Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

ZVI

Skeletizace

Plzeň, 2006 Martin Chlupáč

1.

Zadání

Cílem této práce bylo implementovat skeletizační algoritmus popsaný V. K. Govindanem a A. P. Shivaprasadem [1] a ukázat jeho klady a zápory. Vycházel jsem pouze z odkazovaného článku, který jsem ovšem musel, jak bude dále uvedeno, několikrát doplnit vlastním postupem, neboť některé podstatné údaje pro korektní implementaci v něm nejsou uvedeny.

2.

Popis algoritmu

Algoritmus sestává ze dvou částí; první je prostá skeletizace, kdy o odstranění jednotlivého pixelu z obrázku rozhoduje pouze výsledek 4 testů týkajících se jeho nejbližšího okolí (pokud již nebyl dříve označen jako součást kostry); druhou částí je vylepšení dosahovaných výsledků, pomocí aktivního vyhledávání některých klíčových pixelů, které jsou následně zpracovávány odlišným způsobem.

2.1 Sekvenční ztenčování

Před stručným popisem vlastního ztenčování je nejprve třeba definovat několik pojmů.

Definice pojmů

- *sousední pixel* – je pixel, který se zvoleným pixelem sdílí hranu (*d-neighbour*, přímý soused), nebo s ním sousedí alespoň přes roh (*i-neighbour*, nepřímý soused). Úplné okolí zahrnuje všechny přímé i nepřímé sousední pixely daného bodu.
- *d-path*, *i-path* – cesta složená z přímo, resp. i nepřímo, sousedících pixelů
- *Crossing number* (*CSN*) – počet přechodů z jedné barvy do druhé (např. z černé do bílé), které se vyskytují na d-path v úplném okolí zvoleného bodu
- *Black neighbours* (*BP*) – počet černých pixelů (0-8) v úplném okolí bodu

Ztenčovací algoritmus

Budu zde popisovat mnou upravenou verzi původního algoritmu, neboť v originálním článku nejsou popsány některé významné detaily a v případě potřeby implementovat algoritmus v jeho původní podobě by bylo nutno dělat řadu časově náročných testů, aby byly tyto detaily vyjasněny.

Nejdříve jsou v obrázku nalezeny hraniční pixely, tedy takové černé pixely, pro které platí, že některý z jejich sousedů (přímých, nepřímých, obou) je bílý. Tyto jsou označeny v daném kroku skeletizace za pixely hraniční a pouze jich se týkají všechny další operace.

Dále jsou jednotlivé hraniční pixely testovány, zda splňují následující podmínky a dle výsledků jsou buď odstraněny, označeny za součást kostry nebo ponechány k pozdějšímu zpracování v dalších krocích.

Hraniční pixel je z obrázku odstraněn pokud následující podmínky jsou splněny:

1. CSN daného bodu je rovno 1
2. BP > 1
3. BP < 7
4. pixel nebyl již dříve označen za součást skeletonu

Pixel je označen za součást kostry pokud alespoň jedna z prvních dvou podmínek není splněna.

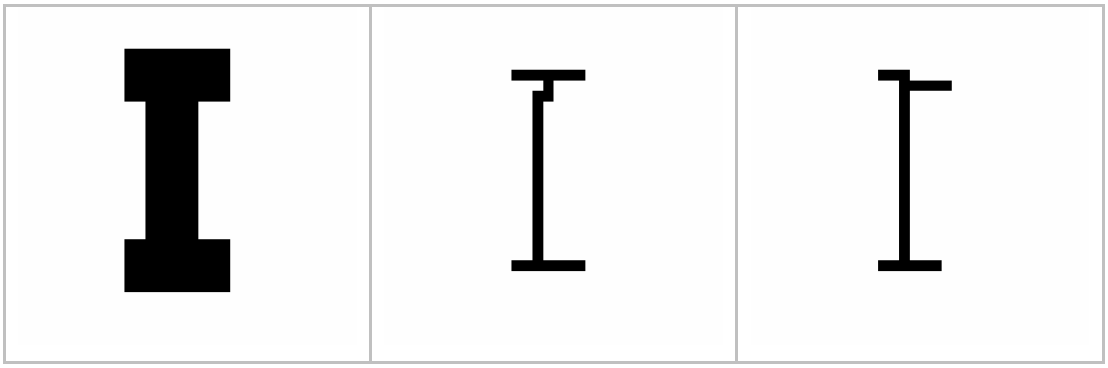
V ostatních případech zůstává pixel nezměněn.

První z podmínek zaručuje, že výsledná kostra bude tvořit d-path. V případě požadavku na i-path by pak bylo třeba ji modifikovat. Odkazy na důkaz lze nalézt v původním článku [1].

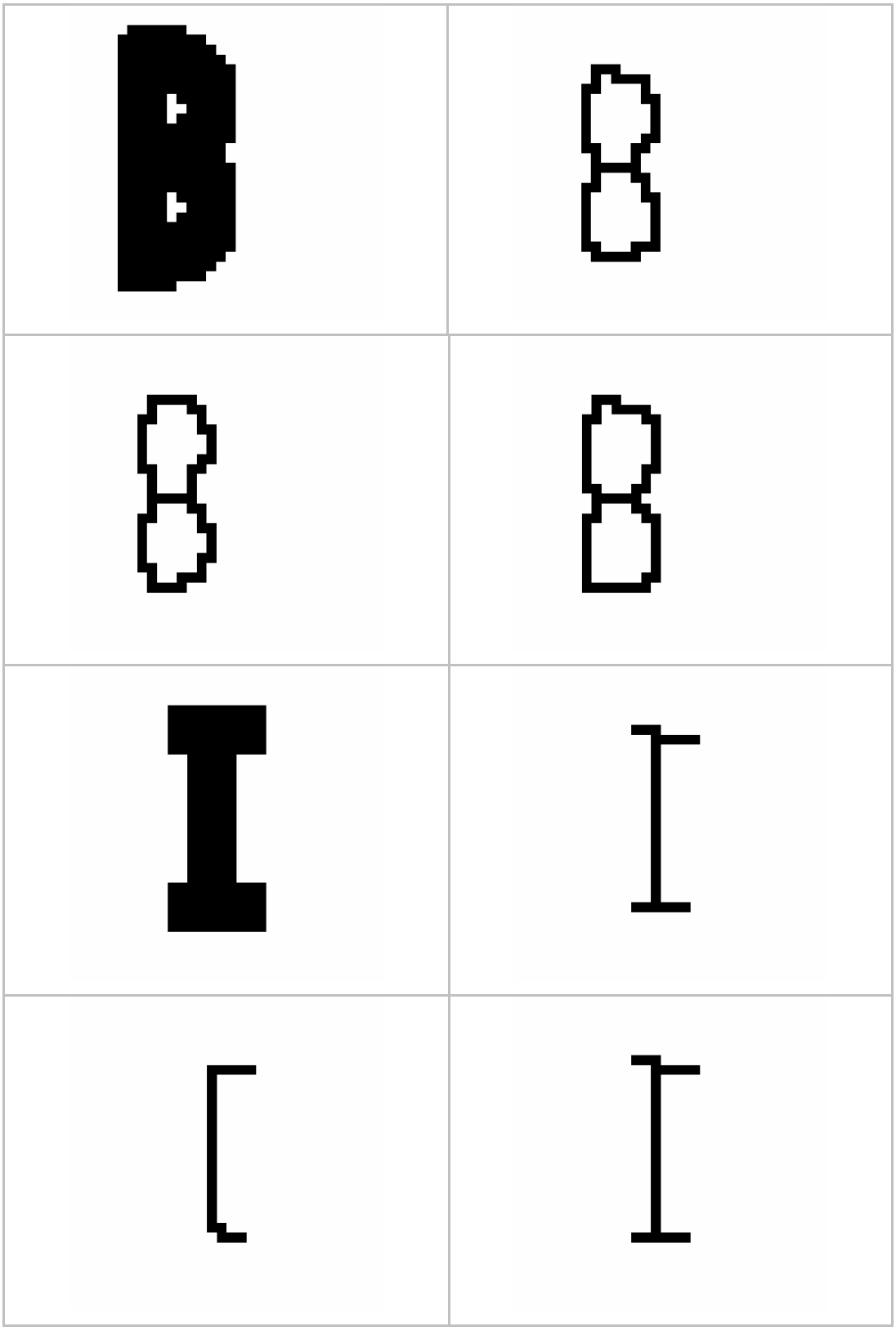
Výsledky

Obecně pro většinu skeletizačních algoritmu platí, že jejich výsledky více či méně závisejí na pořadí, ve kterém jsou pixely v obrázku zpracovávány. Autoři tohoto algoritmu v článku nespecifikují jaký postup zvolili, resp. uvádějí: *„The sensitivity of the skeleton to direction and starting point of peeling is avoided to a greater extent by incorporating appropriate simple logic in the tracing algorithm.“*

Já jsem nevyužil žádnou z pokročilejších metod, ale provedl jsem jednoduchý pokus, ve kterém jsem postupně vyzkoušel všechny čtyři základní směry a zvolil jsem ten, který dával subjektivně nejlepší výsledky – odshora dolů a zleva doprava.

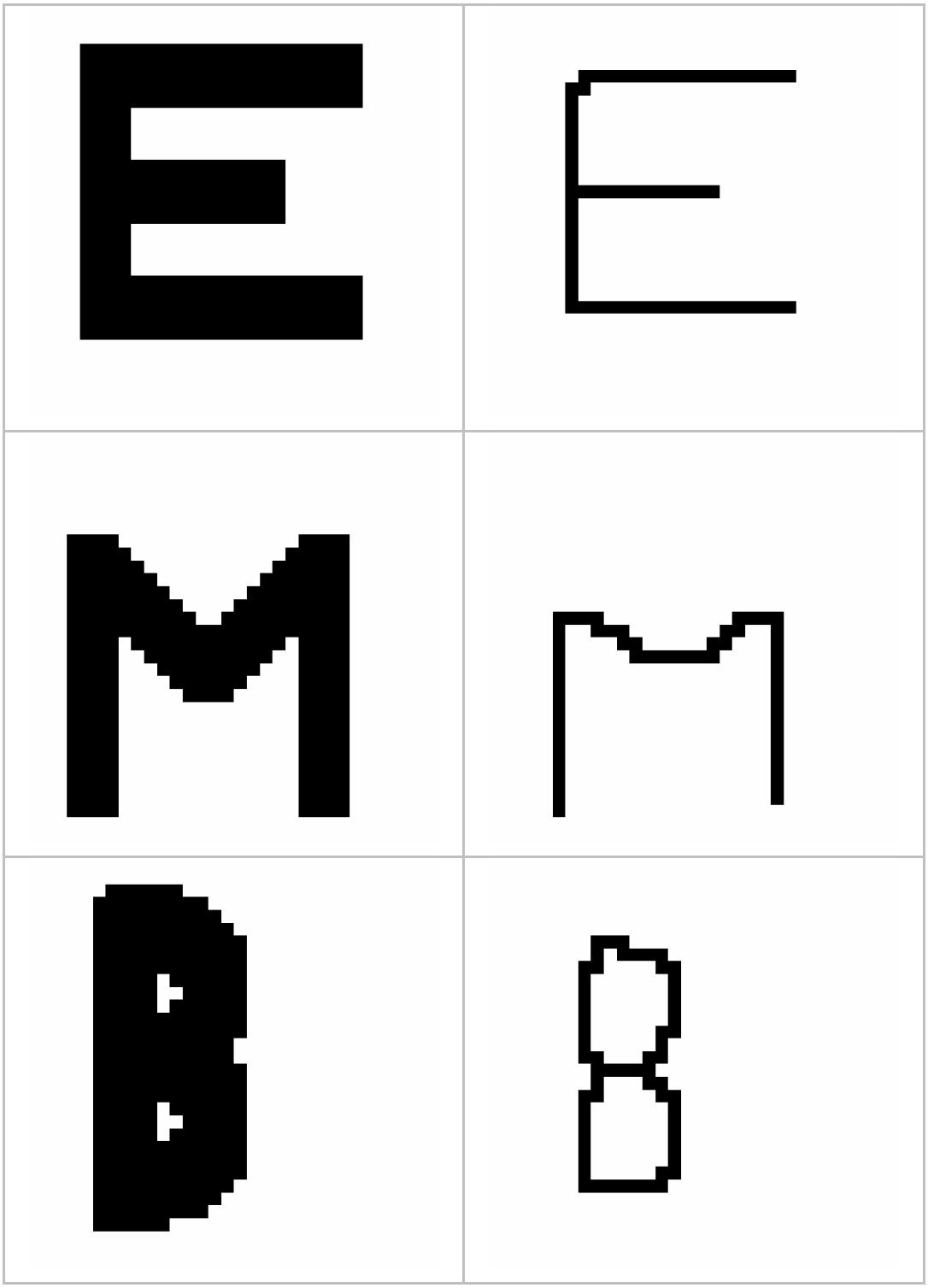


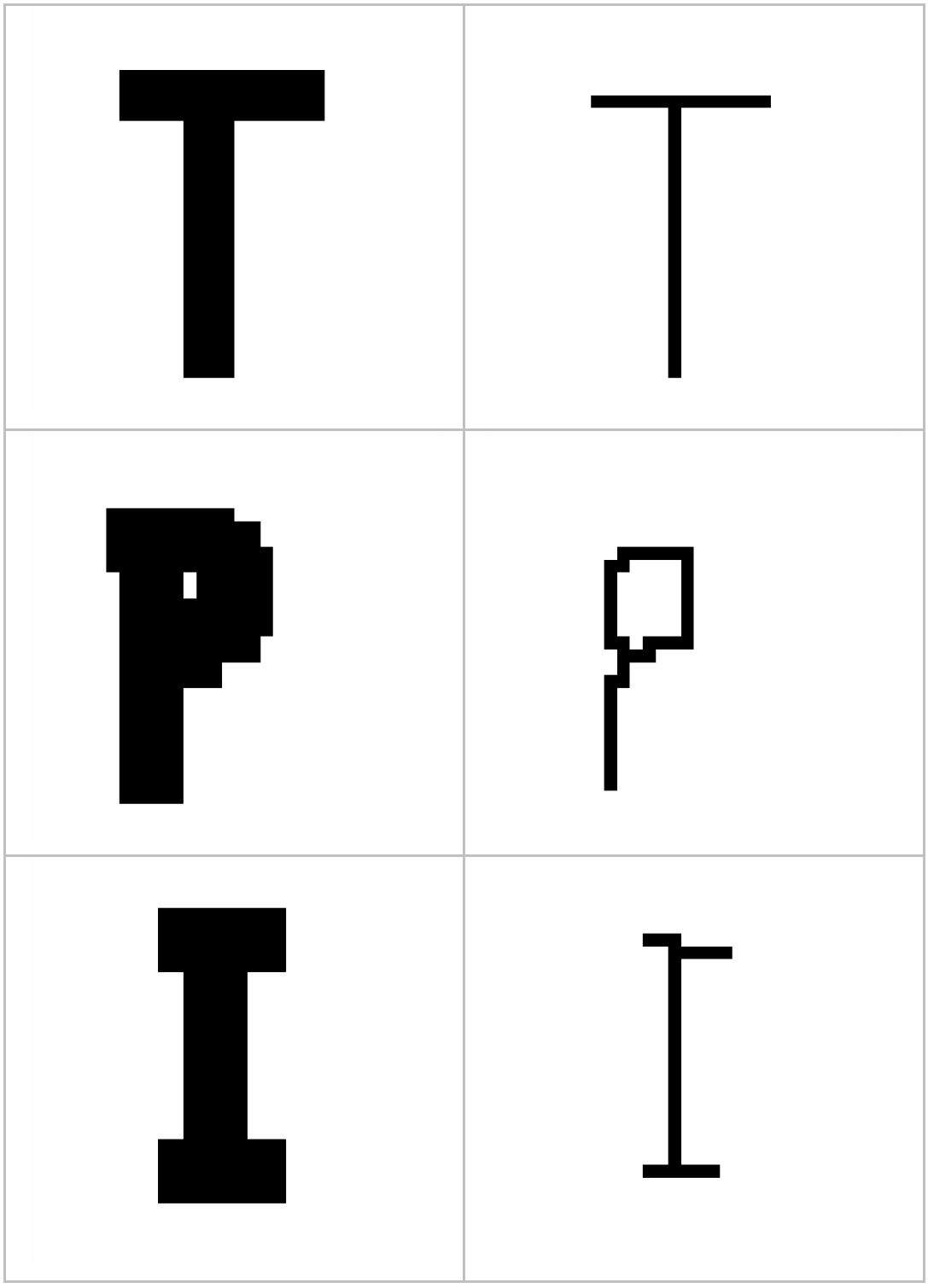
Obrázek 2.1: Srovnání originálu a skeletonů vzniklých při zpracování ve směru „po sloupcích“ resp. „po řádcích“. Z obrázku je zjevná závislost výsledku za směru zpracování



Obrázek 2.2: Srovnání originálu a koster vzniklých při hledání obrysu pomocí i-neighbour, d-neighbour, resp. obou současně

V dalších ukázkách bude použito hledání obrysu pomocí úplného okolí bodu. Pixely jsou zpracovávány odshora dolů po jednotlivých řádcích .





Obrázek 2.3: Ukázky výsledků skeletizace

2.2 Adaptivní varianta algoritmu

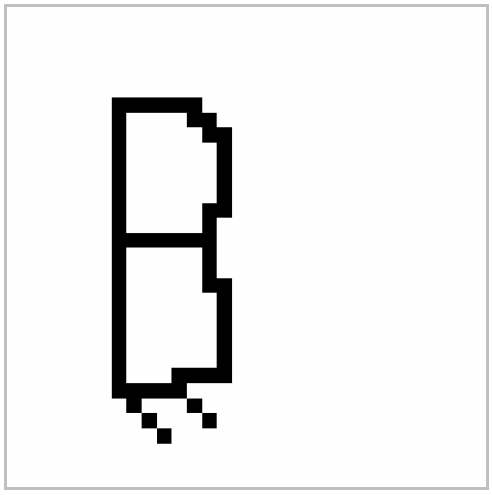
Z uvedených ukázek je zřejmé, že algoritmus trpí některými neduhy, které vedou k deformacím výsledného tvaru vytvořené kostry. Při bližším zkoumání vyjde najevo, že problematickými jsou ta místa, ve kterých dochází ke spojování jednotlivých čar tvořících kostru; bylo by tedy vhodné takovým oblastem věnovat zvýšenou pozornost. Především jde o situaci, kdy u ostrých vnitřních úhlů nedochází ke včasnému odstraňování bodů, které se nacházejí v jeho vrcholu.

U původní verze algoritmu se navíc objevovaly defekty, podobné tomu jaký je na předchozím obrázku u písmene E, ve všech pravoúhlých spojeních čar. Tyto deformace se mi podařilo ve značné míře potlačit vhodně zvoleným směrem, ve kterém algoritmus prochází body v obrázku. Negativním dopadem v tomto případě je asymetrie, již lze pozorovat u písmene M.

Původní algoritmus tedy jako zvláštní případy řešil nejen spojení čar v ostrém úhlu, ale i pravoúhlá spojení typu T a L. Já jsem ve své verzi použil jen jednu úpravu, která pomáhá potlačit oba druhy defektů současně. Tento postup jsem byl nucen zvolit, neboť v původním článku nejsou některé detaily zpracování pravoúhlých spojů upřesněny.

V prvním kroku proběhne odstranění jednotlivých hraničních bodů dle dříve popsaného postupu. Nově se k němu však přidává krok druhý, kdy jsou odstraněny i body splňující kriterium **BP == 7**.

Kdybychom tuto podmínku přidali přímo k předchozím pravidlům, došlo by k výraznému poškození výsledné kostry diagonálně umístěnými artefakty.



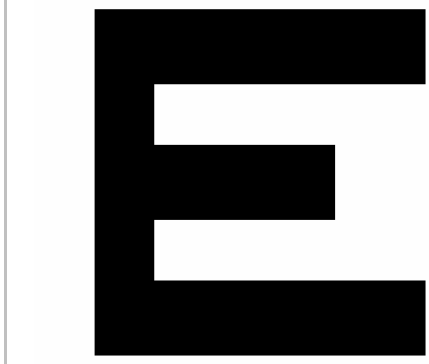
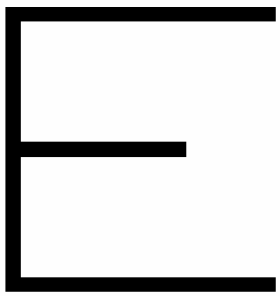
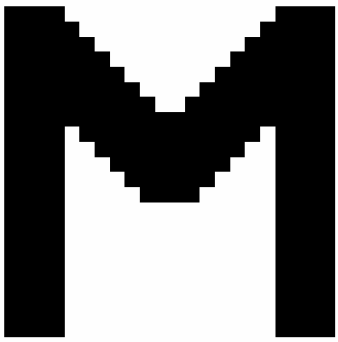
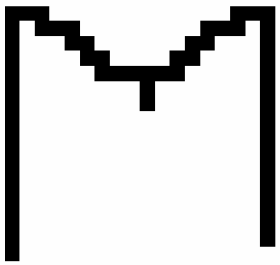
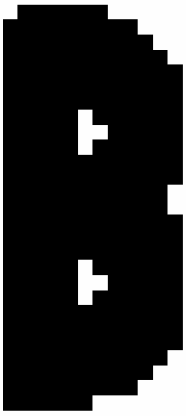
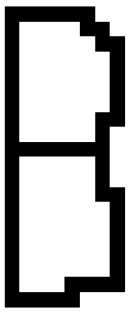
Obrázek 2.4: Kostra písmene B s viditelnými artefakty po přidání dodatečného pravidla k původnímu setu pravidel

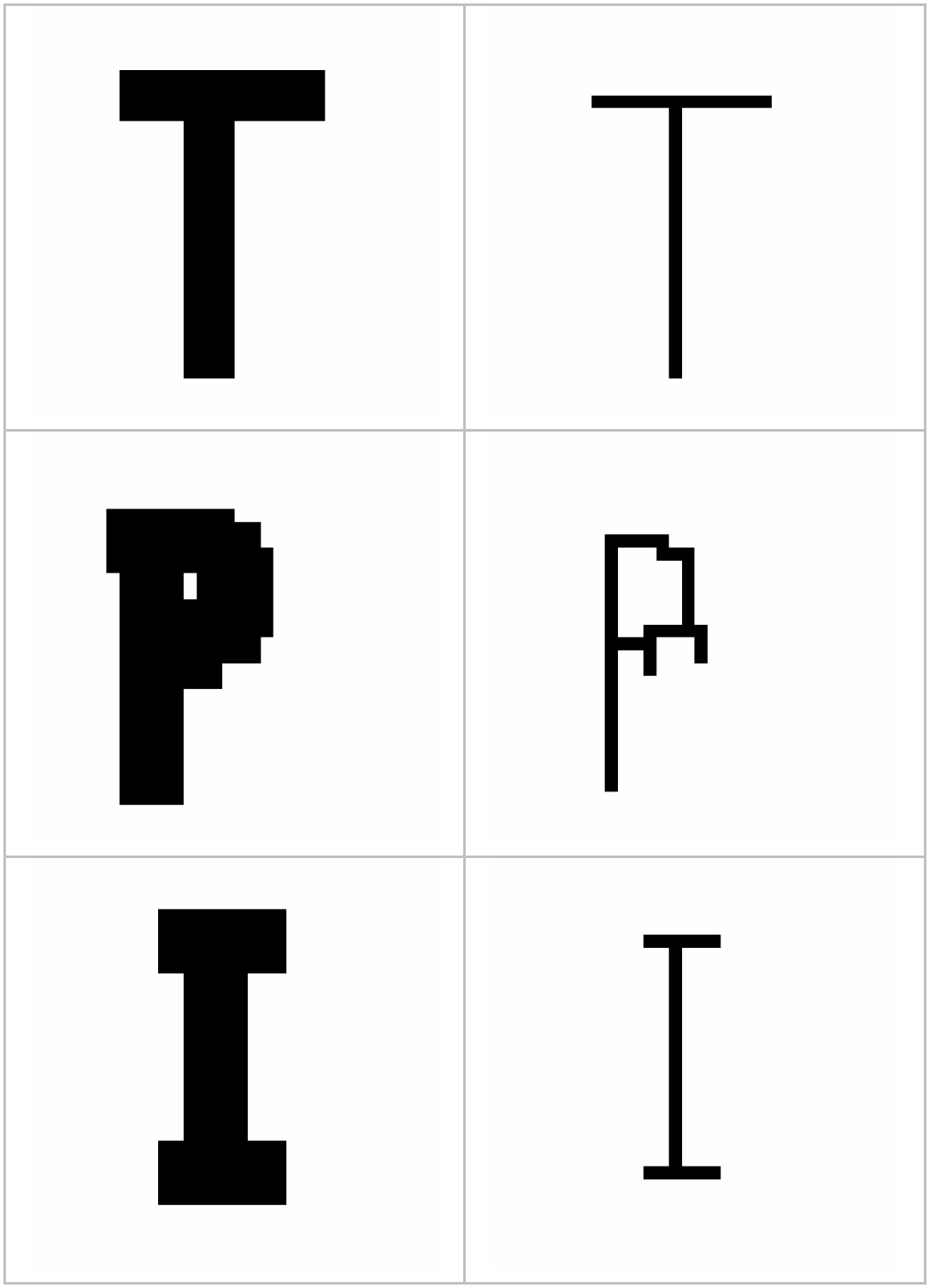
Kromě přidaných artefaktů je ovšem zřejmé, že ke zlepšení došlo. Postupoval jsem tedy tak, že nejprve byla na obrázek aplikována původní množina pravidel, poté na kopii obrázku byla uplatněna množina pravidel rozšířená o nové omezení a výsledné obrázky byly zkombinovány přes logický součet.

Tento postup je blízký způsobu, který je popsán v původním článku. Autoři algoritmu použili pro odstranění artefaktů heuristiky, kdy kladou podmínku na

minimální délku čáry, která je jinak odstraněna jako artefakt (podrobnosti však nejsou uvedeny).

Výsledky



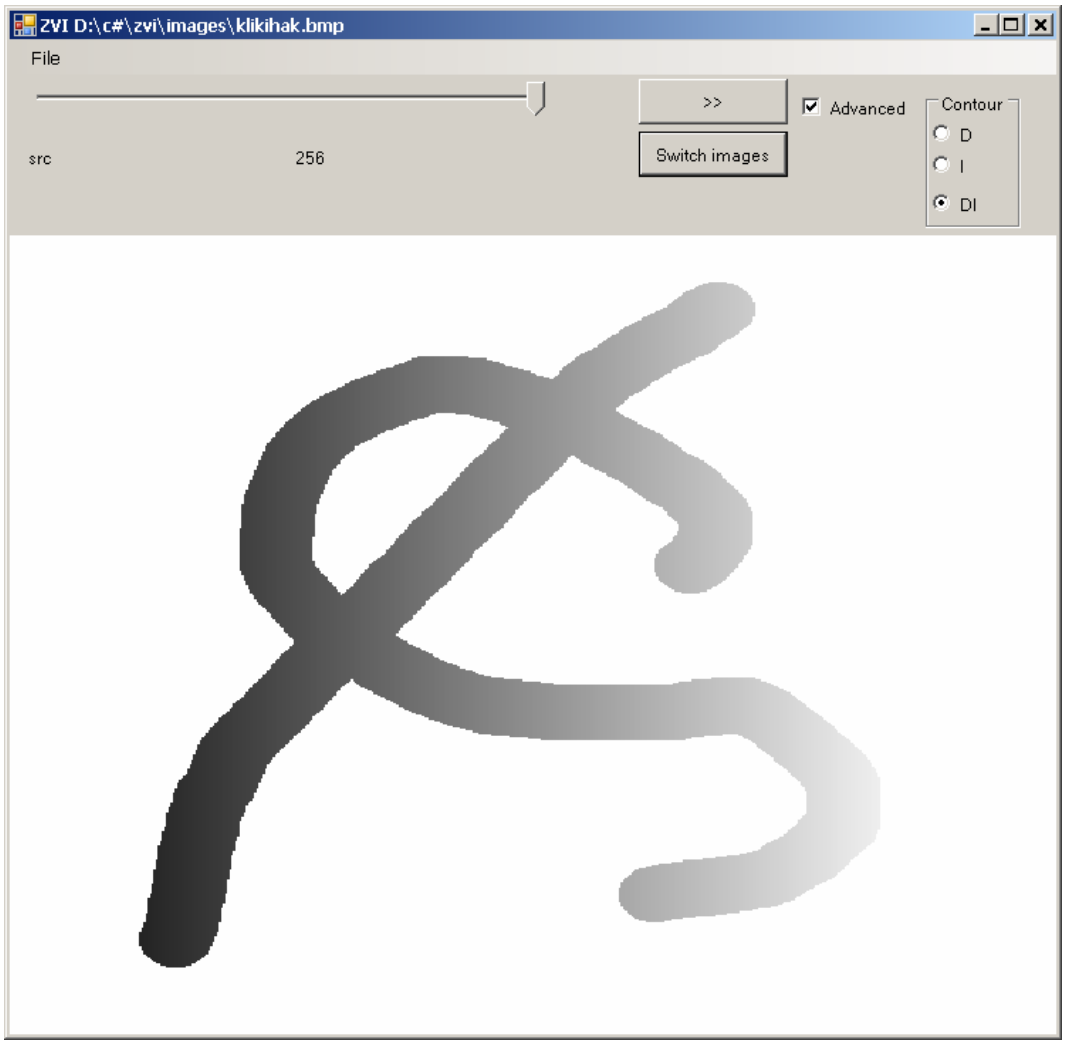
Obrázek 2.5: Ukázky výsledků skeletizace

Jak Obrázek 2.5 demonstruje, ve všech případech došlo ke zlepšení výsledků skeletizace, pouze u písmen P a M přibyly artefakty ve svislém směru. Jejich vzniku původní algoritmus zabraňoval pomocí výše zmíněné heuristiky.

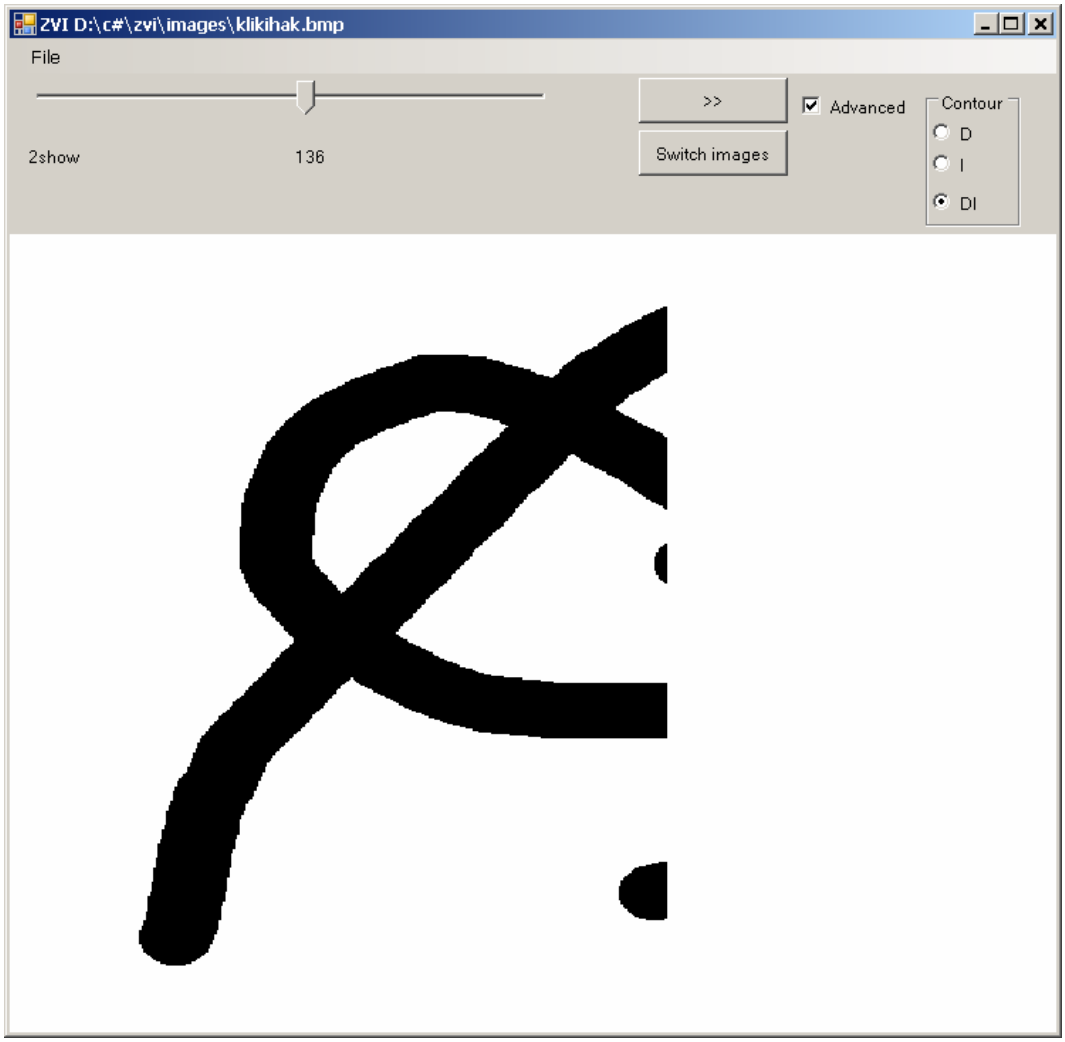
3.

Implementace

Celý algoritmus jsem implementoval v prostředí .NET a jazyce C#. Výsledná aplikace pracuje s obrázky v *True Color* a při načítání je převádí do šedé škály.



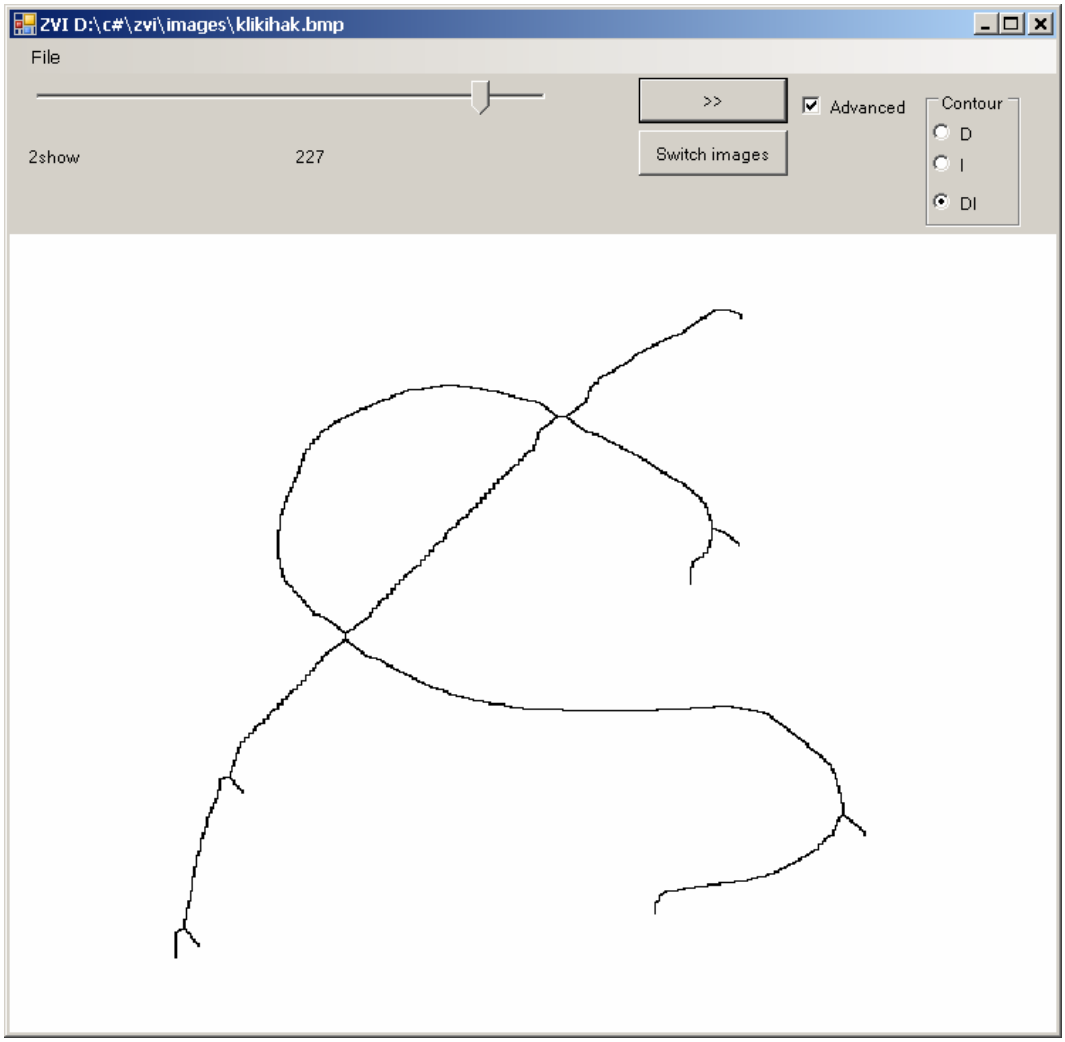
Obrázek 3.1: Základní okno aplikace s načteným obrázkem po převodu do šedých tónů



Obrázek 3.2: Posuvníkem v horní části obrazovky lze obrázek naprahovat

Dále lze zvolit zda mají být hrany obrázku (*Contour*) hledány pomocí d-neighbours, i-neighbours nebo jejich kombinace a zda má být použita základní verze algoritmu nebo jeho verze hledající kritická místa (*Advanced*).

Postupným klikáním na tlačítko „>>“ jsou prováděny jednotlivé kroky skeletizace. V každém okamžiku lze přepnout (*Switch images*) na původní verzi obrázku, aby ji bylo možno porovnat s aktuálním výsledkem skeletizace.



Obrázek 3.3: Výsledek skeletizace

Kdykoli je též možno uložit aktuální obrázek do souboru formátu PNG (viz menu *File*)

Poznámka k implementaci

Prostředí .NET se ukázalo být velmi nevhodným pro manipulaci s obrázky, neboť pro práci s nimi nabízí metody zcela neoptimalizované vůči rychlosti zpracování. Důsledkem je, že i prostý převod obrázku do šedé škály trvá velmi dlouho a pro vlastní skeletizaci jsem byl nucen použít náhradní datové struktury (*bool[,]*), neboť práce přímo s hodnotami jednotlivých pixelů obrázku byla neúměrně pomalá.

Tuto pomalost lze řešit přepnutím do neřízeného (*unmanaged*) kódu a pracovat pak s hodnotami přes pointery; to však naprosto popírá celou filosofii .NETu a pro školní demonstraci algoritmu není otázka rychlosti natolik kritická, aby se bylo třeba uchylovat k takto „radikálním“ řešením.

4.

Závěr

Původní algoritmus popsaný v [1] se mi nepodařilo implementovat z důvodu jeho nedostatečného popisu, kdy některé důležité informace nebyly v článku vůbec uvedeny či byly pouze naznačeny. Pro jeho přesnou implementaci by tak bylo třeba dalších pokusů, aby byly jednotlivé detaily upřesněny.

Úspěšně jsem však implementoval variantu algoritmu, která je velmi blízká té původní a i přes svoji větší jednoduchost dává velmi podobné výsledky. Moji verzi algoritmu by bylo možno dále vylepšit, pokud by byla přidána heuristika z algoritmu původního, která pomáhá odstraňovat artefakty, které nyní na výsledné kostře zůstávají.

Literatura

[1] GOVINDAN, V. K., SHIVAPRASAD, A. P. A Pattern Adaptive Thinning Algorithm. *Pattern Recognition*. 1987, vol. 20, no. 6, s. 623-637.

[2] *Picturebox Smoothing* [online]. 2005 [cit. 2006-05-25]. Dostupný z WWW: <http://channel9.msdn.com/ShowPost.aspx?PostID=44031>.