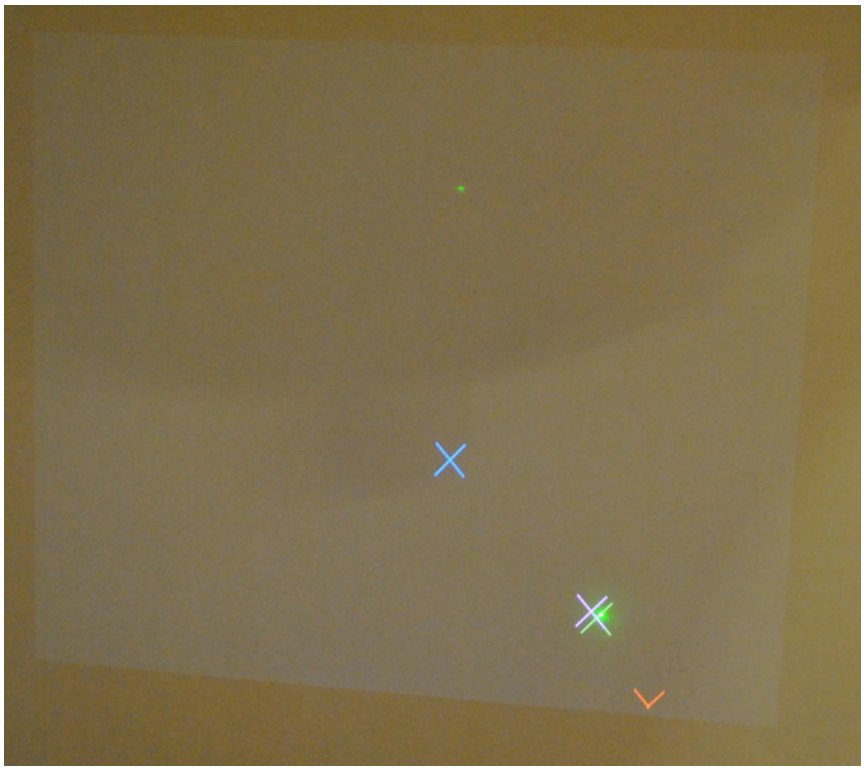
Počítačové zpracování obrazu

Projekt Učíme se navzájem

Tomáš Pokorný, Vojtěch Přikryl
Jaroška • 17. září 2010



Obsah

Abstrakt

Projekt Počítačové zpracování obrazu si klade za cíl pomocí kamery a projektoru připojeného k počítači umožnit uživateli, aby laserového ukazovátka mohl používat jako ukazatele na objekty zobrazené na projektoru. Aplikace je napsána v Objective C. Funkce aplikace jsou tyto: kalibrace webkamery pro dané umístění, zachycení a načtení obrazu z webkamery, analýza a hledání nasvíceného bodu, transformace snímaného obrazu pomocí transformační funkce a kalibračních dat a finální vykreslení nasvíceného bodu na projektoru. Rozpracováno je zpřesnění transformace obrazu a v budoucnu jsou plánovány aplikace a hry tento software využívající.

Začátky

M&M

Na soustředění korespondenčního semináře M&M jsem se s touto problematikou setkal v rámci tzv. konfery, což byla několika odpolední práce s následnou prezentací výsledků. Zde jsme dostali již předpřipravenou část softwaru pro snímání obrazu a pro promítání výsledků a dále jsme se zabývali zpracováním tohoto obrazu a jeho správným vykreslováním. Zde vznikly všechny dosud používané transformace a velká většina aktuálního kódu.

PŮVODNÍ CÍLE

Zadání znělo takto:

> *Cílem tématu je vybrat vhodnou detekovanou charakteristiku obrazu, vymyslet algoritmus na detekci vlastnosti v obrazu a pak naprogramovat a testovat na počítači s webkamerou, možná v kombinaci s projektorem.*

Cíl je možné si vybrat, možnosti jsou např. detekce barevné tečky, sledování více teček či trajektorie pohybu tečky či jiného objektu. Bude potřeba z obrazu odstranit pozadí a perspektivní zkreslení. K tomu bude potřeba i geometrické počítání a transformace spolu s měřením či odhadem toho zkreslení. Pokud se povede dobrá kalibrace a odstranění zkreslení, může se promítat zpětná vazba přímo na snímané objekty.

Původně jsme mysleli, že zpracování půjde snadno a že se za chvíli dostaneme k detekci pohybu a vykreslování čáry, ale ukázalo se, že transformace není vůbec jednoduchá a tak jsme u ní prakticky skončili, částečně vlivem její obtížnosti, částečně vlivem začátečnické chyby.

PROGRAMOVÉ VYBAVENÍ

Na M&M jsme pracovali na Linuxovém notebooku, ke kterému byla připojena webkamera a projektor, snímání probíhalo pomocí MPlayeru, ke zpracování sloužil Python s PIL a Tk pro zobrazování výsledků.

Nynější vybavení je notebook s Mac OS X vybavený integrovanou webkamerou iSight, připojovaný k projektoru. Snímání je realizováno pomocí frameworku CocoaSequenceGrabber v programu HledaniBodu. Tento program je napsán v Objective C a v něm rovněž probíhá hledání nejsvětlejšího bodu, transformace jeho souřadnic a jeho vykreslení na projektor.

PRVNÍ KROKY

Ze začátku jsme se učili pracovat s Pythonem, protože jsme ho nikdo pořádně neuměli, a poté jsme přistoupili k vlastnímu zpracování obrazu. Zjišťovali jsme, jak obrázek vypadá, co se v něm všechno dá najít, jak je charakteristický nasvícený bod a zkoušeli jsme ho hledat. Poté přišla na řadu transformace obrazu z kamery, na které jsme strávili dlouhý čas. Vzniklo několik konkurenčních algoritmů, avšak žádný nefungoval nijak zvlášť přesvědčivě. Nakonec se nám podařilo zprovoznit dvě transformace s relativně dobrými výsledky a úspěšně zprovoznit vykreslování.

Teorie

SNÍMÁNÍ A ZPRACOVÁNÍ OBRAZU

Obraz je snímán pomocí frameworku CocoaSequenceGrabber v programu HledaniBodu. Program snímá obrázky tak, jak jsou poskytovány kamerou a hledá v nich nasvícený bod. Při zpracování je vždy obraz převeden na pole bytů, nad kterým se pak provádí jednotlivé metody.

Čtvercová metoda

V této metodě se vždy počítá součet plochy 5x5 pixelů okolo zkoumaného bodu po jednotlivých barevných složkách. Před tímto výpočtem můžou být nastaveny omezující podmínky, kdy je zbytečné výpočet provádět, protože jsou body příliš tmavé než aby na ně bylo svíceno.

Z jednotlivých blokových součtů je zjištěno maximum hodnoty součtu hledané barevné složky a definitivně je vybrán ten bod, který má tento součet nejvyšší. U tohoto maxima jsou ještě kontrolovány součty jednotlivých složek, aby se omezily artefakty kdy pokud na projektor nesvítím se nezachytávaly zobrazené křížky.

Tato metoda se ukázala být poměrně spolehlivá, ale je potřeba najít vhodné nastavení konstant pro zahazování bodů, protože obzvláště při denním osvětlení či osvětlení různými zdroji světla jsou snímané body různých barev a je potřeba co nejvíce špatných zahodit, ale nezahodit s nimi i ty, co chceme najít.

Křížová metoda

Tato metoda je ořezanou verzí metody, o níž jsem dlouho uvažoval, ale je implementačně náročná. Ta je založena na faktu, že hledáme bod, tzn. nějaké malé osvícené kolečko a tudíž má střed a několik bodů v okolí výrazně jiné vlastnosti než zbytek obrázku. Ideální stav je hledat 2 mezikruží, což je ale poměrně obtížné a tak jsem v této metodě použil jen výrazně zjednodušenou verzi.

V této verzi je sledována pouze barva právě zkoumaného bodu P, jeho nejbližších sousedů I a poté 4 bodů O vzdálených 5 pixelů na

každou ze 4 stran od bodu P. Body I, které jsou blízko bodu P ještě mají podobné zabarvení jako zkoumaný bod a naopak body O jsou již dostatečně daleko, takže se na nich bod P již neprojevuje. Z bodů I a P je získán vždy průměr jednotlivých barevných složek ze všech 4 bodů.

U každé skupiny bodů je možné nastavovat maximální a minimální hodnoty jasu jednotlivých složek, což dává poměrně silný nástroj k odfiltrování nežádoucích bodů. Tato metoda se ukázala spolehlivější než čtvercová, protože dokáže odfiltrovat velké jasné plochy.

KALIBRACE

Kalibrace je prvním z dějů souvisejících s transformací. Cílem kalibrace je zjistit, v jakých bodech na snímaném obraze se zobrazují rohy obrazu zobrazovaného dataprojektorem. To je důležité proto, abychom dokázali přepočítat bod nalezený na obrazu webkamery na bod, který zobrazíme na projektoru. Nyní k funkci kalibrace. Program postupně rozsvítí 4 čtverečky v rozích obrazu projektoru a vždy chvíli počká, aby se kamera přizpůsobila, pak je sejme již popsaným způsobem a najde je. Tímto získáváme 4 kalibrační body, se kterými již můžeme počítat v transformacích.

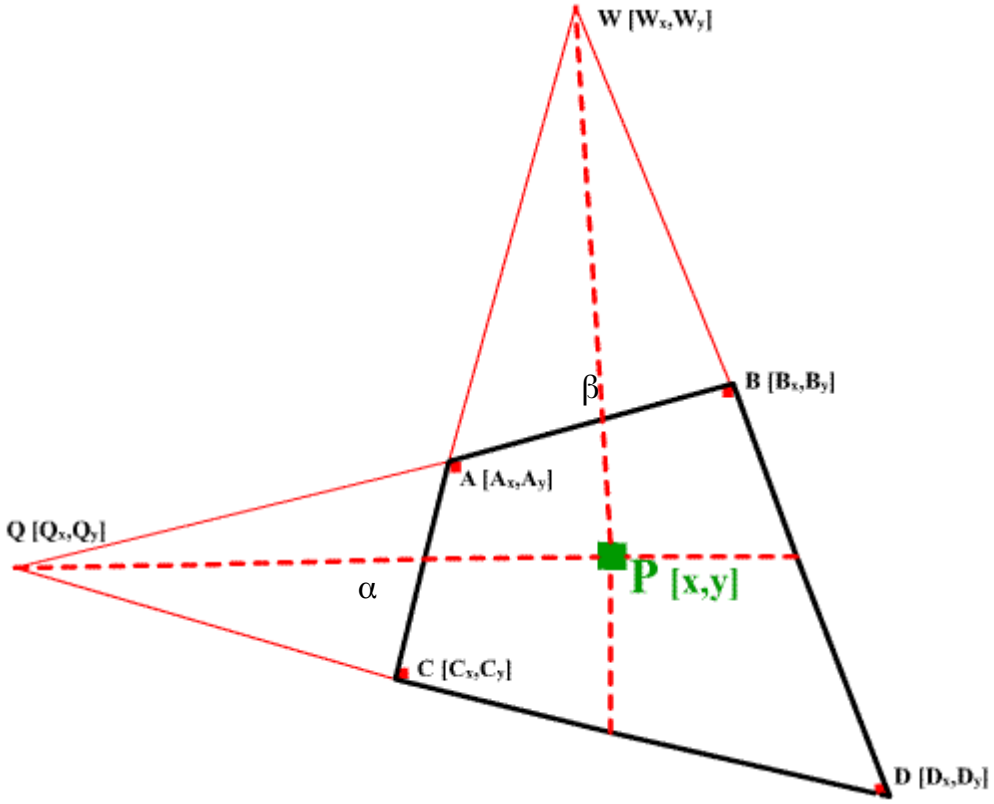
TRANSFORMACE

Toto je nejdůležitější a matematicky nejobtížnější část celého projektu, která zajišťuje správné zobrazení bodu na projektoru. Co to je a proč je potřeba?

Předpokládejme, že máme projektor, který promítá na plátno obdélníkový obraz. Pokud by ho promítal zkreslený, má své vlastní funkce ke korekci tohoto zkreslení. Pokud nyní obraz snímáme kamerou, získáme nějaký útvar. Optimální situace nastává, pokud kamera snímá ze stejného bodu, jako promítá projektor. V tom případě je snímaný útvar obdélník se stejným poměrem stran k promítanému obrazu. Tady nám stačí úplně základní transformace a to jednoduché přinásobení konstantou odpovídající poměru délky strany snímaného a promítaného obrazu. Toto je ale situace, která nastane velmi zřídka. Většinou snímáme kamerou obraz zkreslený, protože pokud si představíme to, co snímá kamera, jde o jakýsi nekonečně vysoký čtyřboký jehlan.

Pokud snímáme útvar mimo osu projektoru, bude snímaný útvar určitým řezem daného jehlanu, tudíž to bude docela obecný čtyřúhelník. My potřebujeme tento čtyřúhelník roztáhnout vhodně tak, abychom získali obdélník, který bude odpovídat obdélníku promítanému projektorem, abychom mohli tento obdélník zobrazit. To je právě záležitost transformace

Transformace se 2 body



Toto je nová transformace vymyšlená Lukášem Langerem a je v současnosti používaná. Základem je to, že při zachycení zkresleného obrazu se původně rovnoběžné přímky určené rohovými body A, B, C, D budou sbíhat a tudíž se protnou v nějakém bodě (Q, W). Tento bod jednoduše nalezneme prodloužením stran. Tímto bodem prochází i přímka určená zachyceným bodem P a rovnoběžná se stranou. Přímka procházející zachyceným bodem P a rovnoběžná se stranou AC nebo AB bude při zkreslení procházet nalezeným průsečíkem. Nalezneme průsečík této přímky se stranou AB nebo AC, nazveme ho α a β a určíme poměr části k průsečíku k celé této straně. Toto provedeme pro obě rovnoběžné strany a získané poměry stačí vynásobit šířkou a výškou zobrazovaného obrazu. Tato transformace dává nejlepší výsledky s přesností okolo 5cm.

Poměrová transformace

Toto byla nejlepší námi naprogramovaná transformace. Základem je fakt, že i ve zkresleném útvaru zůstanou zachovány poměry vzdáleností a to speciálně poměrů vzdáleností nalezeného osvíceného bodu od stran. Pokud tuto úvahu zobecníme na dva rozměry, získáme rovnice, pomocí kterých můžeme získané souřadnice nalezeného bodu přepočíst na souřadnice bodu na obrazovce.

Grafická transformace

Tato transformace se uplatnila hlavně v začátcích. Je postavena na funkci z knihovny PIL, která umí převést zadaný čtyřúhelník na obdélník. To je přesně to, co potřebujeme, ovšem má to jednu poměrně zásadní nevýhodu. Protože je to funkce z grafické knihovny, znamená to, že tuto transformaci umí provést pouze s obrázkem, nikoli jen daty. Je tedy potřeba vytvořit obrázek, zakreslit do něj nalezený bod, obrázek transformovat a opět transformovaný bod nalézt. To jsou hlavní 2 důvody, proč je transformace pomalá, což je její hlavní nevýhoda. Musí provést transformaci bitmapy, tzn. přepočítat každý pixel a pak my ji ještě musíme projít, abychom transformovaný pixel nalezli. Navíc je to transformace závislá na Pythonu, při přepsání transformace do Objective C musela být tato transformace vypuštěna.

Při průzkumu zdrojových kódů PIL byly nalezeny rovnice, které se používají pro tuto transformaci, v současnosti jsou snahy ji zprovoznit. Je implementována dle zdrojových kódů, ale nechová se tak, jak má.

Další transformace

Během vývoje vznikly ještě 2 další transformace a to transformace přibližná, která vždy předpokládala, že útvar je lichoběžník a to postupně v obou rozměrech a výsledek pak zprůměrovala. Toto byla používaná transformace v začátcích, než jsme přišli na chybu v transformaci poměrové. Se stoupajícím zkreslením rapidně klesá přesnost.

Druhá ze vzniklých transformací byla založena na úhlech a postupném posouvání kalibračních bodů do vrcholů promítaného čtyřúhelníka. Při nasimulování její činnosti ale bylo zjištěno, že byly úvahy chybné a tato transformace nemůže fungovat.

ZOBRAZENÍ

Poté, co je nalezen bod, který se má vykreslit na projektoru, ho již stačí jenom vykreslit. Obraz se vykresluje do černého okna o velikosti obrazu promítaného projektorem. Tato část nepotřebuje nijaké zvláštní komentáře, jediné, na co je třeba si dávat pozor, je, abychom křížek v místě vykreslovaného bodu vykreslovali správnou barvou, kterou nevyhodnotí program jak nasvětlený bod ukazovátkem.

Kreslení

Další možností je kreslení ukazovátkem. V tomto případě se přepočítaný bod uloží do Beziérovy křivky, která se při každém nově příchozím bodu opět vykreslí. Pokud se bod nepodaří nalézt, tak se čeká, než dojde nějaký platný bod a na tento bod se kreslení přesune

bez toho, aby kreslilo čáru. Tímto se tedy dají kreslit i přerušované čáry. Opět je nutné dávat pozor na barvu vykreslování. Kresba se dá resetovat a dá se zvolit tloušťka čáry.

Implementace

Celý proces a vzájemná volání metod jsou zakreslená v grafu v souboru schema_kompletni.vue a pdf.

PROTOKOLY

V celém programu se na všechny činnosti spojené se zpracováním obrazu používají protokoly. Každá nově vzniklá třída, která má vykonávat jednu ze základních činností, musí implementovat daný protokol, aby mohla být do programu zařazena. Hlavní program volá právě ty metody, které jsou zaprotokolovány a žádné jiné. Pro případné nastavování parametrů se používá kontrolérů, které fungují nezávisle na hlavním programu.

`ProcessProtocol`

Tento protokol standardizuje činnosti spojené se zpracováváním obrazu.

```
-(id)initWithSize:(NSSize)aSize;
```

Inicializuje danou třídu, dostává jako parametr rozlišení obrázku, tudíž si může spočítat délku bytového pole.

```
-(NSPoint)getThePointFromImage:(NSImage *)anImage;
```

Tato metoda je volána při každém hledání bodu v obraze. Přijímá jako parametr obrázek, který se má zpracovat a vrací bod, jehož souřadnice jsou normalizovány tak, že nejvíce vlevo či nahoře je 0 a nejvíce vpravo či dole je 1. Tímto je zajištěna nezávislost na rozlišení kamery. Všechny body, které se předávají mezi třídami, jsou takto normalizovány.

`TransformProtocol`

Tento protokol standardizuje transformaci nalezeného bodu.

```
-(id)initWithCalibrationArray:(NSArray *)calArray;
```

Inicializuje danou třídu. Tato metoda je volána při spuštění programu před tím, než je poslán první bod k transformaci. Jako parametr dostane pole 4 `ZOPointů`, kdy každý reprezentuje jeden kalibrační bod braný po směru hodinových ručiček.

```
-(void)setCalibrationArray:(NSArray *)modCalArray
```

Tato metoda je volána vždy po dokončení kalibrace. Parametr je stejný jako u minulé metody, dojde k nastavení kalibračních konstant dané transformace.

`-(NSPoint)transformPoint:(NSPoint)point;`

Tato metoda je volána při každé transformaci obrazu, přijímá jako parametr normalizovaný NSPoint a vrací opět normalizovaný NSPoint, který je transformovaný. Pokud dostane jako parametr bod s nulovou souřadnicí, transformaci neprovede a vrátí bod tak, jak jej dostala.

`ProjectorProtocol`

Tento protokol standardizuje zobrazování transformovaných bodů a další činnosti okolo.

`-(void)setPoint1:(NSPoint)aPoint;`

Přijímá jako parametr transformovaný normalizovaný bod a ten poté vykresluje.

`-(void)setPoint2:(NSPoint)aPoint;`

Přijímá jako parametr transformovaný normalizovaný bod a ten poté vykresluje.

`-(void)goFullscreen;`

Tato metoda po svém zavolání zajistí, přechod panelu do fullscreen zobrazení na druhém monitoru s rozlišením 800x600 pixelů. Pokud není druhý monitor připojen, tak se nic neděje.

`-(void)leftFullscreen;`

Tato metoda po svém zavolání zajistí, aby se zrušilo fullscreen zobrazení na druhém monitoru. Pokud není zapnut fullscreen režim, nic se neděje.

SNÍMÁNÍ OBRAZU

`CocoaSequenceGrabber`

Ke snímání je použit framework `CocoaSequenceGrabber`, který po příchodu každého snímku volá metodu `cameraDidReceiveFrame` ve `WindowController`. Tato metoda je srdcem celého programu, protože postupně volá metody jednotlivých tříd vedoucí k nalezení bodu, transformaci jeho souřadnic a vykreslení.

ZPRACOVÁNÍ OBRAZU

`ZOProcessImage`

```
-(id)initWithSize:(NSSize)aSize;
```

Inicializuje objekt, spočítá délku pole, nastaví velikost do vnitřní proměnné.

```
-(NSPoint)getThePointFromImage:(NSImage *)anImage;
```

Obraz je převeden na pole bytů, z nichž každý udává jednu složku barvy pixelu. Program proto skáče po 4 prvcích pole (složky obrazu jsou "RGBA") a pro každý pixel spočítá pomocí metody `getSumSquareAtIndex` součet jednotlivých složek ve čtverečku 5x5. Každý pixel je ověřován, jestli má dostatečný jas a pokud nemá, sumační čtverec se nepočítá.

Poté se hledá nejjasnější sumační čtverec. V druhém průchodu se obarvují body, které byly zhozeny, protože v prvním průchodu by to vadilo při počítání sumačních čtverců. Filtrování je i na výstupu "čtverečkovací" metody, kontroluje se minimální jas nejjasnějšího sumačního čtverce. Nakonec se index převede na souřadnice pomocí `pixelCoordinatesAtIndex` a normalizuje.

```
-(void)sumSquareAtIndex:(int)index toArray:(int *)sum;
```

Jsou spočítány souřadnice levého horního a pravého dolního rohu a pokud oba dva leží v obrázku, spočítá součet jednotlivých jasových složek ve čtverci 5x5.

```
-(NSPoint)pixelCoordinatesAtIndex:(int)index;
```

Převede index pole, předaný jako parametr, na souřadnice daného bodu v obraze. Vrací nenormalizovaný bod.

`ZOProcessController a ZOProcess2Controler`

Slouží k nastavování properties `ZOProcessImage` resp. `ZOProcess2Image`. Má vlastní okno, ve kterém se posuvníky dají nastavovat minimální a maximální hodnoty jednotlivých jasových složek. Zaškrtnutí tlačítka Together vede ke společnému nastavování všch tří jasových posuvníků.

```
-(void)handleShowSettingsWindow:(NSNotification *)aNotify;
```

Tato metoda zachycuje notifikaci poslanou při stisku tlačítka Show Settings v hlavním okně. Zobrazí konfigurační okno `ZOProcessImage`.

```
-(void)setXY:(int)aY;
```

Tyto metody slouží k nastavení jednotlivých proměnných. Jsou používány při nastavování sliderů, které jsou navázány na tyto proměnné. Také zajišťují změny dalších proměnných, pokud je zaškrtnuto tlačítko Together.

```
-(NSMutableDictionary *)dictionaryWithConfigValues;
```

Tato metoda vytvoří `NSMutableDictionary`, kde pod klíči - jmény proměnných se skrývají jejich hodnoty v objektech `NSNumber`.

```
- (NSString *) pathForDataFile;
```

Tato metoda vrátí řetězec - cestu k souboru s uloženými daty. Pokud neexistuje složka, vytvoří ji.

```
- (IBAction) saveDataToDisk:(id) sender;
```

Tato metoda uloží pomocí NSKeyedArchiver hodnoty proměnných na disk. Využívá k tomu 2 výše popsané metody. Je volána z konfiguračního GUI tlačítkem Save.

```
- (IBAction) loadDataFromDisk:(id) sender;
```

Tato metoda načte z disku z cesty vrácené pathForDataFile soubor s proměnnými a jednotlivé proměnné dle něj nastaví.

ZOProcess2Image

```
-(id)initWithSize:(NSSize)aSize;
```

Inicializuje objekt, spočítá délku pole, nastaví velikost do vnitřní proměnné.

```
-(NSPoint)getThePointFromImage:(NSImage *)anImage;
```

Hledá bod v poskytnutém obrázku. Ten je převeden na pole bytů a poté jsou procházeny jednotlivé body. Pokud vyhoví nějaký bod minimálnímu a maximálnímu jasu, pak je zavolána metoda `sumCrossAtIndex` a její výstup opět prochází testováním jasu. Pokud projde, tak se porovná s nejlepším dosud dosaženým výsledkem a popřípadě se za něj vymění. Nejlepší výsledek je pak převeden na souřadnice pomocí `pixelCoordinatesAtIndex` a ty jsou normalizovány a vráceny.

```
-(NSPoint)pixelCoordinatesAtIndex:(int)index;
```

Viz ZOProcessImage.

```
-(struct colResults)sumCrossAtIndex:(int)index;
```

Tato metoda nejdříve vytvoří a vynuluje strukturu výsledků a poté do ní ukládá jasy jednotlivých vnitřních a vnějších bodů. U každého bodu spočítá jeho index a pokud je index v poli a nepřetéká řádek, připočte jednotlivé jasové složky a zvýší dělitele dané skupiny o 1. Na konci se jednotlivé složky po skupinách průměrují - vydělí se svým dělitelem.

KALIBRACE

ZOCalibrateController

```
-(void)setSize:(NSSize)aSize;
```

Pomocí této metody se nastavuje velikost snímaného obrazu. To pak slouží pro případné korekce.

```
-(void)calibrate;
```

Tato metoda je volána po kliknutí na tlačítko v GUI. V případě, že je více obrazovek (tzn. je připojen projektor), umístí kalibrační panel `calPanel` na celou obrazovku a spustí časovač `calTimer` volající po uplynutí času metodu `handleCalTimer`.

```
-(void) handleBlankTimer: (NSTimer *) aTimer;
```

Tato metoda zobrazí kalibrační bod a nastaví `calTimer` volající po uplynutí času metodu `handleCalTimer`.

```
-(void)handleCalTimer:(NSTimer *)aTimer
```

Tato metoda uloží do calPointsArray souřadnice aktuálního bodu. Protože je zobrazen kalibrační bod, uloží se jeho souřadnice. Poté se zruší zobrazení kalibračního bodu. Pokud již byly všechny zobrazeny, vynuluje se `calPointsArrayIndex`, pole calPointsArray se odešle pomocí notifikace a zruší se fullscreen zobrazení kalibračního okna. Pokud ještě nebyly všechny body zobrazeny, nastaví a spustí se `calTimer` volající po uplynutí času metodu `handleBlankTimer`.

```
-(NSString *)description;
```

Tato metoda vypíše souřadnice uložených kalibračních bodů.

```
-(NSArray *)calibrationArray;
```

Tato metoda vrátí pole kalibračních bodů.

```
-(ZOCalibrationData *)calibrationData;
```

```
@property NSPoint point;
```

Pomocí této metody se nastavuje bod, který se používá pro kalibraci.

`ZOCalibrateView`

```
-(void)setCalPoint:(int)index;
```

Tato metoda slouží k nastavení zobrazovaného kalibračního bodu.

```
-(void)drawRect:(NSRect)dirtyRect;
```

Tato metoda slouží k vykreslování kalibračních bodů. Podle čísla uloženého v `calPoint` se zobrazí červený čtverec v daném rohu. pokud je číslo mimo rozsah (0-3), žádný čtverec se nezobrazí.

TRANSFORMACE

Všechny tyto transformace splňují TransformProtocol, takže se dále budu zabývat konkrétními odlišnostmi jednotlivých transformací

Poměrová transformace

`ZOTransform`

```
-(double)getRightRootOfPolynomWithA:(double)a B:(double)b andC:
(double)c;
```

Tato metoda spočítá kořeny kvadratické rovnice a vrátí ten, který je v rozsahu monitoru.

```
-(void)setCalibrationArray:(NSArray *)modCalArray
```

Tato metoda spočítá a nastaví z pole `ZOPoint`ů transformační konstanty v poli `PTK`

```
-(NSPoint)transformPoint:(NSPoint)point
```

Pomocí pole PTK tato metoda sestaví rovnici a pomocí `getRightRootOfPolynomWithABandC` spočítá její kořeny, které pak použije k transformaci.

Transformace se dvěma body

`ZO2PointTransform`

```
-(void)setCalibrationArray:(NSArray *)modCalArray
```

Tato metoda z poskytnutých čtyř kalibračních bodů spočítá rovnice jejich spojnic `g`,`h`,`k`,`l`, jejich průsečíky, které uloží do pole `PTB` a různé vzdálenosti, které uloží do pole `PTD`.

```
-(NSPoint)transformPoint:(NSPoint)point
```

Tato metoda spočítá rovnice přímek spojujících daný bod s pomocnými z PTB a spočítá průsečíky těchto přímek se stranami kalibračního čtyřúhelníku. Poměry na těchto stranách jsou pak použity pro transformaci.

Čtyřúhelníková transformace

`ZOQuadTransform`

```
-(void)setCalibrationArray:(NSArray *)calArray
```

Tato metoda z poskytnutých bodů spočítá pomocná čísla `d0` - `d7`

```
-(NSPoint)transformPoint:(NSPoint)point
```

Tato metoda pomocí rovnic použitých v PIL spočítá transformované souřadnice zadaného bodu.

ZOBRAZENÍ

Standardní protokolované metody jsou dostatečně popsány v popsiu protokolů, takže se budu zabývat jen dalšími metodami nad rámec protokolů.

ZOProjectorController

Zde není nic již nepopsaného.

ZOProjectorView

```
-(NSBezierPath *)crossAtPoint:(NSPoint)aPoint;
```

Tato metoda vykreslí pomocí 2 úseček křížek o dané velikosti `r` okolo zadaného bodu a vrátí `NSBezierPath` s tímto křížkem.

```
-(void)setPoint1:(NSPoint)aPoint;
```

Nastaví bod `aPoint` do `point1`.

```
-(void)setPoint2:(NSPoint)aPoint;
```

Nastaví bod `aPoint` do `point2`.

```
- (void)drawRect:(NSRect)dirtyRect;
```

Tato metoda vykreslí křížky o nastavené barvě na souřadnicích zadaných v `point1` a `point2`.

ZODrawingController

Zde jsou kromě protokolovaných metod ještě další, které souvisí s GUI sloužícím ke konfiguraci kreslení.

```
-(void)setWidth:(float)aWidth;
```

Protože posuvník i textové pole jsou navázány na proměnnou `width`, volá se tato metoda při změně šířky kreslené čáry. Kromě nastavení proměnné `width` je nová hodnota poslána `ZOProjDrawingView` pomocí metody `setLineWidth`.

```
-(IBAction)resetDrawing:(id)sender;
```

Tato metoda pouze pošle zprávu `resetDrawing` kreslícímu objektu `drawView`.

```
-(IBAction)saveBezier:(id)sender
```

Tato metoda uloží aktuální kresbu na zadané místo. Kresbu uloží jako její popis pomocí metody `description`. Tento soubor lze pak pomocí Perlového skriptu `bezpath2svg.pl` převést na jednoduchou SVG grafiku, se kterou pak lze pracovat mnoha programy, například Inkscape.

ZOProjDrawingView

```
-(NSBezierPath *)crossAtPoint:(NSPoint)aPoint;
```

Viz výše.

```
-(void)setPoint1:(NSPoint)aPoint;
```

Tato metoda jednak uloží nastavený bod a jednak kreslí křivku `drawedPath`. Pokud dostane nulový bod, tak nastaví proměnnou `drawing` na `NO` a nic nekreslí. Až přijde nenulový bod, přesune se do něj bez kreslení čáry, nastaví `drawing` na `YES` a kreslí dále.

```
-(void)setPoint2:(NSPoint)aPoint;
```

Viz výše.

```
-(void)resetDrawing;
```

Uvolní současnou drawedPath a vytvoří novou, jejíž počátek nastaví na 0,0 a drawing na NO.

```
@property (readonly)  NSBezierPath * drawedPath;
```

Vrátí aktuální `drawedPath`.

```
- (void)drawRect:(NSRect)dirtyRect;
```

Tato metoda vykreslí křížky na nastavených souřadnicích a také vykreslí křivku `drawedPath`.

Aktuální stav

AKTUÁLNĚ FUNGUJÍCÍ

Aktuálně funguje celý proces - načtení, zpracování, transformace i zobrazení popřípadě kreslení. Jednotlivé části procesu jsou standardizovány protokoly, nastavování má každá část zvlášť ve svém vlastním okně. Volba konkrétních tříd pro jednotlivé činnosti se provádí pomocí GUI. Lze ukládat nastavení minimálních a maximálních hodnot pro zpracování obrazu . Také lze ukládat nakreslené křivky a tyto soubory lze pak pomocí Perlového skriptu `bezpath2svg.pl` převádět na SVG.

ROZPRACOVANÉ

Rozpracovaná je úprava dokumentace, hlavně části Implementace, která popisuje činnost jednotlivých tříd a metod, aby odpovídala současnému stavu. Probíhají také experimenty se čtyřúhelníkovou transformací, která ale zatím nepracuje tak, jak má.

PROBLÉMY

Prvním, zásadním a nejhloupějším problémem byl problém s poměrovou transformací. Jeho podstata byla v odlišném číslování rohů obdélníka při kalibraci a při transformaci. Tato chyba nás zdržela o jedno odpoledne, kdy jsme mohli pokračovat v programování. Byla odhalena až asi v polovině prezentace konfery na soustředění M&M.

Problémy byly při přepisování do Objective C, ale byly úspěšně překonány. Největším problémem bylo vždy rozběhat danou transformaci, protože při přepisování do počítače vznikla vždy alespoň jedna chyba, která se dost těžko hledala. Tyto problémy se táhly dlouhou dobu, je to vidět i na tom, že první námi naprogramovaná transformace se podařila zprovoznit až za dlouhou dobu a například zprovoznit dvoubodovou transformaci trvalo od 27. ledna, kdy byla uveřejněna na SVN až do 18. března, kdy vyšla stabilní revize.

Dalším problémem bylo kreslení, při němž jsem si nevšiml, že `[NSBezierPath bezierPath]` je autorelease a divil jsem se, že do ní pak při dalším volání metody nešlo zapisovat.

Při přechodu na využívání vazeb se vyskytl problém se čtením hodnot. Sice fungovalo nastavování, ale v GUI se to nezobrazovalo. Příčina nebyla zjištěna, nyní to již funguje tak, jak má.

Budoucnost

DOKUMENTACE, UKLÁDÁNÍ

Nejdůležitějším cílem je zdokumentovat třídy a metody. Stále probíhají experimenty s rozlišením VGA, které umožní přesnější ukazování, ale musí být dobře nastavené zahazování pixelů, protože jinak počítač nestíhá obraz zpracovávat. Dalším cílem je přidání možností ukládání kalibračních dat a nakresleného obrázku ze `ZOProjDrawingView` v nějakém rozumném formátu.

MOŽNÉ APLIKACE

Kreslení

Místo toho, aby se zobrazoval jen křížek pod nasvíceným bodem bude laser za sebou zanechávat barevnou stopu. Tímto způsobem bude možné kreslit jednoduché obrázky a může být i možnost změny barvy. Opět je ale potřeba, aby barva čáry kreslené laserem nebyla stejná jako nasvícený bod, aby nebyla chybně zachycena jeho poloha.

Tato aplikace je již ve své základní podobě hotova.

Střílení balonků

Jednoduchá hra pro 1, v budoucnosti 2 hráče s různě barevnými laserovými ukazovátky, kteří budou pomocí svícení na balonky zobrazené na projektoru na tyto balonky střílet a získávat za ně body.

Ovládání myši

Pomocí laserového ukazovátka bude možné ovládat kursor myši. Zde je problém s barvou oken, která musí být diametrálně odlišná od barvy ukazovátka, aby nedocházelo k chybným detekcím. Rovněž by bylo potřeba detekovat kliknutí, například pomocí bliknutí laserem.

Odkazy a zdroje

Seminář M&M: http://mam.mff.cuni.cz/

Referenční příručka Python Image Library

Mac OS X Reference Library

Zdrojové kódy programu a jeho stránky: http://code.google.com/p/zpracovaniobrazu/

Blog se zprávami z aktuálního postupu: http://zpracovaniobrazu.blogspot.com/

Poděkování

Tomáši Gavenčakovi za vedení konfery na M&M

Karlu Královi a Lukáši Langerovi za vymyšlení poměrové transformace

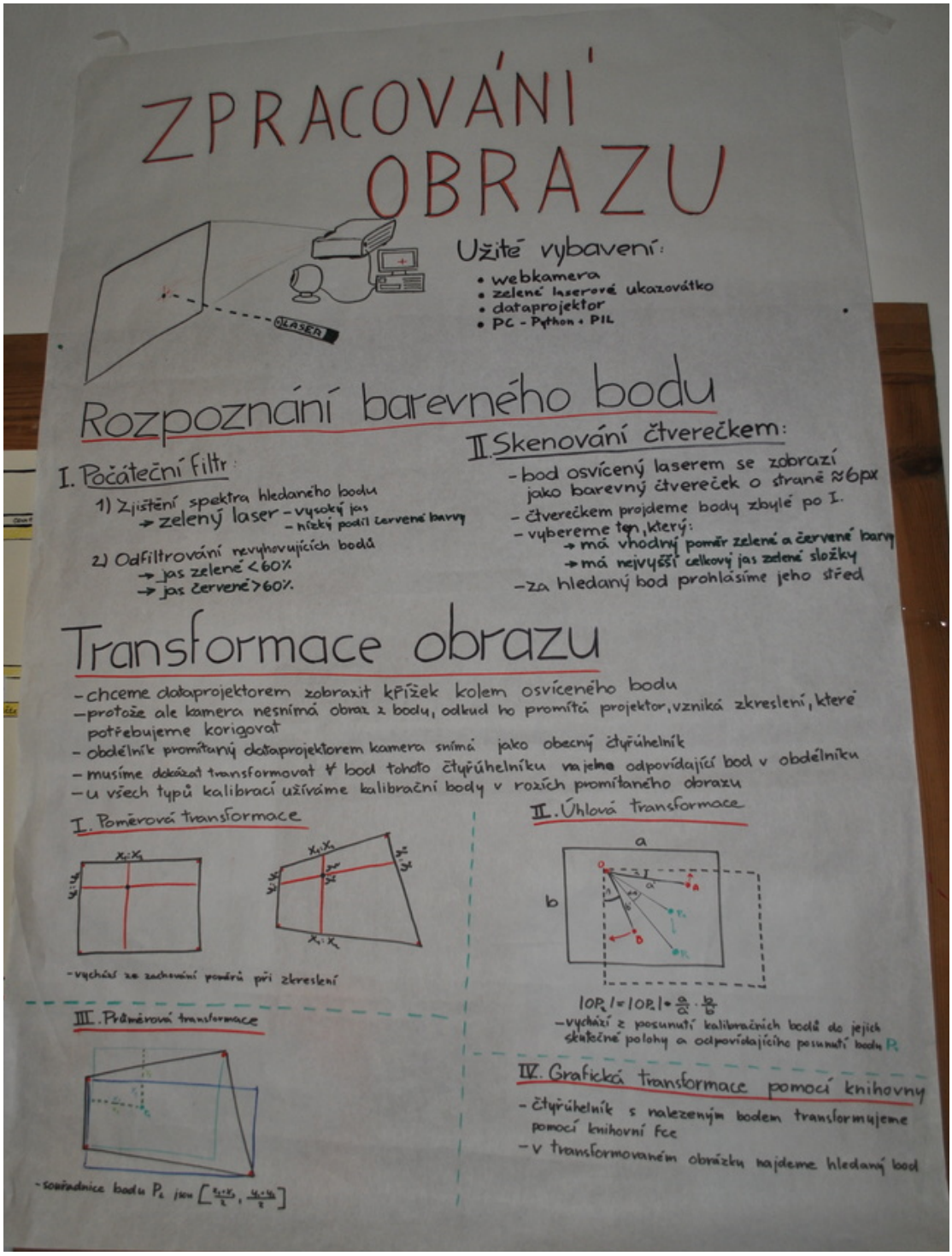
Lukáši Langerovi za vymyšlení dvoubodové transformace

Petrovi za spolupráci na konfeře

Mgr. Marku Blahovi za technické zázemí a podporu při pokračování projektu

A všem dalším, kteří mě svými radami nasměrovali na správnou cestu

Poster z M&M


ZPRACOVÁNÍ OBRAZU
LASER
Užité vybavení:
• webkamera
• zelené laserové ukazovátko
• dataprojektor
• PC - Python + PIL
Rozpoznání barevného bodu
I. Počáteční filtr:
1) Zjištění spektra hledaného bodu
→ zelený laser - vysoký jas
- nízký podíl červené barvy
2) Odfiltrování nevyhovujících bodů
→ jas zelené < 60%
→ jas červené > 60%.
II. Skenování čtverečkem:
- bod osvícený laserem se zobrazí jako barevný čtvereček o straně ≈6px
- čtverečkem projdeme body zbylé po I.
- vybereme ten, který:
→ má vhodný poměr zelené a červené barvy
→ má nejvyšší celkový jas zelené složky
- za hledaný bod prohlásíme jeho střed
Transformace obrazu
- chceme dataprojektorem zobrazit křížek kolem osvíceného bodu
- protože ale kamera nesnímá obraz z bodu, odkud ho promítá projektor, vzniká zkreslení, které potřebujeme korigovat
- obdélník promítaný dataprojektorem kamera snímá jako obecný čtyřúhelník
- musíme dokázat transformovat ∀ bod tohoto čtyřúhelníku na jeho odpovídající bod v obdélníku
- u všech typů kalibrací užíváme kalibrační body v rozích promítaného obrazu
I. Poměrová transformace
- vychází ze zachování poměrů při zkreslení
II. Úhlová transformace
a
b
- vychází z posunutí kalibračních bodů do jejich skutečné polohy a odpovídajícího posunutí bodu P
III. Průměrová transformace
- souřadnice bodu P jsou
IV. Grafická transformace pomocí knihovny
- čtyřúhelník s nalezeným bodem transformujeme pomocí knihovní fce
- v transformovaném obrázku najdeme hledaný bod