DUM č. 18 v sadě

35. Inf-11 Objektové programování v Greenfoot

Autor: Lukáš Rýdlo

Datum: 08.06.2014

Ročník: studenti semináře

Anotace DUMu: Filtr vykreslení kruhu. Zesvětlení obrazu. Výpočet pozice středu kruhu v obrazu. Předávání vstupních údajů jako parametry z příkazové řádky.

Materiály jsou určeny pro bezplatné používání pro potřeby výuky a vzdělávání na všech typech škol a školských zařízení. Jakékoliv další využití podléhá autorskému zákonu.

evropský sociální fond v ČR
EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
OP Vzdělávání pro konkurenceschopnost
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Projekt: konzolový BMP editor v C
Filtr kruhu a volby příkazové řádky

Úvod

Tato lekce ukazuje další možný efekt spočívající ve vykreslení kruhu. Tento útvar je zvolen záměrně, aby si starší studenti procvičili analytickou geometrii a viděli její aplikaci do programátorské praxe. Mladší studenti budou většinou překvapeni možností „spočítat“ body kruhu v grafu. Úloha bude rozšířena o různé variace (změna polohy a poloměru, zesvětlení pixelů v oblasti kruhu).

Před započetím programování je důležité udělat „analytický rozbor“ a se studenty probrat vzorec pro kruh. Vyjdeme ze vzorce pro kružnici se středem v bodě [0;0], tedy $x^2+y^2=r^2$. V další fázi jej upravíme pro kruh a podiskutujeme nad významem ostré a neostré nerovnosti. Nakonec se pokusíme vymyslet (nebo najdeme na internetu), jak se vzorec změní při posunu středu. Studenty k řešení vedeme a podporujeme ve vlastní úvaze nebo aktivním hledání informací. Nebudeme řešení prozrazovat, protože najít řešení problému je hlavní programátorova (analytikova) práce. Samotné zapsání programu po vyřešení problému je už řemeslná (kodérova) práce.

Jako doplněk program rozšíříme o možnost zadávat všechny potřebné údaje (názvy souborů a volbu konkrétního filtru) z příkazové řádky. Studenty nasměrujeme k načtení informací o významu argumentů funkce main (argc a argv), které právě nesou informace předané z příkazové řádky. Bylo by vhodné, aby sami přišli na jejich význam. Můžeme je navést na vhodné zdroje na webových stránkách: http://www.sallyx.org/sally/c/c14.php, http://stackoverflow.com/questions/3898021/mainint-argc-char-argv nebo http://en.wikipedia.org/wiki/Entry_point.

Úkoly s řešeními

1. Do souboru bmpfilters.h a bmpfilters.c doplňte nový filter, který do souboru vykreslí modrý kruh do rohu o souřadnicích [0;0] s poloměrem 30 px. Uvidíme tedy jen čtvrtinu kruhu (pokud bude obrázek větší než 30 px. Vytvořte bmp2circle.c, který bude vypadat stejně jako bmp2gray.c, jen bude místo volání filtru pro kreslení šedé volat kreslení kruhu. Překompilujte do objektových souborů bmp2circle.c a bmpfilters.c, slinkujte a vyzkoušejte. Deklarace filtru nechť je následující:

```
void makeBlueCircle(sPixel ** image, fileHeader * fh, picHeader * ph);
```

Po teoretickém úvodu by řešení nemělo být nijak složité, pouze je potřeba si uvědomit, že nemá smysl nerovnici o dvou neznámých řešit, ale stačí ji vložit do podmínky při cyklickém procházení všech pixelů a pak jen vyplňovat pixel, když je podmínka splněna. V rámci vyšší efektivity kódu můžeme cyklus omezit tak, že budeme procházet do šířky i do výšky pixely v intervalu 0 až 30 (poloměr), jinde nemůže být podmínka splněna z definičního oboru. Je nutné ale použít i podmínku pro šířku/výšku obrázku kvůli případům, kdy je obrázek menší než 30 pixelů. V ukázkovém kódu jsou opět vynechávky (druhou mocninu ve vzorci počítejte jednoduše z definice druhé mocniny pomocí základní početní operace, netřeba vkládat matematickou knihovnu). V hlavičkovém souboru přibude pouze uvedená deklarace. Do bmpfilters.c přibude implementace:

```
void makeBlueCircle(sPixel ** image, fileHeader * fh, picHeader * ph)
{
  int x, y;

  for(y=0;y<□ □ y<=30;y++) {
          for(x=0;x<□ □ x<=30;x++) {
             if (□+□<=900) {
                   image[x][y].blue = □;
                   image[x][y].green = 0;
                   image[x][y].red = 0;
             }
          }
  }
}
```

Je možné, že kružnice bude deformovaná, špatně umístěná nebo u obrázků s počtem bytů v řádku nedělitelným 4 bude zšikmená. Pak je chyba v řešení úkolu o načítání nebo ukládání s doplňkem o čtení/zápisu bytů navíc do dělitelnosti 4. Obvykle to bývá jejím opomenutím, špatným posunem na začátek obrazových dat (chybná hodnota offsetu v fseek) nebo opomenutím v tom, že šířka řádky není počet bytů, ale počet pixelů. Kompilaci provedeme sekvencí příkazů:

```
gcc -c -o bmpfilters.o bmpfilters.c
gcc -c -o bmp2circle.o bmp2circle.c
gcc -o bmp2circle bmp2circle.o bmpfilters.o bmp.o
```

2. Upravte předchozí funkci, aby kreslila kruh doprostřed obrázku a aby jeho poloměr byla čtvrtina šířky nebo výšky obrázku, podle toho, co je menší.

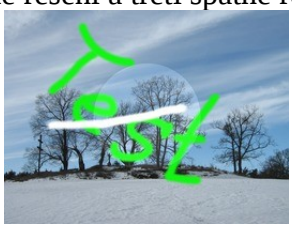
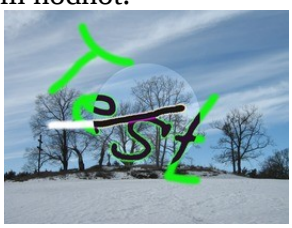
   Řešení vyžaduje jen drobnou úpravu filtru. Proto bude stačit překompilovat pouze bmpfilters.c a linkovat. Samotná úprava kódu vypadá takto (v cyklech je pochopitelně možno cyklit přes

všechny body, ale stačí jen přes „čtverec opsaný kruhu“, což je rychlejší a známe jeho střed/těžiště i délku strany, což pro určení souřadnic rohů stačí):

```
void makeBlueCircle(sPixel ** image, fileHeader * fh, picHeader * ph)
{
  int xc = □/2;   //circle center-x
  int yc = □/2;  //circle center-y
  int r  = (□<□?ph□width:ph□height)/4; //radius
  int x, y;

  for(y=□-□;y<□+□;y++) {
          for(x=□-□;x<□+□;x++) {
             if ((□-□)*(□-□)+(□-□)*(□-□)<□*□) {
                   image[x][y].blue = □;
                   image[x][y].green = 0;
                   image[x][y].red = 0;
             }
          }
  }
}
```

3. Zkopírujte předchozí funkci pod novým názvem makeLightCircle (nezapomeňte i na hlavičkový soubor a main). Upravte předchozí funkci, aby kruh nepřekreslil modře původní obrázek, ale aby se obrázek v místě kruhu zesvětlil (obrázek bude patrný, jen v místě kruhu bude světlejší). Viz ukázka. Zkuste na různých obrázcích, především na takovém, který bude už v místě kruhu mít hodně světlé barvy nebo bílou. Nesmí se stát, že se místo ztmaví. Viz následující tři obrázky, první je původní vzor, druhý správné řešení a třetí špatné řešení s přetečením hodnot.

Stačí upravit řádky s přiřazením barvy v podmínce kruhu. Nebudeme přiřazovat konkrétní barvu, ale všem třem složkám zvýšíme jejich hodnotu o společnou konstantu (třeba o 25). Při tom je ale nutné zkontrolovat, že po přičtení 25 nepřeteče číselný rozsah proměnné, protože by hodnota klesla zase k nízkým číslům a pixel by se ztmavil. Zde je ukázka pro modrý kanál s vynechanou proměnnou symbolem □ , symbol ○ označuje vynechaná čísla, která musí student sám dopočítat:

```
image[x][y].blue  = ( □>○ ? ○ : □+○ );
```

4. Opakované zadávání stejných názvů souborů při testech je otravné. Zařiďte, aby program raději přebíral název vstupního a výstupního souboru jako parametry na příkazové řádce. Pokud uživatel nepředá na příkazové řádce žádné parametry, vypište informaci o tom, jak má program spouštět a vraťte chybu ERR_ARG.

   Použijeme proměnné argc a argv ze záhlavím mainu. V první máme předaný počet parametrů, v druhé pole textových řetězců (polí znaků), které nesou požadované parametry. Vzorové řešení ukazuje pouze nové a upravené části mainu s vynecháním jednoho znaku symbolem □, celé podmínky znakem ○ .

```
if (arg□!=□) {
   printf("Chybne volani. Program prebira nazev vstupniho a vystupniho
souboru.\n");
   printf("%s infile.bmp outfile.bmp\n\n",arg□[□]);
   return ERR_ARG;
}

status = readBMPFile(arg□[□], □fh, □ph, □image);
if(○) makeLightCircle(image, &fh, &ph);
if(status==ERR_OK) status = writeBMPFile(arg□[□], □fh, □ph, □image);
```

5. Zkontrolujte, jakou návratovou hodnotu program vrací, pokud ho spustíte bez parametrů nebo s parametry v různých chybových situacích (špatné jméno vstupního souboru, soubor není bmp apod.). Pracujte v Linuxu a použijte v příkazové řádce vypisování příkazem `echo $?`, který vypíše návratový kód posledně spuštěné aplikace.

6. Pracujte v Linuxu v konzoli (bash). Opište do příkazové řádky následující rozsáhlý složený příkaz, který hromadně přidá do všech .bmp souborů v adresáři světlý kruh naším programem. Pokud se konverze nezdaří, vypíše „CHYBA a název souboru“. Za tímto účelem do adresáře nakopírujte několik souborů s koncovkou .bmp, aby některé byly obrázky a jiné ne. Příkaz pište jako jeden řádek a vypadá takto:

```
for S in *.bmp; do ./bmp2circle $S circle-$S >/dev/null;if [ $? -ne
0 ]; then echo CHYBA $S; fi; done
```

Můžete zkusit, totéž s filtrem převodu na šedou a pak spustit následující příkaz, který obsahuje volání programu convert z balíku ImageMagick, který slouží ke grafickým konverzím a úpravám v Linuxu (v LiveCD Knoppix je předinstalovaný), a porovnat jeho výstup s výstupem svého programu. Příkaz postupně převede všechny bmp soubory v adresáři na jpg soubory v odstínech šedé.

```
for S in *.bmp; do convert $S -type grayscale ${S%bmp}jpg >/dev/null;
fi; done
```

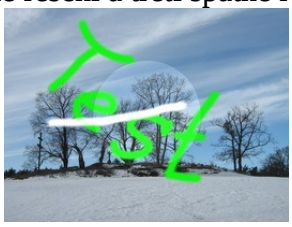
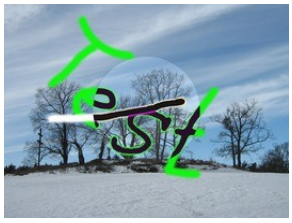
Úkoly

1. Do souboru bmpfilters.h a bmpfilters.c doplňte nový filter, který do souboru vykreslí modrý kruh do rohu o souřadnicích [0;0] s poloměrem 30 px. Uvidíme tedy jen čtvrtinu kruhu (pokud bude obrázek větší než 30 px. Vytvořte bmp2circle.c, který bude vypadat stejně jako bmp2gray.c, jen bude místo volání filtru pro kreslení šedé volat kreslení kruhu. Překompilujte do objektových souborů bmp2circle.c a bmpfilters.c, slinkujte a vyzkoušejte. Deklarace filtru nechť je následující:

```
void makeBlueCircle(sPixel ** image, fileHeader * fh, picHeader * ph);
```

2. Upravte předchozí funkci, aby kreslila kruh doprostřed obrázku a aby jeho poloměr byla čtvrtina šířky nebo výšky obrázku, podle toho, co je menší.

3. Zkopírujte předchozí funkci pod novým názvem makeLightCircle (nezapomeňte i na hlavičkový soubor a main). Upravte předchozí funkci, aby kruh nepřekreslil modře původní obrázek, ale aby se obrázek v místě kruhu zesvětlil (obrázek bude patrný, jen v místě kruhu bude světlejší). Viz ukázka. Zkuste na různých obrázcích, především na takovém, který bude už v místě kruhu mít hodně světlé barvy nebo bílou. Nesmí se stát, že se místo ztmaví. Viz následující tři obrázky, první je původní vzor, druhý správné řešení a třetí špatné řešení s přetečením hodnot.

4. Opakované zadávání stejných názvů souborů při testech je otravné. Zařiďte, aby program raději přebíral název vstupního a výstupního souboru jako parametry na příkazové řádce. Pokud uživatel nepředá na příkazové řádce žádné parametry, vypište informaci o tom, jak má program spouštět a vraťte chybu ERR_ARG.

5. Zkontrolujte, jakou návratovou hodnotu program vrací, pokud ho spustíte bez parametrů nebo s parametry v různých chybových situacích (špatné jméno vstupního souboru, soubor není bmp apod.). Pracujte v Linuxu a použijte v příkazové řádce vypisování příkazem `echo $?`, který vypíše návratový kód posledně spuštěné aplikace.

6. Pracujte v Linuxu v konzoli (bash). Opište do příkazové řádky následující rozsáhlý složený příkaz, který hromadně přidá do všech .bmp souborů v adresáři světlý kruh naším programem. Pokud se konverze nezdaří, vypíše „CHYBA a název souboru“. Za tímto účelem do adresáře nakopírujte několik souborů s koncovkou .bmp, aby některé byly obrázky a jiné ne. Příkaz pište jako jeden řádek a vypadá takto:

```
for S in *.bmp; do ./bmp2circle $S circle-$S >/dev/null;if [ $? -ne
0 ]; then echo CHYBA $S; fi; done
```

Můžete zkusit, totéž s filtrem převodu na šedou a pak spustit následující příkaz, který obsahuje volání programu convert z balíku ImageMagick, který slouží ke grafickým konverzím a úpravám v Linuxu (v LiveCD Knoppix je předinstalovaný), a porovnat jeho výstup s výstupem svého programu. Příkaz postupně převede všechny bmp soubory v adresáři na jpg soubory v odstínech šedé.

```
for S in *.bmp; do convert $S -type grayscale ${S%bmp}jpg >/dev/null;
fi; done
```