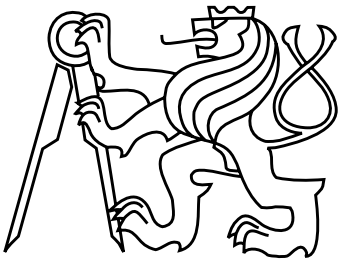
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

Vývojová sada pro rozšířenou realitu v mobilním prostředí

David Menger

Vedoucí práce: Ing. Roman Berka, Ph.D.

Studijní program: Softwarové technologie a management, Bakalářský
Obor: Softwarové inženýrství
20. května 2013

Poděkování

Děkuji vedoucímu práce za trpělivost a pomoc s projektem a dále děkuji oběma testerům za projevenou ochotu se testu zůčastnit.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze 2013 .....

Abstract

This bachelors thesis is focused on design and implementation of a toolkit for development of a object recognition applications in video. Development of mobile applications is growing in general and current frameworks aren't keeping pace. The aim is to create modern framework, which will be instrumental in faster understanding of a computer vision and in a same time to simplify a routine programming. These things can contribute to the expansion and development of Augmented Reality applications in real world.

Abstrakt

Bakalářská práce je zaměřena na návrh a implementaci sady nástrojů pro vývoj mobilních aplikací rozpoznávajících objekty ve videu. Vývoj mobilních aplikací se obecně rozmáhá a stávající nástroje nestačí držet tempo. Cílem je vytvořit moderní framework, který přispěje k rychlejšímu pochopení problematiky vývojáři a zároveň zjednodušit rutinní činnosti profesionálům. To může přispět k rozšíření a rozvoji Augmented Reality aplikací v reálném světě.

Obsah

Seznam obrázků

Seznam tabulek

1. Úvod

V dnešní době, kdy končí takzvaná éra webu 2.0 a nastupuje éra webu 3.0 [1], kterou charakterizují klesající tržby ze stolních počítačů, se kterými se potýká stále více tradičních hráčů na tomto trhu a zároveň prudce stoupají prodeje k internetu připojených mobilních zařízení, které se často výkonem vyrovnají běžným stolním počítačům a co se týče vybavenosti multimediálním hardware je naprosto převyšují. Lidé si zvykají používat svá mobilní zařízení, ať už jsou to tablety, nebo mobilní telefony s operačním systémem, pro přístup k tradičním internetovým informačním a komunikačním službám s využitím maxima vlastností, které tato zařízení nabízejí. Jde především o relativně solidní výkon a odezvu, nové UX možnosti které nabízí dotykové prostředí, senzory jako jsou akcelerometry, čidla vnějšího osvětlení, zařízení pro geolokaci (ať už GPS, nebo Glonass) a v neposlední řadě právě kvalitní fotoaparáty schopné snímat video ve vysokém rozlišení.

Všechny možnosti, které tato zařízení nabízejí, respektive aplikace které je využívají, jsou výborným marketingovým nástrojem, jak zaujmout a získat zákazníka, respektive uživatele. To si řada firem uvědomuje a díky tomu vzniká poptávka po takových multimediálních aplikacích. Důkazem jsou například AR brýle Google Glass [2], které jsou pro Augmented Reality (AR) aplikace jako stvořené. Primárně operují s kamerou, hlasovým ovládáním a jednoduchým zobrazovacím zařízením v rovině oka a nabízejí prostor pro celou řadu aplikací, které budou primárně zpracovávat obraz z této kamery a vytvářet v jejich uživateli prostřednictvím získané informace z obrazu nějaký konkrétní dojem, či zážitek.

Současnému rozvoji AR aplikací ale bohužel neodpovídá tempo rozvoje nástrojů, které takové aplikace dovolí vývojářům vytvářet. Obecně lze říci, že buď nejsou moderní a využívají letité přístupy k programování, což může být pro řadu uživatelů odrazujícím faktorem, nebo jsou pro širší vývojářskou veřejnost nedostupné. Jednoduchý, moderní, ale zároveň robustní nástroj zde chybí.

Obrázek 1.1: Google Glass

Rozvoj takových technologií je pak předzvěstí vzniku poptávky po takových aplikacích a v důsledku i vzniku tlaku na vývojáře, aby se s nimi seznámili a ve svých aplikacích je byli schopni kreativně využít. V současnosti je ale křivka učení Computer Vision (CV) technologií celkem pozvolná a to může být pro malé hráče na trhu a rozvoj technologie limitující faktor. Obecně mezi programovacími jazyky začínají hrát prim jednoduchá a intuitivní řešení, která se sice příčí zavedeným praktikám, ale zpřehledňují kód a nabízejí intuitivní abstrakci nad složitějšími technikami, které dříve vyžadovaly mnohem komplexnější znalosti problematiky. A to je právě trend, který je třeba sledovat.

2. Vymezení cílů

2.1. Polemika

Cílem projektu by měla být příprava základní sady nástrojů, která současnou generaci vývojářů provede úskalími Computer Vision, zjednoduší jim vývoj CV aplikací a výrazně zvedne křivku učení práce s CV a tím zrychlí vývoj AR a podobných aplikací.

Cílem by ale nemělo být vytvářet úzký a jednoúčelový nástroj pro konkrétní aplikaci, ale spíše ukázat cestu, jakou by se měl mohl vývoj těchto aplikací ubírat a otevřít tak tuto, v současnosti rostoucí oblast multimediálních aplikací širší odborné veřejnosti. Framework totiž musí být použitelný jak začátečníkem, který jen potřebuje využít jeho vlastnosti, aby si ušetřil čas, tak i odborníkem, který musí mít možnost vidět do jeho nitra a ovlivňovat jeho dění.

Důležitým cílem je, aby samo programování mělo pro uživatele frameworku edukativní hodnotu. Aby při jeho používání zároveň získával o frameworku vědomosti bez nutnosti použít dokumentaci.

2.2. Shrnutí a definice cílů práce

Cílem práce je vytvořit:

1. dostupný nástroj pro co nejširší spektrum vývojářů, který dokáže pomoci vývojářům se zkušenostmi s Computer vision
2. nástroj, který zjednoduší začátečníkům vstup do světa CV pomocí moderních trendů v programování. Čím jednodušší totiž pro vývojáře bude učení a práce s technologií, tím rychlejší bude její rozvoj.

3. Rešerše aktuálního stavu CV

3.1.Současná situace v Computer Vision na mobilních zařízeních

3.1.1. Používaný hardware a platformy

Trh mobilních operačních systémů je v současné době velice roztříštěný a zatím se nenašel leader, který by jednoznačně určil platformu, která bude dominantní. Podle současných výsledků [3] má nejvyšší marketshare OS Android v těsném závěsu před iOS firmy Apple, ale kvůli rozmanitosti platforem, na kterých Android běží, je stále iOS v pohledu podnikatelského sektoru jeho rovnocenným partnerem. Co se týče dalších platforem, jako jsou Windows Mobile a RIM (BlackBerry), nebo další, jejich podíly jsou více méně zanedbatelné, i když dohromady s ostatními tvoří celkem širokou část spektra, což potvrzuje současnou platformní roztříštěnost.

Obrázek 2.1: Vlevo Samsung Galaxy S3 (Android), vpravo Apple iPhone 5 (iOs)

Co se týče hardware, rozdíly mezi tablety a mobilními telefony jsou z pohledu potřeb AR i vývoje aplikací natolik zanedbatelné, že nemá smysl se jimi nějak zabývat. Tato zařízení až na naprosté výjimky mají relativně kvalitní zařízení pro záznam obrazu. Podle katalogu serveru Mobilmania [4], který v době vzniku této práce obsahuje 658 mobilních telefonů s operačním systémem, z nichž celých 620 jich dovoluje natáčet videozáznam, což je drtivá většina. Dále více než polovina z nich (přesně 317) dokáže pořizovat videozáznam v rozlišení o minimálním počtu řádků 480-ti. To je pro CV více než dostačující. Navíc prakticky neexistuje takové retailové mobilní multimediální zařízení, které by nemělo barevný displej, nebo by bylo stavěno na platformě, která by nenabízela SDK pro vývoj aplikací.

3.1.2. Vývojářské komunity mobilních platforem

Mobilní platformy jsou relativně mladé a to samé platí pro jejich vývojáře, ale samozřejmě ne pro všechny. Mezi zkušenější programátory iOS zařízení totiž pronikli i vývojáři, kteří vytvářeli aplikace pro MacOS X, který je už na trhu od roku 2001 a podobné je to se zkušenými Javovými programátory. Pro obě platformy ale platí, že v jejich vývojových sadách jsou maximálně uplatňovány nové principy tak, aby bylo možné co nejlépe využít potenciál zařízení.

Vývojová prostředí obou platforem jsou tak relativně přístupná širší komunitě a řada mladých programátorů právě na těchto platformách začíná vyvíjet, zejména proto, že je mobilním platformám předpovídána dlouhá budoucnost. Navíc platformy s sebou nenesou historická břemena jejich stolních předchůdců a jsou tak mnohem jednodušeji použitelné.

3.1.3. Současné úspěšné CV a AR aplikace a vývojářské zázemí

Augmented Reality ve svých aplikacích využívají jak silní internetoví hráči, jako je Google nebo Yelp ke zvýšení User Experience (UX) ze svých služeb, tak i známé značky k jednorázovým marketingovým akcím. Google využilo Augmented Reality ve spojení se svými mapovými podklady například v aplikaci Google Skymap [5], která umožňuje identifikovat souhvězdí na nočním nebi.

Například Yelp využil AR k zobrazování hodnocených podniků prostřednictvím pohledu skrze fotoaparát telefonu už v roce 2009 [6]. Dále časopis Bussines insider shrnuje [7] nejzajímavější marketingové kampaně velkých značek, jako jsou Nivea, National Geographic, BMW, nebo Starbucks, které využívají Augmented Reality k zaujetí uživatele reklamou.

Obrázek 2.2: Aplikace Google Skymap

Ze současné situace lze víceméně vyvodit, že využití AR spočívá spíše v marketingových aktivitách velkých hráčů. To znamená, že jejich vývoj je finančně náročnější a nemůže si ho tak dovolit každý a to nejen z pohledu samotné implementace, ale i vypracování celé myšlenky a konceptu propagace hotového řešení. Mezi aplikacemi ale nejsou jen jednorázové marketingové nástroje, ale i náznaky užitečných služeb, což dokládá to, že Google vyvíjí brýle Google Glass. Jejich přijetí veřejností by znamenalo průlom v této oblasti.

V současnosti je pro mobilní platformy obecně vývojářská základna velice široká. Vyvíjet a publikovat aplikace je snadné nejen díky propagačním nástrojům AppStore od Apple, nebo Google Play určeném pro Android. Tyto systémy propagace aplikací pak motivují vývojáře k práci s platformou, protože jednotlivcům slibují nejen možnost seberealizace a velkého výdělku, ale i perspektivu v budoucím uplatnění. Google, Apple i Microsoft využívají známé programovací jazyky, nad kterými staví vývojová prostředí, která se všichni snaží maximálně zjednodušovat tak, aby byl vývoj aplikace co nejméně náročný a otevřel se tak širší odborné veřejnosti. API pro využití služeb, jako jsou GPS, kompas, akcelerometr, nebo gyroskop jsou jednodušeji přístupná, že je mnohem jednodušší zrealizovat zajímavý nápad a to je klíčové v rozvoji těchto platforem.

Vývoj se obecně stává jednodušším, protože se začínají uplatňovat nové principy. Kód začíná být maximálně objektový, objevují se koncepce, jako Fluent Interface [8] Martina Fowlera, které zjednodušují kód, je vyzdvihována filosofie KISS (keep it simple stupid) nebo DRY (dont repeat yourself) a obecně je programátor maximálně abstrahován od hardwaru, aby se mohl maximálně koncentrovat na svou práci a nechal podružnosti za sebe řešit jiné systémy. Důkazem toho, a zároveň extrémem, kam toto může dojít, jsou frameworky, které umožňují vyvíjet nativní aplikace v javascriptu s přispěním technologie HTML5. Tyto přístupy jsou zejména patrné mezi webovými nástroji, jako je například PHP framework Nette [9], který dokázal, že může existovat nástroj sloužící současně pro vývoj složitých systémů i jednoduchých mikrostránek zároveň. Nette totiž může jednoduše začít používat webový vývojář znalý PHP a ono za něj vyřeší celou řadu opakujících se úkonů a bezpečnostních návyků a zároveň vývojáři umožní postavit na Nette i rozsáhlý komplexní web bez toho, aby ho nějak limitovalo.

3.1.4. Existující frameworky a řešení pro Computer Vision v mobilních platformách

3.1.4.1.OpenCV

OpenCV [10] (opensource computer vision) patří mezi frameworky s dlouhou historií. Je zde od roku 1999 a stále kolem něj žije celkem početná komunita vývojářů, která jej neustále vyvíjí. Mezi výhody OpenCV patří, že je napsáno v C a díky tomu je možné ho používat na celé řadě platforem. Existuje totiž v mutacích, respektive buildech pro Android, Linux, Windows i iOS.

Další důležitá výhoda OpenCV spočívá v jeho rychlosti, protože obsahuje primárně funkce na nejnižší úrovni práce s obrazovými daty, což vytváří zároveň i nevýhodu v podobě nutnosti znalosti principů a rozhraní na nejnižší úrovni abstrakce od problematiky. Současně je ale celý framework řádně zdokumentován a na internetu existuje spousta jednoduchých příkladů použití OpenCV a tak se i přes to, že je framework spíše sadou funkcí, než vývojovým prostředím, lze jeho používání naučit.

V souvislosti s OpenCV bych velice rád zmínil i nástroj, který se jmenuje Aruco [11]. Aruco je lightweight nástavba nad OpenCV, která má objektový základ, ale objekty používá jen k udržení pořádku v projektu a eventuelní škálování aplikace, co se týče možnosti vytvoření vlastního modulu pro dekódování a mapování značek.

Aruco je v tomto směru inspirativním řešením, které naznačuje, i když dost nepřehledným způsobem, jak jednotlivé funkce v OpenCV spojovat a abstrahovat nad nimi.

3.1.4.2.ARToolKit

Historie ARToolKitu [12] sahá také ke konci devadesátých let a je rovnocennou alternativou k OpenCV. Stejně jako OpenCV je napsáno v C a C++. Rozdílem mezi oběma knihovnami je zejména jejich určení a úroveň abstrakce. OpenCV obsahuje nejen nástroje pro Augmented Reality, ale je mnohem komplexnější v možnostech zpracování obrazu. Naopak ARToolKit se přímo specializuje na detekci fragmentů v obrazu a mapování virtuálního objektu na obraz. ARToolkit je například často využíván ve spojení s Unity3D [13], multiplatformním prostředím pro vývoj 3D aplikací, zejména her.

3.1.4.3.Další řešení pro Computer Vision

Je důležité, zmínit i další, pro mobilní platformy krkolomná, či spíše nereálná, ale známá řešení pro Computer Vision. Jde například o SimpleCV, jehož výhodou je v názvu uvedená jednoduchost, ale omezení pouze pro platformy schopné spustit Python je velikým handicapem. Důvod, proč si zaslouží zmínku je, že reprezentuje modernější styl programování a jedná se o jedno z mladších řešení. Je třeba nezapomínat také na MathLab, který se pro Computer Vision používá velmi často, ale jeho použití na mobilních platformách je také vyloučeno.

3.1.5. Zhodnocení současné situace v kontextu s cíli

Předně lze konstatovat, že skok, který během posledních let zažil hardware, na kterém lze mobilně provozovat aplikace pro Computer Vision, potažmo Augmented reality je natolik velký, že na něj ještě volně dostupné nástroje pro programátory a vývojáře ještě úplně nestačily zareagovat. Respektive frameworky, které ještě před tím existovaly, se stačily trendu přizpůsobit co se týče zpřístupnění jejich knihoven, nicméně stále nevznikl nástroj, který by následoval současné webové, ale i mobilní trendy vývoje. Z toho důvodu je místo pro takový framework zcela odůvodněné. Je totiž třeba mít nástroj, který dovolí i začátečníkovi využít pokročilé metody starších frameworků s tradicí novým jednoduchým způsobem.

4. Analýza a návrh implementace

4.1. Možné koncepty a varianty

Koncepce návrhu frameworku v sobě nese několik podproblémů, jejichž řešení jednoznačně určí, jakým směrem se jeho filosofie bude ubírat. Jde zejména o otázku využití některého, již hotového řešení, dále vyvstává otázka koncepce celé architektury frameworku, volba platformy pro framework a filozofie celého systému.

Předně je nutné, si vytyčit dílčí cíle, aby bylo jednodušší na tyto otázky odpovědět. Prvořadě, framework musí být jednoduše použitelný pro začátečníky, aby ho neměli potíž pracovat se složitějšími funkcemi a aby za ně framework řešil některé základní potíže, ale zároveň umožnil pokročilým uživatelům ovlivňovat celý veškeré dění pod jeho pokličkou. Musí být také škálovatelný. To se týká jak výkonu, tak i jeho vývoje. Zkrátka i když přeroste ve skutečně komplexní nástroj, musí si zachovat stále svou jednoduchost vůči uživateli a výkonnost lehkého nástroje. Samozřejmě v ideálním případě by se mělo jednat o multiplatformní řešení.

4.2. Otázka využití stávající knihovny jako základu

Předně je nutné zdůraznit, že problematika Computer vision je komplexní záležitost a že jakékoliv "vynalézání kola" by se mohlo na použitelnosti nástroje značně podepsat. To nahrává použití již hotového řešení. Pokud by toto řešení mělo být využito, muselo by splňovat výše uvedené cíle, zejména pak použitelnost na více platformách, škálovatelnost a existenci podpory nástroje i v budoucnu. V tomto jednoznačně vede OpenCV, protože splňuje požadavky na výkonnost, multiplatformnost a současná situace, kdy se na repositáři vývojové verze každým dnem objeví více než dva commity od komunity která ho spravuje, budí jistotu, že bude podporován po delší dobu. Soupeřem OpenCV je pak ARToolkit, proti kterému hraje jen fakt, že je zaměřen úžeji, než OpenCV a nemusel by nabízet dostatečnou škálovatelnost. Ta je ovšem uvedena mezi klíčovými požadavky.

Nabízí se ještě eventualita využití a rozšíření frameworku Aruco. Ten je ovšem už relativně zastaralý a nenabízí inspiraci co se architektury týče. Jeho vývoj se sice zatím nezastavil, ale struktura frameworku je natolik zkostnatělá, že nedovoluje další racionální rozšiřování.

Použití OpenCV se v tomto světle jeví jako nejlepší řešení. Umožní totiž vývojářům, kteří ho znají, aby viděli do vnitřního fungování frameworku a mohli si ho upravit dle svých potřeb.

4.3. Volba platformy

Jak vyplývá z rešerše, současná situace mezi podíly na trhu mobilních platforem nemá vedoucího hráče. Navíc každá platforma využívá jiný programovací jazyk. Apple používá pro svůj iOS nástavbu nad C, proprietární Objective C, Google pro svůj Android zvolila metodu sandboxovaní aplikací v Java Virtual Machine (z čehož vyplývá, že použitá platforma je Java) a Microsoft používá pro vývoj aplikací prostředí s C#. OpenCV je ale napsáno v C a C++ a existuje ve verzi pro Android, z čehož vyplývá, že by mohl existovat nástroj, který dovede kód napsaný v C zkompilovat do Javy. Tím je Android NDK [14]. Pokud by pak bylo možné framework zkompilovat i pro Android za předpokladu, že jeho hlavní smyčka se bude nacházet právě v C kódu. A jak uvádí dokumentace: “Typical good candidates for the NDK are self-contained, CPU-intensive operations that don't allocate much memory, such as signal processing, physics simulation, and so on.” [14], je stává se pak toto řešení ideálním. Computer vision smyčku je pak možné vyvíjet pouze v C a následně ji zkompilovat do Javy. Nové Windows phone také už knihovny v C++ podporují.

Protože je ale mezi kritérii požadavek na jednoduchost řešení, podstata jazyka C, který je neobjektový, respektive dovoluje objektové myšlení velice omezeně, nabízí se využití C++, které je již objektové na vyšší úrovni a dovoluje použitou knihovnu OpenCV mnohem elegantněji zapouzdřit.

4.4. Architektura a filozofie frameworku

Stávající frameworky mají procedurální základ a jejich koncepce je postavena na sadě funkcí. Ty novější již uživateli umožní více využít objektový styl práce, ale celou věc komplikuje fakt, že nejsou dostupné napříč platformami. Frameworky koncipované jako sady funkcí bohužel nemají takový edukativní vliv na jejich uživatele, jako nástroje využívající objektových principů.

Architektura musí splňovat výše uvedené principy, zejména co se týče intuitivity, modernosti řešení a rozšiřitelnosti. Začátečník musí mít možnost jednoduše a rychle začít používat všechny dostupné funkce frameworku i za předpokladu, že o nich ví jen minimální základ a zároveň musí být systém rozšiřitelný.

Protože v dnešní době má většina IDE v zabudovanou funkci s našeptáváním, která napovídá názvy funkcí, může tato možnost v některých případech zcela vývojáře odprostit od nutnosti pohledu do dokumentace. Proto by měla být maximálně využita tato vlastnost moderních IDE a to tak, že spíše místo metod s celou řadou neurčitých parametrů bude lepší použít více metod s dostatečně výmluvným názvem a jedním až dvěma parametry s využitím objektů a konstant.

Lze také říci, že každé zpracování obrazu se točí kolem jedné smyčky obsahující řadu funkcí, respektive proceduru zpracování sejmutého rámce obrazu. Skrze jednotlivé funkce pak protéká jen konkrétní bitmapa a občas parametry spojené s rozpoznaným obrazem. Obecně lze říci, že vzniká jakási pipeline, kterou obraz protéká.

Pohodlí vývoje samozřejmě zvyšuje, když vývojář nemusí při ladění aplikace odbíhat kvůli změně některého parametru do jiné části konkrétního souboru, či celého projektu a má vše důležité, co k ladění parametrů potřebuje na jednom místě. A mělo by být maximálně jednoduché do řetězce přidat, či vyřadit nějakou funkci, popřípadě upravit parametr. Zde se nabízí výše uvedený pattern Fluent Interface [8], který uživatele nenutí pokaždé ukončovat řádek kódu, aby mohl nastavit objektu nějaký parametr.

To obecně platí i při vývoji komponent. Každá komponenta, respektive objekt by měl mít jasně danou zodpovědnost, aby při vývoji určité části aplikace nebylo třeba přecházet mezi řadou souborů. K tomuto bude ideální využít některé zákonitosti patternu GRASP [15] (General Responsibility Assignment Software Patterns), protože ten dobře definuje systém, jakým se má s objektem rámce nakládat. A dále se inspirovat některými principy SOLID [16] (Single responsibility, Open-closed, Liskov substitution, Interface segregation and Dependency inversion), jako jsou Single Responsibility pro zdůraznění odpovědnosti objektů, Interface segregation principle a Liskov substitution principle kvůli rozšiřitelnosti objektů a jednoznačnosti implementace. Sám interface, jímž v C++ může být základní třída od které dědí její podtřídy, by měl určovat význam objektu a způsoby jeho použití.

4.5. Řešení objektového modelu

Aby bylo možné celou architekturu aplikace zpřehlednit a vytvořit flexibilní a rozšiřitelný základ, je třeba definovat základní strukturu objektového modelu postavenou na co nejmenším počtu tříd. Aby nebyl uživatel zmatený, když bude potřebovat rozšířit stávající řešení o další implementace, musí samotná struktura frameworku už na první pohled dávat ke konkrétnímu rozšíření návod. Proto je pevný, robustní, ale zároveň minimalistický základ velice důležitý.

Co se týče možností, jakým způsobem objektovou strukturu navrhnout, je třeba si nejdříve shrnout, jakým způsobem se obrazová data obecně zpracovávají. Zpracování obrazových dat je řešeno většinou řetězením základních funkcí pro modifikaci, či feature recognition za sebe. Dá se také říci, že programátor na začátku smyčky dostane nově vzniklý rámec ze záznamového zařízení, tedy kamery telefonu, provede na něm úpravy, eventuálně rozpoznává vzniklé artefakty a výsledek, ať už v podobě upraveného obrazu, či pozic rozpoznaných objektů zobrazuje na výstupu. Celé toto řešení tak připomíná Pipeline. Jednotlivé funkce vždy dostanou obrazová data, eventuálně doplněná o pole souřadnic rozpoznaných artefaktů. Tato data jsou pak často předávána od funkce k funkci.

Z této obecné definice lze vyvodit několik závěrů, respektive potencionálních řešení. Není pochyb, že vždy bude existovat zdroj obrazu, obrazový výstup či 3D canvas a jednotlivé kroky pro modifikaci obrazu. Také se dá s jistotou říci, že bude třeba obrazová data s výstupy ve formě souřadnic rozpoznaných artefaktů nějakým způsobem obalit, aby nebylo nutné mezi jednotlivými kroky předávat celé řady parametrů.

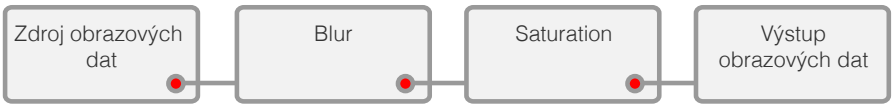
To samo o sobě už dává tušit, že v objektovém modelu bude existovat interface reprezentující zdroj obrazu, abstraktní třída reprezentující konkrétní funkci či metodu zpracování obrazu a interface reprezentující obrazový výstup.

4.6. Způsob řetězení jednotlivých funkcí pro zpracování obrazu

Logicky se nabízí, že výstupem každého objektu reprezentujícího zdroj obrazu budou obrazová data, pravděpodobně ve formě kontejneru. Tato data by pak byla referencí předávána mezi objekty sloužící k zpracování obrazu a následně by byl tento kontejner předán objektu reprezentujícímu obrazový výstup. To ale znamená, že programátor musí při každém zapojení objektu pro zpracování obrazu do řetězce kontejner s obrazovými daty předat a dále by musela být obecně definována procedura smyčky, na jejíž začátku stojí právě zdroj obrazu a na konci obrazový výstup.

Mnohem elegantnější by bylo, kdyby si objekty obrazová data předávaly samy. Toto je proveditelné za předpokladu, že objekty budou předem znát, kde a kdy si mají obrazová data vyzvednout, popřípadě kam je po zpracování mají předat. Dále je také potřeba nějakým způsobem odbourat nutnost vytvářet proceduru smyčky. V tomto případě se nabízí jako ideální řešení Event driven model a tou událostí, která celý řetězec uvádí do chodu je pak vnik obrazového rámce na zdroji obrazových dat. Toto, samo o sobě pořád ale neřeší potřebu zbavit programátora nutnosti vytvářet proceduru, která se po vniku události spustí.

Pokud by už měl zdroj obrazu předem nadefinováno, který objekt pro zpracování obrazu si kontejner s obrazovým rámcem přebere, mohl by mu ho při vzniku události rovnou předat. Pak by stačilo, aby programátor pouze nadefinoval, kterému objektu má obrazový zdroj data předat. V konečném důsledku by se pak objekt, který data zpracovává, dostal do stejné situace jako obrazový zdroj před ním, protože by potřeboval jen znát, kam má data dále předat.



Obrázek 3.1: Schéma objektové struktury

Dalo by se říci, že pak každý objekt sloužící k zpracování obrazu je zároveň i obrazovým zdrojem pro následující objekty v řetězci a tím pádem musí obsahovat referenci na objekt, který po něm následuje, aby mohl zpracovaná obrazová data dále předat. V konečném důsledku pak může dědit metody pro předávání obrazu do následujících částí řetězce od obrazového zdroje.

Díky tomuto by pak programátor jen nadefinoval, jakým způsobem na sebe jednotlivé články řetězce navazují a celý řetězec objektů by zakončil obrazovým výstupem. Obrazový výstup pak s ostatními články řetězce sdílí jen interface, kterým mu budou předána obrazová data.

4.7. Způsob práce s objekty pro zpracování obrazu

To, jak jsou definovány základní typy objektů a způsob interakce mezi nimi, respektive způsob předávání obrazových dat, dává jasně najevo, jak bude definice obrazového řetězce vypadat. Programátor nejdříve vytvoří instanci objektu reprezentující zdroj dat. Dále vytvoří instanci objektu, který si data převezme a nadefinuje vstupní parametry, se kterými v ní bude obraz zpracován. Poté již může v obrazovém zdroji vytvořit referenci na objekt zpracovávající obraz. To je samo o sobě více psaní, než by bylo zapotřebí při procedurálním způsobu předávání dat mezi funkcemi. Je potřeba toto zbytečné psaní omezit.

```
CameraSource *camSrc = new CameraSource();
Blur *blur = new Blur();
blur->amount = 5;
camSrc->setFollowing(blur);
```

Nejprve je nutné se zbavit nutnosti nejprve definovat oba po sobě jdoucí objekty a až poté je zapojit za sebe. To lze provést několika způsoby. Jedním z nich je možnost předat první objekt druhému už v konstruktoru, kde se pak v prvním objektu vytvoří reference na druhý. To ušetří programátorovi jeden řádek psaní.

```
CameraSource *camSrc = new CameraSource();
Blur *blur = new Blur(camSrc);
blur->amount = 5;
```

Stále je ale potřeba definovat proměnné, ve kterých budou existovat reference na objekty, aby bylo možné objektům nastavovat parametry. To lze sice provést už v konstruktoru, ale bylo by to značně nepraktické, zvláště pokud by bylo parametrů mnoho. Ideální by bylo, kdyby programátor nemusel vůbec definovat proměnné pro ukládání těchto objektů a mohl nastavovat parametry elegantněji, například stylem Fluent interface. Protože ale na konstruktoru samotném nelze volat žádné metody, to lze až na konkrétní proměnné obsahující referenci na instanci, je nutné získat instanci objektu jinak. Nejlépe bez nutnosti volat konstruktor. To lze provést tak, že instanci vrátí statická metoda - Factory. Taková statická metoda může obsahovat volání konstruktoru a jen vracet samotnou instanci.

```
CameraSource *camSrc = new CameraSource();
camSrc->setFollowing(Blur::newBlur()->setAmount(5));
```

Pak už se jen potýkáme s problémem, že nelze za objekt řešící zpracování obrazu zavěsit další článek řetězce. Řešení je ale jednoduché. Metoda, kterou se vytvoří ve zdroji obrazu reference na následující článek řetězce vrátí právě onen následující článek řetězce. To pak dokáže maximálně zpřehlednit kód. Potíž může nastat, když bude programátor potřebovat řetězec rozvětvit. Pak už musí vytvářet proměnné pro uzly, ve kterých se řetězec větví.

```
CameraSource::newCameraSource()
    ->setFollowing(Blur::newBlur()
        ->setAmount(5))
    ->setFollowing(Saturation::newSaturation()
        ->setSaturation(255));
```

Pro případy větvení kódu, ale i pro obecné zpřehlednění složitějšího kódu pak může být příjemným nástrojem objekt, který bude reprezentovat právě jednu přímou cestu v řetězci, do kterého by programátor postupně vkládal reference na jednotlivé části řetězce. Taková třída je se pak stává základem celého řetězce a logicky je i dalším elementárním článkem. Aby ho bylo možné takový kontejner zapojit do stávající struktury s obrazovými zdroji, výstupy a objekty pro zpracování obrazu, je nutné ho vybavit stejnými rozhraními, jako mají jednotlivé články řetězce. To dává tomuto kontejneru článků nový rozměr, protože se tímto sám stává článkem řetězce.

Kontejner sekvence článků řetězce pro zpracování obrazu je pak základem pro další vrstvy abstrakce nad procesem zpracování obrazu. Uživatel programátor totiž může vytvořit vlastní třídu, která bude dědit od tohoto kontejneru a v konstruktoru už definuje, které konkrétní články má tento řetězec obsahovat. Vznikne tak nový článek, řešící konkrétní problematiku zpracování obrazu, který bude znovupoužitelný.

4.8. Napojení na OpenCV

Zvolený objektový charakter frameworku jde velmi proti funkčnímu procedurálnímu řešení OpenCV. Je potřeba, aby bylo zachováno objektové řešení a zároveň zůstala možnost využívat funkce OpenCV. Dále bude nutné zajistit kompatibilitu přenášených dat v kontejneru se strukturami používanými OpenCV frameworkem.

Zřejmě nejjednodušším řešením, jak využít stávající funkce OpenCV, ale zachovat objektový ráz frameworku bude obalování jednotlivých OpenCV funkcí do jednotlivých tříd článků řetězce. Ideálně, aby každé funkci, či skupině funkcí plnících společný cíl odpovídala právě jedna třída. Uživatelé, kteří mají s OpenCV zkušenost, pak mezi názvy tříd rozpoznají původní OpenCV základ a budou ho schopni využít.

Dále pak musí být zajištěno, aby data na nejnižší úrovni abstrakce byla s OpenCV kompatibilní. Primárně jde o cv::Mat, který reprezentuje bitmapu rámce obrazu. Matice cv::Mat ale nejsou stavěny pro držení v referenci na C heap, což by dost celý projekt limitovalo, takže bude nutné všechny základní datové typy OpenCV reprezentující obrazová data nějakým způsobem obalit. K tomu nám může posloužit právě kontejner obrazových dat. Tím by všechny problémy spojené s integrací OpenCV do frameworku měly být vyřešené.

4.9. Shrnutí koncepce řešení

Kvůli zajištění kompatibility s více platformami a pro umožnění přechodu širší skupiny uživatelů (těch, kteří se již seznámili s OpenCV) k framework u bude framework bude objektovou nástavbou nad OpenCV. To zajisti, že bude framework jednoduše rozšiřitelný a odborníci znalí problematiky Augmented Reality tím budou mít zjednodušené nejen zjišťování, jak framework funguje uvnitř, ale i jeho eventuelní rozšiřování. Jako jazyk bude použito C++, které je objektové a je možné ho portovat na tři nejvýraznější platformy (iOS, Android, Windows Phone). Kvůli přehlednosti celého konceptu budou využity zejména principy objektově orientovaného designu SOLID [16].

Základní objektová struktura pak bude postavena na datovém typu reprezentující zdroj obrazových dat, z něj dědící článek řetězce sloužící k jejich zpracování, interface definující výstup obrazových dat, dále kontejner nesoucí obrazové a jiné informace a nakonec kontejner jednotlivých článků řetězce zpracování obrazových dat. Články, respektive konkrétní třídy sloužící ke zpracování obrazových dat pak budou obalovat stávající OpenCV funkce.

K vytvoření instance objektu pro zpracování dat bude sloužit statická Factory metoda a parametry jednotlivých objektů se budou nastavovat settery ve stylu Fluent Interface. Jednotlivé články řetězce pak budou pospojovány do stromové struktury prostřednictvím referencí na následující článek řetězce.

5. Popis řešení

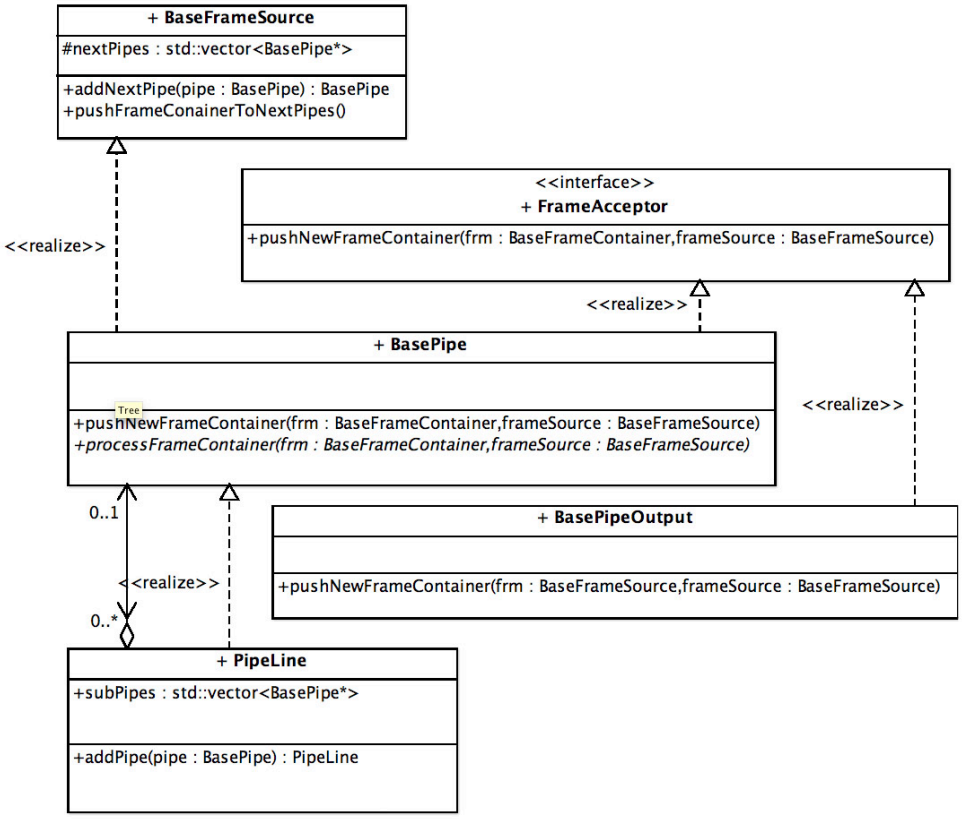
Jak bylo již řečeno, celý framework se ponese v objektovém duchu, kde sama struktura objektů bude vysvětlovat procesy, jimiž je obraz zpracováván. Objektová struktura celého řetězce zpracování obrazu pak bude tvořit strom, respektive řetězec referencemi spojených objektů.

Dále kvůli přenositelnosti aplikace mezi platformami bude k implementaci použit jazyk C++ a framework bude obalovat existující knihovnu OpenCV.

5.1. Základ architektury

Řešení postavené na bázi pipeline, kdy vždy existuje zdroj obrazu, bloky které ho zpracovávají a výstup, který může být buď ve formě obrazu, či 3D renderu. To definuje základní koncepci objektového modelu. Obraz vzniká v objektu typu BaseFrameSource, což je Factory třídou pro objekt typu FrameContainer, který obsahuje obrazová data a další parametry, které je nutné mezi jednotlivými metodami přenášet. Za každý zdroj obrazu typu BaseFrameSource pak lze navázat jeden nebo více objektů pro zpracování obrazu, které jsou typu BasePipe, který dědí od třídy BaseFrameSource a které je možné dále řetězit. To že třída BasePipe dědí od třídy BaseFrameSource jen zdůrazňuje fakt, že je každý uzel stromu je zároveň zdrojem obrazových, či jiných AR dat. Může tak vzniknout celá stromová struktura s kořenem v právě jednom zdroji obrazových dat typu BaseFrameSource, který dovolí moduly (objekty) pro zpracovávání obrazu nejen řetězit, ale i větvit. Každá vzniklá větev je pak zakončena objektem typu BaseFrameOutput, který se pak stará o předání správných dat k zobrazení.

Aby bylo možné abstrahovat nad elementárními procedurami zpracování obrazu, je třeba vytvořit ještě objekt, který bude schopný právě instance typu BasePipe pojmout a všechny je spojit do jednoho řetězce. Řešením je třída PipeLine, která dědí od třídy BasePipe, tudíž ji je možné samotnou zapojit do řetězce jako jeden modul. Dovoluje dokonce řetězit i hotové objekty typu PipeLine.



Obrázek 4.1: Základní objektový model

Třída Pipeline tak uspokojuje požadavek na škálovatelnost řešení. Díky ní je možné teoreticky do nekonečna abstrahovat nad dílčími kroky zpracování obrazu a spojovat je do konkrétních celků. Samotná stromová struktura dává začátečníkovi jednoduchý pattern, jakým způsobem má s frameworkem pracovat.

5.2. Přenos a zpracování obrazových dat

Jak již bylo uvedeno, obrazová data jsou zapouzdřena v objektu BaseFrameContainer, který primárně obsahuje pouze objekt Frame, který je kontejnerem obrazových dat reprezentovaných strukturou cv::Mat z frameworku OpenCV. Obrazová data jsou zpracovávána uvnitř objektů typu BasePipe, které v metodě processFrameContainer modifikují objekt BaseFrameContainer a posílají ho dále.

Princip je následující. Objekt dostane zodpovědnost za BaseFrameContainer v momentě, kdy je z venčí objektem typu BaseFrameSource zavolána metoda pushNewFrameContainer, kde jsou v parametrech předány údaje o zdroji obrazových dat a samotná obrazová data v BaseFrameContaineru. Samotná metoda pushNewFrameContainer je pak zodpovědná za zavolání procedury zpracování obrazu a předání dat do dalších uzlů stromu. Zpracování obrazových dat pak probíhá ve virtuální metodě processFrameContainer, která je klíčem k abstrakci nad základními funkcemi OpenCV.

Každá komponenta, respektive uzel stromu, je podtypem třídy BasePipe a na nejnižší úrovni abstrakce, tedy v momentě, kdy je třeba obalit základní funkce, které poskytuje OpenCV, by měla metoda processFrameContainer provést právě jednu konkrétní operaci definovanou OpenCV funkcí.

Po provedení zpracování obrazových dat, je pak zavolána metoda pushFrameContainerToNextPipes z rodičovské třídy BaseFrameSource, která se stará o předání kontejneru s obrazovými daty do navazujících objektů. Objekt třídy BaseFrameSource totiž obsahuje pole objektů typu BasePipe, které definuje uživatel, a říká tím, do kterých dalších uzlů stromu se mají data poslat ke zpracování dále. V metodě pushFrameContainerToNextPipes se tak řeší eventuelní uvolnění paměti v případě, že už neexistují další uzly, kam by bylo možné data předat (tj pole s BasePipes je prázdné), nebo v případě že se má za vlastním uzlem strom větvit tak řeší naklonování BaseFrameContaineru.

Toto řešení, že každý objekt obaluje jednu, respektive více funkcí, které jsou ale jen různými implementacemi plnící stejný účel, umožňuje expertům znalých problematiky OpenCV prohlédnout, co se za konkrétními uzly řetězce skrývá a dovoluje jim jednoduše vytvářet další vlastní moduly.

5.3. Principy jednotlivých uzlů řetězce, Pipes

Každý Pipe objekt, tedy objekt typu BasePipe, který má za úkol nějakým způsobem zpracovat či modifikovat obrazová data by se měl držet uvedených základních principů, které zajišťují právě splnění cílů frameworku.

5.3.1. Statická metoda init vracející instanci objektu

Aby bylo možné využít Fluent interface, nechceme nikde vytvářet novou proměnou a instanci inicializovat konstruktorem, na kterém už nelze hned volat další metody. Řešení s továrničkou init dovoluje na objektu volat metody bezprostředně po jeho vytvoření bez nutnosti přiřazovat ho proměnné. To vytváří pořádek v kódu a zkracuje tak definici hlavní smyčky na co nejmenší počet řádků.

Metoda init by v ideálním případě neměla vyžadovat žádné další parametry s výjimkou parametrů bezpodmínečně nutných k vytvoření objektu schopného okamžitě začít pracovat. Například rotace obrazu logicky k běhu vyžaduje znát úhel, o který má obraz otočit. Pokud by tedy měla metoda init vyžadovat parametry, měly by vyplývat z podstaty funkce, aby uživateli bylo jasné, že funkce bez těchto parametrů nemůže fungovat.

5.3.2. Žádné public properties, ale jen settery vracející vlastní instanci

Základem pro Fluent interface je, že každý parametr objektu lze nastavit setterem, který vrací vlastní instanci objektu aby umožnil ihned v závěsu zavolat další setter. Tím lze pak velice rychle a přehledně nastavovat všechny parametry na jednom místě. Navíc uživatel hned vidí, které parametry může objektu nastavit a navíc při ladění nemusí procházet velké množství kódu. Parametry objektu jsou pak chráněné a settery mohou eventuálně pozměnit i více parametrů najednou.

5.3.3. Pipe objekt obsahuje výchozí nastavení

Díky výchozím nastavení každého objektu musí být objekt schopný fungovat už v momentě vzniku instance, tj bez jakéhokoliv nastavování parametrů musí být ihned možné Pipe zapojit do řetězce. Jsou samozřejmě výjimky, například u funkce Rotate, kde se předpokládá, že každému dojde že u otáčení obrázku je třeba říci, jak ho otočit.

To výrazně zrychluje celý proces učení, protože začátečníkovi stačí vyrobit jen instanci a hned může vidět efekt.

5.4. Způsob vytváření nových Pipe objektů, podtříd BasePipe

Třída BasePipe sama o sobě zajišťuje předávání obrazových dat uložených v BaseFrameContaineru a definuje abstraktní (virtuální) metodu processFrameContainer. Tato metoda je zavolána v momentě, kdy objekt dostane ke zpracování nový BaseFrameContainer a její úloha spočívá pouze v modifikaci obrazových dat. Nemusí totiž žádná data vracet (je návratového typu void). Obsahuje tedy interface metodu pushNewFrameContainer, která se stará o zavolání metody processFrameContainer a předání objektu do dalších pipes. Dále, protože je metoda processFrameContainer veřejná (public), je možné ji invokovat i z venčí, což využívají objekty typu Pipe.

Limitace, které bohužel vytváří v objektovém modelování jazyk C++ bohužel nedovolují dynamicky vytvářet jednotlivé dílčí metody, které jsou vyžadovány principy frameworku a tak je důležité, aby programátor principy respektoval. Naštěstí jsou dostatečně jednoduché a jejich dodržování má přímý vliv na komfort používání rozšiřujících tříd, že nemá žádný smysl je obcházet či nedodržovat.

Postup lze nejjednodušeji ilustrovat na na třídě PolarRotate, která dokáže prostřednictvím OpenCV funkcí flip a transpose [17] rotovat obraz v krocích po 90-ti stupních. Ta obsahuje konstruktor, který svými parametry odpovídá statické metodě init, dále právě onu statickou metodu init vracející novou instanci typu PolarRotate a metodu setDeg. Metoda setDeg s jediným parametrem degrees nastavuje instanční proměnnou určující úhel rotace a zajišťující validaci vstupního parametru a vrací vlastní instanci třídy PolarRotate, aby bylo možné ji dále použít.

Posledním a nejklíčovějším momentem v celé třídě je implementace virtuální metody processFrameContainer. Ta obsahuje volání OpenCV funkcí flip a transpose v závislosti na nastavení instanční proměnné degrees.

Toto jsou všechny bezpodmínečné kroky směřující k vytvoření vlastní podtřídy BasePipe, která dodržuje principy pro uzly řetězce, Pipes pro metodu s jedním parametrem.

5.5. Koncepce obalování OpenCV funkcí

Jak bylo řečeno, každý Pipe objekt obaluje buď jednu konkrétní OpenCV funkci, nebo více funkcí, které ale plní společný cíl. To lze velice dobře ilustrovat na třídě Threshold, která obaluje OpenCV funkce adaptiveThreshold [18] a threshold, které jsou jen různými implementacemi této metody zpracování obrazu. Důležité je říci, že nejpalčivějším momentem není ani seskupování OpenCV funkcí do logických celků, protože ty se mohou prolínat, ale právě výchozí nastavení jejich parametrů. Obě threshold funkce totiž vyžadují několik parametrů, které určují metodu, která je k segmentaci obrazu použita a určují hranici, která obraz segmentuje.

Tyto údaje samy o sobě vyžadují jistou zkušenost, aby vůbec bylo možné funkci zprovoznit, proto je potřeba je za uživatele nastavit předem a jen mu nechat možnost je modifikovat. Tedy je nutné předem tyto parametry definovat tak, aby uživatel mohl jen Pipe objekt zapojit do řetězce a až posléze jen upravil jeho chování tak, aby mu plně vyhovovalo. To vše za možnosti ihned po kompilaci a spuštění programu vidět výsledek. Threshold má pak předem nastavené použití OpenCV implementace adaptiveThreshold a automaticky dopočítává podle barevné hloubky vstupních dat maximální hranici.

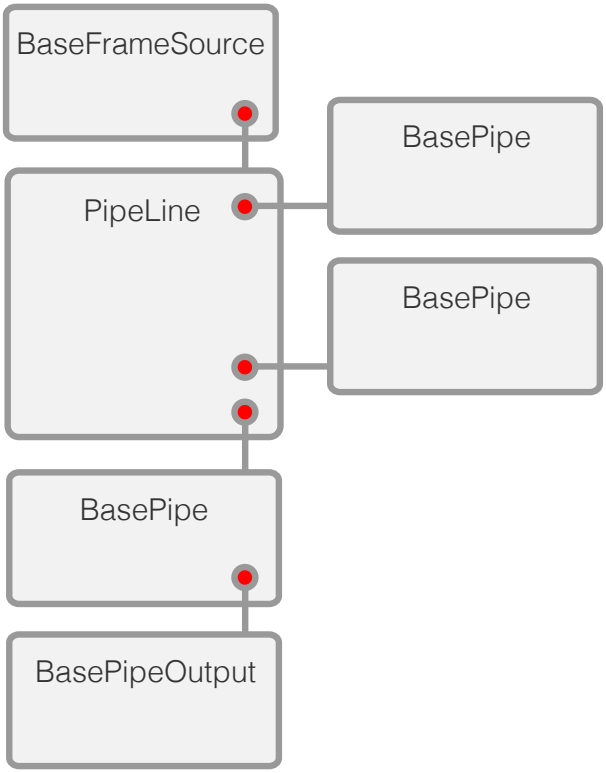
Pomocí vhodně zvolených názvů setterů, jako například setAdaptiveBlockRadius je pak uživateli přesněji vysvětleno, k čemu parametr slouží. Zajímavostí je pak nastavení parametrů výčtového typu prostřednictvím setterů. Protože by uživatel musel dohledávat výčty, je lepší poskytnout mu možnost nastavení takového parametru settery, které sjednocují podobné konstanty ve výčtu. Například pro nastavení typu implementace threshold funkce na typ “Binary” existují konstanty THRESH_BINARY_INV a THRESH_BINARY. Pro nastavení tohoto typu implementace pak existuje jediný setter a to setTypeBinary s parametrem typu boolean, který určuje, zda použít invertovanou variantu této implementace.

5.6. Spojování Pipes do Pipelines a abstrakce nad funkčními celky

Aby byla práce s jednotlivými Pipes a jejich spojování jednodušší, existuje zde třída PipeLine, jejíž základem je vektor jednotlivých BasePipes. Třída PipeLine je zároveň podtřídou BasePipe, tudíž je možné ji samostatně používat jako samostatnou Pipe.

Pro přidání objektu typu BasePipe slouží metoda addPipe která vrací vlastní instanci PipeLine a dovoluje tak pomocí Fluent Interface přidání dalších objektů typu BasePipe. Pro následnou práci s objekty ve vektoru dovoluje třída PipeLine přístup do vektoru prostřednictvím operátoru hranatých závorek.

Pokud programátor nepotřebuje svůj proces nijak větvit, je třída PipeLine nejpřirozenější volbou, jak jednotlivé instance BasePipe držet a pracovat s nimi. Dokonce i v případě, kdy je potřeba program větvit, lze jednotlivé větve reprezentovat objekty typu PipeLine.



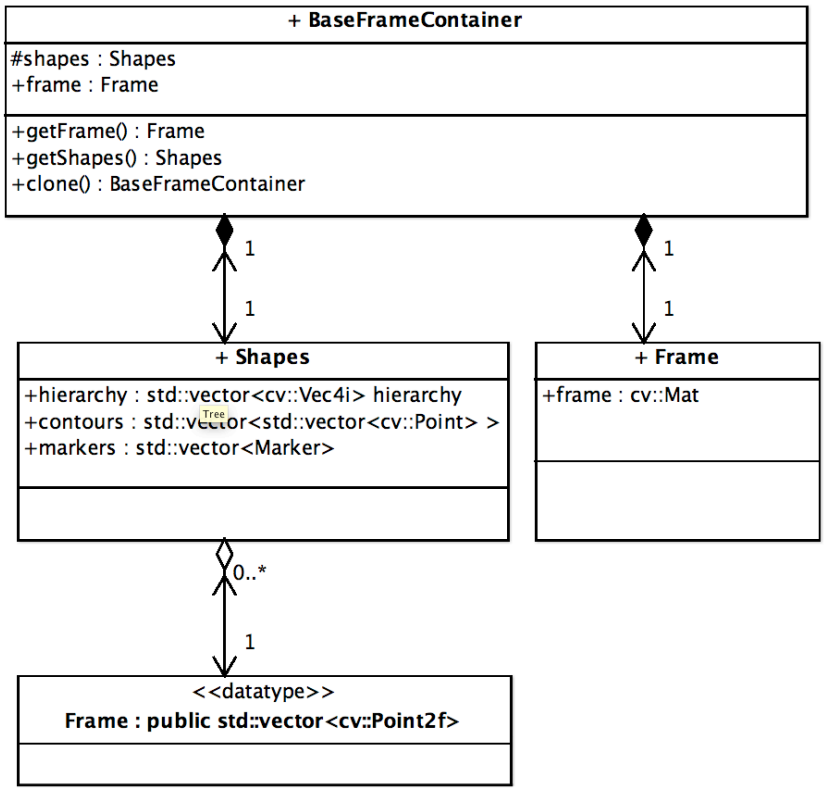
Obrázek 4.2: Princip fungování PipeLine

Pro zjednodušení práce s konkrétními funkčními celky si může programátor vytvořit vlastní podtřídu PipeLine, do jejíhož konstruktoru umístí definici jednotlivých BasePipes, které tato nově vzniklá třída bude obsahovat. Aby pak bylo možné modifikovat z venčí konkrétní parametry, stačí se dotázat prostřednictvím operátoru hranatých závorek na příslušný BasePipe objekt a modifikovat ho, nebo může programátor připravit zkratkovou metodu, která uživateli dovolí parametr některé z BasePipes nastavit.

5.7. Získávání a práce s rozpoznanými objekty v obrazových datech

Zobecněně lze říci, že při zpracování obrazových dat je třeba nějakým způsobem rozpoznávat objekty v obraze ty dále filtrovat a vybírat z nich právě ty, které odpovídají těm hledaným objektům. Každý takový nalezený objekt je definován polem bodů a může být volitelně zařazen v hierarchii. Taková data je tedy také nutno předávat od místa jejich vzniku k místu, kde jsou zobrazena. A to může být provedeno jak renderem do samotných obrazových dat, tak renderem do 3d vrstvy nad obrazovými daty.

Ve frameworku je přenášení těchto dat podobně jako u dat obrazových. Samotná značka, respektive kontura tvořená jednotlivými body je reprezentovaná třídou Marker, která dědí od datového typu std::vector<cv::Point2f>, je tedy s takovýmto vektorem, který používá OpenCV například ve funkci drawContours [19], kompatibilní. Třídu Marker si framework vypůjčuje v drobně pozměněné podobě od frameworku Aruco [11], protože je pro tento účel naprosto ideální.



Obrázek 4.3: Objektový model kontejneru dat

Vektor Markerů je pak pod názvem Markers nesen společně s dalšími daty souvisejícími s rozpoznanými fragmenty v obrazových datech v objektu typu Shapes. Třída Shapes je pak nositelem informací o rozpoznaných konturách, o jejich hierarchii a nakonec nositelem konkrétních rozpoznaných značek. Konkrétně, informace o rozpoznaných konturách jsou reprezentována standardizovaným vektorem definovaným v OpenCV jako std::vector<std::vector<cv::Point> > [20], jež lze získat z Shapes pomocí getteru getCoutours, dále informace o hierarchii kontur je držena v datovém typu std::vector<cv::Vec4i> taktéž definovaném v OpenCV dokumentaci, jež lze získat getterem getHierarchy.

Instance typu Shapes je pak nesena společně s instancí obrazových dat Frame v BaseFrameContaineru. Takové strukturování pak dovoluje v jednotlivých Pipes tato data získat, postupně filtrovat a následně zobrazit.

5.8. Problém opětovného spojení stromu v uzlu

Při zpracování obrazu velmi často vzniká problém, kdy je potřeba větve strom opět v jednom uzlu spojit. Tato potřeba vzniká v momentu, kdy je obraz zpracováván odděleně od rozpoznaných artefaktů. Řešením je objekt, který počká se zpracováním dat až do chvíle, kdy dostane k dispozici všechna data a až teprve potom tato data správně spojí a odešle.

Prototypem tohoto objektu je PipeMerge, který funguje tak, že programátor definuje, z které předchozí Pipe přijde obraz a ze které přijdou rozpoznané artefakty. PipeMerge pak, v momentě kdy má oboje získaná data k dispozici, tato data spojí a odešle dále.

5.9. Příklad použití frameworku na mapování 3D objektu

Na následujícím útržku zdrojového kódu je jednoduchým způsobem ilustrováno, jak být může relativně komplexní aplikace zpracovávající obraz přehledně napsána prostřednictvím objektů frameworku. Příchozímu uživateli pak je ihned jasné, jakým způsobem probíhá zpracování obrazu, jak jsou rozpoznávány jednotlivé značky a jak jsou data zobrazována.

```
CameraFrameSource *frameSource = [[CameraFrameSource alloc] init];

BaseArView *previewLayer = [[BaseArView alloc]
     initWithFrameAndCaptureSession: self.view.frame
     captureSession: [frameSource captureSession]];

ArPipe::PipeLine* pipeline =
     new ArPipe::PipeLine([frameSource frameSource]);

pipeline->addPipe(ArPipe::PolarRotate::init(90));
pipeline->addPipe(ArPipe::BlackAndWhite::init());
ArPipe::BlackAndWhite *blackAndWhite =
             (ArPipe::BlackAndWhite*) pipeline->back();

pipeline->addPipe(ArPipe::Threshold::init());
pipeline->addPipe(ArPipe::FindContours::init()
                     ->setTypeTree());
pipeline->addPipe(ArPipe::DetectPolygons::init()
                    ->setOnlyConvexObjects()
                    ->setRequiredSideCount(4)
                    ->setComplexityKoef(0.5));

ArPipe::FiducidalMarkerIdentifier *mId =
     (ArPipe::FiducidalMarkerIdentifier*)
     pipeline->addNextPipe(ArPipe::FiducidalMarkerIdentifier::init()
                      ->setFrameSource(blackAndWhite)
                      ->setShapesSource(pipeline));

blackAndWhite->addNextPipe(mId);

ArPipe::CameraApply *camApply = (ArPipe::CameraApply*)
     mId->addNextPipe(ArPipe::BlackAndWhite::init()->toColor())
             ->addNextPipe(ArPipe::CameraApply::init());

camApply->addNextPipe(ArPipe::DrawContours::init())
        ->addNextPipe([previewLayer pipeConnector]);
```

Na tomto příkladě je vidět, že se je zde použita jedna stěžejní PipeLine, ve které jsou detekovány jednotlivé značky. PipeLine je zakončena detektorem značek, do kterého je oklikou přenesen jen otočený černobílý obraz pomocí PipeMerge, která pak obrazová data správně spojí a odešle dále. Nakonec jsou rozpoznané artefakty vykresleny do obrazu.

Vykreslení rozpoznaných značek, respektive jejich předání k vykreslení v GL vrstvě pak probíhá dle následujícího příkladu.

```
for (unsigned int i = 0;
             i < frm->getShapes()->getMarkers().size(); i++)
{
     double model_matrix[16];
     frm->getShapes()
             ->getMarkers()[i].glGetModelViewMatrix(model_matrix);
     [arv.glview addModelViewMatrix:model_matrix];
}
```

Rozpoznané objekty stačí pak jeden po druhém iterovat, dotázat se na jejich matici určující pozici a tato data předat 3D vrstvě, která má na starost renderování. Marker také může obsahovat metadata, která mohou konkrétní objekt identifikovat a například upřesnit konkrétní zobrazený objekt.

6. Testování

6.1. Kritéria ke splnění

Ze zadání jasně vyplývá, že primárním kritériem určujícím úspěšnost řešení je urychlení vývoje a dále s tím spojená dostupnost použitých nástrojů pro širší komunitu.

Vzhledem k to mu, že je framework vyústěním mé předešlé semestrální práce řešící právě AR problém detekce značky a mapování 3D objektu na značku, je ideálním testem implementace stejného řešení prostřednictvím vzniklého frameworku a porovnání s původním řešením. Původní řešení totiž bylo relativně časově náročné a sama implementace není relativně složitá. Porovnání obou aplikací nezávislým subjektem z hlediska rychlosti porozumění kódu pak jednoznačně určuje agilitu frameworku oproti jiným řešením. Rychlost porozumění kódu je totiž v přímé úměře se strmostí křivky učení a tím i rychlostí samotného vývoje aplikací.

Druhý test bude více zaměřen na začátečníky a půjde o test zprovoznění jednoduché implementace modifikace obrazu z kamery zařízení a výstup na jeho displej. Účelem testu bude zjistit, zda je snížena vstupní bariéra pro programátora.

Charakter obou testů je víceméně stavěn tak, aby ověřil předpokládané přednosti frameworku, které zároveň jsou kritériem pro splnění zadání práce, tudíž jejich selhání by znamenalo, že zvolené řešení není úspěšné.

6.2. Porovnání implementací řešení AR problému nezávislým subjektem.

6.2.1. Popis AR problému a původní implementace

Cílem aplikace je rozpoznat ve videu standardizovanou značku a mapovat na ní 3D objekt, v tomto případě krychli. Pro implementaci řešení problému bude v obou případech jako platforma zvolen iOS a jako základní framework OpenCV. Ve videu z kamery budou tedy rozpoznávány značky a následně bude ve 3D vrstvě vykreslena krychle.

Původní implementace využívá základní funkce OpenCV a jednoduchou jednoúčelovou nástavbu Aruco, která částečně řeší jednotlivé podproblémy. Architektura je spíše procedurálního charakteru a je využito vyšší vrstvy v ObjectiveC, respektive AVFoundation frameworku k vytvoření základní smyčky programu.

6.2.2. Kritéria testu

Jako nezávislý subjekt byl zvolen programátor se zkušenostmi s vývojem aplikací v Javě, PHP a samozřejmě v C a C++. Programátor má obecný přehled v metodách zpracování digitálního obrazu a komplexní znalosti objektového programování.

Test spočívá v úkolu rozpoznat a určit umístění důležitých procesů ve zpracování obrazu ve smyčce. Potřebným výsledkem testu je převaha nově vzniklého frameworku nad původním řešením. A protože jsou obě řešení v principu podobná, bude nejdříve subjektu předloženo nové řešení a následně řešení původní. Pokud by pak byly obě implementace z hlediska přehlednosti srovnatelné, aplikace která by byla předložená jako druhá v pořadí musí být zákonitě pochopena rychleji než ta první. V tomto testu pak i přes toto znevýhodnění aplikace předložené jako první je očekáván kratší čas pochopení prvního řešení (s použitím frameworku).

Během testu budou zaznamenány časy zjištění elementárních funkcí:

3. použití thresholdovací metody Canny
4. způsob detekce čtverců
5. identifikace značek (markerů)
6. místo, kde jsou markery renderovány

Zároveň bude sledován celkový čas. Tester bude před započetím testu znát pouze účel aplikace a k dispozici dostane informace, že aplikace využívá OpenCV framework a že proces, kterým je obraz zpracováván velice podobný.

6.2.3. Průběh testu

Nejprve byla osobě předložena implementace za pomoci nového frameworku. Vzhledem k tomu, že celá definice Pipeline je obsažena v hlavním ViewControlleru, programátor odhaluje funkci Canny velmi rychle, stejně jako OpenGLES vrstvu, kde je renderován 3d objekt. Osoba váhá nad metodou, která obsahuje detekci čtverců a nesprávně otevírá implementaci FindContours, ale po zjištění účelu funkce z dokumentace OpenCV se správně vrací do implementace DetectPolygons a správně identifikuje proces detekce čtverců. Nakonec se znovu vrací zpět do hlavního ViewControlleru a po krátké prodlevě otevírá SimpleMarkerIdentifier. Tím končí první část testu.

V druhé části testu programátor automaticky znovu otevírá hlavní ViewController. Po hlubším průzkumu jeho implementace objevuje nastavení snímání obrazu a callback zpracovávající obraz. V callbacku identifikuje identickou třídu sloužící k vykreslování Markeru ve 3D. Následně otevírá proceduru sloužící ke zpracování obrazu a detekci Markerů. V proceduře už relativně rychle identifikuje thresholdovací metodu Canny a procedůru detekce čtverců z kontur. Kvůli složitější přístupnosti funkce pro identifikaci markerů pak programátor nachází algoritmus po relativně dlouhé prodlevě.

6.2.4. Výsledek testu

Výsledků testu jasně vyplývá, že s použitím frameworku se stává kód přehlednějším a pochopitelnějším. Protiargumentem může v tomto případě být, že za přehlednost kódu ručí sám programátor, ale v tomto případě dává framework programátorovi jasný příklad, jakým způsobem má kód psát. Nemusí se tedy předem rozmýšlet, jakým způsobem bude aplikace obraz zpracovávat a podle toho navrhovat architekturu své aplikace. Velmi často totiž je třeba si myšlenku nejdříve ověřit na prototypu a až poté implementovat.

	s Frameworkem		bez Frameworku		
	v čase	trvání	v čase	trvání	přínos
Metoda Canny	0:29	0:14	2:17	1:07	79 %
Detekce čtverců	1:22	0:53	2:50	0:33	-38 %
Identifikace Markerů	1:38	0:16	4:09	1:19	80 %
Renderování	0:44	0:15	1:10	1:10	79 %
Celkem	1:38		4:09		60 %

Tabulka 6.1: Výsledek testu porozumění kódu

Výsledek zrychlení procesu porozumění cizího kódu v o šedesát procent kratším čase za vstupní podmínky, že obě implementace jsou více méně totožné po funkční stránce, ale rozdílné po stránce architektury a programátor před prozkoumáním druhé implementace byl oproti předchozí zvýhodněn její znalostí a mohl se o ni opřít, je dostatečně výmluvným výsledkem, který není třeba dokládat na širším vzorku subjektů.

Lze jednoznačně říci, že framework na základě tohoto testu urychluje vývoj CV a AR aplikací.

6.3. “Hello World” test

6.3.1.Popis testu

Test “Hello World” má za cíl zjistit, jak moc obtížné je pro začátečníka vytvořit a spustit základní aplikaci implementující jednoduchou elementární metodu modifikující obraz za předpokladu, že cílová funkce není ve frameworku obsažena. Programátor, který bude účastníkem testu, dostane za úkol zprovoznit základní řetězec zpracování dat obrazu s následujícím cílem: modifikovat barevný odstín obrazu (Hue) a následný výstup obrazu na displej telefonu. Filtr pro modifikaci odstínu nebyl v momentě testu programátorovi k dispozici.

6.3.2.Kritéria testu a vstupní podmínky

Platformou v tomto případě bude Apple iOS a programátorem je nyní zkušený iOS vývojář zaměřený na herní prostředí, který má zkušenosti s ObjectiveC, tak i základní znalosti problematiky zpracování obrazu. Nedisponuje ovšem znalostmi OpenCV, ani jiného CV nástroje.

Vývojář dostane k dispozici dokumentace k oběma nástrojům a k frameworku navíc k dispozici sandbox, který je jeho součásti. Kritériem pak bude čas zprovoznění implementace na zařízení. Programátor jako u předchozího testu začne implementací s použitím nového frameworku. Test proběhne na počítači s vývojářským certifikátem, takže kompilace a deploy na zařízení proběhne stiskem jednoho tlačítka.

6.3.3. Průběh testu a jeho výsledek

S použitím frameworku byl programátor hotov relativně rychle, v čase 9:32 minut. Problém nastal v momentě, kdy bylo třeba do-implementovat chybějící filtr. Programátor použil vyhledávač a zkopíroval výsledné řešení podle návodu do vlastní nové podtřídy BasePipe. V tomto případě se inspiroval implementací filtru Blur.

Ve druhé části testu musel programátor navíc dohledat v dokumentaci Apple způsob jakým snímat obraz z kamery, jak ho převést na OpenCV matici cv::Mat. Implementaci funkce pro modifikaci barevného odstínu použil z předchozí části testu, takže si část práce ušetřil. Přesto mu implementace zabrala 32:47 minut. Nejvíce času programátorovi zabrala právě nutnost dohledávat napojení existující platformy na knihovnu OpenCV.

Z výsledků lze vyvodit, že největší vstupní bariérou pro programátora, který se hodlá seznámit s AR aplikacemi je právě absence jednoduchého sandboxu, který dovede pomoci vytvořit samotnou smyčku zpracování obrazu. Absence konkrétního filtru ve frameworku pak nepředstavovalo nepřekonatelný problém.

Důležité je zopakovat, že se jedná o vývojáře, který má s iOS platformou dostatek zkušeností.

7. Srovnání s existujícími řešeními

7.1. Srovnání rozsahu knihovny a pokrytí potřeb programátora

Nejprve se nabízí srovnání s OpenCV, které je prakticky základem frameworku a framework využívá jeho funkce. Protože framework prakticky obaluje OpenCV knihovnu, měl by mít funkční rozsah stejný. Nicméně v tuto chvíli je framework pouhým základem, protože zatím nepojímá ani polovinu funkcí nabízených OpenCV, ale záleží jen na eventuelní činnosti OpenSource vývojářské komunity, která může framework obklopit a která může chybějící moduly velice rychle podle nastavených pravidel doplnit. Přeci jen jde o systematické obalování OpenCV funkcionalit.

V porovnání s ARToolKitem je OpenCV základ jasnou předností, která dovolí mnohem širší pole využití. Specializace ARToolkitu sice usnadňuje některé dílčí činnosti v rámci AR, ovšem šíře záběru OpenCV, jako základu, dovoluje lepší optimalizace procesů a eventuelní větší pole působnosti.

7.2. Možnosti abstrakce a šíře vývojářské základny

Přijetí každého produktu veřejností je vždy sázkou do loterie a u nově vzniklého vývojářského nástroje není situace nijak jiná. Framework OpenCV je v tuto chvíli používán u 1575-ti projektů na opensource komunitním vývojářském serveru GitHub [21], u frameworku ARToolKit toto číslo dělá 43. Na základě těchto čísel lze říci, že kolem OpenCV existuje čilá vývojářská komunita, na rozdíl od ARToolKitu. Právě důvod, že OpenCV používá široká řada vývojářů by mohlo být dobrým předpokladem pro přijetí nového frameworku.

Co se týče možností abstrakce, kdy OpenCV, ani ARToolKit kvůli své neobjektové charakteristice nedávají vývojáři žádný návod, jak knihovny rozšiřovat. To je příčinou roztříštěnosti jednotlivých rozšíření. V tomto má nově vzniklý framework mnoho předností. Díky jasně dané struktuře rozšíření frameworku je pak mnohem jednodušší kvalitní nástavby do frameworku integrovat jako jeho součásti.

7.3. Srovnání z pohledu začátečníka

Z testů, které byly právě zaměřené na ukazatele blízké křivce učení a práci začátečníka s kódem lze jasně říci, že pro nezkušené uživatele je nový framework ideální pro začátky. Razantní zrychlení porozumění kódu a předem daná struktura aplikace předurčují tento framework právě začátečníkům. Toto kriterium jde pak ruku v ruce i se vstupní bariérou, kterou je spuštění první aplikace.

U OpenCV i ARToolKitu chybí jednoduché sandboxy pro zprovoznění Hello World skriptu i tutoriály jak takový skript vytvořit. Z “Hello World” testu vyplývá, že vstupní bariéra zprovoznění první aplikace je dostatečně nízká a motivuje vývojáře dále s frameworkem pracovat. Důležitým kriteriem je v tomto případě existence hotového sandboxu pro konkrétní platformu. Takovýto sandbox v tuto chvíli existuje pouze pro platformu iOS, ale samotný fakt, že připravit a spustit programátorovi smyčku pro zpracování obrazu bez použití frameworku zabralo o 40 minut více času, než bez připravených nástrojů, je velice pravděpodobné, že zkušeného vývojáře platformy Android toto nezaměstná na o mnoho delší dobu.

8. Závěr

Nově vzniklý framework je krokem kupředu po stránce přehlednosti kódu, elegance řešení, i uživatelské přívětivosti k začátečníkům. Svým rozsahem si nezadá se stávajícími nejsilnějšími hráči na poli mobilních řešení pro Computer vision, jako jsou OpenCV, nebo ARToolKit, ale díky tomu, že je nástavbou zmíněného OpenCV, může mít ambice se jim v budoucnu vyrovnat. Samozřejmě, pokud kolem frameworku vznikne v repositáři na GitHubu [22] dostatečně silná komunita vývojářů.

Framework dle testů splnil dva, relativně účelově vytyčené cíle, které ale stojí na realistických základech. Především klíčovým momentem je, že obsažené nástroje dokáží rapidně zlepšit čitelnost i pochopitelnost kódu, která není na úkor škálovatelnosti řešení.

Framework totiž v tuto chvíli není plnohodnotnou knihovnou metod pro Computer vision, ale ukazuje cestu, jakou se má vývojář ubírat a dává mu k tomu nástroje.

Zkratky

UX	Z anglického výrazu user experience, jinými slovy dojem, který je v uživateli vyvolán.
AR	Z anglického výrazu Augmented reality, doslovně přeloženo jako rozšířená realita. Používá se v souvislosti s přetvářením reálného obrazu podle kontextu.
GPS	Mezinárodní satelitní síť pro určování geografické polohy.
CV	Z výrazu computer vision. Znamená počítačové vidění obrazu.
SDK	Z výrazu software development kit, používá se jako označení vývojářské sady nástrojů pro určitou platformu.
iOS	Operační systém používaný firmou Apple na jejích mobilních zařízeních.
API	Zkratka pro aplikační programový interface.
PHP	Interpretovaný programovací jazyk používaný pro vývoj webu.
3D	Trojrozměrné prostředí.
ObjectiveC	Programovací jazyk vycházející z jazyka C, používaný v iOS

Literatura

[1] JAMISON, JAY. Web 3.0: The Mobile Era. In: *Techcrunch.com* [online]. 2012. vyd. [cit. 2013-04-20]. Dostupné z: http://techcrunch.com/2012/08/11/analysis-web-3-0-the-mobile-era/

[2] Google Glass. GOOGLE.COM. *Google.com* [online]. 2012. vyd. [cit. 2013-04-20]. Dostupné z: http://www.google.com/glass/start/

[3] Android And Windows Phone Gain, BlackBerry Loses In Smartphone OS Share According To Kantar. In: *Techcrunch.com* [online]. [cit. 2013-04-20]. Dostupné z: http://techcrunch.com/2013/04/01/android-and-windows-phone-gain-blackberry-loses-in-smartphone-os-share-according-to-kantar/

[4] Katalog mobilů MOBILMANIA.CZ *Mobilmania.cz* [online] 2013. vyd. [cit. 2013-04-20]. Dostupné z: http://www.mobilmania.cz/default.aspcx/katalog-mobilu/

[5] Google Sky Map. GOOGLE.COM. [online]. [cit. 2013-04-20]. Dostupné z: http://www.google.com/mobile/skymap/

[6] EASTER EGG: Yelp Is the iPhone's First Augmented Reality App. In: *Mashable* [online]. 2009. vyd. [cit. 2013-04-20]. Dostupné z: http://mashable.com/2009/08/27/yelp-augmented-reality/

[7] 11 Amazing Augmented Reality Ads. In: *Bussines Insider* [online]. 2012. vyd. [cit. 2013-04-20]. Dostupné z: http://www.businessinsider.com/11-amazing-augmented-reality-ads-2012-1?op=1

[8] Wikipedia - Fluent Interface. FOWLER, Martin. *Martin Fowler* [online]. [cit. 2013-04-20]. Dostupné z: http://www.martinfowler.com/bliki/FluentInterface.html

[9] Nette *Nette,org* [online]. [cit. 2013-04-20]. Dostupné z: http://nette.org

[10] Open CV. *Open CV* [online]. [cit. 2013-04-20]. Dostupné z: http://opencv.org

[11] Aruco. *Aplicaciones de la Visión Artificial* [online]. [cit. 2013-04-20]. Dostupné z: http://www.uco.es/investiga/grupos/ava/node/26

[12] ARToolKIT. *ARToolKit* [online]. [cit. 2013-04-20]. Dostupné z: http://www.hitl.washington.edu/artoolkit/

[13] Unity3D. *Unity3d* [online]. [cit. 2013-04-20]. Dostupné z: http://unity3d.com

[14] Android NDK. GOOGLE. *Android developers* [online]. [cit. 2013-04-20]. Dostupné z: http://developer.android.com/tools/sdk/ndk/index.html

[15] GRASP (object-oriented design). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-04-20]. Dostupné z: http://en.wikipedia.org/wiki/GRASP_(object-oriented_design)

[16] SOLID (object-oriented design). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2013-04-20]. Dostupné z: http://en.wikipedia.org/wiki/SOLID_(object-oriented_design)

[17] Operations on Arrays. *OpenCV Documentation* [online]. [cit. 2013-05-04]. Dostupné z: http://opencv.willowgarage.com/documentation/cpp/operations_on_arrays.html#cv-transpose

[18] Miscellaneous Image Transformations. *OpenCV Documentation* [online]. [cit. 2013-05-04]. Dostupné z: http://opencv.willowgarage.com/documentation/cpp/miscellaneous_image_transformations.html

[19] Drawing Functions: DrawContours. *OpenCV Documentation* [online]. [cit. 2013-05-04]. Dostupné z: http://opencv.willowgarage.com/documentation/drawing_functions.html#drawcontours

[20] Structural Analysis and Shape Descriptors: FindContours. *OpenCV Documentation* [online]. [cit. 2013-05-04]. Dostupné z: http://docs.opencv.org/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=findcontours#findcontours

[21] GITHUB. *GitHub* [online]. [cit. 2013-05-07]. Dostupné z: https://github.com

[22] ArPipe Framework. In: MENGER, David. *GitHub* [online]. [cit. 2013-05-13]. Dostupné z: https://github.com/megii/ArPipe-Framework