N-bodová deformácia obrazu využitím deformačného modelu ARAP

Správa k semestrálnej práci predmetu MI-DZO

Tomáš Fedor
fedortom@fit.cvut.cz

13. mája 2015

Abstrakt

Táto správa popisuje implementáciu As-Rigid-As-Possible (ARAP) deformácie obrazu vytvorenej v rámci predmetu MI-DZO.

1 Úvod

As-Rigid-As-Possible deformácia 2D obrazu náchádza uplatnenie najmä u ilustrátorov, grafikov a animátorov ktorí pracujú s kreslenými postavičkami. Grafik už nemusí kresliť každý frame animácie zvlášť ale stačí ak poskytne niekoľko základných póz a tento algoritmus môže použiť na inbetweening. V oveľa kratšom čase a oveľa jednoduchšie tak dokáže vytvoriť plynulú animáciu.

V semestrálnej práci predmetu MI-DZO som sa zameral len na interaktívnu transformáciu obrazu použitím ARAP modelu. Algoritmus samotný sa dá ale rozšíriť a použiť napríklad pri shape matchingu alebo auto-paintingu.

2 Algoritmus

Cieľom algoritmu je deformovať obrázok z pôvodnej (*source*) pózy do cieľovej (*target*), pri zachovaní čo možno najvernejšieho zobrazenia vzhľadom k pôvodnému tvaru. To znamená, že počas deformácie by nemalo dochádzať k zmene mierky ani k vzniku artefaktov na rôznych častiach obrázku. Užívateľ by mal nadobudnúť pocit, že pracuje s objektom, ktorý má takmer fyzikálne vlastnosti z reálneho sveta, deformuje sa predvídateľne. Deformácia by sa dala prirovnať k manipulácii s gumovým objektom, ktorý sa vždy snaží vrátiť do svojho pôvodného tvaru (zachováva rigiditu).

Deformácia obrazu neprebieha priamo nad každým pixelom obrázku, ale nad vrcholmi „embedding lattice", čo je mriežka (alebo matica) namapovaná na obrázok. Pri zmene polohy vrcholov mriežky dochádza k deformácii tak, že na všetky body v jednom štvorci mriežky sa aplikuje rovnaká transformácia. Transformácia medzi štvorcami sa však môže líšiť.

Samotná úprava mriežky ale nezabezpečí zachovanie konzistentného tvaru. Užívateľ môže dynamicky definovať kotrolné body a určiť im novú polohu, čím

poruší rigiditu obrazu. Zachovanie konzistentného tvaru dosiahneme *regularizáciou* mriežky, čiže nájdením takej transformácie, ktorá minimalizuje vzdialenosť medzi vrcholmi štvorca mriežky pôvodného tvaru a nového nastavenia jeho vrcholov. Každý štvorec mriežky tak musí udržovať informáciu o originálnom nastavení.

Toto riešenie umožňuje efektívnu implementáciu, ktorá produkuje výborné výsledky. Zmenou veľkosti štvorcov mriežky škálujeme problém. Platí, že čím hustejšia sieť (teda menšie štvorce), tým presnejšia a prirodzenejšia ale za to pomalšia deformácia.

Celý algoritmus je iteratívny a počas celého behu dovoľuje užívateľovi dynamicky meniť nastavenie cieľovej polohy:

1. vytvorenie mriežky
2. opakuj:
   (a) nastavenie cieľovej polohy – presun vrcholov matice asociovaných s kontrolnými bodmi na požadované umiestnenie
   (b) regularizácia mriežky – zachovanie čo možno najkonzistentnejšieho tvaru
   (c) prekreslenie obrázku

2.1 Embedding Lattice

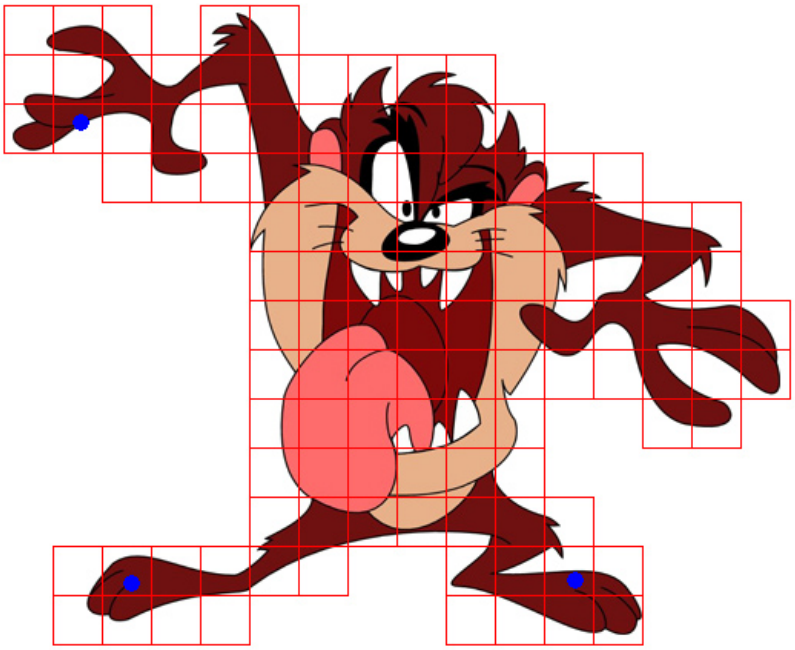
Každý štvorec mriežky musí vedieť o svojej pôvodnej polohe, pretože tá je použitá počas regularizácie. Vzniknú tak oddelené štvorce v rámci mriežky, ktoré sa môžu rotovať a posúvať bez toho, aby ovplyvnili inú časť mriežky (princíp *coupled bodies*). Mriežka teda určuje dva typy štvorcov: štvorec skladajúci sa zo štyroch vrcholov matice (tieto vrcholy pokrývajú obrázok a užívateľ s nimi môže manipulovať cez kontrolné body) a *rigídne štvorce*, ktoré určujú pôvodnú polohu daného štvorca matice. Každý vrchol matice si udržuje informáciu o vrchole príslušného rigídneho štvorca. Keďže rigídne štvorce nie sú medzi sebou priamo prepojené, každý vrchol mriežky má informáciu o 1 až 4 vrcholoch rigídnych štvorcov. Modifikácia vrcholu mriežky nijak nezmení polohu „rigídnych" štvorcov. Na obrázkoch v tomto texte je matica zobrazená červenou farbou a rigídne štvorce modrou.

Pre zefektívnenie deformácie má tiež zmysel nastaviť jednotlivým vrcholom mriežky rôzne váhy v závislosti od vzdialenosti od kontrolného bodu. Vrcholy s vyššími váhami sa dostanú k cieľovej pozícii rýchlejšie.

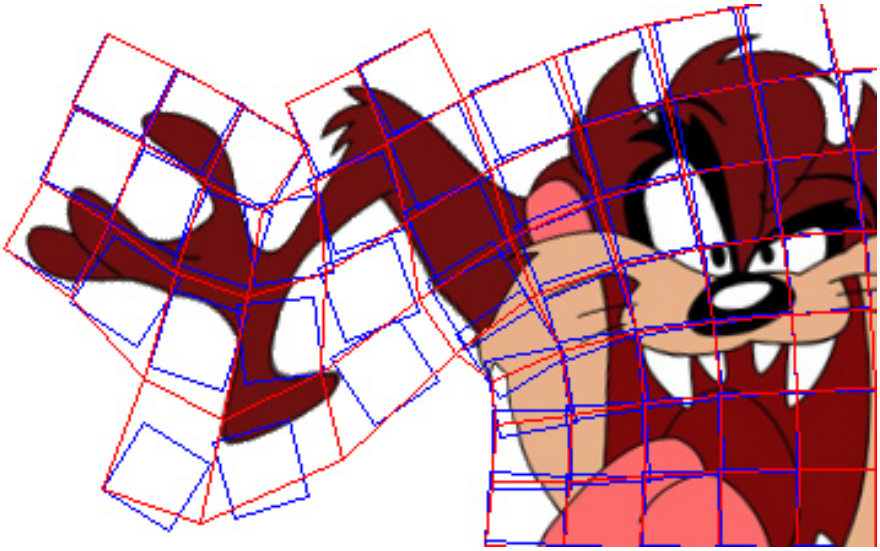
2.2 Regularizácia

Regularizácia zabezpečuje lokálnu rigiditu obrazu, pretože modifikácia mriežky neberie žiaden ohľad na zachovanie tvaru. Cieľom je nájsť takú rotáciu a transláciu rigídnych štvorcov, ktoré majú čo najmenšiu vzdialenosť k asociovaným vrcholom matice.

V 2D priestore existuje analytické riešenie[2] [1]. V nasledujúcich rovniciach platí, že všetky vektory sú dvojzložkové riadkové vektory súradníc x a y, $\mathbf{p}_c$ a $\mathbf{q}_c$ je centroid pôvodnej, resp. cieľovej polohy a $\perp$ značí kolmý vektor. Obecne $\mathbf{p}$



Obr. 1: Vykreslená mriežka cez obrázok a definované kontrolné body



Obr. 2: Detail mriežky a jednotlivých rigídnych štvorcov po regularizácii a presune vrcholov matice do centroidu vrcholov pridružených rigídnych štvorcov

bude odkazovať na vrcholy rigídneho štvorca (pôvodná poloha), $\mathbf{q}$ bude odkazovať na vrcholy mriežky (cieľová poloha) a w_i je váha vrcholu matice. Ďalej platí

$$\hat{\mathbf{p}}_i = \mathbf{p}_i - \mathbf{p}_c$$

$$\hat{\mathbf{q}}_i = \mathbf{q}_i - \mathbf{q}_c$$

Optimálnu rotáciu dosiahneme následovne:

$$\mathbf{R}* = \frac{1}{\mu} \sum_i w_i \begin{pmatrix} \hat{\mathbf{p}}_i \\ \hat{\mathbf{p}}_i^{\perp} \end{pmatrix} (\hat{\mathbf{q}}_i^T \; \hat{\mathbf{q}}_i^{\perp T})$$

$$\mu = \sqrt{\left(\sum_i w_i \hat{\mathbf{q}}_i \hat{\mathbf{p}}_i^{\top} \right)^2 + \left(\sum_i w_i \hat{\mathbf{q}}_i \hat{\mathbf{p}}_i^{\perp T} \right)^2}$$

Po vypočítaní rotácie už transláciu dosiahneme jednoducho

$$\mathbf{t}^{*T} = \mathbf{p}_c^T - \mathbf{R}^* \cdot \mathbf{q}_c^T$$

Po aplikovaní rotácie a translácie dostaneme novú polohu rigídnych štvorcov a je ešte poterbné modifikovať polohu vrcholov matice. Nová poloha každého vrcholu bude centroid všetkých pripojených vrcholov rigídnych štvorcov.

Pri definovaní novej cieľovej polohy sa začnú vrcholy asociované s kontrolnými bodmi pohybovať rýchlo k požadovanému nastaveniu a budu so sebou „ťahať" zbytok obrázku, ktorý sa bude snažiť zachovať si svoj tvar. Spočiatku bude obrázok pomerne flexibilný no čím bližšie budú kontrolné body k cieľovej polohe, tym rigídnejším sa stane a posuny vrcholov matice už nebudú také dramatické.

Ak by užívateľ odstránil všetky kontrolné body, obraz sa vráti do svojej pôvodnej polohy až na globálnu rotáciu a posun.

2.3 Projekcia obrázku

Po úprave mriežky je potrebné prepočítať a prekresliť obrázok. Pre každý štvorec matice sa z pôvodného obrázku (a jeho matice) a nového nastavenia matice vypočíta inverzná homografia a aplikuje sa na každý pixel daného štvorca. Na resampling je možné použiť metódu najbližšieho suseda, kedy sa výsledná farba z pôvodného obrázku získa zaokrúhlením súradníc 3 alebo bilineárnou interpoláciou 5, ktorá je pomalšia, no za to zbavuje výsledný obrázok „kockovaných" hrán.

3 Implementácia

Za implementačný jazyk som zvolil Python, kvôli požiadavku na interaktivitu aplikácie. Python ponúka jednoduché knižnice pre rýchle vytvorenie GUI a manipuláciu s obrázkami. Navyše sa s rozšírením *numpy* často používa na matematické výpočty a umožňuje písať časti programu v jazyku C, ak je rýchlosť výpočtu kritická. Na vytvorenie GUI som použil knižnicu *TkInter* a na manipuláciu s obrázkom *Pillow*.

V nasledujúcich subsekciách sa pokúsim poukázať na zaujímavejšie časti implementácie a riešenie niektorých problémov.

Obr. 3: Resampling použitím metódy najbližšieho suseda

3.1 Maska

Pri načítaní obrázku sa vytvorí maska ako 2D boolean pole, kde hodnotu *true* majú body objektu a hodnotu *false* body pozadia. Na určenie pozadia som použil farbu ľavého horného pixelu obrázku s toleranciou 10 v oboch smeroch, čo sa na testovaných obrázkoch ukázalo ako dostatočné. Maska bola použitá jednak na vytvorenie embedding lattice, jednak pri projekcii a aplikovaní homografie.

Aj napriek tomu, že sa maska vypočítava iba raz na začiatku programu bol výpočet implementovaný v jazyku C.

3.2 Embedding Lattice

V mojej implmentácii som za veľkosť štvorca matice zvolil 32 pixelov (šírka). S týmto nastavením program produkoval relatívne dobré výsledky a zároveň bol celý výpočet zvládnutení v reálnom čase bez výraznejších poklesov na plynulosti. Pre umiestnenie embedding lattice sa najprv vypočíta bounding box z masky obrázku a následne sa box vytvorí tam, kde existuje aspoň jeden pixel popredia.

Váhy vrcholov matice sa prepočítavajú pri každom pridaní alebo odobraní kontrolného bodu (mapovaného na najbližší vrchol). Váha kontrolného bodu bola nastavená na 10000 a lineárne sa znižovala so vzdialenosťou v matici. Samotný výpočet pripomína *Breadth-First Search*, ktorý začína z každého vrcholu asociovaného ku kontrolnému bodu. Výsledná váha vrcholu je maximum ktoré sa dá týmto prechodom dosiahnúť.

Obr. 4: Resampling použitím bilineránej interpolácie

3.3 Projekcia

Homografia je počítaná s využitím solveru knižnice *numpy*, ktorá poskytujé rozhranie ku skompilovanému kódu z jazyka C. Následné aplikovanie homografie však z výkonnostných dôvodov prebieha v C. Homografia je aplikovaná na každý bod v štvorci, ktorý je definovaný štyrmi vrcholmi. Keďže skoro nikdy nedosiahneme geometrický štvorec s paralelnými hranami je potreba nájsť hranice štvorca.

Pre nájdenie hraníc som použil *algoritmus Bresenhamovej čiary* používaný na rasterizáciu čiary definovanej dvomi bodmi. Na všetky body medzi nájdenými hranicami sa aplikuje homografia, ktorej výsledkom budú súradnice na pôvodnom obrázku. Ak by sme však použili resampling naozaj na každý bod medzi týmito hranicami, deformovali by sme aj časť pozadia, ktoré je obsahom niektorých hraničných štvorcov matice. Resampling sa teda aplikuje iba pre tie pixely, ktorých maska je nastavená na *true* a teda ide o objekt. Na kontrolu masky som použil metódu najbližšieho suseda a na samotný resampling bilineárnu interpoláciu.

3.4 Optimalizácia

Dodatočnou optimalizáciou je obmedzenie projekcie. Vzhľadom na to, že projekcia obrázku je výpočetne najpomalšia časť celého programu, a nie je nutné prekresľovať obrázok po každej iterácii (pretože užívateľ si to pod istou hra-

Obr. 5: Maska obrázku, červené okolie bolo identifikované ako pozadie

nicou aj tak nevšimne) obmedzil som projekciou tak, že sa snažím dosiahnuť 30 snímkov za sekundu. Lepším riešením by bolo sledovať zmenu vrcholov matice po regularizácií a prekresľovať obrázok vždy keď je zmena väčšia než určitý limit, na ktorého implementáciu už nebol dostatok času.

4 Záver

Výsledkom mojej práce je funkčná aplikácia, ktorá užívateľovi umožňuje načítať obrázok a dynamicky s ním manipulovať tak, že si obrázok zachováva čo možno najlepšie svoj pôvodný tvar, rigiditu. Program som odtestoval na niekoľkých rôznych obrázkoch.

Aj napriek tomu, že som nestihol implementovať všetko, čo som chcel, som so svojou prácou spokojný. Nezostal mi čas na dostatočné ošetrenie projekcie obrazu, keď ho užívateľ nastaví mimo zobrazenú plochu. Kvôli spôsobu akým počítam hranice pri projekcii sa môžu začať vykresľovať artefakty, v horšom prípade môže nastať pád aplikácie.

Ďalším rozšírením by bol lepší výpočet masky, ktorý by umožnil rozdeliť obraz na viac komponent. V mojej implementácii sa rozoznáva iba popredie a pozadie no jednoduchou úpravou by bolo možné nájsť všetky súvislé komponenty a umožniť tak užívateľovi manipulovať s viacerými objektami v obraze osobitne.

Podobnou úpravou by bola aj lepšia manipulácia so štvorcami, ak je ich obsah predelený pozadím. V takom prípade by bolo výhodné vytvoriť niekoľko kópií štvorca v matici tak, aby každý spracovával iba jednu časť obrázku.

Literatúra

[1] Scott Schaefer, Travis McPhail, and Joe Warren. Image deformation using moving least squares. In *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, pages 533–540, New York, NY, USA, 2006. ACM.

[2] Daniel Sýkora, John Dingliana, and Steven Collins. As-rigid-as-possible image registration for hand-drawn cartoon animations. In *Proceedings of International Symposium on Non-photorealistic Animation and Rendering*, pages 25–33, 2009.

A Výsledky