A05450 Martin Lipinský
ZCU ZS2006 martin@lipinsky.cz

Semestální práce z předmětu PC

„Přebarvování souvislých oblastí“

1. Zadání

Naprogramujte v ANSI C přenositelnou konzolovou aplikaci, která provede v binárním digitálním obrazu obarvení souvislých oblastí pomocí níže uvedeného algoritmu. Vaším úkolem je tedy implementace tohoto algoritmu a funkcí rozhraní (tj. načítání a ukládání obrázku, apod.). Program se bude spouštět příkazem ccl.exe <inpf><outf>. Symbol <inpf> zastupuje úplnou specifikaci vstupního souboru s binárním obrazem, symbol <outf> úplnou specifikaci výstupního souboru, který vytvoří Vaše aplikace. Takže Váš program může být během testování spuštěn například takto:

```
ccl.exe e:\images\img-inp-01.pgm e:\images\img-res-01.pgm
```

Váš program nechť vytvoří výsledný soubor s obarveným obrazem v uvedeném umístění a s uvedeným jménem. Vstupní i výstupní obraz bude uložen v souboru ve formátu PGM, který je popsán níže. Obarvení proveďte podle níže popsaného algoritmu. Testujte, zda je vstupní obraz skutečně černobílý. Musí obsahovat pouze pixely s hodnotou 0x00 a 0xFF. Pokud tomu tak není, vypište chybové hlášení a oznamte chybu operačnímu prostředí pomocí nenulového návratového kódu. Hotovou práci odevzdejte v jediném archivu ZIP, pojmenovaném Vaším osobním číslem, tj. např. A03487.zip. Archiv nechť obsahuje všechny zdrojové soubory potřebné k přeložení programu, makefile a dokumentaci ve formátu PDF nebo PostScript. Bude-li některá z částí chybět, kontrolní skript Vaši práci odmítne.

Algoritmus:

Přebarvování souvislých oblastí (Connected Component Labelling) je dvouprůchodový algoritmus z oblasti počítačového vidění. Pracuje tak, jak je uvedeno v následujícím popisu:

1. průchod: Procházejte obraz po řádcích. Každému nenulovému pixelu (tj. pixelu představujícímu objekt) na souřadnicích [i, j] přiřaďte hodnotu podle hodnoty jeho sousedních pixelů, pokud nenulové sousední pixely existují (zkoumáme pixely dané maskou: [i-1,j-1],[i-1,j][i-1,j+1][i,j-1]) . Všechny sousední pixely dané maskou již byly obarveny v předchozích krocích.

- Jsou-li všechny sousední pixely součástí pozadí (mají hodnotu 0x00), přiřaďte pixelu [i, j] dosud nepřidělenou hodnotu – novou barvu.

- Má-li právě jeden ze sousedních pixelů nenulovou hodnotu, přiřaďte obarvovanému pixelu hodnotu nenulového sousedního pixelu.

- Je-li více sousedních pixelů nenulových, přiřaďte obarvovanému pixelu hodnotu kteréhokoli nenulového pixelu ze zkoumaného okolí. Pokud byly hodnoty pixelů v

masce různé (došlo k tzv. kolizi barev), zaznamenejte ekvivalenci dvojic hodnot do zvláštní datové struktury - tabulky ekvivalence barev.

2. průchod: Po průchodu celého obrazu jsou všechny pixely náležející oblastem (objektům) obarveny, některé oblasti jsou však obarveny více barvami (díky kolizím). Proto projděte znovu celý obraz po řádcích a podle informací o ekvivalenci barev přebarvěte pixely těchto oblastí. Po tomto kroku odpovídá každé oblasti označení (obarvení) jedinou, v jiné oblasti se nevyskytující hodnotou (barvou).

Vstupní a výstupní formát:

Na vstupu i výstupu Vašeho programu budiž obrázek v souboru ve formátu PGM. Formát tohoto souboru je následující:

P5	znaky 'P' a '5' a znak nový řádek (LF, ASCII 0x0A)
1024 768	počet sloupců, mezera, počet řádek (zapsaný řetězcem znaků)
255	index nejvyšší hodnoty šedé, resp. úrovně jasu (řetězcem znaků)
ČČČuu$$AAAA. . .	byty představující hodnoty jednotlivých pixelů

Každý pixel je v souboru uložen jako jeden byte (hodnota pixelu je tedy vlastně úroveň šedé). Byte s hodnotou 0x00 znamená úplně černý pixel, zatímco byte s hodnotou 0xFF znamená úplně bílý pixel.

2. Analýza problému

Zadaná úloha se skládá z několika různých problémů. Jednak jde načtení a „rozebrání" vstupního souboru s obrázkem, pak vlastní detekce a obarvení souvislých ploch, a nakonec uložení upraveného obrázku zpět na pevný disk. Vlastní práce s obrázkem je pak dále možno rozdělit na první poměrně přímočarý průchod a oindexování obrázku, zjištění kolizí barev po prvním průchodu a jejich zaznamenání, a závěrečné přeindexování obrázku tak, aby každá jedna plocha skutečně odpovídala jedné barvě.

Pro zaznamenání kolizí mezi barvami byla vybrána matice sousednosti (jednoduché dvourozměrné pole) a jednoduché operace nad ní (více níže) a její následné přepsání do reindexovacího jednorozměrného pole. Matice sousednosti je sice relativně paměťově náročná, zvláště u větších obrázků s velkým počtem indexů, oproti tomu je však proti jiným strukturám (seznam sousednosti) velmi snadno a přímočaře implementovatelná.

3. Popis implementace

Program je rozdělen do celkem čtyř částí. `ccl.c` obsahuje kontrolu vstupních parametrů a jednoduché volání dalších funkcí. Zdrojový soubor `fio.c` obsahuje funkce na načtení a oparsování vstupního souboru, a zapsání výstupního souboru. Hlavní výkonná část programu zajišťující všechny úkony detekce a obarvování obrázku je obsažena ve zdrojovém souboru `color.c` a všechna chybová hlášení včetně potřebných funkcí jsou obsaženy v souboru `errors.c`. Zajímavé části jsou detajlněji popsané dále.

3.1 První obarvovací průchod

První průchod je lineární a velmi jednoduchý. Celý obrázek je procházený bod po bodu, a v zřetězené podmínce se zkoumá zda kterýkoli z jeho předchozích sousedů

má již obarvený bod. Pokud se takový najde, dostane aktuálně zkoumaný bod jeho barvu, pokud ani jeden ze sousedů barvu nemá, přidělí se zkoumanému bodu nová barva.

3.3 Naplnění a alokace matice sousednosti

Po prvním průchodu vznikne obrázek kde spousta ploch je obarvena více barvami (kolize). Ty je nutno detekovat a připravit přeindexovací pole aby bylo možno finálně přebarvit obrázek. Z různých mořností implementace uložení kolizí (matice sousednosti, BVS, seznam sousednosti) byla vybrána matice sousednosti. Je sice více paměťově náročná, její implementace je však velmi jednoduchá a průhledná. Po prvním průchodu je znám počet celkově přidělených indexů. Je dynamicky alokovaná matice velikosti početIndexů^2 a celý obrázek se znovu projde. Přitom se zkoumá zda aktuální bod má stejnou barvu, pokud ne, do matice se zapíše jednička na souřadnice [barva_aktBodu, barva_kolSouseda] i na souřadnice [barva_kolSouseda, barva aktBodu]. Minimalní počet zkoumaných sousedů se jeví v tomto průchodu 3 (přímého předchůdce je možno vynechat, protože ten je použit jako první v případě že existuje při obarvování a tudíž v tomto směru nemůže kolize nastat).

3.4 Vytvoření přeindexovacího pole

Aby bylo možno obrázek finálně přebarvit, je potřeba z matice sousednosti vygenerovat přepisovací pole který index odpovídá které barvě. Je naalokována paměť pro přeindexovávací pole délky početIndexů. Matice sousednosti získaná v předchozím kroku je doplněna o jedničky na diagonále a ve druhém kroku projitá „Floyd Marshallovym algoritmem“. Jeho podrobný popis je připojený ve speciálním souboru `fast_ccl.pdf` (anglicky).

```
/* Floyd-Marshall algoritmh*/
for (j=1; j<actualIndex; j++) {
  for (i=1; i<actualIndex; i++) {
    if (matrix[((actualIndex)*i)+j+1]) {
      for (k=1; k<actualIndex; k++) {
        if ((matrix[(actualIndex)*i+k+1]) || (matrix[(actualIndex)*j+k+1])) {
          matrix[(actualIndex)*i+k+1]=1;
        }
        else {
          matrix[(actualIndex)*i+k+1]=0;
        }
      }
    }
  }
}
```

Tím jsou sousedící barvy snadno naplnitelné do reindexovacího pole.

```
/* naplneni reindexovaciho pole z upravene matice sousednosti */
for (i=1; i<actualIndex; i++) {
  for (j=1; j<actualIndex; j++) {
    if (matrix[((actualIndex)*i)+j+1]==1) {
      reindex[j]=actualColor;
      pom++;
      clearCoulumn(j);
    }
  }
  if (pom) {
    /* novou barvu davame jen pokud se behem pruchodu radku alespon jednou resilo */
    actualColor++;
    pom=0;
  }
}
```

Paměťová náročnost (matice sousednosti) by šla částečně řešit implementací mapovací funkce charu na á bitu (v současné implementaci je použit 1byte na jeden boolean), tedy potřebu jedné osminy paměti na matici sousednosti.

Co se rychlosti týče, bylo by možné doprogramovat rychlejší úpravu matice sousednosti tedy optimalizaci n^3 složitého Floyd-Marshallova algoritmu. Podrobný popis je opět v přiloženém souboru `fast_ccl.pdf`.

4. Uživatelská příručka

Instalace:

Program je dodáván v .zip archivu. Pro jeho rozbalení je potřeba program zip respektive unzip (na win platformě vinzip). Pro vlastní překlad se předpokládá dostupnost nástroje pro překlad (gcc, wcc386) a nástroje make (wmake).

unix (gcc):

```
unzip ccl.zip
cd ccl
make
copy ccl /target/directory/
```

Rozbalíme archiv, přesuneme se do jeho vnitřku. Otevřeme a případně upravíme makefile, defaultně je připraven pro překlad na platformě linux. Spustíme překlad příkazem `make` a výslednou binarku ccl zkopírujeme na potřebné místo, pravděpodobně do nějakého /bin adresáře v cestě

windows (watcom):

```
winzip ccl.zip
cd ccl
make.bat
copy ccl.exe d:\target\directory
```

Rozbalíme archiv a vstoupíme do nově vzniklého adresáře ccl. Zde spustíme připravený dávkový soubor `make.bat`. Pokud jsou v cestě potřebné nástroje z balíku watcom (http://www.openwatcom.org/), dojde k přeložení a sestavení výsledného programu ccl.exe. Ten je pak možno zkopírovat na potřebné místo.

Používání

Program ccl (ccl.exe) je konzolová aplikace určená k obarvení souvislých oblastí v černobílém obrázku formátu .pgm. Spouští ze příkazem:

windows:

```
ccl.exe <infile> <outfile>
```

unix:

```
ccl <infile> <outfile>
```

Pokud nejsou zadány dva povinné parametry, tedy vstupní soubor a výstupní soubor, vypíše program chybové hláčení s návodem na použití.

5. Závěr

Semestrální práce byla dotažena do stádia funkčního programu přeložitelného na platformách w32 i linux (a pravděpodobně i dalších). Program by byl dále optimalizovatelný, zejména co se týče náročnosti na paměť (optimalizace ukládání kolizních dat) a rychlosti (fundovanější algoritmus na detekci kolizí).