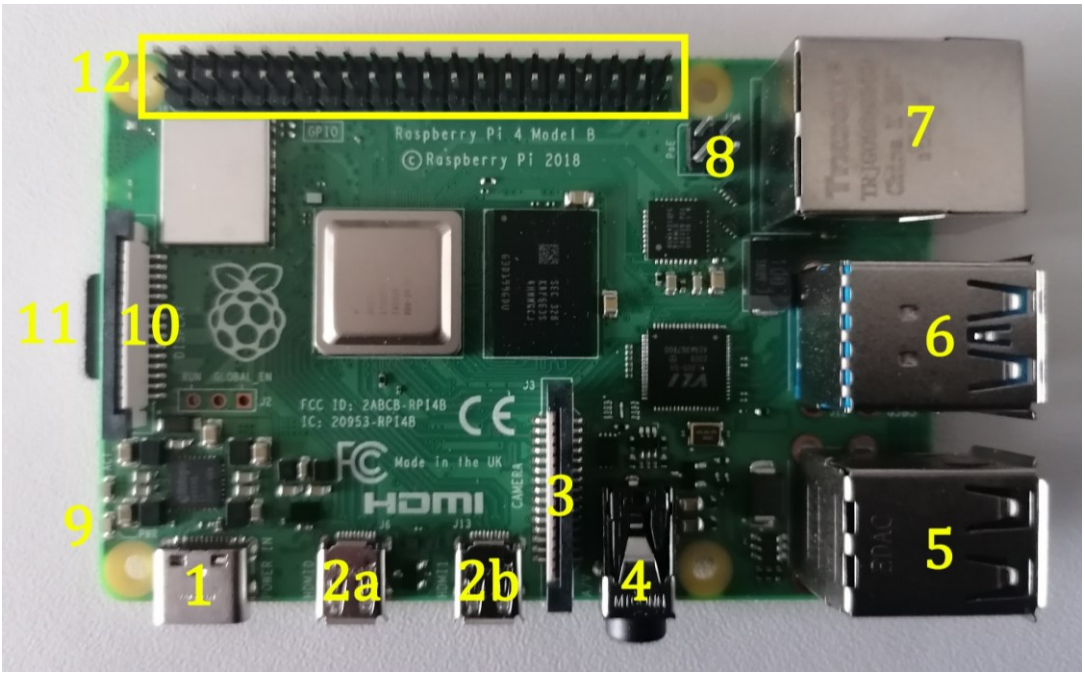
Raspberry Pi lez popsat jako minipočítač o velikosti bankovní karty, který je schopen propojení s různými vstupními a výstupními zařízeními jako monitor, televize, myš či

klávesnice, čímž lze efektivně konvertovat tento systém do „plně vybaveného“ stolního počítače za nízkou cenu. [41]

Jedná se o programovatelné zařízení, které na své základní desce obsahuje většinu funkcí jako běžný stolní počítač s výjimkou periferií a vnitřní paměti. Pro nastavení Raspberry Pi je nutné připojení SD karty s nainstalovaným operačním systémem.

5.4.1 Modely

Během let došlo k oblibě těchto zařízení a následnému vzniku velké řady modelů a verzí. Většinou došlo ke změně procesoru, paměti či jiného komponentu. Mezi nejznámější modely patří např.: Raspberry Pi Zero, Raspberry Pi 1, Raspberry Pi 2B, Raspberry Pi 3 a Raspberry Pi 4. V současné době byly také vydány i modely páté verze zařízení s označením Raspberry Pi 5.



Obrázek 17 Raspberry Pi 4B

5.4.2 Vybavení desky

1) Port USB-C slouží k propojení zařízení s počítačem a také jako přívod napájení.
2) Dva porty microHDMI umožňují připojení displejů a monitorů až do rozlišení 4K.
3) Konektor pro připojení kamerového modulu.
4) Konektor pro připojení audio k desce (Audio Jack 3,5mm).

5) Skupina dvou konektorů USB 2.0 umožňuje připojení a přesun souborů mezi zařízeními.
6) Skupina dvou konektorů USB 3.0 slouží k propojení a přesunu dat mezi zařízeními. Tento typ umožňuje větší rychlost přenosu.
7) Port Ethernetu přidává možnost napojení Raspberry Pi do sítě.
8) Hlavice PoE+ HAT umožňuje napájení přes Ethernet pro přídavný aktivní chladič a mini displej.
9) Kontrolka ON indikuje zapnutý stav zařízení.
10) Konektor pro připojení displejového modulu přímo od firmy Raspberry.
11) Slot SD karty se nachází na spodní straně zařízení a slouží k připojení systému, na kterém zařízení poběží. Také poskytuje dodatečné úložné místo.
12) Skupina pinů pro připojení externích součástek k Raspberry Pi. Nachází se zde například piny pro napájení, uzemnění a množství dalších GPIO pinů.

Výrobci desek Raspberry Pi také vydali další zařízení či rozšíření, která lze přímo připojit k desce a následně s nimi pracovat jako např. modul kamery.

5.4.3 Využití

Raspberry Pi lze využit nejen pro tvorbu desktopového počítače, ale také jako součást mnoha dalších projektů. V současné době došlo k rozmachu využití těchto desek pro ovládání miniaturních robotů. Hojně se také vytvářejí projekty, které toto zařízení využívají k nahrazení webových služeb a serverů např. pro komunikaci s tiskárnou či přenos informací pomocí webových protokolů.

V některých firmách jsou tyto desky využity pro správu kamerových systémů, kdy umožňují ukládání a rychlou správu uložených dat. V restauracích je možné jej také najít jako ovladač správy digitálních podpisů.

Tvorba nových modelů a dodatečných zařízení neustává a je jen otázkou času, kam se tyto desky postupně dostanou.

II. PRAKTICKÁ ČÁST

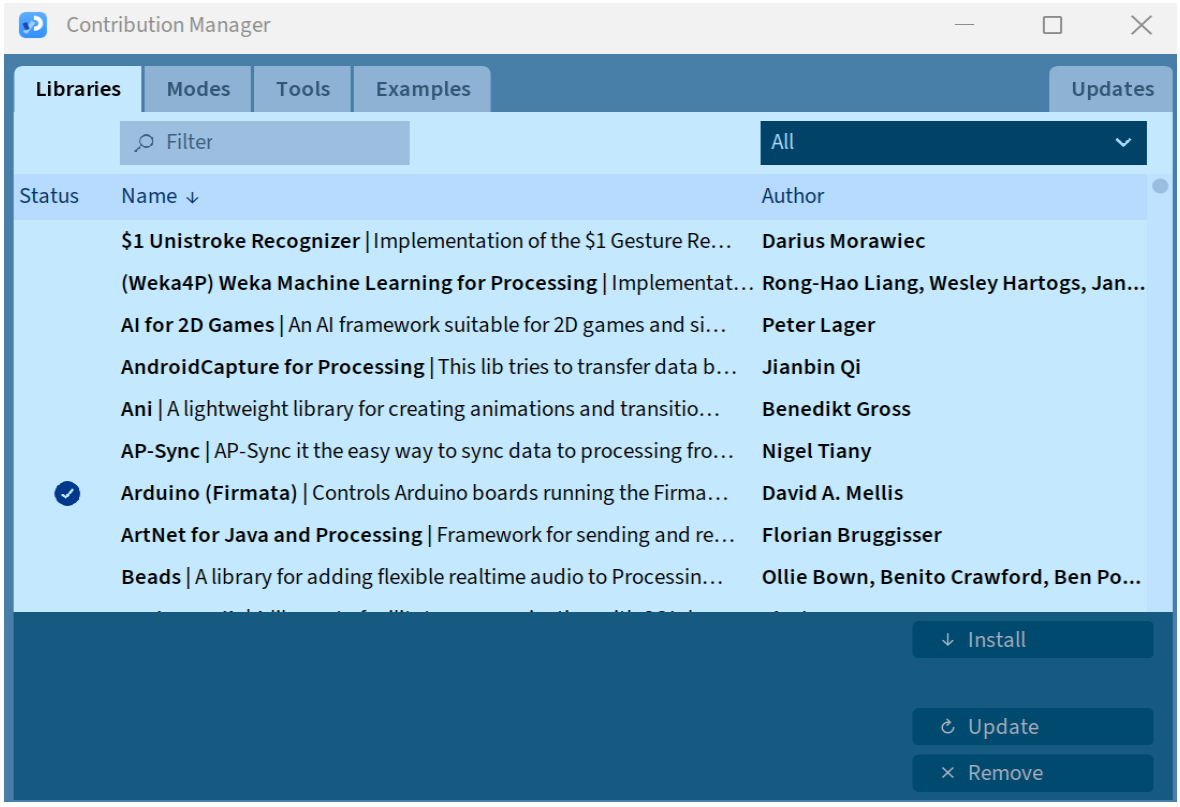
6 PRÁCE V NÁSTROJÍCH

V této kapitole praktické části budou popsány postupy práce ve zvolených nástrojích. Součástí bude také přehled nejvíce využitých knihoven a dalších možností, kterými jsou tyto nástroje vybaveny.

6.1 Processing

6.1.1 Knihovny

Velkou součástí práce s aplikací Processing bude propojení a využití knihoven v kódu. Tyto balíčky můžeme přidat v menu Sketch > Import Library > Manage Libraries. Tímto se nám otevře Contribution Manager, který nám umožňuje přidat různé knihovny a další nástroje. V tomto nově otevřeném menu můžeme scrollovat postranním tlačítkem nebo vyhledávat dle klíčového slova. Požadovanou knihovnu pak označíme a stiskem tlačítka Install přidáme do aplikace Processing. Modrý symbol s fajfkou značí, že je daná knihovna nainstalovaná. Většina z nich také obsahuje dodatečné informace, jak je použít. Pro následné použití v projektu je potřeba danou knihovnu či její část přidat pomocí příkazu *include*.



Obrázek 18 Contribution Manager v Processing

6.1.2 Použité knihovny

Sound

Software Processing disponuje rozsáhlou knihovnou s názvem Sound. Jedná se o knihovnu, jejíž účelem je vytvořit jednoduchý způsob pro práci se zvukem. Syntaxe nutná pro její využití je minimalizována za účelem jednoduchosti použití daným uživatelem. Umožňuje přehrát, analyzovat či syntetizovat zvuk. Zároveň je vybavena nástroji, které tvoří oscilační vlny, generují hluk či přidávají efekty a upravují zvukové soubory. Více informací o jejím použití a příkazech, které obsahuje, lze najít na následné webové stránce: *https://processing.org/reference/libraries/sound/index.html*

Minim

Jedná se o knihovnu vytvořenou uživateli Damienem Di Fedem a Andersonem Millsem pro software Processing, jejíž účelem je přidat jednoduchý způsob, jak integrovat zvuk do sketchů a zároveň zůstat flexibilní pro více pokročilé uživatele. Knihovna je volně dostupná ke stažení v Contribution Manageru a více informací lze najít na webové stránce: *https://code.compartmental.net/minim/*

Video

Knihovna Video přidává dodatečnou funkcionalitu již rozsáhlému programu Processing. Přidává například funkce pro přehrání souboru či nahrání nového videa za použití kamery. Vzhledem ke způsobu, jakým je tato knihovna zpracována, lze přečíst soubor přímo z internetu a také podporuje velkou řadu formátů. Lze ji nainstalovat pomocí Contribution Manageru. Pro další informace ohledně této knihovny a jejich funkcí lze zhlédnout stránku *https://processing.org/reference/libraries/video/index.html*

Firmata

Knihovna Firmata pomocí svých protokolů umožňuje komunikaci s hostitelským počítačem. Výsledkem je poté snazší implementace firmwaru bez nutnosti tvorby vlastních protokolů a nových objektů pro rozdílná programovací prostředí. Pro další informace se lze obrátit na webovou stránku *https://docs.arduino.cc/retired/hacking/software/FirmataLibrary/*

6.1.3 Dodatečné možnosti a nástroje

Jedním z frekventovaně navštěvovaných míst bude také Contribution Manager pro nástroje. Processing je v základu vybaven těmito hlavními nástroji: Movie Maker, Create Font, Color

Selector a Archive Sketch. Pokud si uživatel přeje přidat dodatečné nástroje, musí přejít do menu Tools > Manage Tools. Nově otevřené okno funguje stejně jako u knihoven, lze skrze něj buď scrollovat nebo vyhledávat dle klíčového slova. Po označení zvoleného nástroje klikneme na tlačítko Install. Dokončení instalace zjistíme podle modrého ukazatele s fajfkou. Nástroj poté následně můžeme otevřít v menu Tools v hlavním okně aplikace.

6.2 Arduino IDE

6.2.1 Psaní sketchů

Programy psané v nástroji Arduino IDE se zde nazývají sketche. Jsou tvořeny v textovém editoru prostředí a po uložení dostávají příponu *.ino*. Soubory s touto koncovkou musí být uloženy ve složce se stejným názvem jako její přidružený soubor. V typickém systému se nacházejí ve složce Arduino v uživatelské složce Dokumenty. Seznam těchto sketchů lze zobrazit v knize sketchů, do které lze vstoupit tlačítkem s ikonou složky na postranní liště prostředí.

6.2.2 Balíčky pro desky

Pomocí manažeru desek, který je dostupný po kliknutí na tlačítko s ikonou desky na postranní liště, lze stáhnout nezbytné balíčky, které některé desky mohou vyžadovat. Po otevření manažeru můžeme v seznamu hledat potřebné balíčky či takzvaná „jádra“ pro naše desky. Každá deska vyžaduje jádro při kompilaci a uploadu kódu na desky. V menu lze vybrat z nabídky různých jader např. AVR, MegaAVR, SAM či jiné, ale také vybrat verzi k instalaci.

6.2.3 Knihovny

Velkou součástí práce s prostředím Arduino IDE je užití knihoven. Tým Arduino a komunita během let vytvořila tisíce knihoven, které se dají stáhnout online. Uživatel si také může vybrat z řady knihoven, které se dají stáhnout přímo v prostředí, s tím, že tyto knihovny jsou většinou nejdříve aktualizované a většinou obsahují dobře zpracovanou dokumentaci. Tyto knihovny mohou umožnit komunikaci mezi prostředím a různými druhy senzorů, Wi-Fi modulů či jinými komponenty na desce. Některé zpracovávají komunikaci v reálném čase.

Stisknutím tlačítka s ikonou knihovny na postranní liště zobrazíme manažera knihoven. V nově otevřeném menu můžeme hledat, případně scrollovat a najít potřebné knihovny.

U většiny knihoven lze zvolit verzi, kterou chceme nainstalovat, a také zobrazit dokumentaci, jak danou knihovnu použít.

Nainstalované knihovny můžeme použít dvěma způsoby. Některé knihovny již obsahují předpřipravené sketche s již obsaženou knihovnou, které můžeme otevřít v menu File > Examples > {Library} > {Example}. (Library a Example jsou zástupné názvy, záleží na dané knihovně.) Druhým způsobem je přidání knihovny do již rozpracovaného projektu použitím příkazu *#include <Library.h>* na začátek daného sketche. (*Library.h* je ukázkový příklad. Ostatní knihovny budou obsahovat vlastní hlavičkové soubory.)

6.2.4 Sériový monitor a plotter

Sériový monitor a plotter jsou nástroje, které umožňují zobrazení dat ve formě vhodné pro uživatele. Monitor umožňuje zobrazit data, která přicházejí ve streamu z Arduino desky. Plotter umožňuje zobrazit data ve formě grafů, což se hodí například pro zobrazení výkyvů v napětí či jiných veličinách.

Od verze IDE 2 se monitor nyní otevírá jako nové okno konzole namísto úplně nového okna mimo prostředí. Pro použití monitoru ve sketchy musíme pomocí příkazu *Serial.begin(9600)* nastavit přenosovou rychlost. (9600 bitů za sekundu v tomto případě určuje přenosovou rychlost.) Poté musíme přidat příkazy, které předají informace monitoru, jako např. *Serial.print()* a jeho variace. Po následném uploadu programu do desky můžeme pomocí tlačítka po pravé horní straně otevřít sériový monitor, ve kterém se nám zobrazí přijaté informace z desky. Prostředí nám umožňuje otevřít více oken monitoru souběžně pro více běžících sketchů.

Nástroj plotter funguje na podobném principu jako monitor. Pro použití tohoto nástroje je třeba, aby sketch obsahoval alespoň jednu numerickou proměnou, tudíž minimálně jedno celé nebo desetinné číslo (typu *int* či *float*). Poté musíme pomocí příkazu *Serial.begin(9600)* nastavit přenosovou rychlost, v tomto případě na 9600 bitů za sekundu. Také musíme danou hodnotu předat plotteru pomocí příkazu *Serial.print()*. Stejným způsobem také připravíme popisy a vzhled požadovaného grafu. Po následném uploadu kódu do desky a otevření sériového plotter tlačítkem v pravém horním rohu prostředí bychom měli otevřít plotter, který již získává informace a zobrazuje je ve formě uživatelem zvoleného grafu.

6.3 Fritzing

6.3.1 Součástky

Při instalaci softwaru Fritzing dojde také k vytvoření knihovny součástek, které se v průběhu mění dle vývoje a verze aplikace. Tyto součástky jsou rozděleny do takzvaných „binů“, ke kterým lze přistoupit v menu na pravé straně aplikace. Pro vložení vybrané součástky do plochy návrhu již stačí část podržet a přetáhnout na vybrané místo v prostoru. V základním balení aplikace najdeme několik binů. Nejdůležitější z nich najdeme v sekcích „Core“ a „Mine“. Core bin obsahuje základní součástky, které jsou dále rozděleny do menších kategorií dle společných rysů. V binu Mine poté najdeme importované součástky, které v běžné verzi projektu nebyly dodány. Pokud hledáme určitou součástku, můžeme použít okno pro vyhledávání na levé straně menu. V dalších binech lze najít součástky roztřízené podle určitých kritérií, např. výrobce.

6.3.2 Tvorba obvodu

V sekci Montážní deska můžeme vytvořit požadovaný obvod. Při otevření této sekce dojde k vytvoření experimentální desky, ke které budeme postupně připojovat další komponenty. Součástky můžeme přidávat z menu na pravé straně aplikace, tlačítkem Delete je poté můžeme odstranit. Pokud potřebujeme specifickou součástku, která není součástí základní nabídky aplikace, musíme si danou součástku naimportovat. Poté ji podržením přesuneme do pracovní plochy. Stejným způsobem vložíme také i ostatní součástky, které chceme připojit. Při kliknutí a označení určitého komponentu můžeme v menu Inspectoru na pravé spodní straně aplikace upravit dodatečné specifikace. Například u rezistorů můžeme změnit odpor. Každý pin vybraných komponentů má svůj popisek, pokud přejedeme myší na místo připojení komponentu, dostaneme informace například o polaritě této strany připojení.

Po umístění součástek do pracovní plochy a do experimentální desky se na této desce zelenou barvou zvýrazní linky pro připojení. Následně můžeme kliknout a táhnout od potřebného pinu či linky na desce, po přesunutí myši na požadované místo již jen pustíme tlačítko myši, čímž dojde k vytvoření spojujícího kabelu. U tohoto kabelu můžeme dle požadavků upravit jeho vlastnosti. Pravým tlačítkem myši klikneme na kabel a přejedeme myší nad možností Barva propojky, poté vybereme žádoucí barvu. Pokud potřebujeme kabel zahnout použijeme klávesu Ctrl.

Zpracovaný náčrt poté můžeme exportovat pomocí File > Export > požadovaný formát.

7 PRÁCE S OBRÁZKY

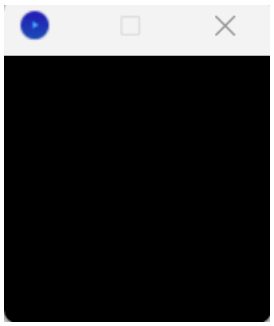
7.1 Obrázek a jednoduché úpravy barev

7.1.1 Tvorba prázdného obrazu

Ne vždy budeme mít k dispozici již předpřipravený obrázek, za tímto účelem lze v aplikaci Processing přímo vytvořit prázdný obrázek, do kterého je možné v rámci stejného programu rovnou něco vykreslit. Použití funkce *createImage* vytvoří prázdný černý obrázek o zvolené velikosti.

Prázdný obraz – Processing

```
PImage img;

void setup(){
  size(200,200);
  // Tvorba prázdného obrázku o velikosti 200x200 pixelů přímo v programu.
  img = createImage(200,200,RGB);
}

void draw(){
  background(0);
  // Vykreslí obrázek na koordinátech 0,0.
  image(img, 0, 0);
}
```



Obrázek 19 Prázdný obrázek

7.1.2 Načtení obrázků

Jako jedna z nejzákladnějších funkcí programu je funkce nahrání obrázku ze souborů na disku počítače. Pokud umístíme do našeho programu složku s názvem *data*, do které následně vložíme požadovaný obrázek, můžeme do programu daný obrázek nahrát a zobrazit. Dosáhneme toho pomocí funkce *loadImage*, do jejíž argumentů uvedeme název souboru umístěného ve složce projektu.

Načítání obrázku – Processing

```
PImage img;

void setup(){
  size(640,360);
  // Načtení obrázku že složky data v projektu pod zadaným jménem.
  img = loadImage("alaska.jpg");
}

void draw(){
  background(0);
  // Vykreslí obrázek na koordinátech 0,0.
  image(img, 0, 0);
}
```

Obrázek 20 Načtený obrázek

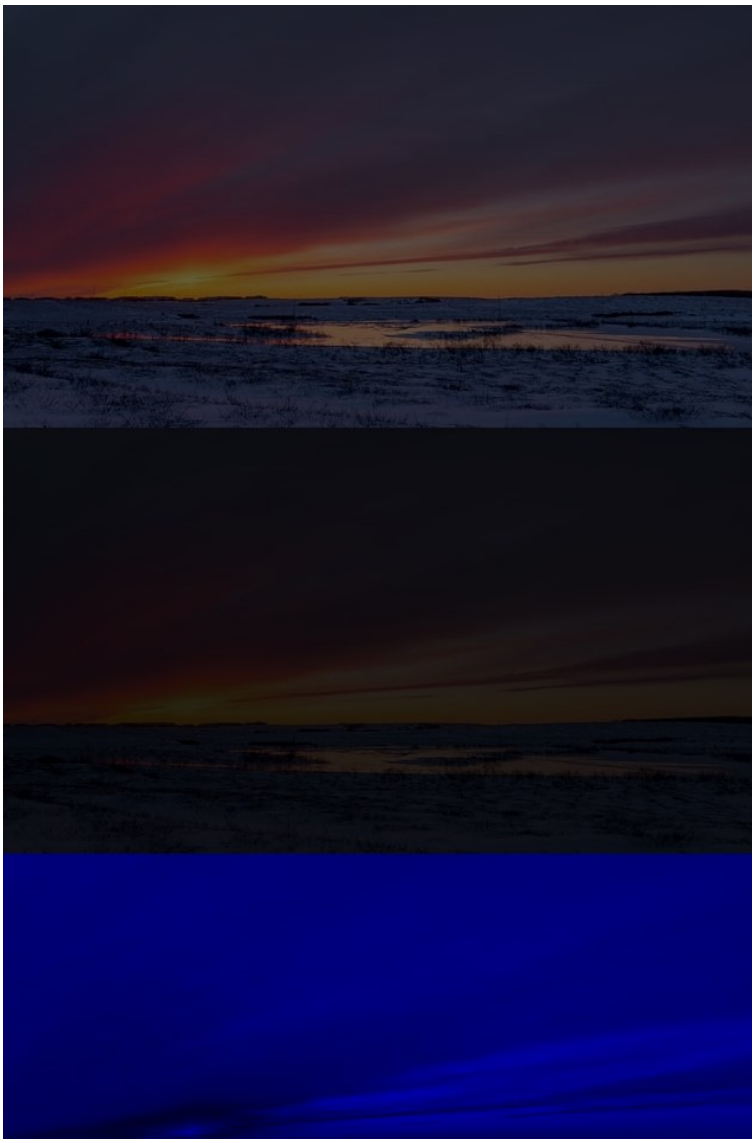
7.1.3 Úprava odstínu, jasu a světelnosti

Úpravy vlastností obrázku lze docílit vícero způsoby. Processing je v základu vybaven velkou škálou funkcí, které uživatelům této aplikace umožňují snadný postup v jejich programech. Cíle tohoto příkladu lze dosáhnout nejen pomocí použití předpřipravených funkcí, ale také lze napřímo přistoupit do dat načteného obrázku a výsledný obraz pozměnit ručně.

Úprava odstínu, jasu a světelnosti – Funkce – Processing

```
PImage img;

void setup(){
  size(640,1080);
  // Načtení obrázku že složky data v projektu pod zadaným jménem.
  img = loadImage("alaska.jpg");
}

void draw(){
```

```
  background(0);
  // tint() s jedním argumentem mění světelnost obrázku.
  tint(100);
  // Vykreslí obrázek na koordinátech 0,0.
  image(img, 0, 0);

  // tint() se dvěma parametry mění průhlednost.
  tint(100,100);
  image(img, 0, 360);

  // tint() se třemi parametry mění RGB obrazu.
  tint(0, 0, 255);
  image(img, 0, 720);
}
```



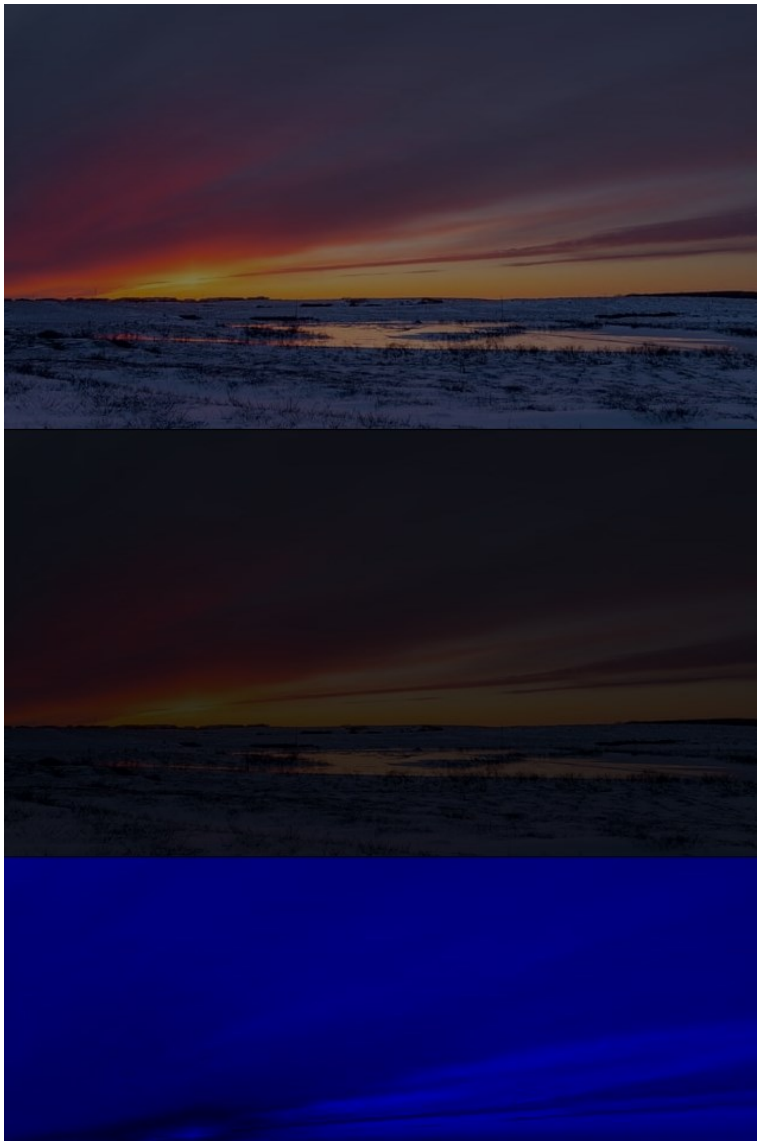
Obrázek 21 Ukázka úpravy odstínu a jasu

Stejný problém se dá řešit i bez použití funkce *tint* pomocí přímého přístupu k pixelům načteného obrázku. Tento způsob řešení je demonstrován v následující tabulce, kde dojde k tvorbě obrázku s podobným výsledkem.

Úprava odstínu, jasu a světelnosti – Přímý přístup – Processing

```
PImage i1;

void setup(){
  size(640,1080);
  i1 = loadImage("alaska.jpg");
}

void draw(){
    image(i1, 0, 0);
    // Nahrává informace o pixelech do Pixels[].
    i1.loadPixels();

    PImage i2 = createImage(i1.width, i1.height, RGB);
    PImage i3 = createImage(i1.width, i1.height, RGB);
    PImage i4 = createImage(i1.width, i1.height, RGB);

    for (int y = 1; y < i1.height; y++){ // Projde pixely shora dolů.
      for (int x = 1; x < i1.width; x++) { // Projde pixely zleva doprava.

      // Pozice pixelu v poli.
      int loc = x + y * i1.width;
      i2.pixels[loc]   =   color(0.5   *   red(i1.pixels[loc]),   0.5   *
green(i1.pixels[loc]), 0.5 * blue(i1.pixels[loc]));
      }
    }
    i2.updatePixels();

    for (int y = 1; y < i1.height; y++){ // Projde pixely shora dolů.
      for (int x = 1; x < i1.width; x++) { // Projde pixely zleva doprava.

      // Pozice pixelu v poli.
      int loc = x + y * i1.width;
      i3.pixels[loc]   =   color(0.2   *   red(i1.pixels[loc]),   0.2   *
green(i1.pixels[loc]), 0.2 * blue(i1.pixels[loc]));
      }
    }
    i3.updatePixels();

    for (int y = 1; y < i1.height; y++){ // Projde pixely shora dolů.
      for (int x = 1; x < i1.width; x++) { // Projde pixely zleva doprava.

      // Pozice pixelu v poli.
      int loc = x + y * i1.width;
      i4.pixels[loc] = color(0,0, blue(i1.pixels[loc]));
      }
    }
    i4.updatePixels();

    image(i2, 0, 0);
```

```
    image(i3, 0, 360);
    image(i4, 0, 720);
}
```



Obrázek 22 Ukázka odstínu a jasu pomocí přístupu k pixelům

7.2 Geometrické objekty

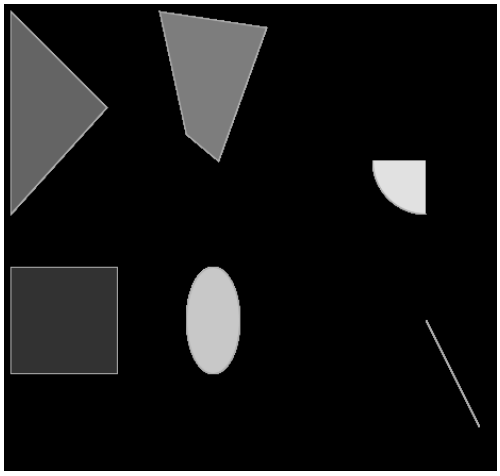
7.2.1 Tvorba geometrických objektů ve 2D

Processing je vybaven řadou nástrojů pro rychlou tvorbu geometrických objektů. Použití těchto funkcí je nastíněné v následujícím příkladu.

Tvorba geometrických objektů ve 2D – Processing

```
void setup(){
  size(500, 500);
```

```
  background(0);
}

void draw(){
  fill(100);
  // Funkce pro tvorbu trojúhelníku, první dva argumenty tvoří první bod, další dvojice představují další body. Bod 10,10 Bod 10, 200 a Bod 100,100.
  triangle(10, 10, 10, 200, 100, 100);

  fill(50);
  // Funkce pro tvorbu obdélníku či čtverce (existuje i square(x,y,"výška a šířka"), první dva argumenty tvoří první bod, třetí argument určuje šířku a čtvrtý výšku.
  // Přidáním dalších 4 argumentů lze zakulatit rohy.  V pořadí TL, TR, BR, BL. TL = top left...
  rect(10, 250, 100, 100);

  fill(125);
  // Funkce pro tvorbu čtyřstranného polygonu, každá dvojice argumentů udává bod, musí být v hodinovém či proti hodinovému pořadí.
  quad(150, 10, 250, 25, 205, 150, 175, 125);

  fill(200);
  // Funkce pro tvorbu kruhu či elipsy, první dva argumenty tvoří první bod, třetí argument určuje šířku a čtvrtý výšku.
  ellipse(200, 300, 50, 100);

  fill(225);
  //Funkce pro tvorbu oblouku, první dva argumenty tvoří první bod, třetí argument určuje šířku, čtvrtý výšku, pátý začátek úhlu v radiánech, šestý konec úhlu v radiánech.
  arc(400, 150, 100, 100, HALF_PI, PI);

  stroke(175);
  // Funkce pro tvorbu úsečky, první dva argumenty tvoří první bod, další dva argumenty tvoří druhý bod.
  line(400, 300, 450, 400);
}
```



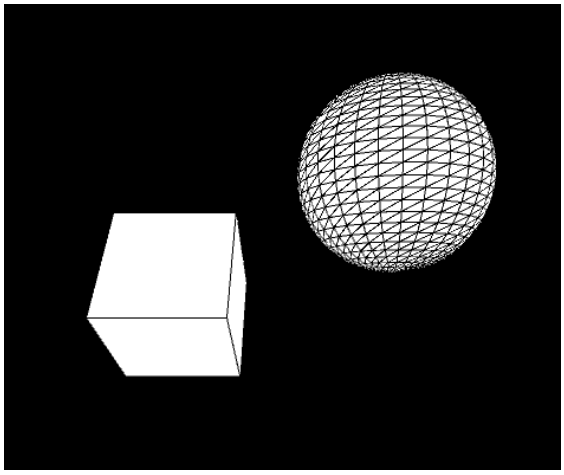
Obrázek 23 Tvary ve 2D

7.2.2 Tvorba geometrických objektů ve 3D

Tvorba 3D objektů může být někdy složitým a zdlouhavým procesem, naštěstí je software Processing vybaven funkcemi, které umožní relativně jednoduchou práci s těmito objekty. Pro užití těchto nástrojů je třeba specifikovat práci ve 3D prostoru pomocí dodatečného argumentu *P3D* při tvorbě pozadí.

Tvorba geometrických objektů ve 3D – Processing

```
void setup(){
  // Musí být uveden argument render módu ve funkci size.
  size(500, 500, P3D);
  background(0);
}

void draw(){
  // Pro změnu umístění objektu slouží funkce translate() a rotate_() se
zadanou osou.
  // Pomocí funkcí pushMatrix a popMatrix můžeme uložit aktuální pozici
matice na stack, a druhým příkazem ji vrátit do původního stavu.
  pushMatrix();
  translate(150, height/2, 0);
  rotateX(10);
  box(100);
  popMatrix();

  pushMatrix();
  translate(350, height/4, -100);
  sphere(100);
  popMatrix();
}
```



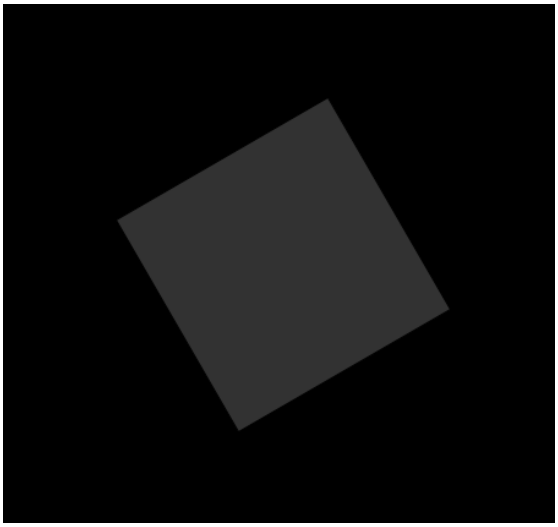
Obrázek 24 Tvary ve 3D

7.2.3 Rotace a translace objektu

Rotace objektů můžeme dosáhnout několika způsoby. Nejjednodušším způsobem je použití předdefinovaných funkcí zabudovaných přímo v softwaru Processing. Druhým způsobem je poté použití matic a přímý přesun bodů objektu po prostoru.

Rotace a translace objektů – Funkce – Processing

```
void setup(){
  size(500, 500);
  background(0);
}

void draw(){
  fill(50);
  // Funkce translate určuje dispozici objektů v prostoru vzhledem k původním souřadnicím.
  translate(width/2, height/2);
  // Funkce rotate otáčí objekt o zadaný úhel. 0 až 2 PI, nebo radiány.
  rotate(PI/3.0);
  rect(-100, -100, 200, 200);
}
```



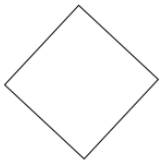
Obrázek 25 Rotace a translace objektu

Součástí kódu pro rotaci pomocí matice je také implementace animace otáčení objektu, její rychlost lze nastavit pomocí parametru *angle* v sekci *draw*.

Rotace a translace objektů – Matice – Processing

```
float angle = 0; // Úhel rotace
PVector translation = new PVector(250,250); // Vektor translace
PVector[] points; // Pole pro uložení bodů objektu.
PVector[] newpoints; // Pole pro uložení nových bodů.

void setup() {
```

```
  size(500,500);
  points = new PVector[4]; // Počet bodů
  points[0] = new PVector(-50, -50);
  points[1] = new PVector(50, -50);
  points[2] = new PVector(50, 50);
  points[3] = new PVector(-50, 50);

  newpoints = new PVector[points.length];
}

void draw() {
  background(255);

  applyTransform();

  drawObject();

  // Úhel rychlosti animace.
  angle += 0.01;
}

// Funkce pro výpočet posunu bodů pomocí transformace.
void applyTransform() {
  float[][] rotationMatrix = {
    {cos(angle), -sin(angle)},
    {sin(angle), cos(angle)}};

  for (int i = 0; i < points.length; i++) {
    float x = points[i].x * rotationMatrix[0][0] + points[i].y *
rotationMatrix[0][1];
    float y = points[i].x * rotationMatrix[1][0] + points[i].y *
rotationMatrix[1][1];

    x += translation.x;
    y += translation.y;

    newpoints[i] = new PVector(x,y);
  }
}

// Funkce pro vykreslování objektu.
void drawObject() {
  for (int i = 0; i < newpoints.length; i++) {
    int nextIndex = (i + 1) % newpoints.length;
    line(newpoints[i].x,      newpoints[i].y,      newpoints[nextIndex].x,
newpoints[nextIndex].y);
  }
}
```



Obrázek 26 Rotace objektu pomocí matice

7.3 Animace

Animace umožňují přivést obrázky k životu. V současné době jsou animace tvořeny řadou obrázků v sekvenci, které se vysokou rychlostí střídají, čímž dojde ke vzniku „pohybu“ obrazu. Aplikace Processing disponuje možností vysokého počtu opakovaní částí programu, ve které lze pomocí inkrementace hodnot dosáhnout podobného efektu.

7.3.1 Animace obrázku

Základní animace může být vytvořena jednoduchou translací polohy obrázku při každém dalším opakování programu. Vznikne tím pohyb, při kterém obraz putuje v prostoru a mění svou polohu a rotaci.

Animace obrázku – Processing

```
PImage obj;
// Proměnné pro pozici a rotaci objektu.
float x,y;
float rot;

void setup(){
  size(1000,1000);
  obj = loadImage("alaska.jpg");
  x = 0.0f;
  y = width/2.0f;
  rot = 0.0f;
}

void draw(){
  background(255);

  translate(x, y);
  rotate(rot);
```

```
  image(obj, 0,0);

  // Změna polohy objektu.
  x+= 1.0;
  rot += 0.01;
  if (x > width){
    x = 0;
  }
}
```

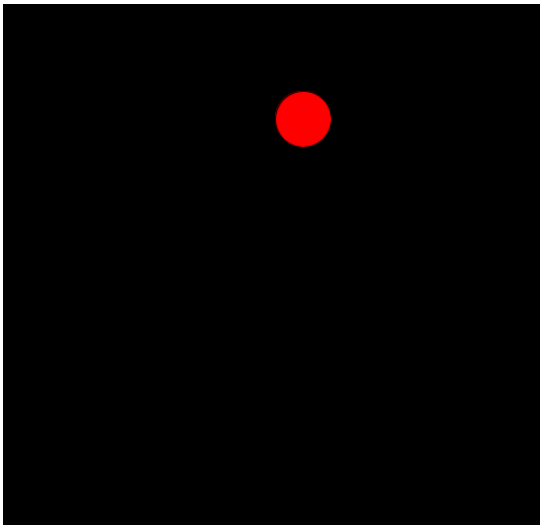
Obrázek 27 Animace obrázku

7.3.2 Animace podle polohy kurzoru

Processing obsahuje funkce pro práci s polohou kurzoru, díky kterým můžeme v každé iteraci bloku *draw* resetovat pozadí a upravit polohu elipsy dle polohy kurzoru. Tímto způsobem lze docílit animace se závislostí na kurzoru uživatele.

Animace podle polohy kurzoru – Processing

```
void setup(){
  size(500, 500);
}

void draw(){
  background(0);
  fill(255, 0, 0);
  ellipse(mouseX, mouseY, 50, 50);
}
```



Obrázek 28 Animace podle polohy kurzoru

7.4 Křivky

Křivky jsou nedílnou součástí dnešní grafiky. Existuje velká řada variací a druhů, z nichž lze většinu vytvořit i v prostředí Processing. V této sekci dojde k ukázkám tvorby určených křivek.

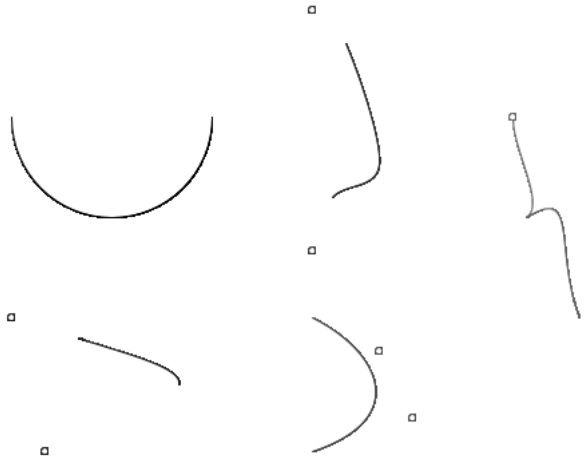
7.4.1 Základní křivky a Bézierova křivka

Processing je vybaven řadou funkcí, které naprosto poslouží pro tvorbu jednoduchých základních křivek. Níže je uveden příklad s použitím těchto funkcí a jejich strukturou.

Základní křivky a Bézierova křivka – Processing

```
void setup(){
  size(500,500);
  background(255);
  smooth();
}

void draw(){
  stroke(0);
  // Funkce pro tvorbu jednoduché křivky, podle elipsy, první dva argumenty tvoří bod, další dva určují šířku a délku, pátý a šestý určují začátek a konec úhlu v radiánech.
  arc(100, 100, 150, 150, 0, PI);

  stroke(25);
  // Funkce curve() pro tvorbu křivky spline, první dva argument tvoří první kontrolní bod, další pár argumentů tvoří počátek křivky, další pár argumentů tvoří konec křivky.
  // Poslední pár argumentů tvoří druhý kontrolní bod. Elipsy slouží k znázornění kontrolních bodů.
  curve(25, 250, 75, 265, 150, 300, 50, 350);
  ellipse(25, 250, 5, 5);
```

```
  ellipse(50, 350, 5, 5);

  stroke(50);
  // Funkce pro tvorbu průběžné spline o více bodech. Uzavřeno mezi bloky
beginShape() a endShape(). Užívá se curveVertex.
  // První vertex určuje kontrolní bod, druhý vertex počátek křivky, každý
bod až do předposledního je bodem křivky, předposlední bod je konec křivky,
poslední bod je druhý kontrolní bod.
  beginShape();
  curveVertex(250, 20);
  curveVertex(275, 45);
  curveVertex(300, 135);
  curveVertex(265, 160);
  curveVertex(250, 200);
  endShape();
  ellipse(250, 20, 5, 5);
  ellipse(250, 200, 5, 5);

  stroke(75);
  // Funkce pro tvorbu Bézierových křivek.
  // První dva argumenty tvoří začátek křivky, druhý pár tvoří první
kontrolní bod, třetí pár tvoří druhý kontrolní bod a čtvrtý pár tvoří konec
křivky.
  bezier(250, 250, 300, 275, 325, 325, 250, 350);
  ellipse(300, 275, 5, 5);
  ellipse(325, 325, 5, 5);

  stroke(100);
  // Funkce pro tvorbu průběžné Bézierovy křivky o více bodech.
  // Uzavřeno mezi bloky beginShape() a endShape(). Užívá se bezierVertex();
  // bezierVertex(cpx1, cpy1, cpx2, cpy2, x, y)
  beginShape();
  vertex(400, 100); // První bod
  bezierVertex(400, 125, 425, 165, 410, 175);
  bezierVertex(450, 150, 430, 200, 450, 250);
  endShape();
  ellipse(400, 100, 5, 5);
}
```



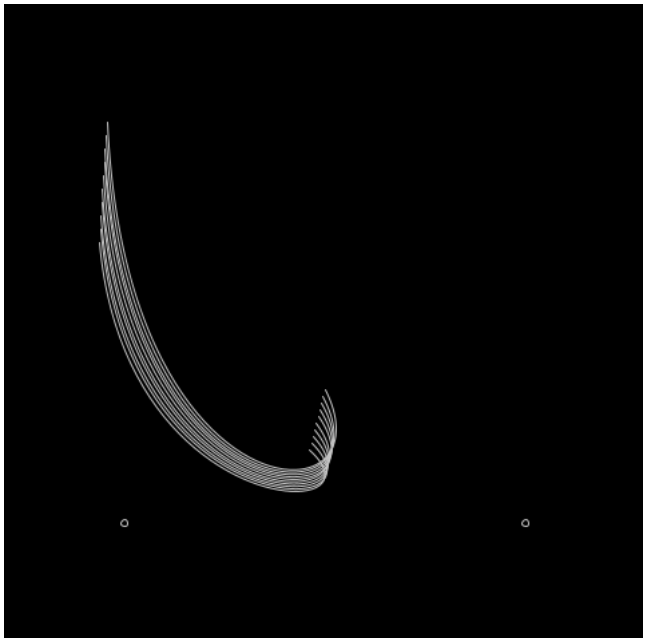
Obrázek 29 Základní křivky

7.4.2 Bézierovy křivky závislé na poloze kurzoru

Křivky je možné průběžně měnit i v závislosti na poloze kurzoru myši. V tomto příkladu lze měnit úhel a trajektorii křivky na ose *Y*.

Bézierovy křivky závislé na poloze kurzoru – Processing

```
void setup(){
  size(500,500);
  stroke(200);
  noFill();
  // Bez vyplnění křivek.
}

void draw(){
  background(0);
  // Aplikuje funkci bezier, pohyb myši nahoru a dolů mění hodnotu X, výška
křivky na 100 + i za každou iteraci.
  // Kontrolní body 100, 400 a druhý 400, 400.
  // Konec křivky v rozestupu 8 a 2 pro hodnoty X a Y.
  for (int i = 0; i < 100; i += 10) {
    bezier(mouseY-(i/16.0),  100+i,  100,  400,  300,  400,  250-(i/8.0),
300+(i/2.0));
  }
  ellipse(100, 400, 5, 5);
  ellipse(400, 400, 5, 5);
}
```



Obrázek 30 Bézierovy křivky podle kurzoru

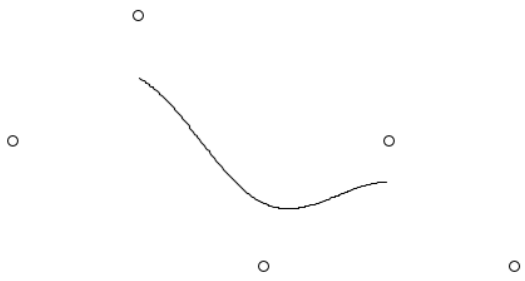
7.4.3 B-Spline křivka

B-Spline křivka již nemá předdefinovanou funkci, je proto nutné vytvořit pomocnou funkci, která má za úkol určit a vypočítat lokaci pomocných bodů křivky, podle kterých se bude tvarovat.

B-Spline křivka – Processing

```
float[] pointsX = {50, 150, 250, 350, 450}; // Pozice X bodů
float[] pointsY = {150, 50, 250, 150, 250}; // Pozice Y bodů

void setup() {
  size(500, 300);
  background(255);
  smooth();
  noFill();

  // Vykreslení kontrolních bodů.
  for (int i = 0; i < pointsX.length; i++) {
    ellipse(pointsX[i], pointsY[i], 8, 8);
  }

  stroke(0);
  drawBSpline(pointsX, pointsY);
}

void draw() {
}

void drawBSpline(float[] x, float[] y) {
  int n = x.length;
  if (n > 3) {
    beginShape();
```

```
    //curveVertex(x[0],y[0]);
    for (int i = 0; i < n - 3; i++) {
      for (float t = 0; t <= 1; t += 0.01) {
        float[] xPts = new float[4];
        float[] yPts = new float[4];
        for (int j = 0; j < 4; j++) {
          if (i + j < n) {
            xPts[j] = x[i + j];
            yPts[j] = y[i + j];
          } else {
            xPts[j] = x[n - 1];
            yPts[j] = y[n - 1];
          }
        }
        float xVal = calculateBSplinePoint(xPts, t);
        float yVal = calculateBSplinePoint(yPts, t);
        curveVertex(xVal, yVal);
      }
    }
    //curveVertex(x[n-1],y[n-1]);
    endShape();
  }
}

// Výpočet bodů křivky.
float calculateBSplinePoint(float[] points, float t) {
  float p0 = points[0];
  float p1 = points[1];
  float p2 = points[2];
  float p3 = points[3];

  float t2 = t * t;
  float t3 = t * t2;

  float b0 = (-t3 + 3*t2 - 3*t + 1) / 6.0;
  float b1 = (3*t3 - 6*t2 + 4) / 6.0;
  float b2 = (-3*t3 + 3*t2 + 3*t + 1) / 6.0;
  float b3 = t3 / 6.0;

  return b0 * p0 + b1 * p1 + b2 * p2 + b3 * p3;
}
```



Obrázek 31 B-Spline

7.4.4 NURBS

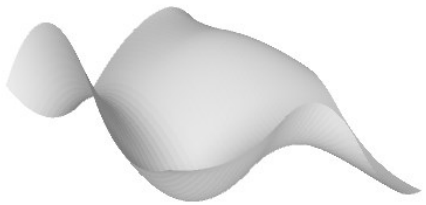
NURBS nelze vytvořit jednoduše využitím již vytvořené funkce. Pro úspěšné zobrazení povrchu NURBS je třeba vytvořit funkce, které vypočítají plochu podle zadaných kontrolních bodů.

Tento příklad byl založen na starém kódu od Alasdaira Turnera. Došlo k úpravě funkcí a přesunu do novější verze programu Processing.

NURBS – Processing

```
float[] u_knots = {
  0.0, 0.150, 0.300, 0.450, 0.600, 0.750, 0.850, 1.0
};
float[] v_knots = {
  0.0, 0.150, 0.300, 0.450, 0.600, 0.750, 0.850, 1.0
};

PVector[][] ctrl_pts;

int u_ctrl_pts = 5;
int v_ctrl_pts = 5;

float u_spacing;
float v_spacing;

void setup() {
  size(800, 600, P3D);

  ctrl_pts = new PVector[u_ctrl_pts][v_ctrl_pts];

  // Nastavení kontrolních bodů s náhodnou výškou.
  u_spacing = (width / u_ctrl_pts);
  v_spacing = (width / v_ctrl_pts);

  for (int i = 0; i < u_ctrl_pts; i++) {
    for (int j = 0; j < v_ctrl_pts; j++) {
      ctrl_pts[i][j] = new PVector(u_spacing * i, random(0, height), -
v_spacing * j);
    }
  }
}

void draw() {
  background(255);
  lights();
  translate(width / 2, height / 2, -width / 2);
  rotateY(frameCount * 0.01);
  translate(-width / 2, -height / 2, width / 2);

  int u_deg = u_knots.length - u_ctrl_pts - 1;
  int v_deg = v_knots.length - v_ctrl_pts - 1;

  // Vykreslení povrchu.
```

```
  for (float u = u_knots[u_deg]; u <= u_knots[u_knots.length - u_deg - 1]
- 0.01; u += 0.01) {
    beginShape(QUAD_STRIP);
    for (float v = v_knots[v_deg]; v <= v_knots[v_knots.length - v_deg -
1]; v += 0.01) {
      PVector pt_uv = new PVector();
      PVector pt_u1v = new PVector();
      for (int i = 0; i < u_ctrl_pts; i++) {
        for (int j = 0; j < v_ctrl_pts; j++) {
          float basisv = numbasis(v, j, v_deg, v_knots);
          float basisu = numbasis(u, i, u_deg, u_knots);
          float basisu1 = numbasis(u + 0.01, i, u_deg, u_knots);
          PVector pk = PVector.mult(ctrl_pts[i][j], basisu * basisv);
          PVector pk1 = PVector.mult(ctrl_pts[i][j], basisu1 * basisv);
          pt_uv.add(pk);
          pt_u1v.add(pk1);
        }
      }
      noStroke();
      fill(255);
      vertex(pt_uv.x, pt_uv.y, pt_uv.z);
      vertex(pt_u1v.x, pt_u1v.y, pt_u1v.z);
    }
    endShape();
  }
}

float numbasis(float u, int k, int d, float[] knots) {
  if (d == 0) {
    return endbasis(u, k, knots);
  } else {
    float b1 = numbasis(u, k, d - 1, knots) * (u - knots[k]) / (knots[k +
d] - knots[k]);
    float b2 = numbasis(u, k + 1, d - 1, knots) * (knots[k + d + 1] - u) /
(knots[k + d + 1] - knots[k + 1]);
    return b1 + b2;
  }
}

float endbasis(float u, int k, float[] knots) {
  if (u >= knots[k] && u < knots[k + 1]) {
    return 1;
  } else {
    return 0;
  }
}

void mousePressed() {
  // Náhodné vybraní nových kontrolních bodů po kliknutí.
  for (int i = 0; i < u_ctrl_pts; i++) {
    for (int j = 0; j < v_ctrl_pts; j++) {
      ctrl_pts[i][j] = new PVector(u_spacing * i, random(0, height), -
v_spacing * j);
    }
  }
}
```



Obrázek 32 NURBS

7.5 Efekty a pokročilé úpravy obrazu

7.5.1 Rozostření

Jedním ze základních efektů práce s obrazem je rozostření. Tohoto efektu lze v aplikaci Processing dosáhnout několika způsoby. Jedním z nich je použití funkce *filter*. Druhou možností je použití matice rozostření na pixely obrázku. Součástí příkladu bude ukázka použití obou těchto způsobů.

Rozostření – Funkce – Processing

```
PImage img;

void setup() {
  size(640, 360, P2D);
  img = loadImage("alaska.jpg");
  image(img, 0, 0);
  filter(BLUR, 2);
}


void draw() {
}
```

Obrázek 33 Blur pomocí funkce

Rozostření – Matice – Processing

```
float value = 1.0/9.0;
float[][] matrix = {{ value, value, value},
                    { value, value, value},
                    { value, value, value}};
PImage i1;

void setup(){
  size(640,360);
  i1 = loadImage("alaska.jpg");
}

void draw(){
    image(i1, 0, 0);
    // Nahrává informace o pixelech do Pixels[].
    i1.loadPixels();

    PImage i2 = createImage(i1.width, i1.height, RGB);

    for (int y = 1; y < i1.height-1; y++){ // Projde pixely shora dolů, přeskočí horní a spodní pixel.
      for (int x = 1; x < i1.width-1; x++) { // Projde pixely zleva doprava, přeskočí levý a pravý první pixel.
        float Red = 0; // Barvy pro následný součet získaných hodnot po průchodu tabulky.
        float Green = 0;
        float Blue = 0;

        // Průchod nejbližšími pixely od -1 po 1 v ose x a y.
        for (int adjy = -1; adjy <= 1; adjy++){
          for (int adjx = -1; adjx <= 1; adjx++){
            // Pozice sousedních pixelů pro daný pixel.
            int coord = (y + adjy)*i1.width + (x + adjx);

            // Výpočet barev podle hodnot v tabulce a barevné hodnoty na pozici sousedního pixelu.
            Red += matrix[adjy+1][adjx+1] * red(i1.pixels[coord]);
            Green += matrix[adjy+1][adjx+1] * green(i1.pixels[coord]);
```

```
            Blue += matrix[adjy+1][adjx+1] * blue(i1.pixels[coord]);
          }
        }
        // Změna barvy pixelu v tabulce pixelů.
        i2.pixels[y*i2.width + x] = color(Red, Green, Blue);
      }
    }
    i2.updatePixels();

    image(i2, 0, 0);
}
```

Obrázek 34 Blur pomocí matice

7.5.2 Konvoluce

Konvoluce otevírá dveře mnoha možnostem práce s obrazem. Dochází při ní k úpravám hodnot pixelů v závislosti na zvolené matici efektu a sousedních pixelů.

Konvoluce – Processing

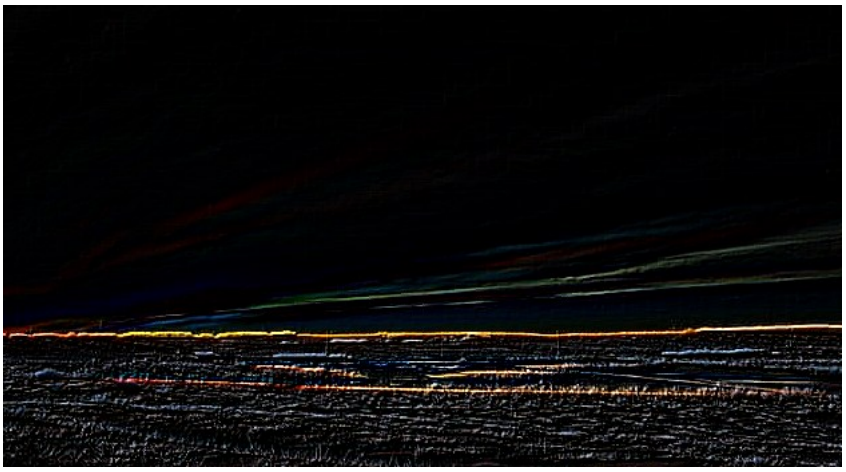
```
// Matice pro zostření.
float kernel[] = {
  0, -1, 0,
  -1,  5, -1,
  0, -1, 0
};

void setup() {
  size(640, 360);
  PImage img = loadImage("alaska.jpg");
  PImage sharpened = convolution(img);
  image(sharpened, 0, 0);
}

// Funkce pro konvoluci.
PImage convolution(PImage image) {
  int kernelRadius = int(sqrt(kernel.length) / 2);
  int width = image.width;
```

```
  int height = image.height;
  PImage output = createImage(width, height, RGB);

  for (int x = kernelRadius; x < width - kernelRadius; x++) {
    for (int y = kernelRadius; y < height - kernelRadius; y++) {
      float r = 0;
      float g = 0;
      float b = 0;

      for (int i = -kernelRadius; i <= kernelRadius; i++) {
        for (int j = -kernelRadius; j <= kernelRadius; j++) {
          float k = kernel[(i + kernelRadius) * kernelRadius + j +
kernelRadius];
          color c = image.get(x + i, y + j);
          r += red(c) * k;
          g += green(c) * k;
          b += blue(c) * k;
        }
      }

      r = constrain(r, 0, 255);
      g = constrain(g, 0, 255);
      b = constrain(b, 0, 255);
      output.set(x, y, color(r, g, b));
    }
  }

  return output;
}
```

Obrázek 35 Konvoluce pro ostření obrazu

7.5.3 Detekce hran

Detekce hran můžeme docílit použitím kódu z předchozího příkladu s názvem *Konvoluce*. V daném programu lze změnit základní matici, čímž dojde k naprosto odlišnému efektu programu. Nová matice pro detekci hran bude: $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$.



Obrázek 36 Detekce hran

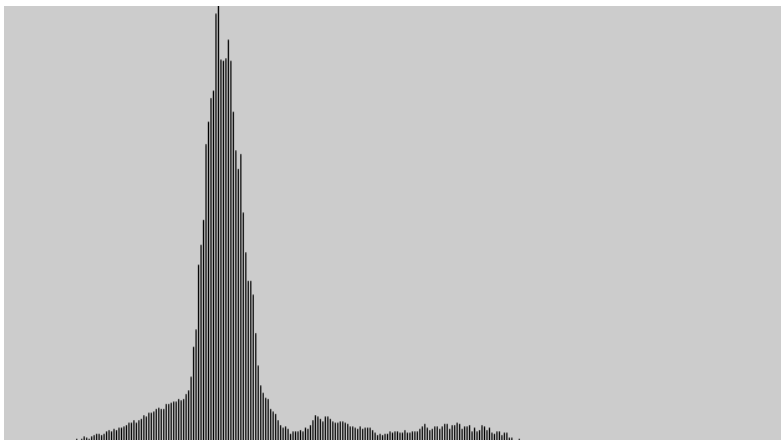
7.5.4 Histogram

Histogram představuje graf hodnot výskytu světelnosti v obrazu. Processing nedisponuje předdefinovanou funkcí pro výpočet a zobrazení histogramu, v základu však obsahuje řadu funkcí pro práci s barvou a pixely. V následujícím příkladu se spočítané hodnoty zanesou do pole a následně zobrazí jako graf hodnot.

Histogram – Processing

```
PImage img;
int[] histogram = new int[256]; // Pole hodnot histogramu.

void setup() {
  size(640, 360);
  img = loadImage("alaska.jpg");
  img.loadPixels();

  // Výpočet histogramu.
  for (int i = 0; i < img.pixels.length; i++) {
    int pixel = img.pixels[i];
    int brightness = (int) (red(pixel) * 0.299 + green(pixel) * 0.587 +
blue(pixel) * 0.114);
    histogram[brightness]++;
  }

  // Vypsání histogramu do grafu.
  int maxHeight = 0;
```

```
  for (int i = 0; i < histogram.length; i++) {
    if (histogram[i] > maxHeight) {
      maxHeight = histogram[i];
    }
  }
  float binWidth = width / histogram.length;
  for (int i = 0; i < histogram.length; i++) {
    float y = height - (histogram[i] * height / maxHeight);
    line(i * binWidth, height, i * binWidth, y);
  }
}
```



Obrázek 37 Histogram

7.6 Barvy a gradient

7.6.1 Odstín

Processing podporuje řadu funkcí pro práci čistě s barvou. Některé z nich jsou uvedeny v následujícím příkladu, který je doplněn o změnu odstínu v závislosti na uživatelském inputu klávesnice.

Odstín – Processing

```
color c;
float hue = 0;

void setup() {
  size(500, 500);
  c = color(255, 255, 255, 255); // Bíla barva
}

void draw() {
  background(255);
  c = color(255, 255, 255, 255); // Bíla barva
  c = color(hue, 255, 255); // Nastavení barvy
  fill(c);
  noStroke();
```

```
  ellipse(width/2, height/2, 200, 200);
}

void keyPressed() {
  if (keyCode == LEFT) {
    hue -= 10;
    if (hue < 0) {
      hue += 360;
    }
  } else if (keyCode == RIGHT) {
    hue += 10;
    if (hue >= 360) {
      hue -= 360;
    }
  }
}
```


Obrázek 38 Odstín

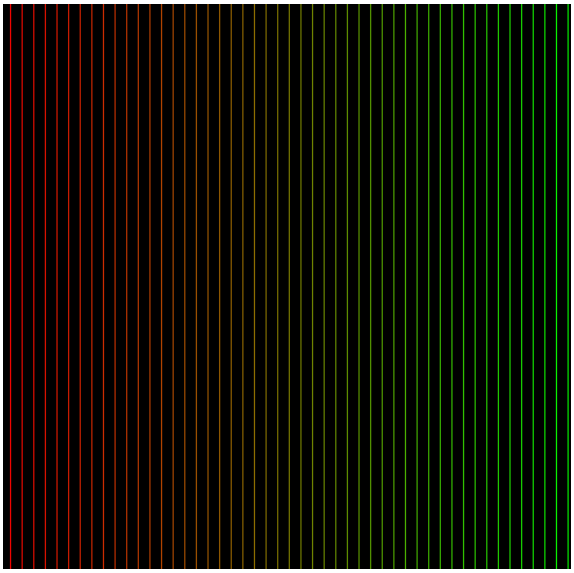
7.6.2 Gradient

Gradient představuje postupnou změnu odstínu a typu barvy v určeném obrázku. Tohoto efektu můžeme docílit pomocí úprav barev v cyklech. Řešení je nastíněno v tomto příkladu.

Gradient – Processing

```
void setup() {
  size(500, 500);
  background(0);
}

void draw() {
  for (int x = 0; x <= width; x += 10) {
    float inter = map(x, 0, width, 0, 1);
    color c1 = color(255, 0, 0); // Red
    color c2 = color(0, 255, 0); // Green
    color c = lerpColor(c1, c2, inter);
    stroke(c);
```

```
    line(x, 0, x, height);
  }
}
```



Obrázek 39 Lineární gradient

8 PRÁCE SE ZVUKEM

Tato část práce je zaměřena na práci se zvukem a jeho vizualizaci s využitím softwaru Processing. Jsou zde obsaženy příklady s použitím knihovny Sound a dalších nástrojů pro práci s audiem, kterými je aplikace vybavena.

8.1 Tvorba a přidání zvuku

8.1.1 Načtení zvukového souboru

Jedna ze základních vlastností práce se zvukem je schopnost přečíst zvukový soubor. Této funkcionality lze docílit pomocí funkcí v knihovně Sound a Minim. Zvolený zvukový soubor musí být vložen ve složce programu a jeho název musí být upraven v závislosti na programu.

Načtení zvukového souboru – Processing

```
import ddf.minim.*;

float a = 0;
Minim minim;
AudioPlayer player;

void setup() {
  minim = new Minim(this);
  player = minim.loadFile("audio.mp3");
  player.play();
}

void draw() {
}
```

8.1.2 Nahrávání zvuku

Druhou esenciální složkou úpravy zvuku je možnost načtení zvuku za běhu programu a následné uložení do souboru. V této sekci jsou připraveny dvě ukázky s rozdílným účelem.

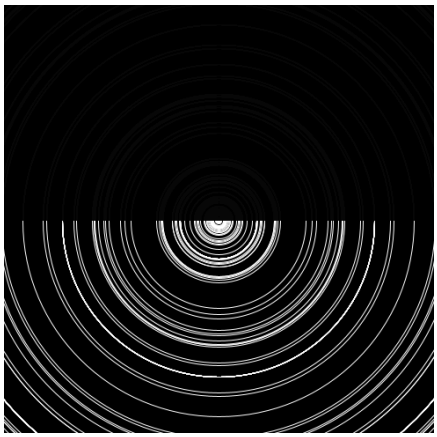
První ukázka snímá zvuk z mikrofonu a vizuálně převádí jeho vstup do kruhu, který se v horní polovině neustále aktualizuje, zatímco ve spodní polovině zobrazuje již dosažené hodnoty zvuku.

Druhá ukázka zvýrazňuje záznam zvuku pomocí pohybu vln a umožňuje nahrávání a následné uložení požadovaného úseku.

Nahrávání zvuku – Záznam – Processing

```
import ddf.minim.*;
import ddf.minim.ugens.*;
```

```
Minim minim;
AudioInput in;

void setup() {
  size(500,500);
  smooth();

  minim = new Minim (this);
  in = minim.getLineIn(Minim.STEREO,512);

  background(0);
}

void draw() {
  fill(0, 16);

  // Vykreslení zvuku.
  noStroke();
  rect(0, 0, width, height/2);
  stroke(255);
  noFill();
  float r = 0;
  for (int i = 0; i < in.bufferSize(); i++) {
    r += abs(in.mix.get(i)) * 20;
  }
  ellipse(width/2, height/2, r, r);
}

void stop() {
  in.close();
  minim.stop();
  super.stop();
}
```



Obrázek 40 Zaznamenání zvuku

Nahrávání zvuku – Uložení – Processing

```
import ddf.minim.*;
import ddf.minim.ugens.*;
```

```
Minim minim;
AudioInput input;
AudioOutput output;
FilePlayer player;
AudioRecorder record;
boolean recorded;

void setup() {
  size(500, 180, P3D);

  minim = new Minim(this);
  input = minim.getLineIn(Minim.STEREO, 2048);
  record = minim.createRecorder(input, "record.wav");
  output = minim.getLineOut( Minim.STEREO );

  textFont(createFont("Calibri", 10));
}

void draw() {
  background(100);
  stroke(200);

  // Kreslení zvuku.
  for(int i = 0; i < input.left.size()-1; i++)
  {
    line(i, 50 + input.left.get(i)*50, i+1, 50 + input.left.get(i+1)*50);
    line(i,     150     +     input.right.get(i)*50,     i+1,     150     +
input.right.get(i+1)*50);
  }

  if (record.isRecording())
  {
    text("Nahrávání! R pro zastavení nahrávání.", 5, 15);
  }
  else if (!recorded)
  {
    text("R pro nahrání.", 5, 15);
  }
  else
  {
    text("S pro uložení nahrávky.", 5, 15);
  }
}

// Při stisku tlačítka.
void keyReleased() {
  if (!recorded && key == 'r')
  {
    if (record.isRecording())
    {
      record.endRecord();
      recorded = true;
    }
    else
    {
```

```
      record.beginRecord();
    }
  }
  if (recorded && key == 's')
  {
    if (player != null)
    {
        player.unpatch(output);
        player.close();
    }
    player = new FilePlayer(record.save());
    player.patch(output);
    player.play();
  }
}
```

Obrázek 41 Nahrávání zvuku

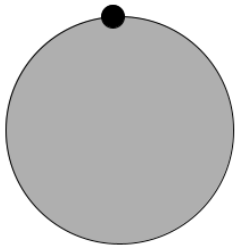
8.1.3 Tvorba oscilátoru

Oscilátor může být v mnoha případech nedílnou součástí několika zvukových nástrojů. Aplikace Processing je vybavena řadou funkcí, které disponují jednoduchou implementací a zobrazením těchto vln a oscilací.

Tvorba oscilátoru – Processing

```
import processing.sound.*;

SinOsc oscillator;
float frequency;
float amp;
float speedFactor;

void setup() {
  size(400, 400);

  oscillator = new SinOsc(this);
  oscillator.play();
  frequency = 440; // Frekvence
  amp = 0.5; // Amplituda
  speedFactor = 0.0001;
}

void draw() {
  background(255);
```

```
  // Výpočet bodů elipsy.
  float x = width / 2 + sin(frameCount * speedFactor * frequency) * 100;
  float y = height / 2 + cos(frameCount * speedFactor * frequency) * 100;

  // Vykreslení oscilátoru.
  stroke(0);
  fill(175);
  ellipse(width/2, height/2, 200, 200);
  fill(0);
  ellipse(x, y, 20, 20);

  // Změna frekvence oscilátoru podle pozice myši.
  frequency = map(mouseX, 0, width, 100, 1000);
  oscillator.freq(frequency);
  speedFactor = map(frequency, 100, 1000, 0.001, 0.05);
}
```



Obrázek 42 Oscilátor

8.2 Filtry a transformace

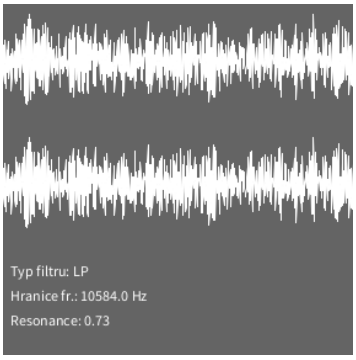
8.2.1 Tvorba jednoduchého filtru

Z nejznámějších filtrů lze zmínit Low Pass, High Pass a Band Pass. Tyto funkce jsou implementovány i v oficiální knihovně Sound od Processing, ale také v knihovně Minim jako Moog Filter.

Tvorba jednoduchého filtru – Processing

```
import ddf.minim.*;
import ddf.minim.ugens.*;

Minim minim;
MoogFilter moog;
AudioOutput output;

void setup() {
  size(300, 300);
```

```
  minim = new Minim(this);
  output = minim.getLineOut();

  // Filtr s hranicí frekvence 1200 a 0.5 resonancí.
  moog = new MoogFilter(1200, 0.5);

  // Filtrace hluku.
  Noise noize = new Noise(0.5f);

  noize.patch(moog).patch(output);
}

// Změna typu filtru klávesnicí.
void keyPressed() {
  if (key == 'q') moog.type = MoogFilter.Type.LP;
  if (key == 'w') moog.type = MoogFilter.Type.HP;
  if (key == 'e') moog.type = MoogFilter.Type.BP;
}

void draw() {
  background(100);
  stroke(255);

  // Vykreslení vln.
  for(int i = 0; i < output.bufferSize() - 1; i++)
  {
    float x1 = map(i, 0, output.bufferSize(), 0, width);
    float x2 = map(i+1, 0, output.bufferSize(), 0, width);

    line(x1, 50 + output.left.get(i)*50, x2, 50 + output.left.get(i+1)*50);
    line(x1,     150     +     output.right.get(i)*50,     x2,     150     +
output.right.get(i+1)*50);
  }

  text("Typ filtru: " + moog.type, 10, 225);
  text("Hranice fr.: " + moog.frequency.getLastValue() + " Hz", 10, 245);
  text("Resonance: " + moog.resonance.getLastValue(), 10, 265);
}

// Nastavení frekvence a resonance pomocí pohybu myši.
void mouseMoved() {
  float freq = constrain(map(mouseX, 0, width, 200, 12000 ), 200, 12000);
  float rez = constrain(map(mouseY, height, 0, 0, 1 ), 0, 1);

  moog.frequency.setLastValue(freq);
  moog.resonance.setLastValue(rez);
}
```



Obrázek 43 Filtr zvuku

8.2.2 FFT

Fast Fourier Transformation má velkou řadu využití v oblasti obrazu a zvuku. Implementace byla již provedena v knihovnách a její použití lze aktivovat pomocí funkce *FFT*.

FFT – Processing

```
import processing.sound.*;

FFT fft;
AudioIn in;
int bands = 512;
float[] spectrum = new float[bands];

void setup() {
  size(512, 360);
  background(255);

  // Vstupní linka připojená k analyzéru FFT.
  fft = new FFT(this, bands);
  in = new AudioIn(this, 0);

  in.start();

  fft.input(in);
}

void draw() {
  background(255);
  fft.analyze(spectrum);

  for(int i = 0; i < bands; i++){
  // Výsledek FFT je normalizován.
  // Zvětšení amplitudy o 5 bodů.
  line( i, height, i, height - spectrum[i]*height*5 );
  }
}
```

Obrázek 44 FFT

8.3 Tvorba efektů

8.3.1 Echo

Efekt Echo je nejjednodušším zvukovým efektem, přesto se zatím nejedná o efekt přímo zavedený do základních funkcí Processingu. Tohoto efektu lze dosáhnout pomocí funkce *Delay* a mixování výstupů zvuku. Příklad aplikuje efekt Echo na zvuk snímaný z mikrofonu v reálném čase.

Echo – Processing

```
import processing.sound.*;

AudioIn in;
AudioIn in2;
Delay effect;

void setup() {
  size(640,360);
  background(255);

  // Vstupní linky. Vstup 2 se přehraje hned, na vstup 1 se aplikuje Delay
a Amp změna.
  in = new AudioIn(this, 0);
  in2 = new AudioIn(this, 0);
  in2.play();

  // Tvorba Delay efektu.
  effect = new Delay(this);

  in.play();
  in.amp(0.5);

  effect.process(in, 5);
  effect.time(0.5);
}

void draw() {
}
```

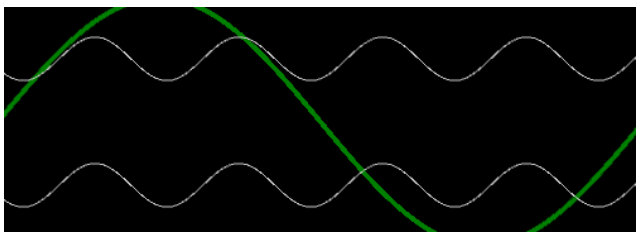
8.3.2 Vibrato a Tremolo

Efekty Tremola a Vibrata fungují na podobném principu. Liší se v cíleném bodě jejich modulací. Následující příklad využívá změnu amplitudy a frekvence různých typů vln oscilace, aby dosáhl modulací v hlasitosti a výšce tónu.

Vibrato a Tremolo – Processing

```
import ddf.minim.*;
import ddf.minim.effects.*;
import ddf.minim.ugens.*;

Minim minim;
Oscil wave;
Gain gain;
AudioInput in;
AudioOutput out;

void setup()
{
  size(512, 200, P3D);
  minim = new Minim(this);

  // Vstupní linka.
  in = minim.getLineIn();

  // Výstupní linka.
  out = minim.getLineOut();

  // Tvorba sinové vlny.
  wave = new Oscil(440, 0.5f, Waves.SINE);

  gain = new Gain(0);
  gain.patch(out);
  wave.patch(gain);
}

void draw()
{
  background(0);
  stroke(200);
  strokeWeight(1);

  for(int i = 0; i < out.bufferSize() - 1; i++)
  {
    line(i, 50  - out.left.get(i)*50,  i+1, 50  - out.left.get(i+1)*50);
    line(i, 150 - out.right.get(i)*50, i+1, 150 - out.right.get(i+1)*50);
  }

  // Vykreslení vlny oscilátoru.
  stroke(0, 128, 0);
  strokeWeight(4);
  for(int i = 0; i < width-1; ++i)
  {
```

```
    point(i, height/2 - (height*0.49) * wave.getWaveform().value((float)i
/ width));
  }
}

void keyPressed()
{
  switch(key)
  {
    case 'q':
      wave.setWaveform(Waves.SINE);
      break;

    case 'w':
      wave.setWaveform(Waves.QUARTERPULSE);
      break;

    case 'e':
      wave.setWaveform(Waves.SQUARE);
      break;

    case 'r':
      wave.setWaveform(Waves.SAW);
      break;

    case 't':
      wave.setWaveform(Waves.TRIANGLE);
      break;

    default: break;
  }
}

void mouseMoved()
{
  // Úprava amplitudy a frekvence pomocí polohy myši.
  float amp = map(mouseY, 0, height, 1, 0);
  wave.setAmplitude(amp);

  float freq = map(mouseX, 0, width, 110, 880);
  wave.setFrequency(freq);
}
```



Obrázek 45 Vibrato a Tremolo

8.3.3 Reverb

Efekt Reverb je jedním z prvních efektů, které jsou již přímo zabudovány do prostředí Processingu. Můžeme jej zavolat pomocí funkce a typu *Reverb*. Následující příklad aplikuje tento efekt na zvuk snímaný z mikrofonu v reálném čase.

Reverb – Processing

```
import processing.sound.*;

AudioIn input;
Reverb effect;

void setup() {
  size(500,500);
  background(255);

  // Vstupní linka.
  input = new AudioIn(this, 0);

  // Tvorba Reverb efektu.
  effect = new Reverb(this);

  input.play();
  effect.process(input);
}

void draw() {
}
```

8.3.4 Stereo Panning

Stereo Panning označuje změnu hlasitostí odlišných mono zvuků jedné stereo stopy. Tohoto lze docílit použitím funkce *Pan* pro změnu hlasitosti jedné ze stran stereo stopy. V následujícím příkladu se změna hlasitosti provádí na zvukové stopě v reálném čase v závislosti na poloze myši.

Stereo Panning – Processing

```
import ddf.minim.*;
import ddf.minim.ugens.*;

Minim minim;
AudioPlayer player;

void setup() {
  size(400, 400);

  minim = new Minim(this);
  player = minim.loadFile("audio.mp3");
  player.play();
}
```

```
// Pozice myši udává změnu Panningu.
void draw() {
  background(255);
  float panpos = map(mouseX, 0, width, -1.0, 1.0);
  player.setPan(panpos);
}

void stop() {
  player.close();
  minim.stop();
  super.stop();
}
```

8.4 Vizualizace zvuku

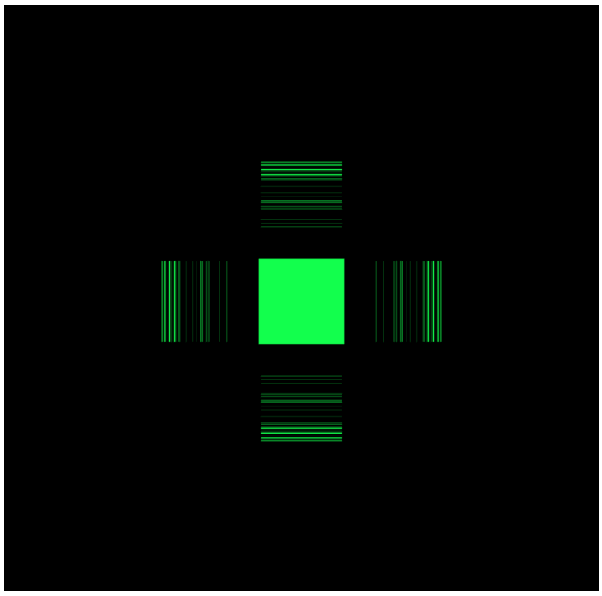
8.4.1 Vizualizace hlasitosti zvuku

Vizualizace zvuku představuje podstatnou část integrace obrazu a zvuku. Processing je vybaven řadou funkcí, které tuto práci usnadňují. Lze použít funkce zmíněné v sekcích praktické částí ohledně práce s obrazem a audiem. Může být použito dvourozměrných tvarů, nebo trojrozměrných tvarů s eliminací jednoho rozměru, jako tomu je v tomto příkladu.

Vizualizace hlasitosti zvuku – Processing

```
import ddf.minim.*;

float a = 0;
Minim minim;
AudioPlayer player;

void setup() {
  size(800, 800, P3D);
  smooth(8);
  minim = new Minim(this);
  player = minim.loadFile("audio.mp3");
  player.play();
}

void draw() {
  background(0);
  translate(400,400);

  for (int i = 0; i < player.bufferSize() - 1; i++) {
     pushMatrix();
     fill(#12FF4D, 150);
     popMatrix();
     strokeWeight(1+player.right.get(i));

     stroke(0);
     box(100, 100, 55+player.right.get(i)*200);
     box(55+player.right.get(i)*500, 100, 100);
     box(100, 55+player.right.get(i)*500, 100);
```

```
  }
}
```



Obrázek 46 Vizualizace zvuku 2D

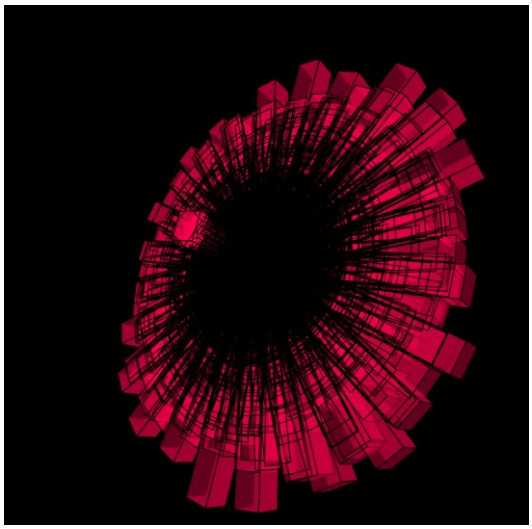
8.4.2 Vizualizace hlasitosti zvuku ve 3D

Vizualizace ve 3D funguje na stejném principu jako vizualizace zvuku ve 2D. Je nutné specifikovat render mód na *P3D*. V následujícím příkladu byla také implementována otočná kamera, která umožňuje interakci s pohybem myši.

Vizualizace hlasitosti zvuku ve 3D – Processing

```
import peasy.*;
import ddf.minim.*;

float a = 0;
PeasyCam cam;
Minim minim;
AudioPlayer player;

void setup() {
  size(800, 800, P3D);
  smooth(8);
  cam = new PeasyCam(this, 180);
  cam.setMinimumDistance(50);
  cam.setMaximumDistance(500);
  minim = new Minim(this);
  player = minim.loadFile("audio.mp3");
  player.play();
}

void draw() {
  background(0);
```

```
  rotateY(.5);
  rotateZ(.5);

  for (int i = 0; i < player.bufferSize() - 1; i++) {
     rotateX(50+player.right.get(i)/100);

     pushMatrix();
     fill(#FF004D, 100);
     popMatrix();
     strokeWeight(1+player.right.get(i));

     stroke(0);
     box(10, 10, 55+player.right.get(i)*200);
     box(55+player.right.get(i)*50, 10, 10);
     box(10, 55+player.right.get(i)*50, 10);
  }
}
```



Obrázek 47 Vizualizace zvuku 3D

9 NAHRÁVÁNÍ A ÚPRAVA VIDEA

V následující sekci jsou uvedeny příklady a ukázky spojené s nahráváním a úpravou videí.

9.1 Přidání a tvorba souboru videa

9.1.1 Načtení videa

Načtení videa z kamery je v aplikaci Processing vcelku jednoduché. Lze využít typu *Capture*, pomocí kterého dojde k propojení aplikace Processing s webovou kamerou.

Načtení videa – Processing

```
import processing.video.*;

Capture webcam;

void setup() {
    size(600,400);
    smooth();

    // Pro případ několika kamer.
    println(Capture.list());

    webcam = new Capture(this, width, height, 30);
    webcam.start();
}

void draw() {
  background(200);
  image(webcam,0,0);
}

void captureEvent(Capture webcam) {
  webcam.read();
}
```

9.1.2 Nahrávání videa

Nahrávání videa už není tak přímočaré jako ostatní postupy. Nejsou zde vytvořené funkce pro přímé uložení videa. Tento nedostatek se dá překonat několika způsoby.

Prvním z nich je použití funkce *saveFrame*. Pomocí této funkce se při běhu programu uloží snímky obrazu, který skript vykreslil. Následně lze použít již předinstalovaný nástroj Movie Maker, abychom sekvenci obrázků přeměnili na video.

Druhý způsob obnáší instalaci knihovny *VideoExport*. Použití funkcí přidaných touto knihovnou umožní převod obrázkové sekvence v reálném čase. Pro její správnou funkci je třeba instalovat modul *FFMPEG* a vložit jej do proměnného prostředí a cesty počítače.

Nahrávání videa – Processing

```
//import com.hamoid.*;
import processing.video.*;

Capture webcam;
//VideoExport videoExport;

void setup() {
    size(600,400);
    smooth();

    println(Capture.list());

    webcam = new Capture(this, width, height, 30);
    webcam.start();

    //videoExport = new VideoExport(this);
    //videoExport.startMovie();
}

void draw() {
  background(200);
  image(webcam,0,0);
  saveFrame("frame-####.jpg");
  //videoExport.saveFrame();
}

void captureEvent(Capture webcam) {
  webcam.read();
}
```

9.2 Úpravy videa

9.2.1 Přehrání videa

Processing je vybaven typem a funkcemi pro rychlé přehrání videí. Lze toho dosáhnout pomocí typu *Movie*.

Přehrání videa – Processing

```
import processing.video.*;

Movie movie;

void setup() {
  size(600,400);
```

```
  movie = new Movie(this, "movie.mp4");
  movie.loop();
}

void movieEvent(Movie movie) {
  movie.read();
}

void draw() {
  image(movie, 0, 0);
}
```

9.2.2 Úprava videa během záznamu

Processing podporuje přístup k pixelům snímaného záznamu, tímto způsobem lze v reálném čase upravit snímaný obraz. Následující ukázka pouze upravuje barvu pixelů, bylo by však možné také použít již uvedené postupy ze sekce praktické části o obrázcích.

Úprava videa během záznamu – Processing

```
import processing.video.*;

Capture video;

void setup() {
  size(640, 360);

  video = new Capture(this, 640, 360);
  video.start();
}

void captureEvent(Capture video) {
  video.read();
}

void draw() {
  loadPixels();
  video.loadPixels();

  for (int x = 0; x < video.width; x++) {
    for (int y = 0; y < video.height; y++) {
      int loc = x + y * video.width;

      float r = red(video.pixels[loc]);
      float g = green(video.pixels[loc]);
      float b = blue(video.pixels[loc]);

      // Lze použít stejné či jiné principy ze sekce praktické části o
obrázcích.
      color c = color(r*0.5, g*0.8, b*0.5);
      pixels[loc] = c;
    }
```

```
  }

  updatePixels();
}
```

9.2.3 Úprava souboru videa

Úpravy již vzniklého souboru videa lze docílit kombinací předchozích dvou příkladů této kapitoly. Po načtení souboru pomocí typu a funkcí pro *Movie*, můžou být příslušné pixely obrazu upraveny.

Úprava souboru videa – Processing

```
import processing.video.*;

Movie movie;

void setup() {
  size(600,400);

  movie = new Movie(this, "movie.mp4");
  movie.loop();
}

void movieEvent(Movie movie) {
  movie.read();
}

void draw() {
  image(movie, 0, 0);

  loadPixels();
  movie.loadPixels();

  for (int x = 0; x < movie.width-1; x++) {
    for (int y = 0; y < movie.height-1; y++) {
      int loc = x + y * movie.width;

      float r = red(movie.pixels[loc]);
      float g = green(movie.pixels[loc]);
      float b = blue(movie.pixels[loc]);

      color c = color(r*0.5, g*0.5, b*0.8);
      pixels[loc] = c;
    }
  }

  updatePixels();
}
```

10 PRÁCE S EMBEDDED SYSTÉMY

V této kapitole jsou zpracovány základy vizualizace procesů s využitím embedded systémů. Příklady obsahují náčrt zapojení obvodů použitých pro získaní informací z daných systémů, kód použitý na platformě, kód ze softwaru Processing a vzhled výsledné vizualizace. Kapitola je dále rozdělena na menší části dle použitého systému.

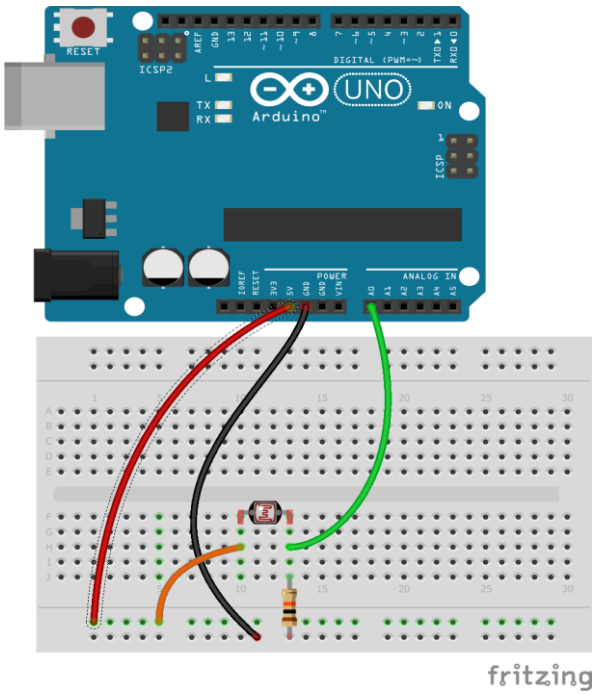
10.1 Arduino

Součástí této sekce jsou ukázky příkladů pro vizualizaci procesů s použitím Arduina, prostředí Arduino IDE a softwaru Processing.

10.1.1 Fotorezistor

K vytvoření obvodu sloužícího k příkladům s použitím informací z fotorezistoru budeme potřebovat několik součástek:

- Arduino Uno
- Experimentální deska a kabely na propojení
- Fotorezistor
- 5k Ω či 10k Ω rezistor



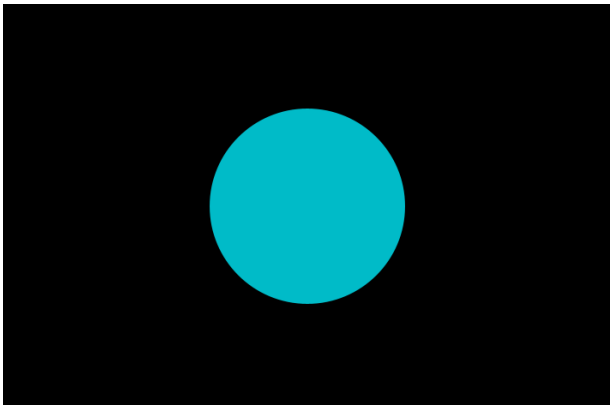
Obrázek 48 Zapojení fotorezistoru

Fotorezistor – Arduino IDE

```
unsigned int ADCValue;

void setup() {
  // Otevře sériovou komunikaci s přenosovou rychlostí 9600.
  Serial.begin(9600);
}

void loop() {
  // Přečte data a pošle je po sériové komunikace.
  int val = analogRead(0);
  val = map(val, 0, 300, 0, 255);
  Serial.println(val);
  delay(50);
}
```

Fotorezistor – Processing

```
import processing.serial.*;

Serial port;
int sensorVal = 0;
static String val;

void setup()
{
  size(720, 480);
  noStroke();
  noFill();

  String portName = "COM3";
  port = new Serial(this, portName, 9600);
}

void draw()
{
  if (port.available() > 0) {
    val = port.readStringUntil('\n');
    try {
       sensorVal = Integer.valueOf(val.trim());
    }
    Catch (Exception e) {
      ;
    }
    println(sensorVal);
  }
  background(0);

  float c = map(sensorVal, 0, width, 0, 400);
  float d = map(sensorVal, 0, width, 40,500);

  fill(0, c, 200);
  ellipse(width/2, height/2, d/2+100, d/2+100);
}
```

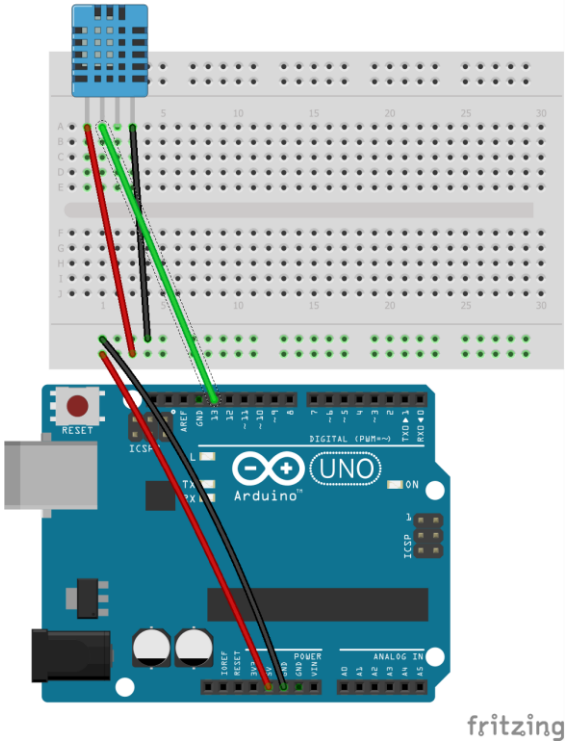


Obrázek 49 Fotorezistor

10.1.2 Teplota a vlhkost

K vytvoření obvodu sloužícího k příkladům s použitím informací ze senzoru DHT11, který získává údaje ohledně teploty a vlhkosti vzduchu, budeme potřebovat několik součástek:

- Arduino Uno
- Experimentální deska a kabely na propojení
- Senzor DHT11



Obrázek 50 Zapojení DHT11

V tomto příkladu odešleme informace ze senzoru DHT11 po sériové lince. V aplikaci Processing spustíme vypracovaný program, který vytvoří grafické rozhraní pro intuitivní zobrazení těchto dat.

Teplota a vlhkost – Arduino IDE

```
#include <DHT.h>

#define DHTPIN 13
#define DHTTYPE DHT11
DHT dht(DHTPIN,DHTTYPE);

void setup() {
  Serial.begin(9600);
  dht.begin();

  Serial.println("Humidity (%) Temperature (C)");
}

void loop() {
  delay(100);
  float humidity = dht.readHumidity();
  float temp = dht.readTemperature();
  Serial.print(temp);
  Serial.print(",");
  Serial.print(humidity);
  Serial.print("\n");
}
```

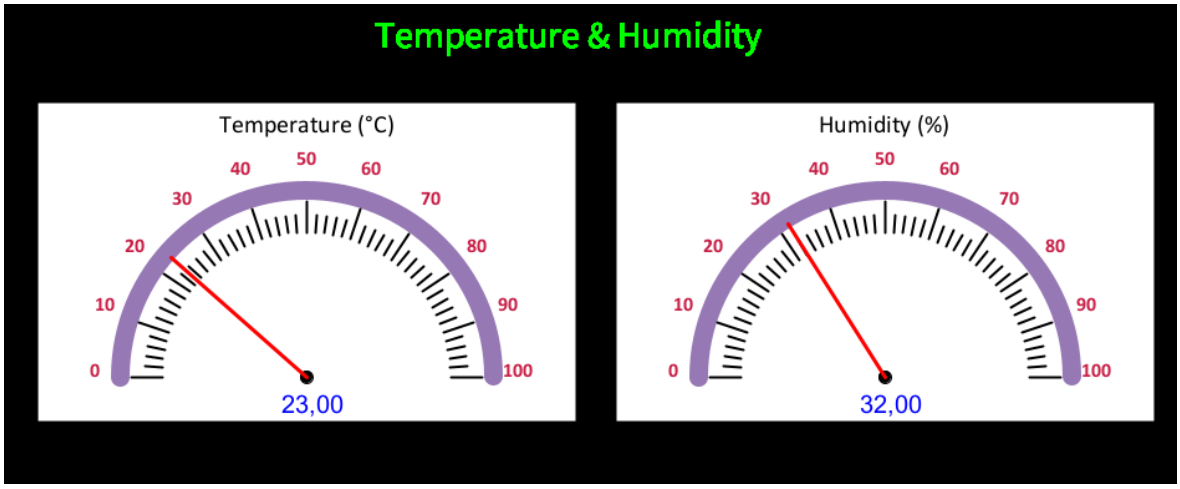
K použití této ukázky je nutné nainstalovat knihovnu Meter.

Teplota a vlhkost – Processing

```
import meter.*;
import processing.serial.*;

Serial port;

Meter meter1, meter2;

void setup(){
  size(1000, 400);
  background(0,0,0);

  port = new Serial(this, "COM3", 9600);

  // Metr teploty.

  meter1 = new Meter(this, 25, 80);
  meter1.setTitleFontSize(20);
  meter1.setTitleFontName("Calibri");
  meter1.setTitle("Temperature (°C)");
```

```
  String[]                              scaleTemp                              =
{"0","10","20","30","40","50","60","70","80","90","100"};
  meter1.setScaleLabels(scaleTemp);
  meter1.setScaleFontSize(16);
  meter1.setScaleFontName("Calibri bold");
  meter1.setScaleFontColor(color(200,30,70));

  meter1.setDisplayDigitalMeterValue(true);
  meter1.setArcColor(color(150,120,180));
  meter1.setArcThickness(15);

  meter1.setMaxScaleValue(100);
  meter1.setMinInputSignal(0);
  meter1.setMaxInputSignal(100);

  meter1.setNeedleThickness(3);

  // Metr vlhkosti.

  int m1x = meter1.getMeterX();
  int m1y = meter1.getMeterY();
  int m1width = meter1.getMeterWidth();

  meter2= new Meter(this, 25 + m1x + m1width, m1y);
  meter2.setTitleFontSize(20);
  meter2.setTitleFontName("Calibri");
  meter2.setTitle("Temperature (°C)");

  String[]                           scaleHumidity                             =
{"0","10","20","30","40","50","60","70","80","90","100"};
  meter2.setScaleLabels(scaleHumidity);
  meter2.setScaleFontSize(16);
  meter2.setScaleFontName("Calibri bold");
  meter2.setScaleFontColor(color(200,30,70));

  meter2.setDisplayDigitalMeterValue(true);
  meter2.setArcColor(color(150,120,180));
  meter2.setArcThickness(15);

  meter2.setMaxScaleValue(100);
  meter2.setMinInputSignal(0);
  meter2.setMaxInputSignal(100);

  meter2.setNeedleThickness(3);
}

void draw(){
  textSize(30);
  fill(0,255,0);
  text("Temperature & Humidity",250,40);

  if (port.available() > 0){
      String value = port.readString();

      String[] values = split(value,",");
      float temp = float(values[0]);
```

```
        float humidity = float(values[1]);

        meter1.updateMeter(int(temp));
        meter2.updateMeter(int(humidity));
    }
}
```



Obrázek 51 Teplota a vlhkost

10.1.3 Matice Arduina

Novější verze Arduina jsou vybaveny maticí LED světel. V ukázce se využívá přenosu dat po sériové lince z Processing do Arduina ke změně rozsvíceného obrázku.

Matice Arduina – Arduino IDE

```
#include "Arduino_LED_Matrix.h"

ArduinoLEDMatrix matrix;
char val;

void setup() {
  Serial.begin(9600);
  matrix.begin();
}

const uint32_t happy[] = {
    0x19819,
    0x80000001,
    0x81f8000
};
const uint32_t heart[] = {
    0x3184a444,
    0x44042081,
    0x100a0040
};

 void loop() {
```

```
    if (Serial.available()) {
      val = Serial.read();
    }

    // Načte frame srdce do matice při charakteru 1. Jinak frame obličeje.
    if (val == '1') {
     matrix.loadFrame(heart);
    } else {
     matrix.loadFrame(happy);
    }

    delay(10);
}
```

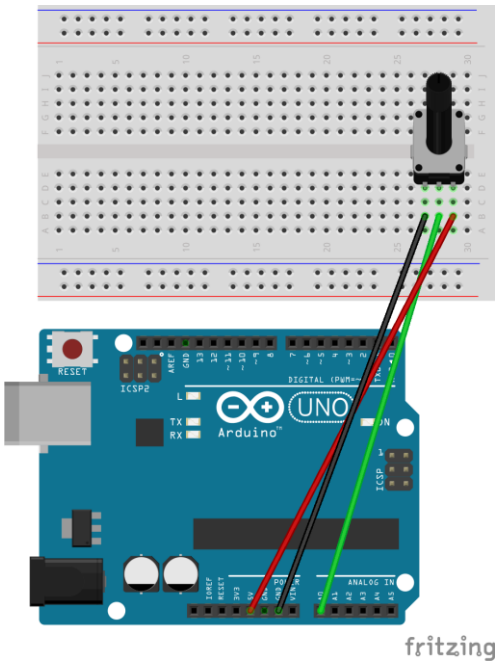
Matice Arduina – Processing

```
import cc.arduino.*;
import org.firmata.*;
import processing.serial.*;

Serial port;

void setup()
{
  size(600, 300);
  noStroke();
  noFill();

  textSize(128);
  text("q - Happy", 50, 120);
  fill(0, 400, 600);
  text("w - Heart", 50, 250);

  String portName = "COM3";
  port = new Serial(this, portName, 9600);
}

void draw(){
}

// Odešle po sériové komunikaci
void keyPressed() {
  if (key == 'q') {port.write('0'); println('0');}
  if (key == 'w') {port.write('1'); println('1');}
}
```

10.1.4 Potenciometr

K vytvoření obvodu sloužícího k příkladům s použitím informací z potenciometru, který získává údaje z napětí v závislosti na poloze otočného mechanismu, budeme potřebovat několik součástek:

- Arduino Uno
- Experimentální deska a kabely na propojení
- Potenciometr 1 kΩ



Obrázek 52 Zapojení potenciometru

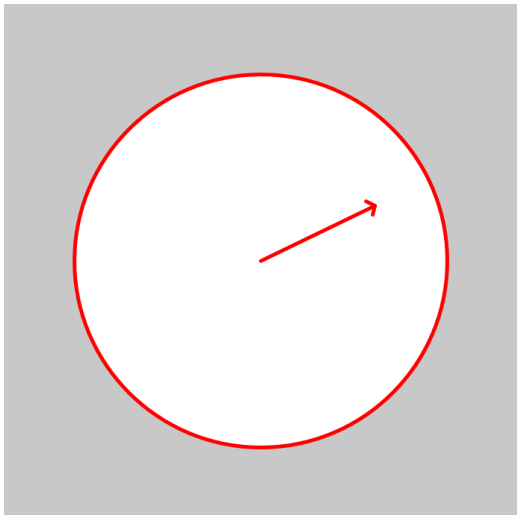
Potenciometr – Arduino IDE

```
#define POTENCIOMETR_PIN A0

int num;

void setup(){
  Serial.begin(9600);
}

void loop(){
  num = (analogRead(POTENCIOMETR_PIN)/4);
  Serial.write(num);
  delay(100);
}
```

Potenciometr – Processing

```
import processing.serial.*;

Serial myPort;

float mapval;
int val;
```

```
void setup(){
  size(700, 700);
  myPort = new Serial(this, "COM3", 9600);
}

void draw(){
  if (myPort.available() > 0) {
    val = myPort.read();
    mapval = map(val,0,255,0,6.500);

    background(200);
    imageMode(CENTER);

    ellipse(350,350, 500, 500);
    translate(350,350);
    rotate(mapval);

    strokeWeight(5);
    stroke(255,0,0);
    drawArrow(0,0,170,270);

    println(val);
  }
}

void drawArrow(int x, int y, int len, float ang) {
  pushMatrix();
  translate(x,y);
  rotate(radians(ang));

  line(0,0,len,0);
  line(len,0,len-8,-10);
  line(len,0,len-8,10);
  popMatrix();
}
```



Obrázek 53 Potenciometr

10.2 STM32

Na platformě STM32 můžou být aplikovány stejné příklady jako u kapitoly Arduina, se stejným kódem bylo dosaženo stejných výsledků u kapitoly Fotorezistor a Potenciometr.

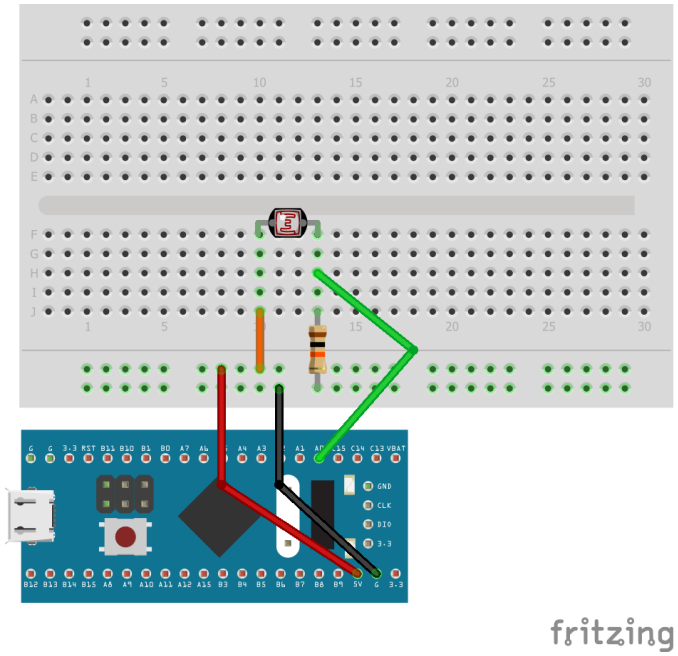
V daných příkladech je pouze nutné změnit pin, do kterého jsou naše vstupy připojeny a port komunikace. Zapojení STM32 k těmto příkladům bude uvedeno v této sekci práce.

STM32 můžeme ovládat pomocí Arduino IDE, je však nutné doinstalovat ovladač desky a nastavit jej ve vývojovém prostředí. Pro nahrání programu do STM32 je také nutné změnit přepínač *boot0* na hodnotu 1. Při použití převodníku USB na UART sběrnici připojíme piny 3V3 s pinem 3V3 na STM32, GND s pinem GND na STM32, TXD s pinem A10 a RXD s pinem A9. Poté můžeme použít sériovou metodu nahrání, či změnit bootloader čipu tak, aby bylo možné nahrávat kód pomocí USB.

10.2.1 Fotorezistor

K vytvoření obvodu sloužícího k příkladům s použitím informací z fotorezistoru budeme potřebovat několik součástek:

- STM32
- Experimentální deska, převodník a kabely na propojení
- Fotorezistor
- 5k Ω či 10k Ω rezistor

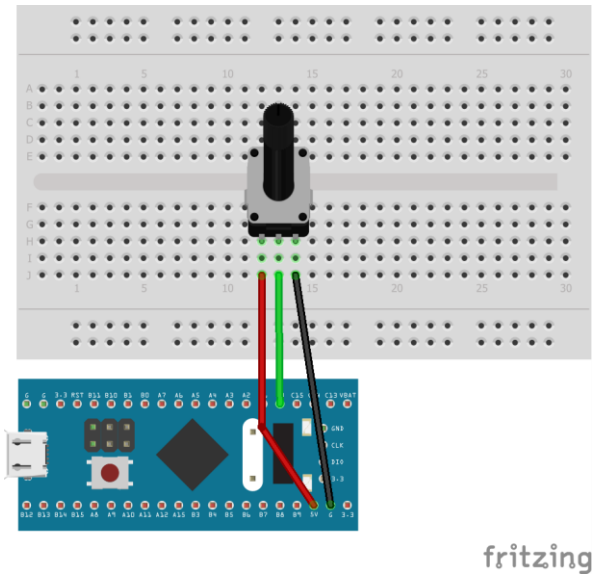


Obrázek 54 Zapojení fotorezistoru na STM32

10.2.2 Potenciometr

K vytvoření obvodu sloužícího k příkladům s použitím informací z potenciometru, který získává údaje z napětí v závislosti na poloze otočného mechanismu, budeme potřebovat několik součástek:

- STM32
- Experimentální deska, převodník a kabely na propojení
- Potenciometr 1 kΩ



Obrázek 55 Zapojení potenciometru na STM32

10.3 Raspberry Pi

V následující sekci jsou uvedeny příklady s použitím aplikace Processing na platformě Raspberry Pi.

10.3.1 Raspberry Cam

K příkladu je nutné připojit kamerový modul přímo k Raspberry Pi, které již má dedikovaný kamerový port. Pro propojení kamery s aplikací Processing se využívá knihovna *GLVideo*. Tato knihovna je v nové verzi aplikace nefunkční, je proto nutné stáhnout starší verzi aplikace Processing, nebo počkat až ji autor aktualizuje. Běžná knihovna *Video* není na Raspberry Pi podporována.

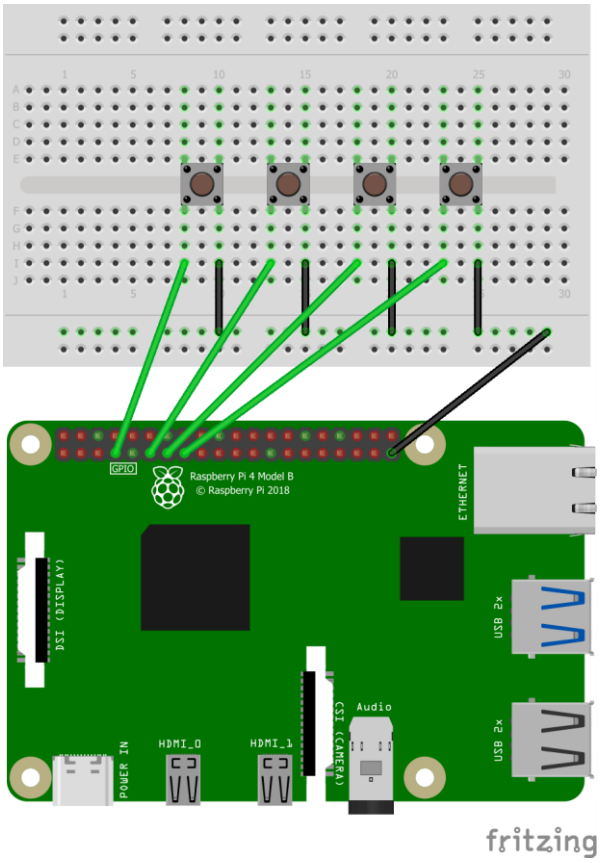
Raspberry Cam – Processing

```
import gohai.glvideo.*;
GLCapture video;

void setup() {
  size(320, 240, P2D);

  video = new GLCapture(this);
  video.start();
}

void draw() {
  background(0);
  if (video.available()) {
    video.read();
  }
  image(video, 0, 0, width, height);
}
```

10.3.2 Tlačítka

K vytvoření obvodu sloužícího k příkladům s tlačítky, které ovládají vykreslování kruhu, budeme potřebovat několik součástek:

- Raspberry Pi
- Experimentální deska a kabely na propojení
- 4 či více tlačítek



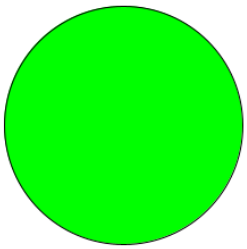
Obrázek 56 Zapojení tlačítek

V aplikaci Processing na Raspberry Pi je nutné doinstalovat knihovnu Hardware I/O.

Tlačítka – Processing

```
import processing.io.*;

int csize = 200;
color ccolor = color(255,0,0);
int[] pins = {4, 17, 27, 22};

void setup() {
  size(500, 500);

  for (int i = 0; i < pins.length; i++) {
    GPIO.pinMode(pins[i], GPIO.INPUT_PULLUP);
  }
}

void draw() {
  background(255);

  if (GPIO.digitalRead(pins[0]) == GPIO.LOW) {
    ccolor = color(0,0,255);
  }
  if (GPIO.digitalRead(pins[1]) == GPIO.LOW) {
    ccolor = color(0,255,0);
  }
```

```
  if (GPIO.digitalRead(pins[2]) == GPIO.LOW) {
    csize += 10;
  }
  if (GPIO.digitalRead(pins[3]) == GPIO.LOW) {
    csize -= 10;
  }

  fill(ccolor);
  ellipse(250,250,csize,csize);
}
```



Obrázek 57 Kruh ovládaný tlačítky

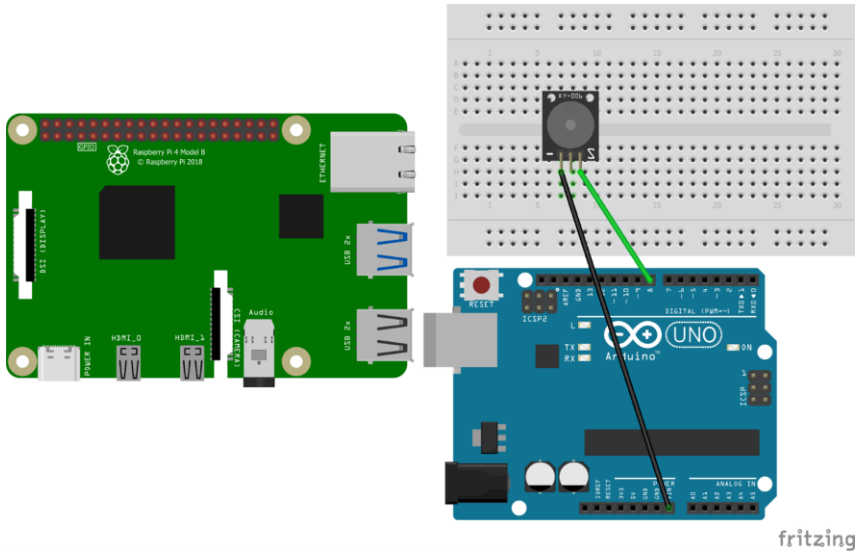
10.4 Propojení systémů

V tomto finálním příkladu dojde k propojení Arduina a Raspberry Pi za účelem spojení předešlých obrazových a zvukových kapitol. Účelem je ukázat některé pokročilé možnosti aplikace Processing a její integrace. K ukázce byl v aplikaci vytvořen model klaviatury s určenými klávesami, které korespondují se zvolenými tóny v programu pro Arduino. Arduino převezme po sériové komunikaci odeslané hodnoty z Raspberry Pi a na jejich základě zahraje požadovanou notu.

K vytvoření obvodu budeme potřebovat několik součástek:

- Raspberry Pi
- Arduino Uno
- Experimentální deska a kabely na propojení
- USB kabel

- KY-006 pasivní bzučák, nebo jiný typ bzučáku



Obrázek 58 Zapojení Arduina a Raspberry Pi k příkladu

V nákresu není znázorněno propojení Arduina a Raspberry, protože aplikace Fritzing neumožňuje propojení přes USB. Komunikace může být provedena pomocí USB kabelu přímo mezi Raspberry Pi a Arduinem. Jako další možnost je propojení použitím RX a TX pinů na těchto dvou systémech. V tomto případě je nutné redukovat napětí Arduina z 5V na 3.3V, aby nedošlo k poškození Raspberry Pi.

Kód z příkladu by bylo možné také aplikovat při propojení Arduina například se stolním počítačem. Bylo by však nutné změnit port komunikace v programu aplikace Processing a nastavení v prostředí Arduino IDE.

Ukázka využívá hlavičkový soubor *pitches.h* od Mika Putnama. Tento soubor je volně dostupný ke stažení například z GitHubu. Daný soubor poté musí být přiložen ve stejné složce jako program k Arduinu.

Propojení systémů – Arduino IDE

```
#include "pitches.h"

// Pole s notami
int melody[] = {
  NOTE_C4,  NOTE_CS4,  NOTE_D4,  NOTE_DS4,  NOTE_E4,  NOTE_F4,  NOTE_FS4,
NOTE_G4, NOTE_GS4, NOTE_A4, NOTE_AS4, NOTE_B4
};
int val;

void setup() {
  Serial.begin(9600);
}

void loop() {
  if (Serial.available()) {
     val = Serial.read();
   }

  if (val < 12) {
    switch (val) {
      case 0:
        tone(8, melody[0], 50);
        break;
      case 1:
        tone(8, melody[1], 50);
        break;
      case 2:
        tone(8, melody[2], 50);
        break;
      case 3:
        tone(8, melody[3], 50);
        break;
      case 4:
        tone(8, melody[4], 50);
        break;
      case 5:
        tone(8, melody[5], 50);
        break;
      case 6:
        tone(8, melody[6], 50);
        break;
      case 7:
        tone(8, melody[7], 50);
        break;
      case 8:
        tone(8, melody[8], 50);
        break;
      case 9:
        tone(8, melody[9], 50);
        break;
      case 10:
        tone(8, melody[10], 50);
        break;
      case 11:
```

```
        tone(8, melody[11], 50);
        break;
      default:
        break;
    }
  }

  val = 12;
  delay(10);
}
```

Propojení systémů – Processing

```
import cc.arduino.*;
import org.firmata.*;
import processing.serial.*;

Serial port;

void setup()
{
  size(712,300);
  background(0);
  noStroke();

  // Tvorba klaviatury
  fill(255);
  rect(0,0,100,300);
  rect(102,0,100,300);
  rect(204,0,100,300);
  rect(306,0,100,300);
  rect(408,0,100,300);
  rect(510,0,100,300);
  rect(612,0,100,300);

  fill(0);
  rect(50,0,80,165);
  rect(176,0,80,165);
  rect(356,0,80,165);
  rect(468,0,80,165);
  rect(584,0,80,165);

  textSize(128);
  text("a", 20, 265);
  text("s", 125, 265);
  text("d", 220, 265);
  text("f", 335, 265);
  text("g", 423, 265);
  text("h", 525, 265);
  text("j", 650, 265);

  fill(255);
  textSize(100);
  text("w", 54, 150);
  text("e", 190, 150);
```

```
  text("t", 378, 150);
  text("z", 488, 150);
  text("u", 595, 150);

  String portName = "/dev/ttyACM0"; // Port pro Raspberry
  //String portName = "COM3"; // Port pro Windows
  port = new Serial(this, portName, 9600);
}

void draw(){
}

// Odešle po sériové komunikaci
void keyPressed() {
  if (key == 'a') {port.write(0); println('0');}
  if (key == 'w') {port.write(1); println('1');}
  if (key == 's') {port.write(2); println('2');}
  if (key == 'e') {port.write(3); println('3');}
  if (key == 'd') {port.write(4); println('4');}
  if (key == 'f') {port.write(5); println('5');}
  if (key == 't') {port.write(6); println('6');}
  if (key == 'g') {port.write(7); println('7');}
  if (key == 'z') {port.write(8); println('8');}
  if (key == 'h') {port.write(9); println('9');}
  if (key == 'u') {port.write(10); println("10");}
  if (key == 'j') {port.write(11); println("11");}
}
```

Obrázek 59 Klaviatura k příkladu

ZÁVĚR

Tato bakalářská práce se zabývala průzkumem možností v oblastech vizuálního a zvukového přenosu spojeného s prací v softwarovém nástroji Processing. V práci jsou obsaženy popisy všech zvolených vývojových prostředí. Dále jsou nastíněny základní principy práce s obrazem, audiem a také propojení s využitými embedded systémy. V rámci výstupů z praktické časti práce byly ověřeny popsané koncepty, na kterých tyto obrazové a zvukové efekty stojí.

Nástroj Processing obsahuje řadu funkcí, které se při tvorbě příkladů k praktické části ukázaly jako užitečné. Pokud během tvorby práce nebylo nalezeno jednoduché a přímé řešení vybraného problému, většinou pomohlo prozkoumaní oficiálních či komunitních knihoven. Tyto knihovny pokrývají velkou řadu témat například od nastavení kamer až po websockety. Aplikace Processing se také ukázala jako velmi interaktivní nástroj. Práce s interaktivními částmi skriptu dosáhla rychlého a účinného zapojení uživatele do programu.

Následné propojení aplikace s vybranými embedded systémy proběhlo přímočaře. Aplikace byla přizpůsobena pro rychlou a intuitivní integraci například s Arduinem či Raspberry Pi. Komunikace s Arduinem byla realizována po sériové lince použitím knihovny Firmata. Implementace pro Raspberry Pi může být stažena přímo do zařízení, odkud může být využívána.

Je důležité zdůraznit, že i přes všechny možnosti, kterými nástroj Processing disponuje, je stále prostor pro další experimentování, inovace a objevování nových technik a přístupů. V digitálním umění se neustále vyvíjí nové trendy a technologie, v důsledku čehož je nutné být otevřen novým možnostem a být připravený adaptovat případně změny.

V závěru bych rád poděkoval všem, jejichž díla a přístup k audiovizuální tvorbě splnila funkci inspirace při psaní této práce. Věřím, že tato práce poskytne cenné informace a případně dále zvýší zájem budoucí generace umělců a vývojářů v oblasti audiovizuálního umění.

SEZNAM POUŽITÉ LITERATURY

[1] FRY, Ben, Casey REAS a Dan SHIFFMAN. Processing [online]. 2021 [cit. 2023-10-27]. Dostupné z: https://processing.org/

[2] VANTOMME, Jan. Processing 2: Creative Programming Cookbook. Birmingham: Packt Publishing Ltd., 2012. ISBN 978-1-849517-94-2.

[3] SHIFFMAN, Daniel. Learning Processing: a beginner's guide to programming images, animation, and interaction. Boston: Morgan Kaufmann/Elsevier, 2008. ISBN 978-0-12-373602-4.

[4] Getting Started with Arduino IDE 2. SÖDERBY, Karl a Jacob HYLÉN. Arduino Docs [online]. 2021, 17.1.2024 [cit. 2023-11-11]. Dostupné z: https://docs.arduino.cc/software/ide-v2/tutorials/getting-started-ide-v2/

[5] SÖDERBY, Karl. Downloading and installing the Arduino IDE 2. Online. Arduino Docs. 2021, 28. 2. 2024. Dostupné z: https://docs.arduino.cc/software/ide-v2/tutorials/getting-started/ide-v2-downloading-and-installing/. [cit. 2023-11-11].

[6] WHAT IS FRITZING, HOW DOES IT WORK AND HOW TO USE IT? Online. Soldered. 2023, 9. 1. 2023. Dostupné z: soldered.com/learn/what-is-fritzing-how-does-it-work-and-how-to-use-it/. [cit. 2023-11-13].

[7] Download Fritzing. Online. Fritzing. 2023, 2. 1. 2024. Dostupné z: https://fritzing.org/download/. [cit. 2023-11-13].

[8] SYSOS. Úsečka. Online. Multimediaexpo. 2022. Dostupné z: http://www.multimediaexpo.cz/mmecz/index.php/%C3%9Ase%C4%8Dka. [cit. 2024-02-24].

[9] Čtverec a obdélník. Online. Umíme matiku. 2017. Dostupné z: https://www.umimematiku.cz/cviceni-obdelnik-ctverec. [cit. 2024-02-15].

[10] Trojúhelník. Online. Umíme matiku. 2017. Dostupné z: https://www.umimematiku.cz/book/cviceni-trojuhelnik. [cit. 2024-04-17].

[11] KALVODA, Tomáš. Kružnice a elipsa. Online. Přípravný kurz matematiky. 2021, 25. 6. 2021. Dostupné z: https://kam.fit.cvut.cz/deploy/bi-pkm/mirror/textbook/sec-kruznice-elipsa.html. [cit. 2024-02-21].

[12] Objemy a povrchy dalších typů hranolů. Online. Portál středoškolské matematiky. 2010. Dostupné z: https://www.karlin.mff.cuni.cz/~portal/objemyaobsahy/?page=Objemyapovrchydalsichhranolu. [cit. 2024-02-21].

[13] Koule. Online. VOJÁČEK, Jakub. Matematika pro každého. 2008. Dostupné z: https://maths.cz/clanky/85-koule. [cit. 2024-02-24].

[14] SYSOP. Křivka. Online. Multimediaexpo. 2022. Dostupné z: http://www.multimediaexpo.cz/mmecz/index.php/K%C5%99ivka. [cit. 2024-02-29].

[15] NAWAZ, Bisma. What is a Bézier curve? Online. Educative. 2024. Dostupné z: https://www.educative.io/answers/what-is-a-bzier-curve. [cit. 2024-02-12].

[16] B-spline, NURBS křivky. Online. Západočeská univerzita v Plzni. 2015. Dostupné z: https://home.zcu.cz/~bastl/GM1/GM1_lecture05.pdf. [cit. 2024-02-24].

[17] FELKER, Petr. Křivky a plochy I. Online. DCGI. 2016. Dostupné z: https://cw.fel.cvut.cz/old/_media/courses/b0b39pgr/10-krivky.pdf. [cit. 2024-02-13].

[18] B-Spline. Online. Wolfram Mathworld. 2024. Dostupné z: https://mathworld.wolfram.com/B-Spline.html. [cit. 2024-02-13].

[19] ŠÍR, Zbyněk. Racionální křivky a NURBS. Online. Matematický ústav UK. 2024. Dostupné z: https://www.karlin.mff.cuni.cz/~sir/NPGR021/gm06racionalniKrivky.pdf. [cit. 2024-02-14].

[20] Matice. Online. Matematika polopatě. 2006. Dostupné z: https://www.matweb.cz/matice/. [cit. 2024-02-23].

[21] Inverzní matice. Online. Matematika polopatě. 2006. Dostupné z: https://www.matweb.cz/inverzni-matice/. [cit. 2024-02-23].

[22] What is image blurring? Online. Educative. 2024. Dostupné z: https://www.educative.io/answers/what-is-image-blurring. [cit. 2024-01-20].

[23] Obrazová konvoluce. Online. VŠB – Technická univerzita Ostrava. 2024. Dostupné z: https://www.cs.vsb.cz/licev/lzs%20I_cviceni/Cv_5_konvoluce.pdf. [cit. 2024-01-21].

[24] What is Edge Detection – An Introduction. Online. Great Learning. 2022. Dostupné z: https://www.mygreatlearning.com/blog/introduction-to-edge-detection/. [cit. 2024-01-17].

[25] ŽÁRA, Jiří. Moderní počítačová grafika. 2., přeprac. a rozš. vyd. Brno: Computer Press, 2004. ISBN 8025104540.

[26] TEKALP, A. Murat. Digital video processing. Upper Saddle River, NJ: Prentice Hall PTR, 1995. ISBN 0-13-190075-7.

[27] OTRUBA, Vítězslav. Barvy. Online. IS Masarykovy univerzity. 2014. Dostupné z: https://is.muni.cz/el/1431/jaro2014/C6150/um/Barvy.pdf. [cit. 2024-01-16].

[28] ANGELICA, Laura. What is Gradient and The Best Way to Use Gradients in Your Design. Online. Mockitt. 2023. Dostupné z: https://mockitt.wondershare.com/dictionary/gradient.html. [cit. 2024-01-15].

[29] Co jsou barvy a jak mezi nimi uspět bez námahy? Online. Trigama. 2022. Dostupné z: https://www.trigama.eu/cs/blog/detail/co-jsou-barvy-a-jak-mezi-nimi-uspet-bez-namahy. [cit. 2024-01-16].

[30] Lekce 2 - Úvod do počítačové grafiky – Základy optiky, barevné modely. Online. ITNetwork. 2024. Dostupné z: https://www.itnetwork.cz/grafika/uvod/uvod-do-pocitacove-grafiky-optika-modely. [cit. 2024-01-16].

[31] Vector vs Raster Graphics. Online. GeeksforGeeks. 2022. Dostupné z: https://www.geeksforgeeks.org/vector-vs-raster-graphics. [cit. 2024-01-27].

[32] What is image compression? Online. SHELDON, Robert. TechTarget. 2022. Dostupné z: https://www.techtarget.com/whatis/definition/image-compression. [cit. 2024-01-28].

[33] Native File Formats. Online. University of Michigan Library. 2024. Dostupné z: https://guides.lib.umich.edu/c.php?g=282942&p=1885348. [cit. 2024-01-29].

[34] The Guide to Vector Design. Online. CorelDRAW. 2024. Dostupné z: https://www.coreldraw.com/en/learn/guide-to-vector-design/vector-file-types/. [cit. 2024-01-29].

[35] What Are the Different Types of Video Formats. Online. Mailchimp. 2024. Dostupné z: https://mailchimp.com/resources/video-formats. [cit. 2024-02-02].

[36] CHRISTENSEN, Mads G. Introduction to audio processing. New York, NY: Springer Science+Business Media, 2019. ISBN 978-3-030-11780-1.

[37] Embedded Systems. Online. HeavyAI. 2024. Dostupné z: https://www.heavy.ai/technical-glossary/embedded-systems. [cit. 2023-11-15].

[38] What is Arduino? Online. Arduino Docs. 2022. Dostupné z: https://docs.arduino.cc/learn/starting-guide/whats-arduino/. [cit. 2023-11-18].

[39] VODA, Zbyšek. Průvodce světem Arduina. Vydání druhé. Bučovice: Martin Stříž, 2017. ISBN 978-80-87106-93-8.

[40] A Beginner's Guide to Developing on STM32. Online. BISWAL, Sanskar. Medium. 2020. Dostupné z: https://medium.com/theteammavericks/a-beginners-guide-to-developing-on-stm32-b7fd38966aa0. [cit. 2023-11-15].

[41] What Is Raspberry Pi? Models, Features, and Uses. Online. BASUMALLICK, Chiradeep. Spiceworks. 2022. Dostupné z: https://www.spiceworks.com/tech/networking/articles/what-is-raspberry-pi/. [cit. 2023-11-28].

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

PCB	Printed Circuit Board
DHT11	Senzor teploty a vlhkosti vzduchu
IDE	Integrated Development Environment
ADC	Analog-to-Digital Converter
DAC	Digital-to-Analog Converter
SNR	Signal-to-Noise Ratio
IIR	Infinite Impulse Response
FFT	Fast Fourier Transform
ARM	Advanced Risk Machine
NEMA	National Engineering Manufacturers Association
UART	Universal Asynchronous Receiver/Transmitter
SPI	Seriál Peripheral Interface
RGB	Red, Green, Blue
CMYK	Cyan, Magenta, Yellow, Key
HSL	Hue, Saturation, Lightness
2D	Two-dimensional
3D	Three-dimensional
GPIO	General Purpose Input/Output
NURBS	Non-Uniform Rational B-Spline
LED	Light-emitting diode

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

PŘÍLOHA P I: SD KARTA

Přiložená SD karta obsahuje:

- Bakalářskou práci ve formátu .pdf — BP_KoplLukas_2024.pdf
- Skripty a výstupy v adresáři — Results
- Skripty a výstupy ve formátu .zip — Results.zip
- Informační soubor k obsahu SD karty — Info.txt