VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

KOMIKS KAMERA PRO ANDROID

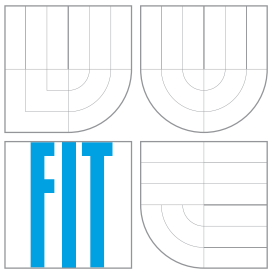
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

LUBOMÍR ŠTEFL

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

KOMIKS KAMERA PRO ANDROID

COMICS CAMERA FOR ANDROID

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — LUBOMÍR ŠTEFL
AUTHOR

VEDOUCÍ PRÁCE — Ing. ISTVÁN SZENTANDRÁSI
SUPERVISOR

BRNO 2015

Abstrakt

Tato bakalářská práce se zabývá návrhem a implementací aplikace komiks kamera v reálném čase pro mobilní zařízení s operačním systémem Android. Práce popisuje základy práce s operačním systémem Android, způsob práce s knihovnou OpenGL ES 2.0, možnost zpracování obrazu pro potřeby této práce a jejich následné použití v aplikaci. Výsledkém této práce je aplikace pro platformu Android, která bude zobrazovat komiksovou kameru v reálném čase s efektem napodobující komiks a při vyfocení se obrázek uloží do paměti mobilního zařízení.

Abstract

This bachelor's thesis deals with the design and implementation of the application Comics camera in real time. The application is designed for mobile device with Android operating system. The thesis describes basic operations with Android, way of working with the OpenGL ES 2.0 library, the possibility of image processing and using this library in the application. The result of thesis is an application for Android platform, which will display Comics camera in real time with the comics effect. After taking a picture the application saves it to the device memory.

Klíčová slova

Android, kamera, komiks, video, mobilní zařízení, OpenGL ES

Keywords

Android, camera, comics, video, mobile device, OpenGL ES

Citace

Lubomír Štefl: Komiks kamera pro Android, bakalářská práce, Brno, FIT VUT v Brně, 2015

Komiks kamera pro Android

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Istvána Szentandrásiho

.....
Lubomír Štefl
19. května 2015

Poděkování

Chtěl bych poděkovat panu Ing. Istvánu Szentandrásimu za vedení této bakalářské práce a za všechny rady a ochotu, kterými přispěl ke vzniku této bakalářské práce.

Obsah

Kapitola 1

Úvod

V této době chytrých telefonů a dalších vychytávek převažuje na trhu právě jeden operační systém a to systém Android, pod hlavičkou firmy Google. Tomuto systému lehce konkuruje na mobilních zařízení pouze systém iOS od firmy Apple. Hardware na mobilních zařízeních se zlepšuje každým dnem a to vybízí k vyvíjení náročnějších aplikací, ovšem ne každý má možnost si tyto nové mobilní zařízení kupovat a tak při vývoji aplikací je nutno myslet i na uživatele ne se zrovna nejnovější technologií.

Jak je jistě známo na platformu operačního systému Android je již spousty aplikací. Při průzkumu trhu aplikací s tématikou komiks kamery nebylo nalezeno příliš mnoho řešení, které by přesně splňovali tuto problematiku. Tak jsem se snažil navrhnout a implementovat co nejlepší řešení, které klade hlavně důraz na jednoduchost ovládání aplikace a funkčnost.

Při průzkumu trhu jsem narazil na 3 aplikace, které odpovídají podobným parametrům zadaných touto prací, ovšem při testování těchto aplikací jsem narazil buďto na velice pomalé vykreslování kamery v reálném čase, nesmyslné ovládání aplikace, při kterém má nezkušený uživatel problémy při ovládání této aplikace a také jsou pomalejší na slabších zařízeních.

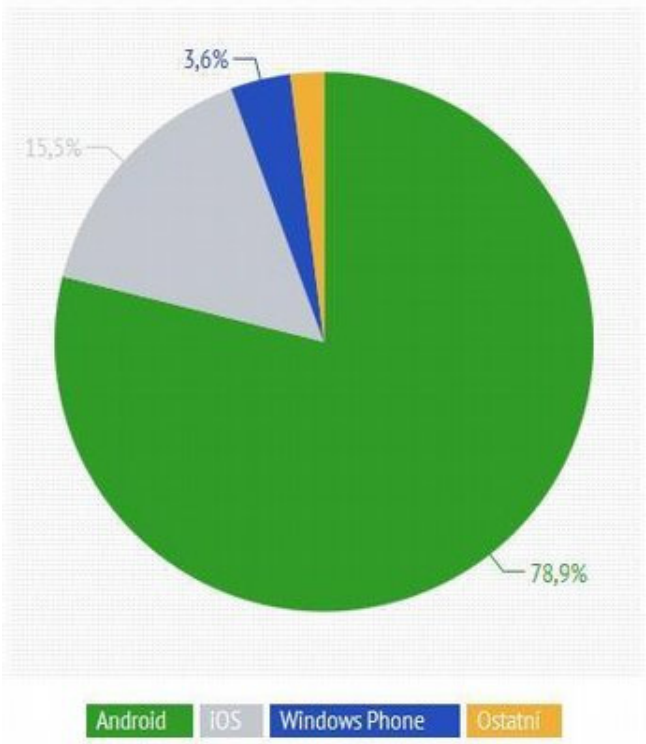
Cílem této práce bylo vytvořit aplikaci komiks kamera v reálném čase, která bude umožňovat vytvářet fotky v reálném čase za pomocí efektu jak ze světa komiksu.

Kapitola 2 se stručně věnuje popisu současné situace Androidu a také popisuje jednotlivé části při tvorbě aplikace pro tento operační systém. Kapitola 3 se věnuje popisu knihovny OpenGL ES, která je využita v této práci. Vysvětluje stavební kameny této aplikace, a jakým způsobem pracuje se zobrazovanými daty. Kapitola 4 se věnuje popisu zpracování obrazu. V této kapitole jsou popsány nejdůležitější metody, které jsou zapotřebí při tvorbě komiksové kamery a filtry, které tyto metody využívají právě tak, aby výsledný obraz odpovídal zadaným požadavkům. Kapitola 5 se věnuje návrhu aplikace, ve které jsou blíže popsány detaily aplikace a analýze již existujících aplikací. Kapitola 6 se věnuje podrobné implementaci aplikace a jsou zde popsána zajímavé řešení implementace aplikace a následně se podrobně věnuje testování aplikace a jejím dopadům na konečnou implementaci.

Kapitola 2

Platforma Android

Android je nejpopulárnější operační systém mezi mobilními zařízeními. Tento operační systém je založen na jádru Linuxu. Android je vyvíjen jako open source a o jeho vývoj se stará společnost Google. Nyní(květen 2015) je nejnovější verze 5.1 Lollipop. Operační systém Android má téměř 80% podíl na trhu s chytrými telefony, který je graficky zobrazen na obrázku 2.1.



Obrázek 2.1: Pozice Androidu na trhu[1]

Architektura operačního systému Android je rozdělena do několika vrstvách[15]. První a nejnižší vrstva tvoří jádro operačního systému. Toto jádro Androidu je postaveno na jádře z operačního systému Linux. Využívá se zde mnoho vlastností z Linuxu jako např. správa paměti, správa sítí, zabudované ovladače atd. Další vrstvou jsou knihovny, které

[1] Viz. http://e-svet.e15.cz/technika/zeleny-robot-dominuje-android-ma-na-trhu-chytrych-telefonu-podil-79-procent-1058541

jsou napsány v programovacím jazyce C nebo C++. Tyto knihovny jsou zavedeny z důvodu používání různých komponent právě v těchto programovacích jazycích. Tyto funkce jsou poskytnuty pomocí prostředí Android Application Framework. Další vrstva obsahuje aplikační virtuální stroj. V této vrstvě jsou obsaženy základní knihovny programovacího jazyka Java. Poslední a nejvyšší vrstvu tvoří základní aplikace, které využívají uživatelé operačního systému Android.

2.1 Struktura aplikací

Aplikace pro platformu Android používají až 6 základních komponent a jsou to [3]:

- Aktivity (Activity) - jsou to základní stavební bloky uživatelského rozhraní. Aktivitu je možno si představit jako jednotlivé okno aplikace, obsahující grafické rozhraní pro interakci s uživatelem. Při přechodu mezi aktivitami jsou staré aktivity ukládány na zásobník, kde je na vrcholu uložena právě běžící aktivita. Při stisku tlačítka zpět se ze zásobníku odstraní nynější aktivita a je spuštěna ta, která byla na dalším místě v zásobníku.
- Fragmenty (Fragments) - jsou volitelná vrstva, pomáhají měnit konfiguraci aktivit za účelem podpory velkých i malých obrazovek na mobilních zařízeních.
- Pohled (View) - je základním stavebním kamenem uživatelského rozhraní. Všechny pohledy v jednom okně jsou uspořádány do stromu. Slouží pro zobrazování textových polí, obrázků, tlačítek atd.
- Služby (Service) - představují proces běžící na pozadí, navrženy k neustálému provozu, které nemají grafické rozhraní. Většinou se používají k vykonání dlouho trvajících úkolů, ke kontrole dostupných aktualizací, přehrávání hudby na pozadí atd.
- Poskytovatelé obsahu (Content provider) - poskytují úroveň abstrakce pro jakékoliv data uložená v zařízení, ke kterým lze přistupovat z více aplikací. Android podporuje uživatele v tom, aby svá data kromě svých aplikací zpřístupnili i ostatním aplikacím. Slouží pro sdílení dat mezi aplikacemi.
- Přijímače přenosů (Broadcast provider) - jsou systémové zprávy, které kolují v zařízení a upozorňují aplikace, na různé události například reakce na oznámení o nízkém stavu baterie, doručení SMS zprávy, vložení karty SD.

2.2 Vývojové prostředky

V současné době se používají pro vývoj aplikací pro Android dvě vývojové prostředí. První je prostředí Eclipse s přidaným pluginem Android Development Tools(ADT) a debugování pomocí Android SDK.

Druhé vývojové prostředí je Android studio, které v nedávné době (prosinec 2014) přešlo z beta verze do public verze. Nyní (květen 2015) se Android studio nachází ve verzi 1.2. Toto vývojové prostředí je nové vývojové prostředí pro tvorbu Android aplikací. Android studio je oficiální vývojové prostředí firmy Google a tudíž má plnou podporu od této společnosti. Má za úkol nahradit dosud používané vývojové prostředí Eclipse a do budoucna se předpokládá, že se primárně bude používat právě Android studio. Android studio je postavené nad komunitní verzí IntelliJ IDEA

Kapitola 3

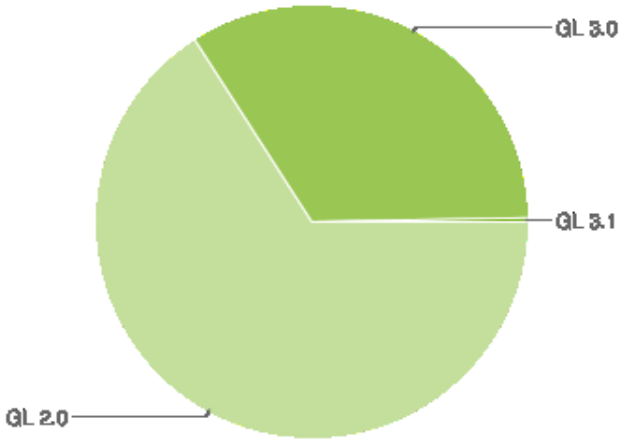
OpenGL ES

OpenGL ES poskytuje podmnožinu funkcionality OpenGL určené primárně pro mobilní a vestavěné systémy. OpenGL ES podporuje pouze pevnou řadovou čárku. Na obrázku 3.1 můžete vidět zastoupení jednotlivých verzí OpenGL ES na trhu. V následujících podkapitolách je popsána jednotlivá práce s touto knihovnou.

3.1 Verze

Existuje několik verzí OpenGL ES a to[3]:

- OpenGL ES 1.0 a 1.1 - podporováno od Android 1.0 a vyšší
- OpenGL ES 2.0 - podporováno od Android 2.2 (API level 8) a vyšší
- OpenGL ES 3.0 - podporováno od Android 4.3 (API level 18) a vyšší
- OpenGL ES 3.1 - podporováno od Android 5.0 (API level 21) a vyšší



Obrázek 3.1: Verze OpenGL ES na trhu[1](duben 2015)

Z obrázku 3.1 vyplývá, že nejvhodněji zvolená verze OpenGL ES i díky komerčnímu použití bude verze 2.0, která v tuto dobu (duben 2015) zastupuje téměř 70% zařízení na trhu.

[1]https://developer.android.com/about/dashboards/index.html

3.2 Zobrazování

Jak plyne z obrázku tak pro tuto práci byla vybrána knihovna OpenGL ES 2.0. Android podporuje OpenGL přes framework API pomocí jazyka Java a nástroj Native Development Kit (NDK), která umožňuje napsat část aplikace v nativním kódu (jazyk C).

V Android frameworku se nacházejí dva základní prvky, které dovolují manipulaci s grafikou a to třída `GLSurfaceView` a rozhraní `GLSurfaceView.Renderer`.

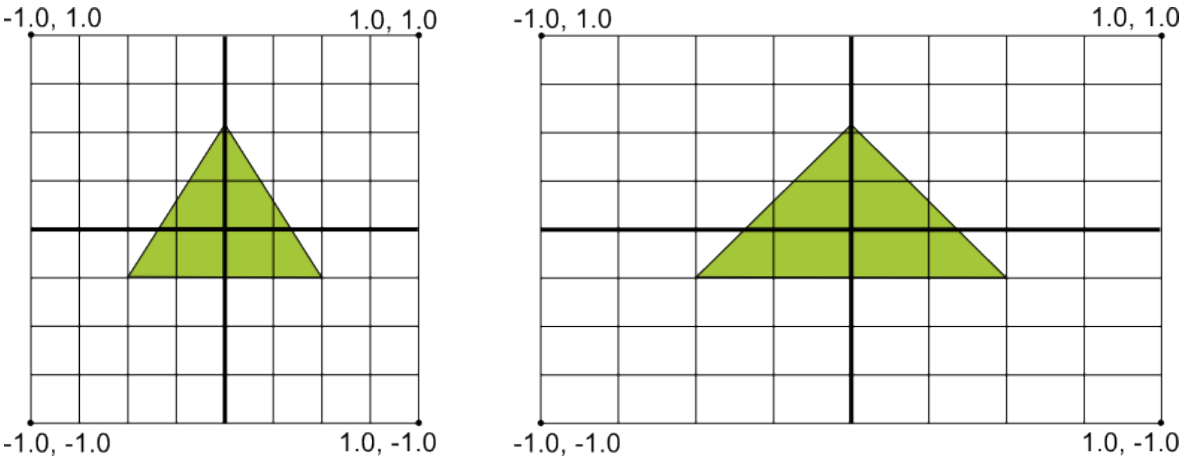
`GLSurfaceView` je třídy `View` a umožňuje kreslit a manipulovat s objekty použití voláním OpenGL API. `GLSurfaceView.Renderer` rozhraní definuje metody potřebné pro kreslení grafiky v `GLSurfaceView`. Toto rozhraní vyžaduje implementaci následujících metod[4]:

- `onSurfaceCreated()` - Systém tuto metodu volá pouze jednou a to když vytváří `GLSurfaceView`. Nastaví parametry OpenGL prostředí, nebo inicializuje OpenGL grafické objekty.
- `onDrawFrame()` - Tato metoda je volána systémem pokaždé při překreslení u třídy `GLSurfaceView`. Metoda je použita primárně při provedení kreslení (překreslení) bodu grafického objektu.
- `onSurfaceChanged()` - Systém tuto metodu volá, když GLSurfaceView mění geometrii, vkládá změny velikosti, nebo mění orientaci obrazovky zařízení.

OpenGL není implicitně deklarována při použití, musí být nadeklarována v souboru AndroidManifest.xml.

Mapování

Jedním ze základních vlastností zobrazení na Android zařízené je, že při různých velikostech obrazovky můžou kolísat velikosti a tvary obrazců. OpenGL předpokládá jednotný čtvercový souřadnicový systém. V základním nastavení považuje displej za čtvercový (jak je vidět na 3.2)



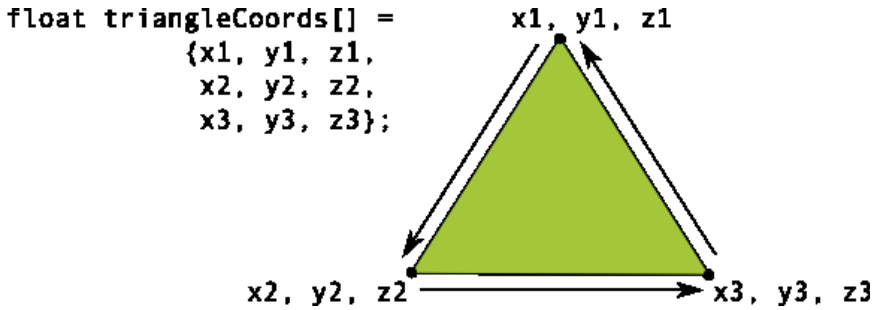
Obrázek 3.2: Vykreslení stejného obrazce při jiných velikostech obrazovky[4].

Vlevo na obrázku 3.2 je možno vidět základní souřadnicový systém v knihovně OpenGL a vpravo je možno vidět typickou orientaci zobrazení na Android zařízeních. Aby došlo ke správnému zobrazení, musí se vytvořit projekční matice a matice pohledu kamery. Projekční

matice přepočítá souřadnice, aby byli korektní se zobrazením na Android zařízení. Matice pohledu kamery převede objekty do pohledu současné pozice kamery [3].

Tvary obrazců

V OpenGL je povrch definován třemi nebo více body ve tří rozměrném prostoru. Sada má přední a zadní stranu. Tyto strany jsou poznat podle toho jak jsou definované. Metoda, která se k tomuto vykreslování používá se nazývá face culling[2]. Tato metoda bude popsána jen velice jednoduše. Je to technika, kdy se podle definované metody vykreslí jen polygon otočený směrem k/od kamery.



Obrázek 3.3: Příklad vykreslení v proti směru hodinových ručiček[3].

Na obrázku 3.3 jsou definovány body trojúhelníku v pořadí, proti směru hodinových ručiček. Ve výchozím nastavení OpenGL je plocha, která je kreslena proti směru hodinových ručiček přední plochou. Zadní plocha je uspořádána ve směru hodinových ručiček.

Pipeline

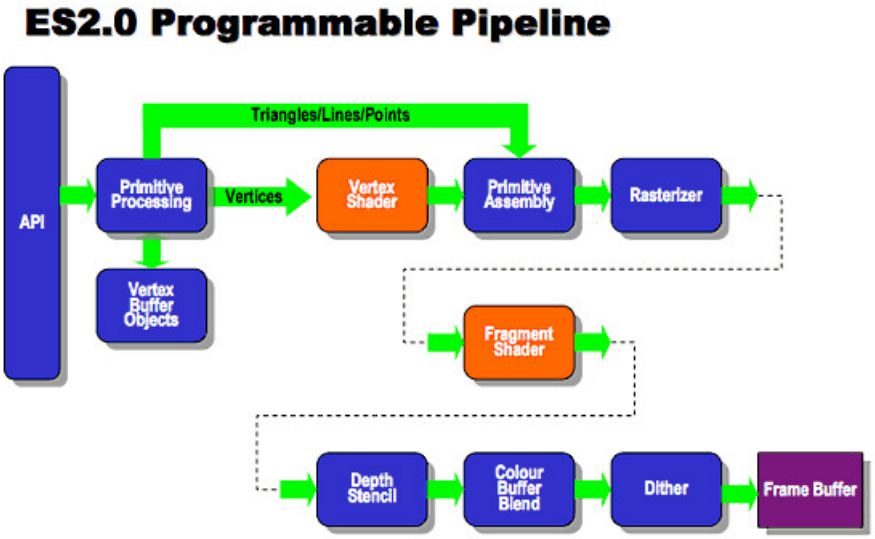
V OpenGL ES je každý požadavek na rendering zpracován pomocí vnitřní `pipeline`[11]. Na počátku jsou zpracovány primitiva body, čáry, trojúhelníky. Každý vrchol primitiva následně prochází programovatelným vertex shaderem. Následuje rasterizace primitiv a vzniklé fragmenty jsou předány do fragment shaderu. Výstup z toho fragmentu se ořezává na základě hodnot depth a stencil bufferu. Dále dochází k míchání s již existujícími pixely frame bufferu. Poté dochází k ditheringu a naposled se provádí zápis do frame bufferu. Celý tento průchod pipeline můžete vidět na obrázku 3.4.

Shadery

Nedílnou součástí OpenGL ES jsou shadery. Shader je program, který slouží k řízení jednotlivých částí programovatelného grafického řetězce grafické karty (GPU). Shadery jsou psané v jazyce GLSL (Open GL Shading Language)[9]. Shadery se dají zpracovat paralerně a jsou velmi rychlé. Zde jsou typy shaderu podporované v knihovně OpenGL ES 2.0:

- Vertex shader - zpracovává vrcholy a může je například násobit pohledovými maticemi.
- Fragment shader - je prováděn na každém pixelu grafické scény. Určuje výslednou barvu pixelů.

[2]https://www.opengl.org/wiki/Face_Culling
[3]https://www.khronos.org/opengles/2_X



Obrázek 3.4: OpenGL ES 2.0 Pipeline[3]

Implementace shaderu v Android prostředí probíhá pomocí textových řetězců. V těchto textových řetězcích je zapsán celý kód shaderu. Do shaderů jsou předávány hodnoty pomocí funkcí implementované v knihovně OpenGL ES. Například funkce `glUniform1f` předá uniform proměnné nějakou hodnotu. Proměnná *attribute* se předává pouze vertex shaderu právě jako atribut. Dále se zde nacházejí další proměnné, které specifikují výstupní hodnoty. Výstupní barvu specifikuje hodnota v proměnné *gl_FragColor*. Proměnná *gl_Position* obsahuje pozici současného vrcholu[14]. V OpenGL se nacházejí i další proměnné, ovšem ty v této práci nebudou zapotřebí.

V shaderech probíhá práce s různými druhy proměnnými. Proměnná *uniform* je globální proměnná. Proměnná *varying* poskytuje rozhraní mezi vertex a fragment shaderem.

Než se začne se shadery pracovat, je zapotřebí provést jejich kompilaci a linking. Shadery pracují s textovými řetězci. Nejprve se musí shader vytvořit, to se provede voláním funkce `glCreateShader`, kde je jako parametr uveden typ shaderu(`GL_VERTEX_SHADER` nebo `GL_FRAGMENT_SHADER`). V následujícím kroku je předán funkci `glShaderSource`. Jako parametr je předán textový řetězec, ve kterém je načtený kód shaderu. Následně se shader zkompiluje pomocí funkce `glCompileShader`.

V dalším kroku následuje spojení vertex a fragment shaderu dohromady. To se provede pomocí funkce `glAttachShader`. Po spojení shaderů je proveden linking pomocí funkce `glLinkProgram`. Nakonec jsou shadery aktivovány pomocí funkce `glUseProgram`.

Kapitola 4

Zpracování obrazu

Existují dva typy přístupu k počítačové grafice a to vektorová nebo rastrová. Vektorová ukládá přesná geometrická data např. souřadnice bodů. Rastrová, jejíž základem je pravidelná síť pixelů.

Tato práce se zabývá hlavně komiksovým efektem. Tento efekt napodobuje kreslený film. Výsledkem tedy je segmentovaný obraz s redukovaným počtem barev. Jak je již řečeno obraz by měl vypadat jako komiks a toho se dosáhne ohraničením přechodů barev černou barvou a redukování barev na obrazu. Ovšem je zapotřebí brát ohled na výpočetní možnosti mobilních zařízení, některé algoritmy na úpravu obrazu jsou velice náročné. I když se tyto algoritmy budou zdát nejlepší pro úpravu obrazu, na mobilních zařízeních nemusí být použití těchto algoritmů ideálním řešením.

Efekty s obrazem jsou také prováděny pomocí konvolučních filtrů nebo geometrických transformací.

4.1 Detekování hran

Hrana se na obrazu nachází, kde se prudce mění jas. Neboli tam, kde se mění barva. Každý přechod má svojí šířku pásma. Čím je šířka pásma užší, tím je hrana ostřejší. V podkapitolách jsou uvedeny algoritmy pro detekci hran[8].

Robertsův algoritmus

Je to nejjednodušší operátor. Používá okolí pouze 2x2 bodu. Nevýhoda je, že detekuje i šum jako hrany, používá derivaci pouze v jednom bodě. Tento algoritmus lze použít pouze na obrázky bez šumu, dnes se již už pomalu nepoužívá. Operátor je tvořen dvěma konvolučními maskami.

$$h_1 = \begin{bmatrix} +1 & 0 \\ 0 & -1 \end{bmatrix}, h_2 = \begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix} \tag{4.1}$$

Kde h_1 a h_2 jsou různé míry gradientu ve dvou orientacích. Výsledný gradient se vypočítá:

$$|M| = \sqrt{h_1^2 + h_2^2} \tag{4.2}$$

Sobelův algoritmus

Oproti Robertsovu algoritmu počítá derivaci z okolí 3x3 bodů, má dvě masky, hranu detekují vždy tou maskou, která probíhá souběžně s nulami v masce[1].

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * A \quad a \quad G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * A \tag{4.3}$$

kde A je zdrojový obraz, G_y a G_x jsou konvoluční masky. Výsledný gradient Sobelova algoritmu je vypočten:

$$G = \sqrt{G_x^2 + G_y^2} \tag{4.4}$$

Kde G je výsledná hodnota gradientu.

4.2 Odstranění šumu

Šum je svázán s původní informací. Vzhledem k tomu, že nemá informaci o původní funkci, tak nelze rozeznat co je šum a co původní signál. Filtry vyhodnocují okolí pixelu a z toho se zjistí co je šum a co ne. Šum je vlastně velice podobný hranám, akorát s tím rozdílem, že šum se nalézá pouze v malém rozsahu na jednotlivých místech. Je možno si ho představit například jako černé tečky v modrém prostředí. Filtry odstraňující šum se dělí na konvoluční a lokální statistiky okolí. Konvoluční odstraní šum spočítáním průměru z okolních pixelů[13].

Šum je v této práci zmiňován kvůli kvalitě výsledného obrazu komiksového efektu. Šum je z hlediska komiksového efektu nevítaným jevem. Pro dosažení co nejlepšího výsledného obrazu je nutno právě tento šum odstranit. Výsledný obraz bude potom mít hladké a jednotné plochy, kterými se právě komiksový efekt vyznačuje.

Gausův filtr

Řadí se mezi lineární filtry. Je efektivní k potlačení Gaussova šumu. Počítá hodnotu pomocí konvoluce maskou, která se skládá ze součástí určených Gaussovou funkcí.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{4.5}$$

Touto metodou se rozmaže obrázek. Z toho můžou vést další problémy například při detekci hran.

Mediánový filtr

Řadí se mezi nelineární filtry. Pracuje na principu, že všechny hodnoty seřadí podle hodnoty, z těchto hodnot vybere hodnotu, která se nachází přesně uprostřed vzestupně seřazených hodnot. V případě sudého počtu prvků, se vezmou obě dvě prostřední hodnoty a ty se poté zprůměrují. Tato hodnota se potom stává novou hodnotou pixelu. Tento filtr je vhodný k odstranění náhodného šumu.

[1] Viz. http://en.wikipedia.org/wiki/Sobel_operator

4.3 Prahování

Prahování se nám bude hodit, při tvorbě komiksového černobílého obrazu. Na obrázku 4.1 je uvedeno, jak při prahování bude vypadat zpracovaný obrázek. Prahování funguje

Obrázek 4.1: Příklad převodu prahováním[2].

způsobem, že podle daného předpisu upravuje vstupní hodnoty. Je stanoven daný práh, který se určí například z histogramu hodnot. Když hodnota bude menší než práh, je dána místo ní bílá barva, a když větší tak je na dané místo dána černá barva na daný pixel. Matematickým popisem je prahování vyjádřeno[2]:

$$f(c)\begin{cases} A & \text{pro } c < p \\ B & \text{pro } c \geq p \end{cases} \tag{4.6}$$

Kde:
c je vstupní hodnota barvy
$f(c)$ je výsledná hodnota
p je prahovací hodnota
A a B jsou jsou nové hodnoty pro vstupní hodnotu c pod a nad prahem.

4.4 Transformace barev

Počet barevných odstínů, které se mohou vyskytovat v původním obrázku, dosahuje hodnot přes šestnáct miliónů barev. V případě aplikace bude nejlepší počet barevných odstínů v obraze snížit. Samozřejmě při snížení počtu barevných odstínů v obrazu dojde ke ztrátě informace. V počítačové grafice existují metody, které z málo barev dokáží vytvořit iluzi bohaté barevné škály a to polotónování a rozptylování. Při polotónování výsledný obraz zvětší svoje rozlišení, protože každý pixel obrazu je nahrazen maticí bodů, ve které je daleko

[2] http://cs.wikipedia.org/wiki/Prahov%C3%A1n%C3%AD

méně barev než v původním obrazu. Při rozptylování zůstane obraz v původní velikosti. Obě tyto metody v počítačové grafice využívají schopnost lidského oka vytvářet z několika bodů u sebe vjem jediného bodu.

Transformace barev je relevantní z hlediska výběru palety barev pro komiksový filtr. Níže uvedené metody nebyly v plánu implementace této aplikace, ale jsou plánovány do budoucího vývoje této aplikace. Pro příklad jsou zde uvedeny základní metody rozptýlení (uvedeny na převodu odstínů šedi do černé a bílé):

Náhodné rozptýlení

O tom zda výsledný pixel bude černá nebo bílá, využijeme původní velikost intenzity a generátor náhodných čísel, které se budou mezi sebou porovnávat a barva výsledného pixelu vznikne právě z těchto dvou porovnání.

Pravidelné (maticové) rozptýlení

Používá předem dané vzorky matic na nahrazení daného pixelu dle jeho hodnoty. Tím pádem dojde ke zvětšení obrazu. Výsledný obraz působí dojmem, jak kdyby byly tmavší či světlejší místa vyšrafovány.

Distribuce zaokrouhlování chyby

Zpracovává pixely po řádcích od shora dolů. Prahováním je určena výstupní intenzita ze vstupních pixelů které se nacházejí na vstupu. Zanedbaná hodnotu při zaokrouhlení je uložena a tato hodnota je použita k modifikaci dalších pixelů. Chyba se distribuuje pouze mezi sousedními pixely, které ještě nebyly zpracovány.

Pro příklad by výsledný obrázek měl být podobný obrázku na obrázku 4.2.

Obrázek 4.2: Ukázka komiksového obrázku[3]

Barevná paleta 3-3-2

Tato metoda je uvedena pro příklad, jak se v grafice omezuje barevné spektrum efektivním způsobem. Při práci v barevném prostoru je práce s pixely analogická, ovšem jsou tu

[3] http://i.ytimg.com/vi/WZfGuiYiu4o/hqdefault.jpg

dvě rozšíření. První rozšíření se týká zpracování vstupního barevného pixelu, který nese barevnou informaci v určitém barevném modelu, většinou RGB, tak jsou rozptylovací algoritmy prováděny po třech různých složkách. Druhé rozšíření se týká výstupní barevné škály. Místo původní černé a bílé se zde používá daleko více barev. Zaokrouhlování do této škály znamená pouze menší úpravu metod.

Tato metoda rozdělí barevné prostředí do 8 bitů tj. 256 barev, jelikož tato paleta zahrnuje barevný model RGB, tak nejde rozdělit na 3 stejné díly a proto je modrá barva aplikována pouze dvěma bity, jelikož lidské oko na modrou není tak citlivé. Červená a zelená jsou aplikovány shodně po třech bitech. Zařazení původních pixelů do této palety se provádí podle vzorců pomocí bitových součinů, logické operace and a celočíselného dělení.

Kapitola 5

Návrh aplikace

V této kapitole budou přiblíženy jednotlivé části návrhu aplikace. Nejprve jsou rozebrána již existující řešení, klady, zápory a výsledné hodnocení těchto aplikací. Pomocí vyhodnocení těchto aplikací jsou dále rozebrány návrhy grafického rozhraní a zpracování obrazu. Poslední část kapitoly se věnuje uložení fotografie do paměti.

5.1 Analýza již existujících řešení

V této kapitole byla provedena analýza již existujících aplikací, zhodnoceny jejich plusy a mínusy a na základě těchto vlastností byla inspirována tato aplikace. Při návrhu aplikace jsem zhodnotil všechny plusy a nechal se inspirovat kladnými vlastnostmi aplikací. Snažil jsem se vyvarovat nepřehledných částí aplikace, které se v těchto aplikacích vyskytly.

Na Google play jsem vybral 3 aplikace, které se mi zdáli dobré i špatné a ty zde mám pro ukázku uvedeny pro příklad, jak aplikace nemá vypadat, či jak by mohli vypadat všechny vlastnosti těchto aplikací jsou rozvedeny níže.

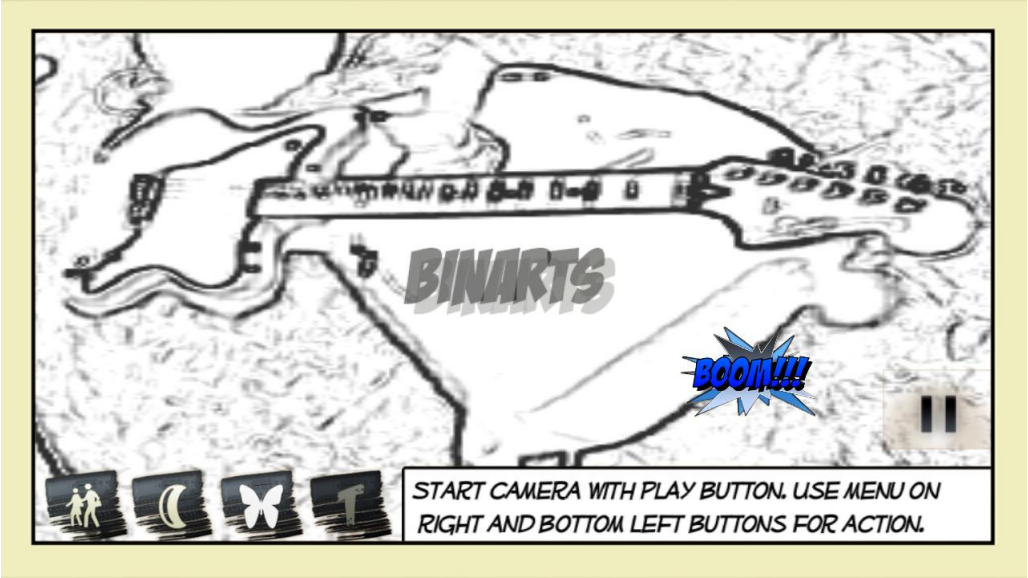
Comics Camera[1]

Tato aplikace má na Google play přes deset tisíc stažení a hodnocení 2,5 z 5. Musím podotknou, že z této aplikace jsem si vzal příklad, jak to nemá vypadat. Jak můžete vidět na obrázku 5.1. Při prvním spuštění aplikace jsem si moc nevěděl s aplikací rady, což může dost uživatelů odradit od používání aplikace. V této aplikaci jsou použité efekty a to: polotónování barev barevného schématu CMYK, zvýraznění hran, jak je vidět na obrázku 9 a efekt pomocí prahování. Dosti nesmyslné mi na této kameře přijde focení pomocí tlačítka „BOOM!!!“. Nevýhodou také je, že aplikace při normálním režimu převádí barvy na paletu 256, což je dost nepřirozené, ale jestli autor měl na mysli tímto způsobem udělat dojem komiks kamery, tak to se mu moc nepovedlo.

Cartoon Camera[2]

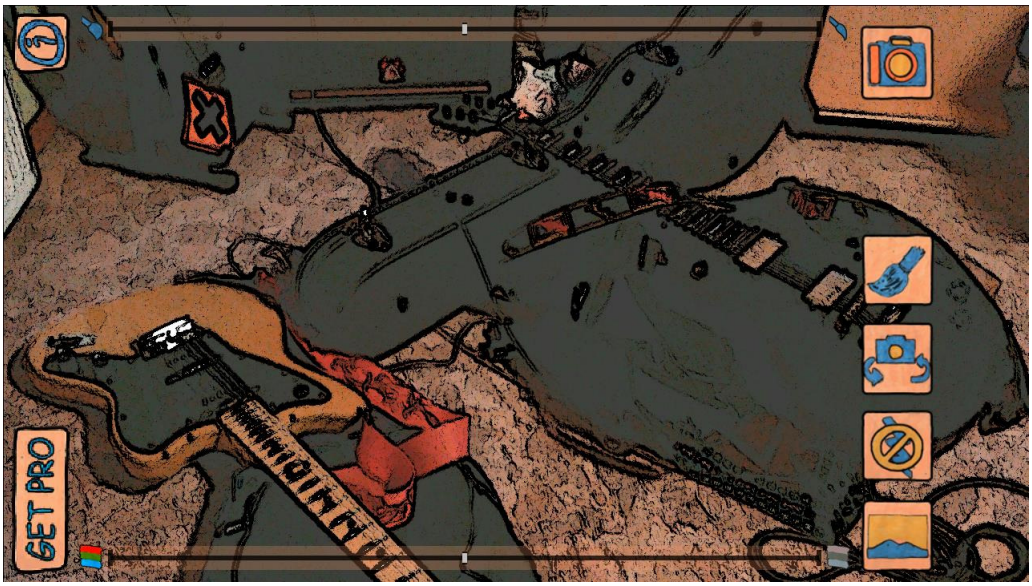
Tato aplikace už na Google play vypadá znatelně lépe. Stažení této aplikace činí již přes 10 miliónu a hodnocení dosahuje 4,2 z 5, což je velice pěkné. Při spuštění aplikace se ihned spustí obrazovka, která je zobrazena na obrázku 5.2. Na této obrazovce je možno nastavit šířku čar, barevnost, změnit efekty, je jich zde implementováno poměrně hodně, ovšem

[1] https://play.google.com/store/apps/details?id=binarts.apps.comics
[2] https://play.google.com/store/apps/details?id=com.fingersoft.cartooncamera



Obrázek 5.1: Obrazovka aplikace Comics Camera

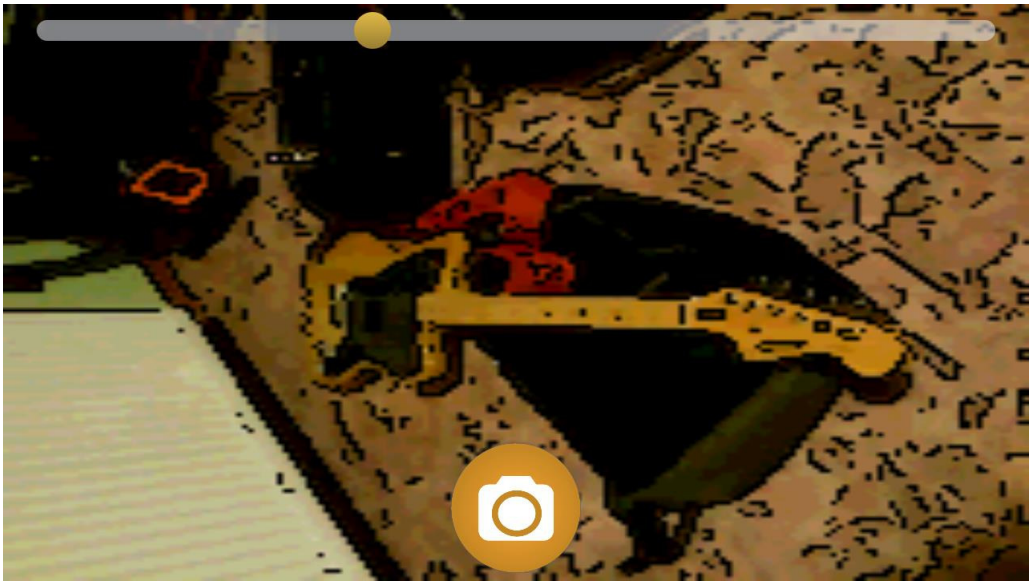
nějaké jsou přístupné až po zakoupení pro verze, která stojí 30,25 Kč. Dále se zde nachází tlačítko pro focení, přístup do galerie, změna kamery a taktéž podporuje blesk, který si je možno zapnout či vypnout. V této aplikaci je možno si vybrat z celkem 12 efektů (cartoon, sépie, barevná kresba a další). Na pravé straně vidíte, že tlačítka nejsou u kraje aplikace, to je způsobeno tím, že by zde měla být reklama, ale ta se zobrazuje pouze na výšku. Tato aplikace mi přišla nejlepší ze všech, které jsem zkoušel.



Obrázek 5.2: Obrazovka aplikace Cartoon Camera.

Cartoon Camera HD[3]

Tato aplikace má na Google play 100 tisíc stažení a hodnocení 3,8 z 5. Při spuštění aplikace se nám zobrazí úvodní menu, ve kterém je možno si nastavit kvalitu efektu a rozlišení kamery (v mém případě to nešlo nastavit a zůstalo tam nejmenší rozlišení, které je dostupné v této aplikaci). Při přejití do hlavní části aplikace se zde nachází posuvník pro změnu citlivosti barev a tlačítko na vyfocení. Jak je vidět na obrázku 5.3. Tato aplikace má pouze jeden efekt a také nemá možnost nahlížet na již pořízené obrázky. V této aplikaci se mi líbí jednoduché ovládání, ale také pouze jeden efekt, což je oproti jiném aplikacím omezující.



Obrázek 5.3: Obrazovka aplikace Cartoon Camera HD

5.2 Grafické rozhraní aplikace

Při návrhu uživatelského rozhraní bylo bráno v potaz ihned upoutat uživatelovu pozornost. Po projití již existujících řešení jsem si vzal z těchto řešení to nejlepší a snažil se dát vše dohromady. Aplikace tedy byla navržena způsobem, že uživatel po spuštění aplikace přejde do hlavní části aplikace na hlavní obrazovku, jak můžete vidět na obrázku 5.4.

Zde je řešeno i jednoduché ovládání aplikace, kde se v pravé části horní obrazovky aplikace nachází dvě tlačítka. První tlačítko „Photo“ slouží pro pořízení fotky a druhé tlačítko „Change“ slouží pro změnu mezi filtry prahování a komiksovým filtrem. V levé horní části obrazovky můžete vidět dva posuvníky(SeekBar)[1], kde první zleva slouží k nastavení intenzity barev komiksového filtru a druhý posuvník slouží k nastavení prahu při vytváření hran.

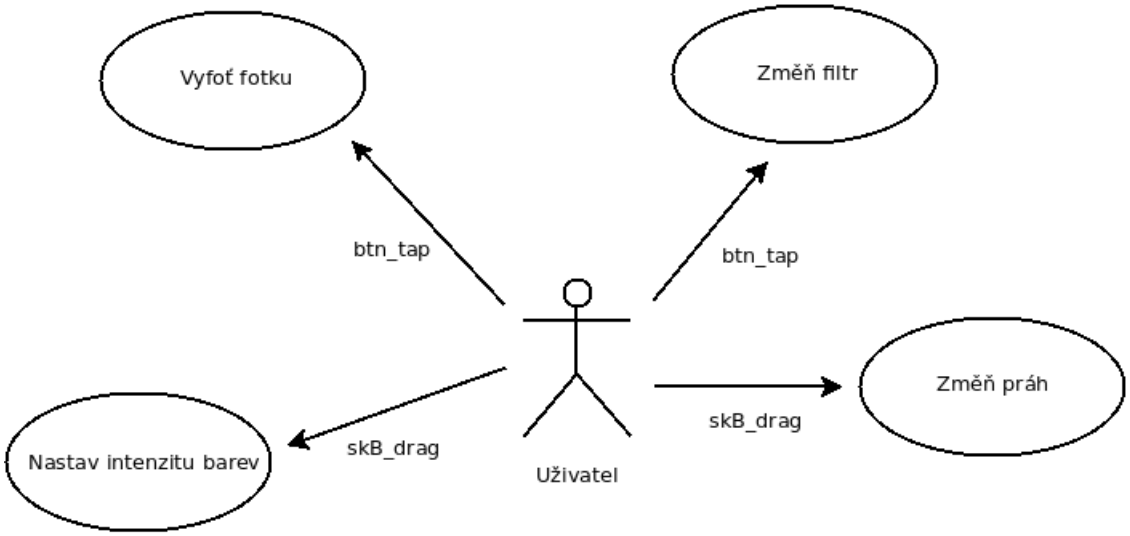
Zbytek okna aplikace bude tvořit náhled kamery v reálném čase. Na tento náhled bude přímo aplikovaný filtr, který bude zvolen. Tím pádem uživatel bude okamžitě vidět, jaké má nastaveny jednotlivé hodnoty a vydedukuje si výslednou fotku, která bude pořízena.

[3]https://play.google.com/store/apps/details?id=com.usefullapps.cartooncamera

Obrázek 5.4: Původní návrh aplikace

Popis případů použití

V této podsekci jsou znázorněny pomocí use-diagramu 5.5 základní vlastnosti hlavní obrazovky aplikace. Jak je vidět i z obrázku ovládání aplikace je velice jednoduché. Tento návrh by měl uživateli se okamžitě zorientovat v aplikaci a umět jí ihned používat.



Obrázek 5.5: Use case diagram možností hlavní obrazovky

Vysvětlení zkratek případů použití:

- `btn_tap` - stisk tlačítka

- `skB_drag` - změna hodnoty na posuvníku

5.3 Zpracování obrazu

V této kapitole je podrobněji popsán návrh zpracování obrazu. Obraz bude zpracován pomocí knihovny OpenGL ES[12]. Pomocí této knihovny se bude zpracovávat obraz po jednotlivých pixelech, které budou přicházet z kamery. Tento způsob je výrazně rychlejší oproti již existujícím aplikacím, které řeší jednotlivé výpočty na efekty pomocí vnitřního CPU. Moje aplikace bude tyto efekty vytvářet pomocí knihovny OpenGL ES.

Pro aplikaci byly navrženy dva filtry. První filtr bude pouze černobílý obrázek, který nejprve spočítá hrany a poté prahuje výsledek. Druhý filtr už bude barevný a to bude právě náš výsledný filtr. Komiksový efekt bude pracovat na podobném principu jako černobílý efekt. Hrany budou zobrazeny taktéž černou barvou. Změna bude pouze v tom, že se bude zobrazovat barva, kde bude hodnota menší než zvolená hodnota hrany.

5.4 Uložení fotografie

Samozřejmě nedílnou součástí této aplikace je uložení fotografie. Zde se nachází více způsobů, jak fotografii uložit. Při návrhu aplikace jsem bral v potaz dvě řešení způsobu uložení fotografie do paměti mobilního zařízení. První možnost bude fotografii uložit přímo z kamery bez použitých filtrů a poté na tuto pořízenou fotografii znovu aplikovat filtr vybraný uživatelem aplikace. Druhá možnost je, že z pixelů, na které je už aplikován filtr je vytvořena bitmapa a ta se poté uloží do paměti.

Prvním způsobem je získána velikost fotografie ve stejném rozlišení, jaké je rozlišení fotoaparátu. Ve druhém případě je získána velikost pouze tak velkou, jaká je velikost rozlišení displeje.

Další otázkou uložení fotografie do paměti je, kam ji vlastně uložíme. Zde se také nacházejí dvě možnosti. První možnost je uložit fotografii do interní paměti. Některá mobilní zařízení nemají příliš velkou kapacitu a je možnost, že se kapacita brzy zaplní a uživatelé nebudou mít fotografie kam ukládat. Další možností je ukládat fotografie do externí paměti. Toto je o poznání lepší řešení, ale zde se nachází také jeden problém. Ne každý uživatel musí mít ve svém mobilním zařízení paměťovou kartu. Právě proto spoléhám na Android API na získání adresy určené pro ukládání obrázků.

Kapitola 6

Implementace a testování

V této kapitole budou popsány všechny důležité části aplikace, které byly implementovány v této práci. Implementaci aplikace jsem prováděl v programovacím jazyce Java, ve vývojovém prostředí Android Studio, které je zmíněno v kapitole 2.2 a frameworku Android SDK.

Nejdříve je popsáno nastavení aplikace. Poté následuje popis grafického rozhraní, zpracování obrazu a uložení výsledného obrazu do paměti. Nakonec je ukázka výsledné aplikace.

6.1 Nastavení Androidu

V aplikaci je používána knihovna OpenGL ES verze 2.0, u které je požadována minimální verze Android 2.2, a proto je tato verze nastavena jako minimální pro tuto aplikaci. Při spuštění aplikace se okamžitě provádí kontrola, zda dané zařízení podporuje námi využívanou knihovnu OpenGL ES 2.0. Pro správný chod aplikace musí být na zařízení k dispozici kamera. Cílovým systémem je Android 4.4 Kit Kat tedy verze SDK 19[7].

Kód 6.1: Ukázka nastavení SDK verze na zařízení Android

```
<uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="19" />
```

Důležitou roli v nastavení aplikace hraje soubor `AndroidManifest.xml`, kde je specifikováno na jaké verze operačního systému Android lze aplikaci nainstalovat, požadovaná práva pro správný chod aplikace, jaké periferie jsou potřeba, odkaz na počáteční bod aplikace a jméno aplikace.

6.2 Grafické rozhraní

V této části je podrobnější implementace grafického rozhraní aplikace. Oproti původnímu návrhu řešení 5.4 jsem musel udělat pár změn. Změny nastávají v ovládacích prvcích. První změna nastává při změně filtru. Při prahovém filtru je schován posuvník na změnu intenzity barvy, který se zde neuplatní. 6.1 Důvodem schování posuvníku bylo odstranění nechtěné interakce uživatele s nepotřebnými ovládacími prvky. Další změnou oproti původnímu návrhu je nastavení pozadí ovládacích prvků. Při průsvitném pozadí nebyly ovládací prvky příliš vidět, a proto bylo zvoleno barevné pozadí ovládacích prvků.

Při implementaci grafického rozhraní byly dvě možnosti, jak tyto prvky implementovat.

- Rozdělení na GUI (XML) a kód (Java)
- pouze kód (Java)

Z mé strany byla zvolena první varianta. Tato varianta se jeví jako profesionálnější způsob implementace, jednodušší změny v grafickém rozhraní a přehlednější kód. Pro grafické prvky jsou důležité dva soubory typu XML. První z těchto souborů je `activity_main.xml`, ve kterém je nastaveno hlavní zobrazení aplikace. Ve druhém souboru `gui.xml` jsou definovány a nastaveny jednotlivé grafické prvky aplikace.

Zde můžete vidět jak probíhá volání grafického rozhraní v aplikaci:

Kód 6.2: Vytvoření grafického rozhraní

```
GL2View mGLView = new GL2View(this);
setContentView(R.layout.activity_main);
ViewGroup parent = (ViewGroup) findViewById(R.id.relativ);
parent.addView(mGLView);
View view = LayoutInflater.from(this).inflate(R.layout.gui, null);
parent.addView(view);
```

Neméně důležitou vlastností je reakce uživatele s aplikací pomocí ovládacího prvku. Tomuto jevu se říká událost. V aplikaci je používána událost `setOnClickListener()` pro tlačítka[1] a událost`setOnSeekBarChangeListener` pro posuvník[2].

Zde můžete vidět příklad práce s posuvníkem a jeho reakcí na jeho možnosti:

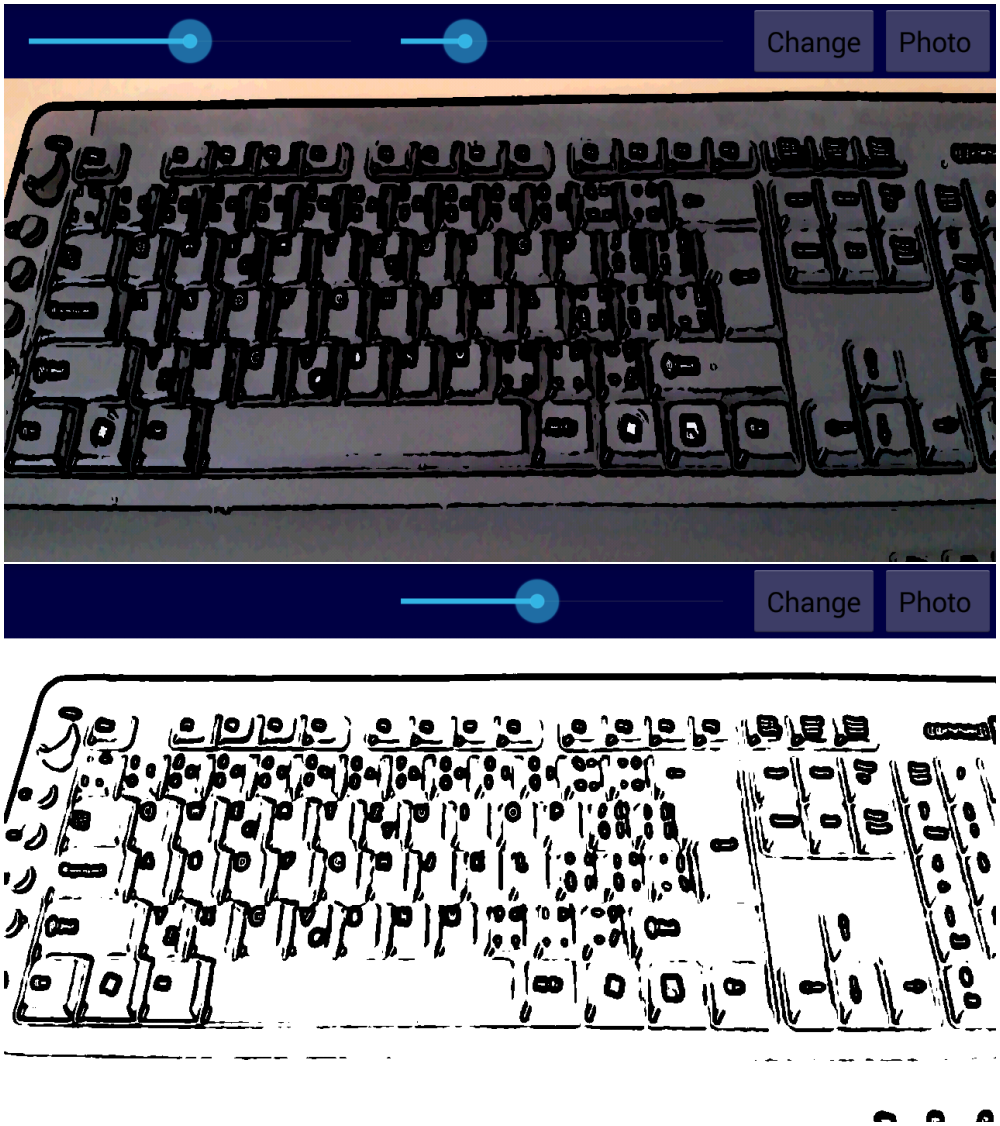
Kód 6.3: Vytvoření grafického rozhraní

```
seekBarb.setOnSeekBarChangeListener
  (new SeekBar.OnSeekBarChangeListener() {
  public void onProgressChanged(SeekBar seekBar, int i,
                                boolean b) {
    //hodnota získaná z posuvníku
  }
  public void onStartTrackingTouch(SeekBar seekBar) {
    //začátek práce s posuvníkem
  }
  public void onStopTrackingTouch(SeekBar seekBar) {
    //konec práce s posuvníkem
  }
});
```

6.3 Zpracování obrazu

V této části jsou popsány podstatné části implementace aplikace. Budou zde uvedeny důležité části zpracování obrazu z kamery a problémy, které nastaly při implementaci.

[1] http://developer.android.com/guide/topics/ui/controls/button.html
[2] http://developer.android.com/reference/android/widget/SeekBar.html



Obrázek 6.1: Grafické rozhraní aplikace

GLView

Zpracování obrazu je implementováno ve třídě `MyRenderer`. Tato třída pracuje s metodami, které jsou popsány v 3.2.

V metodě `onSurfaceCreated` jsou nastaveny jednotlivé buffery, které budou potřeba v další části programu. Dále jsou zde vytvořeny shadery 6.4 a z nich vytvořen prográmek pro zpracování pixelů obrazovky[10].

Kód 6.4: Příklad vytvoření shaderu

```
int vertexShader=GraphicTools.loadShader(GLES20.GL_VERTEX_SHADER,
                                         GraphicTools.vs_Image);

 int fragmentShader = GraphicTools.loadShader(
 GLES20.GL_FRAGMENT_SHADER, GraphicTools.fs_EdgeDetector);
```

```
GraphicTools.sp_EdgeDetector = GLES20.glCreateProgram();
GLES20.glAttachShader(GraphicTools.sp_EdgeDetector,
                      vertexShader);
GLES20.glAttachShader(GraphicTools.sp_EdgeDetector,
                      fragmentShader);
GLES20.glLinkProgram(GraphicTools.sp_EdgeDetector);
```

V této metodě je nadále inicializována kamera. V inicializaci kamery se nastaví, aby `SurfaceTexture`[5] byla nastavena jako živý pohled z kamery.

Metoda `onSurfaceChanged` je potřeba při změně orientace obrazovky. Jelikož je aplikace nastavena pouze na horizontální zobrazení, tak se tato metoda téměř nevyužívá. Zavolá se pouze při spuštění aplikace, kdy se nastaví velikost projekční matice a souřadnice zobrazovací plochy.

V metodě `onDrawFrame` probíhá veškerá práce se zpracováním obrazu z kamery. Nastaví se zde pozice pohledu kamery. Vypočítá se projekční matice. Důležitým prvkem v této metodě je otočit projekční matici o devadesát stupňů. Kdyby se tato matici neotočila, výsledný obraz by byl otočen právě o devadesát stupňů.

Po této akci je projekční matice poslána do metody `Render`, kde probíhá zpracovávání. V této metodě probíhá komunikace se shadery. Nejprve než se s těmito programky začne pracovat, tak musí být přidány do OpenGL ES prostředí[6]. Zde se těmto programkům předají souřadnice a textura, se kterou má pracovat a hodnoty, které jsou nastaveny pomocí ovládacích prvků grafického prostředí. Aplikuje projekční matici a poté se obraz upravený z kamery zobrazí pomocí:

Kód 6.5: Vykreslení textury na displej

```
GLES20.glDrawElements(GLES20.GL_TRIANGLE_FAN, 6,
                      GLES20.GL_UNSIGNED_SHORT, drawListBuffer);
```

Shadery

Důležitou částí jsou právě shadery 3.2. V těchto programcích se vytváří jednotlivé části GPU. Shadery jsou napsány pomocí jazyka GLSL(OpenGL Shading Languague).

V aplikaci je vytvořen jeden vertex shader a dva fragment shadery. Tyto dva programky jsou stěžejní částí aplikace. Zde se provádí převod obrazu přijímaného z kamery na převod na požadovaný filtr. Hodnoty v shaderu se dají měnit pomocí posuvníků, které se nacházejí na obrazovce aplikace. Hodnota posuvníku pro výběr citlivosti hran dosahuje hodnot [10,80). Nastavení hodnot je nastaveno tak, aby hodnota nemohla být menší jako deset. Takto je omezena detekce hran na určité hodnoty. V shaderu je tato hodnota vynásobena 0,01 a tím se pokryje celý rozsah hodnot při detekování hran. Posuvník pro změnu intenzity pracuje na stejném principu, akorát je omezen na hodnoty [2,60]. Kdy se tedy místo původních 255 odstínu barvy vloží právě tolik odstínů kolik je zadáno posuvníkem.

V těchto programech se provádí detekce hran pomocí Sobelova algoritmu 4.1. Práh, který je vyhodnocen tímto algoritmem se aplikuje přímo na původní obraz. Hodnota prahového filtru se zadává posuvníkem na hlavní obrazovce. Podle této hodnoty se bude určovat, zda bude na daném pixelu černá či bílá barva. Při komiksovém filtru vstupuje do hry ještě nastavení rozsahu hodnot reprezentující intenzitu. Omezení barevného spektra se zadává druhým posuvníkem na obrazovce aplikace. Podle zadané hodnoty se upraví barvy z původní velikosti barevného spektra na hodnoty zadané posuvníkem. Barevné kanály se vyhodnocují

zvlášť jak pro červenou, zelenou tak i modrou. Pomocí hodnoty zadané druhým posuvníkem se nastaví hodnota, kdy pixel bude vyhodnocen jako hrana. V kladném případě se místo barevného pixelu zobrazí černá a v opačném případě se zobrazí barva pixelu omezená v předchozím kroku.

Zde jsou popsané jednotlivé kroky jak probíhá převod původního obrazu na obraz s komiksovým efektem v shaderu.

1. Shaderu jsou předány hodnoty a souřadnice pixelu, na který se má změna provést. Dále jsou předány hodnoty proměnných, které jsou nastaveny posuvníky. Pro práh [10,80) a pro barevné spektrum [2,60].

2. Redukce barevného spektra, na hodnotu zadanou posuvníkem = zadaný práh.

3. Výpočet pomocí Sobelova algoritmu.

4. Zjištění hodnoty z výpočtu Sobelova algoritmu = hodnota podrobená testu hrany [0.0,1.0].

5. Na základě hodnoty výsledku Sobelova algoritmu, probíhá porovnání s hodnotou získané z posuvníku nastavující hodnotu prahu.

6. Když je hodnota ze Sobelova algoritmu < hodnota zadaná posuvníkem, tak se na místo daného pixelu vloží černá barva. V opačném případě se danému pixelu vloží hodnota redukovaného barevného spektra.

Při vytváření černobílého efektu probíhá postup totožně, akorát je vynechán bod 2 a v bodu 6 se na místo barevného spektra vloží bílá barva.

6.4 Uložení obrázku

Při řešení uložení obrázku do paměti jsem si nejprve musel zvolit, jakou metodu ukládání použiji viz. 5.4. Rozhodl jsem se pro ukládání textur. Tato volba se mi zdála daleko rychlejší a navíc nemusím znovu prohánět vyfocenou fotografii filtry.

Nejprve je za potřebí získat přefiltrované pixely. Čtení pixelů probíhá následujícím způsobem:

Kód 6.6: Čtení pixelů z bufferu

```
IntBuffer intBuffer = IntBuffer.wrap(bitmapBuffer);
gl.glReadPixels(x, y, w, h, GL10.GL_RGBA,
                GL10.GL_UNSIGNED_BYTE, intBuffer);
```

Kde x a y jsou počáteční body displeje, w a h je šířka a výška displeje, formát, typ a nakonec buffer pomocí, kterého tato funkce pracuje s pixely. Poté se zpracuje pixel po pixelu a nakonec je vytvořena bitmapa. Pro ukázku se bitmapa tvoří takto:

Kód 6.7: Vytvoření bitmapy

```
Bitmap.createBitmap(bitmapSource, w, h, Bitmap.Config.ARGB_8888);
```

Po vytvoření bitmapy je nutno fotografii uložit do paměti. Ukládá se tím stylem, že nejprve je nutno zjistit, zda-li se v zařízení vyskytuje požadovaná složka, do které se aplikace ukládá soubory. Když by se složka v zařízení nevyskytovala, tak se musí vytvořit.

Samozřejmě každá fotografie musí mít svoje jedinečné jméno. Pro vytvoření jedinečného jména jsem zvolil formát, který bude vypadat následovně: `CC_datum.jpg`, kde datum představuje přesný čas na vteřiny ve tvaru: `rok,měsíc,den,hodina,minuta,vteřina`.

Samozřejmě nesmí být zapomenuto kam vlastně se fotografii uloží. Pro uložení byla zvolena externí paměť. Dále je nutno uživatele upozornit při pořizovaní fotografie. Pro tuto informaci je použit informační dialog. Jedná se o zprávu, která se zobrazí ve spodní části obrazovky. Jde o tzv. Toast objekt[3]. Ukázku informačního dialogu viz:6.8

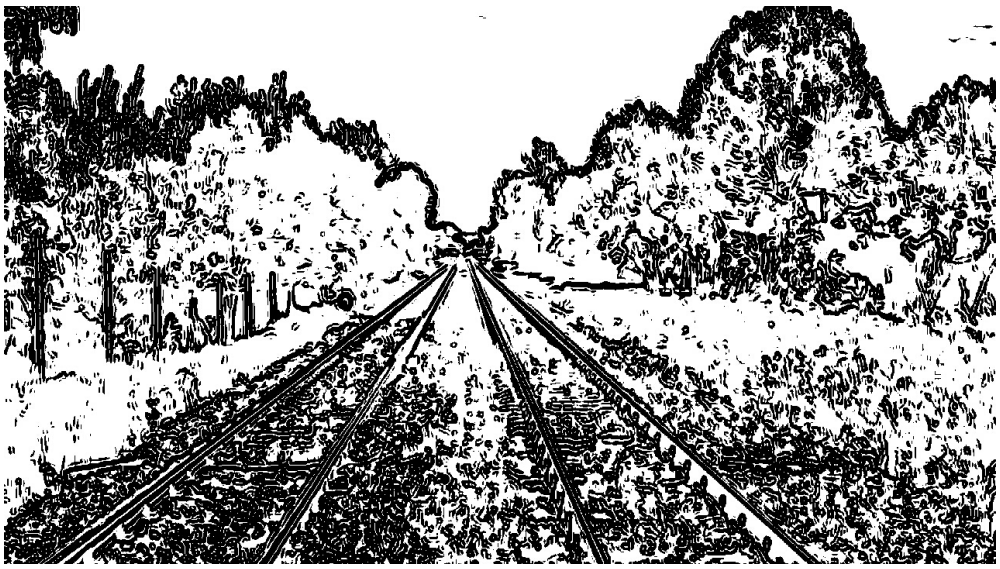
Kód 6.8: Vytvoření informačního dialogu

```
Toast.makeText(getApplicationContext(), "taking photo...",
                                        Toast.LENGTH_SHORT).show();
```

6.5 Výsledná aplikace

Po implementaci všech důležitých částí aplikace může být popsán finální výsledek aplikace. Jak již bylo zmíněno aplikace poskytuje dva filtry. Mezi těmito filtry lze jednoduše přepínat pomocí tlačítka na změnu filtru. Na následujících obrázcích jsou pořízeny fotografie se stejným nastavením citlivosti detekce hran. Samozřejmě intenzita barvy neměla na černobílý efekt žádný vliv.

První filtr je černobílý filtr, který je ilustrován na obrázku 6.2. Tento filtr se vyznačuje detekcí hran, kde nejsou detekovány hrany, tak tam zobrazí pouze bílou barvu. Druhý filtr, který byl implementován je komiksový filtr, který je ilustrován na obrázku 6.3.



Obrázek 6.2: Výsledné obrázky aplikace - Černobílý efekt

6.6 Testování

V této kapitole bude popsáno, jak byla vytvořená aplikace testována. Testování probíhalo na přístroji LG Optimus L9 s operačním systémem Android Kit Kat(4.4.2) a rozlišením

[3]Informační dialog viz. http://developer.android.com/guide/topics/ui/notifiers/toasts.html

Obrázek 6.3: Výsledné obrázky aplikace - Komiksový efekt

1280x720 pixelů. Dále na zařízení HTC Desire 500 se systémem Android Jelly Bean(4.1.2) a rozlišením 800x480 pixelů. Po menších problémech při zobrazování ovládacích prvků na zařízeních s rozdílným rozlišením byl tento problém vyřešen a aplikace se zobrazuje na všech zařízeních dle její specifikace.

Při průběžném testování na uživatelích byly zjištěny nedostatky na grafickém rozhraní. První připomínky spočívaly v nepřehlednosti prvků, které byly připnuty na horní okraj obrazovky. Tyto prvky se jevily nepřehledné z důvodu, že obraz splýval s těmito prvky, a tak tyto prvky nebyly příliš dobře vidět. Tento problém byl vyřešen přidáním pozadí pod tyto prvky. Druhá připomínka spočívala v matení uživatele. Při spuštění černobílého efektu se uživateli zobrazoval posuvník, kterým se dá nastavovat intenzita barev komiksového efektu. Tento problém byl vyřešen elegantním skrytím posuvníku při neúčásti komiksového filtru.

Úkoly

Pro lepší testování aplikace, jsem ji umístil na oficiální obchod od firmy Google, Google play[4]. Díky tomuto umístění aplikace bylo testování aplikace značně jednoduší. Testování probíhalo stylem nainstaluj, spusť a pracuj. Tedy stylem pozorování jednotlivce. Uživatelům byly zadány dva úkoly. První úkol spočíval ve spuštění aplikace a nastavení filtru, který je nastaven při spuštění na černobílý efekt. Úkolem bylo nastavení citlivosti detekování hran. Dalším úkolem bylo přepnout z počátečního filtru na následující filtr. Uživatel tedy musel přepnout do komiksového efektu. Při přepnutí do tohoto efektu měl uživatel za úkol změnit intenzitu barev. Taktéž dostal za úkol změnit citlivost detekování hran. Po nastavení do hodnot, které uživateli vyhovují měl pořídit fotku.

[4]https://play.google.com/store/apps/details?id=com.comicscamera.xstefl05.comicscamera

Výsledky

Testování této aplikace probíhalo na deseti uživatelích různých věkových kategorií. Aplikace byla instalována z obchodu Google play. Každý uživatel měl k dispozici svoje mobilní zařízení s operačním systémem Android. Nejčastěji uživatelé testovali na Android verzi 4.4 KitKat, pro kterou je aplikace primárně určena.

Z důvodu velice jednoduchého ovládání aplikace neměli uživatelé při zadaných úkolech příliš mnoho problémů. Při změně citlivosti detekování hran při černobílém efektu se právě ukázala výhoda pouze jednoho posuvníku, díky kterému uživatel okamžitě věděl jaký posuvník slouží pro nastavování hodnot citlivosti hran. Při dalším úkolu už uživatelé věděli, který posuvník slouží na změnu citlivosti detekce hran, a proto nebyl žádný problém změnit intenzitu barev, kdy si uživatel vydedukoval, že má právě použít druhý posuvník. Díky jasným popiskům tlačítek taktéž uživatelé okamžitě věděli, které tlačítko slouží ke změně filtru, a které pro pořízení fotografie. Kdyby si stále nebyli jisti, které tlačítko slouží k jakému účelu, tak při pořizování fotografie na uživatele vyskočí informační dialog s informací, že právě provádí pořizování fotografie.

Uživatelé měli i pár připomínek o které by se aplikace mohla rozšířit. Především by oceněli možnost výběru z více filtrů. Další připomínka byla pro provedení propojení se sociálními sítěmi. Díky tomuto propojení by mohl uživatel okamžitě nahrát fotku právě na tuto síť. Poslední připomínka byla k pořízené fotografii přidat GPS razítko. Uživatel by poté mohl s odstupem času vidět, kde fotografii pořídil.

Kapitola 7

Závěr

Cílem této práce bylo vytvořit mobilní aplikaci pro platformu Android. Po dohodě s vedoucím práce byla jako grafická knihovna zvolena OpenGL ES 2.0, která v současnosti podporuje většinu zařízení se systémem Android. Hlavním požadavkem, který byl splněn, bylo vytvořit komiksovou kameru v reálném čase, která bude fotografovat jako normální fotoaparát. Jediný rozdíl oproti klasickému fotoaparátu je v tom, že fotografie bude ukládat se zadaným efektem, který se zrovna zobrazuje.

Byly celkem zvoleny dva efekty. První efekt je černobílý. Tento efekt vytvoří hrany, při změně barev v okolí daného pixelu a kde není hrana, vloží pouze bílou barvu. Druhý efekt napodobuje komiksový efekt. Tento efekt redukuje barevné spektrum dle zadané hodnoty. Kde se barvy liší o určitou hodnotu, tam efekt vloží černou hranu. Intenzita barev je zadávána pomocí posuvníku a taktéž se dá určit citlivost detekce hran.

Tato aplikace je oproti konkurentům velice rychlá z důvodu, že je využita právě knihovna OpenGL ES. Zpracování pomocí shaderu má za důsledek, velice rychlé zpracování obrazu ze vstupu kamery a tímto způsobem zpracování se liší od svých konkurentů.

Aplikace byla taktéž umístěna na oficiální obchod aplikací pro Android na Google play. Kde po zadání výrazu „comic camera“ je vyhledána okolo 110. shody názvu aplikací.

Do budoucna by se aplikace mohla rozšířit o další funkce. Filtrů by se zde dalo implementovat značné množství. Do komiksového filtru se dá přidat stínování. Udělat nový efekt, který bude zobrazovat obraz z kamery, jako by byl nakreslený tužkou. Dále by se dal implementovat sépie efekt. A různé efekty pomocí metody rozptylováním 4.4.

Vylepšení se nedá namítat pouze ze strany přidávání filtrů. Uživatelé by také určitě ocenili komunikaci aplikace se sociálními sítěmi, které navrhovali samotní uživatelé při testování aplikace. Taktéž by mohla aplikace poskytovat časové razítko pořízené fotografie.

Literatura

[1] Allen Grant: *Android 4*. Computer Press, 2013, iSBN 978-80-251-3782-6.

[2] Cartoonize: Image Cartoonize application [online]. `http://www.cartoonize.net`, 2015-04-28 [cit. 2015-04-28].

[3] Google Inc.: Android developers [online]. `http://developer.android.com/guide/components/activities.html`, 2015-04-28 [cit. 2015-04-28].

[4] Google Inc.: Android developers [online]. `http://developer.android.com/guide/topics/graphics/opengl.html`, 2015-04-28 [cit. 2015-04-28].

[5] Google Inc.: Android developers [online]. `hhttp://developer.android.com/reference/android/graphics/SurfaceTexture.html`, 2015-04-28 [cit. 2015-04-28].

[6] Google Inc.: Android developers [online]. `http://developer.android.com/training/graphics/opengl/draw.html`, 2015-04-28 [cit. 2015-04-28].

[7] HASHIMI, Sayed Y, Satya KOMATINENI a Dave MACLEAN: *Pro Android 2*. Distributed to the book trade worldwide by Springer-Verlag, 2010, iSBN 978-1-4302-2659-8.

[8] Jiří Žára, Bedřich Beneš, Jiří Sochor, Petr Felkel: *Moderní počítačová grafika*. Computer Press, 2004, iSBN 80-251-0454-0.

[9] John Kessenich: The OpenGL ES Shading Languague [online]. `https://www.khronos.org/files/opengles_shading_language.pdf`, 2009-05-12 [cit. 2015-05-12].

[10] Kevin Brothaler: *OpenGL ES 2 for Android: A Quick-Start Guide*. The Pragmatic Programmers, 2013, iSBN 978-1-93778-534-5.

[11] Mark Segal, Kurt Akeley: The OpenGL Graphic System: A Specification [online]. `https://www.opengl.org/documentation/specs/version2.0/glspec20.pdf`, 2004-10-22 [cit. 2015-05-12].

[12] PULLI, Kari: *Mobile 3D graphics: with OpenGL ES and M3G*. Elsevier/Morgan Kaufmann Publishers, 2008, iSBN 978-012-373727-4.

[13] Richard Szeliski: *Computer Vision: Algorithms and Application*. Springer, 2010, iSBN 978-1-84882-934-3.

[14] The Khronos Group Inc.: OpenGL Documentation [online]. `https://www.opengl.org/sdk/docs/`, 2015-04-28 [cit. 2015-04-28].

[15] Wikipedia: Android (operating system) [online]. `http://en.wikipedia.org/wiki/Android_%28operating_system%29`, 2015-04-28 [cit. 2015-04-28].

Příloha A

Obsah CD

Na přiloženém CD se nacházejí následující složky:

- `/zprava/` - Tato písemná zpráva ve zdrojových souborech LaTeXu a výsledné PDF.
- `/aplikace/` - Zdrojové kódy vytvořené aplikace.
- `/video/` - Prezentační video.
- `/apk/` - Instalační apk soubor aplikace.