Univezita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE

Jan Lána

Digitalizace mapy

duben 2001

Kabinet software a výuky informatiky

Vedoucí diplomové práce: RNDr. Josef Pelikán

Studijní program: Informatika

Prohlašuji, že jsem svou diplomovou práci psal samostatně a váhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním diplomové práce.

Jan Lána

Abstrakt

Cílem několika diplomových prací zadaných na MFF UK je vytvořit aplikaci, která by z mapy pro orientační běh v papírové podobě vytvořila digitální model terénu a umožnila jeho procházení. Cílem této práce je z zdigitalizvané a na jednotlivé tiskové barvy rozložené mapě vytvořit trojrozměrný model terénu, čímž přímo navazuje na diplomovou práci pana Čejky [2], obhájenou v roce 1999.

Většina map vzniká dnes „na počítači“, takže se nabízí otázka, zda není efektivnější zabývat se přímo výstupy těchto programů. Tato varianta ovšem naráží na dva zásadní problémy — digitální podoba mapy narozdíl od papírové nebývá dostupná a specifikace formátu souborů programů používaných pro tvorbu map není volně k dispozici. Kromě toho mnoho starších map existuje jen v papírové podobě. V neposlední řadě je tato práce přínosem k obecnému problému převodu papírových dokumentů do digitální podoby.

Při práci s papírovou podobou mapy se ale musím vypořádat s problémem ztráty informace. K té dochází jak při tisku mapy (převod vektorového obrazu do rastrového, přetisky, ...), tak při zpětné digitalizaci (nové rastrování a kvantování, chyby vstupních zařízení, ...). Navíc přiřazení grafických symbolů k jednotlivým značkám není vzájemě jednoznačné. S tím vším si musí, pokud možno automaticky, výsledná aplikace poradit.

Mým úkolem bylo najít a implementovat algoritmy, které rozpoznají jednotlivé topografické značky a na jejich základě vytvoří digitální model terénu. Protože se automatické rozpoznávání tak složitého obrazu, jakým mapy jsou, asi neobejde bez chyb, mělo by být možné provést „ruční“ opravy. Vytvoření editoru, který by provádění těchto oprav umožnil, nebylo cílem ani součástí zadání této práce.

Zaměřil jsem se zejména na rozpoznávání mapových značek a zvláštní pozornost jsem věnoval značkám liniovým. Pochopení jednotlivých značek je základní a nutný krok k vytvoření modelu terénu, neboť prakticky všechny informace v mapě zaznamenané jsou zaznamenané pomocí značek. Liniové značky jsou pak prakticky jediným zdrojem informací o výškových poměrech v terénu.

Digitalizaci map je v poslední době věnována značná pozornost, většinou se ale jedná o projekty nového vytváření map v digitální podobě pomocí ručního přenosu informací z papírových map případně i nové mapování krajiny [1]. O automatickém převodu map se mi nepodařilo najít žádnou literaturu, proto je většina zde uváděných algoritmů mnou vytvořená.

Obsah

Seznam tabulek

Seznam obrázků

1 Úvod

Již delší dobu jsem se zabýval myšlenkou rozpoznat jednotlivé značky v mapě, i když moje motivace byla trošku jiná než cíl této práce — chtěl jsem efektivně a kvalitně uložit větší množství map a poskládat z nich mapu větší, mapu celé republiky. Ve svých úvahách jsem se ale nedostal dále než k řešení otázky rozkladu mapy na jednotlivé barvy (což samo o sobě vede ke značné úspoře místa), proto mě velmi zaujalo téma práce této.

Poněkud nepřesně můžeme *topografickou mapu* (dále jen *mapu*) definovat jako dvourozměrný obraz, který je komponován ze značek plošných, liniových a bodových, jež jsou umísťovány do mapy v závislosti na poloze reálných objektů v krajině. Tento obraz je oproti „originálu“ (krajině) zmenšen v určitém poměru — *měřítku*. Protože pro různé obory lidské činnosti jsou zapotřebí různé informace, existuje mnoho druhů map, které se vzájemě liší měřítkem a množinou použitých značek s pravidly pro jejich rozmísťování.

Tato práce se zabývá mapami pro orientační běh. Je druhým „svazkem“ díla, kteréžto můj předchůdce, autor prvního dílu [2] nazval *Simulátor orientačního běhu*. Celá aplikace by měla z papírové mapy pro orientační běh vytvořit *digitální model terénu*, tj. 3D model krajiny spolu s reálnými objekty v krajině se nacházejícími a v mapě zakreslenými, a umožnit jeho procházení.

Práce, na kterou navazuji, se zabývala převedením papírového vstupu do podoby 2D bitmapového obrazu rozloženého na jednotlivé tiskové barvy a detekcí pravidelných rastrů (více kap. 2). Mým úkolem bylo rozpoznat jednotlivé značky v takto předzpracovaném obrazu, porozumět jim, a na jejich základě vytvořit digitální model terénu. Tento úkol jsem rozdělil na tři části — rozpoznat jednotlivé značky zakreslené v mapě, na jejich základě doplnit třetí (výškovou) souřadnici a poté vytvořit triangulovanou podobu krajiny, viz. obrázek 1.1 (zadáním této práce je vyřešit podstrom „2D→3D“). Každý z jednotlivých úkolů je relativně samostatný a vyžaduje algoritmy z trošku jiné oblasti.

Bohužel jsem již v počáteční fázi řešení úkolu zjistil, že přestože se zdá, že problému automatické digitalizace map je v poslední době věnována pozornost, nepodařilo se mi najít publikované algoritmy řešící tento problém. Většina textů, které jsem nalezl se zabývá otázkou přenositelného formátu uložení digitalizovaných map a prací s těmito údaji.[1] Bylo zřejmé, že se mi nepodaří vyřešit celou problematiku převodu na dostatečné úrovni — jedním řešením bylo pokusit se v hrubých rysech vyřešit všechny části převodu, druhým pak postupovat důkladně a vyřešit jen část. Rozhodl jsem se pro variantu druhou. Domnívám se, že poskládat celou aplikaci z dotažených částí je jednoduší, než přepracovávat větší množství částí hotových jen částečně

Úkol, na který jsem se soustředil, je rozpoznávání liniových značek. Správné rozpoznání liniových značek je nezbytný předpoklad pro doplnění třetí, výškové, souřadnice v terénu, neboť právě liniové značky jsou prakticky jediným zdrojem informace o výškových poměrech. Rozpoznávání liniových značek lze zařadit do problému nazývaným *vektorizace*, který se zabývá otázkou převodu rastrového obrazu do vektorové podoby (patří sem např. převod vytištěných technických výkresů zpět do vektorové podoby [8]). Rozpoznávání mapových značek musí navíc vyřešit problém kolizí — protože většina značek je nadměrné velikosti vzhledem ke zmenšení daném měřítkem, dochází ke kolizím. Ty jsou v průběhu procesu tvorby mapy řešeny buď prostým překrytím kolidujících značek nebo přerušením značek s menší důležitostí či značek rozměrnějších, u kterých lze jejich tvar v místě kolize odhadnout na základě zbytku značky.[2]

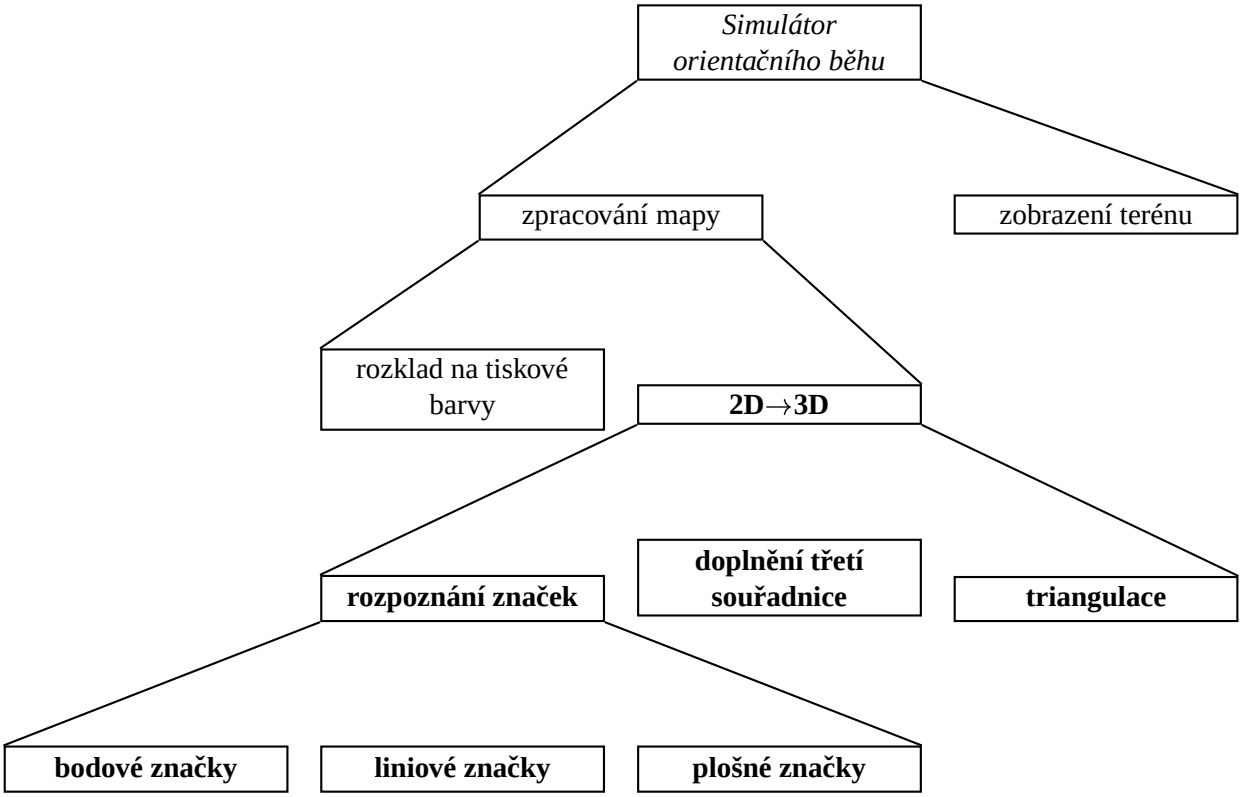
K této nesnázi se přidává další v podobě *přetisků* — mapy jsou tisknuty několika základními barvami a v místech, kde je tisknuto několika z nich zároveň, dochází k přetisku a vzniku barvy nové. Pro účely rozpoznávání značek je třeba znát, které barvy se smísily. Přestože tento problém je schopen program pana Čejky [2] na místech, kde je to alespoň trošku možné, vyřešit, zůstávají místa, kde to možné není — například přetiskem jakékoli barvy s barvou černou vznikne zase barva černá. V místech křížení značek různých barev tedy vznikají další přerušení.

Pro další zpracování je nezbytné rozpoznat značky jako celek, tj. oddělit spojené značky a sloučit oddělené části patřící k jedné značce dohromady.

V další kapitole (kap. 2) se pokusím shrnout práci [2] zabývající se rozkladem na tiskové barvy na kterou na-

[1] zvláštní je, že vytištěná podoba mapy se mezi návrhy neobjevovala ☺

[2] toto je poněkud zjednodušený pohled na řešení kolizí, kartografie, jakožto věda o vytváření map, používá další techniky, jako je posouvání značek, záměna několika značek za jinou, apod. (více např. v [3, 4]). Problém řešení kolizí zaměstnává lidi a počítače i na této „straně barikády“. Je například důvodem, proč doposud není možné zcela automaticky generovat z jednoho souboru mapy různých měřítek ([1]).



Obrázek 1.1: Struktura *Simulátoru orientačního běhu*

vazuji. Kapitola 3 obsahuje obecné úvahy nad úkoly doplnění třetí souřadnice a triangulací. O rozpoznávání značek obecně je kapitola 4 a kapitola 5 se pak zabývá speciálně značkami liniovými. Popis knihovny, která je implementací popsaných algoritmů a jejich výsledky se zabývá kapitola 7.2.

2 Rozklad na tiskové barvy

Tato kapitola se pokusí shrnout práci mého předchůdce, pana Čejky, jejímž výsledkem je diplomová práce *Optické rozpoznávání objektů na mapách určených pro orientační běh* [2]. Z této práce jsem čerpal základní informace o problematice a výstup aplikace, která je její součástí, je mým vstupem. Proto považuji za vhodné zde v několika odstavcích popsat, co se s mapou stane dříve, než se ke mě dostane.

První část textu diplomové práce pana Čejky se zabývá mapami obecně, popisuje jednotlivé skupiny mapových značek a zmiňuje hlavní problémy, které bude při rozpoznávání těchto značek nutné vyřešit. Také přesně specifikuje výstup celé fáze zpracování mapy, tj. formát souboru pro uložení trinagulovaného terénu spolu s dalšími informacemi o krajině.

V další části se věnuje otázkám zpracování digitalizované mapy a rozkladu na tiskové barvy. K digitalizaci mapy je použito stolního scanneru. Zdigitalizovanou mapu je tedy nejprve nutno předzpracovat — odstranit degradace způsobené digitalizací a vhodně upravit obraz pro další zpracování. Odstraňovány jsou degradace vzniklé impulzní odezvou scanneru a vzájemný posun RGB barev sejmutého barevného obrazu. Odstranění degradací vzniklých impulzní odezvou vede k podstatnému zpřesnění detailů, odstranění vzájemného posunu zpřesňuje následnou separaci tiskových barev (posun je u testovaných scannerů menší než jeden pixel).

Kromě odstranění výše zmíněných degradací je obraz upraven vhodně tak, aby se usnadnily další fáze zpracování. Mezi tyto úpravy patří rozšíření barevné škály (= lineární „roztažení“ barevné škály na celý interval) a filtry odstraňující šum a zvýrazňující hrany.

Takto předzpracovaný obraz je separován na jednotlivé tiskové barvy. Protože jednotlivé barvy používané při tisku map úplně nekryjí, je klasifikace schopná rozdělit i některé přetisky dvou různých barev, přetisky více barev (i světlých) jsou moc tmavé pro provedení separace (*přetisk* je situace, kdy je na stejném místě papíru otisk dvou a více barev). Výsledkem je rozklad barevného rastrového obrazu na několik bitmapových

(0/1) obrazů, každý pro jednu tiskovou barvu plus jeden pro místa, kde se klasifikace z nějakého důvodu nezdařila.

Jednotlivé vrstvy jsou pak dále filtrovány sadou kontextových filtrů. Jednak aby se vyřešila místa, kde se klasifikace nezdařila, jednak pro odfiltrování chyb. V této fázi se také detekují rastry — při tisku map se totiž kromě základních tiskových barev používá i několik odstínů těchto barev. Odstíny jsou řešeny tiskem jemného rastru základní barvy. Tyto rastry jsou tedy v závěrečné fázi detekovány a výstupem je další bitmapový obraz pro konkrétní odstín (a odstranění bodů z bitmapy pro sytou barvu).

Na takto rozložené mapě začíná práce moje.

3 Převod 2D→3D

Tato kapitola obsahuje obecné úvahy nad problémem převodu na jednotlivé tiskové barvy separovaného rastrového obrazu na digitální model terénu.

Nejprve několik pojmů používaných v textu: *topografická značka* (dále jen *značka*) je grafický symbol v mapě nesoucí nějakou informaci o mapované krajině, *reálné objeky* (dále jen *objekty*) jsou reálné věci v přírodě, které jsou pomocí značek zachyceny v mapě.[1] Některé značky odpovídají reálným objektům (např. studna), jiné ne (např. vrstevnice). *Digitálním modelem terénu* je virtuální trojrozměrný model krajiny spolu s dalšími informacemi vyčtenými z mapy, jako například druh porostu, vodní toky, rozmístění reálných objektů, apod.

Převod separovaného rastrového obrazu na digitální model terénu jsem rozdělil do tří kroků:

rozpoznání mapových značek: v mapách pro orientační běh se prakticky nepoužívá textových popisů, proto lze většinu informací vyčíst z jednotlivých značek. Informace, které vyčíst nelze (měřítko, ekvidistance, globální pozice mapy . . .) je třeba získat jinou cestou, např. ručním zadáním.

doplnění výškové souřadnice: mapy pro orientační běh, stejně jako mnoho jiných druhů map, zachycují krajinu v půdorysu (tj. v pohledu zhora),[2] při kterém se třetí, výšková, souřadnice „ztratí“. Výška je tedy zaznamenána pomocí značek — vrstevnic, rýh, apod.

triangulace: výstupem zpracování mapy má být triangulovaná podoba terénu zjemněná natolik, aby jednotlivé trojúhelníky obsahovaly vždy právě jeden druh porostu. Druh a hranice porostů lze vyčíst z plošných značek.

Rozpoznávání značek se podrobně věnuje kapitola 4, zde se tedy budu věnovat zbylým dvěma problémům.

3.1 Doplnění výškové souřadnice

I pokud rozpoznám všechny značky v mapě zakreslené, nemám ještě úplnou informaci k vytvoření digitálního modelu terénu. Část informace, kterou značka nese, není totiž do mapy z důvodu přehlednosti zakreslena a vyplývá z kontextu a okolních značek. Nejviditelnější příkladem jsou vrstevnice, které samy o sobě výšku neříkají, přestože jsou jediným zdrojem informací o výškových poměrech terénu (ostatní značky zachycují změny výšky mající pouze lokální charakter). Není to jediný případ — např. směr toku v potocích a řekách také není zakreslen, přestože je možné jej ve většině případech z mapy vyčíst.

Výška se na mapách kóduje pomocí *vrstevnic*, což jsou linie spojující místa se stejnou hodnotou této veličiny. Kdyby někdo krajinu vodorovně rozřezal na stejně tlusté pláty, budou vrstevnice obrysy těchto plátů v půdorysu (při pohledu „zhora“). Tloušťka pomyslných plátů se nazývá *ekvidistance* a bývá u map pro orientační běh většinou 5 m, ale může se různit (je nutné ji zadat jako parametr aplikace, neboť ji není možné z topografických značek vyčíst).

V případě, že není možné s tímto odstupem dostatečně věrně terén zachytit, používají se buď *pomocné vrstevnice* nebo další značky. Pomocné vrstevnice jsou vrstevnice kreslené jen lokálně na místech, kde by normální vrstevnice nestačily zachytit tvar terénu. Kreslí se čárkovanou čárou s poloviční ekvidistancí.

Značky, které se kromě vrstevnic používají k zachycení tvaru terénu, slouží k zachycení detailů, které jsou buď tak jemné, že vrstevnicemi by pořádně zachytit nešly (např. metr hluboká rýha) nebo by jejich zakreslování pomocí vrstevnic bylo nepřehledné.

V krajině je výška většinou prostou a spojitou (nikoliv však hladkou) reálnou funkcí dvou proměnných — zeměpisné šířky a délky. Nebyla by to příroda, kdyby neexistovaly vyjímky. Spojitost není zachována v místech prudkých srázů a ve zvláště zvrhlých případech není

[1] pozor, toto vymezení pojmů není schodné s označením v [2], kde je pro oba pojmy použito slovo „objekt“

[2] opomiňme problémy překreslení kulové plochy do roviny, kteréžto lze díky poměru poloměru Země a mapované oblasti u map pro orientační běh zanedbat

ani prostá. Tyto případy se naštěstí vyskytují vzácně. Topografie map pro orientační běh si v případě nespojitosti vypomáhá značkou „sráz“ a jejími variantami (viz. tab. 4.3) a přerušuje v těchto místech značení pomocí vrstevnic. Druhý případ, překryvy, nelze použitými prostředky zaznamenat.

Z přechozích odstavců vyplývá, že dvě sousední vrstevnice spojují místa s pevně určeným výškovým odstupem (pokud se mezi nimi nevyskytuje značka „sráz“). Chování výškové souřadnice mezi dvěma vrstevnicemi není v mapě většinou zaznamenáno a předpokládá se spojitý přechod, pokud tomu tak v krajině není, používají se další značky lokálního charakteru. Většinou pro ně platí, že jsou nadměrných rozměrů vzhledem ke zmenšení daném měřítkem.

Pokud se podaří určit výšku vrstevnic, je možné snadno dopočítat výšku mezi vrstevnicemi, například lineární interpolací. Jedna z možností je najít k bodu, který mě zajímá, sousední vrstevnice a výšku určit jako vážený průměr výšky těchto vrstevnic, kde váhou bude minimální vzdálenost od vrstevnic. To sice povede k nespojitosti první derivace na vrstevnicích, ale vzhledem k použití trojúhelníků k aproximaci terénu to nemusí vadit. V opačném případě je možné použít interpolace vyšších řádů.

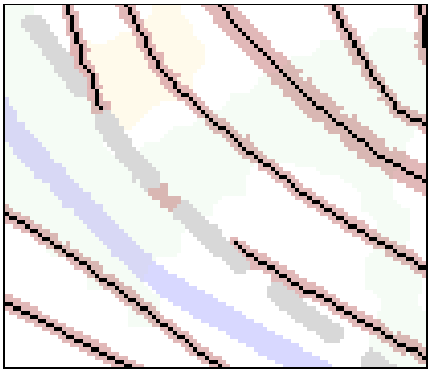
Určit, které dvě vrstevnice spolu sousedí, není těžké. Jedno z řešení je spouštět v pravidelných rozestupech z jednotlivých vrstevnic normály a testovat s kterými vrstevnicemi se protnou. Problém, který určování „sousedských“ vztahů značně znesnadňuje je chybovost algoritmu hledání liniových značek. Ne vždy se mu podaří spojit všechny části jedné značky (vrstevnice). Pracuje totiž „jen s čárami“. Nezná hlubší vlastnosti značek, neboť ty jsou pro různé liniové značky různé. Vrstevnice navíc mají v případě kolize s jinou značkou většinou nejmenší prioritu, takže k jejich přerušování dochází často.[3].

Příklad, kdy dále v této práci popsaný algoritmus nedokázal rozhodnout a přitom je řešení jednoduché, je na obrázku 3.1.

Veliká mezera v prostřední vrstevnici vznikla přetiskem s podobně orientovanou cestou. Pro napojovací algoritmus (kap 5.3) je tato mezera příliš velká, z pohledu vrstevnic je spojení jednoznačné.

Poslední otázkou, kterou je třeba zodpovědět pro korektní určení výšky v terénu, je určit sklon svahu, tj. která ze sousedních vrstevnic je výše a která níže. Tuto informaci by mohlo být možné vyčíst ze *spádnic*, což

[3]důvodem je právě „spojitost“ výšky, díky níž lze odhadnout její chování z okolních vrstevnic



Obrázek 3.1: Napojování vrstevnic — díra

jsou krátké čárky vyskytující se občas po stranách vrstevnic, které určují směr „dolů“ „Občas“ znamená, že se vyskytují jen na místech, kde by mohla být mapa nejednoznačná. To znamená, že nejsou součástí každé vrstevnice, takže jejich hledání může být poměrně komplikované. Navíc občasné slitky bodových značek s vrstevnicemi, které se nepodaří zcela rozpoznat, mohou způsobovat chybovost na spádnicích postaveném algoritmu.

Mnohem realističtější se jeví varianta založená na pomoci operátora, který na vybraných místech sklon svahu zadá a tato informace se pak automaticky rozšíří na okolí. Algoritmus automatického „rozšiřování“ musí být implementován v obou případech, protože jak již bylo řečeno, není spádníce součástí každé vrstevnice, ale jen vybraných.

3.2 Triangulace

Při triangulaci terénu je potřeba vycházet z plošných značek — formát digitálního modelu terénu specifikovaný v [2] totiž předpokládá, že pro každý trojúhelník bude dán jen jeden druh porostu, a tak je třeba případně provést zjemnění.

Triangulace je poměrně starý problém a je známo mnoho algoritmů, jak jej řešit. Triangulací speciálně v oblasti aproximace digitálního modelu terénu se zabývá například kniha [7].

4 Rozpoznávání značek

Tato kapitola popisuje jednotlivé skupiny mapových značek z topografického hlediska a pro značky bodové a plošné i možné způsoby jejich rozpoznávání (o roz-

Značka	O.[a]	Jméno
┼		kříž
┬		posed
↑		krmelec
⊙		hraniční kámen
×		jiný umělý objekt
•		malá kupka
•		balvan
▲		skupina balvanů
V		jáma
V		kamenná jáma
V	✓	jeskyně
V		jáma s vodou
∪		malá prohluben
∪	✓	pramen
O		studna
×		vývrat
O		výrazný strom

[a] může být značka natáčena?

Tabulka 4.1: Topografické bodové značky

poznávání značek liniových je kapitola 5). Při psaní této kapitoly jsem vycházel z knih [2, 3, 4].

Značky lze rozdělit na tři skupiny: *bodové*, *liniové* a *plošné*.

Bodové značky jsou symboly s pevně definovaným tvarem a barvou. Většina má i pevně definovanou orientaci, některé však lze otáčet. Tvar a barva určují druh, případné natočení značky určuje orientaci objektu v terénu. Více v kapitole 4.1.

Liniové značky jsou čáry různé tloušťky, barvy a typu. Buď odpovídají reálným objektům (např. železnice) nebo slouží k zaznamenání třetí, výškové, souřadnice (např. vrstevnice). Čáry bývají většinou kresleny po celé délce stejnou tloušťkou, existují však vyjímky. Topografické stránce liniových značek a požadavkům na rozpoznávací algoritmus se věnuje kapitola 4.2, rozpoznávání a zpracování pak podrobně kapitola 5.

Plošné značky jsou plochy vyplněné souvislou barvou, pravidelným vzorem nebo kombinací obého. Mohou být také ohraničeny čarou plnou nebo tečkovanou, což značí způsob hranice objektů (výrazná/nevýrazná hranice). Více v kapitole 4.3.

4.1 Bodové značky

Bodové značky jsou symboly s pevně daným tvarem, velikostí a barvou. Slouží k zachycení pozice, příp. orientace objektů, které jsou většinou mnohem menší, než aby mohly být do mapy překresleny ve zmenšení daném měřítkem. Tvar a barva jednoznačně určují druh značky a ta druh objektu. Informací, která je bodovou značkou zaznamenána je, že daný druh objektu je na daném místě v terénu. U bodových značek, které se kreslí různě natočené, je navíc z natočení možné určit orientaci objektu.

Seznam bodových značek je v tabulce 4.1. Oproti seznamu v [2] zde záměrně chybí značka „budova“, protože ta podle mého názoru patří spíše do skupiny značek plošných. Pokud ne z hlediska topografického, tak jistě co do způsopu rozpoznávání.

Bodové značky jsou značky s nejvyšší prioritou — pokud dojde ke kolizi s liniovou značkou stejné barvy, je liniová značka přerušena. Při rozpoznávání bodových značek je tedy jediné riziko v záměně části plošné či liniové značky za značku bodovou.

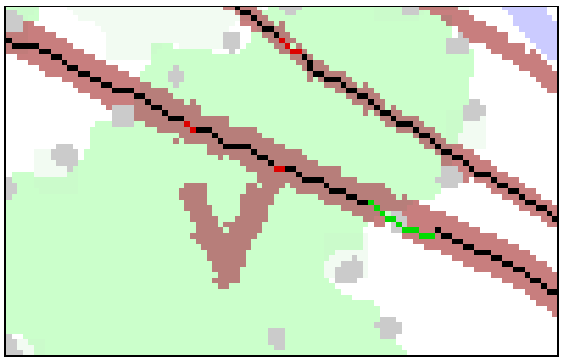
4.1.1 Rozpoznávání značek

O hledání a rozpoznávání symbolů s pevným tvarem bylo napsáno mnoho článků, je možné např. použít korelační metody, které porovnávají oblast s určitým vzorem a testují míru shody [5]. Dovolím si zde místo toho popsat svůj návrh algoritmu postaveném na jiném principu. Protože využívá algoritmů použitých při rozpoznávání liniových objektů, doporučuji čtenáři nejprve přečíst kapitolu 5.1 (definice použitých termínů).

Většina bodových značek je tvořena liniemi. Vyhledávání a klasifikace těchto značek by mohla být zařazena po vyhledání *l-atomů* (definice na str. 10) při rozpoznávání liniových značek. Je nutné zařadit toto vyhledávání ještě před napojování l-atomů v linie, algoritmus používaný při napojování totiž může omylem části značek zapojit do liniových (např. v místě kolize bodové a liniové značky).

Kód programu by vypadal přibližně takto:

```
2 atoms = najdi_latomy(pic);
3 klasifikace_latomů(atoms);
4
5 dots = najdi_bodové_značky(atoms);
6 klasifikace_latomů2(atoms, dots);
7
8 najdi_lepidlo(atoms, pic);
```



Obrázek 4.1: Chyba separace barev — slitek (došlo ke spojení hnědou čarou kreslené vrstevnice a jámy, i když v tomto případě jde spíše o chybu autora mapy)

Do normálního zpracování liniových značek byly přidány řádky 5 a 6[1].

Vlastní vyhledání bodových značek by v seznamu l-atomů provedla funkce *najdi_bodové_značky()*, funkce *klasifikace_latomů2()* by pak rozpoznané l-atomy označila v seznamu *atoms* jako součásti bodových značek (aby při hledání liniových značek „nezlobily“).

Výhodou takovéhoto postupu je využití segmentace obrazu, která se provádí při hledání liniových značek což by v součtu mohlo vést k zrychlení. Druhou výhodou je jednoduchá implementace funkce *klasifikace_latomů2()*, jejíž nějaká podoba bude muset být stejně implementována[2].

Nevýhodou navrženého algoritmu je jeho relativní neuniverzálnost — pokud bych chtěl pracovat s jinou množinou bodových značek které by byly neliniového charakteru (jaké se vyskytují například v turistických mapách), musel bych použít jiný algoritmus nebo tento nějakým ne zcela triviálním způsobem rozšířit.

Problém, na který bude třeba dát pozor, jsou slitky — při digitalizaci mapy a v průběhu procesu rozkladu na tiskové barvy dochází někdy ke spojení v mapě oddělených souvislých oblastí (obr. 4.1). Při prohledávání l-atomů je pak potřeba dát pozor na další čáry vyskytující se v bezprostřední blízkosti bodové značky (navazující na ni).

[1] funkci *najdi_latomy()* lze v knihovně ML najít pod jménem *ml_pic_find_atoms()*, funkci *klasifikace_latomů()* pod jménem *ml_atoms_classify()* a funkce *najdi_lepidlo()* se jmenuje *ml_pic_glue()*

[2] funkce *klasifikace_latomů2()* má za úkol odstranit ze seznamu ty l-atomy, které patří bodovým značkám. Jiné možné řešení je rozpoznat a z obrazu odstranit bodové značky ještě před začátkem hledání značek liniových

Značka[a]	Popis
0,1	vodící linky sever-jih
0,1 x	vodní tok
0,1	meliorační příkop
0,1	úzká bažina
0,4	železnice
0,4	zpevněná cesta
0,4	vozová cesta
0,3	cesta
0,2	pěšina
0,1	průsek
0,1	elektrické vedení[b]
0,1	zeď
0,1	překonatelný plot
0,1	nepřekonatelný plot
0,1	hráz
x	silnice

[a] čísla vedle linií udávají tloušťku čáry v mm, mohou se však v jednotlivých mapách mírně lišit

[b] příčné čáry označují sloupy

Tabulka 4.2: Topografické liniové značky popisující reálné objekty

4.2 Liniové značky

Liniové značky jsou velmi důležitou třídou značek. Slouží nejen k zaznamenání polohy a podoby reálných objektů, ale jejich správné rozpoznání je základ k určení nadmořské výšky v jednotlivých místech mapy, tj. k doplnění třetí, výškové, souřadnice k jinak jen dvourozměrnému obrazu. To klade velké nároky na nízkou chybovost rozpoznávání — chybné rozpoznání bodové nebo plošné značky má v podstatě jen lokální význam, chyba při rozpoznávání vrstevnic může vést k chybnému určení nadmořské výšky celé oblasti.

Nejprve bych rád dal přesnější význam třem termínům, které v dalším textu používámi: *typem čáry* se rozumí způsob pravidelného přerušování čáry — plná (nepřerušovaná), čárkovaná a tečkovaná. Jednotlivým již nepřerušovaným úsekům čárkované čáry říkám *čárky*, úsekům tečkované čáry *tečky*.

Nalezenení a rozpoznání liniových značek patří do skupiny úloh nazývaných *vektorizace*. Tato třída úloh zajímá informatiku již poměrně dlouho, neboť se vyskytuje i v mnoha jiných oblastech, například při digitalizaci technických výkresů [8].

Ovšem při zpracování map je vektorizace pouze základním krokem — s informacemi získanými vektori-

Značka	Popis
	vrstevnice[a]
	pomocná vrstevnice
	hluboká rýha
	mělká rýha
	schůdný sráz
	neschůdný sráz
	kupka
	prohlubeň

[a] každá pátá je zvýrazněná

Tabulka 4.3: Topografické liniové značky popisující výškové rozložení terénu

zací je třeba dále automaticky pracovat. Proto např. nalezení jedné čárky čárkovanou čarou kreslené značky nemá žádný význam, spíše naopak — přerušovaná čára značí něco jiného než čára plná a proto je pro výsledek dalšího zpracování lépe tuto čárku „nenajít“ než aby došlo k omylu.

K určení o jakou linovou značku se jedná je potřeba znát barvu, tloušťku a typ čáry. Tyto údaje je zapotřebí v průběhu vektorizace zjistit.[3]

Liniové značky jsou čáry obecné křivosti sloužící k zaznamenání reálných objektů liniového charakteru (tab. 4.2) nebo k zachycení výškové souřadnice (tab. 4.3). Jsou kresleny černou, hnědou nebo modrou čarou různé tloušťky a typu. U většiny značek je tloušťka pevně daná a po celé délce se nemění, u některých to neplatí. Buď mají tloušťku libovolnou, ale pro celou značku pevnou (silnice) nebo se může i v průběhu jedné značky měnit (vodní tok, hluboká rýha). U těchto značek tloušťka zachycuje skutečný rozměr reálného objektu (silnice, vodní tok) nebo je změnou tloušťky zaznamenaná jiná informace (u značky „rýha“ je to změna hloubky).

Nepříjemné na tloušťku měnících značkách je, že k uložení informace, kterou nesou, nestačí pouze jejich dráha, ale je nutné ukládat i změny tloušťky. Kvůli této vlastnoti některých značek bude nutné buď vhodným způsobem rozšířit formát navržený v [2] pro uložení liniových objektů nebo tyto značky do výsledného digitálního modelu terénu ukládat jako značky plošné.

Rozpoznávací algoritmus musí u čar kreslených čárkovanou i tečkovanou čarou najít a spojit jednotlivé čárky a tečky, z kterých jsou tvořeny. Podobný problém jej čeká i u čar plných — ty jsou často přerušeny záměrně tvůrcem mapy v místech, kde dochází ke kolizi s jinou značkou, ať již bodovou nebo liniovou (liniové značky mají nižší prioritu než značky bodové, protože je možné jejich chování v místě kolize odhandout ze zbytku značky). K přerušení může dojít také v důsledku přetisku části značky jinou barvou která nejde odseparovat. Algoritmus musí být také schopen vyřešit správné napojení v místech křížení a spojování (resp. rozdělování) čar.

Mapa je poměrně komplikovaný obraz, při jehož interpretaci udělá snadno chybu i zkušený člověk, proto je asi zcela automatické a bezchybné strojové rozpoznávání pravděpodobně nereálné. Bude tedy zřejmě zapotřebí umožnit „ruční“ opravy. Pro práci operátora, který by chyby opravoval je vhodnější, když rozpoznávací algoritmy budou schopny označit místa, kde nedokáží rozhodnout (než aby za každou cenu našly řešení) a případně navrhnou možné varianty. Jiné možné řešení téhož je si u jednotlivých spojů pamatovat kvalitu nebo „míru jednoznačnosti“. Operátor pak může zkontrolovat jen ty spoje, které jsou horší než zadaná mez.

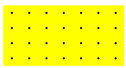
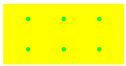
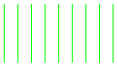
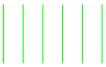
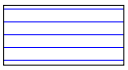
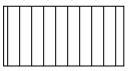
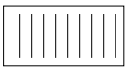
V případě, že by bylo možné i v průběhu dalšího zpracování získat jiné možné varianty napojení, mohly by další části aplikace v případě nekonzistence rozpoznaných značek samy zkusit provést opravu. Příkladem může být doplnění výškové souřadnice (kap. 3.2), kdy je možné díky hlubší analýze významu značek takové chyby nalézt.

Rozpoznávání liniových značek jsem se věnoval velice důkladně, takže je tomuto tématu věnována celá kapitola 5.

4.3 Plošné značky

Plošné značky slouží k zaznamenání informací o porostu a povrchu terénu nebo objektů, které je možné i po zmenšení daném měřítkem přímo zakreslit obrysem. Značí se plochou vyplněnou vzorem (pravidelným i nepravidelným) nebo souvislou barvou, případně kombinací obého. Vzor má v případě map pro orientační běh povahu bodů nebo rovnoběžných čar různého typu (čárkované nebo plné). Seznam všech značek je v tabulce 4.4.

[3] barva, tloušťka a typ čáry určují druh značky, bohužel ne jednoznačně — např. *vrstevnice* i *rýha* se značí stejně (jen *rýha* může měnit tloušťku)

Značka	Popis
	budova
	otevřený prostor
	pole
	sad
	neprůchodný hustník
	průchodný hustník
	průchodný les
	řídký les
	paseka
	hustý podrost
	řídký podrost
 *nebo* 	vodní plocha
	průchodná bažina
	neprůchodná bažina
	bažinatý les
	zpevněná plocha
	zastavěná plocha
	zakázaný prostor
	rozbitý povrch[a]
	kamenné pole[b]

[a] rastr je nepravidelný
[b] rastr je nepravidlený

Tabulka 4.4: Topografické plošné značky

4.3.1 Rozpoznávání značek

Jeden z možných postupů rozpoznávání plošných značek je v ponechání tohoto rozpoznávání až po rozpoznání značek bodových a liniových, ty z obrazu odstranit a pak teprve hledat značky plošné. Vzorky jsou povahy bodů nebo čar, bylo by tedy možné nechat je rozpoznat jako jednotlivé bodové či liniové značky a pak je „najít“ v datových strukturách bodových a plošných značek. Vlastní vyhledávání plošných značek by pak znamenalo nalezení souvislých barevných ploch v obraze, kde již nic jiného není, za kterým by následovalo prohledání seznamů bodových a liniových značek, jestli některé nejsou součástí značek plošných a provést úpravy těchto seznamů.

Jiná možnost (z pohledu detekce liniových značek samozřejmě jednoduší, neboť by odpadly problémy s detekcí rastrů, viz. kap. 5.2.3) je najít a z obrazu odstranit plošné značky ještě před rozpoznáváním liniových značek.[4]

U linií je třeba rozlišit dva případy — čáry nebo body tvoří výplň (vzor) nebo tvoří linii zvýrazňující hranici značky. U hraničních čar bývá pro jednu značku několik možností, včetně žádné čáry.

Je dobré se zamyslet ještě nad dvěma problémy — díry v plochách a posun barev.

Pokud dojde ke kolizi plošné značky se značkou bodovou či linovou jiné barvy (ke které dochází velmi často), dojde k přetisku. Ne všechny přetisky lze rozseparovat (např. přetisk černu barvou) a tak v plošné značce vznikají „díry“. Pokud je tedy uvnitř plochy díra, mohla vzniknout z tohoto důvodu a je vhodné ji zacelit a přidat k ploše. Při zacelování je však nutné postupovat opatrně — přestože černou čárkovanou čáru (*pěšina*) obklopuje zelená plocha (*hustník*), neznamená to, že na pěšině rostou stromy.

Druhým problémem je lícování barev při tisku mapy. Lícování barev je obecný problém vícebarevného tisku — každá tisková barva je nanášena na papír zvlášť a pokud se mezi nanesením barev list papíru pohne (resp. neprojde přesně tiskovými válci), jednotlivé barvy jsou navzájem posunuté nebo pootočené. Hraniční čáry plošných značek nemusí pak vést přesně po hranicích vyplněných ploch, protože jsou tisknuty jinou barvou. Proto, je-li to možné, je lepší použít mapu kde je tento posun v nejhorším případě na hranici viditelnosti (tj. méně než $0{,}1$ mm), neboť čáry používané k ohraničení značek mají tloušťku $0{,}1$ mm.

[4] pro každý druh rozpoznávání je snažší rozpoznávat obraz, v kterém už byly ostatní typy značek nalezeny a odstraněny. Určit pořadí mi připomíná otázku „slepice nebo vejce¿‘

Druhým důvodem, proč hraniční čáry nemusí vést přesně po hranici, je dodatečné zpracování (filtrace) v průběhu procesu separace tiskových barev, které okraje ploch mnohdy o několik pixelů pozmění.

5 Rozpoznávání liniových značek

Tato kapitola se věnuje podrobně rozpoznávání liniových značek. Požadavky, které jsou na rozpoznávací algoritmus kladeny byly již zmíněny v kapitole 4.2, tato kapitola obsahuje konkrétní postupy, které většinu těchto požadavků řeší.

Díky tomu, že liniové značky jsou vždy po celé délce kresleny jednou barvou, můžem zpracovat jednotlivé barevné vrstvy nezávisle jednu po druhé. Algoritmy jsou tedy uváděny jen pro jednu vrstvu, ve výsledné aplikaci musí být aplikovány vícekrát — pro každou barvu, kterou jsou liniové značky kresleny.

Základem celého rozpoznávání je nalezení *liniových atomů*. Liniové atomy jsou kostry jednotlivých souvislých oblastí „rozkrájené“ v místech křížení nebo rozdělování. O tom, jak je najít je kapitola 5.2. Z těchto základních kamenů jsou pak složeny jednotlivé liniové značky. Nejprve najdu všechny možné sousedy a cesty, kterými je mohu spojit. Protože však důvodů přerušení čar je několik, je i několik druhů spojů. O druzích spojů a jejich vytváření je kapitola 5.3. Posledním úkolem je vybrat z možných kandidátů toho pravého (sem patří i případné rozhodnutí nevybrat žádného). To je asi nejtěžší úkol, o mém pokusu o jeho vyřešení je kapitola 5.4.

Bohužel se mi nepodařilo najít literaturu zabývající se vyhledáváním a napojováním čar obsahující konkrétní algoritmy (do rukou se mi dostala pouze diplomová práce pana Voříška [8], zabývající se vektorizací technických výkresů, ale algoritmy v této práci uváděné nelze v mapách použít — jsou navrženy pro hledání úseček a napojování je řešeno jen ve velmi omezené míře). Proto je celý zde uváděný postup mnou vytvořený, až na základ ztenčovacího algoritmu, který je převzat z knihy [5].

5.1 Definice

Vstupem programu je na jednotlivé tiskové barvy rozseparovaná zdigitalizovaná mapa. Tento obraz se skládá z jednotlivých barevných separací, kterým budu říkat *vrstvy*. Každá vrstva přísluší právě jedné tiskové barvě nebo odstínu a lze na ní pohlížet jako na bitmapový (0/1) obraz kreslený příslušnou barvou. Vrstvy se skládají z pixelů. Ve většině případů pracuji s jednou, libovolnou, ale pevně danou vrstvou. Pokud bude potřeba vrstvy rozlišit, budu se odvolávat na barvu separace, kterou je vrstva kreslená — např. „pixel v zelené vrstvě . . . “.

Pixely značím malými písmeny z konce abecedy: x, y, ... Hodnota pixelu může nabývat hodnot:

$$x = \begin{cases} 1 & \text{pixel je obarvený,} \\ 0 & \text{jinak (má barvu pozadí).} \end{cases}$$

(pixely s hodnotou 1 budu označovat jako *tmavé*, s hodnotou 0 jako *světlé*). Symbolem $\overline{x}$ budu označovat opačnou hodnotu, tj. pro $x = 0$ je $\overline{x} = 1$ a pro $x = 1$ je $\overline{x} = 0$.

Sousedy pixelu budu označovat stejným písmenem s horním indexem: x^0, x^1, . . . ,x^7.. Jejich rozložení je dáno následovně:

x^1	x^0	x^7
x^2	x	x^6
x^3	x^4	x^5

Množinu všech sousedů pixelu x budu značit $\mathbf{S}(x) = \{x^0, x^1, \ldots, x^7\}$. Platí: $y \in \mathbf{S}(x) \Rightarrow x \in \mathbf{S}(y)$.

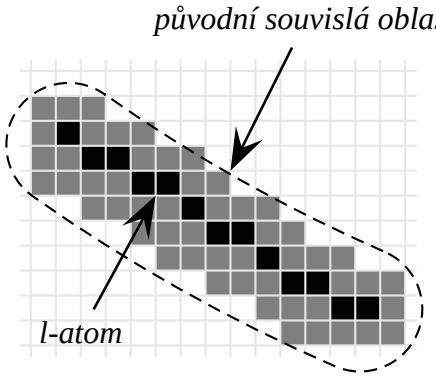
Obraz je „orámován“ světlými jakobypixely, s kterými už nepracuji. To proto, aby každý pixel, včetně pixelů krajních, měl osm sousedů.

Souvislá oblast je neprázdná množina pixelů $\mathbf{P}$, pro kterou platí, že $\forall x, y \in \mathbf{P}$ existuje posloupnost $x_0, \ldots, x_n \in \mathbf{P}$, $n \in \mathbb{N}$, $x_0 = x$, $x_n = y$ taková, že $x_i \in \mathbf{S}(x_{i-1})$ pro $\forall i \in \{1, 2, \ldots, n\}$ (jinými slovy: mezi každými dvěma pixely existuje cesta která celá vede uvnitř oblasti). Navíc pro každé $y \notin \mathbf{P}$, pro které existuje $x \in \mathbf{P}$ takové, že $y \in \mathbf{S}(x)$, platí, že $y = 0$ (tj. $\mathbf{P}$ je maximální).

Jednoduchá čára je souvislá oblast $\mathbf{P}$, kde $\forall x, y \in \mathbf{P}$, existuje právě jedna posloupnost vzájemě různých $x_0, \ldots, x_n \in \mathbf{P}$, $n \in \mathbb{N}$, $x_0 = x$, $x_n = y$ takových, že $x_i \in \mathbf{S}(x_{i-1})$ pro $\forall i \in \{1, 2, \ldots, n\}$ (neboli cesta mezi lib. pixely v oblasti je jednoznačná). Číslu n budu říkat *délka cesty*.

Koncový pixel jednoduché čáry $\mathbf{P}$ je pixel $x \in \mathbf{P}$, pro který $|\{\, y \mid y \in \mathbf{S}(x),\ y \in \mathbf{P} \,\}| \leq 1$. Každá jednoduchá čára má jeden nebo dva koncové pixely.

Délka jednoduché čáry je délka cesty mezi jejími koncovými pixely. V případě, že má čára jen jeden koncový pixel, je délka 0 (což je konzistentní se zbytkem definice).



Obrázek 5.1: Souvislá oblast a l-atom

Formalismu bylo učiněno zadost. Pojmy by měly být intuitivně jasné. Doufám, že definice se s touto představou shodují.

5.2 Hledání liniových atomů

Tato kapitola se pokusí popsat, jakým způsobem najít v obraze *liniové atomy* (dále jen *l-atomy*). L-atom je idealizací jednotlivých úseků plnou nebo čárek u čárkovanou nebo teček u tečkovanou čarou kreslených čar. Z pohledu definic z kapitoly 5.1 je to jednoduchá čára. Vzniká ztenčením jednotlivých souvislých oblastí v obraze. Za ideálních podmínek vede osou jednotlivých souvislých úseků čar a čárek (obr 5.1). V případě teček l-atomy zkolabují do jednoho bodu, tj. jednoduché čáry délky 0. L-atomy jsou základními stavebními kameny pro rozpoznávání linií.

K uložení dráhy l-atomu postačí jednoduché kódování. Začnu z koncového pixelu a postupuji postupně po jeho jednotlivých pixelech a každý krok zakóduji číslem 0–7 podle toho, do kterého souseda jsem se posunul. Z jednoznačnosti cesty spojující lib. dva pixely jednoduché čáry vyplývá, že je tento postup deterministický.

Tento způsob kódování jsem až dodatečně našel popsaný v knize [5] pod názvem *Chain Codes.* V publikované verzi je kódování směrů pootočeno o $90°$.

> Příklad: kód l-atomu nakresleného na obrázku 5.1 by v případě, že začnu v levém horním rohu, byl: 5, 6, 5, 6, 5, 5, 6, 5, 5, 6, 5, 6.

Rád bych upozornil na obrázek 5.2 a 5.3, kde jsou příklady čar, které nejsou jednouché. Nesplňují podmínku jednoznačnosti. Podmínka jednoznačnosti, kterou na jednoduché čáry kladu, nemá žádný hlubší význam než zjednodušit postup kódování a práci s l-atomy — např. při určování směrů.

Toto kódování je relativní, je tedy zapotřebí zapamatovat si absolutní souřadnice některého pixelu l-atomu, nejlépe počátečního (aby byla práce s l-atomy efektivnější, zapamatovávám si i souřadnice koncového bodu).

K nalezení l-atomů jem použil následující postup: nejprve ztenčím všechny souvislé oblasti na tloušťku jednoho pixelu a vzniklou síť rozdělím na l-atomy. Postup ztenčování popisuje kapitola 5.2.1, dělení sítě kapitola 5.2.2. Kapitola 5.2.3 se věnuje klasifikaci takto nalezených l-atomů, neboť zdaleka ne všechny patří k liniovým značkám.

5.2.1 Ztenčování

Jako základ algoritmu použitého k ztenčování jsem použil algoritmus publikovaný v [5] pod jménem *Thinning algorithmus.* Pracuje na principu postupného ztenčování — při každém průchodu obrazem „odloupne slupku“ o tloušťce jednoho pixelu a přitom hlídá, aby oblasti původně souvislé zůstaly souvislé i nadále. Prochází jednotlivé tmavé pixely obrazu a u každého testuje, jestli je možné jej odstranit — zesvětlit. Poté, co projde celý obraz, nastaví barvu z tmavé na světlou u všech pixelů, u kterých určil, že je může odstranit, a začne procházet celý obraz znovu. To dělá tak dlouho, dokud dochází ke změnám.

Test probíhá následovně: označíme-li

$$F(x) = \sum_{i=0}^{7} x^i$$

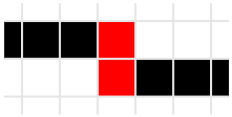
(počet tmavých sousedů x) a

$$Z(x) = \overline{x^7}x^0 + \sum_{i=1}^{7} \overline{x^{i-1}} \cdot x^i$$

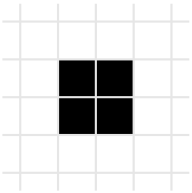
(počet změn ze světlé na tmavou při průchodu dokola po sousedech x), pak tmavý pixel x lze odstranit, pokud jsou splněny všechny čtyři následující podmínky:

$$\begin{aligned} F(x) &\in \langle 2, 6\rangle \\ Z(x) &= 1 \\ x^0 \cdot x^2 \cdot x^6 &= 0 \quad \text{nebo} \quad Z(x^0) \neq 1 \\ x^0 \cdot x^2 \cdot x^4 &= 0 \quad \text{nebo} \quad Z(x^2) \neq 1 \end{aligned}$$

Z uvedeného plyne, že změny mohu provádět dřív než až po průchodu celým obrazem, takže není nutné mít celý obraz uložený v paměti dvakrát.



Obrázek 5.2: Růžky



Obrázek 5.3: Kolečko

Tento algoritmus je pomětně jednoduchý a rychlý, bohužel má dva drobné nedostatky. Za prvé nezaručuje, že výsledkem bude síť jednoduchých čar. Algoritmus totiž „zapomíná na růžky“ — ani jeden ze zvýrazněných pixelů na obrázku 5.2 není možné odstranit (není pro ně splněna podmínka $Z(x) = 1$). Druhým, závažnějším, nedostatkem algoritmu je možnost „vypaření“ celé souvislé oblasti (nezůstane z ní žádný tmavý pixel). Příklad je na obrázku 5.3. Při průchodu algritmu budou všechny jeho pixely označeny za smazatelné. Tímto „vypařováním“ jsou postiženy oblasti, které mají tvar kruhu se sudým průměrem. U větších oblastí je toto riziko minimální neboť prakticky vždy se od ideálního kruhu liší (stačí jeden pixel). U malých teček, které jsou složeny jen z několika pixelů je pravděpodobnost mnohem vyšší a skutečně k tomu v reálných případech dochází.

Rozšířil jsem tedy publikovaný algoritmus: „vypařování“ lze zabránit testem

$$x^0 + x^1 + \cdots + x^7 > 0$$

(tj. alespoň jeden soused je tmavý) před vlastním odstraněním pixelu označeného za smazatelný. V případě neplatnosti podmínky mazání neprovedu. $x^0, \ldots, x^7$ jsou aktuální hodnoty sousedů v průběhu mazání.

Odstranění přebytečných růžků lze provést průchodem celého obrazu po skončení ztenčovacího algoritmu a otestováním každého tmavého pixelu. Pokud splní alespoň jednu z následujících podmínek, smažu jej (ze-světlím):

$$x^0 \cdot x^2 \cdot \overline{x^4} \cdot \overline{x^5} \cdot \overline{x^6} = 1$$
$$x^2 \cdot x^4 \cdot \overline{x^6} \cdot \overline{x^7} \cdot \overline{x^0} = 1$$
$$x^4 \cdot x^6 \cdot \overline{x^0} \cdot \overline{x^1} \cdot \overline{x^2} = 1$$
$$x^6 \cdot x^0 \cdot \overline{x^2} \cdot \overline{x^3} \cdot \overline{x^4} = 1$$

(jedná se o čtyřikrát to samé, jen vždy „pootočené o $90°$“). Při procházení obrazu musím stejně jako u předchozího vylepšení testovat momentální stavy pixelů v průběhu procházení.

Výsledkem takto rozšířeného algoritmu je síť již jen jeden pixel širokých čar. Křižovatky zůstaly.

Poslední, co je třeba vyřešit je určení tloušťky čáry. Algoritmus v této podobě neumožňuje z výsledku určit tloušťku původní čáry.

Protože čáry mohou měnit tloušťku, je rozumné uvažovat o tloušťce v každém pixelu ztenčené čáry. Variant, jak tloušťku spočítat je několik, nejkorektnější by asi bylo v každém bodě ztenčené čáry sestrojit kolmici a spočítat pixely původní čáry, kterými prošla. Toto řešení je ale výpočetně náročné a nese v sobě další otázky — např. co je kolmice na jednoduché čáře.

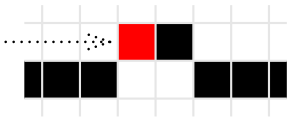
Další možné řešení je založené na jiné myšlence — při každém ztenčovacím průchodu „sloupnu vrstvu“ jednoho pixelu. Je tedy rozumné považovat za „tloušťku pixelu“ číslo průchodu, při kterém byl smazán některý soused. Při ztenčování u každého pixelu, který se rozhodnu odstranit, projdu všechny tmavé sousedy a nastavím jim tloušťku na hodnotu rovnou počtu zatím započatých průchodů obrazem.

Použil jsem druhé řešení pro jeho rychlost a jednoduchost, které navíc v praxi dává velice dobré výsledky.

5.2.2 Řezání

Výsledkem ztenčování je síť jednoduchých čar, která ovšem může obsahovat křižovatky (uzly) a cykly. V této síti najdu jednotlivé l-atomy. L-atomy budou jednotlivé úseky mezi křižovatkami, v místech křížení budou jednotlivé l-atomy končit (začínat). Cykly rozdělím na několik dílčích l-atomů.

Může se zdát nelogické čáry takto rozřezávat, když hned po fázi hledání l-atomů přijde na řadu jejich napojování. Proč si přidělávat práci, proč nenechat čáry rovnou v celku? Důvodem je obtížné určení, kterou cestou čára za uzlem pokračuje a jestli vůbec pokračuje, jestli například v tomto místě nekončí a nepřipojuje se k jiné čáře. Řešení těchto otázek nechám až na napojovací algoritmus (kap. 5.3).



Obrázek 5.4: Zbytečné dělení jednoduché linie

Při hledání l-atomů procházím obraz (výsledek ztenčování) po řádkách zleva doprava ze zhora dolů a hledám tmavé pixely. Pokud na nějaký narazím, vydám se po jeho tmavých pixelech a zapamatovávám si cestu. V případě, že dojdu na místo, kde další postup už není možný (konec čáry), prohlásím zapamatovanou cestu za l-atom a odstraním ji z obrazu (pixely zesvětlím) Podobně postupuji, pokud narazím na místo, kde další postup není jednoznačný (křižovatka) — vrátím se o jeden krok zpět a zapamatovanou cestu (o jeden krok kratší) prohlásím za l-atom a také ji odstraním z obrazu. Pixel, kde jsem původně zastavil, a všechny jeho tmavé sousedy prohlásím za *křižovatku*. Tento „ústup“ o jeden krok není v této fázi nutný, usnadňuje však další fáze napojování (v podstatě jde o to, že mám pak při napojování větší „manévrovací prostor“).

Poté, co jsem takto našel a z obrazu odstranil jeden l-atom, pokračuji v procházení a prohledávání obrazu v místě, kde jsem byl přerušen.

Tento postup je rychlý a jednoduchý. Nevýhodou je, že v některých místech zbytečně rozdělí jinak korektní jednoduché čáry — kamenem úrazu je začátek cesty, kterýžto se stane koncovým (resp. počátečním) pixelem vznikajícího l-atomu. Ten je vždy na prvním tmavém pixelu ve zbývajícím obraze při procházení zleva doprava zhora dolů. To ale nemusí být, a často nebývá, koncový pixel nějaké jednoduché linie. Příklad je na obrázku 5.4.

Řešením by bylo v okamžiku, kdy narazím na čáru se nejprve pokusit najít její konec. To ale v obecném případě není jednoduché — čára může tvořit smyčky a procházet mnoha křižovatkami, připojené čáry mohou tvořit „uši“ apod. (pravděpodobně jediným řešením by bylo procházení do hloubky).

Na druhou stranu je pro napojovací algoritmus takováto přerušení velmi jednoduché korektně napojit, takže v konečném důsledku má použitý postup za následek pouze mírné zvětšení počtu l-atomů.

5.2.3 Klasifikace

Algoritmus hledání l-atomů jak byl popsán v předchozích dvou kapitolách je „všežravý“ — zpracuje do podoby l-atomů vše, na co ve vrstvě narazí, včetně bodových a plošných značek. Proto je vhodné l-atomy projít a pokusit se označit ty, které liniovým značkám nepatří.

Nejjednoduší řešení je u l-atomů vzniklých z plošných značek tvořených souvislou barevnou plochou. L-atomy, které takto vzniknou mají velkou tloušťku a navíc se tato v průběhu l-atomu hodně mění. L-atom zamítnu, pokud je splněna alespoň jedna z podmínek:

$$width > k_1$$
$$\frac{\sqrt{S^2}}{l} > k_2$$

kde *width* je průměrná tloušťka l-atomu, S^2 je střední kvadratická chyba tloušťky a l je délka l-atomu.

Je dobré tuto klasifikaci provádět jen u vrstev, kde má praktický smysl — v modré, zelené a černé. V jiných se zároveň liniové značky a souvislou barvou vyplněné plošné značky nevyskytují. Tento způsob klasifikace je poměrně jednoduchý, a má jistou chybovost. Je lépe se chybám vyhnout, než je opravovat.

Klasifikace l-atomů vzniklých z bodových značek je obtížnejší. Jedno z možných řešení je naznačeno v kapitole 4.1.1, které spočívá v zařazení rozpoznávání bodových značek právě do tohoto místa zpracování značek liniových.

V současné verzi knihovny se klasifikace „bodových“ l-atomů neprovádí. Naštěstí nerozpoznání takovýchto l-atomů při dalším rozpoznávání čar příliš nevadí, neboť jsou jen málokdy prohlášeny za součást liniových značek. Rozpoznání těchto l-atomů by mírně zvýšilo spolehlivost hledání značek liniových.

Úkolem, který je potřeba vyřešit je rozpoznání l-atomů vzniklých z ostatních částí plošných značek. Tedy těch, které nejsou tvořeny souvislou barevnou plochou (jedná se hlavně o rastry ať už bodové nebo liniové). Tyto l-atomy způsobují velkou chybovost v rozpoznávání liniových značek — ve většině případů jsou také prohlášeny za liniové značky a nesmyslně popropojovány. Pokusy o řešení tohoto problému jsou rozebrány v kapitole 6.4.

5.3 Lepidlo

Název kapitoly trošku obrazně, ale přesně, vystihuje druhý krok při hledání liniových značek: najít všechna možná spojení s jinými l-atomy pro každý konec každého l-atomu (tedy „lepidlo“ mezi jednotlivými l-atomy). Cílem je najít každé i jen trochu možné pokra-

```
1  while (Q_p < limit) do
2        O = O \ {p}; U = U ∪ {p};
3
4        foreach x ∈ S(p) \ U do
5          if (x ∈ T)
6            q = penalty_C(x, p, ...);
7          else
8              q = penalty_S(x, p, ...); fi
9
10         if (x ∉ O)
11           O = O ∪ {x}; Q_x = q + Q_p;
12         elsif (Q_x > q)
13             Q_x = q + Q_p;
14         fi
15       od
16
17       if (O ≡ ∅) exit; fi
18
19       p = x ∈ O pro které je Q_x minimální;
20
21       if (p ∈ T)
22         nalezl_jsem_spoj( ... );
23         U = U ∪ S(p);
24       fi
25 od
```

Obrázek 5.5: Jádro algoritmu vyhledávajícího možná napojení l-atomů

čování l-atomu. Pod slovem „možné“ myslím „v nějaké reálné situaci v mapě se vyskytující“ — nemá smysl se zabývat spoji s l-atomy na druhém konci mapy.

Algoritmus, který jsem pro hledání možných spojů navrhl je lokální prohledávání okolí konce l-atomu. To znamená projít pixely v okolí konce l-atomu a pokusit se spojit je s jinými l-atomy, které naleznu.

Množinu pixelů, které projdu při prohledávání budu nazývat *prohledávaná oblast*. Velikost této množiny je zhora omezena velikostí mapy (počtem pixelů v mapě), ve reálných případech je její velikost mezi jednotkami a tisícovkami pixelů.

Pro snažší popis jádra algoritmu na vyhledávání spojů označím

$\mathbf{L} = \{\, x \mid x$ je pixel nějakého l-atomu $\}$

$\mathbf{T} = \{\, x \mid x$ je koncový pixel nějakého l-atomu $\}$

Platí $\mathbf{T} \subseteq \mathbf{L}$.

Algoritmus je na obrázku 5.5. Význam proměnných použitých v algoritmu a jejich počáteční hodnoty:

limit kladná konstanta udávající „odvážnost“ algoritmu.

p právě aktivní pixel. Počáteční hodota je koncový pixel l-atomu, který chci prodloužit.

U množina pixelů, které nemá cenu prohledávat. Buď jsou součástí některé linie nebo je již algoritmus procházel. Počáteční hodnota $\mathbf{U} = \mathbf{L} \setminus \mathbf{T}$.

O množina otevřených pixelů, tj. těch, které jsem navštívil, ale ještě neotestoval. Počáteční hodnota $\mathbf{O} = \{p\}$.

Q_x kvalita pixelu x. Počáteční hodnota $Q_x = 0$ pro počáteční pixel, pro ostatní pixely v obraze libovolně (při první návštěvě pixelu je tato hodnota nastavena)

(pro připomenutí: $\mathbf{S}(x)$ značí souseda x, přesná definice na str. 9)

Aby byla zaručena konečnost algoritmu (a nejen kvůli tomu) musí funkce $penalty_S()$ a $penalty_C()$ vracet pro libovolný vstup kladnou hodnotu. Konečnost pak plyne z toho, že hodnoty Q_x do množiny **O** přidávaných pixelů jsou vždy větší než Q_x odebíraného pixelu a z toho, že počet pixelů v **O** se stejnou hodnotou Q_x je vždy konečný.

Pokud si současně s přidáváním, resp. změnou, Q_x budu ukládat, resp. měnit, uspořádané dvojice (p, x) (neboli odkud jsem se do pixelu x dostal), bude po skončení algoritmu v seznamu pro každý pixel y, kterým jsem prošel, právě jedna dvojice (q, y) (q je pixel z kterého vedla do y nejlepší cesta, cesta, díky které má Q_y takové, jaké má). Z těchto dvojic pak mohu poskládat pro každý pixel v **U** optimální cestu do počátku.

Základní otázkou pro další úvahy je, proč jsou l-atomy patřící k jedné značce rozděleny, proč to není jen jeden l-atom. Při analýze jsem dospěl k tomu, že rozdělení vznikají jedním z těchto tří důvodů:

„křižovatky“ — rozdělení zapříčinil algoritmus hledání l-atomů (kap. 5.2.2). To se stává hlavně v místech uzlů, kde se potkávalo více čar, a dále pak jako cena za jednoduchost algoritmu řezajícího šíť čar na jednotlivé l-atomy,[1]

„přetisky“ — rozdělení je způsobeno přetiskem se značkou jiné barvy, kterou se nepodařilo odseparovat.

[1] jedná se o problém dělení jinak souvislých čar — podrobně popsán v závěru kapitoly 5.2.2

„prázdný prostor“ — v mapě je skutečně liniová značka přerušena. Důvodem je úmysl tvůrce mapy, který tak vyřešil kolizi s jinou značkou nebo je čára kreslena čárkovanou či tečkovanou čarou, u kterých dělení vyplývá „z definice“,

Protože se jednotlivé možnosti velmi liší v příčinách, které k přerušení vedly, budou se velmi lišit prohledávané oblasti pro nalezení možných spojů. Následující podkapitoly se věnují úvahám nad optimálními prohledávanými oblastmi pro jednotlivé případy, neboli návrhem funkcí $penalty_S()$ a $penalty_C()$.

Funkce $penalty_S()$ má za úkol „ocenit“ každý krok, každý posun, do dalšího pixelu. Proto je v jejím případě kladen důraz na rychlost. Úkolem funkce $penalty_C()$ je „ocenit“ možného kandidáta na spojení — volá se vždy, když na nějakého při procházení prohledávané oblasti narazím. Protože to se stává jen několikát na jeden spoj, může být tato funkce, relativně k $penalty_S()$, pomalá.

Zbývá vyřešit otázku, který postup použít — jak určit, z jakého důvodu je značka přerušená, když momentálně „znám“ jen seznam l-atomů. Odpověď je jak z politické diskuse — na tuto otázku prostě neodpovím. Použiji všechny tři možné postupy a u každého nalezeného spoje si poznamenám, kterým jsem ho našel. Rozhodnutí, jakým způsobem k přerušení došlo a tedy kterou z možností zvolit nechám až na další krok napojování (kap. 5.4).

Na hledání napojení je dobré pohlížet jako na hledání možností pro další fázi Tyto možnosti jsou navíc ohodnoceny číslem udávajícím kvalitu spoje.

Předpokládané modely chování spojů v místech přerušení jsou rozebrány v následujících kapitolách. Záměrně jsou psány s neurčitými konstantami k_1 a k_2 (pro každý alg. různé) — pomocí těchto parametrů lze „ladit“ uvedené algoritmy pro konkrétní typ map. Hodnoty pro mapy určené pro orientační běh lze nalézt v souboru `penalty.c`. Pro jejich nalezení jsem použil metodu pokusů a omylů.

Ještě bych odkázal na kapitolu 6.1 popisující zvolený způsob počítání se směry neboť jej hodnotící funkce používají.

5.3.1 Křižovatky

Úkolem tohoto druhu spojů je zacelit rozdělení jinak souvislých čar, vzniklá v průběhu hledání l-atomů — křižovatky a „falešná“ rozdělení (kap. 5.2.2). Cílem je, aby napojovací algoritmus (kap. 5.4) znal všechny možnosti, které byly i v původním obraze, takže většinou dojde k propojení všeho se vším. Spočtená penalizace slouží k vzájemému porovnání. Omezení, které je na spoj kladeno je, že musí vést po pixelech, které algoritmus vyhledávající l-atomy označil za „křižovatky“ a že nesmí tvořit krátké smyčky (připojit se na svůj druhý konec). V reálných případech jsou spoje tohoto druhu velmi krátké (několik pixelů) a „levné“.

Funkce $penalty_S(x, p, \dots)$ zjednodušeně vypadá takto:

```
1 if (jsem na křižovatce)
2   penalty = k1 · současná délka cesty+
3     k2 · diff(směr kroku do p, směr kroku do s);
4 else
5     penalty = ∞;
6 fi
```

kde funkce $\mathrm{diff}()$ je definována na straně 19, p je pixel, z kterého zkoumám další cestu a x je zkoumaný soused p.

Funkce $penalty_C(x, p, \dots)$ vypadá takto:

```
1 if (připojuji se na svůj druhý konec a
2     délka l-atomu je moc krátká)
3   penalty = ∞;
4 else
5     penalty = k1 · diff(směr mého l-atomu,
6       směr připojovaného l-atomu);
7 fi
```

kde „můj“ l-atom je ten, který prodlužuji a „připojovaný“ je ten, který jsem právě nalezl.

5.3.2 Přetisky

Tyto spoje mají najít další pokračování linie, která byla přerušena z důvodu přetisku značkou jiné barvy, který se nepodařilo rozseparovat. Spolehlivým „trhačem“ ve všech vrstvách je přetisk s černou barvou (samozřejmě kromě černé vrstvy). Přerušení tohoto druhu mohou být velmi dlouhá, zvláště pokud se jedná o vrstevnice, na jejichž „kolize“ tohoto typu nebývá při tvorbě mapy prakticky brán zřetel (obr. 3.1).

Funkce $penalty_S(x, p, \dots)$ tedy musí „koukat, kam šlape“ — pokud je v černé vrstvě tmavý pixel, je krok levný, jinak drahý. Nutnost povolit „chůzi“ po pixelech, které nejsou v černé vrstvě tmavé plyne z toho, že ztenčováním vznikne mezi koncem l-atomu a skutečným koncem mezera několik pixelů (velikost této mezery se přibližně rovná tloušťce linie a její existence vyplývá z principu použitého algoritmu). Důležité pro spoje tohoto typu tedy není ani tak délka „v černé“, jako dodržet směr postupu.

Funkce $penalty_S(x, p, \dots)$:

```
1  if (diff(směr kroku do p, směr kroku do s) > 2
2        nebo směr mého l-atomu je neznámý)
3    penalty = ∞;
4  else
5      penalty = k1 ·
6        diff(směr mého l-atomu, směr cesty)^2;
7
8      penalty = penalty + k2;
9
10     if (pixel x v černé vrstvě je tmavý)
11       penalty = penalty/k3;
12     fi
13 fi
```

kde „směr mého l-atomu“ je směr l-atomu, který prodlužuhuji a „směr cesty“ je směr, který má současná část vytvořeného spoje. Vztah na řádku 6 je penalizace za změnu směru, kterážto je závislá na druhé mocnině odchylky od směru l-atomu. Na řádku 8 se pak přičítá penalizace za délku cesty.

5.3.3 Volný prostor

Spoje přes volný prostor mají za úkol najít spojení v místech, kde byla linie přerušena již autorem mapy (na tyto spoje můžeme pohlížet jako na „skutečné“ spoje, zatímco na ostatní jako na „opravárenské“). V této fázi rozpoznávání ovšem nemůžeme zjistit, jaké důvody autora k přerušení vedly, nevíme ani, jestli nejde třeba o skutečný konec značky. Důležité pro spoje tohoto typu je směr a délka. Právě délka spojů tohoto typu výrazně ovlivňuje celkový výsledek — příliš krátká nemusí nalézt všechny spoje, příliš dlouhá spojí i různé značky a navíc výrazně prodlouží výpočet (je třeba projít více pixelů). Penalizace za změnu směru má za cíl znevýhodnit spoje „do zatáčky“.

```
1  if (diff(směr kroku do p, směr kroku do s) > 2)
2    penalty = ∞;
3  else
4      penalty = k1 ·
5        diff(směr mého l-atomu, směr cesty)^2;
6
7      penalty = penalty + k2;
8
9      if (směr mého l-atomu je neznámý)
10       penalty = penalty + k3;
11     fi
12 fi
```

kde „směr mého l-atomu“ je směr l-atomu, který prodlužuhuji a „směr cesty“ je směr, který má současná část vytvořeného spoje.

Vztah na řádku 6 je penalizace za změnu směru, kterážto je závislá na druhé mocnině odchylky od směru l-atomu. Na řádku 8 se pak přičítá penalizace za délku cesty.

Na tomto místě je možná dobré zmínit práci s tečkovanou čarou. Z jednotlivých teček totiž nelze směr linie určit. Proto jsem aritmetiku směrů rozšířil o směr ∞, neboli „neznámý“, způsobem popsaným v kapitole 6.3. Funkce počítající směr l-atomu u těch, které jsou příliš krátké (což je případ teček) vrací právě směr ∞. Díky tomu lze napojení těchto krátkých l-atomů hledat stejnými algoritmy.

Uvedený algoritmus je velmi podobný algoritmu pro výpočet pro přetisk (kap. 5.3.2), jen povoluje prohledávání i l-atomů, „nemajících směr“ (slabší podmínka na řádku 1). Problémem bylo, že funkce diff() vrací pro rozdíl s neznámým směrem vždy nulu, což zvýhodňovalo spoje od takovýchto l-atomů. Tomu zabraňuje řádek 10.

5.4 Lepení

Tato kapitola popisuje postup nalezení liniových značek v seznamech l-atomů a možných napojení. Slovem *linie* budu v této kapitole nazývat datovou reprezentaci celé nebo jen části nějaké liniové značky. Linie je tvořena jednotlivými l-atomy a spoji.

Jeden z možných postupů je pokusit se v seznamech l-atomů a možných napojení nalézt všechny části jedné značky, vytvořit z nich linii a odstranit je ze seznamů, načež celý postup zopakovat. To dělat tak dlouho, dokud se mi ze zbytku podaří něco postavit.

Jiný možný postup je nechat jednotlivé části linií srůstat tak dlouho, dokud nevytvoří celé linie. Tento postup má oproti předchozímu jednu výhodu — v uzlech, kde nedokážu rozhodnout na základě l-atomů které se zde střetávají, možná dokážu rozhodnout, pokud budu znát celé linie, které se potkávají, takže pokud počkám, až linie srostou na druhých koncích, možná nakonec dokážu rozhodnout i původně neřešitelná místa.

Jádro spojovacího algoritmu, který vychází z druhé varianty je na obrázku 5.6 (vlastní algoritmus je trošku složitější než uvedený — hledání napojení je nutné dělat zvlášť pro jednotlivé konce linií, takže tělo cyklu je ve skutečnosti zdvojeno. Princip se však nijak nemění a o jeho zachycení mi šlo). Celý algoritmus provádím dokola tak dlouho, dokud dochází ke spojování.

Konečnost je zaručena ubývajícím počtem jednotlivých linií (při spojování), kterých je na začátku konečně mnoho.

```
1  foreach l in lines do
2
3    if (je_dobrá_linie(l, ...))
4      G1 = dobrá_napojení(l, ...);
5      g1 = vyber_nejlepší(G1);
6
7      if (g1 ≠ null)
8        s = soused(l, g);
9        G2 = dobrá napojení(s, ...);
10       g2 = vyber_nejlepší(G2);
11
12       if (g1 == g2)
13         spoj_linie(l, s, g1);
14       fi
15     fi
16   fi
17 od
```

Obrázek 5.6: Jádro algoritmu spojujícího linie

Význam jednotlivých proměnných:

lines seznam všech linií, počáteční nastavení je seznam všech l-atomů klasifikovaných jako součásti liniových značek, v průběhu algoritmu (při volání *spoj_linie()* se tento seznam mění,

G_1, G_2 seznamy možných napojení,

g_1, g_2 jedno (nebo žádné) konkrétní napojení,

s linie, ke které jsem se vybraným spojem připojil.

Funkce *je_dobrá_linie()*, *dobrá_napojení()* a *vyber_nejlepší()*, kterým budu říkat *hodnotící funkce napojení*, mají za úkol posoudit jednotlivé linie a napojení. Než popíšu jejich význam, zamysleme se nad jednotlivými typy čar (plná, čárkovaná, tečkovaná). Pravidla pro jejich přerušování jsou zcela odlišná a odlišné jsou i údaje, z kterých mohu v místech přerušení vycházet. Proto budou i hodnotící funkce napojení různé. Celý spojovací algoritmus tedy provedu zvlášť pro jednotlivé typy čar, vždy s jinou sadou funkcí. Jednotlivým variantám pro různé typy linií se věnují podkapitoly 5.4.1, 5.4.3 a 5.4.2.

Funkce *dobrá_napojení()* (řádek 4, 9) má za úkol vybrat ze všech možných napojení podmnožinu těch, která jsou pro konkrétní linii použitelná a funkce *vyber_nejlepší()* (řádek 5, 10) pak z něj vybere (nebo nevybere) vhodného kandidáta. Pokud funkce *vyber_nejlepší()* žádného kandidáta nevybere, může to být z důvodu špatných kandidátů (je na ně přísnější) nebo proto, že mezi nimi nedokáže rozhodnout (je více dobrých kandidátů).

Důvodem vzniku dvou funkcí místo jedné vracející rovnou vhodného kandidáta napojení je právě v druhém důvodu zamítnutí ve funkci *vyber_nejlepší()* — vhodnost kandidáta na spojení je závislá nejen na kandidátu samém, ale i na kandidátech ostatních — pokud je kandidát špatný, ale nemá žádnou „konkurenci," mohu jej použít. Pokud je kandidátů více, ale jeden je výrazně lepší než ostatní, mohu jej použít také. Pokud je však kandidátů více a jsou v podstatě stejně dobří, neumím rozhodnout a spojení neprovedu.

Funkce *je_dobrá_linie()* (řádek 3) má za úkol rozpoznat dobrý „základní kámen" pro linie daného typu. Pracuji totiž se seznamem linií, které mohou být součástí linie uplně jiného typu nebo vůbec nemusí tvořit nějakou liniovou značku (chyby v klasifikaci l-atomů).

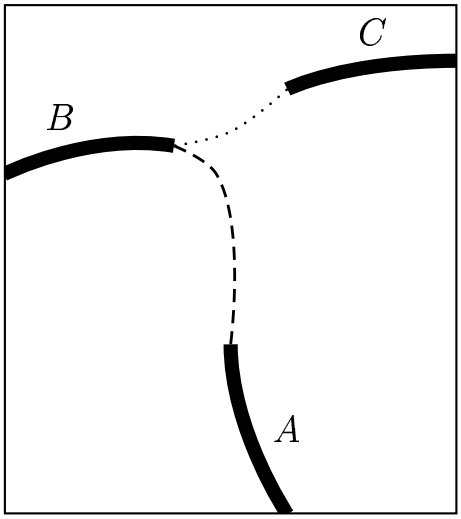
Jednotlivé části linie nemusí být tak jednoznačné, neboť jejich příslušnost k linii vyplynula z kontextu — byly vybrány funkcemi *dobrá_napojení()* a *vyber_nejlepší()*. Základní kámen však vybírám jen na základě jeho vlastností.[2]

Funkce *spoj_linie* (řádek 12) jen prací s datovými strukturami provede vlastní spojení.

Jak již bylo řečeno, konkrétní hodnotící funkce napojení jsou popsány v následujících podkapitolách. Zde bych se ještě pozastavil nad řádky 7–15 algoritmu 5.6. V podstatě říkají toto: dobré spojení je takové spojení, které se líbí oběma stranám. Důvodem pro tento krok jsou situace typu jako na obrázku 5.7 — pokud bych hledal napojení linie A, pravděpodobně bych nalezl linii B, ale pro tu je mnohem lepším partnerem linie C. V tomto případě by tedy při napojování A nebyla podmínka na řádku 12 splněna a ke spojení by nedošlo.

Toto, stejně jako závěr poznámky o hledání vhodného základního kamene jsou vedeny snahou napojit jen ta místa, v kterých jsem přesvědčen o správnosti a ostatní raději nechat nerozhodnuté. Buď se díky změnám v okolí situace vyjasní nebo bude muset rozhodnout operátor. Tento přístup vychází z úvahy v závěru kapitoly 4.2 o opravách chyb.

[2] není to tak uplně přesné — s funkcí je to jako s výběrem z intervalu, i z malého intervalu lze vybrat nekonečnou posloupnost, stejně tak funkce se může „nafouknout" do velké složitosti. Klíčové jsou ty tři tečky ve výčtu parametrů funkce — ty skrývají další informace, které může funkce použít. V implementované verzi se po vybrání kandidáta v případě sebemenší nejistoty pokusí tohoto pomocí postupu na řádcích 4–15 algoritmu 5.6 prodloužit a teprve pokud se prodloužení zdařilo, prohlásí jej za základní kámen.



Obrázek 5.7: Napojování linií

5.4.1 Plné linie

Plné (nepřerušované) čáry by měly být pokud možno nepřerušované a dlouhé. Dalším vodítkem při napojování tohoto typu linií je tloušťka — měla by být u obou spojovaných částí (skoro) stejná. Toto velice užitečné pravidlo bohužel neplatí pro tloušťku měnící značky.[3]

Funkce *je_dobrá_linie(l, ...)* vychází pouze z údaje o délce l-atomu. Pokud je délka větší než vhodně zvolená konstanta, prohlásím l-atom za dobrého kadidáta.

Funkce *dobrá_napojení(l, ...)* postupně prochází všechny možné spoje a testuje je. Její schématická podoba je následující:

```
1  foreach g ∈ { x | x je spoj vedoucí z l } do
2
3    if (|l.width − l2.width| > k1)
4      return FALSE;
5    fi
6
7    if (g.typ == „křižovatka“)
8      if (l.length > k2 & l2.length > k2 &
9          l.length + l2.length > k3)
10       return TRUE;
11     fi
12
13   elsif (g.typ == „přetisk“)
14     if (l.length > k4 & l2.length > k4 &
15         l.length + l2.length > k5)
16       return TRUE;
17     fi
18
19   elsif (g.typ == „volný prostor“)
20     if (l.length > k6 & l2.length > k6 &
21         l.length + l2.length > k7)
22       return TRUE;
23     fi
24   fi
26   return FALSE;
```

kde g je nějaký spoj z linie l, l_2 je linie, se kterou se pomocí g spojím. Doporučované vztahy konstant: $k_1 < 1$, $k_2 \in \langle 3, 10 \rangle$, ostatní $\langle 10, 100 \rangle$.

Tento algoritmu vychází z řečeného — v každém případě musí mít spojované linie přibližně stejnou tloušťku (toto pravidlo je v případě plných čar velmi užitečné) a musí být dostatečně dlouhé. Při spojování přes „křižovatku“ je k_2 je voleno výrazně menší než k_4 či k_6, tudíž je podmínka na minimální délku linií mnohem slabší. Slouží pouze k zabránění spojování s chybami separace — jednopixelovými trhlinkami v barevné separaci způsobenými většinou špatně separovatelným přetiskem. Zapříčiní vznik „plenence“ krátkých l-atomů v bezrostředním okolí. Tyto trhlinky lze sice programem pana Čejky korigovat, bohužel při této korekci dojde i k zvětšení značek a tím i vzniku slitků.

Funkce *vyber_nejlepší()* pracuje na principech, které již byly dříve zmíněny — pokud je odstup v kvalitě nejlepšího spoje před ostatními dostatečně velký nebo pokud nemá „soupeře“, vyberu tento spoj. Jinak zamítám všechny.

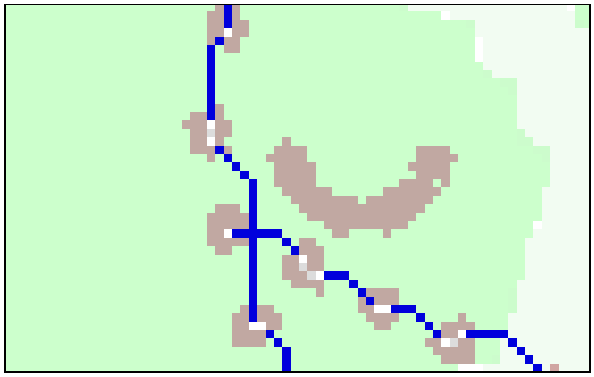
5.4.2 Tečkované linie

Základním rozlišovacím znamením pro tečkované čáry je délka l-atomu — musí být dostatečně malá. Protože tečky málokdy tvoří ideální kruh malého poloměru, je většinou výsledek ztenčování krátký l-atom, nikoliv l-atom délky 0, jak bylo pro zjednodušení v úvodu kapitoly 5.2 napsáno.

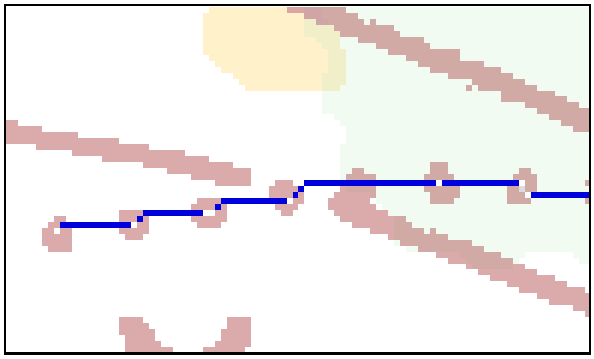
Části tečkovanou čarou kreslené čáry lze spojovat jenom spoji typu volný prostor. Spoje přes „křižovatku“ nedávají u tečkovaných čar smysl, spoje přes přetisk také ne, navíc občas spojujovaly nesouvisející linie.

Druhým vodítkem je směr. Jak již bylo řečeno, pokoušet se určit směr linie jen na základě jedné tečky nemá smysl. Pokud jich je ale spojeno více, směr linie

[3] největším problémem je značka „hluboká rýha“, neboť ta nejenom mění tloušťku, ale navíc ji mění mnohdy velmi rychle. Navíc bývá většinou krátká a často se kříží s vrstevnicemi. Rozpoznání této značky proto nedopadá moc dobře. Nepřišel jsem na způsob, jak se tomu vyhnout a přitom výrazně nezhoršit rozpoznávání ostatních linií.



Obrázek 5.8: Křížení tečkovaných čar — společnou tečku „ukořistila“ připojující se linie místo linie průběžné



Obrázek 5.9: Slitek v místě křížení plné a tečkované čáry

už určit lze. Výsledkem je, že spojování začne v nějakém přehledném místě a pak se šíří dál, což je správně.

Má to ovšem drobnou chybu — v místech spojení více takovýchto čar tečku tvořící křižovatku „ukořistí“ ten, kdo „přijde dřív“ (obr. 5.8)

Funkce *je_dobrá_linie()* tedy pouze otestuje délku a pokud je tato dost malá, l-atom schálí. Při implementaci jsem přišel ještě na jednu podmínku neschválení — tloušťku. Pokud je l-atom tlustý přibližně 1, jedná se o osamocenou tečku a ta asi žádné značce nepatří.

Funkce *dobrá_napojení()* vypadá takto:

```
1  foreach g ∈ { x | x je spoj vedoucí z l } do
2
3    if (l.width ~ 1 nebo l_2.width ~ 1
4          nebo g.typ <> „volný prostor“)
5      return FALSE;
6    fi
7
8    if (l.length ~ 1 & l_2.length ~ 1)
9      return TRUE;
10   fi
11
12   if (l.length > 1 & diff(směr l, směr g) > k_1)
13     return FALSE;
14   fi
15
16   if (l_2.length > 1 & diff(směr l_2, směr g) > k_1)
17     return FALSE;
18   fi
19
20   return TRUE;
```

Podmínka na ř. 8 říká, že pokud jsou obě dvě linie jen tečka, beru je vždy. V tomto případě nemá cenu uvažovat nad směry. V případě, že je linie delší (poslední dvě podmínky), zajímá mě i rozdíl směrů.

Nabízí se otázka, proč nepoužít také průměrnou délku mezery, neboť tečkované čáry jsou kresleny s pravidelnými rozestupy. Důvodem je situace jako na obr. 5.8 nebo na obr. 5.9, kdy se nějaká tečka „zatoulá“

Funkce *vyber_nejlepší()* je stejná jako u plné čáry.

5.4.3 Čárkované linie

Čárkované čáry by měly být pravidelné — mít přibližně stále stejný odstup i délku jednotlivých čárek. Navíc by pobobně jako u plných čar měla pomoci toušťka čáry (naštěstí neexistují tloušťku měnící čárkovanou čarou kreslené značky). Je třeba ovšem pamatovat na přerušení přes „křižovatku“, protože ta mohou rozdělit jednotlivé čárky na kratší části. Řešením je ustoupit v tomto případě od požadavku stejných délek. Ovšem za předpokladu, že spojované linie jsou „primitivní“, tj. zatím nespojené s jinými.

Funkce *je_dobrá_linie()* vychází z délky l-atomu. Pokud je tato v určitých mezích, l-atom schválím.

Funkce *dobrá_napojení()* vypadá takto:

```
1  foreach g ∈ { x | x je spoj vedoucí z l } do
2
3    if (|l.width − l_2.width| > k_1)
4      return FALSE;
5    fi
6
7    if (g.typ == „křižovatka“)
8      if (l.length > k_2 & l_2.length > k_2 &
9            l.length + l_2.length > k_3 &
10           l i l_2 jsou primitivní)
11       return TRUE;
12     fi
```

```
13
14  elsif (g.typ == „přetisk“)
15      if (l.length > k4 & l2.length > k4 &
16        l.avgB ≃ l2.avgB &
17        l.avgS ≃ l2.avgS ≃ g.length)
18        return TRUE;
19      fi
20
21  elsif (g.typ == „volný prostor“)
22      if (l.length > k5 & l2.length > k5 &
23        l.avgB ≃ l2.avgB &
24        l.avgS ≃ l2.avgS)
25        return TRUE;
26      fi
27  fi
```

kde *avgB* značí průměrnou délku čárek, z kterých je linie složená, *avgS* průměrnou délku spojů.

Při praktických pokusech jsem zjistil, že si plné a čárkované čáry „konkurují“ — o některých místech oba testy prohlásí, že je „jejich“. Řešením je nejprve provést vyhledávání čárkovaných čar a poté ty linie, které se skládají z malého počtu (1–5) dlouhých čárek prohásit za plnou čarou.[4]

6 Ostatní

V této kapitole bych se rád věnoval zajímavým problémkům řešeným v průběhu práce, na které nezbylo místo v ostatních kapitolách nebo mi připadají natolik zajímavé, abych je popsal důkladněji.

6.1 Směry a úhly

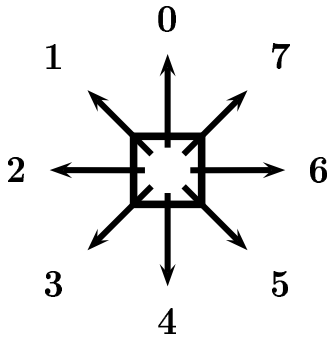
Pro uložení směru jsem použil jednoduché kódování — každému z osmi základních směrů jsem přiřadil číslo od nuly do sedmičky (obr. 6.1), směry mezi jsou reprezentovány čísly desetinými (při implementaci jsem konkrétně zvolil čísla s pevnou desetinou tečkou (fixed-point float) s přesností 2^{-11}, což umožňuje i s „necelými“ úhly pracovat stejně efektivně jako s „celými“).

K uložení úhlů jsem použil stejného kódování:

$$2\pi = 8$$

Tento způsob kódování byl již použit v kapitole 5.2.2 a vychází z něj číslování sousedů pixelu (kap. 5.1).

[4] tento postup ovšem do množiny plných čar „přihazuje“ linie vzniklé podle jiných pravidel. Abych udržel nízkou chybovost plných čar, musím být na čárkované přísný. Řešením by bylo vhodným způsobem sjednotit podmínky pro plné a čárkované čáry.



Obrázek 6.1: Základní směry

Takto zavedené směry umožňují efektivní implementaci základních operací: „vpravo vbok“ $(u + 6) \bmod 8$, „vlevo vbok“ $(u+2) \bmod 8$, „čelem vzad“ $(u+4) \bmod 8$. Úhel mezi dvěma směry:

$$\mathrm{diff}(u,v) = \begin{cases} |u-v| & \text{pro } |u-v| \leq 4, \\ 8-|u-v| & \text{jinak.} \end{cases}$$

Zde je dobré si všimnout, že takto zavedený rozdíl směrů $\mathrm{diff}(u,v) \leq 4$ pro lib. dva směry — beru vždy menší část kruhu směry vytknutou.

6.2 Směr jednoduché čáry

Při hledání napojení, ale i jinde, potřebuji znát směr konce l-atomu nebo linie. Nejjednoduším způsobem je prohlásit za směr směr posledního kroku. To ale nedává příliš dobré výsledky — kroky vedou vždy jen jedním ze základních směrů (celočíselné hodnoty) a dělení kruhu jen na osm dílků je příliš hrubé. Navíc l-atomy vznikají ztenčováním, takže směr posledního kroku mnohdy bývá úplně jiný, než skutečný směr linie. Bylo by tedy rozumné považovat za směr průměr směrů několika posledních kroků nebo je proložit hladkou křivkou a její směr použít.

Výpočet směru linie musí být rychlý, navíc by bylo šikovné, kdyby byl inkrementální, tj. pokud prodloužím l-atom o jeden krok, mělo by jít spočítat směr nové linie jen pokud možno ze směru původního a směru posledního kroku.

Jako řešení jsem zvolil vážený průměr několika posledních kroků, kdy největší váhu má směr poslední a důležitost ostatních exponenciálně klesá se vzdáleností. Tuto vlastnost současně s inkremenální vlastností má metoda označovaná jako *EWMA* (*exponentially*

weighted moving average) [6], která počítá novou hodnotu jako:

$$d_n = d_{n-1} \cdot q + s \cdot (1 - q) \qquad (*)$$

kde d_n je nová hodnota, d_{n-1} je stará hodnota, s je směr posledního kroku a q vhodná konstanta z intervalu $(0, 1)$.

Tento vztah ale nefunguje pro směry korektně: musím vyřešit přechod mezi nulou a sedmičkou (pokud je např. starý směr okolo jedničky a poslední krok má směr 7, mělo by být výsledkem číslo okolo nuly, nikoliv okolo trojky). Proto je potřeba s ještě před započítáním upravit takto:

```
1 if (s < d_{n-1})
2   if (d_{n-1} - s > 4)
3     s = s + 8;
4   fi
5 else
6     if (s - d_{n-1} > 4)
7       s = s - 8;
8     fi
9 fi
```

Připojením rovnice $(*)$ za tuto úpravu dostaneme funkci $\mathrm{dcalc}(d, s, q)$, která z parametrů vypočte nový směr.

6.3 Neznámý směr

Směr l-atomu určuji z posledních několika kroků l-atomu pomocí funkce $\mathrm{dcalc}()$. U krátkých l-atomů (kam patří např. l-atomy vzniklé z teček) tento postup nelze použít, navíc by výsledek stejně postrádel smysl (směr tečkované čáry nemohu určit z jedné tečky). Proto jsem zavedl konstantu „neznámý směr“ — směr ∞ (shodou okolností je tato konstanta definována jako číslo 8, což hezky koresponduje se zvoleným označením). Výše uvedené funkce jsem dodefinoval takto (x značí libovolný směr různý od ∞):

$$\begin{aligned} \mathrm{diff}(\infty, \infty) &= \text{error} \\ \mathrm{diff}(\infty, x) &= \mathrm{diff}(x, \infty) = 0 \\ \mathrm{dcalc}(x, \infty, q) &= \text{error} \\ \mathrm{dcalc}(\infty, x, q) &= x \end{aligned}$$

6.4 Pokusy o detekci nepravidlených rastrů

Protože rastry, ať už bodové nebo liniové, které jsou součástí plošných značek, při rozpoznávání značek liniových velmi „zlobí“, pokusil jsem se implementovat jednoduchý algoritmus pro jejich detekci. Protože jsem při tomto pokusu získal pár postřehů, které mohou být využity při návrhu algoritmu rozpoznávající plošné značky (včetně rastrů), rozhodl jsem se tomuto pokusu věnovat pár řádků.

Cílem je rozpoznat bodové nepravidelné rastry (např. značka „rozbitý povrch“) a pracuje v místě klasifikace l-atomů (kap. 5.2.3). Tento postup je pro pravidelné rastry nevhodný, možný způsob detekce těchto rastrů je naznačet například v [2].

Algoritmus, který jsem použil rozdělí okolí každého „krátkého“ l-atomu na osm sektorů a spočte počet „krátkých“ l-atomů v jednotlivých sektorech. Pokud je v různých směrech nalezeno dostatek sousedů, označím l-atom za součást rastru. V druhém kroku projdu všechny takovéto l-atomy a ty, které nemají dostatek „sousedů-rastrů“ opět odznačím (odstraním tak chyby např. v místech spojování tečkovaných čar). Nakonec ještě jednou projdu neoznačené „krátké“ l-atomy a znovu prohlédnu jejich okolí. Pokud najdu dostatek sousedů označených jako rastr (teď mám ale menší požadavky na počet a všesměrovost), prohlásím jej také za rastr. Poslední krok opakuji tak dlouho, dokud se něco mění.

Domníval jsem se, že algoritmus na tomto principu bude stačit, ale není tomu tak. Liniové značky, které vedou dostatečně blízko sebe jsou také prohlášeny za rastr. Zpřísnění pravidel zase vede k nenalezení některých částí rastru. Proto se tato klasifikace standartně v knihovně ML neprovádí. [1]

Jedna z možností je předsunout detekci plošných značek před detekci linií, obávám se však, že tato pak bude mít podobné problémy s liniovými značkami.

7 Implementace

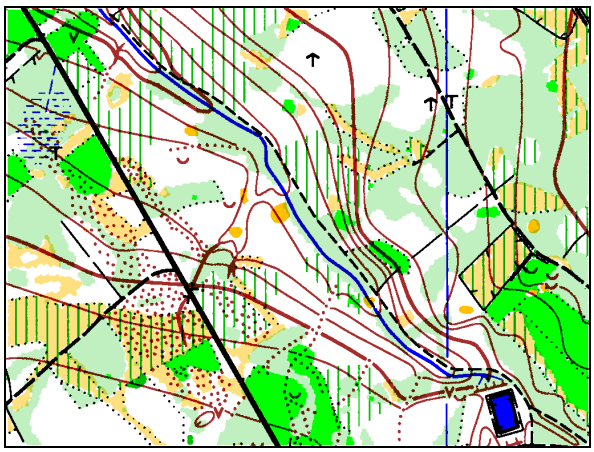
Předchozí kapitoly se nesly v teoretické rovině. Součástí této práze je i praktická implementace popsaných algoritmů — knihovna ML. Jejím výsledkům a popisu knihovny s testovacími aplikacemi bych se rád věnoval zde.

[1] vyzkoušet si vlastnosti tohoto algoritmu je možné pomocí programu `raster` — (kap. 7.3) nebo překladem knihovny s parametrem `-DCLASSIFY_RASTER`

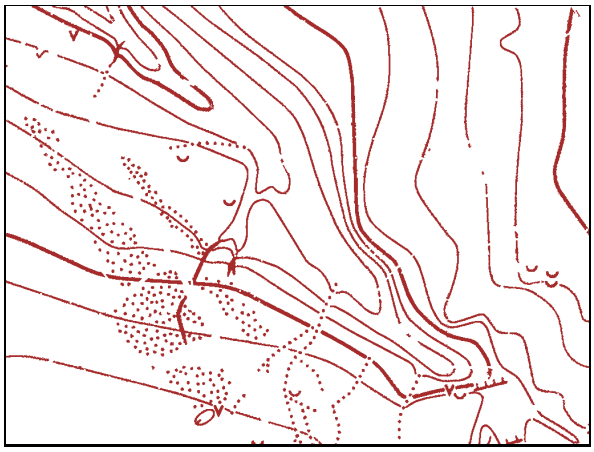
7.1 Příklady

Tato kapitola má za úkol ukázat praktické výsledky implementované podoby popsaných algotritmů a ukázat problematická místa. Obrázky zde uváděné vznikly za pomoci testovacích aplikací (kap. 7.3) a byly ručně upraveny pomocí programu GIMP (změny barev, skládání obrázků, výřezy; obrázky nebyly nijak retušovány nebo dokreslovány).

Základní testovací obrazem byl stejný výřez, jaký použil pan Čejka k předvedení separace barev (obr. 7.1) neboť je poměrně komplikovaný a tudíž obsahuje mnoho míst vhodných k demonstraci problémů. Výsledky ukážu na tmavě hnědé vrstvě — obrázek 7.2 (vstevnice, rýhy, jámy, . . .).

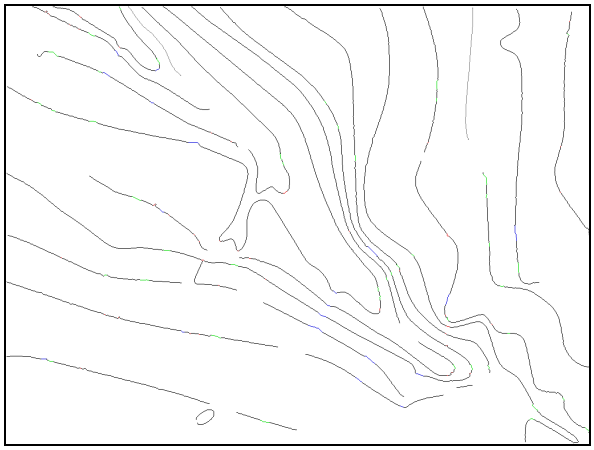


Obrázek 7.1: Testovancí výřez mapy

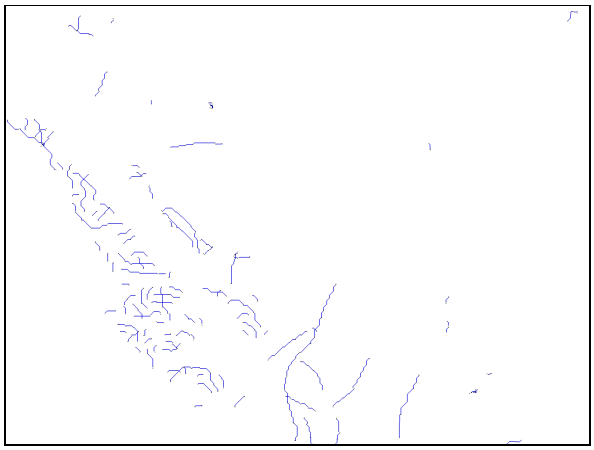


Obrázek 7.2: Tmavě hnědá vrstva

Celkové výsledky rozpoznávání čar jsou na obrázku 7.3 (plné čáry) a 7.4 (tečkované čáry)[1]. Význam jednotlivých barev je následující: černé jsou l-atomy samotné, červené jsou spoje typu „křižovatky“, zelené jsou spoje typu „přetisk“ a modré jsou spoje typu „volný prostor“. Na celkových obrázcích toto barevné rozlišení není příliš patrné, v jednotlivých konkrétních případech už ano.



Obrázek 7.3: Rozpoznané čáry — dlouhé



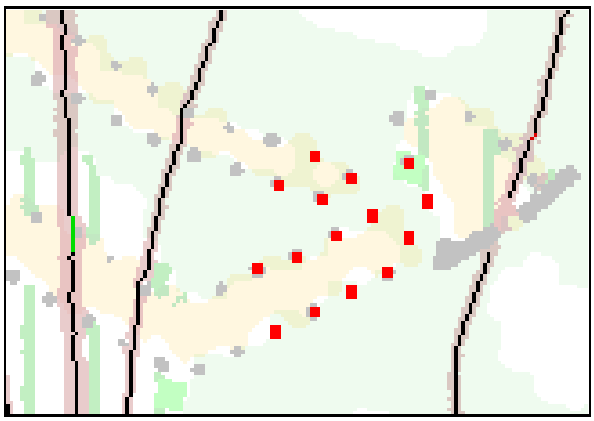
Obrázek 7.4: Rozpoznané čáry — tečkované

Na první pohled je viditelná chyba v levé střední části tečkovaných čar — nebyla provedena klasifikace l-atomů a tak byla plošná značka „rozbitý povrch“ chybě klasifikována jako skupina liniových značek.

Provedení klasifikace, tak je popsána v kapitole 6.4, tento problém odstraní, neboť velkou část této plošné značky nalezne. Důvodem, proč je standartně klasifikace vypnuta, lze najít v černé vrstvě — obrázek 7.5.

[1] čárkované čáry se v hnědé vrstvě nevyskytují

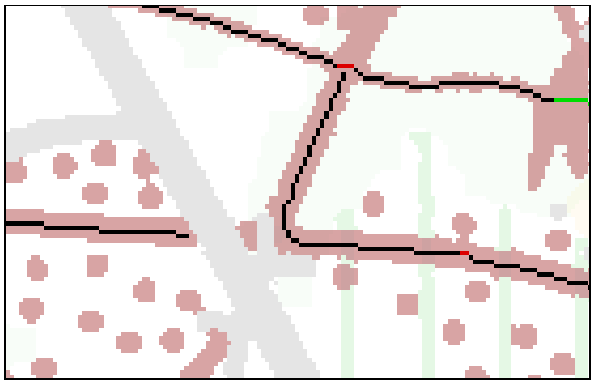
Zde bylo tečkovanou kreslené zvýraznění okraje plošné značky chybně detekováno jako rastr.



Obrázek 7.5: Chybá detekce rastru

Příkladem, kdy chyba vznikla již při hledání l-atomů je obrázek 7.6. V tomto místě mapy dochází ke křížení tří značek — hnědou čarou kreslená vrstevnice, taktéž hnědou barvou kreslená rýha a černě kreslená zpevněná cesta s mostkem (pravděpodobně přes tuto rýhu). Toto místo nelze popsaným postupem správně rozpoznat, neboť vrstevnice i rýha splynuli v jeden l-atom a ten je z pohledu dalšího zpracování nedělitelný.

Tento příklad je ukázkou toho, že bez pomoci operátora se rozpoznávání pravděpodobně neobejde, neboť toto místo ač pro člověka nikterak složité je pro navržený algoritmus zcela neřešitelné[2].



Obrázek 7.6: Chybná detekce l-atomu

[2]pravděpodobnost, že i po rozšíření algoritmu aby dokázal takováto místa řešit, se najde jiné, jinak zapeklité, je podle mého odhadu blízká jedné ☺

7.2 Knihovna MAPLINE

Knihovnu MAPLINE (dále jen ML) jsem se snažil napsat v maximální možné míře nezávisle na platformě. Je psána v ANSI C a používá (kromě libC) jen knihovnu GLib (tato knihovna je v současnosti na mnoha platformách a obsahuje jednak definice typů s pevnou bitovou délkou nezávisle na platformě a dále pak velmi efektivní implementace práce se základními datovými strukturami jako je obousměrně zřetězené seznamy či hašovací tabulky). Knihovna ML-TOOLS obsahuje další funkce, které jsou šikovné ale nejsou nezbytné a potřebují další knihovny (libPNG). Součástí zdrojových textů jsou i „make-files“ pro přeložení a slinkování knihoven a testovacích programů pod operačním systémem Linux.

Hlavičkový soubor knihovny ML se jmenuje `ml.h`, knihovny ML-TOOLS pak `mlt.h`.

Všechny funkce z obou knihoven začínají předponou `ml_`, definice `ML_` a typy `ML`. U funkcí další část jména udává hlavní datovou strukturu, se kterou pracujou.

Domnívám se, že zde není vhodné místo k podrobnému popisu jednotlivých funkcí. Jejich význam je myslím snadno pochopitelný z názvu funkce a použití v testovacích programech. Popíši jen nejdůležitější datové struktury.

Jednotlivé l-atomy jsou reprezentovány strukturou *MLLineAtom*. Protože v mnoha případech pracuji s celou množinou l-atomů, existuje struktura *MLAtoms*. Ta kromě seznamu všech prvků obsahuje hašovací tabulku odkazů na prvky, kde klíčem je souřadnice koncového pixelu. Z toho plyne omezení, že na jednom místě nemohou začínat (končit) dva l-atomy. Ve skutečnosti však platí silnější tvrzení, že každý pixel obrazu náleží maximálně jednomu l-atomu.

Spoje mezi l-atomy jsou reprezentovány také strukturou *MLLineAtom*, množina spojů má ovšem vlastní strukturu *MLGlue*. Důvodem k jiné struktuře je, že pro spoje již pravidlo o jediném „obyvateli“ pixelu neplatí. *MLGlue* obsahuje také hashovací tabulku se stejným klíčem jako *MLAtoms* (souřadnice koncových pixelů), ale prvky jsou seznamy spojů začínajících v daném místě.

Struktura pro uložení linií se jmenuje *MLLine*. Obsahuje statistické údaje o linii a seznam ukazatelů na jednotlivé l-atomy a spoje, které ji tvoří. Pro snadnou práci s liniemi a jednoduší práci s alokací paměti jednotlivé l-atomy a spoje „vlastní“ struktury *MLAtoms* a *MLGlue* a linie mají jen odkazy na položky těchto množin (tímto

způsobem pracují funkce pro práci s uvedenými strukturami).

Posledními dvě důležité struktury jsou *MLPictureSep* a *MLPictureUI8*. Slouží k uložení rastrové podoby mapy nebo jiných údajů potřebných v průběhu výpočtu. Různí se pouze v datovém typu pro uložení jednoho pixelu — *MLPictureSep* slouží pro načtení a uložení celé mapy. Proto musí mít datový typ pro uložení každého pixelu 16 bitů. Ve většině případů však vystačím s menším počtem bitů a tak v *MLPictureUI8* je na pixel jen 8 bitů.

Další informace je možné vyčíst přímo ze zdrojových souborů. V adresáři `lib` jsou zdrojové texty knihovny ML, v adresáři `tools` zdrojové texty knihovny ML-TOOLS a adresář `misc` obsahuje jednotlivé testovací programy. Soubory `atoms.c`, `glue.c`, `line.c`, `lineatom.c`, `lines.c` a `picture.c` obsahují funkce pro práci se stejnojmenými datovými strukturami.

V souboru `direct.c` jsou funkce pro počítání se směry, ovšem valná většina těchto funkcí je i v `ml.h`. Funkce v souboru `lfind.c` slouží k nalezení l-atomů, v `lally.c` k nalezení spojů a vlastní spojování je v `lconn.c`. Hodnotící funkce pro shledání spojení jsou v `penalty.c` a pro spojování v `connect.c`.

7.3 Demonstrační programy

Součástí instalace jsou jednoduché demonstrační programy. Tyto programy si parametry vezmou z příkazové řádky, na standartní výstup vypisují průběh výpočtu (záleží na parametrech překladu) a vytvoří soubor `<jméno>.png` (`<jméno>` = jméno programu), v kterém je výsledek. Jedinou vyjímku tvoří program `lines`, který jich vytvoří víc — každý pro jeden druh čar (`lines_long.png`, `lines_dash.png` a `lines_dot.png`).

Cílem při tvorbě těchto programů nebyla uživatelská přítulnost, ale demonstrace funkčnosti knihovny ML.

Všechny programy mají poslední parametr nepovinný. Pokud je uveden, považuje se za jméno PCL souboru, pokud uveden není, čte se PCL soubor ze standartního vstupu. PCL je formát používaný programem pana Čejky [2] pro uložení separované mapy.

Pod pojmem „zobrazí výsledek ... “ je myšleno „uloží obrázek, který obsahuje výsledek ... “.

thin zobrazí výsledek ztenčování. Program se spouští příkazem

```
thin <mask> [pclfile]
```

kde `mask` je číslo vrstvy, se kterou se má pracovat a `pclfile` je nepovinný parametr. Spuštěn bez parametrů vypíše jednoduchou nápovědu.

latoms zobrazí jednotlivé nalezené a klasifikované l-atomy. Bílou jsou značeny normální liniové l-atomy, modře l-atomy patřící plošným značkám a červeně případné detekce rastrů (záleží na parametrech překladu knihovny). Program se spouští příkazem

```
latoms <mask> [pclfile]
```

kde `mask` je číslo vrstvy, se kterou se má pracovat a `pclfile` je nepovinný parametr. Spuštěn bez parametrů vypíše jednoduchou nápovědu.

glue zobrazí jednotlivé možné spoje. Modře jsou kresleny spoje přes „volný prostor“, zeleně „přetisk“ a červeně přes „křižovatku“. Program se spouští příkazem

```
glue <mask> #glue [pc-
lfile]
```

kde `mask` je číslo vrstvy, se kterou se má pracovat, `#glue` určuje, kolikátý nejlepší spoj se má pro každý l-atom zobrazit (0–nejlepší, 1–druhý nejlepší, ... , pokud u některého l-atomu už tolik možností není, nezobrazí u něj nic) a `pclfile` je nepovinný parametr. Spuštěn bez parametrů vypíše jednoduchou nápovědu.

area zobrazí prohledávané oblasti. Barevné rozlišení je stejné jako u programu `glue`. Program se spouští příkazem

```
area <mask> [pclfile]
```

kde `mask` je číslo vrstvy, se kterou se má pracovat a `pclfile` je nepovinný parametr. Spuštěn bez parametrů vypíše jednoduchou nápovědu.

lines zobrazí nalezené linie. Program se spouští příkazem

```
lines <mask> [pclfile]
```

kde `mask` je číslo vrstvy, se kterou se má pracovat a `pclfile` je nepovinný parametr. Spuštěn bez parametrů vypíše jednoduchou nápovědu.

pcl2png převede mapu ve formátu PCL do barevného obrázku ve formátu PNG. Program se spouští příkazem

```
pcl2png basefilename
```

kde `basefilename` je jméno PCL souboru bez přípony .pcl.

8 Závěr

Tato diplomová práce ukazuje postup, jak v digitalizované a na jednotlivé tiskové barvy separované mapě určené pro orientační běh nalézt liniové značky. Uvedené algoritmy umožňují vyhledat v obraze jednotlivé části liniových značek a poskládat z nich značky celé. Hlavní důraz je kladen na správnost napojení. V místech, kde rozhodnout nedokáží, jsou schopny poskytnout možné varianty.

První část práce je věnována rozboru celého úkolu rozpoznávání mapy, popisu jednotlivých skupin topografických značek používaných v mapách pro orientační běh a možnými postupy jejich rozpoznávání. Těžiště práce je pak v části druhé, která se podrobně věnuje značkám liniovým. Proces nalezení značek je rozdělena do tří fází — nalezení jednotlivých částí linií, nalezení možných napojení a výběrem vhodného spoje. Základem první fáze je publikovaný algoritmus, který jsem opravil a rozšířil. Analýza a řešení druhé a třetí fáze je kompletně mým dílem. Při řešení druhé fáze — hledání možných napojení — jsem nejprve rozdělil důvody přerušení značek, navrhl základní algoritmus a nakonec navrhl na základě úvah a analýzy reálných případů hodnotící funkce pro jednotlivé příčiny přerušení. Podobně jsem postupoval při řešení fáze třetí — spojování částí značek v značku jednu — rodělil jsem linie podle typu čáry do několika skupin, navrhl základní algoritmus a vytvořil hodnotící funkce pro jednotlivé typy.

První dvě části jsou obecně použitelné při vektorizaci, třetí je speciálně zaměřena na mapy pro orientační běh.

Závěr práce je věnován základnímu popisu knihovny, testovacích programů a rozboru výsledků.

Součástí práce je implementace knihovny funkcí realizující jednotlivé popsané kroky převodu, a sada jednoduchých programů demonstrující funkčnost algoritmů a použití knihovny.

Hlavní přínos celé práce je podle mého názoru v návrhu a implementaci konkrétních algoritmů pro řešení problému nalezení liniových značek v mapách, o jehož řešení jsem marně hledal nějakou zmínku v literatuře.

Kam zaměřit další úsilí, otevřené otázky[1]:

- prozkoumat použitelnost algoritmu i na značky kreslené komplikovanějším typem čáry (např. elektrické vedení). Současná verze nalezne základní linii a „čárky“ po stranách, takže rozpoznání by mělo být možné, otázkou je, zda by nebylo lepší označit takovéto liniové značky za další typ čáry a navrhnout pro ně vlastní hodnotící funkce (jako pro plné, čárkované a tečkované čáry). Tento postup by mohl být zvláště úspěšný u značky „schůdný/neschůdný skalnatý sráz“, která se většinou vyskytuje v místech velké koncentrace jiných linií (vrstevnic). Tato místa v současné verzi většinou zůstanou nespojena.

- současná implementace hodnotících funkcí pracuje jen s dvěma stavy — spojit/nespojit. Regulace „míry spolehlivosti“ je možné jen nastavením prahu uvnitř těchto funkcí (pozor, to neznamená, že nemohu při další práci s liniemi zjistit kvalitu spoje! Mohu, co ale nemohu, je zjisit, na jaké „hladině spolehlivosti“ bych již spoj zamítl nebo naopak našel. Jediná možnost spustit lepení l-atomů znovu s jinými parametry). Tím ztrácím jeden „rozměr“ řešení, jehož studium by mohlo v dalších částech aplikace pomoci.

- implementovat další funkce, které by usnadnily volbu jiných možných spojení — současná implementace toto umožňuje, ale poměrně nepohodlně (postupné procházení více datových struktur, vytváření pomocných seznamů apod.)

- zřejmě bude nutné vytvořit editor k zadávání oprav. Zajímavou myšlenkou by bylo spojit tento editor s hodnotícími funkcemi napojení. Jedno korektní napojení může vyjasnit několik dalších — pokud znám delší úsek linie, mohu lépe rozhodnout v místech stýkání těchto linií s ostatními (např. znám přesněji poměr délek čar a mezer u čárkovaných čar). Díky tomu, že vlastní fáze napojování (kap. 5.4) je poměrně rychlá, mohl by operátor přímo řešit problematická místa v jejím průběhu.

- domnívám se, že navržené algoritmy jsou mnohem obecnější a umožňují řešit úkol vektorizace i v jiných oblastech, jako jsou např. technické výkresy.

[1] pokus o zodpovězení jedné otázky plodí většinou další, ještě zajímavější

Bylo by sice asi nutné upravit hodnotící funkce, domnívám se ale, že nikoliv zásadně. Tuto oblast použití knihovny by bylo vhodné prozkoumat.

Index

Literatura

[1] Mjr. Ing. Karel Brázdil. *Rozvoj postupů a metod automatizované tvorby topografických map měřítka* 1 : 25 000. kandidátská disertační práce, Brno, 1993.

[2] Radek Čejka. Optické rozpoznávání objektů na mapách pro orientační běh. diplomová práce, Universita Karlova, Matematicko-fyzikální fakulta, 1999.

[3] Český úřad zeměměřický a katastrální. *SEZNAM MAPOVÝCH ZNAČEK Základní mapy ČR 1:10 000*, 1993.

[4] Český úřad zeměměřický a katastrální. *SEZNAM MAPOVÝCH ZNAČEK Základní mapy ČR 1:25 000*, 1993.

[5] Anil K. Jain. *Fundamentals of Digital Image Processing.* Prentice-Hall International, Inc, 1989.

[6] J. M. Lucas and M. S. Saccucci. Exponentially weighted moving average control schemes: Properties and enhancement. *Technometrics*, 32(1), 1990.

[7] Ing. Jiří Urban. *Projekt II — Práce s grafickou informací Digitální model terénu.* České vysoké učení technické v Praze, 1988.

[8] Karel Voříšek. Prototyp vektorizačního nástroje využívající koutkovou reprezentaci obrazu. diplomová práce, Universita Karlova, Matematicko-fyzikální fakulta, 2000.