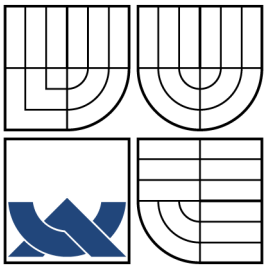
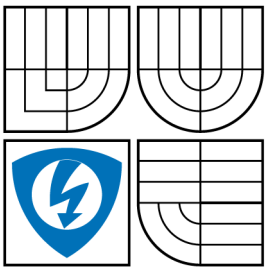


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND
COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

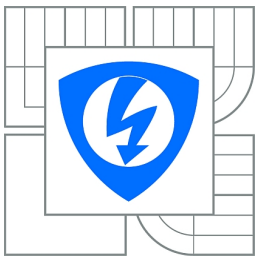
# ŽIVÝ FONT
LIVE TYPEFACE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. MILAN JIŘÍČEK
AUTHOR

VEDOUCÍ PRÁCE — Mgr. PAVEL RAJMIC, Ph.D.
SUPERVISOR

BRNO 2012



**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**

**Fakulta elektrotechniky a komunikačních technologií**

**Ústav telekomunikací**

# Diplomová práce

magisterský navazující studijní obor
**Telekomunikační a informační technika**

| | | | |
|---|---|---|---|
| ***Student:*** | Bc. Milan Jiříček | ***ID:*** | 84337 |
| ***Ročník:*** | 2 | ***Akademický rok:*** | 2011/2012 |

**NÁZEV TÉMATU:**

## Živý font

**POKYNY PRO VYPRACOVÁNÍ:**

Nastudujte způsob definice obrysových (tzv. outline) fontů pomocí Bézierových křivek. Navrhněte metodologii, která by do fontu vnesla "život" formou promyšlené a řízené náhodnosti. Navrhněte omezenou znakovou sadu, na které budou testovány nové algoritmy. Vámi zvolený algoritmus implementujte v Matlabu, JAVA či C, odlaďte a zhodnoťte účinek svého přístupu.

**DOPORUČENÁ LITERATURA:**

[1] Cabarga, L. Logo, Font & Lettering Bible. How Design Books, 2004, ISBN 1581804369

[2] Prautzsch, H. et al. Bezier and B-Spline Techniques. Springer, 2010, ISBN 3642078427

[3] Doňar, B., Zaplatílek K., MATLAB pro začátečníky, BEN - technická literatura, vydání 2. aktualizované, ISBN 80-7300-175-6

| | | | |
|---|---|---|---|
| ***Termín zadání:*** | 6.2.2012 | ***Termín odevzdání:*** | 24.5.2012 |

***Vedoucí práce:*** Mgr. Pavel Rajmic, Ph.D.
***Konzultanti diplomové práce:***

**prof. Ing. Kamil Vrba, CSc.**
*Předseda oborové rady*

**UPOZORNĚNÍ:**

## ABSTRAKT

Práce ve svém začátku popisuje historický vývoj písma, dále pak typografická pravidla využívaná při návrhu znakových sad, a také především teoretické základy tvorby nových fontů moderními metodami, tedy za použití informačních technologií, konkrétně vektorových programů. Okrajově se věnuje i Římskému písmu, dále základnímu rozdělení skladby písem a popisuje také jeho jednotlivé prvky. Další část projektu se zaměřuje na tvorbu digitálního fontu s návazností na jeho oživení za použití algoritmů a metod, které se snadno dají zpracovat v programovacím jazyce Matlab. Popsán je zde i způsob realizace oživení fontu a prezentace vytvořených výsledků je ukázána na několika příkladech. Dále je pozornost věnována soustavě souřadnic, která je důležitá při řešení geometrických transformací, a dvourozměrné vektorové grafice, která se hojně využívá právě při návrhu a vykreslení písma. Detailněji jsou popsány Bézierovy křivky a také jejich varianty Bézierovy kubiky společně s možnostmi rasterizace vektorové grafiky. Část textu je také věnována důležitému algoritmu de Casteljau. Poslední kapitola se zaměřuje na implementaci v programovém prostředí Matlab, tvorbu křivek, potažmo znaků, a způsoby práce s jednotlivými transformacemi. Zde lze také nalézt ukázky transformací při různém nastavení vstupních parametrů. Nakonec je zhodnocen přínos této aplikace a možnosti jejího dalšího rozšíření.

## KLÍČOVÁ SLOVA

font, typografie, písmo, znak, abeceda, Bézier, křivka, grafika, PostScript, OpenType, TrueType, verzálky, minusky, majuskule, minuskule, sazba, Times New Roman, Century, latinka, diakritika, náhodnost, živost, živý, patka, patky, serify, kurzíva, tučné

## ABSTRACT

The work describes the theoretical foundations of modern methods of creating new fonts, that is, using information technology, specifically vector programs. It also marginally shows the history of the font, Roman writing, the basic division of its compositions and describes its individual elements. The next part of the project focuses on creating digital fonts and continues with bringing it to live using algorithms and methods that can easily be processed in Matlab programming language. The method of implementation of the bringing the typeface to live is described and presentation of the generated results are then shown in few examples. Attention is also paid to the coordinates system, which is very important to solve geometric transformations, and two-dimensional vector graphics, which are widely used in the design and to render the fonts. Bézier curves and cubic Bézier curves are described in further detail along with vector graphics rasterization. One part of this text describes very important de Casteljau algorithm. The last chapter focuses on implementation in Matlab programming language, the creation of curves, that means single font characters, and the ways of how the algorithm works with transformations. Some sample images showing transformations using different input arguments are displayed in this section as well. There is evaluation of the contribution of this application and the possibilities of further expansion at the end.

## KEYWORDS

font, typography, writing, character, alphabet, Bézier, curve, graphics, PostScript, OpenType, TrueType, capitals, lower case, capital letters, lower case letters, composition, Times New Roman, Century, latin, diacritics, randomness, liveliness, lively, serif, italic, bold

JIŘÍČEK, Milan *Živý font*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2012. 71 s. Vedoucí práce byl Mgr. Pavel Rajmic, Ph.D.

# PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Živý font“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

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

Brno ..... .....

(podpis autora)

# PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Mgr. Pavlu Rajmicovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.


Brno ..... .....

(podpis autora)

# OBSAH

# SEZNAM OBRÁZKŮ

# SEZNAM TABULEK

# ÚVOD

Fonty jsou nedílnou součástí této doby a všichni se s nimi setkáváme v každodenní rutině, ať již jde o tištěnou nebo elektronickou formu. Jedná se totiž o velmi důležitý prvek v komunikaci člověka. Zvláště uvážíme-li trend nového tisíciletí, který vyžaduje efektivní a rychlé přenosy informací na jakékoliv vzdálenosti.

Dnes převažuje elektronická forma informace nad tou tištěnou. S rozmachem přenosných počítačů, chytrých telefonů a podobných zařízení se zvyšují nároky na prezentaci fontu na obrazovce. Jednotlivé zařízení se postupným vývojem stávají stále miniaturnějšími a tím pádem hůře čitelnými. Nejrůznější algoritmy na vyhlazování písma a tvorba nových, lépe čitelných druhů písem, mají za úkol tento problém co možná nejvíce odstínit — cílem typografie je totiž zajistit čtenáři snažší čtení.

Nesmíme také opomenout marketingovou stránku věci — velké korporace, ale i menší firmy se snaží ke svým zákazníkům promluvit přes nejrůznější média, která ve valné většině obsahují i textové informace, ať jde o kontaktní údaje, prodej zboží, případně oslovení stávajících klientů v dnešní pro podnikatele těžké době zasažené ekonomickou krizí. Prezentace potom může probíhat na televizním přijímači, počítačovém monitoru nebo velkoplošné obrazovce umístěné na rušné ulici velkoměsta. Opět je tedy důležitým prvek vhodná volba příjemně působícího fontu. Díky přesycenosti trhu s reklamou dnes totiž na člověka zapůsobí jen dobře mířená a kvalitně podaná informace.

V této práci se budeme věnovat způsobu definice obrysových (tzv. outline) fontů pomocí Bézierových křivek. Ve své první části ukáže metodologii, která do fontu vnese „život“ formou promyšlené a řízené náhodnosti. Praktická ukázka bude posléze předvedena ve druhé části práce na omezené znakové sadě pomocí vyhotovených algoritmů ve zvoleném programovacím jazyce, kterým bude aplikace Matlab.

Úvodní kapitoly se zaměří na základy typografie a dvourozměrné vektorové grafiky, blíže se také podíváme na bézierovy křivky, které jsou stěžejní pro tvorbu znakových sad v tomto projektu. Dále rozvineme tvorbu digitálního fontu na moderních počítačích a tuto část zakončím náznakem implementace — tedy tvorby algoritmů, které budou operovat se znakem k docílení náhodného efektu.

Poté přijde na řadu souřadný systém, důležitý prvek při práci se znaky. Ty totiž budou mimo jiné oživeny za pomoci geometrických transformací. Nakonec bude popsán způsob implementace v prostředí Matlab, které bylo vybráno jako nejvhodnější varianta. Nakonec se také zhodnotí dosažené výsledky a možnosti dalšího rozšíření a využití naprogramovaných algoritmů.

# 1 TYPOGRAFIE

## 1.1 Úvod do typografie

Písmo moderní doby, tedy tak, jak ho známe dnes, může nabývat několika odlišných významů. Od toho velmi obecného, neboli prostředku pro psanou a tištěnou komunikaci, až po označení jedné písmové rodiny — fontu.

Jestliže písmo splňuje beze zbytku svoji primární, tedy sdělovací funkci, a navíc má ještě zřetelnou estetickou hodnotu, promění se rázem z pouhého prostředku komunikace do svébytného uměleckého díla. To bude také jedním z cílů této práce — vtisknout pokud možno každému znaku neotřelou originalitu tak, aby byl svým způsobem jedinečný. Poté mohou slova a potažmo také celé věty dostat úplně jiný rozměr a působit živým dojmem.

V této kapitole se letmo podíváme na historický vývoj písma, poté rozebereme konstrukci písma a jeho typy, blíže prozkoumáme také písmovou osnovu, patkové písmo, vyrovnávání písma a konečně druhy písma.

## 1.2 Historický vývoj písma

Již v dávných dobách používali naši předkové nejrůznější formy kreseb jako způsob dorozumívání. Ať již šlo o hieroglyfy v Egyptě nebo pradávný knihtisk, vždy bylo úkolem písma předat nějaké příběhy nebo vědomosti dalším lidem. I proto se písmo dále rozvíjelo až do podoby, ve které ho známe dnes.

Historie dnešního písma tedy sahá okolo pěti tisíc let zpět. Na jejím začátku je snaha člověka zaznamenat myšlenky jeskynní malbou a tím si je uchovat pro pozdější potřeby. I když se obrázek postupně graficky zjednodušoval, znázorňoval celou větu (nebyl znakem pro slovo). Dalším stupněm byl ideogram vyjadřující kratší či delší řady hlásek. Například nohy označovaly končetiny nebo chůzi. K vyjádření myšlenky tak byla nutná znalost mnoha znaků. Posledním a prakticky nejdokonalejším stupněm vývoje bylo zaznamenávání hlásek (tyto zvuky nazýváme fonogramy).

Každá z kultur dospěla do různého stupně vývoje písma, což bylo ovlivněno jednak osobitostí jazyka, zeměpisou polohou, společenskými potřebami nebo také písmem sousedních národů. Na rychlost šíření písma a jeho grafickou stránku působily také psací materiály a vyvývejné nástroje.

### Římská abeceda

Jelikož se Římská abeceda používá dodnes, je historický příspěvek Říma k jazyku v celosvětovém pojetí velice rozsáhlý. V procesu romanizace byly vyvinuty jazyky,

| arabská čísla | 1 | 5 | 10 | 50 | 100 | 500 | 1000 | 2012 |
|---|---|---|---|---|---|---|---|---|
| římská čísla | I | V | X | L | C | D | M | MMXII |

Tab. 1.1: Srovnání římských čísel s klasickými arabskými čísly.

které pocházejí přímo z latiny a byly přijaty na mnoha místech světa ať už díky kolonizaci nebo kulturnímu vlivu. Dokonce i moderní angličtina převzala velkou část slovní zásoby z latiny. Římská nebo chcete-li latinská abeceda je na světě nejrozšířenější systém znaků používaný ke psaní a logicky ho tak používá nejvíce jazyků.

Její původní složení bylo 24 písmen odvozených z tvarů verzálek, které Římané tesali do kamene. Snaha o rychlejší psaní vedla ke vzniku tzv. kurzívy (currere — běžeti, z latiny), kterou každý z nás jistě zná, jelikož je nápadná svým šikmým sklonem.

V tabulce 1.1 vidíme ukázku sedmi písmen, které potom díky promyšlenému systému jejich skládání a kombinování používali pro číselná vyjádření a počty. V jejich výtču chybí nula, tu však tento systém nepotřebuje. Nula se začala objevovat až později díky arabským číslům.

### Moderní pojetí písma

Celkově existuje na světě okolo 400 druhů písem, z nichž nejrozšířenější je nám velmi dobře známá latinka, kde se při psaní kromě písmen používají i interpunkční znaménka (.,!). Některé jazyky, jako je čeština nebo němčina, mají navíc diakritická znaménka, jenž upravují výslovnost písmen.

Hlavně díky moderním technologiím se fonty postupem času zdokonalily a s příchodem počítačů se vyrojily spousty nových druhů písma, které měly za úkol jednak usnadnit čtení nebo zatraktivnit font jako nástroj pro prezentaci, což ve větší míře rozvádí kniha J. L. Dusonga [7].

Dostupnost nových efektivních nástrojů pro vektorový návrh grafiky umožňoval zkoušet nové fonty stále více lidem a internet napomohl jejich šíření.

### Číslice

Číslice představují zvláštní složku písemné soustavy. Určování počtu se v historii dělo za pomoci čárek, bodů nebo případně předmětů. Zdokonalený zápis číslic pochází z Indie z pátého století našeho letopočtu. Obsahuje znaky od nuly (0) do devítky (9). Název arabské číslice používáme, protože tuto soustavu posléze převzali Arabové.

#### Specializované typy písma

Pro nevidomé lidi je určeno Braillovo písmo, které vytvořil Louise Braille v roce 1835. Jedná se o reliéfní šestibodový zápis, jenž se vyráží na Pichtově stroji. Obsahuje znaky pro interpunkci, číslice, ale i noty. V běžném životě se s ním můžeme setkat například na obalech léků.

## 1.3 Konstrukce písma

Jako základní písmovou veličinu lze s určitou nadsázkou považovat abecedu. Je to totiž sada písmen v ustáleném pořadí, která se dále dělí na velkou a malou abecedu, viz obrázek 1.1.

VERZÁLKY minusky

Obr. 1.1: Písmena velké a malé abecedy.

Proporce písma určuje tzv. písmová osnova (podrobněji dále). Na kresbě písma se potom podílejí hlavní a vedlejší písmové tahy.

### 1.3.1 Forma písma

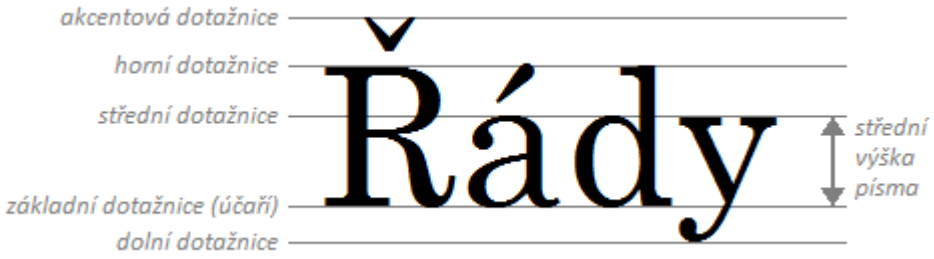
Velká písmena abecedy nazýváme verzálky (neboli majuskule), malá písmena potom minusky (minuskule), opět k vidění na obrázku 1.1.

Majuskule definují znaky, které lze zapsat zpravidla mezi dvě linky písmové osnovy (více o ní dále) — v latince $A, B, C$, atd., takže vyplňují horní a střední psací pásmo. Hodně světových jazyků je používá pro první písmena vlastních jmen (Jan, Brno), ve zkratkách (ČSN, ČR), pro zlepšení čitelnosti apod. Minuskule ($a, b, c$, atd.) se zapisují do čtyř-linkové osnovy, takže mohou zaplňovat pouze střední pásmo, zasahovat také do horního či dolního pásma anebo do všech třech pásem.

### 1.3.2 Písmová osnova

Velmi významnou položku tvoří tzv. písmová osnova. Ta totiž určuje především proporce písma a tvoří je soustava pomyslných vodorovných čar, které můžeme vidět na obrázku 1.2.

Ty vtiskují písmu svůj osobitý charakter a je tedy logické, že se liší u každého písma. Čitelnost písma ovlivňuje především poměr střední výšky k délce horních a




Obr. 1.2: Písmová osnova, ukázka na fontu Century.

dolních dotahů. Pro vylepšení čitelnosti písma je dobré zvětšení střední výška písma a naopak zmenšení horních a dolních dotahů. Čtení je potom pohodlné i v malých velikostech, protože se při zobrazení znaku jeho tahy neslévají.

Na obrázku 1.3 můžeme porovnat čitelnost dvou různých písem s různým poměrem střední výšky, které jsou sice vysázené ve stejné velikosti, ale rozdíl v čitelnosti je zřejmý, jelikož použitá velikost fontu je poměrně malá.

Lorem ipsum dolor sit amet consectetuer facilisi Morbi facilisi tincidunt enim. Habitasse ut Nam Cum sociis scelerisque est et vitae Phasellus ac. Platea convallis parturient justo vel tellus dui libero tristique leo tortor.

Lorem ipsum dolor sit amet consectetuer facilisi Morbi facilisi tincidunt enim. Habitasse ut Nam Cum sociis scelerisque est et vitae Phasellus ac. Platea convallis parturient justo vel tellus dui libero tristique leo tortor.

Obr. 1.3: Srovnání čitelnosti dvou písem s různou střední výškou, vlevo font Times New Roman a vpravo Arial.

### Základní dotažnice

Základní dotažnice, nazývaná též účaří písma, je linka, na které jsou zarovnána jednotlivá písmena do řádku a definují tak jejich dolní hranici. Tato linka je navíc také dolní hranicí pro střední výšku písma.

### Střední dotažnice

Výšku minusek (malých písmen) od základní dotažnice (účaří) definuje střední dotažnice. Tato velikost se potom nazývá střední výška písma, kde střední dotažnice

pro tuto výšku určuje horní okraj.

### Horní dotažnice, verzálková dotažnice

Horní „strop“ pro písmena $b, d, f, h, k, l, t$ určuje tzv. horní dotažnice. Svislé tahy těchto písmen se podle toho nazývají horní dotahy. U mnohých písem zároveň definuje výšku verzálek (velkých písmen). Existují však písma, která mají verzálky paradoxně nižší, než horní dotahy minusek. To proto, aby diakritická znaménka nevystupovala z kresby řádků. Verzálková dotažnice dále ohraničuje výšku verzálek.

### Dolní dotažnice

Dolní dotažnice je linka na spodní hranici svislých tahů minusek $g, j, p, q, y$. Tyto svislé tahy nazýváme dolní dotahy.

### Přetahy

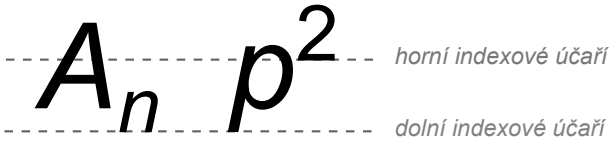
Písmena se zaoblením (jako jsou $o, p, c, b, d$, apod.) mírně překračují své dotažnice, a to jak v minuskách, tak ve verzálkách. Tento jev nazýváme přetah a používáme ho proto, aby se tato písmena nejevila menší, což je zapříčiněno optickým klamem.

### Akcentová dotažnice

Nejvýše položenou linkou písmové osnovy je akcentová dotažnice, která určuje umístění verzálkových akcentů, jelikož minuskové akcenty mají svou vlastní akcentovou dotažnici. Ta se však většinou překrývá s horní dotažnicí.

### Indexová účaří

Indexovým účařím se řídí horní a dolní index. Obrázek 1.4 ukazuje, že pro horní index jej logicky nazýváme horní indexové účaří, pro dolní index pak dolní indexové účaří.




Obr. 1.4: Horní a dolní indexové účaří.

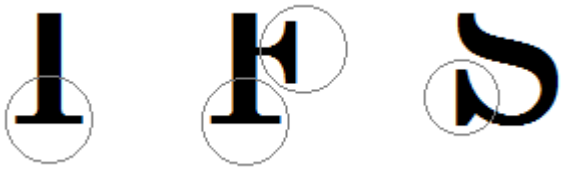
### 1.3.3 Vykreslení písmového znaku

Na kresbě písmového znaku se podílejí hlavní a vedlejší písmové tahy. Hlavní přímý tah nazýváme dřík a přitom platí, že oblý tah se dříkem nenazývá. Nejsnažší příklad lze nalézt v písmenu „I“, které je dříkem celým svým objemem, samozřejmě za předpokladu vhodné volby bezpatkového fontu.

### 1.3.4 Serify (patky)

V elegantním písmu — jako je například font Century — se velmi často používají serify, neboli patky, jak ukazuje obrázek 1.5. U serifů je patrný jejich náběh. Některé fonty mají náběh poměrně agresivní, ty elegantnější z nich potom spíše mírný. Rozdělení na další typy lze spatřit na obrázcích 1.6 a 1.7.

Používání bezpatkového písma není u dlouhých textů doporučeno, jelikož se po nějaké chvíli stává hůře čitelným a snáze se čtenáři unavují oči než při použití písma patkového [1].



Obr. 1.5: Serify, neboli patky, jsou na obrázku zakroužkovány.

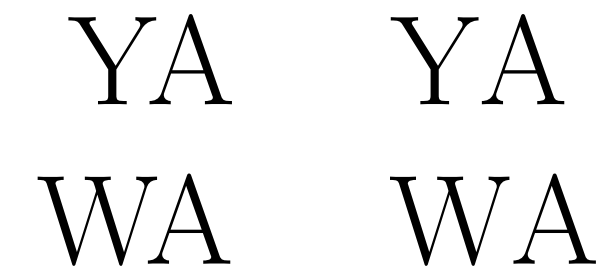
Obr. 1.6: Různé tvary serifů.


Obr. 1.7: Ukázky a typy jednotlivých serifů.

## 1.4 Kerning (vyrovnání) písma

Vyrovnání (někdy také podřezávání nebo prostrkání) písma určuje velikost mezer mezi určitými dvojicemi písmen a je tedy klíčovým prvkem k dosažení lepšího vizuálního dojmu z textu a také čitelnosti slov, potažmo celých vět. Text bez nepřiměřených mezer mezi jednotlivými písmeny vypadá kultivovaněji — kerning má tedy hlavně estetickou funkci, což potvrzuje článek o vyrovnání písma [12].

Aby vše fungovalo jak má, musí písmo obsahovat tzv. kerningové informace. V digitálním světě se jedná především o Par-Wise Kerning, který určuje vyrovnání párů „problémových“ znaků, tedy těch, které se jeví příliš blízko či daleko od sebe. Příklady můžeme najít snadno, namátkou „YA“, „AT“ nebo „WA“ (viz obr. 1.8), ale i mnoho jiných.




Obr. 1.8: Ukázky vyrovnání (vlevo) a nevyrovnání písma.

Za předpokladu, že by spolu tato písmena sousedila a mezi oběma znaky byla zachována stejná vzdálenost jako například mezi písmeny W a L, výsledkem by byl dojem širší mezery. Nevyrovnáním slova LAW vzniká optický problém, kdy je těžko rozpoznatelné, zda se nejedná o mezeru, tzn. znaky LA W. Vyrovnáním se základní plochy sousedících písmen mírně překrývají, jedno písmeno je tak lehce „podsunuto“ pod druhé, což zobrazuje ilustrace 1.9.


Obr. 1.9: Proces vyrovnávání písma.

Výše uvedené příklady se týkaly častěji se vyskytujícího záporného kerningu, existuje však také kladný kerning, kdy se písmena od sebe mírně odsunou. K takové situaci dochází nejčastěji u znaků s diakritikou, ale nejen u nich. Příkladem může být dvojice znaků „rn“, které můžou při použití drobného fontu splynout v jeden celek a tím utvoří písmeno „m“. Odsunem se tento problém vyřeší.

| Písmo | Ukázka |
|---|---|
| normální | Times New Roman |
| kurzíva | *Times New Roman* |
| tučné | **Times New Roman** |
| tučná kurzíva | ***Times New Roman*** |
| kapitálky | Times New Roman |

Tab. 1.2: Porovnání jednotlivých řezů písma Times New Roman.

Moderní softwarové nástroje pro sazbu (DTP) již vyrovnávání řeší plně automaticky. Součástí definice fontu je totiž sada kerningových párů a k nim příslušných vyrovnání. V ojedinělých případech však mohou sazeči tento automatický proces na problematických místech případně ručně doladit. Některé vyspělé programy dokonce dokáží kerningové informace dopočítat i tam, kde chybí, a to na základě geometrických tvarů jednotlivých znaků. Tuto funkci označujeme jako autokerning.

## 1.5 Písmová rodina, řezy písma

Rodinou písma označujeme skupinu řezů odvozených z jednoho typu písma. Řez písma je potom kresebná varianta základního typu písma, kterou používáme pro vyznačování jednotlivých slov nebo delšího textu. Jedná se např. o tučné písmo nebo kurzívu. Díky používání jedné písmové rodiny si zajistíme jednotný vzhled sazby.

Většina typů písma obsahuje čtyři základní řezy, některé typy písma však obsahují daleko více řezů, viz tabulka porovnání řezů fontu Times New Roman 1.2. Jiné typy písma jsou navrženy pouze v jednom řezu a tak rodiny ani netvoří. Písma jedné rodiny mají společné vlastnosti jako je střední výška, výšku verzálek a délka horních a dolních dotahů. Úplná písmová rodina potom obsahuje kromě základního písma také jeho vyznačovací verze stojatého písma a kurzívy.

### Základní písmo

Základní písmo je nevíce používaná varianta písma, kde všechny znaky jsou vzpřímené (stojaté písmo). Jeho aplikace je velmi široká, od sazby knih, časopisů až po webové stránky.

### Vyznačovací řez

Vyznačovacím řezem je např. kurzíva, tučné písmo, velmi tučné, tučná kurzíva nebo kapitálky. Vyznačovací řez se logicky používá ke zvýraznění důležitých slov nebo

ucelených částí textu. Tento řez se liší od základního svou kresbou, která je však ze základního řezu odvozena.

### Kurzíva

Jedná se druh vyznačovacího řezu s mírným sklonem. Není příliš vhodný pro sazbu delšího textu kvůli své horší čitelnosti, avšak pro vyznačování v sazbě je nejčastěji používaným řezem. Kurzívu můžeme dále rozdělit ještě na pravou a nepravou. Pravá kurzíva je speciálně navržený řez písma. Počítačové programy však umožňují naklonit i písmo, které kurzívu neobsahuje. Tomuto jevu říkáme, že se jedná o nepravou kurzívu. Od pravé se ale výrazně liší a především deformuje tvar písma. Obrázek 1.10 srovnává tyto dva druhy řezu a jak je možné vypozorovat, výsledný efekt nepravé kurzívy není ideální a nedosahuje kvalit pravé kurzívy.

*Kurzíva pravá*
*Kurzíva nepravá*

Obr. 1.10: Srovnání pravé (nahoře) a nepravé kurzívy. Přitom byl použit stejný font a stejná velikost písma.

### Kapitálky

Kapitálky jsou o něco menší verze klasických verzálek. Kreslené jsou na střední výšku písma a mohou být opět pravé nebo nepravé. Nepravé kapitálky jsou imitovány z verzálek a mají zeslabenou tloušťku písmových tahů (tzv. duktus, viz dále) oproti základnímu písmu. Takto vyznačený text však v sazbě nebude působit souměrně.

### Duktus

Duktus, neboli tloušťka písmových tahů, je dalším důležitým znakem písmové kresby. Vyjadřujeme ho poměrem tloušťky tahů k výšce písmen, což ovlivňuje výraznost a čitelnost písma. Písma, která mají zesílený duktus pak mohou být polotučná, tučná a velmi tučná, protože u těchto řezů jsou všechny tahy oproti základnímu řezu zesíleny a vysázený text tak působí opticky tmavší. Naopak písma se zeslabeným duktem jsou tenké nebo slabé a jejich tahy jsou tedy zeslabené. Text vysázený těmito písmy se potom zdá opticky světlejší.

## 1.6 Druhy písma

Odborná literatura rozděluje místo mimo jiné na druhy. Uvedu zde ty nejdůležitější, se kterými se dnes setkáváme především na počítači. Jedná se o písmo proporcionální, kde znaky mají různou šířku. Například písmeno „M“ je širší než písmeno „I“. Opakem je neproporcionální písmo, kde všechny znaky mají stejnou šířku. Ta je daná šířkou nejširšího znaku, jak lze vidět na obrázku 1.11.

proporcionální písmo – `neproporcionální písmo`

Obr. 1.11: Druhy písma — proporcionální a neproporcionální.

Ve výčtu dále uvedeme serifové a bezserifové písmo (tzn. patkové a bezpatkové písmo), psané nebo zdobené písmo. Bezpatkové písmo je hojně využíváno na většině internetových stránek, především díky snadnější čitelnosti. Reprezentanty tohoto druhu písma jsou mimo jiné Arial, Verdana, Tahoma nebo Microsoft Sans Serif (Sans Serif volně přeloženo do češtiny znamená „bez patky“). Psané a zdobené písmo najde využití především na různých typech tiskovin oznamujících svatby, výročí, slavnosti, ale také například jako font pro logotyp určitého omezeného segmentu firem. Problematika je větší míře probrána v typografickém manuálu [1]. V této kapitole bylo čerpáno také z knihy [15].

# 2 DIGITÁLNÍ FONT

Pojem font je využíván především v typografii, kde se definuje jako kompletní znaková sada abecedy dané velikosti a jednotného stylu. V dnešní době se slovo font pojí především s použitím variabilních stylů písma ve výpočetní technice. Díky ní tvorba nejrůznějších stylů písem zažila a zažívá, i když s jistou dávkou zpomalení, znatelný rozvoj.

## 2.1 Základy tvorby fontů

Jedním z nejdůležitějším způsobů návrhu moderního fontu v počítačové podobě je využití křivek. V devadesátých letech dvacátého století, kdy výpočetní technika nedosahovala zdaleka takových výkonů jako dnes, bylo důležité zjednodušovat fonty. Příliš mnoho Bézierových bodů totiž vedlo ke značnému zpomalení nahrávání fontů v aplikacích nebo tiskárnách [4].

Dnes je již situace jiná. Počítače se neustále zdokonalují a jejich výkon postupně roste. Díky tomu nemusíme tolik dbát na redukci počtu Bézierových bodů a více se soustředit na kvalitu provedení fontu. K těmto účelům se používají výpočetní nástroje, tedy především software, na rozdíl od dřívějších dob, kdy bylo nutné vše kreslit ručně.

### 2.1.1 Dnešní standard pro tvorbu fontu

Vznik nového písma je v posledních letech na denním pořádku. Velké a bohaté firmy se snaží zaujmout své zákazníky mimo jiné i uceleným a čistým korporátním stylem. K tomu je často zapotřebí vytvořit i nový font, aby korespondoval a ladil s touto vizuální prezentací a upevňují si tak svou korporátní identitu. Takové fonty nazýváme „custom & corporate“ a tvoří ji většinou grafik společně s celým grafickým stylem.

Další požadavky na písmo mohou vzniknout od médií, kdy televizní společnosti font odlaďují pro dobrou čitelnost na obrazovce nebo kvalitní písmo pro usnadnění čtení tiskovin. Mnohdy se můžeme setkat i s tvorbou fontu pro jedinečné akce a projekty (například sportovní klání, festivaly nebo kulturní akce).

### 2.1.2 Metody tvorby fontů

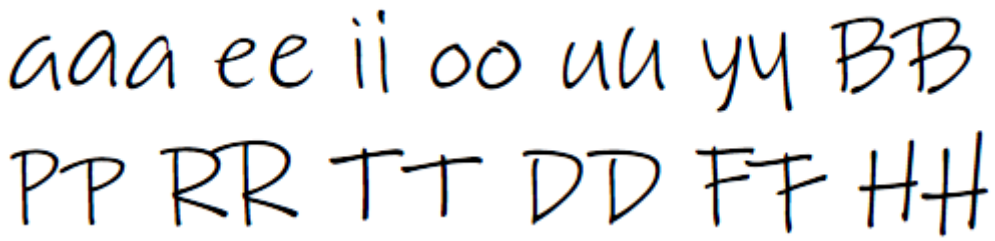
K tvorbě lze využít některý z vektorových grafických programů, kterým může být například Adobe Ilustrator nebo jemu podobný software. Ve většině případů již odpadá nutnost použití pera, pravítka a papíru.

Detailní popis návrhu a tvorby digitálního fontu je zaznamenán v článku Drawing with Beziers, point by point [4]. Tam se mimo jiné uvádí, že by řídící bod měl přesáhnout třetinu vzdálenosti k dalšímu bodu. Tak dosáhneme kvalitních, dobře vypadajících přechodů v křivkách a odfiltujeme příliš ostré zaoblení.

### 2.1.3 Inspirace

Na internetu můžeme nalézt celou řadu fontů, ať se jedná o zdarma stažitelná písma, nebo také placené balíčky. Výběr je velice pestrý a pomůže nám alespoň k základní inspiraci pro tuto práci. Výborný příklad ukazuje v manuálu [10] font Lindsey Pro, který obsahuje vylepšené OpenType vlastnosti včetně alternativních znaků. V praxi to znamená, že je k jednotlivým písmenům přiřazeno několik druhů jejich vizuálního pojetí pro docílení „náhodného“ efektu. Ten se ovšem stejně po nějaké době začne opakovat, jelikož tato databáze variant není nevyčerpatelná.

Font Lindsey Pro je řazen mezi písmo psané. V tomto projektu jej budu využívat také, mimo něj vyzkouším i elegantnější font, konkrétně písmo velmi podobné stylu Arial a také Times New Roman. I přesto nám však Lindsey Pro poslouží jako dobrý příklad toho, co se dá dosáhnout při změnách jednotlivých znaků. Vhodná ukázka je k vidění na obrázku 2.1.




Obr. 2.1: Psané písmo Lindsey Pro s variantami znaků.

## 2.2 Použití počítačové techniky

Způsob návrhu písma byl dříve omezen na rýsovací pomůcky, tužku a papír. Dnes je již situace jiná — tento proces se výrazně usnadnil právě díky moderním výpočetním technologiím. Jednodušší konstrukce fontů zvládne každý, kdo ovládá alespoň základní dovednosti ve vektorovém grafickém programu.

Jak již bylo řečeno, tento typ software se při návrhu písma využívá nejčastěji. Můžeme ale využít i specializované typy programů, např. nadstavby pro LaTeX

a jiné. Moderní fonty využívají i některých vymožeností. Ty budou popsány později. Nyní se podíváme na rozdělení fontů dnešní doby.

### 2.2.1 PostScript

PostScript je programovací jazyk primárně určený ke grafickému popisu tisknutelných dokumentů. Byl vyvinut v roce 1985 společností Adobe Systems Incorporated a jeho hlavní devizou je jeho nezávislost na zařízení, na kterém se daný dokument bude tisknout. Mezi odborníky byl považován za standard pro dražší typy tiskáren, avšak díky svým rozsáhlým možnostem se brzy stal i formátem pro ukládání obrázků.

Tento programovací jazyk využívá aplikaci Bézierových křivek k popisu fontů, což je přesně to, čeho bych rád dosáhl v tomto projektu.

### 2.2.2 TrueType

Tento standard založený na vektorové grafice, vyvinutý koncem 80. let minulého století firmou Apple, měl být konkurentem fontů Adobe Type 1.

Dnes se s nimi setkáváme v operačních systémech Microsoft Windows nebo GNU-/Linux (ve verzi FreeType).

### 2.2.3 OpenType

Nástupcem standardu TrueType se stal nový standard OpenType, který v sobě spojuje to nejlepší z TrueType a konkurenčního standardu PostScript Type 1. Navíc má mnohem lépe řešené problémy s lokalizací písem. Byl vyvinut společností Microsoft a později se k práci na tomto typu písma připojil i Adobe Systems. Podporován je v OS Microsoft Windows, Mac OS X a Linux (opět varianta FreeType).

Mezi hlavní výhody oproti starším standardům řadíme možnost jednoho souboru popisovat až 65 536 znaků, přenositelnost mezi různými OS, lepší podporu pro typografické speciality (např. slitky), a v neposlední řadě také to, že je založen na kódování Unicode, což zaručuje méně problémů s lokalizací písem. Navíc je u něj zaručena zpětná kompatibilita s TrueType a PostScript, a tak znaky mohou být popsány pomocí starých zvyklostí.

## 2.3 Návrh oživení fontů

Některé fonty působí často velmi fádním dojmem. U formálních dokumentů, jako je například i tato práce, se vyžaduje jistá úroveň prezentace, takže nemůžeme oče-

kávat, že bude napsána fontem Comic Sans (viz obrázek 2.2), nýbrž zvolené písmo bude spíše z rodiny stylových patkových fontů Times New Roman (viz tabulka 1.2).


Obr. 2.2: Ukázka písma Comic Sans.

Pro účely demonstrace náhodnosti však budeme raději využívat tučnější font, abychom dosáhli lepší viditelnosti změn u jednotlivých znaků. Pro začátek je dobré vybrat znak abecedy, který svou složitostí nebude komplikovat návrh náhodnosti. Vyvarujeme se tedy použití znaků, které obsahují oko, smyčku nebo serif a zaměříme se na jednoduché písmeno „C“. Posléze, až budou jasné všechny aspekty a neduhy základního návrhu náhodnosti, zkusíme podobný proces provést i u složitějšího znaku, kterým bude písmeno „M“.

Další volba znaků bude probíhat strategicky tak, abychom mohli skládat jednoduché věty, jak bude ostatně ukázáno později. Proto bude zapotřebí vytvořit nejčastěji se vyskytující samohlásky typu „I“ nebo „E“ a poté také velmi používané souhlásky jako „N“, „L“ nebo „H“ k již dříve zmíněným písmenům „C“ a „M“.

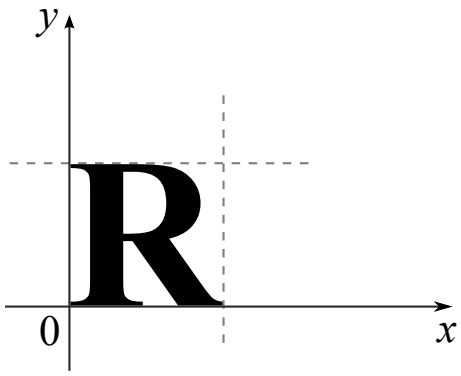
### 2.3.1 Docílení náhodnosti

Jak již bylo zmíněno v úvodu této kapitoly, náhodnost začneme zkoumat na jednoduchém znaku „C“. Ten by bylo možné sestavit z pouhé jedné Bézierovy křivky, která bude mít čtyři řídící body. Takový font by však nepůsobil příliš hezky, navíc by se jednalo o linku bez výplně. Použijeme tedy více řídících bodů, které v základním zobrazení rozestavíme do určité polohy. Poté projdeme každý řídící bod a budeme zkoumat, zda a jak je možné jej posunout k dosažení chtěné náhodnosti.

Nevhodným posunutím řídícího bodu může totiž dojít ke vzniku tvz. zubů, překrytí křivek nebo nevzhledných zaoblení. Ty se tedy pokusíme maximálně eliminovat, jak bude popsáno dále.

Samotné vygenerování náhodných bodů se bude dít v určitém rozmezí, které pro jednotlivé znaky předem určíme. Toto rozmezí bude relativní k souřadnicovým osám, takže se efekt může i několikrát opakovat. Z těchto intervalů algoritmus náhodně vybere hodnoty, porovná určité aspekty, aby nedošlo ke znehodnocení kvality písmene, a vykreslí znak za použití těchto parametrů.

Jednat se bude o posuny na ose $x$ a $y$, ale také jejich kombinace. V souřadném systému budeme kontrolovat rozmezí, přes které se některé body nesmí dostat. Pro představu je uveden obrázek 2.3.




Obr. 2.3: Transformace znaku v souřadnicovém systému.

### 2.3.2 Oživení struktury znaku

Pokud nastavíme správné „mantinely“ pro náhodný posun řídících bodů, docílíme pěkný vzhled fontu při oživení jeho struktury. Navíc je ovšem potřeba zapracovat určité interakce. Některé náhodně vybrané hodnoty by mohly kolidovat s jinými náhodnými hodnotami vedlejšího řídícího bodu.

Ty se budeme snažit eliminovat tím, že určité kombinace vygenerovaných hodnot nebudou možné. Algoritmus je tedy bude znovu generovat až do doby, než dojde k odstranění potencionálních problémů. V tomto okamžiku je dobré myslet na poměr kvality a výkonu. Abychom zbytečně nezpomalovali výpočet tzv. donekonečna, zkusíme najít takový kompromis, který bude vygenerován v rozumném časovém limitu při zachování jisté kvality výsledného fontu.

Náhodnost můžeme zdokonalit i podélnou deformací, přičemž na základě zvolených hodnot docílíme buď roztažení nebo přiměřené zmenšení jednotlivých znaků. K tomu využijeme čtvercovou matici o rozměrech $2 \times 2$ ve formátu

$$\begin{pmatrix} a & b \\ b & c \end{pmatrix},$$

do které dosadíme hodnoty pro souřadný systém tak, abychom docílili kýženého efektu. V tomto případě zvolíme například $a = 0,9$, $b = 0$ a $c = 1$. Výpočet bude potom probíhat následovně:

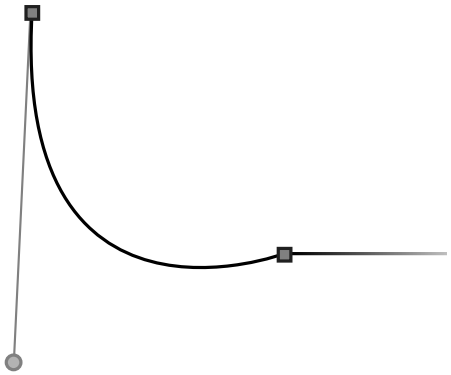
$$\begin{vmatrix} 0,9 & 0 \\ 0 & 1 \end{vmatrix} \times \begin{vmatrix} x_i \\ y_i \end{vmatrix} = \begin{vmatrix} 0,9x_i \\ y_i \end{vmatrix}.$$

Z výsledku je potom patrné, že dojde k transformaci znaku na ose $x$, a to na 90% jeho původní šířky — bude tedy zmenšen. Do výšky zůstanou proporce znaku stejné jako před výpočtem, jelikož jsme osu $y$ vynásobili jedničkou. Pokud bychom k vynásobení osy $x$ použili číslo větší než jedna, potom dojde k roztažení znaku do šířky.

Jak bylo uvedeno, při této transformaci dojde k upravení znaku pouze na ose $x$. To je důležité z hlediska zachování určité kvality. Prozatím nebudeme chtít pracovat s výškou fontu — raději zachováme font v jedné rovině tak, aby dotažnice navzájem korespondovaly. Při změně proporcí na ose $y$ dojde k jistému roztancování písma, ovšem to může působit příliš silně. Proto tuto variantu zkusíme až po horizontální transformaci a s rozmyslem ji poté jemně zakomponujeme.

### 2.3.3 Problémy spojené s náhodností

Abychom zabránili zbytečnému zpomalování vykreslení znaku, je potřeba dbát na správnou vyváženost rychlosti algoritmu oproti nastaveným hranicím možných kombinací náhodných hodnot, jak bylo zmíněno v jedné z předchozích částí. Příliš striktní politika způsobí značné prodlevy při generování složitějších znaků a naopak, příliš liberální přístup může mít za následek kostrbatý font, což zapříčiní nedokonalosti křivky např. v zaoblení, které můžeme pozorovat na detailním obrázku 2.4.



Obr. 2.4: Nevhodné posuny řídících bodů, které způsobí ošklivý zub u zaoblení.

Tomuto nežádoucímu jevu se pokusíme vyhnout optimalizací algoritmu. To může vést ke značnému zefektivnění programu a tím i lepším dosaženým výsledkům.

### 2.3.4 Generování vyrovnání znaků (kerning)

Vyrovnání znaků můžeme rozdělit na dvě skupiny — znaky problémové a bezproblémové. U těch problémových musíme zajistit, aby znaky nezasahovaly příliš do druhého znaku. Příklad lze nalézt u písmen „r“ a „n“. Vygenerováním velmi malé mezery by mohlo dojít ke spojení těchto znaků, tím by vzniklo písmeno „m“, což

by vedlo k dramatickému zhoršení čitelnosti slova nebo dokonce ke změně významu věty.

U bezproblémových dvojic znaků je situace jednodušší v tom smyslu, že nemusíme řešit znehodnocení slova nebo věty. Navíc nám stačí mezeru generovat v relativně malém rozsahu. Velká mezera totiž může evokovat rozdělení slova faktickou mezerou a tím pádem může opět dojít ke zhoršení čitelnosti nebo znehodnocení významu slova, jak ukazuje obrázek 2.5.


Obr. 2.5: Ukázka rozdělení slova.

### 2.3.5 Skládání slov a celých vět

Pro vhodnou ilustraci slova složeného z „živého“ fontu využijeme navržené znaky. Aby nebylo nutné tvořit celou českou abecedu, zaměříme se na jednodušší slova, která nalezneme např. v dětských čítankách.

Slova potom zkusíme spojit do základních vět, přičemž jednotlivá slova slepíme mezerami, které budeme opět náhodně upravovat podobným způsobem, jako např. u generování vyrovnání znaků v předchozí podkapitole. Pozor musíme dát na generování příliš malých mezer, která by znehodnotily čitelnost věty slepením slov k sobě.

Dalším prvkem při oživování fontu může být posun znaku na osách $x$ a $y$. U osy $x$ budeme velice opatrní, aby nedošlo k překryvu jednotlivých písmen, nebo dokonce ke spojení dvou a více slov v nějaký nesmysl.

Na ose $y$ se potom s jistou dávkou nadsázky dá font rozpohybovat, avšak opět za předpokladu, že slovo nebude vizuálně příliš rozděleno. Na obrázku 2.6 vidíme ukázku přílišného posunutí na obou osách. Tím by dramaticky klesla čitelnost slova, což bychom jistě dopustili jen velmi neradi.


Obr. 2.6: Vlevo vhodné a vpravo nevhodné posuvy znaku na osách $x$ a $y$.

Jak již bylo prezentováno dříve, věty se budou skládat z nejpoužívanějších souhlásek a samohlásek. Ve finále by takto složená věta mohla mít tvar jako „máma mele maso“ a podobné. Ideálně se celá věta vykreslí verzálkami. Postupným dalším vývojem algoritmů můžeme přidávat i funkcionalitu pro přechod na minusky, nebo eventuálně kombinaci verzálek a minusek (písmeno velké abecedy na začátku věty, začátku vlastního jména, apod.).

V této kapitole byla použita literatura [4], [1] a [7].

# 3 SOUSTAVA SOUŘADNIC

Soustava souřadnic je systém základních parametrů (referenčních bodů, přímek nebo křivek), který ve zvolené vztažné soustavě usnadňuje určení polohy jednotlivých těles. Soustava souřadnic má matematický obsah a souvisí s popisem jevu, kdežto vztažné soustavy mají fyzikální obsah a souvisí se samotným jevem.

V takovém systému souřadnic je poloha bodu určena skupinou čísel, které se nazývají souřadnice (anglicky coordinates). Ty mohou reprezentovat kromě vzdálenosti například úhel vzhledem k referenčním bodům, přímkám nebo křivkám vybrané souřadné soustavy.

Obecně můžeme souřadný systém označit jako vzájemně jednoznačné zobrazení skrze množinu bodů $n$-rozměrného prostoru a uspořádané za pomoci $n$-tice čísel. Polohu bodu umístěného na přímce (1D) potom definujeme jedním číslem, polohu bodu v rovině (2D) dvojicí čísel a konečně polohu bodu ve 3D prostoru trojicí čísel.

Zavedení soustavy souřadnic je provedeno volbou počátku soustavy souřadnic a souřadnicových os (pro 3D prostor nejčastěji užíváme označení $x, y, z$). Polohu libovolného bodu určíme jednoduše odečtením jeho souřadnic na jednotlivých osách.

Rozeznáváme více druhů souřadnicových systémů. Mezi nejznámější a nejčastěji používané soustavy řadíme kartézskou, homogenní nebo polární. Ty jsou potom klíčové pro určení polohy bodu, jelikož každá má odlišné chování a definici.

#### Ortogonální soustava souřadnic

V případě, že jsou na sebe souřadnicové osy navzájem kolmé v každém svém bodě prostoru, pak se jedná o tzv. ortogonální soustavu souřadnic.

#### Rovinná soustava souřadnic

Pokud je pohyb v rámci soustavy souřadnic omezen, můžeme jej nazvat rovinným. Omezení potom platí například na pohyb pouze na osách $x$ a $y$, tím dosáhneme efektu dvourozměrného prostoru, který budu využívat při řešení tohoto projektu.

Pro převod mezi jednotlivými souřadnými systémy provádíme transformace souřadnic. Nás bude zajímat především převod z kartézských souřadnit na homogenní, což bude popsáno dále v sekci o homogenních souřadnicích.

## 3.1 Kartézské souřadnice

U tohoto typu souřadnicového systému se osy protínají v počátku soustavy souřadnic a jsou na sebe navzájem kolmé, přičemž jednotka je na všech osách stejně velká. Jak již bylo také naznačeno, v prostoru má kartézská soustava souřadnic tři vzájemně

kolmé osy (označované $x, y, z$) a v rovině potom dvě kolmé osy $(x, y)$. Pro zápis kartézských souřadnic v rovině používáme následující tvar: $X = [x, y]$, kde $X$ je bod v souřadném systému umístěný na souřadnicích $x$ a $y$. Bod $P = [0, 0]$ je potom počátek této soustavy souřadnic.

## 3.2 Homogenní souřadnice

Při práci v homogenních souřadnicích budeme uvažovat o bodech ve volném prostoru jako o sloupcových vektorech, které je možné spatřit na ilustraci 5.2 tak, jak se zapisují v programu Matlab.

Homogenní souřadnice našly uplatnění v mnoha praktických oblastech, především potom v počítačové grafice, kde umožňují všechny afinní a projektivní transformace provádět za pomoci násobení matic. Speciální grafický mikroprocesor se tak může specializovat výhradně na tento jeden druh výpočetní operace, což vede k výraznému zvýšení celkové datové propustnosti algoritmů (programů) a tím také ke zrychlení aplikací pracujících s počítačovou grafikou.

Vyjádření maticí je potom výhodné i z hlediska relativně jednoduché inverzní transformace, která je reprezentována inverzní maticí. To vše samozřejmě za předpokladu, že inverzní transformace je na této matici proveditelná.

#### Kartézské a homogenní souřadnice a jejich transformace

Pokud uvažujeme 3D prostor, pak v něm má bod $X$ tři kartézské souřadnice v následujícím tvaru: $X = [x, y, z]$. Při užití homogenních souřadnic má ten samý bod $X$ následující tvar:

$$X = [x, y, z, w]^\top, \qquad x = \frac{x_{kart}}{w}, y = \frac{y_{kart}}{w}, z = \frac{z_{kart}}{w}, w \neq 0, \tag{3.1}$$

kde $w$ je váha s častou hodnotou $w = 1$, jak se uvádí i v sekci 4.6.1 [14].

Dále je vhodné připomenout, že jeden bod v kartézských souřadnicích má nekonečně mnoho vyjádření pro homogenní souřadnice. To je docíleno díky dělení. Pro $w = 0$ platí, že bod se nachází v pomyslném nekonečnu.

Jelikož při řešení tohoto projektu uvažujeme pouze dvourozměrný prostor, nebude se souřadnice na ose $z$ brát v potaz. Tím se homogenní souřadnice upraví pro použití právě ve 2D.

## 3.3 Geometrické transformace

Rozdělení transformací v prostoru je možné následujícím způsobem:

- nelineární,

- lineární,
- afinní,
- eukleidovské,

přičemž pro tyto typy transformací platí následující tvrzení: lineární plní princip superpozice, afinní umí zachovávat rovnoběžnost a poměry délek u rovnoběžek a konečně eukleidovské zachovávají úhly a délky úseček.

Výchozím bodem pro pro následující příklady transformací bude bod $X = [x, y, z, w]^\top$, kde zvolíme $w = 1$. Tento zápis je však vhodný především pro trojrozměrný prostor. Budeme tedy ignorovat souřadnici na ose $z$, čili zvolíme $z = 0$ a dostaneme bod $X = [x, y, w]^\top$, kde opět hodnota $w = 1$. Docílíme tak pro naše potřeby dvourozměrného chování transformací. Homogenní souřadnice nově transformovaného bodu $X' = [x', y', w']^\top$ získáme předpisem

$$X' = \mathbf{A}P, \tag{3.2}$$

kde $\mathbf{A}$ vyjadřuje matici o rozměru $3 \times 3$.

### Transformace identita (eukleidovská)

Identita v podstatě znamená neprovedení žádné změny, jelikož násobíme jedničkou. Matice $\mathbf{A}$ je jednotková matice (to znamená, že má na své hlavní diagonále samé jedničky a ostatní prvky matice jsou nulové).

$$\mathbf{A} = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}, \tag{3.3}$$

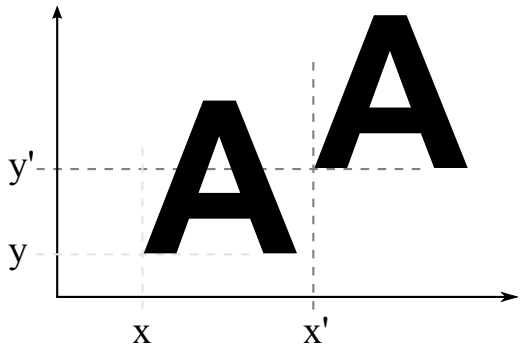
po výpočtu nenastane žádná změna: $[x, y, w] = [x', y', w']$. Transformovaný objekt zůstane stejný jako před transformací.

### Transformace posunutí (eukleidovská)

Posunutí, neboli translace, znamená posunutí vektorem $[x_\top, y_\top] = [x' - x, y' - y]$. Matici $\mathbf{A}$ upravíme z původní jednotkové matice v posledním sloupci takto:

$$\mathbf{A} = \begin{vmatrix} 1 & 0 & x_\top \\ 0 & 1 & y_\top \\ 0 & 0 & 1 \end{vmatrix}, \tag{3.4}$$

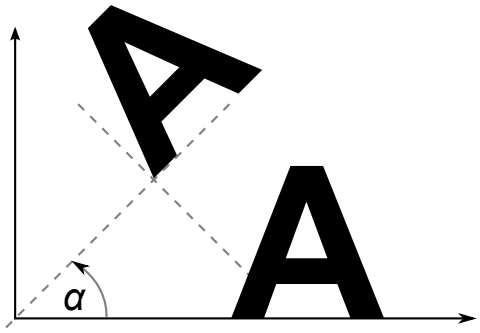
po výpočtu bude transformovaný objekt posunutý ve směru osy $x$, osy $y$ nebo v obou těchto osách, jak je znázorněno v grafu 3.1.




Obr. 3.1: Ukázka transformace posunutí.

### Transformace rotace (eukleidovská)

Jelikož obecný případ volné rotace kolem prostorových os je velmi složitý, použijeme pouze speciální případ, kdy rotace proběhne pouze kolem jedné ze souřadných os. Samotné otočení je potom reprezentováno úhlem otočení, který značíme $\alpha$, viz obrázek 3.2.




Obr. 3.2: Ukázka transformace rotace.

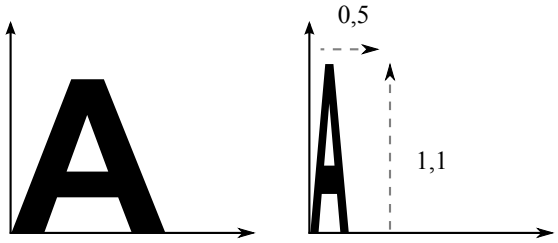
Rotaci v rámci roviny $xy$ o úhel $\alpha$ provedeme následujícím způsobem úpravou matice $\mathbf{A}$:

$$\mathbf{A} = \begin{vmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{vmatrix}, \tag{3.5}$$

### Transformace změna měřítka (afinní)

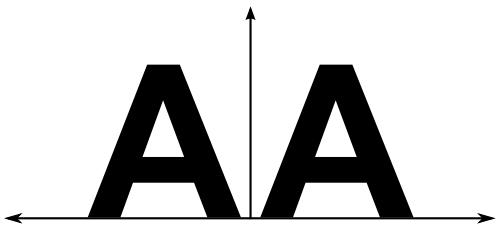
Transformace typu změna měřítka je charakterizována poměrnou změnou velikosti objektu v závislosti na směru souřadnicové osy: $s_x$ nebo $s_y$. Matice $\mathbf{A}$ bude ve tvaru:

$$\mathbf{A} = \begin{vmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{vmatrix}. \tag{3.6}$$




Obr. 3.3: Ukázka transformace změny měřítka.

Existuje i speciální případ změny měřítka, který nazýváme souměrnost (též symetrie nebo reflexe). Potom jeden z koeficientů musí mít hodnotu $-1$. Reflexe dle osy $y$ znamená zrcadlení ve vertikální směru, jak lze vidět na obrázku 3.4.

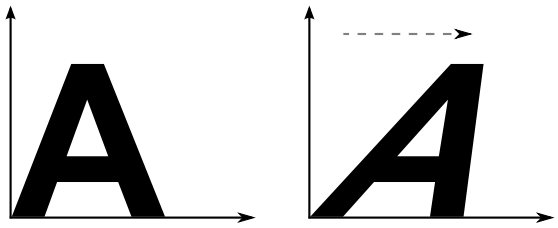


Obr. 3.4: Speciální případ transformace změny měřítka — reflexe.

### Transformace zkosení (afinní)

Transformace zkosení (anglicky shear, takto jsou pojmenovány i proměnné ve zdrojovém kódu algoritmu) je možné aplikovat buď na ose $x$ nebo $y$. Pro zkosení ve směru osy $x$ (viz obrázek 3.5) bude mít matice $\mathbf{A}$ takovýto tvar:

$$\mathbf{A} = \begin{vmatrix} 1 & 0 & 0 \\ k_x & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}. \tag{3.7}$$



Obr. 3.5: Příklad transformace zkosení.

**Obecný tvar matice (afinní)**

Je zajímavé sledovat, že při výpočtu transformace se nemění hodnota souřadnice $w$. Navíc po vynásobení libovolného počtu afinně-transformačních matic mezi sebou dostaneme opět afinní matici. Proto skládáním afinních transformací jsou znovu afinní transformace.

Každou afinní transformaci můžeme vyjádřit maticí $\mathbf{A}$ ve tvaru

$$\mathbf{A} = \begin{vmatrix} a_{11} & a_{12} & x_\top \\ a_{21} & a_{22} & y_\top \\ 0 & 0 & 1 \end{vmatrix}. \tag{3.8}$$

## 3.4 Skládání transformací

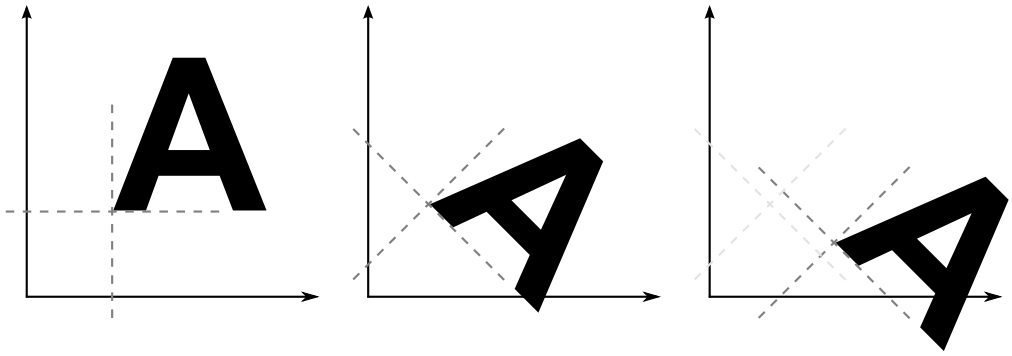
Skládání transformací docílíme poměrně jednoduše, a to násobením jednotlivých matic. U násobení matic však bude záležet na jejich pořadí, jelikož násobení matic není komutativní. Jiných výsledků se tedy dočkáme při transformacích v pořadí posunutí → rotace a rotace → posunutí. Dle sekce 4.8 ve skriptech [14] násobíme matice takto:

$$P' = (\mathbf{A}_n \cdot \ldots \cdot (\mathbf{A}_2 \cdot (\mathbf{A}_1 \cdot P))) = \mathbf{A}_n \cdot \ldots \cdot \mathbf{A}_2 \cdot \mathbf{A}_1 \cdot P = \mathbf{A}P. \tag{3.9}$$

Pro uvedení ukázky použijeme středovou souměrnost kolem bodu $x = 2, y = -1$. Nejprve posuneme celý obraz o vektor $[-2, 1]$ pro přesun do počátku souřadnicového systému. Dále aplikujeme matici souměrnosti ($s_x = s_y = -1$) a posléze obraz posuneme zpátky o vektor $[2, -1]$. Výsledná matice bude mít tuto strukturu:

$$\mathbf{A} = \begin{vmatrix} -1 & 0 & -4 \\ 0 & -1 & 2 \\ 0 & 0 & 1 \end{vmatrix}. \tag{3.10}$$

U všech příkladů matic (posun, rotace, zkosení i změna měřítka) můžeme ověřit, že jsou invertibilní, a to tak, že mají plnou hodnost. Z nového bodu se tak lze vrátit k původnímu. Navíc, jak uvádí [14], součin matic s plnou hodností je opět matice s plnou hodností, a tedy i složená transformace je invertibilní, což je k vidění i na obrázku 3.6.



Obr. 3.6: Ukázka složení transformací — postupně zleva posun, otočení a další posun.

# 4 DVOUROZMĚRNÁ VEKTOROVÁ GRAFIKA

Základním stavebních kamenem vektorové grafik je pouze a jen matematika. Všechny druhy křivek, se kterými se na počítačích setkáváme, jsou vyvinuty za pomoci matematiky a také jejich vykreslování stojí na matematických metodách.

V současnosti je naštěstí k dispozici celá řada grafických aplikací, které od obyčejného člověka nevyžadují rutinní znalosti polynomů třetího řádu, jimiž jsou vektorové obrázky popsány, a tak se při běžném používání těchto programů můžeme klidně věnovat vlastní grafické tvorbě.

K jednoznačným výhodám vektorové grafiky řadíme možnost libovolných změn velikosti nebo proporcí obrázku, aniž by to mělo vliv na jeho kvalitu, což se využívá často například u jednoduchých her nebo vektorových map. Mezi další výhody patří manipulace s jednotlivými prvky obrázku nezávisle na ostatních prvcích a k dobru jde také menší datový objem vektorové grafiky.

Tento typ grafiky se nejčastěji využívá na tvorbu ilustrací, logotypů, elektronickou sazbu a také například při tvorbě flashových animací. Nejznámější vektorové editory (software) jsou na dnešním trhu Illustrator (od společnosti Adobe) a Corel Draw.

## 4.1 Křivky

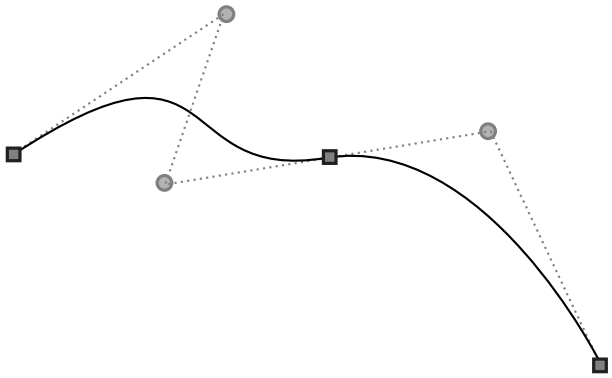
Křivky jsou důležitým elementem v informačních technologiích. Kromě tvorby moderních fontů jsou využívány i v designu, pro modelování nebo také definice dráhy pohybujících se objektů. Křivky mají tzv. řídící body, díky kterým dělíme křivky na základní dva typy — aproximační (nemusí procházet svými řídícími body, jak je možné vidět na obrázku 4.1) a interpolační (naopak musí procházet svými řídícími body, což můžeme pozorovat na obrázku 4.2), jak je uvedeno ve skriptech předmětu Multimediální a grafické procesory [14].

## 4.2 Napojování křivek

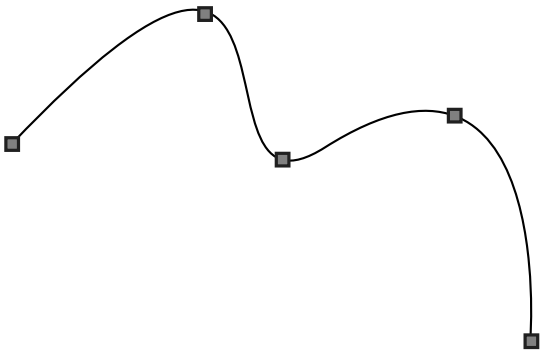
Nejvhodnější metodou pro modelování složitějších tvarů je napojování křivek malé složitosti (např. kubiky). Důvod je prostý — složité tvary nemá smysl modelovat pomocí jedné komplikované křivky, jak je uvedeno v [14].

To bude klíčová úvaha při řešení této práce, kde budeme využívat právě uvedenou metodu. Bližší informace však probereme o něco dále u Bézierových kubik.

Při spojování jednotlivých křivek budou hrát důležitou roli jejich řídící body. Ty dále rozdělíme na kotevní a kontrolní body. Kotevní body určují nejen polohu daného



Obr. 4.1: Aproximační typ křivky.



Obr. 4.2: Interpolační typ křivky.

segmentu (části křivky), ale protože spojují jednotlivé segmenty dohromady, závisí na nich i tvar přechodu jednoho segmentu na druhý. Tím můžeme docílit vytvoření ostrého (rohového) či hladkého spoje. U plynulých napojení napojení je důležité zachovat spojitost, tedy vyhnout se nežádoucím efektům (zub a podobné).

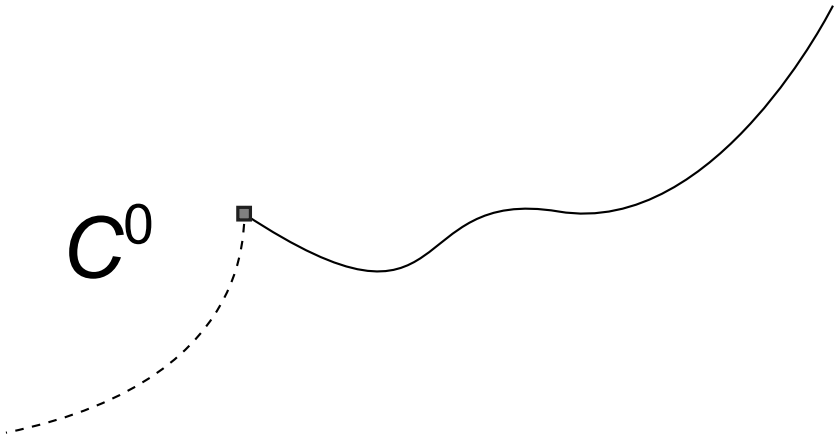
### Parametrická spojitost

Dle skript [14] jsou parametricky nebo také $C^k$-spojitě navázané segmenty složité křivky vyjádřeny následujícím vztahem za předpokladu, že v jejich společném řídícím bodě jsou shodné vektory všech derivací až do řádu $k$:

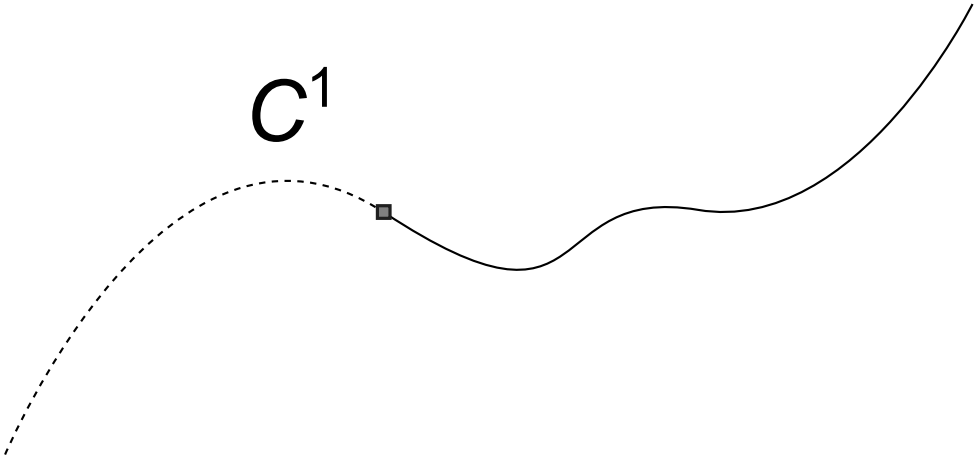
$$\overrightarrow{q_1}'(1) = \overrightarrow{q_2}'(0), \quad \overrightarrow{q_1}''(1) = \overrightarrow{q_2}''(0), \quad \ldots, \quad \overrightarrow{q_1}^{(k)}(1) = \overrightarrow{q_2}^{(k)}(0). \tag{4.1}$$

Potom dva segmenty, které navazují, jsou logicky $C^k$-spojité, protože mají společný uzel:

$$\overrightarrow{q_1}^{(0)}(1) = \overrightarrow{q_2}^{(0)}(0). \tag{4.2}$$




Obr. 4.3: Ukázka ostrého parametrického napojení typu $C^0$.




Obr. 4.4: Ukázka hladkého parametrického napojení typu $C^1$.

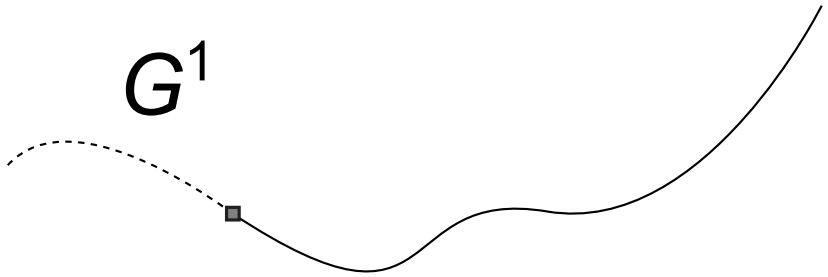
**Geometrická spojitost**

Říkáme, že segmenty křivky jsou geometricky neboli $G^k$-spojitě navázány v případě, že v jejich společném řídícím bodě jsou vektory všech derivací až do řádu $k$ rovnoběžné a mají také souhlasnou orientaci, viz:

$$\exists c_n > 0, n = 1, \ldots, k : \overrightarrow{q_1}^{(n)}(1) = c_n \cdot \overrightarrow{q_2}^{(n)}(0). \tag{4.3}$$

## 4.3 Bézierovy křivky

Bézierovy křivky byly vyvinuty v šedesátých letech minulého století francouzským vynálezcem dr. Pierrem Bézierem. Zajímavostí je, že v té době pracoval pro automobilku Renault, která tyto nové metody plánovala využít k tvorbě moderních futuristických tvarů aut.




Obr. 4.5: Ukázka geometrického napojení typu $G^1$.

U tohoto typu křivky se počáteční ($t = 0$) a koncový ($t = 1$) bod nachází v počátečním a koncovém bodě polynomu, přičemž musí být splněna podmínka konvexního obalu. To znamená, že všechny body Bézierovy křivky leží v konvexním obalu řídícího polynomu, jak je vidět v 4.4.

$$q^{Bez}(0) = P_0 \qquad q^{Bez}(1) = P_n \tag{4.4}$$

Jak je tedy z předchozího odstavce zřejmé, musí být počáteční a koncový bod součástí samotné křivky 4.5.

$$q'^{Bez}(0) = n(P_1 - P_0) \qquad q'^{Bez}(1) = n(P_n - P_{n-1}) \tag{4.5}$$

Tento druh křivky je postaven nejčastěji na Bernsteinových polynomech $B_{k,n}(t)$. Bézierovu křivku $n$-tého řádu definujeme dle [14] jako

$$q^{Bez}(t) = \sum_{k=0}^{n} P_k B_{k,n}(t) \tag{4.6}$$

kde $P_k$ je $n + 1$ řídících bodů a $B_{k,n}$ jsou Bernsteinovy polynomy $n$-tého stupně. Tyto polynomy jsou potom definovány takto:

$$B_{k,n}(t) = \binom{n}{i} t^i (1-t)^{n-1}, t \in \langle 0, 1\rangle, i = 0, 1, ..., n \tag{4.7}$$

Po následné úpravě a dosazení:

$$q^{Bez}(t) = P_0(1-t)^n + P_1\binom{n}{1}t(1-t)^{n-1} + P_2\binom{n}{2}t^2(1-t)^{n-2} + ... + P_n t^n \tag{4.8}$$

Při použití v praxi se nejčastěji volí Bézierova křivka nazývaná kubika (viz 4.14), která má nastavenu hodnotu $n = 3$. Vzorec při této hodnotě bude vypadat následovně:

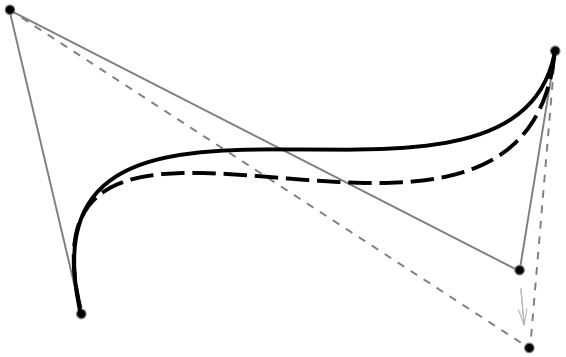
$$q^{Bez}(t) = P_0(1-t)^3 + P_1 3t(1-t)^2 + P_2 3t^2(1-t) + P_3 t^3 \tag{4.9}$$

Anebo kvadrika s hodnotou $n = 2$, která má vyjádření v této podobě:

$$q^{Bez}(t) = P_0(1-t)^2 + P_1 2t(1-t) + P_2 t^2 \tag{4.10}$$

### 4.3.1 Nelokálnost změn

Jednoznačnou nevýhodou tohoto typu křivky je nelokálnost změn. To znamená, že posun jednoho vrcholu řídícího polynomu způsobí změnu tvaru celé křivky, jak se uvádí i literatuře [18] a lze vidět na obrázku 4.6. To samozřejmě platí i pro počáteční a koncový bod křivky.



Obr. 4.6: Klasická Bézierova křivka a posuv řídícího bodu.

V našem případě se však nevýhoda obrátí ve výhodu, protože nelokálnost změn využijeme ke generování náhodnosti jednotlivých znaků. Citlivým posunem pouze jednoho řídícího bodu tedy můžeme dosáhnout značné změny struktury znaku, ovšem v závislosti na jeho složitosti. Příliš složitý znak bude totiž složen z více takových křivek, takže se změny projeví v menší míře. Naopak u jednodušších písmen se tato malá změna projeví ihned a může mít velmi solidní efekt.

### 4.3.2 Další vlastnosti Bézierovových křivek

Mezi další vlastnosti těchto křivek můžeme zařadit zvýšení stupně polynomu, kdy po přidání každého dalšího bodu tohoto řídícího polynomu se k sobě křivka a polynom postupně přibližují. Naopak snížením stupně (tzn. snížením počtu řídících bodů) se snižuje pracnost výpočtu křivky. Z toho plyne jasná výhoda pro zvýšení výkonu v případě aplikace do programu běžícím na počítači.

Právě proto se v reálném užití často počítá s jednoduššími tvary, které na sebe plynule navazují. Navíc změna polohy řídícího bodu v jednom ze segmentů neovlivní tvar křivky v ostatních segmentech. Více o této problematice bude popsáno na následujících stranách.

Pro možnost algoritmizace vytváření Bézierovových křivek se používá rekurzivní vyjádření Bernsteinových polynomů stupně $n$ za pomoci lineární kombinace dvou po sobě jdoucích Bernsteinových polynomů stupně $n - 1$, jak uvádí [2]. Vzorec potom

vypadá následovně:

$$B_{n,i}(t) = (1-t)B_{n-1,i}(t) + tB_{n-1,i-1}(t) \tag{4.11}$$

Další možností pro generování Bézierovových křivek stupně $n$ je algoritmus de Casteljau, který bude předveden a popsán později.

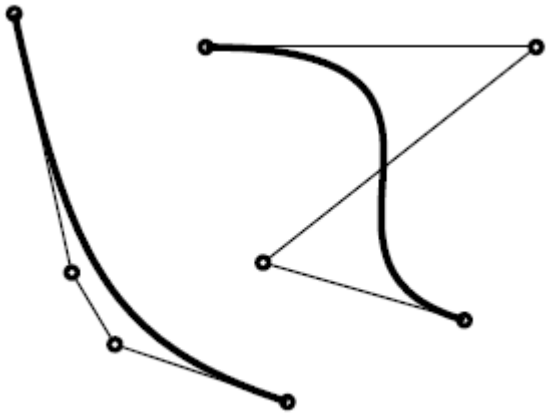
### 4.3.3 Využití Bézierovových křivek

Jak bylo naznačeno již v úvodu této kapitoly, využití tento typ křivky nachází především jako aproximační křivka v praxi technického rázu, k čemuž řadíme i tvorbu fontů. Mimojiné je taková křivka použita při tvorbě jazyka PostScript nebo písma TrueType. [2]

Jednoznačně nejrozšířenějším typem je potom Bézierova kubika, která bude vzápětí popsána více podrobně, nebo také Bézierova kvadrika.

### 4.3.4 Bézierovy kubiky

Bézierova kubika je zadána čtyřmi body $P_0$, $P_1$, $P_2$ a $P_3$. Vychází z prvního bodu $P_0$ a končí v posledním bodě $P_3$.



Obr. 4.7: Ukázky Bézierových křivek — Bézierova kubika.

Již bylo řečeno, že se jedná se o nejčastější a nejpoužívanější variantu Bézierových křivek díky vyváženosti její složitosti v poměru k výpočetní náročnosti. Jak sám název napovídá, kubika pochází ze slova kubický, což je v algebře často spojováno s rozměrem typu $n^3 = n \times n \times n$. Pokusím se tedy o detailnější a hlubší analýzu tohoto vhodného a rozšířeného druhu vektorové grafiky.

Do vzorce Bernsteinových polynomů $B_{k,n}(t)$ tedy dosadíme hodnotu $n = 3$, kde po dosazení dostaneme:

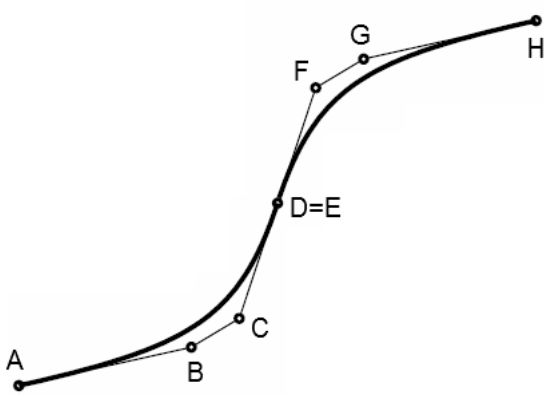
$$q^{Bez}(t) = P_0 B_{0,3}(t) + P_1 B_{1,3}(t) + P_2 B_{2,3}(t) + P_3 B_{3,3}(t) \tag{4.12}$$

kde $P_0$, $P_1$, $P_2$, $P_3$ jsou čtyři řídící body, jak bylo uvedeno výše. Tuto rovnici lze zapsat i v maticovém formátu, který bude vypadat následovně:

$$q^{Bez}(t) = [t^3\, t^2\, t\, 1] \left| \begin{array}{cccc} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{array} \right| \left| \begin{array}{c} P_0 \\ P_1 \\ P_2 \\ P_3 \end{array} \right| . \tag{4.13}$$

### 4.3.5 Plynulé navazování Bézierových kubik

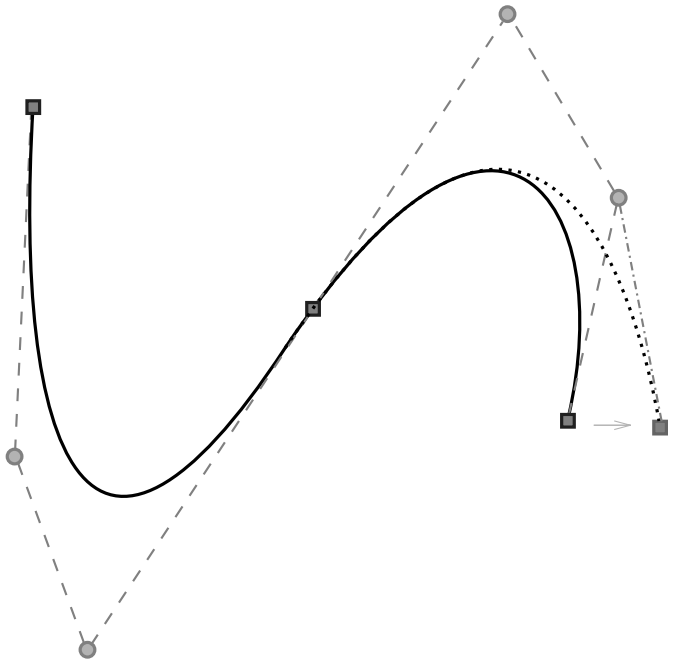
Pro plynulé navazování Bézierových kubik nám postačí, aby splynul první a poslední bod sousedních segmentů ($D = E$) a aby body $C, D = E, F$ ležely v jedné přímce — ta je potom tečnou právě v bodě $D = E$. Pro zachování stejné křivosti obou segmentů ve společném bodě $D = E$ musí tento bod být středem úsečky $CF$ (viz obrázek 4.8 a [13]).




Obr. 4.8: Napojování Bézierových kubik.

Jak již bylo zmíněno dříve, v praxi se často používá více jednoduchých segmentů, které na sebe navazují. Změna řídícího bodu ve vedlejším segmentu potom neovlivní další segmenty, samozřejmě za předpokladu, že se nejedná o společný řídící bod nebo první bod následující po společném řídícím bodu.

Ty totiž musí pro zachování původního segmentu zůstat na stejném místě tak, aby byl poslední bod segmentu a první bod následujícího segmentu byly uprostřed úsečky $CF$ tvořené předposledním řídícím bodem původního segmentu a druhým bodem následujícího segmentu, jak ukazuje obrázek 4.8.



Obr. 4.9: Napojení Bézierových křivek a změna polohy řídího bodu, která nemá vliv na změnu vedlejšího segmentu.

V případě posunu společného řídícího bodu nebo bodu následujícího dojde k efektu, který není žádoucí. Navázání křivek nebude plynulé a bude působit poměrně nepřirozeně a neesteticky. Jednoduché ilustrace 4.9 a 4.10 zobrazují právě tuto problematiku.

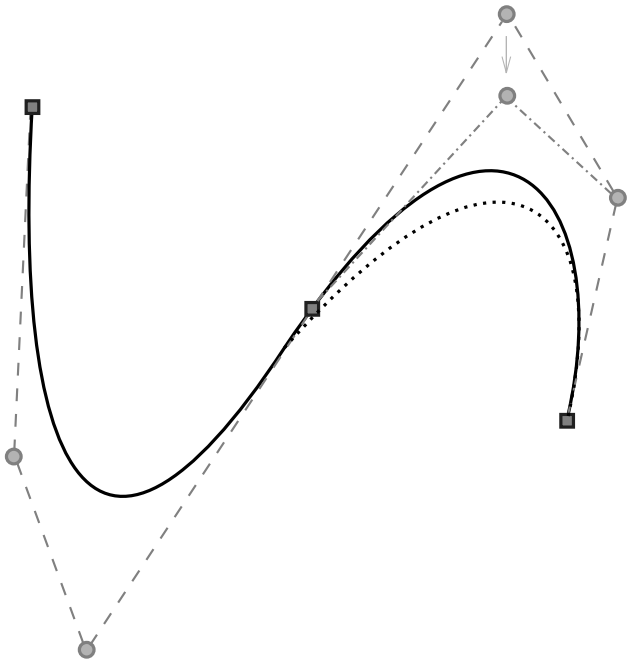
V tomto projektu se řídící body budou upravovat tak, aby nedošlo k nežádoucím efektům, jako například zub, utvořený posunem společného řídícího bodu napojených jednoduchých křivek.

### 4.3.6 Algoritmus de Casteljau

Tento algoritmus slouží k výpočtu hodnoty křivky v zadaném časovém okamžiku [14]. Alternativně umí tuto hodnotu najít i v grafické podobě.

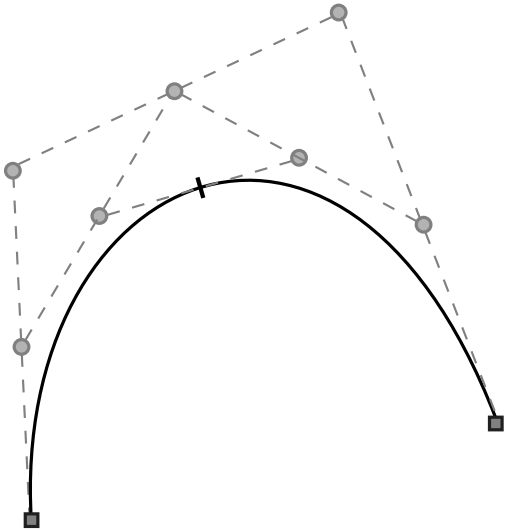
Zásadní pro aplikaci tohoto postupu je použití rekurze. Na vstup přicházejí řídící body $P_0, P_1, ..., P_n$ a časový okamžik $t^* \in [0, 1]$, ve kterém chceme získat bod Bézierovy křivky. Každý takový bod je získán v $n$ krocích.

Popis jednoho ze způsobů aproximace je následující: intuitivní výpočet spočívá především v generování křivky se zvětšujícím se parametrem $t$. Tento přístup ovšem povede k příliš dlouhým segmentům křivky, tedy ke zbytečně řídkému dělení tam, kde je křivka rovná a postačilo by ji aproximovat delšími úsečkami a naopak málo podrobnému dělení v případě velkých oblouků. Efektivní výpočet tak spočívá v rekurzívním dělení křivky tak dlouho, dokud křivka není dostatečně hladká, což lze



Obr. 4.10: Chybné napojení Bézierových křivek a změna polohy řídího bodu, která má vliv na změnu vedlejšího segmentu.

zjednodušeně pozorovat na obrázku 4.11. Ten totiž zobrazuje nejjednodušší způsob dělení, neboli půlení úseček řídícího polynomu.



Obr. 4.11: Ukázka algoritmu de Casteljau půlením úseček.

Bod křivky se s pomocí výpočetní techniky určí dle následujícího rekurzivního vztahu:

$$P_{i,j}(t) = (1-t)P_{j-1,i-1}(t) + tP_{j,i-1}(t) + i = 1, 2, ..., n; j = i, i+1, ..., n \qquad (4.14)$$

Pokud není dosažený průběh Bézierovy křivky tím pravým, který měl vzniknout, je vhodné zvýšit stupeň řídícího polynomu a vytvořit tak nový polynom. Ten bude mít o jeden bod více, i když prakticky popisuje tutéž křivku. Potom pro libovolný bod z $n$+2 bodů $Q_0, Q_1, ..., Q_{n+1}$ nového polynomu tvořeného $n$+1 body $P_0, P_1, ..., P_n$ původního řídícího polynomu platí:

$$Q_i = \frac{i}{n+1}P_{i-1} + (1 - \frac{i}{n+1}P_i), i = 0, 1, ..., n+1 \tag{4.15}$$

### 4.3.7 Shrnutí Bézierových křivek

I když jsou Bézierovy křivky velmi vhodné pro kreslení 2D grafiky, a najdou tedy užití v tomto projektu, při přechodu k 3D grafice se začnou projevovat jejich vlastnosti, které nejsou žádoucí a i díky tomu se v dnešním software používají spíše složitější NURBS křivky [14].

Pro tento projekt se však Bézierovy křivky stanou vhodným nástrojem pro prezentaci algoritmů určujících náhodnost a generujících znakové sady za definovaných podmínek.

Jejich napojování se potom projeví na kvalitě znaku. Zkusíme docílit takové náhodnosti, aby bylo možné při relativně malých změnách dosáhnout změny struktury fontu při zachování dostatečné kvality a ladnosti přechodů mezi jednotlivými kubikami.

## 4.4 Rasterizace vektorové grafiky

Obraz vektorové grafiky je tvořen segmenty, které jsou definovány pomocí úseček a křivek. U těch pak definujeme jejich počáteční bod, směr a délku. Tento pro počítače výhodný způsob uložení dat s sebou ale přináší také jistá omezení a neduhy, mezi které můžeme zařadit například velký objem dat pro grafické soubory s větším počtem přímek a tím pádem také pomalejší odezvu (neboli nároky na paměť systému). Další nevýhoda je složitost převodu z rastrové do vektorové grafiky. U opačného převodu je situace jiná — v případě, že máme obrázek ve vektorovém formátu, můžeme jej relativně snadno převést do formátu bitmapového. Tento převod se často používá například v návrhových CAD systémech, kde dochází k exportu určitých objektů.

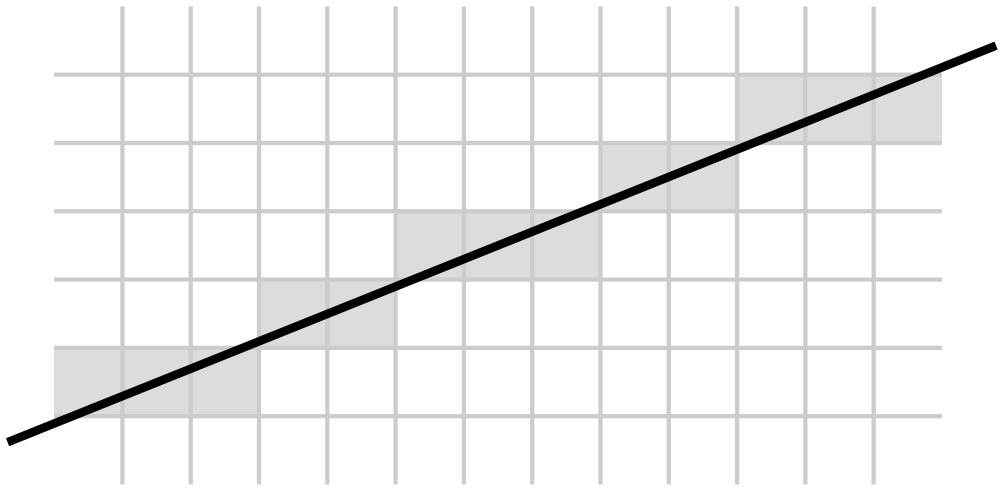
I když dnes ve výpočetní technice převládá především především rastrové (bitmapové) zpracování grafiky a jak bylo řečeno, je pro počítače výhodnější ukládat data jednodušší počítačové grafiky, jakou jsou i outline fonty, ve vektorovém formátu obrazové informace, protože tato informace bude zabírat daleko méně místa na disku,

potažmo v paměti systému. Není tak potřeba ukládat vzhled znaku, ale pouze informaci, o který znak se jedná. Způsob jeho vykreslení si již vezme na starost font, který zná jednoznačná kritéria, jak má který znak vypadat.

Moderní tiskárny mají často implementovány algoritmy, které hravě zvládají poměrně rychle zpracovávat velké množství vektorové grafiky, a tím pádem také textu, který tak svižně dostanou na obyčejný papír. Navíc mají ve své paměti typu ROM obvykle uloženy nejčastěji používané fonty k ještě rychlejšímu zpracování. U dalších výstupních zařízení není situace kolem vektorové grafiky tak jednoznačná. Dnešní monitory grafiku včetně písma převádějí do rastrového (bitmapového) formátu a tak je vždy potřeba zajistit velmi rychlý převod z vektorového formátu, aby vykreslení proběhlo v krátké době, pro lidské oko pokud možno nerozpoznatelné.

### 4.4.1 Rasterizace úsečky

Jelikož je rastrový obraz tvořen posloupností bodů (zatímco vektorový obraz je složen z úseček), bude při převodu vektorové grafiky na bitmapovou grafiku hrát klíčovou roli rychlost v závislosti na kvalitě algoritmu pro převod úsečky na jednotlivé body. Ty nazýváme též pixely a jejich parametry určují kromě souřadnic také barvu. Můžeme říct, že při procesu rasterizace vlastně hledáme částečný nebo úplný popis jednotlivých částí vektorové informace.

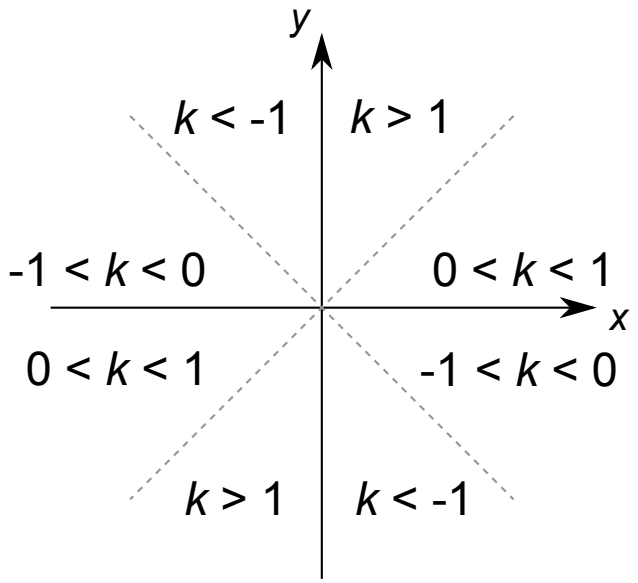


Obr. 4.12: Příklad rasterizace úsesčky (Bresenhamův algoritmus).

Úsečka je po bodu nejjednodušším grafickým prvek (fakticky ji tvoří dva body). Skládáním úseček potom dostáváme složitější útvary, čili logicky i pro vykreslení složitých tvarů bude záležet na rychlosti vykreslení úsečky.

Pro správnou definici úsečky je potřeba určit koncové body v souřadném systému $[x_1, y_1]$ a $[x_2, y_2]$ nebo také můžeme určit počáteční bod $[x_1, y_1]$ a jeho přírůstek

$[\Delta x, \Delta y]$, což odpovídá $[x_2 - x_1, y_2 - y_1]$. Kromě těchto bodů se u úseček může definovat ještě jejich hloubka a popřípadě i jiné parametry. [2]




Obr. 4.13: Různé hodnoty směrnice $k$ pro různé sklony úseček.

Při rasterizaci úsečky dochází k přírůstku s konstantním krokem v závislosti na její směrnici. Ta je dána poměrem přírůstků souřadnic na osách $x$ a $y$.

$$k = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} \tag{4.16}$$

Pokud je $k$ menší než 1, proběhla rasterizace podle osy $x$ s krokem 1, pokud je $k$ větší než 1, pak se rasterizace řídí dle osy $y$ s krokem 1. V případě, že se směrnice rovná 1, pak lze rasterizovat dle osy $x$ nebo $y$. Hlavní osou nazýváme tu, podle které se rasterizuje, druhá osa je potom označována jako vedlejší.

### Algoritmus DDA

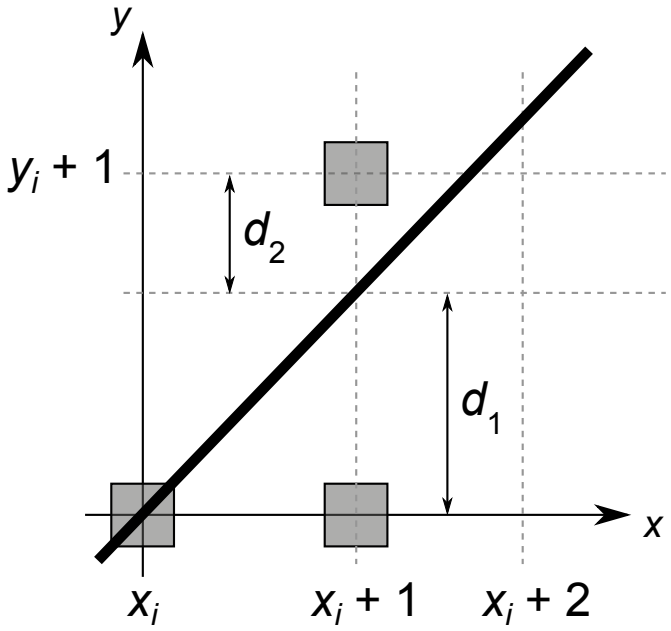
DDA je zkratkou z anglického Digital Differential Analyzer. Jeho základem je lineární interpolace na určitém intervalu mezi počátečním a koncovým bodem. V našem případě se jedná o postupné přičítání přírůstků vycházejících z počátečního bodu. Jeho implementace není nikterak náročná a výpočet probíhá velice rychle. To ovšem odpovídá kvalitě výsledku, který nedosahuje možností jako při použití Bresenhamova algoritmu.

Mluvíme tedy o iteračním algoritmu, kde ke spočítání nových hodnot souřadného systmému využijeme předcházející souřadnice. Pro řídící osu $x$ potom výpočet vypadá takto:

$$x_{k+1} = x_k + 1 \qquad y_{k+1} = y_k + m \tag{4.17}$$

**Bresenhamův algoritmus**

Tento algoritmus spolehlivě určuje, které body rastrového formátu budou vykresleny jako části kýžené rovné úsečky spojující dva body. Využívá k tomu celočíselné aritmetiky (sčítání, odčítání a bitový posuv), což je velmi efektivní a nenáročné vzhledem k výpočetnímu výkonu zařízení (počítače). Vyvinut byl mezi vůbec prvními algoritmy pro práci s grafikou.




Obr. 4.14: Detail volby pixelu na základě průběhu úsečky.

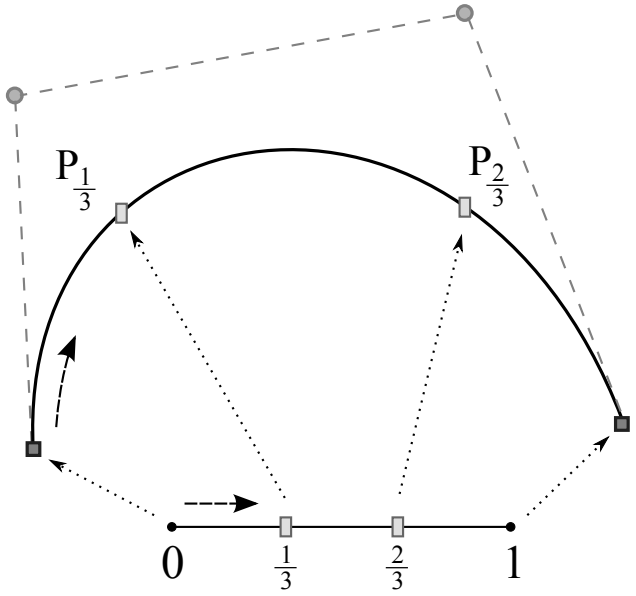
Pokud bereme v potaz směrnici $k$, která určuje osu $x$ jako hlavní, potom práce algoritmu je především rozhodování, jestli se pixel v dalším kroku vykreslí na stejné souřadnici $y$ nebo na souřadnici o jednu vyšší/nižší v závislosti na směru úsečky. Obrázek 4.14 ukazuje, že výběr souřadnice závisí na vzdálenosti bodu od hodnoty souřadnice.

V praxi potom narazíme na různé anti-aliasingové filtry, které vylepšují celkový dojem z použitého fontu a samozřejmě mají také kladný vliv na čitelnost textu, což popisuje i literatura [16] a [3].

### 4.4.2 Rasterizace Bézierovy křivky

Pro rasterizaci křivek je možné využít některou z již popsaných metod výpočtu, ať jde o Bernsteinovy polynomy, nebo ideálně algoritmus de Casteljau, který se jeví jako nejvhodnější varianta rasterizace. Časový průběh je znázorněn na vizualizaci 4.15, kde je jasně vidět, ze s postupem doby na časové ose se generují jednotlivé body křivky směrem od počátku až do koncového řídícího bodu.

Tento algoritmus je rekurzivní, takže opakuje své kroky dokud není splněna určená podmínka. Ta v tomto případě bude nastavena na výsledek vzdálenosti dvou sousedních bodů po dělení. Jakmile je menší než velikost úhlopříčky pixelů, je další dělení zbytečné a rekurze se začne vynořovat, jak uvádí [2]. Nastavení kritérií je možné upravovat dle potřeby, například podle požadovaná hladkosti křivky.




Obr. 4.15: Rasterizace pomocí algoritmu de Casteljau.

# 5 IMPLEMENTACE

Pro implementaci se nabízí několik možností (programovacích jazyků), které bez problémů zvládají práci s grafickými prvky, ať již nativně nebo za použití dostupných knihoven.

Pro tento projekt jsem jako nejvhodnější variantu zvolil program Matlab díky jeho podpoře Bézierových křivek. Tím odpadne potřeba programovat Bézierovy křivky a lze se plně soustředit na implementaci oživení fontu.

Primárně byl Matlab určen pro matematické účely jako interaktivní nadstavba, která zvládala nejrůznější manipulace s maticemi (MatLab — matrix laboratory, neboli maticová laboratoř). S postupem času se však začal rozvíjet v komplexní aplikaci vhodnou k řešení široké palety nejrůznějších problémů. Dnes spousta odvětví stojí na matematických modelech a složitých výpočtech, mimo jiné proto se toto prostředí stalo tak rozšířeným.

Systém postupně získal schopnost řešit určité dynamické systémy, vytvářet nejen grafy, ale i kompletní grafické uživatelské prostředí, interaktivně editovat a skládat dynamické bloky a řešit matematické problémy analyticky.

## 5.1 Programovací prostředí v Matlabu

Jedná se o programové prostředí a skriptovací jazyk čtvrté generace, který umožňuje relativně snadné počítání s maticemi, implementaci algoritmů, vykreslování dvourozměrné grafiky, ale i analýzu a vyhodnocování dat. Obsahuje navíc stovky předpřipravených funkcí, včetně např. funkcí pro práci s grafikou, což se bude velmi hodit právě při řešení tohoto tématu.

Výhodou programovacího prostředí Matlab je jistá podobnost s tradičním a velmi rozšířeným programovacím jazykem C a má tak poměrně jednoduchou a efektivní syntax. Odlišnosti, které mezi nimi můžeme nalézt, jsou víceméně k užitku, např. není nutné deklarovat typ proměnných nebo určovat jejich rozměr.

Matlab je jazyk, ve kterém záleží na velikosti písmen proměnných a funkcí, jedná se tedy o tzv. „Case-sensitive“ prostředí.

### 5.1.1 Proměnné v Matlabu

Jak již bylo uvedeno, Matlab má slabou dynamickou typovou kontrolu a tak proměnné nemají po deklaraci určený datový typ a mohou jej měnit během své existence v rámci běhu programu. Je tak možné např. vytvořit proměnnou typu integer a následně do ní uložit desetinné číslo. Čtení z neexistující proměnné však vyvolá chybu a je tedy dobré těmto událostem předcházet.

Mezi další výhody řadím výbornou práci s pamětí, kdy při prvním použití jména je vytvořena proměnná a je jí přidělena paměť. Matlab je tedy schopen sám alokovat potřebné proměnné. Pokud však již proměnná existuje, je pouze změněna její hodnota, popř. její rozměr.

### 5.1.2 Práce s maticemi

Jak již bylo uvedeno, Matlab je zaměřen především na snadnou a pohodlnou práci s maticemi. Pro jednoduché definování matice stačí vložit její prvky mezi symboly [ ]. Jednotlivé prvky, většinou čísla, se oddělují ve svislém směru znakem konce řádky nebo středníku a ve vodorovném směru znakem mezera nebo čárka. Matice může mít prakticky libovolný počet rozměrů

```
» A = [11,12;21,22]
A =
11 12
21 22
```

Obr. 5.1: Příklad definice matice $A$ v prostředí Matlab.

Speciálním typem matice je potom skalár a vektor. Skalár je matice o rozměrech $1\times1$, zatímco vektor je matice rozměrů $m\times1$ nebo $1\times n$, což značí, že vektor může být jednak sloupcový nebo řádkový. Důležitou a užitečnou vlastností je možnost skládat nejen číselné hodnoty a skalární proměnné, ale i možnost skládání a rozšiřování matic. Vždy je ovšem nutné dodržet plný obdélníkový nebo čtvercový tvar výsledné matice.

```
v=[1;2]
u=[3,4]
```

Obr. 5.2: Sloupcový vektor $v$ a řádkový vektor $u$.

### 5.1.3 Další vlastnosti Matlabu

Podporovány jsou funkce s různým počtem vstupních a výstupních parametrů. Definice funkcí jsou nadefinována formální jména vstupních a výstupních parametrů. Při volání funkce poté není nutné naplnit všechny vstupní parametry nebo ukládat hodnoty vrácené do výstupních parametrů. Tělo funkce potom tyto vstupní parametry musí ignorovat, jelikož nebudou definovány.

Dobrým zvykem ve všech programovacích jazycích je komentování zdrojového kódu. Je tedy dobré uvádět poznámky především na místech, kde není zcela jasná funkce jisté části programu. Komentáře se v Matlabu definují znakem procento (%). Všechen obsah za tímto znakem je posléze ignorován při běhu programu.

Čerpáno bylo z knihy [6], ale i odborného článku [11].

## 5.2 Práce s programem Matlab

Výsledky této práce budou prezentovány jako grafický výstup z aplikace Matlab. Ten totiž, jak již bylo řečeno, poskytuje v tomto směru značné možnosti vizualizace dvojrozměrné grafiky.

Výhodou programu jsou také jeho možnosti přenositelnosti obrazových výstupů do jiných aplikací. Nabízí se zde hned několik vhodných variant exportu, přičemž mezi nejzajímavější patří extenze „jpg“ a „pdf“, jak ostatně uvádí i [6].

Dále bude navíc demonstrován zdrojový kód, který implementuje dříve popsané techniky tvorby fontu pomocí Bézierových křivek, ale také docílení efektu náhodné struktury jednotlivých znaků.

Jelikož by tvorba celé abecedy byla časově velmi náročná, byly pečlivě vybrány jen některá písmena české abecedy, která mohou vhodně utvořit základní slova, popř. sousloví a jednoduché věty.

### 5.2.1 Tvorba křivek

Jak již bylo zmíněno dříve, tvorba jednotlivých znaků je založena na Bézierových křivkách. Tyto je však potřeba nějakým způsobem definovat (popsat) přímo v programu a uložit k pozdějšímu zpracování algoritmem. Zde se nabízí využití maticového zápisu souřadnic jednotlivých bodů, jak je naznačeno v následujícím vyobrazení 5.1 pro znak „C“:

```
function out = data_char_C

out = ...
   [10  4  0  1;  10  0  0  1;   0  0  0  1;   0  4  0  1;  ...
     0  4  0  1;   0  8  0  1;  10  8  0  1;  10  4  0  1;  ...
    10  4  0  1;  10  4  0  1;   8  4  0  1;   8  4  0  1;  ...
    10  4  0  1;  10  4  0  1;   8  4  0  1;   8  4  0  1;  ...
     8  4  0  1;   8  1  0  1;   2  1  0  1;   2  4  0  1;  ...
     2  4  0  1;   2  7  0  1;   8  7  0  1;   8  4  0  1];
```

Listing 5.1: Maticový zápis řídících bodů, postupně zleva $P_0, P_1, P_2$ a $P_3$

## 5.3 Generování výsledků

Návrh fontu bude spočívat nejprve v jeho sestavení v souřadnicovém systému za pomoci Bézierových křivek, což bylo prezentováno v předchozích kapitolách. Jednoduché znaky se spojí z několika křivek, a ty posléze budou algoritmicky upravovány přímo v souřadnicovém systému na osách $x$ a $y$.

Výsledek potom bude grafický výstup, který zobrazí uživateli zvolené znaky, slova nebo ucelené věty. Ty by mohl eventuálně upravovat na základě zvolených parametrů, např. agresivnější pojetí, zlatá střední cesta nebo mírnější náhodnost.


Obr. 5.3: Slovo s oživeným fontem při citlivém pojetí.

Příklad úhledně zpracovaného výstupu lze spatřit na obrázku 5.3, kde vidíme jednotlivé varianty slova „MOST“. To je psáno kapitálkami a v jeho první variantě není nikterak upraveno. V dalších dvou variantách jsou již upraveny znaky zcela náhodným způsobem, přičemž je zvolena citlivější forma generování náhodného efektu. Za povšimnutí stojí jednotlivé znaky, které se jednak mírně naklánějí, nebo případně stahují a roztahují (patrné především u znaku „M“).

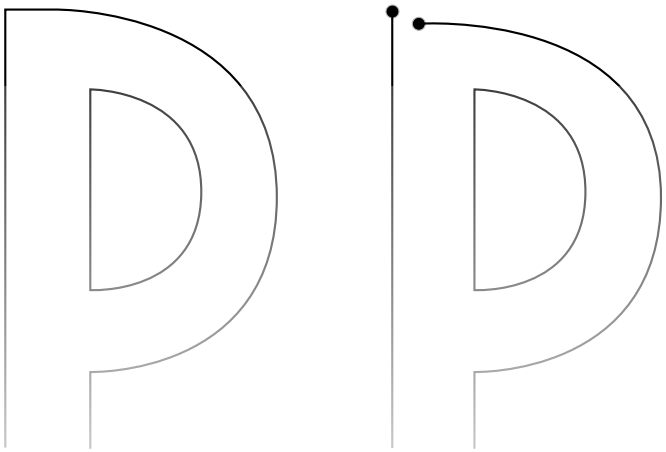
Tyto výsledky budeme dále skládat do celých vět a tím zkompletujeme prezentaci náhodného efektu na různých jednoduchých příkladech.

## 5.4 Oživení znakové sady

K oživení fontu potom může dojít několika způsoby. Jedním z nich je posun řídících bodů Bézierových křivek zapsaných v matici, kde je ovšem potřeba dbát na jistá pravidla, aby nedošlo ke znehodnocení znaku. Příkladem takového znehodnocení může být „rozpojení“ křivek, kdy dojde k posuvu počátečního řídícího bodu křivky a tím pádem také nesouhlasu s koncovým řídícím bodem předcházející křivky 5.4. To způsobí jednak nevzhledné vizuální efekty, ovšem také problém při následném vyplnění (vybarvení) obsahu znaku.

Těmto chybám jsem se snažil předcházet úpravou algoritmu tak, aby samostatně upravoval pouze ty řídící body, které nejsou nikterak vázány na ostatní řídící body. Naopak, pokud mají dva řídící body stejné souřadnice, je potřeba je posunout oba o přesně stejnou vzdálenost.

Operace s písmeny, popř. číslicemi je možné řešit také díky násobení řídicích bodů maticemi. Tyto matice ovšem musí být rozměru $3 \times 3$ a navíc je nutné s nimi



Obr. 5.4: Nežádoucí efekt rozpojení křivek u písmene P.

pracovat v tzv. homogenních souřadnicích, které byly podrobněji popsány v dřívější kapitole. Výhodou je však jednotný výpočet veškerých geometrických transformací, jak je ostatně popsáno v částech 4.6 až 4.8 [14].

## 5.5 Způsob transformace

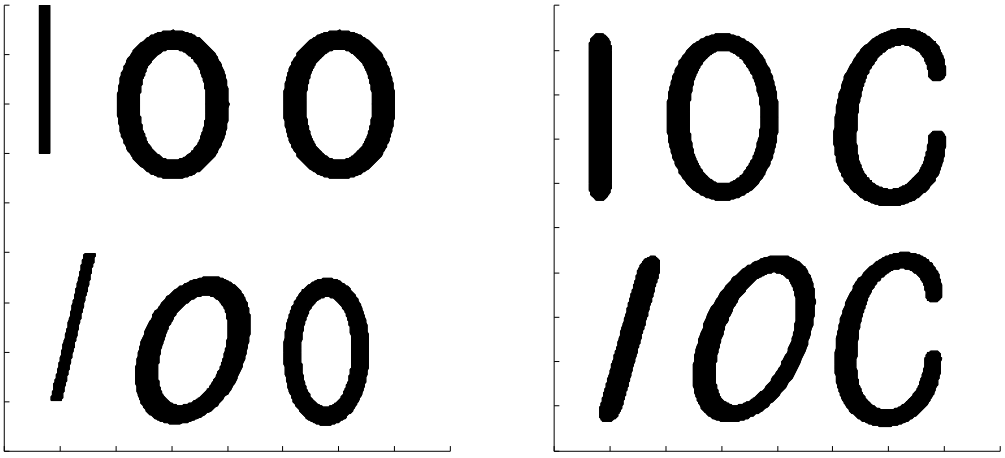
Po zhodnocení výsledků semestrálního projektu bylo zjevné, že přílišná živost patkového písma, jakým je např. Times New Roman, vede k efektům spíše nežádoucím. Tento decentní font se totiž jevil neupraveně a pocit z něj evokoval spíše chaos. Jak se tedy ukázalo, nebyl vhodnou volbou.

Po zralé úvaze jsem zkusil vytvořit bezpatkovou znakovou sadu podobnou fontu Arial a také variantu bezpatkového fontu, jakoby psaného rukou, podobného Comic Sans. Algoritmy jsem posléze spouštěl na těchto dvou znakových sadách a porovnával výsledky, jak je možné spatřit v grafu 5.5.

Důležitým prvkem při tvorbě znaků byl efektivní zápis jednotlivých souřadnic řídících bodů Bézierových křivek. K tomu se v Matlabu jednoznačně jeví jako nejvhodnější datový typ matice, které velmi dobře podporuje, jak již ostatně bylo popsáno dříve.

Každá takováto matice potom reprezentuje jeden znak v jeho základním tvaru, tzn. zatím nikterak upravený nebo oživený. Některé znaky jsou logicky jednodušší a jiné zase složitější, proto se nedá jednoznačně určit velikost univerzální matice. Ta se bude lišit v závislosti na složitosti znaku, přičemž platí, že čím složitější znak je, tím bude složitější zápis matice.

Jak se také ukázalo, je vhodné nastavovat pro každý znak mírně odlišné parametry transformací. Proto jsem ve zdrojovém kódu vytvořil rozhdovací strom za



Obr. 5.5: Výsledek transformace bezpatkového písma typu Arial (vlevo) a psaného písma typu Comic Sans.

použití příkazu *switch*, který si poté s jednotlivými znaky poradí dle předem určených pravidel (zjednodušenou ukázku lze spatřit v tomto zdrojovém kódu 5.2).

```
switch char
    case 'i',
    case 'I',
        points = get_data_char(I);
        width = 8;
        t_a = shear_level_prev - 0.15;
        t_b = shear_level_prev + 0.15;
        shear_level = gen_rand(t_a, t_b);
        t_a = scale_level_prev - 0.1;
        t_b = scale_level_prev + 0.1;
        scale_level = gen_rand(t_a, t_b);
        rotate_level = gen_rand(t_a, t_b);
        t_a = kern_level_prev - 0.2;
        t_b = kern_level_prev + 0.2;
        kern_level = gen_rand(t_a, t_b);
    case 'o',
    case 'O',
        points = get_data_char(O);
        width = 14;
        t_a = shear_level_prev - 0.1;
        t_b = shear_level_prev + 0.1;
        shear_level = gen_rand(t_a, t_b);
        t_a = scale_level_prev - 0.15;
        t_b = scale_level_prev + 0.15;
        scale_level = gen_rand(t_a, t_b);
        t_a = rotate_level_prev - 10;
```

```
        t_b = rotate_level_prev + 10;
        rotate_level = gen_rand(t_a, t_b);
        t_a = kern_level_prev - 0.1;
        t_b = kern_level_prev + 0.1;
        kern_level = gen_rand(t_a, t_b);
    .
    .
    .
end
```

Listing 5.2: Příklad nastavení transformačních parametrů pro různé znaky.

### 5.5.1 Blokové schéma prováděných transformací

Celkový způsob oživení fontu je nejlépe vidět na obrázku 5.6, který ukazuje blokové schéma jednotlivých transformací a dalších operací přehledně tak, jak jsou postupně zpracovávány algoritmem.

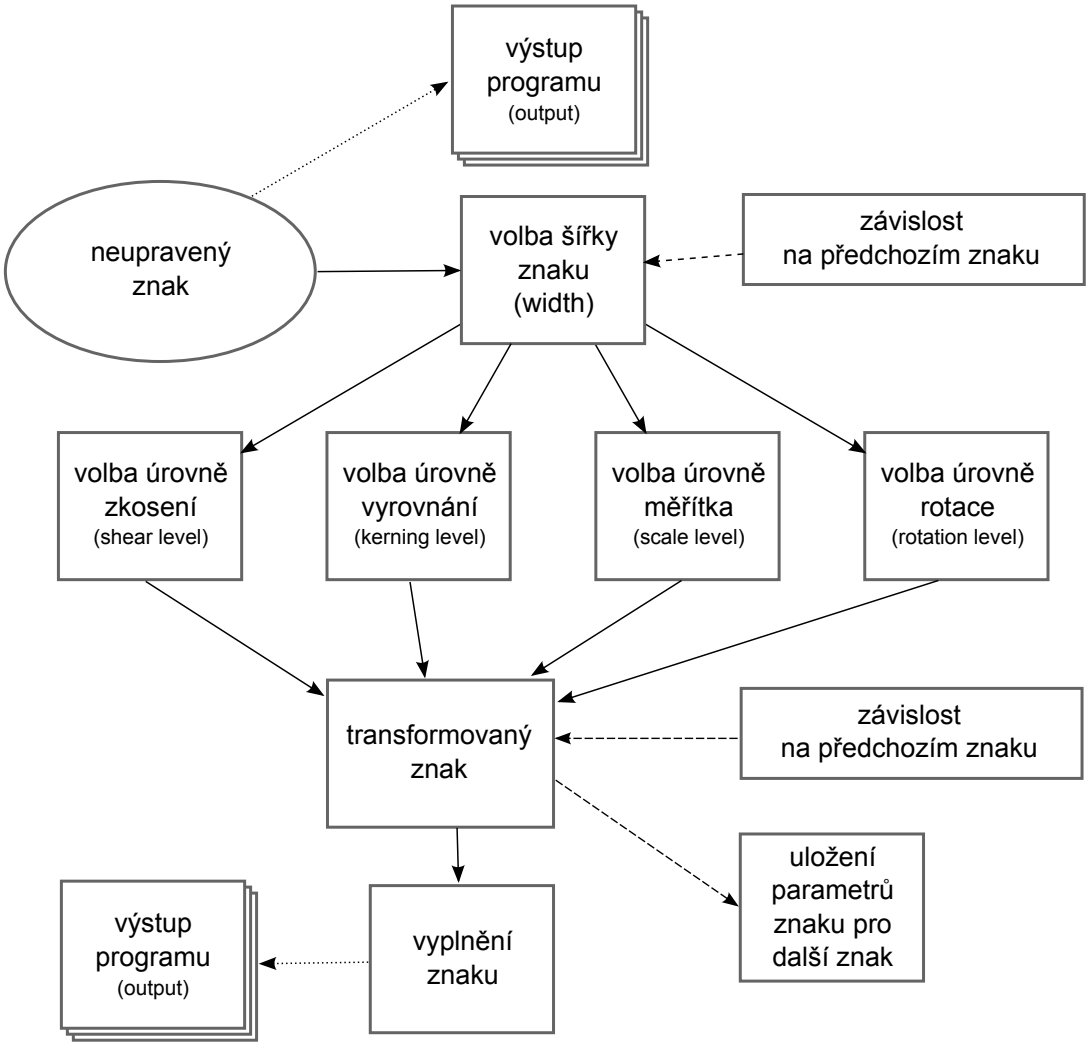
Celý proces začíná vstupem znaku do algoritmu. Jelikož je tvorba znaků časově náročná, nebyla vyrobena kompletní abeceda, a výběr tak není zcela libovolný. Z předdefinovaných znaků ovšem můžeme skládat alespoň základní jednoduchá slova.

Transformace jsou prováděny po jednotlivých znacích tak, jak jsou ve slově uspořádány za sebou, viz obrázek 5.7. Každý takový znak se nejprve zobrazí na výstupu programu neupravený a zároveň se ten samý znak pošle do algoritmu zpracovávajícího transformace. Ten potom zvolí nejdříve vhodnou šířku pro dané písmeno, na kterou může mít vliv i předchozí znak. Dále program vybírá vhodné nastavení úrovně zkosení, úrovně vyrovnání (kerning), úrovně měřítka a úrovně rotace, opět v závislosti na nastavení předchozího písmene tak, aby nedošlo k nevzhledným jevům. Ještě v průběhu zpracování znaku se ukládají hodnoty nastavených úrovní, které se použijí při zpracování dalšího znaku. Výsledný transformovaný znak poté dostane výplň (pokud je tak nastaveno uživatelem) a zobrazí se na výstupu, kde může být porovnán s již dříve zobrazeným neupraveným znakem.

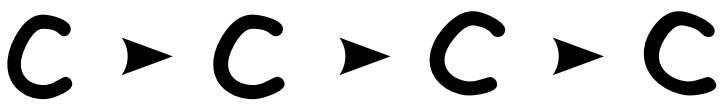
## 5.6 Uživatelské prostředí

Vlastní program se spouští souborem generate_chars.m, který na vstup přijme určité parametry od uživatele. Ty poté při běhu algoritmu zajistí různorodé chování transformací (obr. 5.8), jak je vysvětleno dále.

Znaky se dají v algoritmu měnit, každý znak je uložen ve svém souboru, pojmenovaném data_char_C.m (pro písmeno C) — to je varianta bezpatkového fontu



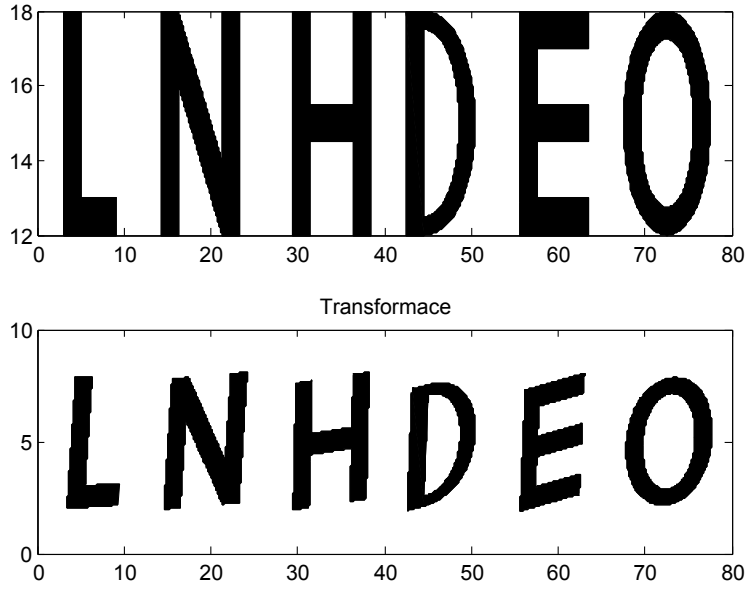

Obr. 5.6: Blokové schéma naznačující postup algoritmu při transformaci znaků.



Obr. 5.7: Kroky transformace, zleva změna měřítka (roztažení), otočení a nakonec zkosení.

(podobného Arialu) anebo data_char_C_II.m - varianta bezpatkového fontu „psaného“ podobného Comic Sans.

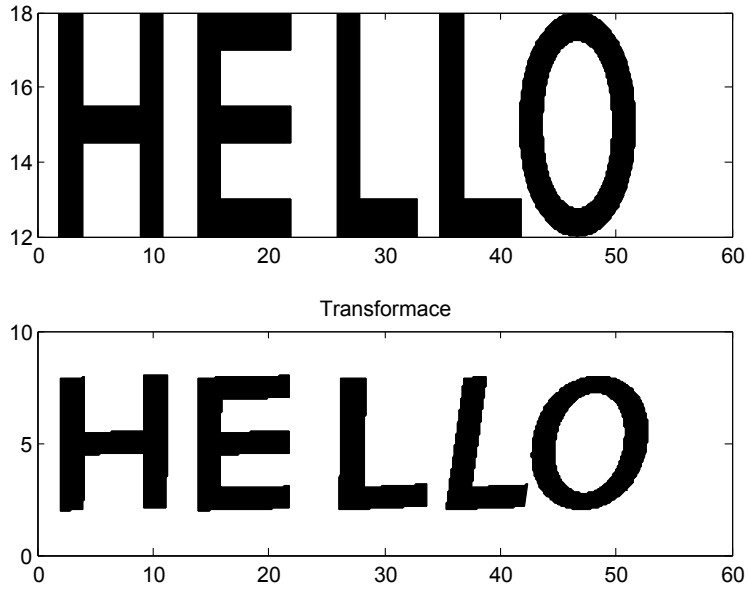
Při samotném návrhu algoritmu jsem přemýšlel, jak jej vylepšit, popř. jaké funkce a nastavení přidat pro uživatelův komfort a pohodlí. Tyto návrhy pro případné vylepšení zadání jsem posléze jednotlivě prošel, zamyslel se nad jejich podstatou, konzultoval s vedoucím práce a postupně zkoušel implementovat.




Obr. 5.8: Výstup programu v podobě dvou subplotů při nastavení náhodného řetězce.

### Zobrazování výsledků

Pro efektivní a lepší srovnání výsledků transformací jednotlivých znaků výstup ukazuje řetězec před transformací a po transformaci. Uživatel tak má jasný přehled o tom, co se při úpravě fontu změnilo. K tomu bylo využito dvou podgrafů, v Matlabu známých jako „subplot“. S těmito vzniklými okny může uživatel libovolně pohybovat po obrazovce a srovnávat si tak výsledky oživení fontu.




Obr. 5.9: Na výstup lze poslat i slovo.

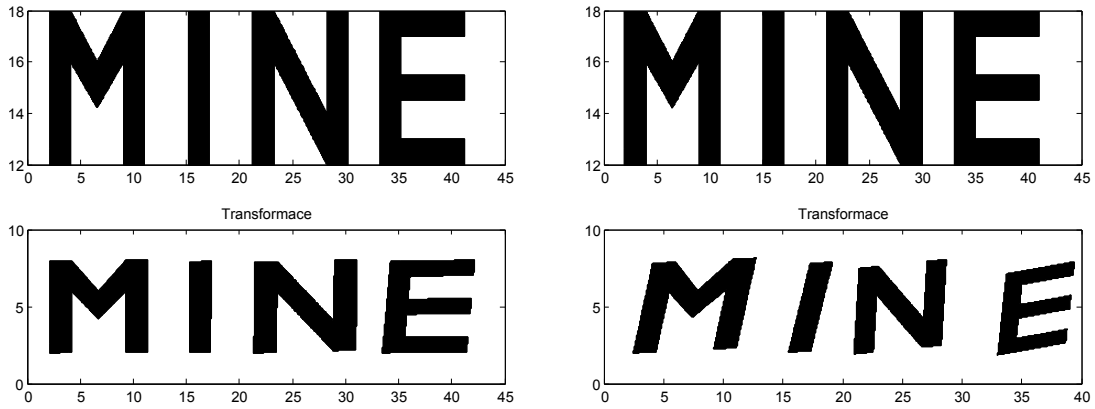
Původní návrh vykresloval neupravený znak a pod něj transformovaný znak přímo do jednoho grafu (v Matlabu pojmenovaného „plot“), což se ukázalo jako ne příliš efektivní řešení. Proto bylo i na doporučení vedoucího využito raději výše zmíněného řešení, jak lze vidět v grafu 5.9.

### Vstupní řetězce

Uživatel může na vstupu programu zadávat řetězec znaků (popř. slov), které chce algoritmem transformovat. Pokud však zadá nepodporovaný znak, program jej na to upozorní a neproběhne.

### Ovládání míry tranformace

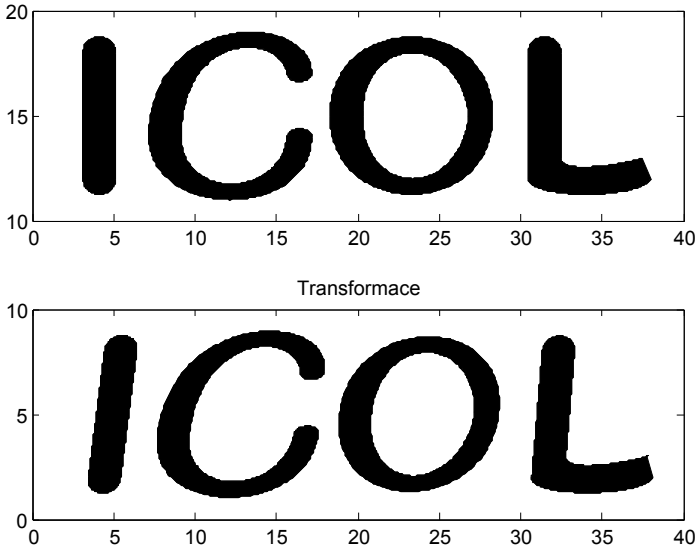
Uživatel také může na vstup programu zadávat (a tím měnit) míru transformace, kterou se posléze bude algoritmus řídit. Tuto míru transformace si lze představit jako určitou stupnici od jedničky do desítky, přičemž nula znamená co nejmenší (nejjemnější) transformaci a desítka nejvyšší míru oživení. Pokud se uživatel rozhodne toto nastavení ignorovat, skript obsahuje tzv. základní nastavení, které má hodnotu čísla 5, tedy střední míra transformace. Porovnání míry transformace je možné vidět na obrázku 5.10




Obr. 5.10: Srovnání nastavení nižší míry transformace (vlevo) a vyšší míry transformace.

### Volba fontu

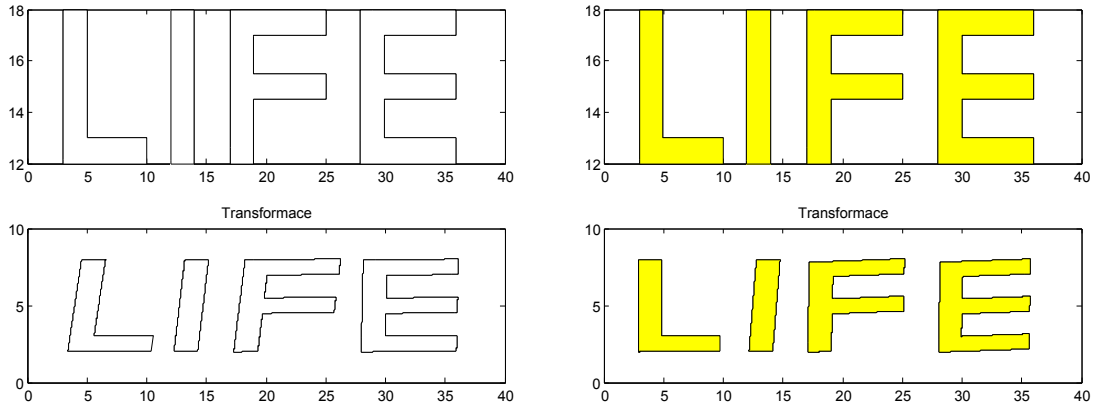
Aby bylo dosaženo co nejrozmanitějších výsledků, je uživateli umožněno volit typ fontu. Na výběr má patkový, bezpatkový a psaný bezpatkový (podobný Comic Sans), který je zobrazen v grafu 5.11. Volbu provádí opět na vstupu programu.




Obr. 5.11: Nastavení psaného fontu.

### Možnost volby výplně

Pro pestřejší paletu výsledných transformací jsem přidal také možnost nastavení výplně znaku. Vyplňování lze tedy zcela vypnout, je ovšem možné také nastavit barvu výplně. Při variantě bez výplně zůstává znak průhledný a je tak vidět pouze jeho obrys, což dobře ilustruje obrázek 5.12.




Obr. 5.12: Různé nastavení výplně znaku.

### Náhodné řetězce

Uvažoval jsem ještě nad dalším možným rozšířením, kterým by mohlo být generování náhodného řetězce znaků (podobného například Lorem Ipsum — viz ukázka 5.13). Lorem Ipsum je pseudolatinský text užívaný v grafickém designu a navrhování,

sloužící pro demonstraci textu při vytváření pracovních ukázek grafických návrhů (například webové stránky). Program by tak na výstup generoval náhodný text. Toto rozšíření jsem však implementoval jen částečně tak, aby alespoň základně vystihovalo myšlenku. Vybírají se tak náhodné znaky z předem určených znaků přímo ve zdrojovém kódu.

> Lorem ipsum dolor sit amet consectetuer facilisi Morbi facilisi tincidunt enim. Habitasse ut Nam Cum sociis scelerisque est et vitae Phasellus ac. Platea convallis parturient justo vel tellus dui libero tristique leo tortor.

Obr. 5.13: Příklad náhodného textu typu Lorem Ipsum.

## 5.7 Zdrojový kód

Při psaní zdrojového kódu počítačového programu je dobré se držet určitých pravidel a konvencí, která usnadní orientaci a zpětné pochopení zdrojového kódu. To využije jedna osoba, která zdrojový kód nepsala a snaží se jej nastudovat, ale také samotný programátor, který aplikaci vytvořil (například za delší časový úsek od napsání kódu).

Pokud píšu zdrojový kód jakékoliv aplikace, vždy se snažím udržovat si v něm pořádek, pojmenovávat proměnné a funkce srozumitelně a psát veškeré komentáře v anglickém jazyce, aby byl program pochopitelný i pro uživatele, který nemluví česky.

Ukázka zdrojového kódu funkce $plot\_several\_bezier\_cubics(h, v, color, fillcolor, plotnum)$ refplotseveral pro vykreslení několika Bézierových kubik v barvě definované parametrem *color* a vyplnění znaku barvou danou parametrem $fillcolor$:

```
% define color for later filling
% could be overwriten like this: 'black';
colorFill = fillcolor;
%colorFill = color; % same as outline
% generate timeline - number of points rendered
t = (0 : 0.01 : 1);
% get matrix length
matrixLength = length(v);
% must be a multiple of 4
```

```
if rem(matrixLength, 4) == 0
    totalX = [];
    totalY = [];
    % loop through curves from input matrix
    for i = 1 : 4 : matrixLength
        a = v(i, 1 : 2);
        b = v(i + 1, 1 : 2);
        c = v(i + 2, 1 : 2);
        d = v(i + 3, 1 : 2);
        [x, y] = cubicbezier(a, b, c, d, t');
        totalX = [totalX; x];
        totalY = [totalY; y];
        hold all;
        figure(h);
        subplot(2, 1, plotnum);
        plot(x, y, color);
    end
    %fill the character using defined color
    fill(totalX, totalY, colorFill);
else
    errInfo = 'Input vector is not complete!';
    disp(errInfo);
end
```

Listing 5.3: Ukázka nastavení transformačních parametrů pro různé znaky.

## 5.8 Možnosti dalšího rozšíření aplikace

Při úvahách o možnostech dalšího praktického využití těchto algoritmů jsem došel k několika variantám. Jednak je můžeme využít jako generátor náhrašky „ručně“ psaného textu, kdy by v určitých vhodných situacích suploval počítačové písmo. Lze si představit například elektronickou virtuální školní tabuli, na kterou přispívají jednotlivý uživatelé, ať už učitelé, nebo koneckonců i studenti.

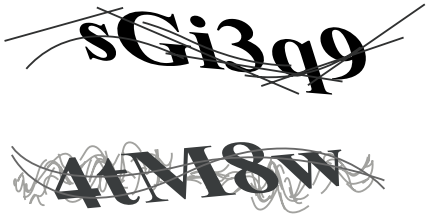
Druhak se dají využít při návrhu nových fontů, kdy si grafik může nechat vygenerovat sadu písma, a to jako například stovky ukázek jednoho různě transformovaného znaku, nebo více odlišných písmen, které budou opět různě tranformovány. Těmito výstupními sadami se poté může inspirovat u návrhu každého znaku, záleží pouze na něm, do jaké míry bude transformace užívat.

Další možnost využití nacházím na kterýchkoliv webových stránkách jako bezpečností prvek. Jistě se většina z nás již setkala s ověřovacími kódy (nazývanými zkráceně CAPTCHA) při odesílání všemožných online formulářů. Toto ověření je

odborně nazýváno Turingův test a na internetu se používá k automatizovanému odlišení skutečných uživatelů (lidí) od robotů (počítačů). Příklad takového obrázku je vidět na ilustraci 5.14.

CAPTCHA je akronym pro anglickou větu „completely automated public Turing test to tell computers and humans apart“, tedy „plně automatický veřejný Turingův test k odlišení počítačů a lidí“. Test spočívá zpravidla v zobrazení obrázku s deformovaným textem (několika znaky), přičemž úkolem uživatele je zobrazený text opsat do příslušného vstupního políčka. Předpokladem je, že mozek člověka dokáže lépe rozeznat deformovaný text, ale internetový robot při použití technologie OCR bude mít s rozpoznáním daleko větší problémy. K prolomení této základní ochrany však může dojít, pokud znaky nejsou dostatečně deformovány. S postupným rozšiřováním online registrací se také tyto robotické aplikace zdokonalují v rozeznávání generovaných obrázkových kódů.

Tento projekt by mohl pomoci vyvíjet lepší náhodné znaky, pro člověka čitelnější, které však pro robota budou obtížně rozpoznatelné.




Obr. 5.14: Ukázka moderních CAPTCHA obrázků.

Dále jsem zkusil experimentovat s jinými možnostmi oživení a přišel jsem s nápadem na kombinování fontu u jednotlivých znaků ve slově. Algoritmus by náhodně vybíral font a přiřazoval jej postupně ke znakům, jak jdou po sobě. Po zkoušce jsem však od další implementace upustil, jelikož mi výsledek nepřišel vhodný, ani zajímavý. Vnášel totiž do slova spíše zmatek a nebyl tak dostatečně užitečný.

# 6 ZÁVĚR

Základem této práce bylo prozkoumat techniky tvorby počítačových fontů, jejich úskalí a překážky, a poté ověřit možnosti generování fontu v jednom z programovacích jazyků dle vlastního výběru. Prostor byl také věnován nastínění problematiky generování náhodných efektů ve stylovém patkovém písmu.

Výběr fontů je v dnešním světě velmi široký a může tedy dobře sloužit k inspiraci při tvorbě nového fontu, potažmo generování náhodných písem díky informačním technologiím a proprietárnímu programovacímu jazyku. Jako inspirace při řešení tohoto projektu posloužily tučné fonty podobné znakové sadě Times New Roman, dále pak dobře známý a velmi rozšířený font Arial, psané písmo typu Comic Sans a v neposlední řadě také font Lindsey Pro, který v sobě uchovává více variant jednotlivých znaků.

V úvodu byl mimo jiné kladen důraz na typografické začátky, historii písma a ve větším měřítku byly také popsány způsoby konstrukce písma, jednotlivé typy nejčastěji používaných fontů a své místo zde našlo i tzv. vyrovnávání písma, které nazýváme také kerning. Poslední dvě podkapitoly v úvodní části se zaměřovaly na písmovou rodinu, řezy písma a také na různé druhy písem. Kapitola měla za úkol definovat mantinely pro tvorbu písma jako celku, což také splnila.

Další kapitola měla za úkol nastínit možnosti konstruování dnešních moderních fontů a seznámili jsme se s problematikou definice obrysových, neboli „outline“ fontů pomocí Bézierových křivek. Probrány byly používané techniky, vhodný software, ale také formáty uložení dat k popisu písmové sady (PostScript, TrueType a OpenType). Část textu je také věnována reprezentaci fontu v počítačích a to jak v dřívějších dobách, tak i dnes. Byly také nastíněny možnosti oživení struktury znaku (a s tím spojené problémy) a ukázány fonty, které přispěly k inspiraci. Poté jsme se seznámili s metodikou návrhu algoritmů zajišťujících náhodné chování jednotlivých znaků fontu, jejich úskalí a řešením těchto problémů.

V kapitole věnující se soustavě souřadnic jsem se zaměřil na kartézské a homogenní souřadnice. Tyto dva systémy totiž hrály klíčovou roli při řešení tohoto projektu. Geometrické transformace (a jejich následné skládání) zakomponované v algoritmu a prováděné na určitých znacích velmi usnadnily celkový proces docílení náhodného efektu.

Část pojednávající o dvourozměrné grafice nám přiblížila teoretický podklad pro tvorbu bodů, úseček a křivek, jejich následné rasteriazace a nastínila také způsoby napojování křivek — Bézierových kubik. Tyto křivky byly popsány jak v teoretické rovině, tak i na několika praktických ukázkách.

Poslední část této práce je věnována implementaci programu pro oživení znakové sady. Jako nejvhodnější software byl vybrán program Matlab především díky

nativní podpoře Bézierových křivek a snadnější manipulaci se znakem v souřadnicovém systému. Díky tomu není potřeba doprogramovávat tyto součásti samostatně, a lze se plně věnovat tomu, co je na této práci důležité — algoritmy pro docílení efektu náhodnosti.

Po stručném popisu programovacího prostředí Matlab je uveden postup grafické prezentace výsledků transformací, které je přehledně zpracováno do blokového schématu. K vidění jsou zde i ukázky zdrojových kódů aplikace a několik stránek textu se zabývá také uživatelským prostředím, způsoby nastavení parametrů a ovládáním transformací. Zmíněny jsou také možnosti dalšího rozšíření programu nebo návrhy pro reálné využití algoritmů.

Jak se ukázalo, algoritmy navržené pro oživení znakové sady můžou najít své uplatnění v praxi nebo také posloužit návrhářům nových fontů při jejich tvorbě. Dalším vylepšováním algoritmů by se navíc dalo docílit zlepšování generovaných výsledků. Doplnění jiných typů fontů a jejich celé abecedy znaků, které je ovšem časově náročné, by jistě přineslo velké vylepšení tohoto projektu.

# LITERATURA


[1] BERAN, V. a kol. *Aktualizovaný typografický manuál*. 4. vydání. Praha, Kafka design, 2005.

[2] BEDNÁR, P. *Tvorba písma OpenType volně dotupnými softwarovými protředky*. Brno, Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2011. 86 s. Vedoucí diplomové práce Mgr. Pavel Rajmic, Ph.D.

[3] BIĽAK, P. *Font hinting* [online]. 2010 [cit. 11. 4. 2012]. Dostupné z URL: <http://www.typotheque.com/articles/hinting>.

[4] CABARGA, L. *Logo, Font & Lettering Bible*. How Design Books, 2004, ISBN 1581804369.

[5] DEAN, P. *Type Terminology* [online]. 2008 [cit. 31. 3. 2012]. Part 1: The Detection of Types. Dostupné z URL: <http://ilovetypography.com/2008/03/21/extreme-type-terminology/>.

[6] DOŇAR, B., ZAPLATÍLEK K. *MATLAB pro začátečníky*. Praha, BEN - technická literatura, vydání 2. aktualizované, 2005, ISBN 80-7300-175-6.

[7] DUSONG, J. L., SIEGWARTOVÁ, F. *Typografie – od olova k počítačům*. Praha, Svojtka a Vašut, 1997.

[8] ENGELEN, J. B. C. *How to include an SVG image in LaTeX*. MESA+ and IMPACT Research Institutes, TST group, University of Twente, Enschede, 2010.

[9] *MATLAB Getting Started Guide*. The MathWorks, Inc., 2011, vydání 17. aktualizované.

[10] MATTESON, S. *Lindsey Pro Type Specimen* [online]. 2009 [cit. 1. 12. 2011]. Dostupné z URL: <http://www.ascenderfonts.com/af/pdf/Lindsey-Pro-Type-Specimen.pdf>.

[11] PÍŠA, P. Matlab — laboratoř nejen pro matematiky. *Sdělovací technika*, 1998, per. 46, č. 6, s. 9–19. ISSN 0036-9942.

[12] PECINA, M. *Typomil — písmo a typografie* [online]. 2004 [cit. 12. 11. 2011]. Dostupné z URL: <http://www.typomil.com>.

[13] PRAUTZSCH, H. et al. *Bezier and B-Spline Techniques*. Springer, 2010, ISBN 3642078427.

[14] RAJMIC, P., SCHIMMEL, J., TRZOS, M. *Multimediální a grafické procesory*. Skriptum. Brno, Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2011.

[15] RAMBOUSEK, J. *Písmo a jeho užití*. Praha, Orbis.

[16] SPIEKERMANN, E. *Font Hinting Explained by A Font Design Master* [online]. 2010 [cit. 15. 4. 2012]. Dostupné z URL: <http://readableweb.com/font-hinting-explained-by-a-font-design-master/>.

[17] ZEMAN, V., RAJMIC, P., SYSEL, P. *Věcné a typografické pokyny a zásady pro psaní studentských prací*, ÚTKO FEKT VUT, Brno, 2012.

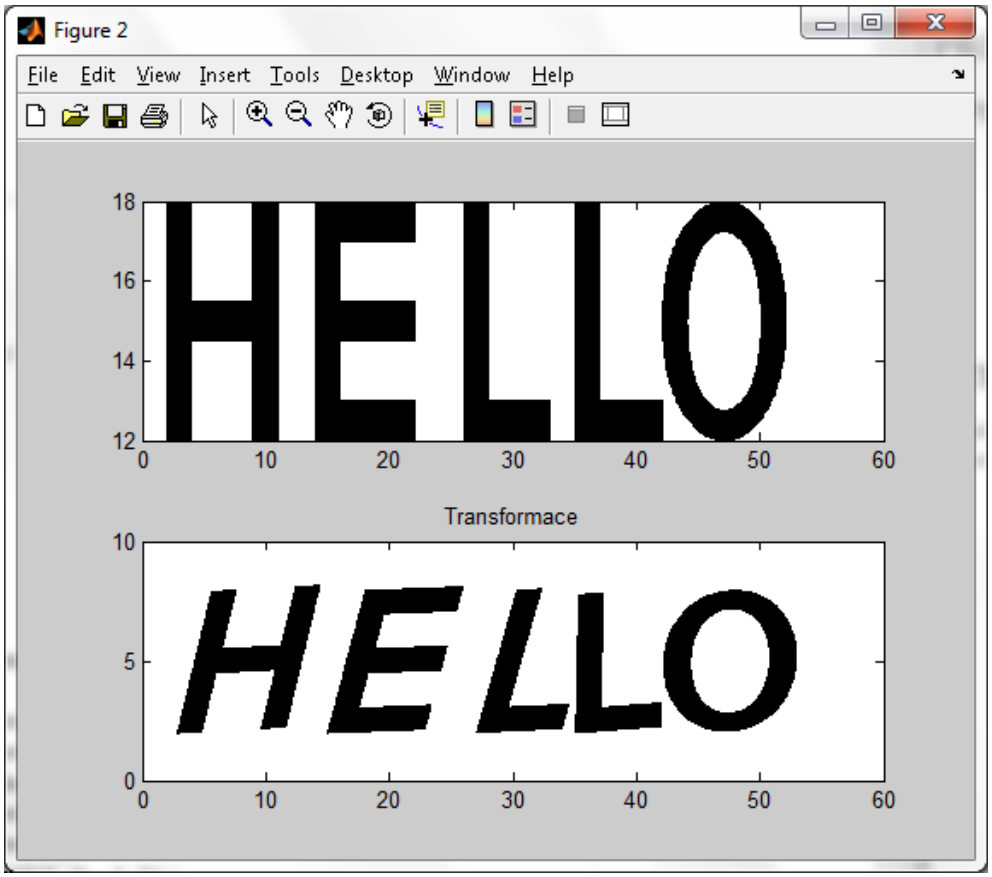
[18] ŽARA, J., BENEŠ, B., SOCHOR, J., FELKEL, P. *Moderní počítačová grafika*. 2. vydání. Brno, Computer press 2004, 609 s., ISBN 80-251-0454-0.

# SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

| | |
|---|---|
| CAD | Computer-aided design — počítačem podporované projektování |
| CAPTCHA | Completely Automated Public Turing test to tell Computers and Humans Apart — plně automatický veřejný Turingův test k odlišení počítačů od lidí |
| DTP | Desktop publishing — tvorba tištěného dokumentu za pomoci počítače |
| GNU | GNU není Unix — GNU's Not Unix |
| JPEG | Joint Photographic Experts Group — velmi často používaný formát pro ukládání obrázků |
| OCR | Optical Character Recognition — optické rozpoznávání znaků |
| OS | operační systém — Operating System |
| Mac OS X | operační systém firmy Macintosh ve verzi X |
| PDF | Přenosný formát dokumentů – Portable Document Format |
| ROM | Read Only Memory — Paměť pouze pro čtení |

# A PŘÍLOHY

## A.1 Ukázka výstupní obrazovky programu Matlab




Obr. A.1: Výstupní obrazovka dvou grafů.

## A.2 CD médium

Přenosné médium obsahující kromě samotné diplomové práce v elektronické podobě také zdrojové kódy aplikace a algoritmů.