FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Vytváření zásuvných modulů pro Adobe Photoshop
Student:	Vít Černý
Vedoucí:	Ing. Jan Buriánek
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

V profesionálním světě počítačové grafiky je velmi populární rastrový editor Adobe Photoshop. I přes jeho nezměrné možnosti je tu občas nutnost vytvořit speciální funkce, které systém dovoluje implementovat pomocí tzv. zásuvných modulů (plugins).
V práci se zaměřte na postup při vytváření zásuvných modulů. Jeden takový modul navrhněte, popište celý postup výroby a specifika jeho vývoje. Pokuste se popis udělat tak, aby z něj mohli čerpat další studenti.
1) Prostudujte systém vytváření zásuvných modulů pro systém Adobe Photoshop. Nalezněte obecné nástroje pro vytváření zásuvných modulů.
2) Navrhněte zásuvný modul, včetně uživatelského rozhraní.
3) Nastudovaný postup výroby ověřte implementací zásuvného modulu.
4) Celý postup výroby zdokumentujte, včetně popisu vstupních a výstupních funkcí a částí zdrojových kódů. Popište odlišnosti od vytváření standardních aplikací.
5) Ověřte robustnost implementovaného zásuvného modulu na různých velikostech obrázků a barevných modelech.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 5. února 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Vytváření zásuvných modulů pro Adobe Photoshop

Vít Černý

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jan Buriánek

15. května 2019

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 15. května 2019 ……

Odkaz na tuto práci

Černý, Vít. *Vytváření zásuvných modulů pro Adobe Photoshop*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Tato bakalářská práce se věnuje postupu při vytváření zásuvných modulů pro aplikaci Adobe Photoshop. Práce také poskytuje základní informace o tom, co jsou zásuvné moduly, jak fungují a jak se vytváří. Dále jsou v práci popsány konkrétní postupy tvorby zásuvných modulů typu filter s použitím jazyka C++ a tvorba uživatelského rozhraní s využitím MFC (Microsoft Foundation Classes). V praktické části je vytvořen zásuvný modul pro mapování barevných komponent v modelu HSB (Hue Saturation Brightness) pomocí Bézierových křivek.

Klíčová slova Adobe Photoshop, Photoshop SDK, počítačová grafika, zásuvný modul, plug-in, rozšíření, filtr, HSB

Abstract

This bachelor thesis describes the process of creating Adobe Photoshop plug-in modules. This thesis also provides information about what plug-ins are, how they work and how to create them. Furthermore, the thesis describes specific methods for creating filter plug-ins using C++ programming language and creating a user interface using MFC (Microsoft Foundation Classes). In the practical part, a plug-in is created for HSB (Hue Saturation Brightness) component mapping using Bézier curves.

Keywords Adobe Photoshop, Photoshop SDK, computer graphics, plug-in, extension, filter, HSB

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisu kódu

Úvod

V profesionálním světě počítačové grafiky je velmi populární rastrový editor Adobe Photoshop. I přes jeho nezměrné možnosti je občas nutnost vytvořit speciální funkce, které systém dovoluje implementovat pomocí tzv. zásuvných modulů (plugins).

Práce je zaměřena na zdokumentování procesu tvorby zásuvných modulů pro aplikaci Adobe Photoshop, a to tak, aby z ní mohli čerpat další studenti nebo zájemci o programování grafických pluginů. Pro snazší pochopení problematiky jsou v práci použity jak triviální, názorné ukázky zdrojových kódů, tak i složitější ukázky, které jsou použity v praktické části.

Výsledek práce využijí uživatelé aplikace Adobe Photoshop, kteří mohou použít hotový zásuvný modul. Dále je práce určena programátorům, kteří se chtějí zabývat vývojem zásuvných modulů a využijí jednak teoretickou část, která se zabývá postupem tvorby zásuvných modulů, tak i praktickou část, která může sloužit jako inspirace pro psaní zdrojového kódu.

Práce začíná teoretickým úvodem do zásuvných modulů z pohledu uživatele. V této kapitole je popsáno, co tvoří zásuvné moduly, jaké existují kategorie a jak se používají. Další kapitola popisuje možnosti vývoje zásuvných modulů ale i další způsoby, jak rozšířit funkcionalitu aplikace Photoshop. Následující kapitola se věnuje fungování zásuvného modulu z pohledu programátora. Jsou zde popsány jednotlivé fáze běhu zásuvného modulu a co se od každé fáze očekává. Následuje praktická kapitola, která se věnuje přípravě SDK, vytvoření a konfigurace projektu včetně uživatelského rozhraní. Výsledkem této kapitoly je spustitelný zásuvný modul. Poslední kapitola řeší zpracování obrazových dat. Je zde popsáno, jakým způsobem jsou organizována obrazová data, jak je přečíst, jak upravit a jak je vrátit zpět aplikaci Photoshop.

Kapitola 1

Cíl práce

Tato práce si klade za cíl seznámit čtenáře se zásuvnými moduly pro aplikaci Adobe Photoshop. Cílem práce je nejenom popsat fungování zásuvných modulů typu filter ale i zdokumentovat postup jejich vytváření včetně uživatelského rozhraní. Dalším cílem této práce je vysvětlit strukturu obrazových dat a jejich zpracování tak, aby byl čtenář schopen vytvořit svůj vlastní zásuvný modul typu filter.

Kapitola 2

Motivace

Představte si situaci, kdy jako programátor chcete vyzkoušet nový algoritmus na zpracování obrazu. Realizace tohoto nápadu by vyžadovala naprogramování samostatné testovací aplikace. Tato testovací aplikace by musela mimo jiné řešit načítání a dekódování grafických formátů, barevné profily (ICC), uživatelské rozhraní, vykreslování, a další záležitosti, které přímo nesouvisí se samotným grafickým algoritmem.

Pro programátora je v tomto případě mnohem elegantnější využít mechanizmu zásuvných modulů. Využití možností zásuvného modulu přináší výhodu v tom, že se více času věnuje samotnému grafickému algoritmu, zatímco méně času je promarněno vývojem samostatné aplikace. Další výhodou je, že zásuvný modul může využívat funkcionalitu hostitelské aplikace (v našem případě Photoshop), jako je například převzorkování, změna bitové hloubky a podobně.
Výhody použití zásuvných modulů [1]:

- odpadá starost s formáty souborů,
- vykreslování řeší hostitelská aplikace,
- levnější a rychlejší vývoj,
- vyšší flexibilita,
- snazší přidávání funkcí.

Kapitola 3

Úvod do zásuvných modulů

3.1 Modul a host

Zásuvné moduly aplikace Adobe Photoshop (plug-in) jsou softwarové komponenty vyvíjené společností Adobe Systems nebo jinými vývojáři, pro rozšíření standartní funkcionality aplikace Adobe Photoshop. Tyto moduly mohou být přidávány nebo aktualizovány nezávisle koncovými uživateli, aby si přizpůsobili Photoshop svým konkrétním potřebám [2].

Tato práce také často odkazuje na **hostitelskou aplikaci** (plug-in host). Hostitelská aplikace je zodpovědná za načítání zásuvných modulů do paměti a jejich spouštění. V této práci je hostitelskou aplikací Adobe Photoshop. Další aplikace, které fungují jako hostitelská aplikace a podporují některé zásuvné moduly Photoshopu: Adobe After Effects, Adobe Premiere Pro, Adobe Illustrator, a další.

Většina hostitelských aplikací jsou samostatné spustitelné aplikace, ale není tomu tak vždy. Hostitelská aplikace může být sama o sobě zásuvný modul. Existuje například zásuvný modul „Photoshop Adapter“, který umožňuje aplikaci Adobe Illustrator spouštět některé zásuvné moduly určené pro aplikaci Adobe Photoshop [2].

3.2 Typy zásuvných modulů

V současné době podporuje Photoshop devět typů zásuvných modulů [2][1]:

Automation moduly přistupují ke všem skriptovatelným událostem ve Photoshopu. To jsou v podstatě všechny akce z menu a nástrojů, které může uživatel použít na obrazovou vrstvu. Tyto moduly se zobrazují v menu File → Automate.

Color Picker moduly umožňují implementovat různé nástroje typu Kapátko pro výběr barev. Spustí se vždy když uživatel požádá o výběr barvy (na-

příklad kliknutím na barvu popředí/pozadí v paletě nástrojů). Výchozí nástroj pro výběr barev se nastaví v menu Edit → Preferences → General.

Export moduly exportují existující obraz. Typicky se používají pro zajištění podpory pro nestandartní (nediskové) typy exportních operací, například odeslání dat do tiskárny nebo jiných zařízení. Tyto moduly jsou přístupné v menu File → Export.

Filter moduly jsou nejznámějším typem pluginů, protože pracují přímo s obrazovými daty a mění jejich vzhled. Standartní úpravy jako jsou „Gaussian Blur“ nebo „Unsharp Mask“ jsou typu Filter. Tyto moduly se zobrazují v menu Filter.

Format moduly, nazývané také „File Format“ nebo „Image Format“ moduly poskytují podporu pro čtení a zápis nepodporovaných typů obrazových formátů. Zobrazí se jako nová možnost v rozbalovací nabídce v dialogovém okně „Open“, „Save As“ a „Save a Copy“.

Stack Renderer moduly se používají ke zpracování skupiny obrazů (definovaných jako „Smart Object“) do jednoho obrazu. Typické použití je seskupení série snímků s nízkou expozicí k vykreslení jednoho obrazu s nižním množstvím šumu. Zobrazují se v menu Layer → Smart Objects → Stack Mode.

Import moduly importují obraz ze vstupních zařízení, jakou jsou skenery nebo videokamery, a vkládají tyto obrazy do nového dokumentu. Používají se také ke čtení nepodporovaných formátů. Tyto moduly jsou přístupné z menu File → Import.

Measurement moduly přidávají položky k měření. Položky k měření lze konfigurovat z dialogového okna Image → Analysis → Select Data Points. Samotné měření lze provádět z menu Image → Analysis → Record Measurements.

Selection moduly upravují které pixely jsou vybrány v existujícím obraze a mohou vracet buď křivku nebo výběr pixelů. Tyto moduly se zobrazují v menu Select.

3.3 Soubor zásuvného modulu

Zásuvný modul musí být umístěn ve specifickém adresáři, aby ho Photoshop rozpoznal. V systému Mac OS musí být soubor zásuvného modulu umístěn v jenom z těchto umístění [2]:

- ve stejné složce jako aplikace Phtoshop,

- ve složce určené v nastavení aplikace Photoshop,
- ve složce určené v souboru Photoshop prefs.

V systému Windows musí být soubor zásuvného modulu umístěn ve složce „Plug-ins“, která se nachází ve složce s aplikací. Například: C:\Program Files\Adobe\Adobe Photoshop CC 2019\Plug-ins.

Jeden soubor obvykle obsahuje pouze jeden zásuvný modul. Je ale možné vytvořit soubor s více zásuvnými moduly. Tato možnost však není doporučena, protože to snižuje kontrolu uživatele nad tím, které moduly jsou nainstalovány. Jsou však situace, kdy může být vhodné mít v jednom souboru více než jeden zásuvný modul. Jedním z příkladů jsou párové import/export moduly nebo sada souvisejících filtrů [2].

Soubory zásuvných modulů by měli dodržovat pravidla uvedená v tabulce níže pro typový identifikátor v systému Mac OS a příponu souboru v systému Windows. Přestože se jedná pouze o doporučení, u starších verzí Photoshopu musí být dodržena, aby byl zásuvný modul správně rozpoznán [2].

Typ zásuvného modulu	Mac OS soubor	Windows přípona
Automation	8LIZ	.8LI
Color Picker	8BCM	.8BC
Export	8BEM	.8BE
Filter	8BFM	.8BF
File Format	8BIF	.8BI
Import	8BAM	.8BA
Measurement	8MEA	.8ME
Selection	8BSM	.8BS
Stack Renderer	8BAM	.8BA

Tabulka 3.1: Typové identifikátory zásuvných modulů

3.4 PiPL metadata

Při spuštění prohledá Photoshop všechny zásuvné moduly, aby v nich našel tzv. PiPL (plug-in property list) metadata. Díky tomu Photoshop ví, do jaké kategorie modul spadá (Filter, Color Picker, …). V dřívějších verzích Photoshopu se kategorie určovala typovým identifikátorem, ale nyní tomu tak není. Pokud PiPL metadata říkají, že se jedná o zásuvný modul typu Filter, bude s ním Photoshop zacházet jako s filtrem, bez ohledu na to, jakého typu je soubor nebo jakou má příponu [1].

PiPL metadata se také používají k informování hostitelské aplikace (nejen Photoshopu) o schopnostech zásuvného modulu. PiPL mohou například informovat Photoshop o barevných módech (RGB, CMYK, …), pro které je zásuvný modul navržen. Když uživatel pracuje v jiném módu, bude název zásuvného modulu zašedlý a nepůjde spustit. Tímto způsobem lze také nastavit práci s vrstvami, průhledností, vyrovnávací pamětí, a mnoho dalších vlastností [1].

Kapitola 4

Metody vývoje

4.1 Photoshop SDK a C++

Využití Adobe Photoshop SDK je jediný způsob, jak vytvořit plně vybavený zásuvný modul. K jeho používání je potřeba znalost jazyka C++ spolu s vývojovým prostředím Microsoft Visual Studio (doporučeno). Nevýhodou je, že pro Photoshop SDK je k dispozici velmi málo materiálů, a pokud nějaké existují, pak jsou zastaralé. Součástí SDK je i podrobná dokumentace, která se však věnuje popisu jednotlivých funkcí a nikoliv návodům [3].

Výhody:

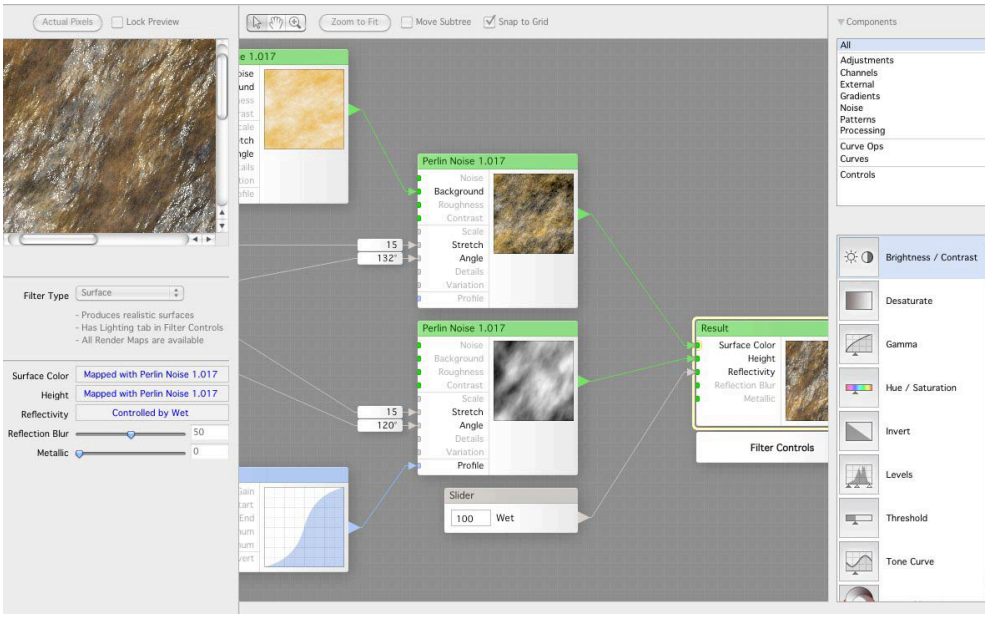
- Lze vytvořit libovolný plugin z kterékoliv kategorie.
- Jediným omezením jsou programátorské schopnosti.

Nevýhody:

- Potřebná znalost jazyka C++.
- Velmi málo dostupných informací.

4.2 Filter Forge [4]

Filter Forge je vizuální vývojový nástroj založený na předpřipravených komponentách, které se vzájemně propojují a vytváří tak posloupnost operací, která vede až ke konkrétnímu výsledku. Filter Forge lze použít jako prototypovací nástroj, který umožňuje zaměřit se na algoritmus a eliminuje potřebu nejprve naprogramovat veškeré nezbytnosti. Navíc má mnoho známých algoritmů pro zpracování obrazu implementovaných jako komponenty. Jeho nedostatkem je, že neumí vytvářet samostatné zásuvné moduly. Místo toho je Filter Forge sám zásuvným modulem, který spouští filtry vytvořené v rámci Filter Forge.



Obrázek 4.1: Filter Forge [5]

Výhody:

- Nejsou potřeba znalosti z programování.
- Systém propojování komponent.

Nevýhody:

- Neumí vytvořit samostatný zásuvný modul.
- Závislost na existujících komponentách.

4.3 Photoshop Scripting

Od verze CS podporuje Photoshop možnosti skriptování. Nejde sice přímo o programování zásuvných modulů, ale je to snazší způsob, jak rozšířit funkcionalitu Photoshopu. Pomocí speciálních příkazů lze spouštět nástroje Photoshopu, další skripty nebo tzv. „scripting-aware“ zásuvné moduly. Dále je možné ve scriptech využívat podmínky, cykly a další konstrukce [2].

Výhody:

- Snazší způsob, jak rozšířit funkcionalitu.
- Mnoho dostupných informací a návodů.

Nevýhody:

- Nezískáme samotný zásuvný modul.
- Omezení na skriptovatelné akce.

Kapitola 5

Jak funguje zásuvný modul

Když chce Photoshop spustit zásuvný modul, spustí jeho vstupní bod, což je funkce, jejíž identifikátor je uveden v PiPL metadatech, většinou pojmenovaná „PluginMain“. Této funkci Photoshop předá čtyři parametry:

1. **Selector** popisuje typ akce, která se od zásuvného modulu očekává.
2. **FilterParamBlock** obsahuje všechny obrazové informace.
3. **Data** slouží k uložení dat, která zůstanou uchovány po dobu spuštění Photoshopu.
4. **Result** slouží k uložení návratového kódu, například hodnota „noErr“.

Prototyp funkce PluginMain vypadá následovně:

```
DLLExport SPAPI void PluginMain(const int16 selector,
                                FilterRecordPtr filterParamBlock,
                                intptr_t* data,
                                int16* result);
```

Výpis kódu 1: Prototyp funkce PluginMain [2]

5.1 Selector

Když je spuštěn zásuvný modul, Photoshop zavolá jeho funkci PluginMain několikrát za sebou, pokaždé s jinou hodnotou v proměnné selector. Možné hodnoty proměnné selector jsou [1]:

filterSelectorAbout říká zásuvnému modulu, že má zobrazit dialog s informacemi o sobě. Pro tento selector Photoshop nepředává funkci PluginMain 2. paramatr. Místo toho je filterParamBlock nastaven na NULL.

filterSelectorParameters říká zásuvnému modulu, že pokud potřebuje od uživatele získat nějaké parametry, pak by měl zobrazit dialog pro zadání potřebných hodnot. Například pro filtr rozostření se zobrazí dialog pro zadání poloměru rozostření.

filterSelectorPrepare říká zásuvnému modulu, aby provedl inicializaci proměnných, naalokoval potřebnou paměť, a nakonec vrátil návratový kód — noErr nebo příslušný chybový kód.

filterSelectorStart říká zásuvnému modulu, že může začít se zpracováním obrazu. Na tomto místě je také vhodné ověřit aktuální parametry a případně nastavit výchozí hodnoty

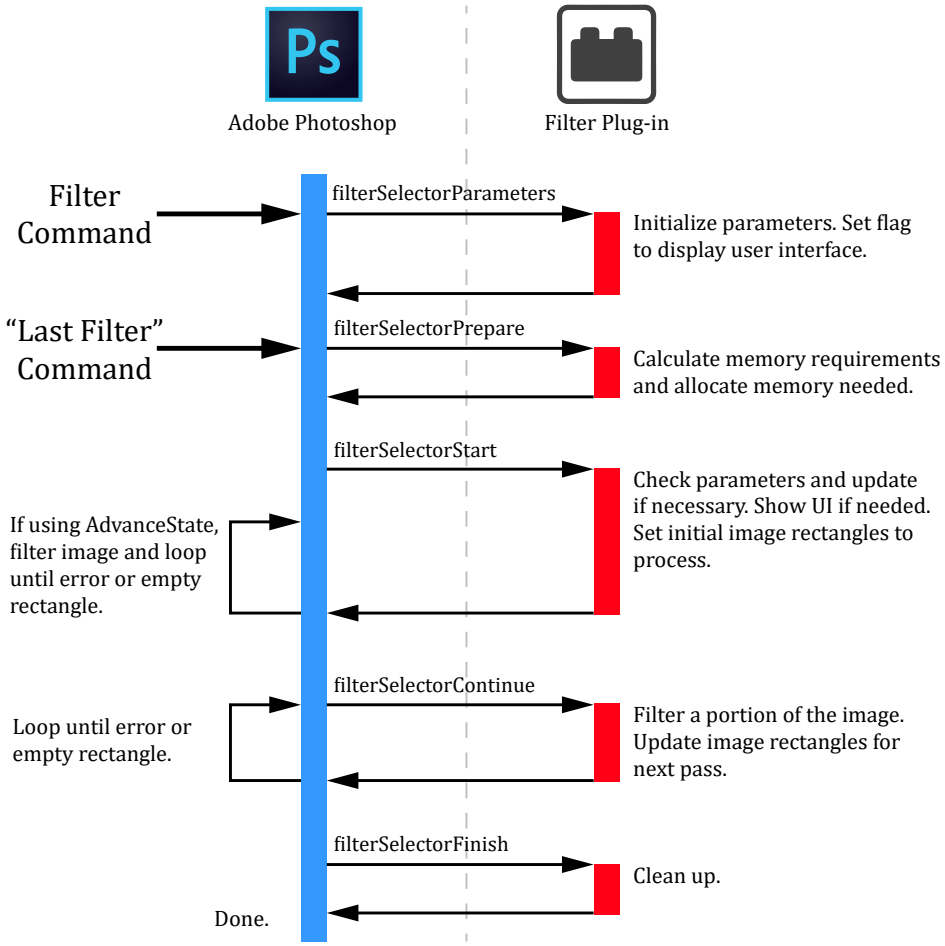
filterSelectorContinue informuje zásuvný modul o tom, že jsou k dispozici další data a může tak pokračovat ve zpracování obrazu. Tento selector je použit pouze pokud je obraz zpracováván po částech. Pokud zásuvný modul používá pro získávání dat funkci AdvanceState(), pak tento selector nebude použit.

filterSelectorFinish říká zásuvnému modulu, aby po sobě uklidil a uvolnil alokovanou paměť před tím, než bude ukončen.

filterBadParameters informuje o tom, že nastal problém s rozhraním zásuvného modulu.

filterBadMode informuje o tom, že zásuvný modul nepodporuje daný obrazový režim.

Celkový tok událostí je znázorněn na obrázku níže. Programování zásuvného modulu spočívá v obsluze těchto událostí. Všimněte si, že pokud je zásuvný modul spuštěn standartně z menu, obdrží jako první „filterSelectorParameters“. Pokud byl ale zásuvný modul spuštěn příkazem „Last Filter“ nebo zkratkou Ctrl + F, obdrží jako první „filterSelectorPrepare“.



Obrázek 5.1: Tok událostí po spuštění zásuvného modulu [2]

5.2 FilterParamBlock

Skrze tento parametr Photoshop předává proměnnou typu ukazatel na FilterRecord. FilterRecord je obrovská struktura, která obsahuje více než 200 proměnných obsahující různé informace o obrazových datech [1]. Většinu času je užitečných pouze několik málo proměnných a ostatní jsou používány ve speciálních případech. Některé důležité proměnné struktury FilterRecord jsou popsány v tabulce 5.1 níže. Důležité je také vědět, že tato struktura je primárním prostředkem, pomocí kterého spolu Photoshop a zásuvný modul komunikují. Struktura FilterRecord také obsahuje několik ukazatelů na funkce, které zajišťují komunikaci s Photoshopem [2].

Proměnná	Význam
imageSize	Šířka a výška obrazu v pixelech.
imageMode	Obrazový mód, například RGB, CMYK atd.
planes	Počet dostupných kanálů (barevné, alfa, maska).
depth	Bitová hloubka 1 barevného kanálu (1, 8, 16, 32).
filterRect	Oblast, která se bude zpracovávat. Jedná se o obdélníkové ohraničení výběru, nebo o ohraničení obrazu, pokud nebyl použit výběr.
inRect	Obdélníková oblast obrazu, ke které chce zásuvný modul přistupovat. Musí být podmnožinou filterRect.
outRect	Jako inRect, ale jde o oblast, do které se bude zapisovat.
inLoPlane	Index prvního kanálu, který chce zásuvný modul získat.
inHiPlane	Index posledního kanálu, který chce zásuvný modul získat.
outLoPlane	Index prvního kanálu, který chce zásuvný modul zapisovat.
outHiPlane	Index posledního kanálu, který chce zásuvný modul zapisovat.
inData	Ukazatel na požadovaná obrazová data.
outData	Jako inData, ale slouží pro zápis dat.
inRowBytes	Velikost 1 řádky pixelů v bytech.
outRowBytes	Jako inRowBytes, ale pro výstupní obrazová data.
abortProc	Funkce, která ověří, zda uživatel nezrušil operaci filtrování.
progressProc	Funkce, která zobrazuje průběh u dlouhých operací.

Tabulka 5.1: Důležité proměnné struktury FilterRecord

5.3 Data

Tento parametr je určen pro uchování dat mezi jednotlivými relacemi. Zásuvný modul při spuštění může nastavit tento ukazatel na svá globální data a Photoshop bude tyto data udržovat během doby spuštění. Při dalším spuštění zásuvného modulu Photoshop tímto parametrem předá poslední uložené hodnoty. Zásuvný modul může takto ukládat například informace od uživatele a při dalším spuštění použít naposledy zadané hodnoty. Podobně v případě, kdy je zásuvný modul spuštěn příkazem „Last Filter“, nemusí zobrazovat uživatelské rozhraní, ale rovnou použije poslední hodnoty [1].

5.4 Result

Pokaždé, když Photoshop spustí zásuvný modul, očekává, že parametr result bude obsahovat chybový kód označující, jestli došlo k chybě nebo jestli aktuální fáze proběhla bez problému. Pokud se vyskytne problém v jakékoliv fázi, je důležité to ihned oznámit Photoshopu, aby nespouštěl opakovaně PluginMain() s dalšími selectory.

Možné chybové kódy jsou „noErr“, označující bezproblémový průběh, nebo další kódy definované v hlavičkovém souboru „PITypes.h“. V případě použití standartního chybového kódu, Photoshop sám zobrazí informační dialog s chybou a jejím vysvětlením. Pokud se jedná o specifickou chybu, je možné ji zobrazit v rámci zásuvného modulu. V takovém případě je potřeba uložit do proměnné result kladnou hodnotu. Kladná hodnota informuje Photoshop o tom, že došlo k chybě, ale nemá zobrazovat žádnou chybovou hlášku [1].

Kapitola 6

Příprava projektu

6.1 Prostředí

Pro vývojáře zásuvných modulů je doporučeno používat stejné nástroje, které byly použity pro sestavení aplikace Photoshop. Tyto informace je možné dohledat v dokumentaci v sekci „BuildInfo“ nebo „What's New“. Například pro verzi Photoshop CC 2019 (Windows) byly použity tyto nástroje: Windows 7 64 bit SP 1, Microsoft Visual Studio 2017 verze 15.6.2, Windows 10 SDK verze 10.0.16299.0 [2]. Většinou ale není problém používat novější verze nástrojů.

6.2 Photoshop SDK

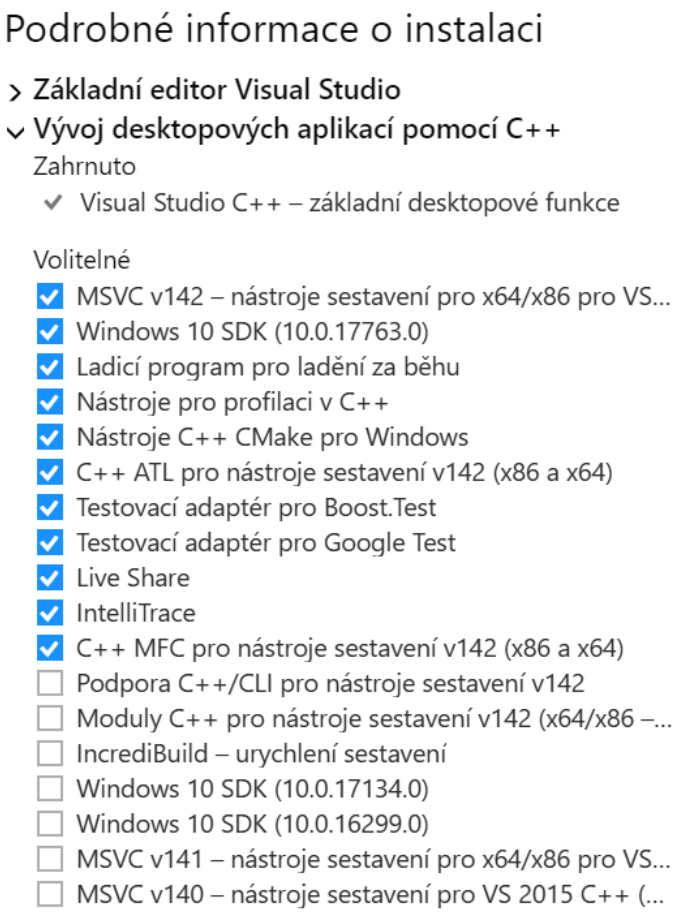
První věc, kterou je potřeba udělat, než začneme vytvářet zásuvný modul, je získat nejnovější verzi Adobe Photoshop SDK. Toto SDK lze stáhnout webových stránek Adobe: https://console.adobe.io/downloads/ps. Adobe nabízí pro Photoshop několik typů SDK: Common Extensibility Platform, Generator SDK, Photoshop Plug-In and Connection SDK a nakonec Scripting. Pro náš projekt budeme potřebovat 3. možnost — Photoshop Plug-In and Connection SDK.

Adobe Photoshop SDK obsahuje dvě části — dokumentaci a zdrojové kódy. Dokumentaci je možné zobrazit otevřením souboru „ReadMe.html“ v prohlížeči. Kromě HTML dokumentace obsahuje SDK i několik PDF souborů, které popisují některá témata více do hloubky, jako je „Adobe Plug-in Component Architecture“ (PICA). Dále obsahuje SDK zdrojové kódy v jazyce C/C++, které obsahují struktury, třídy a hlavičky funkcí, které jsou potřeba při vytváření zásuvného modulu.

6.3 Visual Studio projekt

6.3.1 Konfigurace Visual Studia

Při instalaci aplikace Microsoft Visual Studio vyberte sadu funkcí „Vývoj desktopových aplikací pomocí C++“. Ujistěte se, že v sekci Volitelné je vybraná položka „C++ MFC pro nástroje sestavení“. Tyto funkce budeme potřebovat při vývoji zásuvného modulu. Pokud je již Visual Studio nainstalované, spusťte Visual Studio Installer, zvolte Změnit a ujistěte se jsou nainstalovány všechny potřebné součásti.



Obrázek 6.1: Instalované funkce v aplikaci Visual Studio Installer

6.3.2 Založení projektu

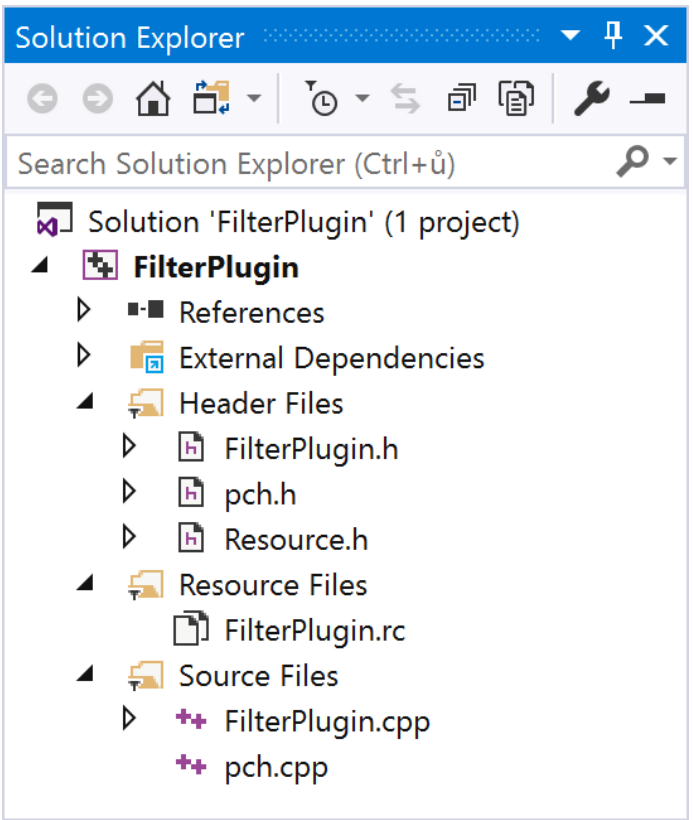
Zásuvný modul funguje jako DLL knihovna, a proto i náš projekt bude typu dynamic-link library s podporou MFC. Využití MFC (Microsoft Foundation

Classes) není povinné, slouží pouze k pohodlnější tvorbě uživatelského rozhraní. V tomto návodu budu předpokládat název projektu „FilterPlugin“.

V aplikaci Visual Studio vytvořte nový projekt typu „MFC Dynamic-Link Library“. Při výběru typu dll knihovny zvolte „Regular DLL with MFC statically linked“. Nebudeme potřebovat žádné přídavné funkce, a proto volby Automation a Windows Socket nechte nezaškrtnuté.

Abychom zjednodušily strukturu projektu, odstraníme všechny nepotřebné součásti. Ve Visual Studiu si zobrazte okno „Resource View“ (View → Other Windows → Resource View) a odstraňte položku VS_VERSION_INFO. Nyní v okně „Solution Explorer“ ve složce „Source Files“ odstraňte soubor „FilterPlugin.def“. Ve složce „Resource Files“ odstraňte soubor „FilterPlugin.rc2“. Ve složce „Header Files“ odstraňte soubory „targetver.h“ a „pch.h“ a nakonec soubor „framework.h“ přejmenujte na „pch.h“.

Pokud používáte starší verzi Visual Studia, která generuje soubory „stdafx.h“ a „stdafx.cpp“, práce se soubory „pch.h“ a „framework.h“ se vás netýká. Struktura projektu by nyní měla vypadat takto:



Obrázek 6.2: Struktura projektu

Provedené změny ve struktuře projektu vyžadují provést změny i ve zdrojových kódech. V souboru „pch.h“ odstraňte řádek s `#include "targetver.h`. Dále v souboru „FilterPlugin.cpp“ odstraňte řádek s `#include "framework.h"`. Nakonec v souboru „FilterPlugin.rc“ odstraňte řádky s `#include "targetver.h"` a `#include "res\FilterPlugin.rc2"`, a to i v sekci TEXTINCLUDE.

6.4 Příprava SDK

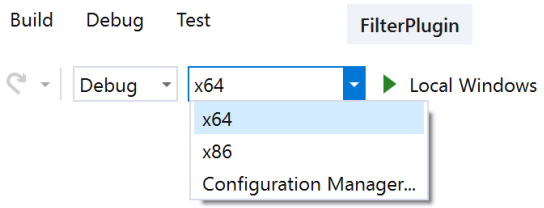
Ve složce, kde jsou uloženy zdrojové kódy, tedy FilterPlugin\FilterPlugin vytvořte novou složku „photoshopapi“. Rozbalte archiv obsahující Adobe Photoshop SDK a ve složce <SDK>\pluginsdk\photoshopapi zkopírujte složky „photoshop“ a „pica_sp“ do naší nově vytvořené složky „photoshopapi“. Dále ve složce <SDK>\pluginsdk\samplecode zkopírujte složku „common“ do naší složky „photoshopapi“. Nyní z naší složky „common“ odstraňte vše kromě složky „includes“. Nakonec ve složce <SDK>\pluginsdk\samplecode\resources zkopírujte soubor „Cnvtpipl.exe“ do naší složky „photoshopapi“.
Struktura naší složky „photoshopapi“ by nyní měla vypadat takto:

```
photoshopapi
├── common
│   └── includes
├── photoshop
├── pica_sp
└── Cnvtpipl.exe
```

Obrázek 6.3: Struktura SDK

6.5 Konfigurace projektu

Od verze CS6 je Photoshop vyvíjen pouze jako 64bitová aplikace, a proto dokáže spouštět pouze 64bitové zásuvné moduly. Jako první vyberte z nabídky Solution Platforms typ x64. Win32 ani x86 nebudou fungovat.



Obrázek 6.4: Změna typu projektu na x64

V okně Solution Explorer vyberte projekt FilterPlugin a následně zobrazte nastavení projektu tím, že v menu Project vyberete Properties. Nyní v sekci General změňte vlastnost „Target Extension“ na „.8bf“. Dále se ujistěte, že vlastnost „Configuration Type“ je nastavena na „Dynamic Library (.dll)“, „Use of MFC“ je nastaveno na „Use MFC in a Static Library“ a „Character Set“ je nastaveno na „Use Unicode Character Set“. Nastavení v sekci General by nyní mělo vypadat takto:

General	
Target Platform	Windows 10
Windows SDK Version	**10.0 (latest installed version)**
Output Directory	$(SolutionDir)$(Platform)\$(Configuration)\
Intermediate Directory	$(Platform)\$(Configuration)\
Target Name	$(ProjectName)
Target Extension	**.8bf**
Extensions to Delete on Clean	*.cdf;*.cache;*.obj;*.obj.enc;*.ilk;*.ipdb;*.iobj;*.resources;*.tlb;*.tli;*.tlh;*
Build Log File	$(IntDir)$(MSBuildProjectName).log
Platform Toolset	**Visual Studio 2019 (v142)**
Enable Managed Incremental Build	No
Project Defaults	
Configuration Type	**Dynamic Library (.dll)**
Use of MFC	**Use MFC in a Static Library**
Character Set	**Use Unicode Character Set**
Common Language Runtime Suppo	No Common Language Runtime Support
.NET Target Framework Version	
Whole Program Optimization	No Whole Program Optimization
Windows Store App Support	No

Obrázek 6.5: Nastavení projektu v sekci General

Nyní se v nastavení přepněte do kategorie C/C++ a do sekce General. Do pole „Additional Include Directories“ vložte odkazy na Photoshop SDK ve složce photoshopapi, konkrétně:

- photoshopapi\photoshop
- photoshopapi\pica_sp
- photoshopapi\common\includes

Výsledný řetězec tedy bude vypadat takto:
„photoshopapi\photoshop;photoshopapi\pica_sp;photoshopapi\common\includes“.
Dále vyberte „Yes (/MP)“ u vlastnosti „Multi-processor Compilation“.

V sekci Preprocessor přidejte tyto definice do pole „Preprocessor Definitions“:

- WINVER=_WIN32_WINNT_WS03
- ISOLATION_AWARE_ENABLED=1
- USING_MFC
- WIN32=1
- _CRT_SECURE_NO_DEPRECATE
- _SCL_SECURE_NO_DEPRECATE

Původní definice jako například „_USRDLL“ nebo „_WINDOWS“ ponechte beze změny. Dále v sekci Browse Information vyberte „Yes (/FR)“ u vlastnosti „Enable Browse Information“. V sekci Command Line přidejte do pole „Additional Options“ tyto přepínače:
„/experimental:external /external:I photoshopapi /external:W0“.
Tyto přepínače způsobí ignorování varování pro všechny zdrojové kódy umístěné ve složce photoshopapi. Pro Visual Studio 2019 je tato funkce ve stavu experimantal, proto je nutné jako první uvést přepínač „/experimental:external“.

Nyní se přepněte do kategorie Linker a v sekci Input odstraňte vše z pole „Module Definition File“. Dále v sekci Advanced změňte vlastnost „Random Base Address“ na No.

Nakonec v kategorii Build Events v sekci Post-Build Event vložte tento příkaz do pole „Command Line“:

```
copy /Y "$(OutDir)$(TargetName)$(TargetExt)"
        "C:\Program Files\Adobe\Adobe Photoshop CC
        ↪  2019\Plug-ins\$(TargetName)$(TargetExt)"
```

Výpis kódu 2: Příkaz Copy pro Post-Build Event

Tento příkaz způsobí, že po každém sestavení projektu se výsledný zásuvný modul zkopíruje do složky zásuvných modulů aplikace Photoshop. Tento příkaz je potřeba upravit pro konkrétní verzi Photoshopu nebo pro nestandartní umístění. Kopírování do složky s aplikací Photoshop vyžaduje oprávnění správce, a proto aby tento příkaz fungoval správně, je nutné, aby bylo Visual Studio spuštěné s oprávněním správce.

6.6 Příprava zdrojových kódů

Jak bylo zmíněno v kapitole 5, Photoshop spouští zásuvný modul několikrát za sebou, pokaždé s jinou hodnotou v proměnné selector. Základ zásuvného modulu proto bude tvořit funkce PluginMain spolu s obslužnými funkcemi pro každý selector.

6.6.1 Příprava obslužných funkcí

Nejprve na konec souboru pch.h (případně stdafx.h) vložíme potřebné hlavičkové soubory:

```
#include "PIDefines.h"
#include "PIAbout.h"
#include "PIFilter.h"
#include "PIUIHooksSuite.h"
#include "PIColorSpaceSuite.h"
#include "FilterPlugin.h"
```

Výpis kódu 3: Vložení potřebných hlavičkových souborů

Nyní v souboru FilterPlugin.h upravíme vygenerovanou třídu CFilterPluginApp. Do privátní části třídy přidáme ukazatel na strukturu FilterRecord, tedy `FilterRecord* m_FilterRecord` a ve veřejné části vytvoříme setter `void SetFilterRecord(FilterRecord* filterRecord);`. Do veřejné části třídy dále přidáme hlavičky obslužných funkcí:

```
void  Parameters(void);
void  Prepare(void);
int16 Start(void);
void  Continue(void);
void  Finish(void);
void  About(void);
```

Výpis kódu 4: Hlavičky obslužných funkcí

V souboru FilterPlugin.cpp nyní implementujeme tyto obslužné funkce, prozatím s prázdným tělem. Dále upravíme konstruktor CFilterPluginApp() tak, aby inicializoval členskou proměnnou m_FilterRecord na NULL. Jako další v souboru FilterPlugin.cpp vytvoříme funkci PluginMain. Visual Studio automaticky vytvořilo objekt `CFilterPluginApp theApp`, který použijeme k volání obslužných funkcí.

```
DLLExport SPAPI void PluginMain(const int16 selector,
                                FilterRecordPtr filterParamBlock,
                                intptr_t* data, int16* result)
{
    theApp.SetFilterRecord(filterParamBlock);

    switch (selector)
    {
    case filterSelectorAbout:
        theApp.About();
        break;
    case filterSelectorParameters:
        theApp.Parameters();
        break;
    case filterSelectorPrepare:
        theApp.Prepare();
        break;
    case filterSelectorStart:
        *result = theApp.Start();
        break;
    case filterSelectorContinue:
        theApp.Continue();
        break;
    case filterSelectorFinish:
        theApp.Finish();
        break;
    }
}
```

Výpis kódu 5: Funkce PluginMain

Nyní implementujeme obslužnou funkci pro selector Continue. Jakmile skončí fáze start, je zásuvný modul opakovaně spouštěn se selektorem Continue, dokud jsou k dispozici nějaká data. Pokud se dostaneme do fáze Continue, řekneme Photoshopu, že už jsme veškerá data zpracovali. Pokud bychom to neudělali, došlo by k zacyklení Photoshopu a zásuvného modulu.

```
void CFilterPluginApp::Continue(void)
{
    m_FilterRecord->inRect = { 0, 0, 0, 0 };
    m_FilterRecord->outRect = { 0, 0, 0, 0 };
    m_FilterRecord->maskRect = { 0, 0, 0, 0 };
}
```

Výpis kódu 6: Funkce Continue

6.6.2 Příprava uživatelského rozhraní

V okně Resource View přidáme nový dialog (Add Resource → Dialog). Prozatím ponecháme výchozí vzhled dialogu s dvěma tlačítky — Ok a Cancel. Dále přidáme do projektu novou třídu pojmenovanou „PluginDialog“, která dědí ze třídy „CDialog“. Do privátní části třídy přidáme ukazatel na strukturu FilterRecord, tedy `FilterRecord* m_FilterRecord` a upravíme konstruktor tak, aby volal konstruktor rodičovské třídy CDialog a inicializoval m_FilterRecord. Konstanta IDD_DIALOG1 je číselný identifikátor našeho nově vytvořeného dialogu.

```
PluginDialog::PluginDialog(FilterRecord* filterRecord) :
↪  CDialog(IDD_DIALOG1), m_FilterRecord(filterRecord)
{
}
```

Výpis kódu 7: Konstruktor třídy PluginDialog

Nyní když máme připravené dialogové okno, můžeme ho nechat zobrazit při spuštění zásuvného modulu. Nejprve je potřeba přidat `#include "PluginDialog.h"` do souboru pch.h a následně upravit funkci Start.

```
int16 CFilterPluginApp::Start(void)
{
    PluginDialog dlg(m_FilterRecord);
    INT_PTR result = dlg.DoModal();
    if (result == IDOK)
        return noErr;
    else
        return userCanceledErr;
}
```

Výpis kódu 8: Funkce Start

6.6.3 Přidání PiPL metadat

Jak bylo popsáno v kapitole 3, PiPL metadata jsou součástí zásuvného modulu a popisují jeho vlastnosti a schopnosti. PiPL se vytváří pomocí souboru typu REZ, který se kompiluje do PiPL souboru. Popis syntaxe jazyka REZ je mimo rozsah této práce, a proto jako základ využijeme už připravený soubor z jiného projektu. Ve složce s SDK se přesuňte do složky ukázkového projektu Dissolve, konkrétně <SDK>\pluginsdk\samplecode\filter\dissolve\common. V této složce zkopírujte soubor „Dissolve.r“ do složky našeho projektu se zdrojovými kódy. Zároveň soubor přejmenujte na „FilterPlugin.r“. Nyní přidejte soubor do projektu tak, že v okně Solution Explorer kliknete pravým tlačítkem na Resource Files, zvolíte Add → Existing Item a vyberete „FilterPlugin.r“.

V tomto souboru odstraňte všechny direktivy pro preprocesor kromě `#include "PIGeneral.h"` a dále celý blok začínající na „resource 'aete'“. Uvnitř bloku „resource 'PiPL'“ odstraňte bloky „FilterCaseInfo“ a „HasTerminology“. Dále změňte vlastnost Name na „FilterPlugin...“ a Category na „Plugin Project“. Nakonec nahraďte případné další výskyty řetězce „Dissolve“ nebo proměnné plugInName řetězcem „FilterPlugin“.

```
#include "PIGeneral.h"

resource 'PiPL' ( 16000, "FilterPlugin", purgeable ){
{
    Kind { Filter },
    Name { "FilterPlugin..." },
    Category { "Plugin Project" },
    Version {(latestFilterVersion << 16 ) | latestFilterSubVersion},
    Component { ComponentNumber, "FilterPlugin..." },

    #ifdef __PIMac__
        CodeMacIntel64 { "PluginMain" },
    #else
        #if defined(_WIN64)
            CodeWin64X86 { "PluginMain" },
        #else
            CodeWin32X86 { "PluginMain" },
        #endif
    #endif

    SupportedModes
    {
        noBitmap, doesSupportGrayScale,
        noIndexedColor, doesSupportRGBColor,
        doesSupportCMYKColor, doesSupportHSLColor,
        doesSupportHSBColor, doesSupportMultichannel,
        doesSupportDuotone, doesSupportLABColor
    },

    EnableInfo {"in (PSHOP_ImageMode, RGBMode, GrayScaleMode,"
                "CMYKMode, HSLMode, HSBMode, MultichannelMode,"
                "DuotoneMode, LabMode, RGB48Mode, Gray16Mode) ||"
                "PSHOP_ImageDepth == 16 ||"
                "PSHOP_ImageDepth == 32" },

    PlugInMaxSize { 2000000, 2000000 },
    MonitorScalingAware {},
    FilterLayerSupport {doesSupportFilterLayers},
}};
```

Výpis kódu 9: Definice PiPL v souboru FilterPlugin.r

Nyní když máme připravený soubor FilterPlugin.r, nastavíme pravidla pro kompilování. Tento soubor se zkompiluje do souboru FilterPlugin.pipl pomocí speciální utility Cnvtpipl.exe ve složce photoshopapi. V okně Solution Explorer klikněte pravým tlačítkem na soubor FilterPlugin.r a vyberte Properties. V sekci General změňte vlastnost Item Type na „Custom Build Tool“ a klikněte na tlačítko Použít. V sekci Custom Build Tool vložte do pole Outputs `$(IntDir)%(Filename).pipl` a do pole Command Line vložte tyto tři příkazy:

```
cl /I photoshopapi\common\includes /I photoshopapi\photoshop /I
↪  photoshopapi\pica_sp /EP /DWIN32=1 /Tc"%(FullPath)" >
↪  "$(IntDir)%(Filename).rr"
photoshopapi\cnvtpipl.exe "$(IntDir)%(Filename).rr"
↪  "$(IntDir)%(Filename).pipl"
del "$(IntDir)%(Filename).rr"
```

Výpis kódu 10: Příkazy pro zkompilování PiPL

Zavřete dialog kliknutím na tlačítko Ok a následně znovu klikněte pravým tlačítkem na soubor FilterPlugin.r a tentokrát vyberte Compile. Tím získáme potřebný PiPL soubor. V souboru FilterPlugin.rc potom najděte řádek s `#include "afxres.rc"` a na následující řádek vložte `#include "FilterPlugin.pipl"`. Podobně v sekci „3 TEXTINCLUDE“ přidejte řetězec `"#include ""FilterPlugin.pipl""\r\n"`.

Nyní máme připravený funkční základ našeho zásuvného modulu. Kliknutím na Build → Build Solution provedete sestavení projektu. Po sestavení projektu se výsledný zásuvný modul automaticky zkopíruje do složky zásuvných modulů aplikace Photoshop. V aplikaci Photoshop můžete spustit zásuvný modul z menu Filter → Plugin Project → FilterPlugin.

Kapitola 7

Zpracování obrazu

První požadavek na obrazová data uděláme v obsluze selectoru Start. Když Photoshop spustí náš zásuvný modul, první věc, kterou musíme udělat, je vyplnit potřebné informace ve struktuře FilterRecord (v naší třídě proměnná m_FilterRecord). Jak bylo popsáno v tabulce 5.1, struktura FilterRecord obsahuje proměnnou filterRect, která popisuje obdélníkovou oblast, která se bude zpracovávat. V případě, že uživatel spustil zásuvný modul s aktivním výběrem, popisuje proměnná filterRect obdélníkové ohraničení daného výběru. V důsledku toho je možné, že zásuvný modul zpracuje více dat, než uživatel označil ve výběru. Photoshop však nakonec automaticky provede maskování a změní tak pouze obraz uvnitř výběru.

Jako první nastavíme proměnnou inRect a outRect na nějakou podmnožinu filterRect, což představuje počáteční část obrazových dat, které chceme zpracovávat. Pokud tyto položky nastavíme správně, v následující fázi Continue budou proměnné inData a outData obsahovat požadovanou část obrazu. Pokud budeme chtít získat další část obrazových dat, opět nastavíme proměnné inRect a outRect na novou podmnožinu fiterRect. Photoshop poté znovu spustí zásuvný modul se selektorem Continue a s novými obrazovými daty. Tento cyklus se opakuje, dokud nenastane chyba nebo dokud se nezpracují všechna data tzn. inRect = outRect = 0 [1]. Modernější a rychlejší způsob výměny dat spočívá v použití funkce AdvanceState().

7.1 Funkce AdvanceState

Od verze 3.0 Photoshop SDK poskytuje funkci AdvanceState. Volání této funkce způsobí, že Photoshop přečte obsah struktury FilterRecord a podle obsahu upraví její proměnné. Tento přístup usnadňuje tvorbu požadavků na data a zároveň snižuje s tím spojenou zátěž. Bez AdvanceState je zásuvný modul opakovaně spouštěn pro každou část obrazových dat. S použitím AdvanceState je možné provést zpracování obrazu během jednoho spuštění se selectorem

Start a obsluha selectoru Continue tak není potřeba.

Je důležité nezapomínat na návratový kód. Pokud vše proběhlo v pořádku, AdvanceState vrátí hodnotu noErr. V opačném případě je potřeba uložit návratovou hodnotu do proměnné result (parametr funkce PluginMain). Pokud AdvanceState nevrátí noErr, není možné ji zavolat znovu ani přistupovat k obrazovým datům, protože proměnné inData a outData budou NULL [1].

Na ukázce níže je vidět použití funkce AdvaceState pro získání všech obrazových dat. Proměnné LoPlane a HiPlane obsahují první a poslední index kanálu, který chceme získat. Pokud například budeme chtít získat všechny barevné kanály v módu CMYK nastavíme první index na 0 a poslední index na 3. Tím říkáme, že chceme barevné kanály s indexy 0, 1, 2 a 3. Můžeme využít i proměnnou planes, která uchovává počet kanálů. Proměnné inputRate/maskRate slouží ke zmenšení obrazu. Hodnota 1 znamená obraz v původní velikosti.

```
int32 PluginDialog::GetData()
{
    int32 error = 0;
    FilterRecord* fr = m_FilterRecord;

    //Vsechny kanaly
    fr->inLoPlane = fr->outLoPlane = 0;
    fr->inHiPlane = fr->outHiPlane = fr->planes - 1;

    //Cely obraz najednou
    fr->inRect = fr->filterRect;
    fr->outRect = fr->filterRect;
    fr->maskRect = fr->filterRect;

    //Puvodni velikost
    fr->inputRate = (int32)1 << 16;
    fr->maskRate = (int32)1 << 16;

    //Volani AdvanceState
    error = (*fr->advanceState)();

    if (error != noErr)
        return error; //Doslo k chybe

    return noErr; //inData a outData nyni obsahuji obrazova data
}
```

Výpis kódu 11: Použití funkce AdvanceState

7.2 Struktura obrazových dat

Po úspěšném volání funkce AdvanceState budou v proměnných inData a outData uložena obrazová data. Tato data ovšem neobsahují pouze barevné pixely ale několik dalších komponent, seřazené přesně v tomto pořadí [1]:

1. **Barevné kanály**
2. **Průhlednost vrstvy**
3. **Maska**
4. **Invertovaná maska**
5. **Alfa kanál**

Které z těchto komponent chceme získat určíme pomocí proměnných LoPlane a HiPlane při vytváření požadavku na data. Pokud například chceme pracovat pouze s barevnými kanály v RGB módu, natavíme LoPlane na 0 a HiPlane na 2. Velikost jednoho barevného kanálu se liší podle obrazového módu a tím je ovlivněna i velikost jednoho pixelu. Například pro 8bitový RGB obraz je velikost jednoho barevného kanálu 1 byte a velikost jednoho pixelu je tedy 3 byty. Přehled nejčastějších módů a velikost pixelu v daném módu je popsán v tabulce níže.

Obrazový mód	Počet bytů
plugInModeGrayScale	1
plugInModeRGBColor	3
plugInModeCMYKColor	4
plugInModeLabColor	3
plugInModeGray16	2
plugInModeRGB48	6
plugInModeCMYK64	8
plugInModeLab48	6

Tabulka 7.1: Obrazové módy a počet bytů na pixel [1]

7.3 Úprava jednotlivých pixelů

Obrazová data jsou v proměnných inData a outData uložena prokládaně. Například pro 8bitový RGB obraz budou data uložena takto: R G B R G B R …, kde R, G a B představuje 1 byte barevného kanálu pro daný pixel. Podobně 16bitový RGB obraz bude uložen: R R G G B B R R …, kde R G

a B opět představuje 1byte barevného kanálu [1]. Pro některé algoritmy je vhodné si data přeuspořádat tak, aby byly uloženy nejprve všechny červené pixely, poté zelené pixely a nakonec modré pixely. Nakonec je ale potřeba uložit data do proměnné outData původním způsobem, tedy prokládaně.

Zpracování obrazu po pixelech typicky probíhá tak, že se z proměnné inData nejprve načtou všechny barevné kanály, které tvoří jeden pixel. Tento pixel se následně upraví a jeho barevné kanály se nakonec uloží do proměnné outData.

Na ukázce níže je funkce, která provádí invertování obrazu. Na tomto příkladu je vidět, jakým způsobem se provádí zpracování obrazu po jednotlivých pixelech. Pro práci s pixely je k dispozici datový typ Color8/Color16/Color32. Ve skutečnosti se jedná o pole 4 čísel, které sdružuje jednotlivé barevné kanály. Kanály musí být v poli uloženy tímto způsobem: 0RGB, CMYK, 0LAB, 000Gray. Pokud tedy obrazový mód nevyužije všechny 4 položky, musí být zleva zarovnány nulami. V této ukázce není nutné používat proměnnou `Color8 pixel`, jedná se pouze o demonstraci použití.

```
void PluginDialog::InvertRGB8(uint8* inputData, uint8* outputData,
                              size_t width, size_t height)
{
    FilterRecord* fr = m_FilterRecord;
    size_t pixels = width * height;
    constexpr int RED = 1, GREEN = 2, BLUE = 3;
    constexpr uint8 maxValue = 255;

    for (size_t i = 0; i < pixels; i++)
    {
        Color8 pixel = {0, inputData[3*i], inputData[3*i + 1],
                        inputData[3*i + 2]};
        pixel[RED] = maxValue - pixel[RED];
        pixel[GREEN] = maxValue - pixel[GREEN];
        pixel[BLUE] = maxValue - pixel[BLUE];

        outputData[3*i] = pixel[RED];
        outputData[3*i + 1] = pixel[GREEN];
        outputData[3*i + 2] = pixel[BLUE];

        if(i % width == 0)
            fr->progressProc((int32)(i/width), (int32)height);
    }
}
```

Výpis kódu 12: Invertování 8bitového RGB obrazu

Všimněte si volání funkce progressProc na konci cyklu. Jak bylo popsáno v tabulce 5.1, funkce progressProc zobrazuje průběh u dlouhých operací. Tato funkce přijímá dva parametry: počet dokončených položek a celkový počet položek. Je dobré tuto funkci volat dostatečně často, aby se plynule zobrazoval průběh, ale ne příliš často, aby tato funkce nezpomalovala cyklus. Ideálním kompromisem je zavolat tuto funkci po dokončení řádky pixelů. Volání funkce progressProc neznamená nutně zobrazení průběhu. Pokud filtrování probíhá dostatečně rychle, Photoshop se rozhodne průběh nezobrazovat.

7.4 Zpracování 16bitového obrazu

Funkce pro invertování 16bitového RGB obrazu bude vypadat velmi podobně. Rozdíl bude v datových typech u parametrů inputData a outputData, které se změní na uint16*. Proměnná pixel bude mít datový typ Color16 a proměnná maxValue bude mít datový typ uint16 a hodnotu 32768.

Při práci s 16bitovým obrazem je jedna důležitá věc, kterou je potřeba mít na paměti. To, čemu Photoshop říká 16bitové barvy nejsou skutečně 16bitové barvy. 16 bitů na kanál by znamenalo rozsah hodnot 0 až 65535. Photoshop však používá omezenější rozsah, a proto výstižnější název by byl 15bitové barvy. Ani to by ale nebylo přesné. 15 bitů na kanál by znamenalo rozsah hodnot 0 až 32767. Skutečný rozsah je ovšem 0 až 32768, tedy 15 bitů + 1 extra hodnota. Photoshop tedy skutečně používá 16 bitů, pouze nevyužívá celý rozsah.

Důvodem pro omezený rozsah jsou jednak historické důvody a také to, že maximální hodnota 32768 má prostřední hodnotu. Díky tomu je možné nahradit některé operace násobení a dělení bitovými posuny a urychlit tak zpracování obrazu [6].

7.5 Dokončení zásuvného modulu

Ve Visual Studiu v okně Resource View 2x klikněte na dialog IDD_DIALOG1 aby se zobrazil náhled dialogu. Nyní pravým tlačítkem myši klikněte na tlačítko OK a vyberte „Add Event Handler…“. Položku Class list nastavte na „PluginDialog“ a položku Message type na „BN_CLICKED“. Jméno funkce ponechte beze změny a potvrďte kliknutím na OK.

Visual Studio nyní vygenerovalo funkci, která se zavolá, když uživatel klikne na tlačítko OK. V této funkci provedeme zpracování obrazu. Nejprve od Photoshopu získáme obrazová data a následně podle obrazového módu zavoláme příslušnou funkci pro invertování obrazu.

```
void PluginDialog::OnBnClickedOk()
{
    FilterRecord* fr = m_FilterRecord;
    int32 error = GetData();

    if (error == noErr)
    {
        int16 width, height;
        width = fr->filterRect.right - fr->filterRect.left;
        height = fr->filterRect.bottom - fr->filterRect.top;
        if(fr->imageMode == plugInModeRGBColor)
            InvertRGB8((uint8*)fr->inData, (uint8*)fr->outData,
                       width, height);
        else if(fr->imageMode == plugInModeRGB48)
            InvertRGB16((uint16*)fr->inData, (uint16*)fr->outData,
                       width, height);
    }

    CDialog::OnOK();
}
```

Výpis kódu 13: Obsluha kliknutí na tlačítko OK

V ukázce je použita funkce GetData, což je funkce z výpisu kódu 11, funkce InvertRGB8 z výpisu kódu 12 a funkce InvertRGB16. Funkce InvertRGB16 je podobná funkci InvertRGB8 pouze místo datových typů uint8 jsou použity datové typy uint16, místo Color8 je použito Color16 a proměnná maxValue má hodnotu 32768.

V tuto chvíli máme připravený funkční zásuvný modul, který je možné spustit uvnitř Photoshopu z menu Filter → Plugin Project → FilterPlugin. Po kliknutí na tlačítko OK provede zásuvný modul invertování RGB obrazu s 8 nebo 16bitovou barevnou hloubkou.

Závěr

Cílem této práce bylo zdokumentovat postup vytváření zásuvných modulů typu filter pro aplikaci Adobe Photoshop včetně uživatelského rozhraní a použití vhodných nástrojů pro vývoj. V úvodu práce byly popsány zásuvné moduly, jaké existují kategorie a k čemu slouží. Následující kapitola představila různé metody vývoje zásuvných modulů typu filter. Z této kapitoly vyšlo najevo, že nejuniverzálnějším způsobem je využití oficiálního SDK a programovacího jazyka C++. V další kapitole bylo vysvětleno fungování zásuvného modulu z programátorského hlediska. Byly také vysvětleny jednotlivé fáze běhu zásuvného modulu a co se od každé fáze očekává.

S těmito teoretickými znalostmi přišlo na řadu vytvoření projektu pro vývoj zásuvného modulu. Vytvoření projektu pro zásuvný modul se ukázalo být poměrně komplikované, a proto bylo v této kapitole podrobně popsáno, jak nakonfigurovat aplikaci Visual Studio i samotný projekt. V této kapitole bylo také popsáno, jak vytvořit PiPL metadata a jak použít externí nástroj Cnvtpipl k jejich zkompilování.

V závěru této práce bylo vysvětleno, jakým způsobem jsou organizována obrazová data a jakým způsobem je zpracovat. Byly také představeny dva možné způsoby komunikace a výměny dat s Photoshopem.

Všechny tyto informace nakonec vyústily v zásuvný modul, jehož zkompilovaná podoba i zdrojové kódy jsou k dispozici na přiloženém CD. Tento zásuvný modul umožňuje mapování barevných komponent v modelu HSB pomocí kubických Bézierových křivek.

Bibliografie

1. KAS, Thomas. How to Write a Photoshop Plug-In, Part 1. *MacTech | The journal of Apple technology.* [online]. 1999 [cit. 2019-04-18]. Dostupné z: `http://preserve.mactech.com/articles/mactech/Vol.15/15.04/PhotoshopPlug-InsPart1/index.html`.
2. ADOBE SYSTEMS INCORPORATED. *Adobe Photoshop SDK* [zip archiv]. 2018 [cit. 2019-04-18]. Dostupné z: `https://console.adobe.io/downloads/ps`.
3. FLOURET, Enrique. Programming Photoshop Plugins. *Photoshop Roadmap* [online]. 2006 [cit. 2019-04-18]. Dostupné z: `https://photoshoproadmap.com/programming-photoshop-plugins`.
4. FILTER FORGE, INC. *Filter Forge*. Verze 8.0. Dostupné také z: `https://filterforge.com/`.
5. NERVOUSCHIMP. *checked out the filter forge node editor?* [online]. 2009 [cit. 2019-04-18]. Dostupné z: `https://www.cheetah3d.com/forum/index.php?threads/4793/`.
6. BONAIREGUY. *16-bit or 15-bit+1?* [online]. 2011 [cit. 2019-05-10]. Dostupné z: `https://forums.adobe.com/thread/792212`.

Příloha A

Seznam použitých zkratek

HSB Hue Saturation Brightness

PiPL Plug-In Property List

SDK Software Development Kit

GUI Graphical User Interface

MFC Microsoft Foundation Classes

API Application Programming Interface

PICA Plug-In Component Architecture

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `8bf` ..... adresář se zkompilovanou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `src` ..... zdrojová forma práce ve formátu LaTeX
- `examples` ..... ukázky použití