VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

WARPING A MORFING OBRAZU

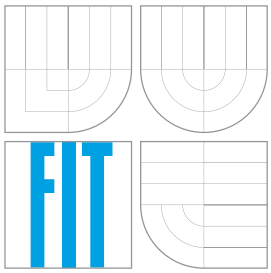
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PETR KUBĚNA

BRNO 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

WARPING A MORFING OBRAZU
IMAGE WARPING AND MORPHING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — PETR KUBĚNA
AUTHOR

VEDOUCÍ PRÁCE — Ing. JIŘÍ VENERA
SUPERVISOR

BRNO 2007

Zadání bakalářské práce/5391/2006/xkuben04

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2006/2007

Zadání bakalářské práce

Řešitel: **Kuběna Petr**
Obor: Informační technologie
Téma: **Warping a morfing obrazu**
Kategorie: Počítačová grafika

Pokyny:

1. Prostudujte problematiku spojenou warpingem obrazu.
2. Prostudujte problematiku morphingu mezi dvěma obrázky.
3. Implementujte tyto algoritmy a demonstrujte jejich funkčnost na vhodných datech.
4. Zhodnoťte Vámi dosažené výsledky a navrhněte možné rozšíření implementovaných algoritmů.

Literatura:

- Po domluvě na vyžádání u školitele.

Při obhajobě semestrální části projektu je požadováno:

- Body 1. a 2..

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Venera Jiří, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2006
Datum odevzdání: 15. května 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Pavel Zemčík
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Petr Kuběna**
Id studenta: 84287
Bytem: Střední 1901/4, 702 00 Ostrava
Narozen: 04. 09. 1984, Ostrava
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
bakalářská práce

Název VŠKP: Warping a morfing obrazu
Vedoucí/školitel VŠKP: Venera Jiří, Ing.
Ústav: Ústav počítačové grafiky a multimédií
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Abstrakt

Warping a morfing jsou počítačové efekty s použitím hlavně v zábavním průmyslu. Práce obsahuje postupy a techniky nutné k jejich vytvoření. Zaměřuje se na operace s dvourozměrným navzorkovaným obrazem. Nedílnou součástí je demonstrační program, který umožňuje uživatelsky tyto efekty vytvářet.

Klíčová slova

počítačová grafika, warping, morfing, mapování, míchání barev, transformace obrazu

Abstract

Warping and morphing are computer effects that are mainly used in show-bussines. Thessis contains procedures and techniques that are necessary for their creation. It is aimed at work with discrete two-dimensional image. Integral part is the demo program, which allow users to create such effects.

Keywords

computer graphics, warping, morphing, mapping, blending, image transformations

Warping a morfing obrazu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jiřího Venery. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Kuběna
14. května 2007

Poděkování

Rád bych tímto poděkoval vedoucímu práce Ing. Jiřímu Venerovi za vhodné směrování a dobré rady, které mi poskytl v průběhu tvorby bakalářského projektu.

Obsah

Kapitola 1

Úvod

Cílem této bakalářské práce, v rámci vymezeném tématem a pokyny, je popsat a vysvětlit základní principy morfování a warpování obrazu, které jsou v současnosti běžně používány. Tato oblast je dnes celkem dobře a detailně prozkoumána, a proto lze bez obav popsat konstrukci, jejíž moderní podoba byla vystavěna zejména za posledních dvacet let.

Morfing obrazu, ač nazýván jinak, se však objevil dávno před érou počítačů, natož počítačové grafiky. Už z roku 1906 pochází první film zobrazující přeměnu mladého muže ve starce. Tento první pokus o proměnu, alespoň co se týká stříbrného plátna, byl proveden jednoduchým prolnutím dvou snímků. Na prvním z nich sedí herec přestrojený za mladého muže, na druhém za starce. Velice brzo se tato metoda stala běžným postupem ve filmech. Většinou se ale jednalo jen o trochu nápaditější prostřih a nikoliv o snahu naznačit přeměnu v jiný objekt.

S rozvojem filmu se objevily další techniky, jak provést metamorfózu. Film *Dracula* přišel počátkem třicátých let minulého století s možností využít ručně animovaných snímků k vytvoření iluze přeměny hraběte Drakuly v hejno netopýrů. O deset let později, roku 1941, byl ve filmu *The Wolf Man* (obrázek 1.1) přeměněn člověk ve vlkodlaka a nazpět. Byla použita technika "stop-motion" – snímek za snímkem byl herec přeličován. Těchto pár vteřin filmu trvalo vytvořit celé hodiny. Metoda stop-motion byla použita sice skoro o desetiletí dříve (za zmínku stojí třeba film King Kong z roku 1933), ale až v tomto filmu to bylo k morfingu.

Čtyřicátá léta jsou také období, ve kterém se začíná popisovat první stránka historie počítačů. Z dnešního pohledu je generace počítačů ze čtyřicátých let označována za nultou. Tyto počítače se nikdy ke grafice nedostaly. Teprve až s druhou generací se dostává ke slovu počítačová grafika. Roku 1962 byl Ivanem Sutherlandem vytvořen první grafický interaktivní software *Sketchpad*. Z tohoto období se zdaleka nejvýznamnějším pro historii morfingu stal rok 1967. Tehdy byla totiž na Torontské univerzitě popsána technika 2D morfingu.

Na konci šedesátých let se počítačová grafika poprvé dostává do filmu. Mezi první takové filmy patří slavný film Stanleyho Kubricka *2001: A Space Odyssey* (2001: Vesmírná odysea). Od této chvíle se čím dál častěji objevovaly počítačem vytvořené filmové efekty. Z pohledu morfingu se ale dlouho nic zajímavějšího nedělo. Teprve roku 1988 se objevil první film s detailně provedeným počítačovým morfingem. Jednalo se o fantasy snímek *Willow*. V něm ukázalo známé studio Industrial Light and Magic proměnu staré čarodějnice v různá zvířata.

V devadesátých letech došlo k rozšíření působnosti počítačového morfingu mimo oblast filmu. Na konci klipu Micheala Jacksona *Black Or White* se volně a na pohled velmi

Obrázek 1.1: Ukázka morfingu z filmu The Wolf Man (1941)

přirozeně morfuje mezi obličeji mužů a žen. Právě na tomto klipu jsou často ukazovány principy 2D morfingu a warpingu. Následoval *Terminator 2: Judgement Day* (Terminátor 2: Den zúčtování), který ukázal na svou dobu úchvatný 3D morfing. Od té doby proběhl přesun těchto technik na osobní počítače a tato situace je tu dodnes.

Zájemci o podrobnější historii můžou navštívit stránky oddělení *ACCAD* na Ohio State University [9], kde můžou nalézt podrobnou dvacetidílnou serii pojednávající právě o dějinách počítačové grafiky.

1.1 Použité zdroje

Semestrální práce, na kterou projekt navazuje, spočívala v nastudování problematiky spojené s morfováním a warpováním obrazu. V návaznosti na ni vznikl také demonstrační program, umožňující jmenované techniky.

Fakta uvedená v celém textu jsou z velké části čerpána z níže uvedených hlavních zdrojů, které nejvíce umožnily pochopit nejen principy spojené s warpingem a morfingem, ale i širší okruh témat počítačové grafiky. Obecnější poznatky se dají získat v knížce Jiřího Žáry, Bedřicha Beneše a Petra Felkela *Moderní počítačová grafika* [5]. Diplomová práce Martina Dobšíka na téma *Morfing* [2] popisuje specifiku a konečně různé detaily poskytla internetová encyklopedie *Wikipedia* [8]. Použity byly samozřejmě i jiné zdroje – ty ale byly vždy spojeny jen s jednou oblastí, a budu je proto citovat vždy až u jednotlivých kapitol.

1.2 Struktura práce

V úvodní kapitole bude nejdřív vysvětlen význam některých klíčových slov. Poté bude popsáno, jak se historie morfingu, která je úzce spjata s filmem, začala prolínat s dějinami počítačů. Konec první kapitoly patří stručnému výpisu některých oblastí, ve kterých se hlavní témata práce uchytila.

Následuje část zabývající se morfingem obrazu, obecným algoritmem a popisem jednotlivých částí. Výjimkou je část popisující warping. Jedná se o největší a nejsložitější část, proto jsem mu vyčlenil celou kapitolu.

Po těchto dvou teoreticky orientovaných kapitolách následuje popis připojeného programu, jeho návrh, implementace a ovládání. V závěru jsou zhodnoceny dosažené výsledky a navržena některá rozšíření a směry, kudy by se případná navazující práce mohla ubírat.

1.3 Klíčová slova a konvence

Je několik pojmů, které je třeba vysvětlit – a to jejich význam lehce specifický pro tuto práci. Mezi nejdůležitější patří už v nadpisu zmíněné pojmy morfing a warping.

Morfing (odvozený z řeckého metamorphosis) je v obecném smyslu přeměna jednoho objektu v jiný. Většinou je snaha, aby tato byla co nejplynulejší – jednak barevně a hlavně tvarově. Práce se zaměřuje na manipulaci s dvourozměrným navzorkovaným obrazem. Proto v dalším textu budu pojmem morfing označovat přeměnu jednoho navzorkovaného obrazu (případně jen jeho části) v jiný.

Druhý ze dvou nejdůležitějších pojmů této práce, *warping*, lze přeložit jako tvarová změna. Obecně je tato změna nelineární a v navzorkovaném obraze se nejčastěji jedna o předpis určující posun každého vzorku v obraze.

Často se v kontextu warpingu a morfingu mluví o *tweeningu*. Jedná se o zkrácenou verzi anglického *in-betweening*, tedy dělání mezi-snímků. Pojem se svým významem lehce překrývá s pojmem morfing, nezapadá do zvoleného systému a dále v práci už nebude zmiňován.

Pro popis jednotlivých algoritmů bude použit pseudokód založený na programovacím jazyce C. Další konvencí (pokud nebude explicitně uvedeno jinak) bude počítání v kartézském souřadnicovém systému (týká se zejména kapitol Morfing a Warping).

1.4 Praktické využití

Předchozí kapitola naznačila, že morfing (a warping jako jeho součást) je pojmem úzce spjatým s filmem. Přestože se s jeho použitím můžeme setkat i jinde, jeho hlavní doménou stále zůstává zábavní průmysl – reklamy, hudební klipy, videa, dělání mezisnímků v animovaných filmech, počítačové hry a další.

Širší použití lze najít u warpingu. Samozřejmě jako prakticky nezbytný proces je všude tam, kde je morfing. Navíc ho lze najít i v jiných oblastech:

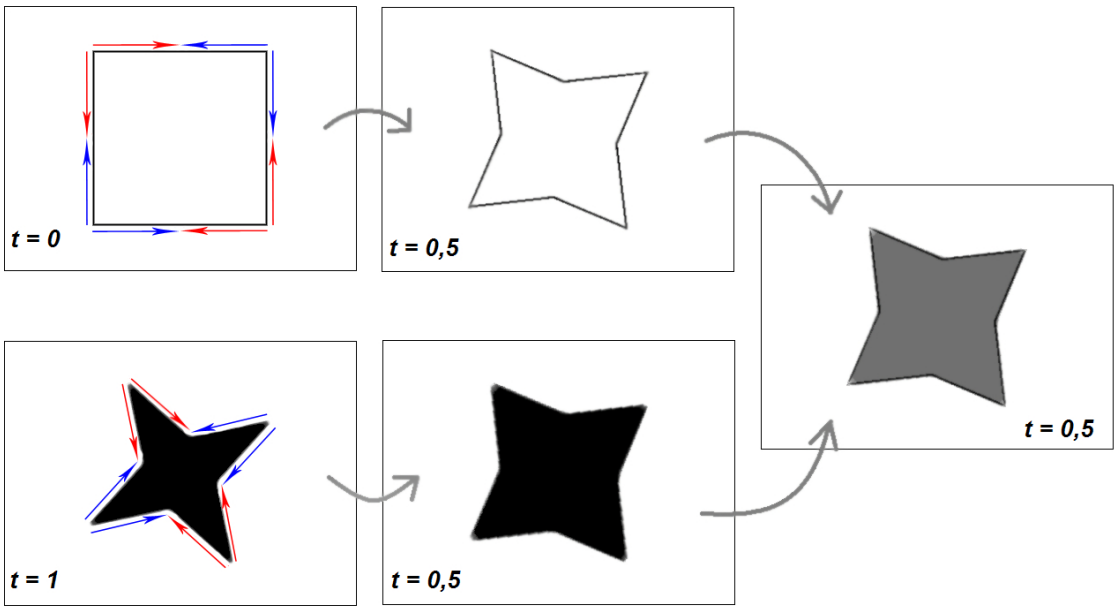
- Zpracování obrazu z různých senzorů – příkladem může být vytvoření korektního obrazu prstu ze stále častěji se objevujících čteček otisků.
- Tvorba grafických efektů (nesouvisejících přímo s morfingem) – rybí oko, různé prohýbání obrazu, efekty zmizení části obrazu a další.

Kapitola 2

Morfing

Tato kapitola podrobněji popisuje, jak morfing probíhá a které fáze obsahuje. Krom dříve uvedených zdrojů byly pro tuto a následující kapitolu použity informace získané ze stránek Princeton University [11] [12].

Jak bylo zmíněno v sekci 1.3, jedná se o proces, při kterém probíhá plynulá změna jednoho objektu v jiný. Tento proces lze definovat pro libovolný počet rozměrů, nejčastěji je ale použit pro dvou- a trojrozměrné objekty. Jak bylo dříve zmíněno pro rámec celé práce, i tato kapitola se bude zabývat jen morfingem navzorkovaného 2D obrazu s rovnoměrně ortogonálně rozloženými vzorky (tedy v praxi běžnými obrázky, například ve formátu .bmp).



Obrázek 2.1: Postup morfingu

Ve chvíli, kdy jsou přijata omezení, lze celý proces morfingu popsat ve třech krocích, které v následujících sekcích rozeberu podrobněji:

1. Vložení vstupních informací
2. Warping

3. Prolnutí

Výstupem morfingu obvykle bývá sekvence obrázků. Body dva a tři je proto nutné pro každý obrázek zopakovat. Obecný pseudokód pro morfing by mohl vypadat následovně:

```
NactiObrazky(vstupniObrA, vstupniObrB);
NactiTopologii(topologiaA, topologieB);
for(i = 0; i < pocetSnimku; i++)
{
  topologieI = InterpolujTopologii(topologieA, topologieB,
                                   f(i/pocetSnimku));
  transformovanyObrA = WarpObr(vstupniObrA, topologieA, topologieI);
  transformovanyObrB = WarpObr(vstupniObrB, topologieB, topologieI);
  snimek[i] = ProlniObrazky(transformovanyObrA, transformovanyObrB,
                            g(i/pocetSnimku));
}
```

kde:

- *vstupniObrA* a *vstupniObrB* jsou struktury obsahující bitmapové obrázky v počátečním resp. cílovém stavu; *transformovanyObrA* a *transformovanyObrB* jsou pomocné struktury pro uložení obrázků a snimek[] je pole těchto struktur o velikosti *pocetSnimku* určené k uložení výstupu
- *topologieA* a *topologieB* jsou struktury obsahující pro daný typ warpingu formátovanou topologickou informaci o obrázcích v počátečním a cílovém stavu; *topologieI* je topologie interpolovaná pro daný časový okamžik
- *InterpolujTopologii(tA, tB, t)* je funkce, jejíž vstupními informacemi jsou údaje o topologii (parametry tA, tB) v čase $t = 0$ a $t = 1$ a čas t z intervalu $< 0; 1 >$, do kterého má výstupní topologii interpolovat
- *WarpObr(obrA, tA, tI)* je funkce, která provede warp obrázku obrA s topologií tA tak, aby odpovídal topologii tI
- *ProlniObrazky(obrA, obrB, ratio)* provede alfa míchání obrázků uložených ve struktuře obrA a obrB v závislosti na poměru ratio (nemusí to být přímo v poměru i/pocetSnimku – viz kapitola 2.3)
- pro funkce *f(x)* a *g(x)* je vstupem číslo z intervalu $< 0; 1 >$ a vrací číslo ze stejného intervalu; současně by v tomto intervalu měly být spojité alespoň v nulté derivaci

Existuje několik různých druhů. Ty se ale liší jen warpovacím jádrem (a tedy i formátem vstupní topologické informace). Proto tyto nejsou členěny na úrovni morfingu, jak bývá zvykem, ale až v kapitole následující, která se zabývá warpingem. Text bude zejména zaměřen na feature-based morfing a mesh morfing, kterých bylo využito v přiloženém programu.

Jako všude, i tady existuje výjimka, a tou je *View morphing* (morfing s kompenzací úhlu pohledu). V tomto případě se ale jedná spíše o rozšíření běžných morfovacích technik, kdy ještě před warpingem vstupuje do procesu tzv. *prewarp* fáze a po prolnutí obrázků následuje ještě *postwarp* fáze.

2.1 Vložení vstupních informací

Na začátku je třeba říct, které vstupní informace je nezbytné poskytnout.

První jsou samozřejmě dva bitmapové obrázky – zdrojový a cílový. Už tady je důležitá vhodná volba. Obrázky by měly zobrazovat tvarově co nejpodobnější objekty; barevná podobnost není tak důležtá, protože i lineárně interpolovaná barva v závislosti na čase vytvoří opticky plynulý barevný přechod. I když pomocí warpingu jde skoro libovolně obraz transformovat, výsledek morfingu dvou velmi odlišných objektů buď nebude vypadat přirozeně, nebo bude nutné vložit velké množství topologických údajů.

Tím se dostáváme k druhému vstupu. Jsou jím topologické informace jednak o obsahu zdrojového a cílového obrázku a jednak o referencích mezi nima. Tyto údaje můžou být dodány ručně nebo je lze generovat algoritmy pro rozpoznávání obrazu. Samozřejmě je taky možné kombinovat oba postupy a to v jakémkoli pořadí. Ruční úprava může sloužit jen k upřesnění/upravení počítačem navržené topologie a naopak počítač může pomáhat s ukotvením topologických informací přesně na hrany. Obecně se dá říci, že ruční vložení těchto údajů dává většinou kvalitnější výsledky. Přinejmenším z části to bude dáno tím, že kvalita výstupu morfingu je nejčastěji hodnocena okem diváka a lidské vnímání je poměrně obtížné kvantifikovat či zalgoritmizovat.

2.2 Warping

Toto téma je stěžejní oblastí bakalářské práce a proto je mu věnována celá kapitola 3. Přesto zde uvedu alespoň základní informace nutné k pochopení postupu morfingu.

Vstupem jsou informace získané ve fázi předchozí (bitmapové obrázky, topologické údaje). K tomu je třeba přesně znát, v jakém čase chceme mezi-snímek získat. Výstupem jsou dva obrázky – transformovaný zdrojový a cílový.

2.3 Prolnutí

Vstupem jsou dva obrázky získané z fáze warpingu a opět čas. Snímky prolneme (třeba pomocí alfa míchání) a tím získáme konečnou podobu mezi-snímku.

I když by se mohlo zdát, že zde není dále co říct, opak je pravdou. V tomto bodě záleží na funkci udávající, jak se snímky prolnou. Nejjednodušší variantou je linearní závislost poměru, ve kterém se celé snímky prolnou na čase ($f(x) = x$). Pro lepší výsledky je tuto možno nahradit libovolnou funkcí. Dobře použitelnými jsou zde třeba vhodně parametrizované Lagrangeovy křivky. S jejich pomocí jde jednoduše uživatelsky nastavovat, jak se v průběhu času bude měnit poměr při míchání.

Další možností je přidat závislost výpočtu poměru míchání na poloze daného bodu v obraze. Cílový obraz se tak může začít odkrývat například od středu.

Poslední rozšíření této fáze, na které lze narazit, je oddělení morfovaného objektu a jeho okolí. Takový postup může zlepšit dojem reálnosti procesu morfování a zvýraznit transformovávaný objekt, ale za cenu nutnosti jasné definice, co je součástí objektu a co už nikoliv.

Kapitola 3

Warping

Jedná se o transformaci vstupního obrazu, kdy tento pomocí libovolné mapovací funkce upravující topologii (kapitola 3.1) změníme. V nejjednodušším případě můžeme za takové funkce považovat i elementární transformace, jakými jsou posun, otočení, zkosení a změna velikosti (kapitola 3.3).

Warping je používán k mnoha obrazovým efektům – příkladem může být třeba známé "rybí oko", spirála, lupa a další. Takové obrazové efekty jsou pro morfing jen těžko použitelné. Pro něj je důležité mít co nejlepší kontrolu, kam se jednotlivé části zobrazovaného objektu po změně dostanou.

K takovému účelu jsou nejvhodnější a prakticky vždy používány metody, na jejichž vstupu jsou zadávány dvojice bodů určující posun odpovídajících míst v obraze. Jednotlivé body bývají nejčastěji součástí větších struktur – úseček, vektorů, sítě trojúhelníků či čtyřúhelníků aj.

Samotný proces má ve své podstatě jen dvě části. Mapování (kapitola 3.1) určí buď kam se má vzorek z původního obrazu posunout, a nebo polohu odkud se má vzít vzorek z cílového obrazu. Druhá část je vlastní určení barvy vzorku (3.2).

Zdrojové kódy úsečkového i síťového warpingu, které byly použity v programu, jsou v příloze A.

3.1 Mapování

Mapování je proces, při kterém je pomocí mapovací funkce popsán vznik nového obrazu ze vzorků obrazu původního. Rozlišují se dva druhy:

- *dopředné mapování* (z anglického forward mapping),
- *zpětné mapování* (z anglického backward/reverse mapping).

U dopředného mapování se určuje, kam je z původního obrazu každý vzorek posunut v obrazu výsledném. Zdrojový kód takové transformace by vypadal následovně:

```
for(int u = 0; u < usize; i++)
{
  for(int v = 0; v < vsize; j++)
  {
    x = fx(u, v);
    y = fy(u, v);
```

```
    barva = NactiBarvu(vstupniObr, u, v);
    UlozBod(vystup, x, y, barva)
  }
}
```

kde

- u, v jsou souřadnice ve zdrojovém obrazu a x, y jsou souřadnice v cílovém obraze
- *fx()* a *fy()* jsou složky mapovací funkce určující nové souřadnice bodu
- *NactiBarvu(obr, a, b)* vrací barevnou informaci na souřadnicích [a, b] v obrázku *obr*
- funkce *UlozBod(list, a, b, barva)* ukládá informaci o bodu se souřadnicemi [a, b] a barvě *barva* do struktury, která je schopná udržet obraz složený z náhodně umístěných bodů na ploše s neceločíselnými koordináty

Tato metoda je sice jednoduchá, ale zásadní nevýhodou jsou neceločíselné a v podstatě náhodné koordináty transformovaných bodů v novém obrazu. Takhle vzniklý obraz je těžké interpretovat. Proto je mnohem rozšířenější použití metody zpětného mapování. U toho se naopak ptáme, odkud máme vzít vzorek z původního obrazu pro konkrétní (celočíselné) koordináty v obrazu cílovém. V tomto případě sice pro změnu získají neceločíselné souřadnice, odkud se ze zdrojového obrazu má vzít barva, tento problém je ale řešen různými způsoby vzorkování (kapitola 3.2).

Pseudokód zpětného mapování:

```
for(int x = 0; i < xsize; i++)
{
  for(int y = 0; j < ysize; j++)
  {
    u = inverse_fx(x, y);
    v = inverse_fy(x, y);
    vystup[x][y] = UrciBarvu(vstup, u, v);
  }
}
```

kde

- u, v jsou souřadnice ve zdrojovém obrazu a x, y jsou souřadnice v cílovém obraze
- *inverse_fx()* a *inverse_fy()* jsou složky zpětné mapovací funkce určující pozici odpovídajícího vzorku ve zdrojovém obraze
- *vystup[a][b]* je dvourozměrné pole pro uložení barev jednotlivých bodů

3.2 Převzorkování

Výsledkem zpětného mapování jsou koordináty, odkud se má z původního obrazu vzít barevná hodnota. Jak ale bylo řečeno, jen výjimečně se stává, že mají složky celočíselné a k tomu v oblasti popsané vstupním obrazem. A právě způsob jak z barevných hodnot na celočíselných souřadnicích získat barvu pro bod nacházející se v prostoru (v tomto případě na ploše) mezi nima s různým úspěchem řeší jednotlivé postupy vzorkování. Některé

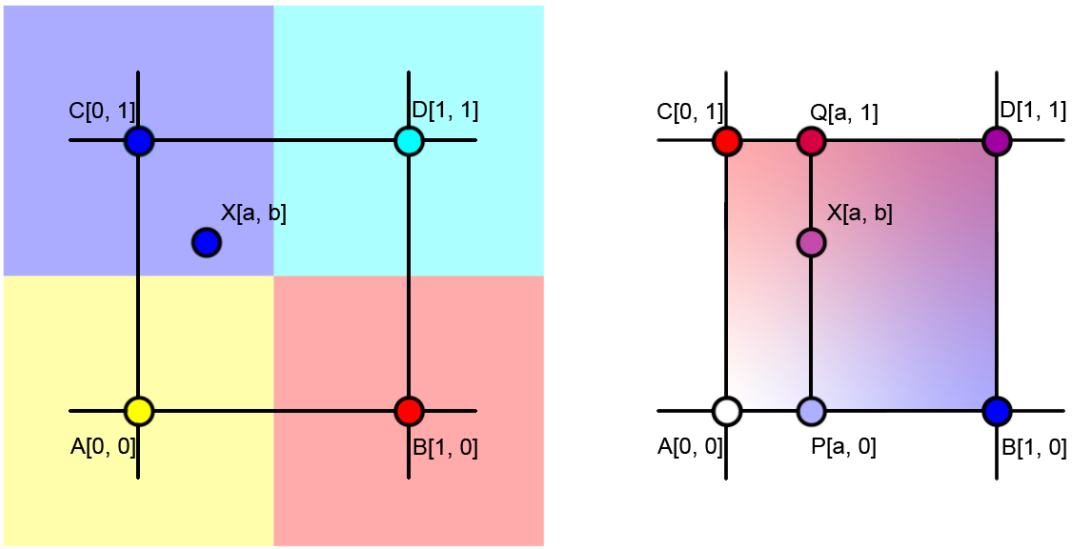
z nich tu budou z praktického hlediska rozebrány. Kapitola se nezmiňuje o teorii vzorkování, problému aliasu ani mnoha dalšími souvislostmi – tyto lze nalézt ve slidech k předmětům Základy počítačové grafiky [4] a Signály a systémy [3].

Základním způsobem je použití tzv. prostého vzorku (point-sample). Ideou je určení barvy daného bodu jen pomocí nejbližšího známého vzorku. V praxi to znamená zaokrouhlení souřadnic na celá čísla a přečtení barevné hodnoty. Tento postup je sice velice rychlý a implementačně jednoduchý, zanechává ale typický čtverečkovaný vzor ve výsledném obraze. Hodí se proto nanejvýš pro tvorbu náhledů, kde je důležitější rychlost, než přesnost výstupu. Následující řádek kódu ukazuje celý výpočet:

```
vystup[x][y] = NactiBarvu(vstup, trunc(u+0.5), trunc(v+0.5));
```

Lepší variantou je bilineární filtrování. K vypočtení barvy každého vzorku se použijí vždy čtyři nejbližší pixely. V reálu situace vypadá tak, že bod, kterému chceme přiřadit barvu, se nachází uvnitř čtverce, u něhož známe barevné hodnoty ve vrcholech. Nejsrozumitelněji se tento postup vysvětlí pomocí obrázku (3.1). Nejdříve se udělá vážený průměr barev levého horního a pravého horního bodu v poměru daném převrácenou hodnotou vzdáleností hledaného místa od levé, resp. pravé hrany čtverce (čím blíže, tím větší váha). Obdobný postup se zopakuje pro levý a pravý dolní vrchol. V této chvíli jsou známy barvy bodů P a Q. Ty jsou krajními body úsečky, na níž leží hledaný bod. Teď už stačí udělat ještě jednu lineární interpolaci a výsledkem je barva bodu X.

Samozřejmě stejného výsledku by se dosáhlo, kdyby se interpolovalo nejdříve vertikálně a potom horizontálně. Většinou je ale u počítačů o něco rychlejší přistupovat k bodům po řádcích.



Obrázek 3.1: Porovnání point-sample (vlevo) a bilineárního (vpravo) filtrování. Všimněte si, že v pravém obrázku lze ze znalosti těchto čtyř vzorků barevně vypočíst jen obsah čtverce.

Výše zmíněný postup odpovídá následujícímu kódu:

```
barva_a0 = interpoluj(NactiBarvu(vstup, trunc(u), trunc(v)),
                      NactiBarvu(vstup, trunc(u)+1, trunc(v)),
```

```
                        u-trunc(u));
barva_a1 = interpoluj(NactiBarvu(vstup, trunc(u), trunc(v)+1),
                        NactiBarvu(vstup, trunc(u)+1, trunc(v)+1),
                        u-trunc(u));
vystup[x][y] = interpoluj(barva_a0, barva_a1, v-trunc(v));
```

Existuje mnoho dalších způsobů vzorkování, pro většinu použití je ale bilineární filtrování plně dostačující. Zjednodušeně se dá říct, že čím větší množství okolních vzorků bere daná metoda v potaz, tím kvalitnější je její výsledek.

Často se stává, že souřadnice některých vypočtených bodů leží mimo vlastní obraz. Prakticky žádný algoritmus implicitně neudává, jak takovému vzorku přiřadit barvu. K určení se používá obvykle jeden ze dvou přístupů:

- Okolí obrázku se přiřadí konstantní barva. Potom lze použít nezměněný algoritmus.
- Bodům v okolí se přiřadí barva nejbližšího bodu v obraze. Většinou lépe zakrývá vykročení z prostoru obrázku. Tato vlastnost může být v některých situacích nežádoucí.

3.3 Elementární transformace 2D obrazu

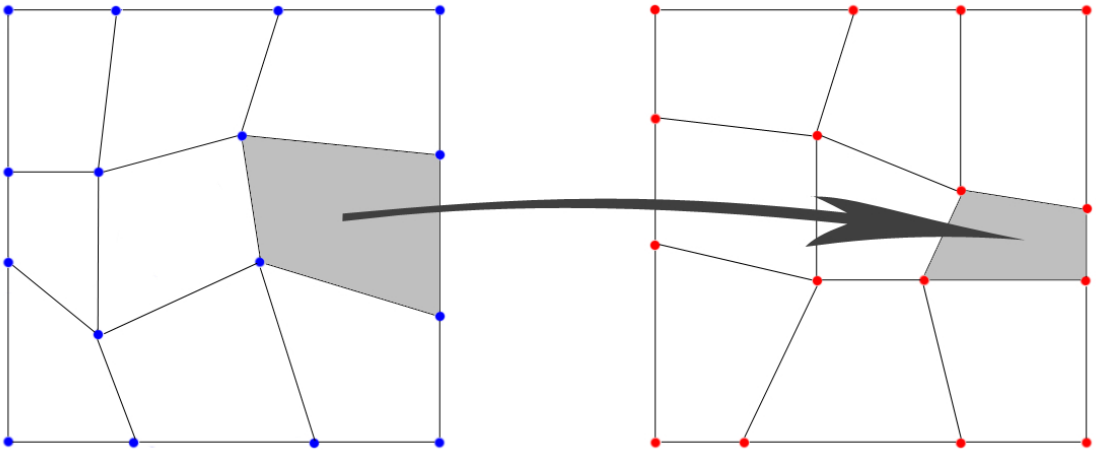
Ještě před popisem poměrně složitých transformačních funkcí, jakými jsou jádra tvarových změn určených k morfingu, je vhodné se zmínit o základních transformacích, které jsou v rovině uskutečnitelné, a proto i používané. Z nich jsou potom skládány složitější. Čtyři základní lineární geometrické transformace jsou:

- *posunutí* (translation)
- *otočení kolem počátku souřadnicového systému* (rotation)
- *změna měřítka* (scale)
- *zkosení* (shear)

Tyto procesy se nejčastěji popisují pomocí transformačních matic. Jsou to čtvercové matice s hranou o jedna větší, než je počet dimenzí, ve kterých danou transformaci provádíme (vzhledem k tomu, že se bavíme o 2D obrazu, mají rozměr 3×3). Přidanou souřadnicí je tzv. homogenizační faktor $w \neq 0$. Jednoduchým násobením těchto matic dospějeme ke skládání transformací (výsledek záleží na pořadí násobení).

3.4 Warpování sítě čtyřúhelníků

V originále označován jako *mesh warping*. Jsou definovány dvě mřížky o stejné velikosti (ve smyslu počtu a návaznosti uzlů). První odpovídá topologii zdrojového obrázku, druhá definuje cílovou. Nejsnáze to jde vysvětlit pomocí obrázku (obr. 3.2). Na něm je znázorněna výchozí a cílová mřížka. Současně je na něm vyznačen jeden z odpovídajících si segmentů. Jak lze z obrázku vyčíst, celá metoda se dá zjednodušit na problém přemapování původního segmentu na nový.



Obrázek 3.2: Ukázka počáteční a cílové mřížky se znázorněným odpovídajícím plošným segmentem

Pro přemapování těchto segmentů existuje několik různých metod (třeba viz [2]). Nejpoužívanější je pro svou rychlost, efektivnost a jednoduchost implementace využito dvouprůchodové řešení. Toto staví principielně na rozdělení transformací ve směru osy x a y. Postup vypadá následovně:

1. Vytvoří se mezi-mřížka tak, že pro každý uzel se vezme informace o vertikální poloze z počáteční a o horizontální z cílové mřížky. Získáme tak mřížku transformovanou jen horizontálně (ve směru osy x).

2. Pro každý řádek se vypočítají průsečíky se svislými úsečkami nově získané mezi-mřížky. Tím se logicky rozdělí daný řádek na segmenty, jejichž počet je o jedno vyšší, než je vertikální rozlišení mřížky. Dva segmenty ale přímo do obrázku nezasahují, a proto se dají zanedbat (první končí na levém okraji obrazu a druhý na pravém okraji začíná).

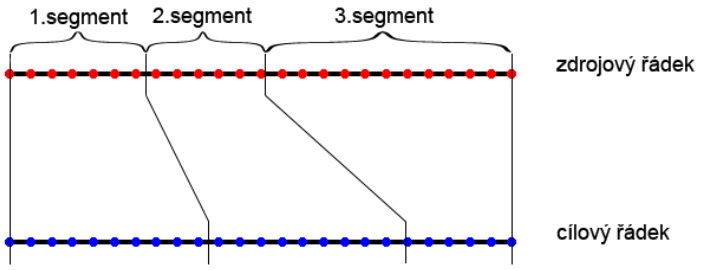
3. Obdobně se vypočítají průsečíky daného řádku s vertikálními čárami počáteční mřížky. Získaly se tak dva řádky rozdělené na stejný počet segmentů. U jednoho z řádků známe i hodnoty jeho vzorků (viz obr. 3.3).

4. Nyní už stačí například lineárně přemapovat z původních segmentů do nových.

5. Celý postup se zopakuje pro vertikální směr s tím, že se místo zdrojové mřížky použije mezi-mřížka, místo mezi-mřížky cílová mřížka a místo zdrojového obrazu získaný mezivýsledek.

Výše popsaný způsob vede na mřížku čtyřúhelníků, na jejichž hranách vznikají "zlomy" v obrazu. Tomu lze zabránit použitím interpolačních křivek (tvořících splajn od začátku do konce řádku resp. sloupce). Pro tyto účely se dají použít například Lagrangeovy křivky, Catmull-Rom splajny aj. Principielně se tedy postup s použitím splajnů nijak neliší od použití navazujících úseček, jen se zkvalitní výstup. Pochopitelně to je za cenu trochu složitějších výpočtů spojených s počítáním splajnů.

Pokud by se měla celkově zhodnotit tato metoda, musela by se vyzdvihnout její rychlost. Naopak, i když je ovládání mřížky jednoduché, jen velice těžko se dosahuje toho, aby každý



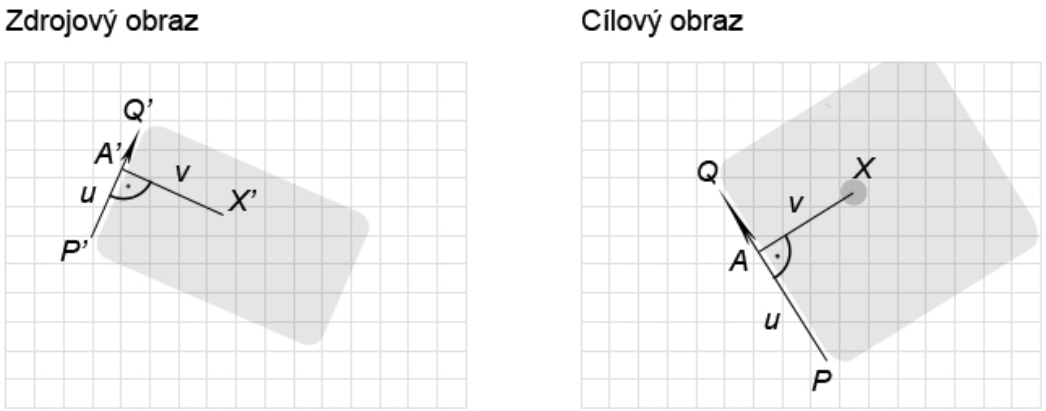
Obrázek 3.3: Schematické znázornění segmentace zdrojového a cílového řádku

uzel byl přesně na svém místě – obzvlášť pokud se editor snaží držet jednotlivé segmenty mřížky konvexní (nutné pro zcela korektní zobrazení). Velice trefně to bylo popsáno ve zdroji [10], kde to přirovnávali ke snaze tlačit lano. Vždy zbydou místa s nevyužitými uzly a současně místa, kde by se hodilo uzlových bodů více.

3.5 Warpovací jádro Feature-based morfingu

Zajímavějším způsobem proměny jednoho objektu v jiný je v každém případě morfing na základě páru úseček (resp. jeho zobecnění pro libovolný počet párů). Tuto techniku představil ve své práci roku 1992 Thaddeus Beier [1].

Základní princip spočívá v umístění dvou orientovaných úseček – jednu do zdrojového, druhou do cílového obrázku tak, aby vyznačovaly odpovídající rys (odtud název). Pro každý bod v cílovém obraze se zjišťuje jeho relativní pozice (specificky definovanou – viz dále) vzhledem k úsečce a pomocí zpětné mapovací funkce se najde odpovídající bod ve zdrojovém obrazu. Pro snadnější pochopení nejdřív popíšu případ, kdy je definován jen jeden pár úseček.



Obrázek 3.4: Základní princip feature-based warpingu

Na začátku je třeba vysvětlit výpočet relativní pozice bodu vzhledem k orientované

úsečce. K tomu využiju obrázek 3.4. K určení se používá dvou parametrů. Parametr v udává absolutní vzdálenost (v pixelech) bodu X od přímky určené body P a Q (krajní body úsečky; rovnice 3.2). Parametr u určuje poměr délek úseček AP ku QP (rovnice 3.1). Bod A je přitom průsečíkem přímky dané body P a Q s její kolmicí procházející bodem X.

$$u = \frac{\overrightarrow{PX} \cdot \overrightarrow{PQ}}{|\overrightarrow{PQ}|^2} \tag{3.1}$$

$$v = \frac{|\overrightarrow{PX} \times \overrightarrow{PQ}|}{|\overrightarrow{PQ}|} = \frac{\overrightarrow{PX} \cdot Perp(\overrightarrow{PQ})}{\overrightarrow{PQ}} \tag{3.2}$$

$$X' = P' + u \cdot \overrightarrow{P'Q'} + \frac{v \cdot Perp(\overrightarrow{P'Q'})}{|\overrightarrow{P'Q'}|} \tag{3.3}$$

$Perp(\vec{x})$ je funkce vracející vektor kolmý k vektoru $\vec{x}$. Navíc tento musí vracet konzistentně buď po, nebo proti směru hodinových ručiček. Rovnice 3.3 pak vyjadřuje výpočet polohy odpovídajícího vzorku ve zdrojovém obrazu. P' je "startovní bod", ke kterému se přičte nejdřív u-násobek vektoru $\overrightarrow{P'Q'}$ a následně k němu kolmý vektor o délce v. Jedná se tedy o vlastní mapovací funkci, kterou už stačí jen převést do programovacího jazyka.

Prozatím se mohla v obraze nacházet pouze jedna dvojice orientovaných úseček. V praxi tato situace umožňuje pouze tři základní transformace celého obrazu – otočení, změnu měřítka (a to jen ve směru orientované úsečky) a posunutí. Taková situace je samozřejmě naprosto nevhodná pro morfing, proto je nutné výpočet zobecnit pro libovolný počet dvojic.

První prerekvizitou k zobecnění je vyjádření nově vypočtené pozice jako vektor posunutí D_i vzhledem k pozici původní pro všechny dvojice:

$$D_i = X'_i - X \tag{3.4}$$

V této chvíli, kdyby se udělal jednoduchý průměr těchto vektorů, výsledek by sice zahrnoval všechny vektory, deformace by ale neprobíhala podle očekávání. Pro korektní transformaci je třeba zohlednit tři parametry:

1. vzdálenost bodu od úsečky
2. délku dané úsečky
3. rychlost úbytku vlivu dané úsečky se vzdáleností

Tím získáme váhový koeficient w pro daný vektor posunu:

$$w = \left(\frac{delka^p}{a + vzdalenost}\right)^b \tag{3.5}$$

kde

- *delka* je proměnná, která udává délku orientované úsečky v cílovém obraze
- proměnná *vzdalenost* udává vzdálenost daného bodu od úsečky. Proto v případě, kdy $u \in \langle 0; 1 \rangle$, je tato rovna v a můžeme ji spočítat podle rovnice 3.2 (v praxi se znovu nepočítá a jen se převezme už vypočtená hodnota). Pro $v > 1$ se $delka = |\overrightarrow{QX}|$ a naopak v případě, že $v < 0$ se $delka = |\overrightarrow{PX}|$.

- parametr a udává sílu dané úsečky v bezprostředním okolí – čím více se a blíží nule, tím větší vliv má úsečka na blízké body (při $a = 0$ má úsečka pro body ležící přímo na ní nekonečnou sílu).
- b je parametr zahrnující právě třetí bod z výčtu požadavků na váhovou funkci
- p udává míru zohlednění délky úsečky

Ve chvíli, kdy je znám výpočet váhy a vektoru posunu, stačí pro daný pixel počítat sumu těchto veličin. Po přičtení posledních v daném bodě, se suma vektorů podělí součtem vah – získáme výsledný vektor posunu, jehož přičtením k pozici bodu v počítaném obraze dostaneme pozici požadovaného vzorku.

A právě tady je jedna z největších slabin feature-based morfingu, respektive jeho warpovacího jádra. Je jí obrovská náročnost výpočtu, která vyplývá z nutnosti pro každý bod spočítat relativní pozici a váhu vzhledem ke každému vektoru! Přitom se při každém jednom počítají dvě neceločíselné mocniny a několik dělení. Jinými slovy doba výpočtu lineárně roste s počtem bodů a k tomu lineárně roste s počtem zadaných dvojic přitom i výpočet pro jednu dvojici je složitý. Další, už ne tak zásadní nevýhodou, jsou někdy neočekávané deformace (spojené například s křížením úseček).

Naopak výhodou je velká volnost při definici transformace. Vyznačují se jen ty vlastnosti, které animátor chce; do jednoho místa lze vložit prakticky libovolné množství topologických dat (narozdíl třeba od postupu při warpování sítě čtyřúhelníků, kde je síť předem daná). Celkově tedy čas navíc strávený při závěrečném renderování snímků stojí za plynulejší a o něco přirozenější přeměnu (pochopitelně stále závisí především na kvalitě práce animátora).

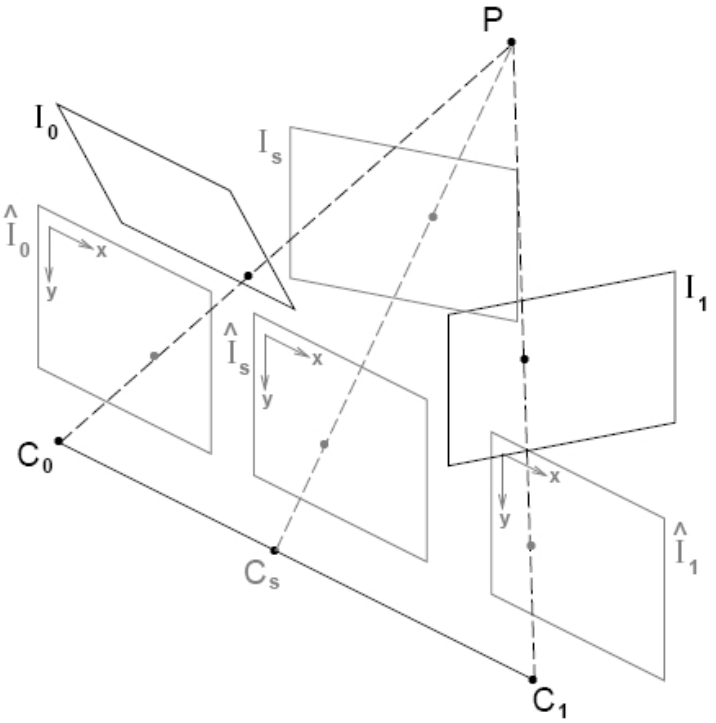
Obrázek 3.5: Ukázka feature-based morfingu. Krajní obrázky jsou vstupní. Výsledný obrázek (uprostřed) vznikl warpingem obou do času $t = 0,5$. V obrázcích jsou ponechány úsečky, které se na výsledku podílely.

3.6 Warpovací techniky View morfingu

Metoda view morfingu vybočuje z řady klasických technik, jejichž jediným cílem bylo udělat plynulou tvarovou změnu. Při tvorbě této techniky si autor uvědomil, že cílem by nemělo být jen vytvořit plynulou změnu, ale udělat ji *věrohodnou*. Například proměnu portrétů dvou osob jde udělat poměrně snadno, ale realističnost přeměny rychle mizí v případě, že jsou vyfoceny z jiného úhlu.

Podstata view morfingu tkví ve snaze kompenzovat různý úhel pohledu na morfovaný objekt v počátečním a koncovém snímku. K tomu je třeba dodat informaci o vzájemné

poloze snímků. Způsob jakým se to udělá je libovolný – nejčastěji vyznačením jednoduchých geometrických objektů v začátečním a cílovém obrazu.



Obrázek 3.6: Schéma view morfingu.

Jak je v oboru obvyklé, i tady lze postup nejsnáz vysvětlit graficky. V daném případě se jedná o obrázek 3.6, který byl převzat z ustavující práce tohoto rozšíření morfovacích technik [6].

Jádro view morfingu má tři fáze:

1. Prewarp fáze – vstupní obrazy I_0 a I_1 se promítnou do společné roviny. Vzniknou tak dva nové obrazy $\hat{I}_0$ a $\hat{I}_1$.

2. Morfing – provede se pomocí kterékoli dříve zmíněné, nebo i jiné, techniky morfování. Vstupem jsou předwarpované obrazy $\hat{I}_0$ a $\hat{I}_1$; výstupem je obraz $\hat{I}_S$, který je koplanární s $\hat{I}_0$ a $\hat{I}_1$

3. Postwarp fáze. Obraz $\hat{I}_S$ je transformován tak, aby odpovídal místě na trajektorii pohledu z počátečního do cílového obrazu. Tím vznikne požadovaný mezisnímek I_S.

Jak je vidět, všechny přidané transformace obrazu, které jsou specifické právě pro view morfing, mají čistě lineární charakter a jsou složeny z elementárních transformací.

Vzhledem k tomu, že zde byl postup uveden jen jako nástin možného příštího rozšíření demonstračního programu, nebudou zde popsány ani matematický základ, ani praktické zkušenosti. Více informací lze nalézt v dříve uvedené práci Stevena Seitze a Charlese Dyera [6].

Kapitola 4

Program

Tato kapitola je zaměřena na popis programu, který je součástí bakalářské práce. V kapitole 4.1 popíšu stanoviska a úvahy, které stály při návrhu. Následovat bude popis implementace (kapitola 4.2). V poslední části bude zevrubně naznačeno ovládání.

4.1 Návrh

Před začátkem návrhu bylo zapotřebí shrnout zadané požadavky na program.

- program má implementovat techniky warpingu a morfingu
- má se jednat o uživatelsky jednoduchý program

A to bylo vše. Jednalo se tedy o velice volnou definici vlastností, kterou bylo nutno upřesnit. První rozhodnutí bylo udělat tento program pod operačním systémem MS Windows XP. Následně byl za vývojové prostředí vybrán *Borland C++ Builder*. Tento výběr byl ovlivněn v první řade velice dobrou spoluprací C++ Builderu s knihovnou *DigILib*. Jedná se o knihovnu pro zpracování obrazu vyvinutou na domovské fakultě FIT VUT v Brně, která byla vytvořena právě jako nadstavba výše zmíněného integrovaného vývojového prostředí. Lehkou nepříjemností v průběhu vývoje aplikace bylo zmizení domovských webových stránek této knihovny. Proto bohužel ani nemůžu dát odkaz.

Vzhledem k tomu, že pro skutečnou práci s grafikou jsou k dispozici profesionální programy, rozhodl jsem se koncipovat program spíš jako demonstrační ukázku, než plnohodnotný grafický editor. To pochopitelně umožnilo jednodušší návrh uživatelského rozhraní.

Z nejdůležitějších bylo rozhodnutí, které techniky budou implementovány. Není pochopitelně časově únosné naprogramovat všechny. Zvolil jsem tedy cestu implementace tří různých morfovacích procesů, se kterými souvisí dvě warpovací jádra. První je nejstarší způsob využívaný ještě před dobou počítačů – jednoduché prolnutí snímků. Druhým, pro počítačovou grafiku důstojnějším, je morfing pomocí pravidelné sítě čtyřúhelníků (mesh morphing). Tento je pro změnu jedním z nejstarších užívaných na počítačích. Nejtěžším vybraným byl morfing soustavy dvojic úseček (feature-based morphing).

Posledním cílem, který byl v této fázi stanoven, bylo jasné oddělení funkcí spojených s warpingem a morfingem od uživatelského rozhraní. To by případně v budoucnu mělo umožnit jednodušší úpravu a rozšíření programu.

4.2 Implementace

V průběhu této kapitoly budou popsány jednak základní struktury, na kterých jsem stavěl, logiku programu a nakonec implementace některých klíčových algoritmů. Před začátkem je nutné se zmínit o externích knihovnách, které byly použity. První z nich je už v předchozí kapitole zmíněný *DigILib*. Ten poskytl důležité struktury a ovládací prvky. Knihovna avi_utils Luciana Wischika [7] hraje podružnější roli – stará se vytvoření AVI souboru z jí předaných snímků.

Protože je vytvářen de facto grafický editor, byť velmi úzce specializovaný, základní objekt musí být schopný zobrazení obrazu. K tomuto účelu výborně posloužil ImageControl – ovládací prvek DigILibu. Součástí tohoto prvku je i struktura ImageStruct, která dokáže uchovat bitmapový obraz společně s informacemi o jeho vlastnostech (rozměry, krok, způsob uložení,...). Pomocí tohoto prvku se tedy jednoduše vyřešil nejen problém uložení, ale i zobrazení snímků.

Dalším problémem byl vstup a uložení topologických dat. Opět byly s výhodou využity ovládací prvky DigILibu – tentokrát pro změnu GraphicsListControl. Jedná se o objekt, který mimo jiné zachytává zprávy o pohybu myši. Analogicky k ImageControlu, který udržuje obraz, tento obsahuje seznam grafických objektů (body, čáry, obdélníky,...), které zobrazuje.

Kombinací ImageControlu a GraphicsListControlu "přilepených" na sebe vznikl základ pro další práci. Ostatní objekty uživatelského prostředí jsou už vcelku běžná tlačítka, posuvníky, nabídky a jiné. Tyto nemá cenu dále rozebírat, protože se jedná o běžné objekty událostmi řízeného uživatelského prostředí.

Raději popíšu některé funkce z knihovny *ImageWM* vytvořené speciálně pro tento projekt, zaměřující se na warping a morfing. Ty vykonávají požadované techniky transformace obrazu. Samy jsou pak volány v reakcích na vzniklé události.

```
void FBMWarpCore(ImageStruct* IS_in, ImageStruct* IS_out,
                 GraphicsListStruct* GLS_start, GraphicsListStruct* GLS_end,
                 float a, float b, float p);
```

Funkce provádějící feature-based warping. `IS_in` je struktura se vstupním obrázkem, `IS_out` je inicializovaná struktura, do které se uloží výstup. `GLS_start` obsahuje topologická data vstupního a `GLS_end` požadovaného obrázku. `a`, `b` a `p` jsou parametry pro výpočet vah orientovaných úseček (více viz kapitola 3.5). Tato verze provádí vzorkování nejbližším sousedem, takže je o něco rychlejší, a proto vhodná pro výpočty náhledu. Pro závěrečné vykreslení je určena obdobná funkce `FBMWarpCoreBilinear()` provádějící bilineární vzorkování.

```
void TransformLineGLCs(GraphicsListStruct* inA, GraphicsListStruct* inB,
                       GraphicsListStruct* out, float ratio);
```

Předpokládá se, že `inA`, `inB` a `out` jsou inicializované struktury stejné velikosti, z nichž `inA` obsahuje úsečky představující topologickou informaci vstupního obrázku a `inB` výstupního. Do `out` se uloží jejich interpolace pro normovaný čas `ratio` (0 odpovídá času vstupního a 1 výstupního obrazu).

```
void MBWarpCore(ImageStruct* IS_in, ImageStruct* IS_out,
                GraphicsListStruct* GLS_start, GraphicsListStruct* GLS_end,
                int MeshXSize, int MeshYSize);
```

Jedná se o ekvivalentní funkci k `FBMWarpCoreBilinear()` s tím, že implementuje warping sítě čtyřúhelníků. V `GLS_start` a `GLS_end` jsou uloženy počáteční resp. cílové pozice uzlů. `MeshXSize` udává horizontální počet uzlů a `MeshYSize` vertikální.

```
void TransformPointGLCs(GraphicsListStruct* inA, GraphicsListStruct* inB,
                        GraphicsListStruct* out, float ratio);
```

Opět ekvivalentní funkce tentokrát k `TransformLineGLCs()`. Tato verze je určena k interpolaci sítě bodů.

```
ImageStruct* Blending(ImageStruct* IS_A, ImageStruct* IS_B, float ratio);
```

Funkce prolne obrazy uložené ve strukturách `IS_A` a `IS_B` v poměru `ratio`. Předpokládá se stejné rozlišení obrazů. Návratovou hodnotou je ukazatel na strukturu s nově vzniklým obrazem.

Právě různou kombinací těchto základních funkcí provádím jak warping tak morfing. Na následujícím příkladu ukážu celý kód pro feature-based morfing:

```
TransformLineGLCs(GLCA->List, GLCB->List, GLCOut->List, timeRatio);
FBMWarpCore(Original_A, Transformed_A, GLCA->List, GLCOut->List,
            FBWeightA, FBWeightB, FBWeightP);
FBMWarpCore(Original_B, Transformed_B, GLCB->List, GLCOut->List,
            FBWeightA, FBWeightB, FBWeightP);
ICOut->Image = Blending(Transformed_A, Transformed_B, crossRatio);
```

Tyto čtyři příkazy jsou skoro všechno, co je nutné pro naprosto funkční proměnu (jedná se o úryvek zdrojového kódu, který je skutečně použit v přiloženém programu).

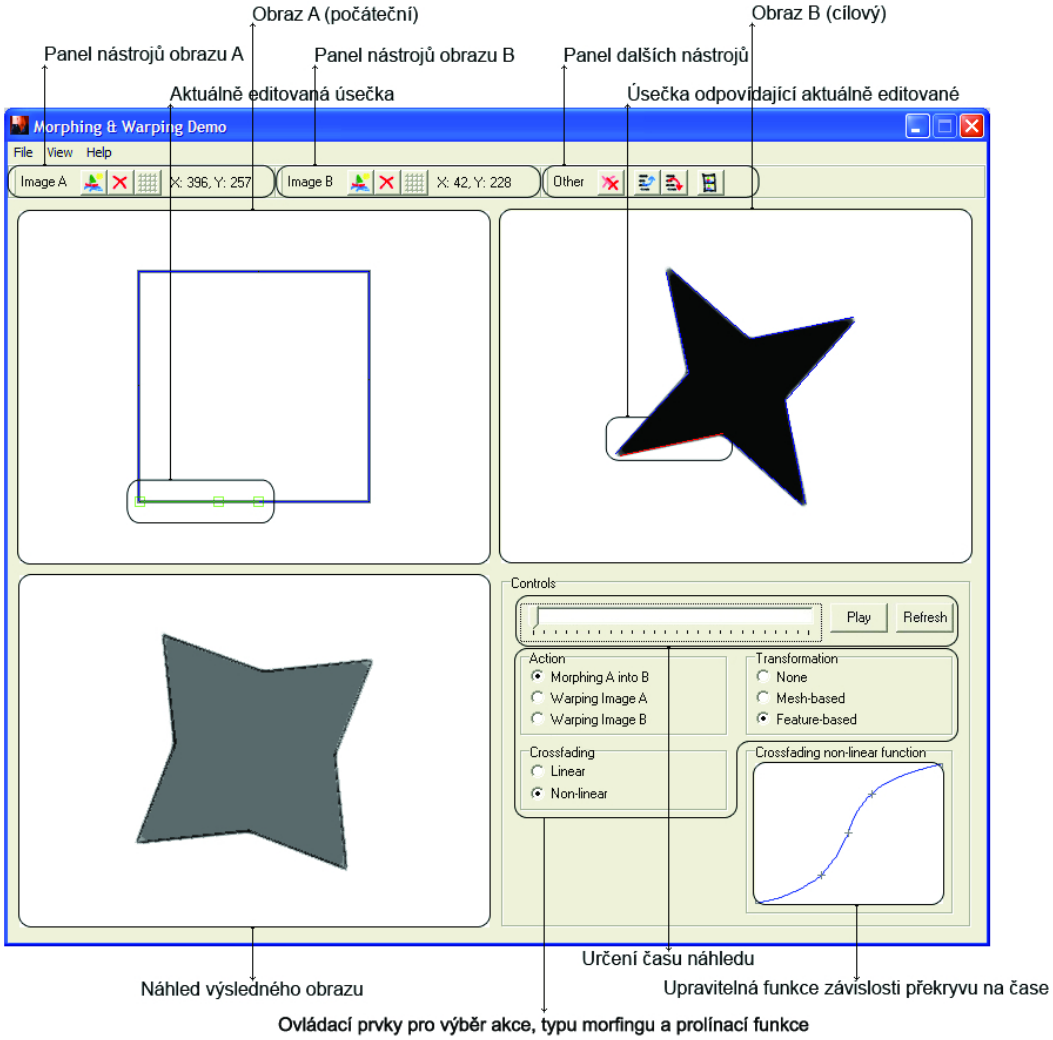
Při implementaci bylo nutno udělat i pár kompromisů. Jedním z nich je omezování pohybu uzlu při síťovém morfingu. V první verzi byla zabudována striktní kontrola, která zamezovala vzniku nekonvexního čtyřúhelníku ve kterémkoli místě mřížky. To sice vedlo k naprosto korektnímu obrazu, na druhou stranu to ale silně omezovalo při editaci. Proto bylo toto omezení zrušeno (ve zdrojovém kódu jej lze stále zakomentované nalézt). Tím ale vznikaly nepřehlédnutelné grafické chyby. Nakonec tedy byla zvolena "zlatá střední cesta" a pohyb byl omezen na plochu v obdélníku rovnoběžném se souřadnými osami a vymezeném čtyřmi uzly, na které pohybovaný přímo navazuje.

Dalším kompromisem je grafické znázornění vektorových dat (tzn. sítě čtyřúhelníků resp. orientovaných úseček). Byl použit vzhled tak, jak jej definuje přímo DigILib. To v některých situacích znamená, že objekty můžou splývat s pozadím a jsou proto špatně viditelné.

Posledním problémem je rychlost vykreslování – týká se hlavně a především feature-based morfingu. Tento algoritmus je inherentně velmi složitý, a proto i na poměrně rychlých počítačích (testováno na Core 2 Duo T7200) může při velkém počtu úseček trvat i více jak deset sekund, než se vykreslí náhled. Obecně všechny algoritmy pro práci s obrazem by bylo vhodné zoptimalizovat pro použití rozšířených instrukcí procesoru (MMX, nebo spíše SSE-SSE4). Navíc by nebyl problém udělat podporu pro vícejádrové procesory – zejména pro dvoujádrové by rozšíření bylo poměrně triviální, přitom vysoce efektivní záležitostí.

4.3 Ovládání

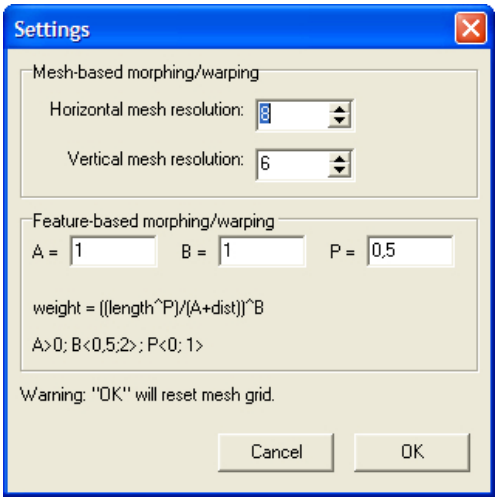
Už před vlastní implementací byla snaha navrhnout jednoduché a účelné uživatelské rozhraní. I když výsledek zdaleka není dokonalý, myslím, že je program intuitivní a tvorba efektů přímočará. Obrázek 4.1 ukazuje stav programu v průběhu editace pomocí feature-based morfingu.



Obrázek 4.1: Screenshot programu s popisky vysvětlujícími funkčnost jednotlivých ovládacích prvků.

Pod nabídkami jsou tři panely nástrojů. První se vztahuje k obrazu A (počáteční), druhá k obrazu B (koncový) a třetí obsahuje nástroje, které se buď vztahují k oběma obrazům (smazaní všech zadaných čar) případně ke stavu celého programu (uložení, nahrání a tvorba videa). Výběr funkce probíhá v rámu *Controls*.

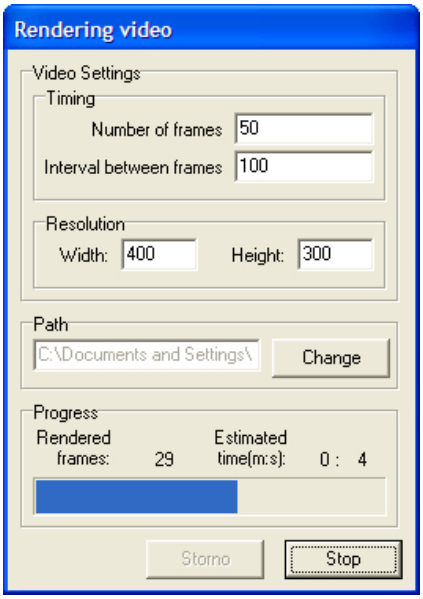
Některé méně používané parametry jsou schovány v nabídce View→Settings. Zde lze upravovat jednak rozměry mřížky mesh morfingu a taky parametry určující výpočet váhy pro feature-based morfing.



Obrázek 4.2: Dialogové okno nastavení.

Video je krom náhledu jediným výstupem programu. Jeho tvorba se provádí pomocí dialogového okna AVI Output. K tomuto dialogu se lze dostat dvěma způsoby. Buď přes nabídku File→Create video, nebo pomocí panelu ostatních nástrojů, na kterém je odpovídající tlačítko.

Vytvořeno bude vždy takové video, jak jsou nastaveny ovládací prvky pro výběr akce, typu morfingu a prolínací funkce v hlavním okně programu.

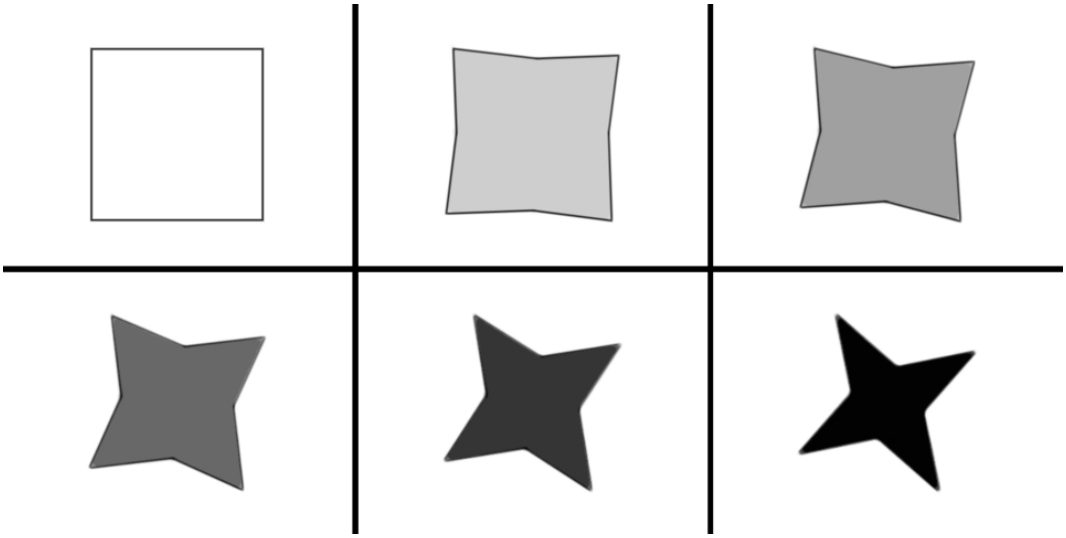


Obrázek 4.3: Dialogové okno pro tvorbu videa.

Podrobněji je ovládání popsáno v uživatelské příručce, která je uložena společně s programem na přiloženém mediu.

4.4 Ukázky přeměn

Následující obrázky jsou vytvořeny pomocí demonstračního programu(na přiloženém mediu jsou mimo jiné uložené videoukázky):



Obrázek 4.4: Přeměna dvou jednoduchých geometrických útvarů.

Obrázek 4.5: Přeměna jednoho obličeje v druhý.

Kapitola 5

Závěr

V oblasti warpingu a morfingu toho bylo napsáno za posledních dvacet let poměrně mnoho. Z velké části se ale lze setkat jen s velice úzce zaměřenými pracemi, které popisují například výhradně jen jeden postup. Přesto je tato oblast už vcelku zasycená a jen těžko se u základů hledají místa, kde by se dalo toto paradigma nějak rozšířit.

Tato práce se pokusila shrnout některé používanější postupy, objasnit jejich základ a nabídnout možnost vyzkoušet si jejich výsledky na demonstračním programu, který je její součástí. Navíc byla snaha o trochu odlišné rozdělení témat, kdy se jednoznačně ukázalo, že většina morfovacích algoritmů se liší jen svým jádrem tvarové změny.

Detailně je popsána a s pomocí obrázků a pseudokódu ukázána metoda morfingu. Následně se přešlo k warpingu a od základů jako elementární transformace se přešlo k jádru. Některé části byly jen naznačeny, důležitější však podrobně vysvětleny. K těm zásadním patří v programu použité warpovací jádra síťového a úsečkového morfingu.

Neoddělitelnou součástí je připojený demonstrační program. Právě v něm je velká část popsaných postupů implementována. Program umožňuje jednoduchou uživatelem řízenou tvarovou změnu jak pomocí čtyřúhelníkové sítě, tak pomocí soustavy dvojic úseček. Je pochopitelně možné si vybrat mezi samotnou tvarovou deformací a kompletním procesem morfingu. Byl provozován a testován na procesorech rodiny x86 a jedná se o jednovláknovou aplikaci.

A právě do tohoto místa směřuje jedno z navržených rozšíření – optimalizace. Tato by mohla být udělána buď konzervativním způsobem – zabudováním podpory pro výpočet ve více vláknech a využitím rozšiřujících instrukčních sad x86 procesorů, jako jsou SSE-SSE4. Nebo mnohem zajímavějším a méně konvenčním přístupem by mohlo být převedení výpočtu z CPU na grafické akcelerátory. V dnešní době se čím dál více jádra grafických karet začínají podobat plnohodnotným procesorům. Sice jednotlivé proudové procesory jsou mnohem slabší než systémové CPU, ale jejich masivní paralelismus zaručuje v některých případech (jakým je i tento) o řád vyšší rychlost výpočtu.

Co se týká textové části, do budoucna by se mohla rozrůst o popis dalších warpovacích a morfovacích technik. K vhodným kadetům patří například jen lehce naznačený morfing s kompenzací úhlu pohledu nebo tri-view morfing.

Na úplný závěr by bylo vhodné zdůraznit, že i když se nejednalo o výzkumnou práci, své ovoce přinesla. Jednak ve formě shrnujícího textu základů warpingu a morfingu a poté i ve formě ukázkového programu.

Literatura

[1] Beier, T.: *Feature-Based Image Metamorphosis*. Computer Graphics, 1992.

[2] Dobšík, M.: *Morfing*. FIT VUT v Brně, 1997.

[3] FIT VUT v Brně: Signály a systémy. `https://www.fit.vutbr.cz/study/courses/index.php?id=83`, 2007, garantem předmětu je Černocký Jan, doc. Dr. Ing.

[4] FIT VUT v Brně: Základy počítačové grafiky. `https://www.fit.vutbr.cz/study/courses/index.php?id=93`, 2007, garantem předmětu je Kršek Přemysl, Ing., Ph.D.

[5] Žára, J., Beneš, B., Felkel, P.: *Moderní počítačová grafika*. Computer Press, 1998, ISBN ISBN 0-7226-049-9.

[6] Steven M. Seitz, C. R. D.: *View Morphing*. Department of Computer Sciences, University of Wisconsin-Madison, 1996.

[7] Wischik, L.: AVI example code. `http://www.wischik.com/lu/programmer/avi_utils.html`, 2002.

[8] WWW stránky: Wikipedia. `http://wikipedia.org`, 2007.

[9] WWW stránky: A Critical History of Computer Graphics and Animation. `http://accad.osu.edu/ waynec/history/lessons.html`, Naposled navštíveno 11.5.2007.

[10] WWW stránky: 2D Image Morphing Algorithms. `http://davis.wpi.edu/ matt/courses/morph/2d.htm`, Naposled navštíveno 6.5.2007.

[11] WWW stránky: Image Compositing and Morphing. `http://www.cs.princeton.edu/courses/archive/fall00/cs426/lectures/composite/`, Naposled navštíveno 6.5.2007.

[12] WWW stránky: Image Warping. `http://www.cs.princeton.edu/courses/archive/fall00/cs426/lectures/warp/`, Naposled navštíveno 6.5.2007.

Dodatek A

Zdrojové kódy vybraných algoritmů

Kód feature-based warpingu použitý v demonstračním programu:

```
//feature-based morphing warp core; point-sampled output
//IS_in  - structure with input image
//IS_out - pre-initialized structure where the output image will be stored
//GLS_start - topological information for input image
//GLS_end   - topological information for output image
//a, b, p   - parametres of the weight function
void FBMWarpCore(ImageStruct* IS_in, ImageStruct* IS_out,
                 GraphicsListStruct* GLS_start, GraphicsListStruct* GLS_end,
                 float a, float b, float p)
{
  float u, v, pqsabs, dist, weight;
  float XS_X, XS_Y,
        pxd_X, pxd_Y,
        pqd_X, pqd_Y,
        pqs_X, pqs_Y,
        DSum_X, DSum_Y, WSum;
  LineStruct *LS_src, *LS_dst;

  for(int j = 0; j < IS_out->YSize; j++)
  {
    for(int i = 0; i < IS_out->XSize; i++)
    {
      DSum_X = 0;
      DSum_Y = 0;
      WSum = 0;

      //for each pixel is calculated its relative position to the vector
      //based on that, weights of the vectors are calculated
      //the shift that each vector generate is added to the sum
      //and is divided by the sum of wights - here we get the final shift
      for(int k = 0; k < GLS_start->Count; k++)
```

```
{
  LS_src = GetLineFromList(GLS_start, k);
  LS_dst = GetLineFromList(GLS_end, k);

  pqs_X = LS_src->X2 - LS_src->X1;
  pqs_Y = LS_src->Y2 - LS_src->Y1;

  pxd_X = i - LS_dst->X1;
  pxd_Y = j - LS_dst->Y1;
  pqd_X = LS_dst->X2 - LS_dst->X1;
  pqd_Y = LS_dst->Y2 - LS_dst->Y1;

  //calculate u,v
  //perpendicular[X,Y] = [-Y, X] ... convention(...for this function)
  u = (pxd_X*pqd_X + pxd_Y*pqd_Y)/(pqd_X*pqd_X + pqd_Y*pqd_Y);
  v = (pxd_X*(-pqd_Y) + pxd_Y*pqd_X)/sqrt(pqd_X*pqd_X + pqd_Y*pqd_Y);

  //calculate X'i
  pqsabs = sqrt(pqs_X * pqs_X + pqs_Y * pqs_Y);
  XS_X = LS_src->X1 + u*pqs_X + v*(-pqs_Y)/pqsabs;
  XS_Y = LS_src->Y1 + u*pqs_Y + v*pqs_X/pqsabs;

  //dist = shortest distance from X to PiQi
  if(u < 0) dist = sqrt((i - LS_dst->X1)* (i - LS_dst->X1)
                   + (j - LS_dst->Y1)*(j - LS_dst->Y1));
  else if(u > 1) dist = sqrt((i - LS_dst->X2)*(i - LS_dst->X2)
                        + (j - LS_dst->Y2)*(j - LS_dst->Y2));
  else dist = abs(v);


  //weight =  ((length^p)/(a+dist))^ b
  //a - determines how strong will be objects on/very close
  //    to line affected
  //b - determines how the relative strenght of the lines
  //    falls with distance
  //p - determines how lenght will affect the strength of the line
  weight = pow(pow(pqs_X*pqs_X + pqs_Y*pqs_Y, p)/(a + dist),b);

  //DSUM
  DSum_X += (XS_X-i) * weight;
  DSum_Y += (XS_Y-j) * weight;

  //WSUM
  WSum += weight;
}

WSum = (WSum == 0) ? 1 : WSum;
```

```
      XS_X = i + DSum_X/WSum;
      XS_Y = j + DSum_Y/WSum;

      //checking position of sample
      XS_X = (XS_X > IS_in->XSize-1) ? (IS_in->XSize - 1) : XS_X;
      XS_Y = (XS_Y > IS_in->YSize-1) ? (IS_in->YSize - 1) : XS_Y;

      XS_X = (XS_X < 0)? 0 : XS_X;
      XS_Y = (XS_Y < 0)? 0 : XS_Y;

      //taking the sample(nearest-point)
      SetImageRGBPixelRGB(IS_out, i, j,
                          ImageRGBPixelRGB(IS_in, (int)XS_X, (int)XS_Y));
    }
  }
}
```

Kód mesh warpingu použitý v demonstračním programu (je použit dvouprůchodový algoritmus):

```
//IS_in  - structure with input image
//IS_out - pre-initialized structure where the output image will be stored
//GLS_start - topological information for input image
//GLS_end   - topological information for output image
//MeshXSize, MeshYSize - horizontal and vertical resolution of mesh
void MBWarpCore(ImageStruct* IS_in, ImageStruct* IS_out,
                GraphicsListStruct* GLS_start, GraphicsListStruct* GLS_end,
                int MeshXSize, int MeshYSize)
{
  GraphicsListStruct GLS_mid;
  ImageStruct* IS_mid;
  PointStruct node;
  PointStruct *node_start, *node_end;
  PointStruct *node_up, *node_down,
              *node_left, *node_right;
  float ratio, position,
        bInPrev, bInNext,
        bOutPrev, bOutNext;
  unsigned C0, C1;

  IS_mid = NewImageRGB(IS_in->XSize, IS_in->YSize);
  InitListSize(&GLS_mid, MeshXSize*MeshYSize);

  //calculate coordinates of ‘‘mid-mesh’’
  for(int i = 0; i < MeshXSize*MeshYSize; i++)
  {
    node_start = GetPointFromList(GLS_start, i);
    node_end = GetPointFromList(GLS_end, i);
    SetPoint(&node, node_end->X, node_start->Y);
```

```
  AddPointToList(&GLS_mid, &node);
}

//first pass - horizontal stretch
for(int j = 0; j < IS_in->YSize; j++)
{
  int i = 0;
  int u = 0;
  bInPrev = 0;
  bOutPrev = 0;
  bInNext = 0;
  bOutNext = 0;

  while(i < IS_in->XSize-1)
  {
    int v = 0;                    //u - columns, v - rows
    //find the intersecting line of the mesh and the current pixel row
    do
    {
      node_up = GetPointFromList(&GLS_mid, v*MeshXSize+u);
      node_down = GetPointFromList(&GLS_mid, (v+1)*MeshXSize+u);
      v++;
    }while(node_down->Y < j);

    //find the next point of intersection in mid and start image
    bOutPrev = bOutNext;
    bOutNext = wavg(node_up->X, node_down->X,
                    (1-(j-node_up->Y)/(node_down->Y-node_up->Y)));
    node_up = GetPointFromList(GLS_start, (v-1)*MeshXSize+u);
    node_down = GetPointFromList(GLS_start, v*MeshXSize+u);
    bInPrev = bInNext;
    bInNext = wavg(node_up->X, node_down->X,
                   (1-(j-node_up->Y)/(node_down->Y-node_up->Y)));

    //while still within limits calculate from where should be taken sample
    while(i <= bOutNext)
    {
      ratio = (float)(i-bOutPrev)/(bOutNext-bOutPrev);
      position = bInPrev + ratio*(bInNext-bInPrev);
      C0 = ImageRGBPixelRGB(IS_in, (int)floor(position), j);
      C1 = ImageRGBPixelRGB(IS_in, (int)ceill(position), j);
      SetImageRGBPixelRGB(IS_mid, i, j,
                          SimpleBlending32(C0,C1,(position-floor(position))));
      i++;
    }
    u++;
  }
}
```

```
  //second pass - the vertical stretch
  for(int i = 0; i < IS_in->XSize; i++)
  {
    int v = 0;
    int j = 0;
    bInPrev = 0;
    bOutPrev = 0;
    bInNext = 0;
    bOutNext = 0;

    while(j < IS_in->YSize-1)
    {
      int u = 0;
      //find the intersecting line of the mesh and current pixel column
      do
      {
        node_left = GetPointFromList(GLS_end, v*MeshXSize+u);
        node_right = GetPointFromList(GLS_end, v*MeshXSize+u+1);
        u++;
      }while(node_right->X < i);

      //find the next point of intersection in final and mid image
      bOutPrev = bOutNext;
      bOutNext = wavg(node_left->Y, node_right->Y,
                      (1-(i-node_left->X)/(node_right->X-node_left->X)));
      node_left = GetPointFromList(&GLS_mid, v*MeshXSize+u-1);
      node_right = GetPointFromList(&GLS_mid, v*MeshXSize+u);
      bInPrev = bInNext;
      bInNext = wavg(node_left->Y, node_right->Y,
                     (1-(i-node_left->X)/(node_right->X-node_left->X)));

      //while still within limits calculate from where should be taken sample
      while(j <= bOutNext)
      {
        ratio = (float)(j-bOutPrev)/(bOutNext-bOutPrev);
        position = bInPrev + ratio*(bInNext-bInPrev);
        C0 = ImageRGBPixelRGB(IS_mid, i, (int)floor(position));
        C1 = ImageRGBPixelRGB(IS_mid, i, (int)ceill(position));
        SetImageRGBPixelRGB(IS_out, i, j,
                            SimpleBlending32(C0,C1,(position-floor(position))));
        j++;
      }
      v++;
    }
  }
  DeleteImage(IS_mid); DestroyList(&GLS_mid);
}
```