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

EVALUACE METOD VYHLEDÁVÁNÍ KLÍČOVÝCH BODŮ

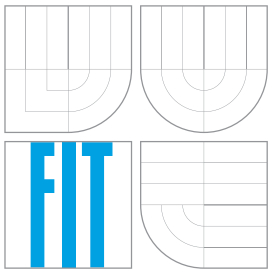
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

JAROSLAV KORDULA

BRNO 2007

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

EVALUACE METOD VYHLEDÁVÁNÍ KLÍČOVÝCH BODŮ

EVALUATION OF KEY POINT DETECTION METHODS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — JAROSLAV KORDULA
AUTHOR

VEDOUCÍ PRÁCE — doc. Dr. Ing. PAVEL ZEMČÍK
SUPERVISOR

BRNO 2007

Zadání diplomové práce

Řešitel:	**Kordula Jaroslav**
Obor:	Výpočetní technika a informatika
Téma:	**Evaluace metod vyhledávání klíčových bodů**
Kategorie:	Počítačová grafika

Pokyny:

1. Seznamte se s problematikou vyhledávání klíčových bodů v obraze, aplikacemi využívajícími klíčové body a dostupnou literaturou na toto téma.
2. Navrhněte vhodnou metodiku evaluace úspěšnosti vyhledávání klíčových bodů v množině obrazů s ohledem na invariantnost vzhledem ke změně projekce, měřítka a jasových poměrů.
3. Vyhodnoťte možnosti implementace evaluačního software podle předchozího bodu a to tak, aby pokud možno vyhodnocení bylo použitelné v „iteračních postupech“ návrhu/učení algoritmů vyhledání klíčových bodů.
4. Implementuje výše popsaný software, demonstrujte jeho funkčnost na vhodném příkladě a vyhodnoťte.
5. Diskutujte dosažené výsledky a další možnosti.

Implementační jazyk:	C/C++/C#
Operační systém:	Windows

Vedoucí:	Zemčík Pavel, doc. Dr. Ing., UPGM FIT VUT
Datum zadání:	1. listopadu 2006
Datum odevzdání:	22. května 2007

Licenční smlouva

Licenční smlouva je uložena v archivu Fakulty informačních technololgí Vysokého učení technického v Brně.

Abstrakt

Cílem práce je seznámení se s programovacím prostředím Microsoft .NET Framework a problematikou vyhledávání klíčových bodů v obraze. Na základě těchto znalostí je požadován návrh metody evaluace používaných detektorů a následná implementace konzolové aplikace umožňující snadné vyhodnocení výstupních dat aplikované metody vyhledávání klíčových bodů v obraze v iteračních postupech návrhu/učení algoritmů za použití navržené metody evaluace. Klíčové body jsou vyhledány v několika bitmapách, které zachycují tutéž scénu v různých směrech pohledu. Zároveň je požadováno vytvoření grafického uživatelského prostředí umožňujícího snadné nastavení podmínek pro evaluaci.

Klíčová slova

vyhledávání vlastností obrazu, vyhledávání klíčových bodů, vyhledávání hran, .NET Framework, GDI+, C#, evaluace

Abstract

The goal of this thesis is to get familiarized with software component Microsoft .NET Framework and the problems of methods of interest point detection. On the basis of this knowledge, it is required to develop a method for the evaluation of used detectors and then implement a console application that is simply able to evaluate the results of interest point detection methods. Such evaluation is important in the process of development of the algorithms for the detection using projected method of evaluation. The interest points are searched in several images which represent the same scene from different angles of view. The requirements also inculde creation of graphic user interface that allows an easy way to setup the evaluation conditions.

Keywords

feature detection, interest point detection, edge detection, .NET Framework, GDI+, evaluation

Citace

Jaroslav Kordula: Evaluace metod vyhledávání klíčových bodů, diplomová práce, Brno, FIT VUT v Brně, 2007

Evaluace metod vyhledávání klíčových bodů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana doc. Dr. Ing. Pavla Zemčíka.

.....
Jaroslav Kordula
15. května 2007

Obsah

Kapitola 1

Úvod

V dnešní době by se jen těžko hledal obor, který by alespoň pro určité své odvětví nepoužíval počítačovou techniku. S rostoucím výkonem a schopnostmi získávají počítače stále se rozrůstající pole působnosti, díky čemuž rapidně roste zájem o toto odvětví. Zároveň to umožňuje vyvíjet aplikace a algoritmy, které by v dřívějších dobách nebylo možné realizovat vzhledem k jejich složitosti a náročnosti. Také díky tomuto faktu existuje stále rostoucí snaha o digitalizaci jakékoli získané informace, což napomáhá procesu zpracování počítačem. Jedním z oborů uplatňujícím se v současnosti je i zpracování obrazu. Nejedná se pouze o obraz statický, ale také o videosekvence. Toto odvětví má širokou škálu uplatnění. Za zmínku stojí algoritmy pro detekci a sledování objektů a částí těla, detekce obličeje, počítačové vidění. Použití nalezne v průmyslových a dopravních aplikacích, biomedicíně a dalších oborech. Jednou z podskupin zpracování obrazu je i detekce klíčových bodů v obraze.

Vyhledávání, neboli detekce klíčových bodů v obraze patří pod obecnější skupinu – vyhledávání vlastností v obraze. Pod tímto pojmem si lze představit metody pro nalezení a vyhodnocení takové informace v obraze, která je pro dané zkoumání relevantní. V současné době existuje celá řada takovýchto osvědčených algoritmů, ale i přes to přetrvává snaha o vytvoření efektivnějších a dokonalejších algoritmů, které by lépe splňovaly požadavky těchto metod.

Úspěšnost metod a tedy i samotných detektorů závisí na určení optimálních parametrů pro použitý algoritmus. Proto vznikl požadavek na vytvoření aplikace, která s pomocí uživatele vyhodnotí výstupní data použité metody a určí její úspěšnost, což vede k usnadnění nalezení těchto parametrů.

Cílem práce je tedy seznámit se s problematikou vyhledávání klíčových bodů v obraze a navrhnout vhodnou metodu pro hodnocení používaných detektorů, která je následně použita při vývoji prostředku sloužícího k evaulaci těchto vyhledávačů.

Kapitola 2 – *Microsoft .NET Framework* popisuje architekturu platformy, její součásti Windows Forms, která slouží pro práci s formuláři, a Graphic Device Interface Plus, jež se používá pro práci s grafikou. Dále jsou zde popsány programovací jazyky, které lze v tomto prostředí použít, podpůrné nástroje a jedny z nejznámějších portací komponenty do jiných operačních systémů.

Kapitola 3 – *Vyhledávání klíčových bodů v obraze* osahuje popis vybraných detektorů klíčových bodů, požadavky na tyto detektory, jejich omezení a metody vyhodnocování úspěšnosti vyhledavačů.

V kapitole 4 – *Požadavky na evaluaci detektorů klíčových bodů v obraze* jsou popsány požadavky, které byly kladeny při návrhu metody evaluace detektorů klíčových bodů v ob-

raze. Zároveň jsou uvedeny implementační požadavky konzolové aplikace pro iterační postup vyhodnocování a grafického uživatelského rozhraní aplikace určené k zadání dodatečných údajů ze strany uživatele, které jsou potřebné pro samotnou evaluaci.

Kapitola 5 – *Struktura projektu a jeho implementace* zahrnuje postup při návrhu a implementaci modulu pro vyhodnocování detektorů, aplikace s grafickým uživatelským rozhraním a konzolové aplikace.

V závěrečné kapitole jsou shrnuty poznatky získané v průběhu tvorby diplomové práce a popsán přínos vytvořené aplikace spolu s možnými budoucími rozšířeními.

Kapitola 2

Microsoft .NET Framework

V této kapitole jsou shrnuty ty vlastnosti platformy Microsoft .NET Framework, jejichž znalost byla nutná při vývoji software. Popsána je jak samotná architektura platformy, tak součásti Windows Forms a Graphic Device Interface Plus, které byly využity při implementaci aplikace s grafickým uživatelským rozhraním. Kapitola také zahrnuje popis podpůrných nástrojů prostředí, které výrazně usnadňují a zefektivňují vývoj aplikací s touto softwarovou komponentou. Obsah této kapitoly je založen na publikovaných informacích [1] [3] [6] [8] [9] [10] [12].

2.1 Architektura

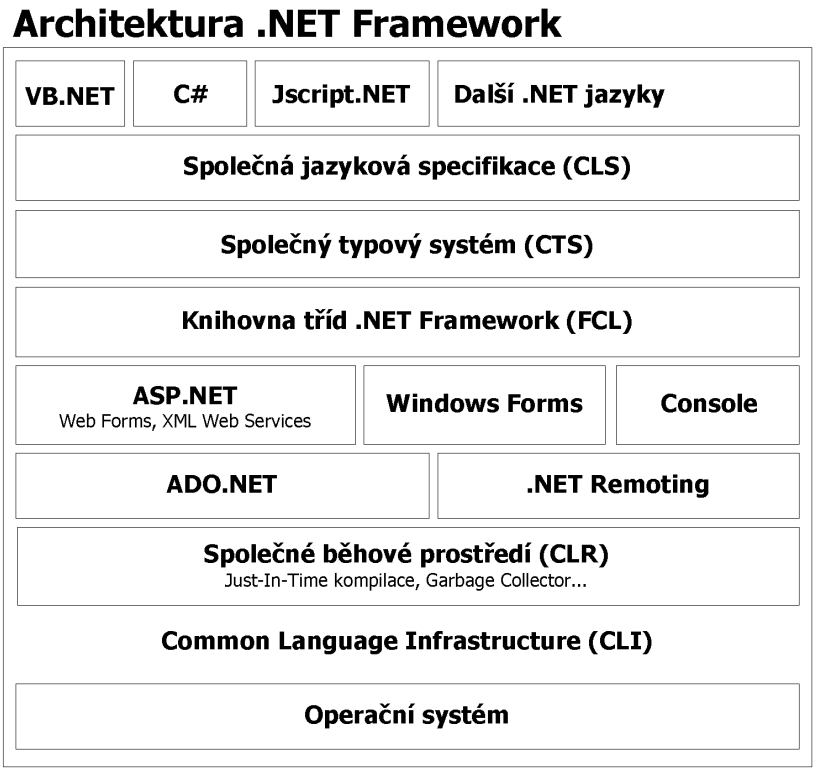
Microsoft .NET Framework je softwarová platforma, která poskytuje bohaté prostředky programátorům pro to, jak poměrně jednoduše vyvíjet nové aplikace. K tomu napomáhá také celá řada zahrnutých již hotových řešeních, které lze kombinovat s vlastním kódem programátora a dosáhnout tak efektivního vývoje požadované aplikace. Primárně je tato komponenta určena pro operační systém Microsoft Windows, nicméně existují verze i pro jiné operační systémy, o kterých se zmíníme později. Mezi hlavními důvody, proč Microsoft začal tuto platformu vyvíjet, byly především vysoká chybovost aplikací při práci s pamětí a konverzi datových typů, problémy s verzemi knihoven a nejednoduchá komunikace mezi knihovnami napsanými v odlišných programovacích jazycích díky jejich nekompatibilitě. Zároveň při snaze zachování zpětné kompatibility a přidávání nových vlastností stávajících vývojových nástrojů a programovacích jazyků se tyto vývojové prostředky stávaly více komplikovanými. Také díky tomu se společnost Microsoft rozhodla vyvinout novou množinu programovacích jazyků, prostředí a vývojových nástrojů.

Při návrhu platformy byl tedy mimo jiné kladen důraz na možnost nabídnout konzistentní objektově orientované programovací prostředí, zachování zpětné kompatibility, bezpečný běh programu, eliminaci výkonnostních problémů skriptovacích nebo interpretovaných prostředí a celkové snížení úsilí při vývoji nových aplikací. To s sebou přineslo následující výhody .NET platformy:

- Objektově orientované programování
- Objektový návrh založený na základní knihovně tříd
- Nezávislost programovacích jazyků
- Efektivní přístup k datům

- Sdílení kódu mezi aplikacemi
- Zdokonalenou bezpečnost

Dvě hlavní části Microsoft .NET Framewrok jsou Common Language Runtime (CLR), známá také jako společné běhové prostředí, a knihovna tříd .NET Framewrok (Framewrok Class Library).



Obrázek 2.1: Architektura .NET Framework.

Společné běhové prostředí (Common Language Runtime)

Common Language Runtime je nejdůležitější součást celé .NET platformy. Jedná se o implementaci specifikace Common Language Infrastructure (CLI), což je specifikace, která popisuje spustitelný kód a běhové prostředí. Díky ní lze použít několik programovacích jazyků na různých platformách bez potřeby přeložení zdrojových kódů pro danou architekturu. Kód, jehož běh je řízen tímto prostředím, se běžně označuje jako „řízený kód“ (managed code).

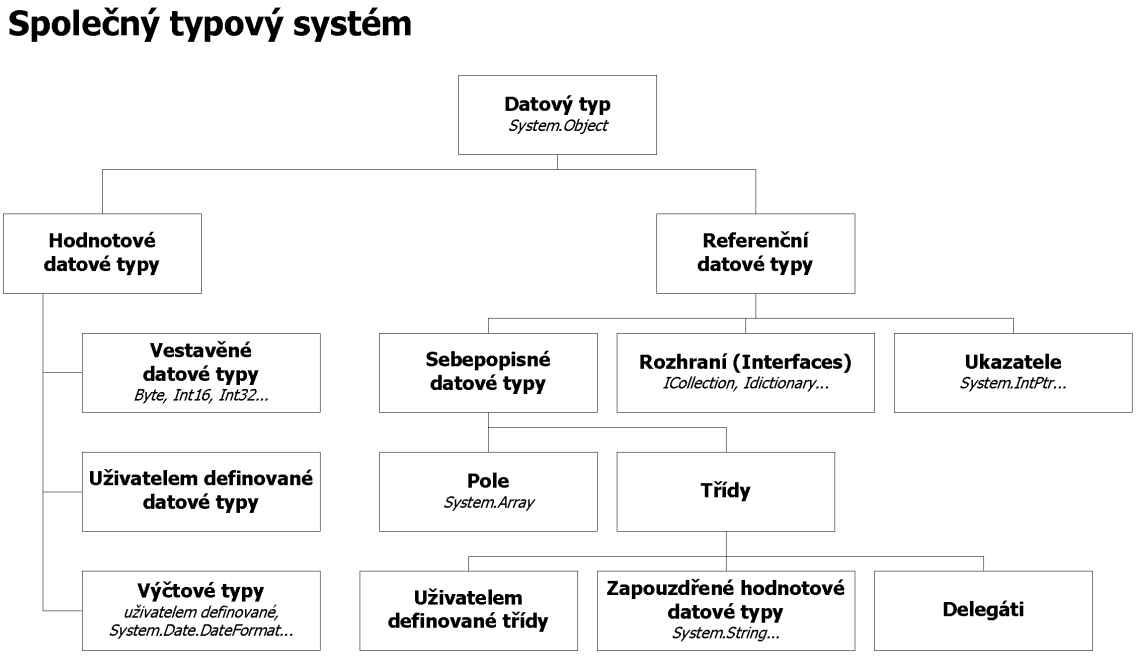
Prostředí se sestává z pěti hlavních částí.

Společný typový systém (Common Type System, CTS)

Tento systém je používaný každým programovacím jazykem postaveným na .NET platformě. Definuje množinu datových typů a operací, které lze použít v rámci odlišných jazykových syntaxí, a zároveň hierarchii datových typů, která je vidět na obrázku 2.2. Jazyk

tedy není omezen, co se týče syntaxe, ale minimální množinou datových typů definovaných CTS. Zajišťuje možnost interakce objektů různých programovacích jazyků z této skupiny. Každý datový typ definovaný CTS musí obsahovat žádný nebo více členů. Mezi tyto členy patří:

- **Pole (Field).** Jedná se o proměnnou, která je součástí stavu objektu. Má vlastní jméno a typ.
- **Metoda (Method).** Funkce, která zajišťuje provedení jisté operace nad objektem (jako změna jeho stavu). U metody je definovaný název, signatura a modifikátory. Signatura specifikuje, jakým způsobem je metoda volána, počet a pořadí parametrů, jejich typ a návratovou hodnotu metody.
- **Vlastnost (Property).** Navenek se chová jako již zmíněný člen – pole, ale tento člen umožňuje validaci parametrů a stavu objektu ještě před samotným přístupem. Pomocí tohoto členu lze také vytvářet „read-only“ nebo „write-only“ pole.
- **Událost (Event).** Mechanismus sloužící ke vzájemné komunikaci objektů.



Obrázek 2.2: Struktura společného typového systému.

Společná jazyková specifikace (Common Language Specification, CLS)

Minimální množina základních pravidel, které by měl splňovat každý kompiler zaměřen na .NET Framework.

Common Intermediate Language (CIL, IL)

Dříve také známý jako Microsoft Intermediate Language (MSIL). Jedná se o programovací jazyk na nejnižší úrovni, který je ještě čitelný člověkem, nezávislý na cílovém CPU. Spolu s tímto mezikódem kompiler produkuje také metadata, která popisují datové typy, jejich definice a další informace. Důležité vlastnosti tohoto jazyka jsou:

- **Objektová orientace, používaní rozhraní**
- **Silné rozlišování mezi hodnotami a referencemi** Tak jako jakýkoli jiný programovací jazyk, CIL nabízí skupinu základních datových typů. Ovšem rozdíl oproti ostatním jazykům je v tom, že CIL silně rozlišuje mezi hodnotovými typy, tedy typy, jejichž proměnné ukládají přímo data, nebo referenčními typy, u kterých je uchovávána adresa paměti, kde se data nacházejí.
- **Silná datová kontrola** Toto je jeden z velmi důležitých aspektů tohoto jazyka. Vždy je přesně určen daný datový typ pro danou proměnnou a není běžně dovoleno, aby některá operace vracela nejednoznačný datový typ. Tento fakt má zajisté negativní vliv na výkonnost aplikace, nicméně je kompenzován výhodami, které vznikají. Patří mezi ně například bezpečnost, garbage collection, mezijazyková součinnost a další.
- **Správa chyb prostřednictvím výjimek** Architektura výjimek zajišťuje, že bezprostředně po vyskytnutí chyby lze tuto chybu zpracovat rutinou k tomu navrženou. Výhodou výjimek je to, že s sebou nesou informace, z kterých lze určit, kde se vyskytl problém, a také to, že je zde podporováno mezijazykové zpracování.
- **Používání atributů**

Just-in-Time Compiler (JIT)

Tento kompiler slouží pro převod mezikódu (CIL) na nativní kód. Jeho výhoda spočívá v tom, že ke kompilaci zdrojového kódu dochází až za běhu aplikace. Nemusí být tedy zbytečně kompilovány ty části, které se při běhu aplikace vůbec neprovedou. Kompilace mezikódu do nativního kódu je mnohem rychlejší než z kódu zdrojového, přesto je zde nevýhoda, která spočívá v prodlevě při prvotním spuštění aplikace, neboť se provádí jisté optimalizační procesy optimalizace, jako například optimalizace pro dané CPU.

Virtual Execution System (VES)

Tato část zajišťuje mimo jiné prostředí pro spouštění řízeného kódu, přímou podporu pro množinu vestavěných datových typů, množinu konstrukcí na řízení toku, model pro správu výjimek.

Common Language Runtime řídí samotný kód a poskytuje prostředky pro jednodušší vývoj. Princip je podobný jako u platformy Java firmy Sun Microsystems. Zdrojový kód není přeložen přímo do binárního kódu, ale existuje zde ještě mezivrstva, takzvaný mezikód, v tomto případě již zmíněný CIL (Common Intermediate Language). To činí takto vyvíjené aplikace přenositelné v rámci existence implementace .NET Framework pro požadovanou platformu.

Existence tohoto prostředí s sebou přináší jisté výhody, ale i nevýhody. Mezi výhody patří například možnost komunikace objektů implementovaných v odlišných programovacích jazycích, mezijazykové zachytávání výjimek, silná typová kontrola, zdokonalená

bezpečnost, existence garbage collectoru, tedy automatické uvolňování objektů z paměti, pokud již nejsou potřeba.

Knihovna tříd Microsoft .NET Framewrok (Framework Class Library, FCL)

Jednou z největších výhod při psaní řízeného kódu z pohledu vývojáře je možnost využít .NET Framework knihovnu tříd. Jedná se o masivní kolekci tříd, které jsou velmi intuitivní a jednoduché na použití. Na druhou stranu existuje celkem velké množství vlastností systému Windows, které nejsou přes tuto základní knihovnu tříd přístupné, a je tedy nutné použít API funkce. Knihovnu tříd lze rozdělit do těchto částí:

- Základní vlastnosti poskytnuté IL
- Kontroly a podpora Windows GUI
- Webové formuláře
- Přístup k datům
- Přístup k adresářové struktuře
- Sítě a web
- .NET atributy a reflexe
- Přístup ke specifickým vlastnostem Windows OS
- Součinnost s COM

Knihovna tříd obsahuje tisíce datových typů. Aby nedocházelo ke konfliktům mezi třídami a aby byla zavedena jistá hierarchie, existují zde takzvané jmenné prostory. Dá se říci, že jmenný prostor seskupuje množinu datových typů, které spolu souvisí. Existuje zde také možnost zanořování jmenných prostorů.

Typy aplikací, jejichž vývoj je prostřednictvím tohoto systému možno vyvíjet:

Webové služby (Web Services). Jedná se o komponenty, které zpřístupňují implementovanou funkcionalitu prostředníctvím internetu.

Webové formuláře (Web Forms). Aplikace založené na HTML. Tyto webové aplikace většinou kooperují s databázovým serverem nebo využívají webové služby a získaná data zobrazují prostřednictvím prohlížeče.

Formuláře Windows (Windows Forms). Takovýto druh aplikací využívá výhody grafického uživatelského rozhraní. Při své činnosti má k dispozici systémové prostředky a také může využívat databáze a webové služby k získání dat.

Konzolové aplikace. Aplikace, které nepotřebují bohaté grafické rozhraní.

Služby Windows (Windows Services).

Knihovna komponent (Component Library). Představuje samostatné komponenty, které lze použít v ostatních zmíněných typech aplikací.

Jemnný prostor	**Obsah**
System	Všechny základní datové typy
System.Collections	Správa kolekcí objektů. Zahrnuje známé typy kolekcí jako zásobníky, fronty, hash-tables...
System.Diagnostics	Ladění aplikací.
System.Drawing	Práce s 2D grafikou. Typicky se používá při práci s formuláři Windows a při vytváření obrázků.
System.EnterpriseServices	Správa transakcí, just-in-time aktivace, bezpečnosti a další.
System.Globalization	Národní jazyková podpora jako porovnávání řetězců, formátování, kalendáře...
System.IO	Práce se souborovým systémem, vstup-výstupní operace.
System.Net	Síťová komunikace.
System.Reflection	Přístup k metadatům...
System.Resources	Správa externích zdrojů.
System.Runtime.InteropServices	Umožňuje řízenému kódu přístup k neřízeným prostředkům systému (COM komponenty, funkce v DLL knihovnách Win32 API).
System.Runtime.Remoting	Vzdálený přístup k datovým typům.
System.Runtime.Serialization	Povoluje instancím objektů, aby měly schopnost přetrvat a znovu se vytvořit ze streamů.
System.Security	Ochrana a bezpečnost dat a zdrojů.
System.Text	Práce s textem v různých kódováních (ASCII, Unicode).
System.Threading	Představuje asynchronní operace a synchronizovaný přístup ke zdrojům.
System.Xml	Práce s XML schématy a daty.

Tabulka 2.1: Výčet vybraných jmenných prostorů.

Garbage collection

Z dřívějších dob jsou známy na Windows platformě dvě techniky správy paměti:

- O správu paměti se stará sama aplikace
- Objekty udržují počet referencí

První technika je využívána především nízkoúrovňovými programovacími jazyky jako např. C++. Výhodou je efektivita a fakt, že používané zdroje nejsou zabírány déle, než je opravdu nutno. Největší nevýhoda je ta, že veškerou tuto činnost mají na starosti sami programátoři, takže často dochází ke špatnému uvolňování zdrojů, což může vést k problémům.

Udržování počtu referencí je založeno na tom, že samy zdroje obsahují informaci o tom, kolik klientů je využívá. Nicméně aktuálnost této informace opět spočívá na dobrém chování samotných klientů, tedy programátorů.

Jak již bylo zmíněno, v .NET Framework se o správu paměti stará Garbage collector. Tato služba zajišťuje uvolňování paměti. Princip je takový, že paměť je dynamicky alokována na hromadu (heap) a v případě, že je hromada pro daný proces téměř plná, garbage collector je zavolán. Aby garbage collector dokázal identifikovat objekty, které již nejsou potřeba, zjišťuje, zda v dané oblasti programu existují objekty, na které již neexistuje reference. Důležitá je informace, že garbage collector není deterministický. Nelze tedy s jistotou říci, kdy bude zavolán. To řídí již popisované Common Language Runtime.

Bezpečnost

.NET Framework výborně doplňuje mechanizmus bezpečnosti, který je zahrnut v Microsoft Windows – bezpečnost založená na rolích. Nabízí totiž bezpečnost na úrovni kódu.

Tato metoda je založena na skutečnosti, kdo je vlastníkem procesu, pod kterým daný kód běží. Na druhou stranu bezpečnost na úrovni kódu je založena na tom, co ve skutečnosti kód provádí a jak moc je věrohodný. Díky silné typové kontrole CIL, je společné běhové prostředí schopné prověřit kód před tím, než je skutečně vykonán. Zároveň lze dopředu určit, jaká bezpečnostní práva bude kód ke svému běhu potřebovat. Tímto mechanismem lze zajistit bezpečnost s větší granularitou, než je tomu u bezpečnosti založené na rolích.

Domény aplikace

Domény aplikace jsou jedním z důležitých novinek v .NET Framework. Slouží k možnosti souběhu instancí aplikace, které potřebují být od sebe navzájem oddělené, ale zároveň musí být schopny spolu komunikovat.

Jedním z řešení v dřívějších dobách bylo sdílení společného procesu. V tomto případě ovšem pád jedné instance mohl negativně ovlivnit druhou. Další možný způsob spočíval v izolování instancí do samostatných procesů, což se při větším počtu instancí mohlo projevit na výkonu. V systému Windows jsou procesy odděleny adresovými prostory. Každý proces má k dispozici 4GB (pro 32-bitové systémy) virtuální paměti, která slouží pro ukládání dat a spustitelného kódu. Tato virtuální paměť je namapována do fyzické paměti nebo paměti disku, přičemž je zajištěno, že žádné dva bloky virtuální paměti nejsou namapovány do stejného paměťového prostoru. Tímto je zabezpečeno, že proces nemůže poškodit nic mimo jeho prostor. Zároveň lze v systému Windows brát proces jako jednotku, které jsou přiřazena vlastní práva. Komunikace mezi takovými procesy je výkonnostně náročná. Tuto

negativní vlastnost lze obejít sdíleným adresovým prostorem, ovšem na úkor toho, že při selhání jednoho procesu dojde k selhání procesů všech, které sdílí tento adresový prostor.

Tyto problémy se snaží řešit právě aplikační domény. Jsou navrženy jako oddělující komponenty, aniž by způsobovaly výkonnostní problémy při přenosu dat mezi procesy. Myšlenka spočívá v tom, že každý proces je rozdělen do několika aplikačních domén. Každá aplikační doména odpovídá jedné aplikaci. Tím probíhá snadné sdílení dat mezi instancemi aplikace.

2.2 Windows Forms a Graphics Device Interface Plus

Windows Forms

V dřívějších dobách byla při vytváření aplikací na platformě Windows k dispozici pouze sada Windows aplikačních rozhraních (Windows API). Tento způsob vývoje byl ovšem fádní a náchylný na chyby. Proto vznikla nutnost vyvinout určitou abstrakci nad těmito API, která by vývoj usnadnila. Později byly vytvořeny různé systémy, například MFC (Microsoft Foundation Class ibrary) nebo ATL (Acive Template Library), které sloužily jako kostra nebo šablona pro vyvíjenou aplikaci. Při implementaci nestandardních požadavků bylo ovšem občas nutné opět sáhnout k použití klasických Windows API, čímž docházelo k jisté nekonzistenci v kódu.

Windows Forms zajišťují jednotný model pro vývoj aplikací pod systémem Windows. Co se týče úrovně abstrakce, jsou Windows Forms srovnatelné s klasickým Windows API. Jedná se o jednu ze základních komponent .NET Framework. Umožňuje vývojářům vytvářet schopné interaktivní aplikace, které lze spustit na jakékoli Windows platformě s nainstalovaným .NET Framework. Tato komponenta výrazně usnadňuje práci, snižuje úsilí a zkracuje dobu vývoje při vytváření aplikace. Windows Forms se skládá ze skupiny tříd vytvořených v samotném .NET Framework. Tyto třídy zajišťují objektově orientovanou obálku nad vytvářením a údržbou formulářů ve Windows, dialogových oken, uživatelských vstupů, rozšířeným grafickým výstupem a dalším. Díky již zmíněné objektové orientaci této komponenty lze velice snadno vytvářet vlastní modifikované formuláře nebo uživatelské kontroly, a to pomocí dědění objektů standardních.

Grafické uživatelské rozhraní

Existuje několik možností, jak lze přistoupit k tvorbě grafického uživatelského rozhraní v závislosti na orientaci aplikace a požadovanou funkcionalitu. V dnešní době existují tři typy rozhraní:

SDI (Single document interface). Tento způsob je založen na tom, že jednotlivé celky grafického uživatelského rozhraní aplikace jsou samostatná okna. Tato okna nemají společné „rodičovské“ okno. Pokud například aplikace umožňuje editaci více dokumentů, může uživatel nabýt dojem, že je spuštěno více instancí aplikace.

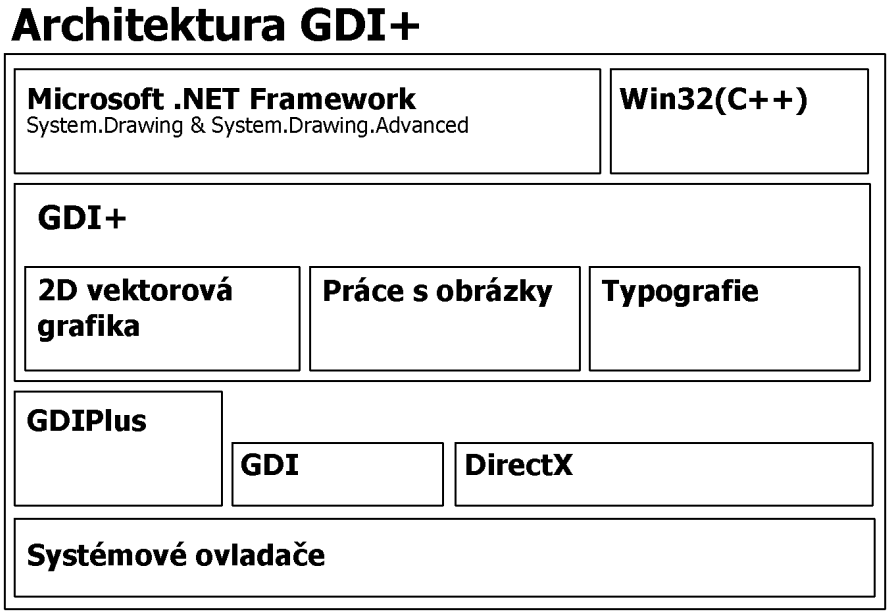
MDI (Multiple document interface). Apliakce s tímto grafickým rozhraním mají tu výhodu, že mají společné rodičovské okno, a dílčí okna aplikace jsou tak uzavřena v jakémsi celku. Mírná nevýhoda tohoto způsobu spočívá v nutnosti uchovávání určitého množství informace o aktuálně otevřených dílčích oknech. Tento problém je ovšem vyřešen způsobem TDI, v současnosti velice oblíbeném.

TDI (Tabbed document interface). V tomto případě se jedná o grafické uživatelské rozhraní, které je založeno na stejném principu jako MDI. Rozdíl je ten, že rodičovské okno obsahuje záložky (tabs), díky kterým se lze navigovat mezi jednotlivými okny. Nevýhodou tohoto způsobu je fakt, že nelze zobrazit současně dvě okna, neboť je vždy viditelné pouze jedno.

Pro MDI grafické uživatelské rozhraní existuje v Microsoft .NET Framework nativní podpora. Objekt, který reprezentuje MDI formulář, je standardní formulář, jemuž je nastavena vlastnost `IsMdiContainer` na `true`. Tímto se stanou aktuální další vlastnosti tohoto objektu, uchovávající informace o jednotlivých oknech aplikace.

Grafika a Graphics Device Interface Plus (GDI+)

Microsoft Windows GDI+ je grafické rozhraní pro zařízení, které umožňuje psát programátorům aplikace nezávislé na těchto zařízeních. Je zodpovědné za zobrazování informace na obrazovku a tiskárnu. GDI+ je následník starší verze GDI, který je nově dostupný v Microsoft Windows XP a Windows Server 2003. Tato nová verze podporuje verzi starší, nicméně nové aplikace by měly používat GDI+, protože mnoho metod bylo optimalizováno a byly přidány nové možnosti. Není to součást .NET Framework, ale je instalována společně s tímto prostředím a je přístupná díky jmennému prostoru System.Drawing. Toto API je postaveno na množině tříd, implementovaných v programovacím jazyce C++.



Obrázek 2.3: Architektura GDI+.

Grafické rozhraní obecně umožňuje programátorům zobrazovat informace na zařízení bez nutnosti znalosti detailů o daném zařízení. Třídy, které obsahuje, nabízejí vykreslování na následující tři druhy:

- **Obrazovka.** Možnost vykreslování na obrazovku je nejdůležitější vlastnost. GDI+ umožňuje vykreslování komplexních tvarů, textu a obrázků s velkou flexibilitou.

- **Obrázky.** GDI+ také podporuje vykreslování do obrázků, které jsou buďto nahrané z fyzického umístění (např. na disku), nebo jsou generovány v paměti.
- **Tiskárna.** Také tisk je podporován v GDI+, což celý proces dělá přímým a jednodušším.

Služby Microsoft Windows GDI+ lze rozdělit do třech hlavních oblastí:

2D vektorová grafika

Vektorová grafika zahrnuje kreslení primitiv (čáry, křivky...), která jsou určena množinou bodů v určitém souřadnicovém systému. Tato část GDI+ zahrnuje třídy popisující daná primitiva, popisuje, jak mají být primitiva vykreslována a zajišťuje také samotné vykreslování.

Zobrazování obrázků

Určité druhy obrázků není možné zobrazit pomocí vektorové grafiky (fotky ve vysokém rozlišení...). Takovéto obrázky jsou uloženy jako bitmapy. Třídy, které uchovávají informace o bitmapách a pracují s nimi jsou komplexnější než třídy pro vektorovou grafiku, proto jich je zde větší množství.

Typografie

Tato část zajišťuje zobrazování textu mnoha způsoby. Jednou z novinek oproti starší verzi je antialiasing, který zajišťuje hladší zobrazování textu na LCD displejích.

Jelikož téměř všechny třídy GDI+ jsou jakousi obálkou nad neřízenými zdroji operačního systému, je nutné na tento fakt brát zřetel a po dokončení práce s těmito zdroji je uvolnit, aby nedošlo k jejich vyčerpání.

Jádrem tohoto API je třída Graphics, která zajišťuje vykreslování přímek, křivek, diagramů, obrazů a textu. Obsahuje velké množství metod, které slouží jak k obecnému použití (Rect class, Point class...), tak ke specifickým účelům (Bitmap class...). Zároveň je zde obsaženo několik struktur, které slouží k organizaci dat, výčtových typů a dalších.

Princip vykreslování

Jak již bylo zmíněno, jádrem GDI+ je objekt `Graphics`. Každá plocha v aplikaci má svůj vlastní objekt `Graphics`, který může být použit pro kreslení na tento povrch. Tento objekt je většinou získán jako parametr při zpracování události `Paint`. Tato událost je vyvolána pokaždé, když systém rozhodne, že daná plocha musí být překreslena. Při vykreslování má tedy programátor možnost odchytit tuto událost nebo může zajistit vlastní implementaci metody `OnPaint()`, která je automaticky volána systémem. Pokud programátor potřebuje vyvolat překreslení oblasti nezávisle na rozhodnutí systému, lze k tomu použít metodu `Invalidate()`, která zneplatní obsah oblasti.

Vykreslování obrázků

Pro práci s obrázky jsou důležité dvě hlavní třídy. Jedná se o třídu `Image` a třídu `Bitmap`. Třída `Image` umožňuje načítání obrázků do paměti, čímž lze s nimi manipulovat, a poskytuje

spoustu informací o vlastnostech obrázku jako velikost, rozměry, barevná hloubka a další. Mezi užitečné funkce, které s sebou tato třída přináší, tedy patří:

- Načítání obrázků ze souboru a zprostředkování informací o něm.
- Ukládání obrázků do souboru různých formátů (JPEG, GIF, PNG...).
- Rotace a převracení obrázků.
- Zpřístupňuje manipulaci s obrázkem prostřednictvím dalších tříd v kombinaci s `Image` objektem. Například kreslení čar prostřednictvím přidruženého objektu `Graphics`.

Samozřejmě zde existuje spousta dalších vlastností a metod pro tuto třídu, ale výše zmíněné jsou používány nejčastěji. Druhá třída `Bitmap` je od této odvozená. Přidává další možnosti pro manipulaci s rastrovými obrázky.

2.3 C# a ostatní .NET programovací jazyky

Tento programovací jazyk byl vyvíjen společně se softwarovou komponentou Microsoft .NET Framework. Je určen především pro použití s tímto prostředím, na což byl brán zřetel při jeho vývoji. Nelze ho ovšem považovat jako součást .NET Framework. Jedná se o objektově orientovaný programovací jazyk, který kombinuje pozitivní vlastnosti všech podobných programovacích jazyků, které v době vývoje byly již rozšířené. Jak napovídá označení programovacího jazyku, jeho syntaxe je postavena na kombinaci syntaxe jazyků C a C++, i když někteří vývojáři tvrdí, že je syntaxe spíše podobná programovacímu jazyku Java. Díky již zmíněnému mezikódu není C# jediným programovacím jazykem, který lze v .NET prostředí použít.

Mezi další známé jazyky patří například Visual Basic .NET. Jazyk vychází ze starší verze Visual Basic 6, nicméně změny, které byly provedeny pro lepší použitelnost s .NET prostředím, dělají z této verze téměř nový samostatný jazyk. Výrazný rozdíl mezi těmito dvěma verzemi je v tom, že verzi novější již nelze přeložit do nativního kódu, ale pouze do mezikódu.

U dalšího známého jazyka Visual C++ .NET opět přibyla jistá rozšíření oproti verzi Visual C++ 6. S jistými omezeními jej lze tedy také použít pro kompilaci do mezikódu. Vzhledem k možnosti použití nízkoúrovňových pointerů ovšem nelze zajistit silnou typovou kontrolu. U tohoto programovacího jazyka lze ještě docílit kompilace do tzv. neřízeného kódu (unmanaged code), kdy dochází ke kompilaci přímo do nativního kódu a je vynecháno společné běhové prostředí.

Mezi další známé programovací jazyky patří Visual J# .NET, JScript .NET a další.

2.4 Podpůrné nástroje prostředí Microsoft .NET

Nejpopulárnějším nástrojem pro vývoj aplikací v .NET prostředí je sada nástrojů Visual Studio. V současné době je aktuální verze Visual Studio 2005. Jedná se o následníka předešlé verze Visual Studio .NET. Tento nástroj podporuje tvorbu aplikací především prostřednictvím řízeného kódu. Lze jej využít k vytvoření různých druhů aplikací – konzolové aplikace, Windows aplikace, Windows Mobile aplikace, ASP.NET aplikace a dalších. Programovací jazyky, které jsou v tomto prostředí podporovány, jsou C#, C++, Visual Basic.NET, J# a další.

Některé užitečné vlastnosti Visual Studia 2005:

- **IntelliSense** IntelliSense napomáhá při programování tím způsobem, že zobrazuje programátorovi přístupné třídy, metody a vlastnosti, které jsou v daném kontextu k dispozici.
- **Návrháři** Visual Studio zahrnuje vizuální WYSIWYG návrháře, kteří napomáhají programátorům upravit design jejich aplikací.
- **Ladění** Jedna z nejdůležitějších vlastností Visual Studia je schopnost krokovat aplikaci řádek po řádku, jak jsou prováděny.
- **Organizace** Jelikož je Visual Studio vytvořeno pro vývoj aplikací, nabízí intuitivní metody pro organizaci souborů se zdrojovými kódy do projektů a projekty do řešení.

2.5 Microsoft .NET Framework a jeho portace

Ačkoli je Microsoft .NET Framework přednostně určen pro platformu Windows, existuje v dnešní době několik projektů zabývajících se portací tohoto prostředí na platformy jiné. Mezi nejznámější patří projekt Mono a DotGNU Portable.NET.

Mono

V současné době je tento open-source projekt veden firmou Novell. Mezi operační systémy, které jsou podporovány, patří Linux, FreeBSD, UNIX, MacOS X, Solaris a Windows. Sestává se ze tří hlavních částí: hlavní komponenty, které zahrnují C# překladač, virtuální stroj a základní třídy, dále pak balíček nástrojů pro vývoj a balíček zajišťující možnost portace Windows aplikací do systému Linux. Součástí je také vývojové prostředí, které nese název MonoDevelop. Vývoj tohoto systému stále aktivně pokračuje, i když zde existují jistá omezení způsobená licenčními podmínkami společnosti Microsoft. Důkazem toho, že je API tohoto systému používáno, je i nemalé množství aplikací, které pod tímto systémem vznikly. Samotný překladač pro tento systém je napsán v C#, což se nepříjemně projevuje na době kompilace.

DotGNU Portable.NET

Tento projekt vznikl jako reakce GNU. Stejně jako Mono je vyvíjen jako open-source a podporuje více operačních systémů jako Linux, BSD, MacOS X, Solaris, AIX a Windows. Oproti systému Mono je překladač napsán v ANSI C, což zaručuje dobrou přenositelnost a hlavně výrazně kratší dobu kompilace. Na druhou stranu implementace JIT (Just-in-Time compilation) a podpora knihoven je u tohoto systému slabší než v případě Mona.

Jako největší nevýhoda se momentálně jeví problémy spojené s knihovnou Windows.Forms, jejím nahrazením v těchto systémech a se vzájemnou kompatibilitou. Dále také existence vývojového prostředí Microsoft Visual Studio, které prozatím nemá žádnou adekvátní náhradu.

Kapitola 3

Vyhledávání klíčových bodů v obraze

Metody vyhledávání klíčových bodů spadají pod skupinu, které se obecně říká detekce vlastností. V počítačovém vidění a zpracovávání obrazu se jedná o metody, které extrahují informace z obrazu a u každého bodu obrazu rozhodují, zda se jedná o vlastnost hledaného typu, či nikoliv. Výsledkem těchto metod je podmnožina bodů obrazu ve formě izolovaných bodů nebo křivek či regionů [13].

Neexistuje jednotná definice pro vlastnost obrazu. Podle literatury [7] by se dalo říci, že se jedná o „zajímavou“ část zkoumaného obrazu. Tyto metody jsou základem mnoha algoritmů v oboru počítačového vidění, a proto je jejich efektivita klíčová pro algoritmy, jež je využívají.

Skupiny vlastností, které jsou v obraze vyhledávány, lze rozdělit do následujících typů:

Hrany jsou takové body, které leží na hranici mezi dvěmi oblastmi. V praxi jsou hrany většinou definovány jako množiny bodů s vysokým gradientem.

Rohové body, klíčové body byly dříve detekovány za použití metod pro vyhledávání hran v obraze jako součást procesu, ale postupem času byly algoritmy vyvinuty do takové podoby, že již tyto metody nebyly potřebné.

Blobs (klíčové oblasti, klíčové body) narozdíl od metod vyhledávání rohových bodů poskytují doplňující popis obrazu, co se oblastí týče. Tyto detektory jsou schopné vyhledat oblasti v obraze, které jsou příliš hladké na to, aby byly detekované detektory rohových bodů.

Vrcholy lze považovat za křivky, které reprezentují osy souměrnosti. Metody vyhledávající tuto vlastnost se často používají k nalezení cesty v obrazech krajin nebo k zvýraznění krevních cév v obrazech.

3.1 Detektory hran

Hrany jsou místa v obrazech s vysokým jasovým kontrastem [5]. Jelikož se hrany často objevují v obraze jako hranice objektů, detekce hran je značně používána při rozdělení obrazu do oblastí, které odpovídají různým objektům. Reprezentace obrazu prostřednictvím nalezených hran má mimo jiné výhodu v tom, že při udržení většiny informací o obraze je dosaženo výrazného snížení objemu dat potřebných k popisu obrazu.

Díky tomu, že jsou hrany převážně tvořeny vysokými frekvencemi, lze je podle literatury [13] detekovat aplikováním horní propusti ve frekvenční oblasti nebo pomocí konvoluce obrazu s odpovídajícím jádrem[1] závisejícím na aplikované metodě v souřadném systému. V praxi se používá druhá zmíněná možnost vzhledem k její menší výpočetní složitosti a vzhledem k faktu, že dosahuje lepších výsledků. Protože hrany odpovídají vysokým světelným gradientům, lze je zvýraznit výpočtem derivace obrazu. Pozice hrany může být určena pomocí maxima první derivace nebo průnikem nulou druhé derivace. Následuje seznam vybraných detektorů založených na první derivaci. Jejich detailní popis lze nalézt v odborné literatuře [4] [5] [13]. Porovnání výstupů jednotlivých detektorů je zobrazeno v tabulce 3.1.

- **Robertsův detektor**

  Robertsův detektor provádí jednoduché, výpočetně nenáročné měření plošného gradientu obrazu. Proto zvýrazňuje oblasti vysokého plošného gradientu, které často odpovídají hranám. Ve většině případů je vstupem, stejně jako výstupem, obraz ve stupních šedi. Hodnoty pixelů ve výsledném obraze odpovídají absolutní hodnotě plošného gradientu vstupního obrazu v daném bodě. Díky malému okolí, které používá Robertsův operátor, je tento detektor citlivý na šum.

- **Sobelův detektor**

  Oproti Robertsově detektoru lépe aproximuje první parciální derivace. Díky této skutečnosti je směrově závislý. Jeho použití se často uplatňuje při detekci vodorovných a svislých hran.

- **Prewittové detektor**

  Po vypočtení hodnoty první derivace gradientu je nutné identifikovat pixely, které odpovídají hranám. Nejjednodušší cesta je prahování, při kterém pixely, jejichž gradient je vyšší než odpovídající práh, odpovídají právě hledaným hranám. Altenativní technika, která produkuje jeden pixel široké hrany, spočívá ve zjištění lokálního maxima v obraze. Sofistikovanější metodu používá Cannyho hranový detektor.

- **Cannyho hranový detektor**

  Tento detektor byl navržen jako optimální detektor hran se třemi kritérii:

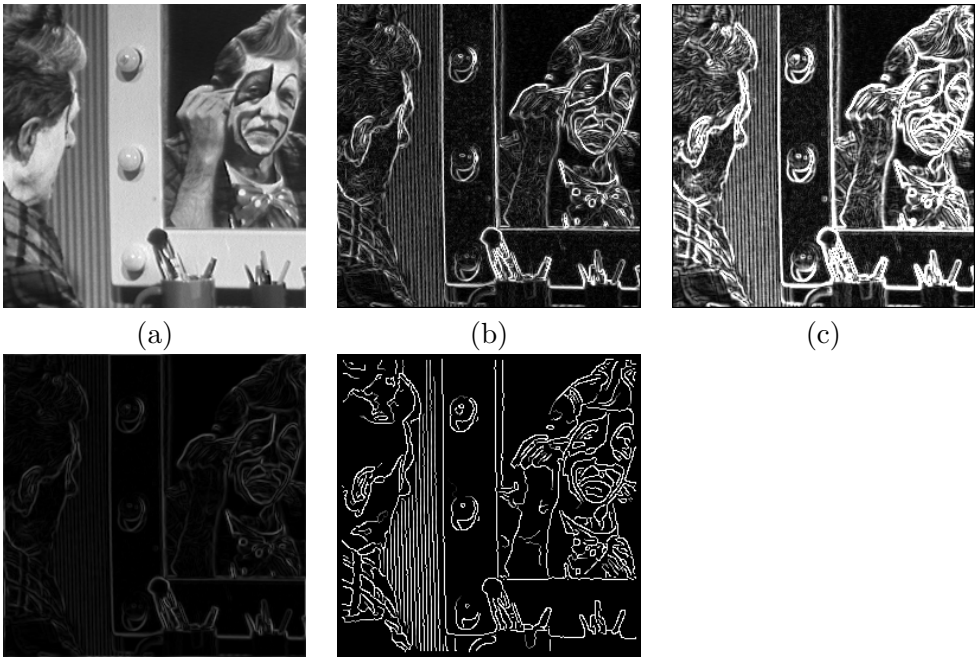
  - *Detekční kritérium* zajišťuje neopomenutí významných hran.
  - *Lokalizační kritérium* minimalizuje rozdíl mezi skutečnou a nalezenou pozicí hrany.
  - *Kritérium jednoznačnosti* zajišťuje, aby detektor nereagoval vícekrát na jednu hranu.

  Jako vstup slouží obraz ve škále šedé barvy a výstupem je obraz zobrazující nalezené jasové nespojitosti.

  Cannyho hranový detektor pracuje jako vícestupňový proces. Nejdříve je obraz vyhlazen Gausovou konvolucí[2] a následně jsou na obraze zvýrazněna místa s velkou

[1] Jádro (kernel) je matice o malém rozměru, která se používá při konvoluci obrazů. Struktura matice závisí na kýženém efektu konvoluce.

[2] Gausova konvoluce slouží k rozostření obrazu, odstranění detailů a šumu.



(a) (b) (c)

(d) (e)

Tabulka 3.1: **Výstupy detektorů založených na první derivaci:** (a) Origiální obrázek, (b) Robertsův detektor, (c) Sobelův detektor, (d) Prewittové detektor, (e) Cannyho hranový detektor

hodnotou první derivace. Následuje proces sledování těchto regionů, který je řízen dvěma prahy, a zajišťuje zvýraznění maxim, tedy hledaných hran.

Detektory založené na druhé derivaci

Do této skupiny lze zahrnout Zero-crossing detektor, ve kterém je pro výpočet druhé derivace použit LoG filtr (Laplacian of Gaussian)[3]. Výhodou tohoto filtru je fakt, že výpočet druhé derivace není závislý na orientaci hrany.

Hledání průchodů nulou (Zero-crossing detektor)

Tento detektor je také znám jako Marr edge detector, Laplacian of Gausian detector. Princip tohoto detektoru závisí na hledání takových bodů, u nichž dojde v Laplaceově operátoru[4] ke změně znaménka (hodnota projde nulou). Jsou to takové body v obraze, kde se intenzita rapidně mění, tedy hrany. Body s takovými vlastnostmi nelze ovšem vždy označit jako hranové. Proto je lepší tento detektor posuzovat jako druh detektoru vlastností než specifický detektor hran.

Výstupem tohoto detektoru je většinou binární obraz s čárami širokými jeden pixel, které označují body, jejichž hodnota Laplaceova operátoru prochází nulou. Výsledky tohoto

[3]Podle literatury HIPR LoG filtr spočívá v použití Gausovského rozostření obrazu, následně použití Laplaceova filtru za účelem nalezení hran v obraze.

[4]Laplaceův operátor je diferenciální operátor ve vektorové analýze, definovaný jako divergence gradientu daného pole. Literatura [7] uvádí, že se v tomto případě jedná o míru druhé plošné derivace obrazu.

(a) (b) (c)

Tabulka 3.2: **Zero-crossing detektor:** (a) Origiální obrázek, (b) Aplikace LoG filtru, (c) Aplikace Zero-crossing detektoru na (b).

detektoru jsou z velké části ovlivněny vyhlazovacím filtrem, jenž je na začátku procesu aplikován. Čím je vyhlazení větší, tím je nalezeno méně těchto bodů. Detailní popis tohoto detektoru lze nalézt v literatuře [2]. Aplikace LoG filteru na originální zpracovávaný obrázek a výsledek tohoto detektoru je zobrazen v tabulce 3.2.

Obecným problémem metod vyhledávání hran je jejich citlivost na šum ve zpracovávaném obraze. Ta je způsobena výpočtem derivace, která vyzdvihuje vysoké frekvence, a tudíž posiluje šum. V Cannyho hranovém detektoru a Zero-crossing detektoru je tento problém do jisté míry eliminován aplikací vyhlazujícího operátoru před vlastním výpočtem derivace.

3.2 Požadavky na detektory

Podle literatury [13] je po ideálním detektoru hran vyžadováno, aby přesně určil bod hrany. Tento bod by neměl být vynechán, zatímco body, které neodpovídají hranám, by neměly být chybně detekovány. Tyto dva požadavky se navzájem rozcházejí. Rozhodování, zda se jedná o bod hrany, je založeno na prahování. Pokud velikost gradientu daného bodu je větší než stanovený práh, je bod označen jako bod hrany, v opačném případě nikoliv. Pokud je práh nastaven na příliš velkou hodnotu, může dojít k vynechání některých bodů hran. Pokud je práh nastaven příliš nízko, může být naopak v obraze detekováno příliš šumových bodů. Proto cílem ideálního hranového detektoru je stanovit optimální práh. Zároveň by měl být ideální detektor odolný vůči jasovým změnám a změnám kontrastu, rotaci, perspektivě, změně měřítka a již zmíněnému šumu.

Aby bylo možné srovnávat jednotlivé detektory, vznikl požadavek na nalezení způsobu určení úspěšnosti těchto detektorů. Toto obecné vyhodnocení ovšem není zcela jednoznačné, neboť nelze určit jaké vlastnosti obrazu byly při detekci požadovány nalézt. Mezi kritéria, na která je při určování úspěšnosti některých metod brán zřetel, jsou:

- pravděpodobnost detekce falešných hran
- pravděpodobnost nedetekovaných hran
- střední odchylka detekované hrany od hrany skutečné
- tolerance algoritmu k šumu a dalším vlastnostem

3.3 Omezení detektorů

- Hrany detekované pomocí klasických metod vyhledávání klíčových bodů často nekorespondují s hraničními objekty. V mnoha obrázcích nízké kvality některé z konvenčních metod produkují falešné hrany a mezery a jejich použití je proto limitováno.
- Technika detekce hran závisí na informaci obsažené v lokálním blízkém okolí obrázku.
- Ve většině případů postupy při detekci hran ignorují vyšší řád uspořádání.
- Body, které jsou v obraze detekovány, jsou následně spojovány, aby se určily hranice v obraze. Tento proces probíhá nejdříve sdružením jednotlivých hran do segmentů a následně sdružením segmentů hran do hranic. To ovšem občas vede k nespojitostem a mezerám v obraze.
- Aby se odstranily mezery v hranicích, uchylují se často metody spojování hran k interpolacím.
- Často je obtížné identifikovat a klasifikovat falešné hrany.

3.4 Vyhodnocování úspěšnosti vyhledávání klíčových bodů

Neexistuje obecný postup, jak vyhodnocovat úspěšnost vyhledávačů klíčových bodů. Jedná se o velice kompilkovanou operaci vzhledem k existenci mnoha možných kritérií. Některá z možných kritérií jsou uvedena v následujícím seznamu.

- **Přesnost umístění a opakovatelnost**

  Přesnost umístění (location accuracy) je jedním z nejpoužívanějších kritérií. Určuje, s jakou přesností je detekovaný bod umístěn v daném 2D prostoru. Opakovatelnost (repeatability rate) vyhodnocuje geometrickou stabilitu detekovaných bodů mezi obrázky stejné scény zachycené pod různými pohledy nebo pozměněné určitou geometrickou transformací. Detekovaný bod je „opakován“ v případě, že bod z prvního obrázku je správně detekován i v druhém. Opakovatelnost je procentuální vyjádření těchto bodů vzhledem k celkovému počtu bodů detekovaných v obou obrázcích. Vyhodnocování na základě přesnosti umístění a opakovatelnosti má tu nevýhodu, že pokud se provede vyhlazení obrázku, dojde sice ke zlepšení ze strany opakovatelnosti, ale naopak ke degradaci přesnosti umístění [11].

- **Množství informace v obraze** Jedná se vyhodnocování na základě míry charakteristické zvláštnosti bodu. Tato metoda je založena na podobnosti lokálního deskriptoru hodnoty šedé vypočítaného v daném bodu. Tento výpočet je prováděn pro jednotlivé zkoumané obrázky. Množství informace se měří na základě entropie [11].

Mezi další kritéria patří vyhodnocování na základě necitlivosti detektoru na šum a jasové změny obrázku nebo metody založené na subjektivním posouzení ze strany uživatele, jejichž výsledky se mohou výrazně lišit.

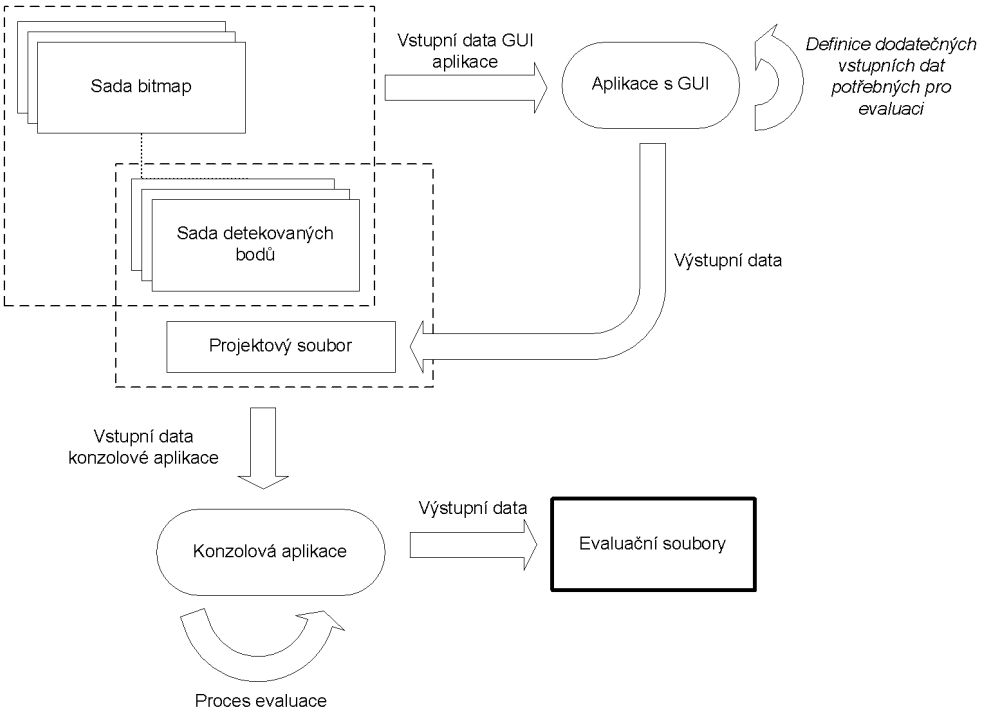
Kapitola 4

Požadavky na evaluaci detektorů klíčových bodů v obraze

Hlavním cílem diplomového projektu bylo vyvinutí aplikace s grafickým uživatelským rozhraním a konzolové aplikace, které za pomoci stanovené metody pro hodnocení úspěšnosti vyhledávání klíčových bodů v obraze umožňují evaluaci detektorů. Jako vstupní data pro vyhledavače slouží sada bitmap, ve kterých je zachycena ta samá scéna v různých úhlech pohledu. Zpracováním takové skupiny bitmap detektory je zkoumána jejich invariantnost vzhledem ke změně projekce, měřítka a jasových poměrů. Výstupními daty vyhledávačů je skupina souborů reprezentující detekované body v jednotlivých bitmapách. Informace uložené o těchto bodech jsou ovšem vztažené vždy k souřadnému systému dané bitmapy. To neumožňuje jejich vzájemné vyhodnocení. Proto bylo potřeba vymyslet způsob, kterým by se informace o bodech převedly do společného systému vhodného pro evaluaci. Takovýmto systémem je v tomto případě společný souřadnicový systém. Samotné převedení není ovšem realizovatelné, aniž by nebyly stanoveny dodatečné informace.

Pokud je určen ve dvojici souřadných systémů takový objekt, jehož obraz odpovídá jinému obrazu téhož objektu, lze souřadnice bodů uvnitř tohoto objektu přepočítat na souřadnice společného souřadného systému. Nejprimitivnější objekt, který lze k této metodě využít, je trojúhelník. Jeho použití s sebou ovšem nese určité nevýhody. Jednou z nich je nemožnost zohlednit vliv prespektivy na vnitřní souřadnice v objektech při převodu na společný souřadný systém, což vede k nepřesnosti ve vyhodnocovací metodě. Tento nedostatek lze vyřešit použitím jiného, složitějšího objektu – čtyřúhelníku. Jelikož nebyla známa míra nepřesnosti při použití trojúhelníku, byla implementována metoda evaluace s tímto objektem s možností snadného rozšíření apliakce o evaluaci za pomocí čtyřúhelníku.

Definice těchto objektů je ona přidaná informace ze strany uživatele, která umožňuje vlastní evaluaci. Určení těchto objektů uživatelem by nebylo možné bez grafické reprezentace bitmap a bodů v nich detekovaných. Proto vznikl požadavek na vytvoření aplikace s grafickým uživatelským rozhraním. Po fázi definice objektů jsou k dipozici kompletní data potřebná k procesu samotného vyhodnocení. Tuto akci zajišťuje konzolová aplikace jakožto samostatný celek. Tím je zajištěna možnost použití při iterativním návrhu a učení algoritmů pro vyhledávání. Proces postupu od definice počáteční dodatečné informace až k samotné evaluaci zachycuje obrázek 4.1.



Obrázek 4.1: Proces použití aplikací při evaluaci.

4.1 Aplikace s grafickým uživatelským rozhraním

Cílem bylo navrhnout a implementovat takové grafické uživatelské rozhraní, které by bylo pro uživatele dostatečně intuitivní a umožňovalo mu snadnou a efektivní práci.

Vzhledem k typu aplikace, ve které lze pracovat s několika dokumenty najednou, byly vhodné dva přístupy při návrhu grafického uživatelského rozhraní. Jedná se o v dešní době velice rozšířené TDI (Tabbed document interface) a MDI (Multiple document interface). Typ rozhraní TDI s sebou přináší jednodušší manipulaci s otevřenými dokumenty v aplikaci a lepší přehled ze strany uživatele. Na druhou stranu oproti MDI není možné při tomto typu rozhraní zobrazit několik dokumentů současně. Proto je v aplikaci použito rozhraní typu MDI na úkor zhoršené manipulace s dokumenty, což lze vykompenzovat jinými prostředky. Uživatel tak má možnost zobrazení několika bitmap najednou pro případné srovnání.

Jelikož aplikace slouží k práci se sadou bitmap a definici dodatečné informace k této sadě, měla by umožňovat snadnou správu této sady jako přidávání bitmapy spolu s detekovanými body, odebrání bitmapy, určení pořadí v sadě a další operace. Uživatel by měl mít zároveň přehled o tom, jaké bitmapy patří do editované sady. Proto by uživatelské rozhraní mělo obsahovat seznam těchto bitmap. Stejně tak pro objekty definované u jednotlivých bitmap by měl existovat jejich seznam pro snadnější manipulaci a přehled.

Při definování objektů v bitmapě je nutné brát ohled na přesnost při definici objektů a umožnit uživateli pomocí prostředků aplikace zadat co nejvíce odpovídající údaje. Proto byla zakomponována do aplikace možnost zobrazení oblasti bitmapy se zvětšeným faktorem přiblížení, a tím bylo docíleno zredukování zanesení nepřesnosti ze strany uživatele. Aby bylo možné dodatečně upravit pozici a tvar objektu, byla uživateli umožněna dodatečná editace objektu.

Jelikož aplikace pracuje s řadou parametrů, jejichž hodnota závisí na uživatelově rozhodnutí, je v aplikaci zpřístupněno jejich nastavení. Zároveň má uživatel možnost nastavit design objektů v aplikaci. Tato funkčnost je v aplikaci z toho důvodu, že zvolený vzhled by nemusel vždy vyhovovat zobrazené bitmapě a grafická reprezentace by se tak stala nepřehledná.

Celkový proces definování dodatečné informace a práce s bitmapami je pojat v aplikaci formou práce s projektovým souborem. Uživatel má tedy možnost vytvoření nového projektu, import bitmap s detekovanými body, možnost uložení projektu do projektového souboru, který je následně použit jako součást vstupních dat pro konzolovou aplikaci.

4.2 Konzolová aplikace

Konzolová aplikace slouží ke zpracování projektového souboru spolu se soubory obsahujícími detekované body pro bitmapy. Mělo by být zajištěno její jednoduché volání, aby ji bylo možné použít při procesu evaluace detektorů. Jelikož se jedná o automatizovanou akci, neměla by aplikace vyžadovat interakci uživatele kromě počátečního zadání parametrů. Parametry aplikace by měly umožnit snadné definování vstupních dat, určení dodatečných nastavení evaluačního procesu, a tím umožnit uživateli vyřazení nastavení uložených v projektovém souboru. Jelikož evaluace může být v závislosti na množství dat časově náročná, mělo by být zajištěno průběžné informování uživatele o vnitřním stavu aplikace.

4.3 Vstup a výstup software

- Vstupem aplikace s grafickým uživatelským rozhraním je sada obrázků, ve které byly pomocí detektorů klíčových bodů nalezeny tyto body, a příslušná sada souborů nesoucí informaci o detekovaných bodech.
- Výstupem aplikace s grafickým uživatelským rozhraním je projektový soubor nesoucí dodatečné informace od uživatele potřebné pro evaluaci.
- Vstupem konzolové aplikace je projektový soubor získaný pomocí uživatelovi interakce prostřednictvím grafického uživatelského rozhraní spolu se sadou souborů s odpovídajícími detekovanými body.
- Výstupem konzolové aplikace je informace o vyhodnocení úspěšnosti použitého detektoru klíčových bodů na sadě vstupních bitmap.

Kapitola 5

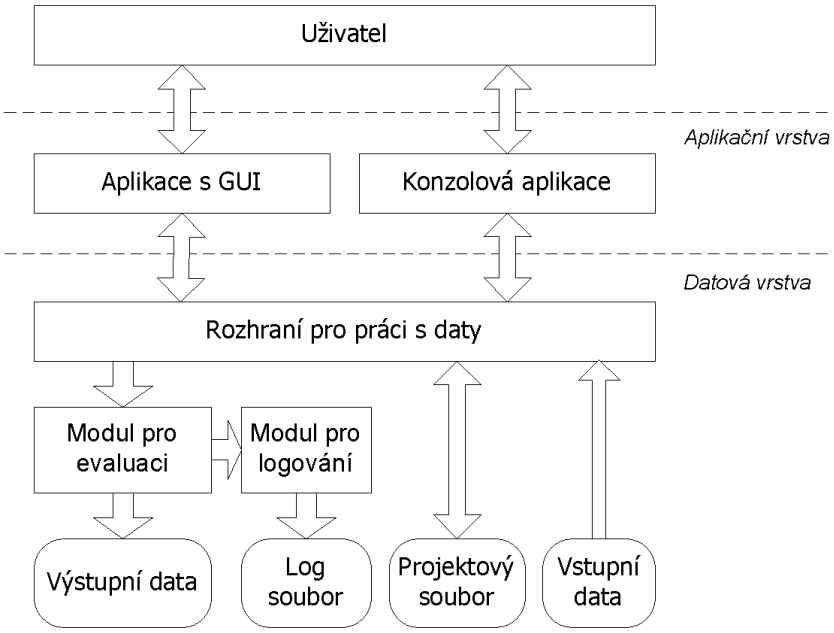
Struktura projektu a jeho implementace

Z implementačního hlediska nebyly na software kladena žádná omezení či požadavky. Jelikož se na Ústavu počítačové grafiky převážně používá operační systém Microsoft Winows, byl software pro tento systém vyvíjen. Konkrétně pro verzi Microsoft Windows XP. Je známo několik přístupů, jak lze vytvořit software pro tento operační systém. Díky možnosti efektivního vývoje a existenci kvalitních podpůrných prostředků byla pro implementaci zvolena platforma Microsoft .NET Framework verze 2.0 spolu s programovacím jazykem C#, který je pro tuto platformu jazykem nativním. V dnešní době existuje řada portací pro jiné operační systémy, které umožňují běh takto vyvíjeného software. Jelikož nebyly kladeny požadavky na přenositelnost, software na nich testovaný nebyl, a tudíž nelze zaručit jeho ekvivalentní funkčnost především u aplikace s grafickým uživatelským rozhraním díky neúplné náhradě části WinForms.

Jako vývojové prostředí byl použit produkt společnosti Microsoft – Microsoft Visual Studio 2005. Jedná se v dnešní době o jedno z nejrozšířenějších a zahrnuje pro vývojáře spoustu usnadnění, která zefektivňují práci programátora. Správa zdrojových kódů pro vyvíjený software je pod tímto prostředím pojata ve formě strukturovaného celku, který je označován anglickým termínem – solution. Tento celek lze dále rozdělit na jednotlivé části zvané projekty. V prostředí lze vytvořit několik typů projektů lišících se strukturou a účelem použití. Programátor má tak možnost rozdělit implementaci software do dílčích logických celků, které jsou určeny pro dané specifické operace. Zároveň je tak zajištěna lepší spravovatelnost zdrojových kódů a možnost navrhnout strukturu software takovým způsobem, aby bylo možné tyto projekty v případě potřeby opakovaně používat i pro vývoj softwaru jiného.

Jak již bylo zmíněno, skládá se vyvíjený software ze dvou samostatných aplikací. Oběma těmto aplikacím bylo potřeba umožnit přístup k datům, práci s daty a jejich vyhodnocení. Proto je struktura software rozdělena na dvě vrstvy. První vrstva zahrnuje jednotlivá uživatelská rozhraní, která zajišťují zobrazování relevantních informací uživateli a interakci s uživatelem. Druhá vrstva poskytuje operace nad daty tohoto software. Zahrnuje modul pro práci s projektovým souborem. V tomto modulu jsou udržována veškerá data, se kterými aplikace pracují, vnitřní stavové informace projektu a potřebné metody pro práci s těmito informacemi. Dále je zde modul pro evaluaci, který poskytuje metody zajišťující veškeré operace týkající se vlastního procesu evaluace. Tyto moduly jsou provázané a navzájem spolu komunikují. Pro zaznamenávání akcí, které jsou prováděny v průběhu evaluace, je součástí

této vrstvy modul pro logování, který spolupracuje s modulem pro evaluaci. Rozdělením struktury software na tyto vrstvy je zajištěna nezávislost uživatelských rozhraní na datech. Schéma struktury software je zobrazeno na obrázku 5.1.



Obrázek 5.1: Schéma struktury software.

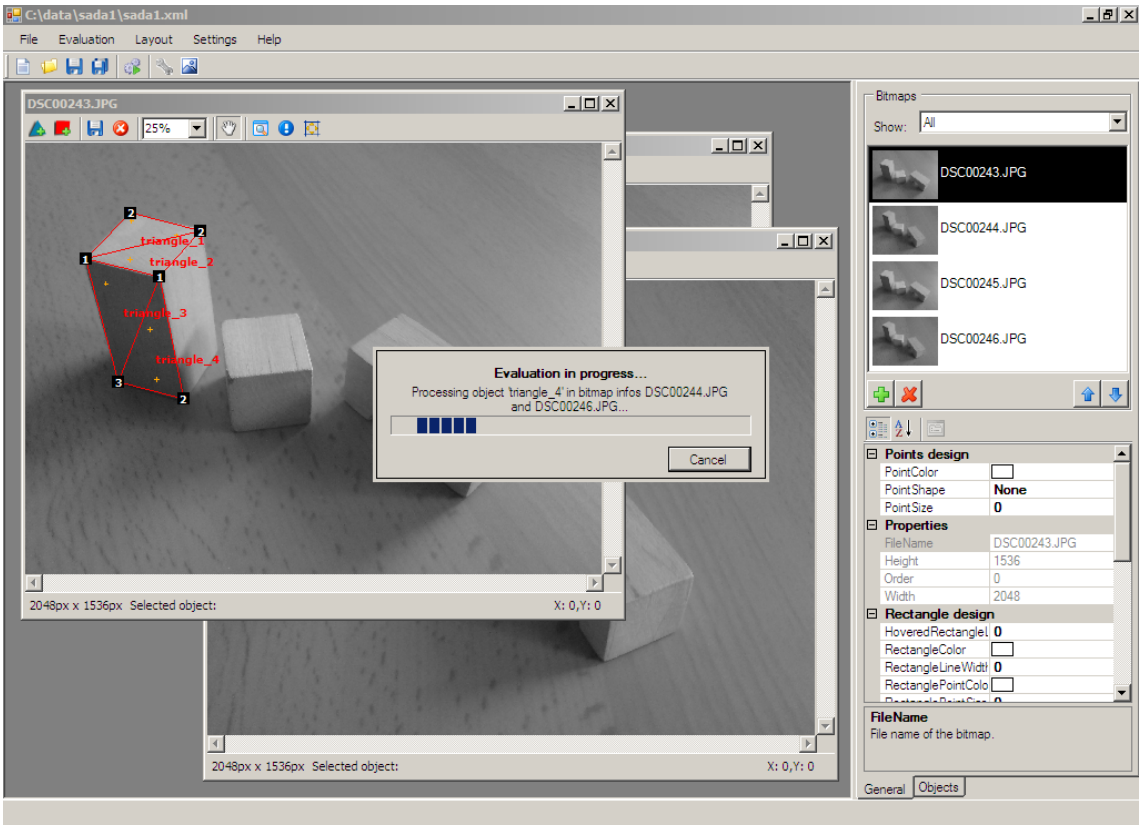
5.1 Aplikace s grafickým uživatelským rozhraním

Vývoj grafického uživatelského prostředí v Microsoft .NET Framework spočívá v použití takzvaných kontrolů. Jedná se o součásti rozhraní, které poskytují určitou specifickou funkčnost. K dispozici jsou standardní již implementované kontroly, jejichž funkčnost je používána v mnoha typech aplikace. Pokud tyto kontroly programátorovi nevyhovují, je zde možnost implementace kontrolů vlastních. Tento koncept zajišťuje možnost opakovaného použití již vytvořených kontrolů a zjednodušuje správu zdrojových kódů.

Jak již bylo zmíněno, byl pro své výhody použit pro grafické uživatelské rozhraní typ MDI. Jeden z důvodů, proč byl tento typ zvolen, je jeho existující nativní podpora v platformě Microsoft .NET Framework, konkrétně součásti WinForms. Grafické uživatelské rozhraní aplikace je tedy tvořeno hlavním formulářem, který slouží jako kontejner pro jednotlivé podřazené editační formuláře bitmap.

Specifikace uživatelského rozhraní probíhala v několika fázích v závislosti na požadavcích uživatelů. V konečné fázi je celkový design aplikace je rozdělen na tři hlavní části, jak lze vidět na obrázku 5.3. V horní části se nachází hlavní menu aplikace. Přes toto menu jsou přístupné operace s projektem, jako je jeho vytvoření, otevření, uložení, zavření a další. Dále je možné přes menu nastavit rozvržení podřazených editačních formulářů, nastavení parametrů evaluace a designu aplikace. U nastavení rozvržení editačních formulářů má uživatel k dispozici tři možnosti. Jedná se o kaskádové rozvržení, uspořádání vertikálně a uspořádání editačních formulářů vertikálně.

Uživatel má také přes toto menu možnost spuštění okamžitého procesu evaluace. Jelikož tato akce může podle objemu vstupních dat trvat delší časový úsek, což by vedlo ke zhoršené odezvě uživatelského rozhraní, je proces evaluace vykonáván v samostatném vlákně. V průběhu vyhodnocování jsou uživateli zobrazovány informace o momentálně prováděné operaci jak lze vidět na obrázku 5.2. V případě potřeby může uživatel proces přerušit pomocí tlačítka na informačním panelu. Po úspěšném vyhodnocení je zobrazen formulář s výsledky evaluace. Na tomto formuláři je k dispozici tlačítko pro jejich uložení.

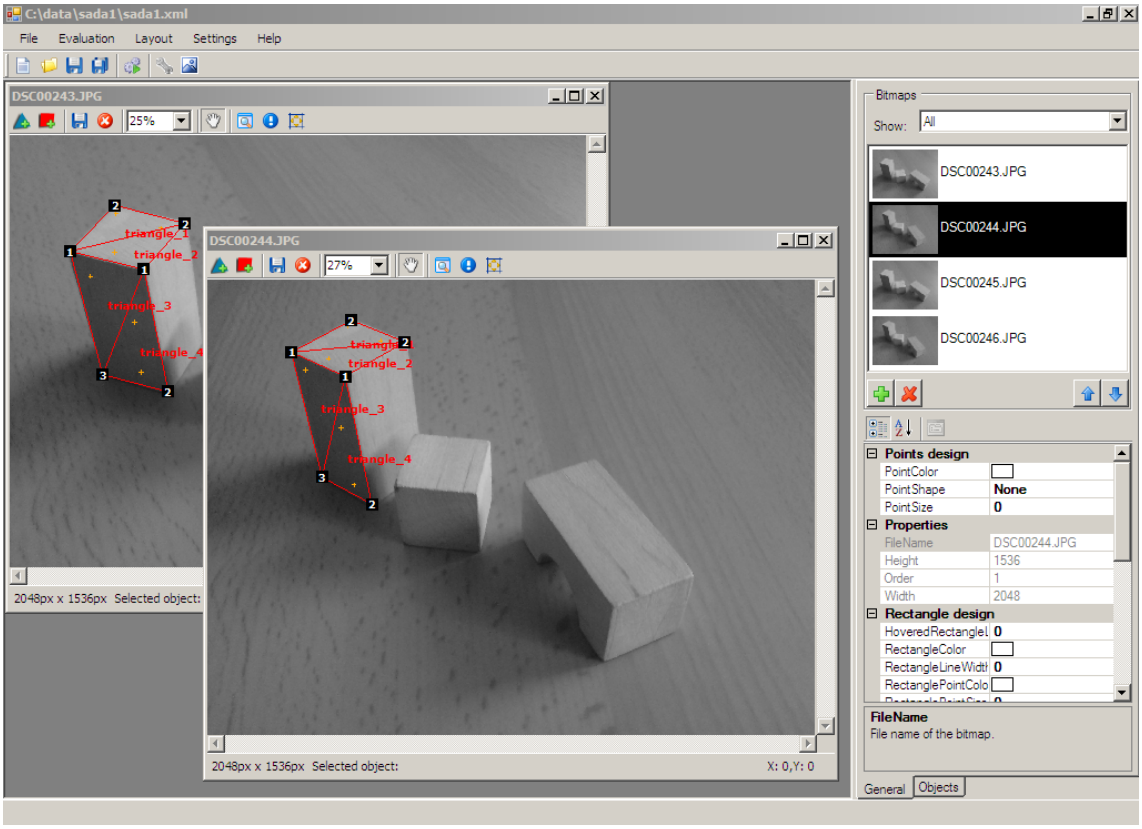


Obrázek 5.2: Asynchronní evaluace v aplikaci s grafickým uživatelským rozhraním.

Pod hlavním menu se nachází panel zpřístupňující vybrané akce z menu prostřednictvím tlačítek. Uživateli je tak umožněno vyvolat požadovanou akci bez nutnosti jejího hledání ve struktuře menu, což vede k efektivnější práci.

V pravé části grafického uživatelského rozhraní je umístěn vlastní kontrol, panel, který obsahuje dvě záložky. Na první záložce je zobrazen seznam bitmap obsažených v aktuálním projektu. Každá položka seznamu se skládá z náhledu bitmapy a názvu jejího souboru. Tento seznam slouží k výběru požadované bitmapy a zobrazení editačního formuláře. Zároveň lze přes tuto záložku měnit pořadí bitmap, přidávat bitmapy, odebírat bitmapy z projektu a definovat obecný design grafických objektů v bitmapě. K nastavení designu slouží editační rozhraní ve spodní části záložky. V případě, že uživatel vybere položku ze seznamu, je automaticky zobrazen editační formulář pro práci s bitmapou – `BitmapViewer`. V případě, že je aktivní jeden z otevřených editačních formulářů, je na druhé záložce panelu zpřístupněn seznam objektů obsažených v této bitmapě. Přes tento seznam lze vybírat jednotlivé objekty a díky editačnímu rozhraní, které je umístěno v dolní části, lze měnit některé vlastnosti

objektu včetně jeho designu.



Obrázek 5.3: Grafické uživatelské rozhraní aplikace.

Největší část formuláře aplikace zabírá část třetí. Jedná se o plochu, která je vyhrazena pro podřazené formuláře, sloužící pro editaci jednotlivých bitmap. Díky typu grafického uživatelského rozhraní lze docílit zobrazení několika editačních formulářů najednou.

Editační formulář bitmapy – `BitmapViewer`

Jelikož editační formulář má specifickou funkčnost určenou pro tuto aplikaci, bylo ho nutné implementovat pomocí vlastního kontrolu. Grafické rozhraní pro tento formulář se skládá z horního menu, které zpřístupňuje akce pro danou bitmapu, spodního panelu a hlavní části. Spodní panel zobrazuje průběžné informace o rozměrech editované bitmapy a poloze kurzoru vůči souřadnicím bitmapy. Hlavní část forumláře je určena pro vykreslování samotné bitmapy. Jelikož se při implementaci vyskytly výkonnostní problémy, které byly způsobeny neefektivním vykresolváním bitmapy standardními prostředky prostředí, bylo nutné vytvořit vlastní kontrol, který zajišťoval překreslování pouze té části bitmapy, která byla momentálně zobrazována. Vzhled editačního foruláře lze vidět na obrázku 5.4.

Účel formuláře spočívá v poskytnutí možnosti uživateli vytvářet a editovat objekty v bitmapě potřebných při procesu evaluace. Formulář pracuje ve dvou různých módech.

- První implicitní mód je určen k editaci již vytvořených objektů. Uživatel může vybrat kliknutím levého tlačítka myši na objekt nebo vrchol objektu, který hodlá editovat.

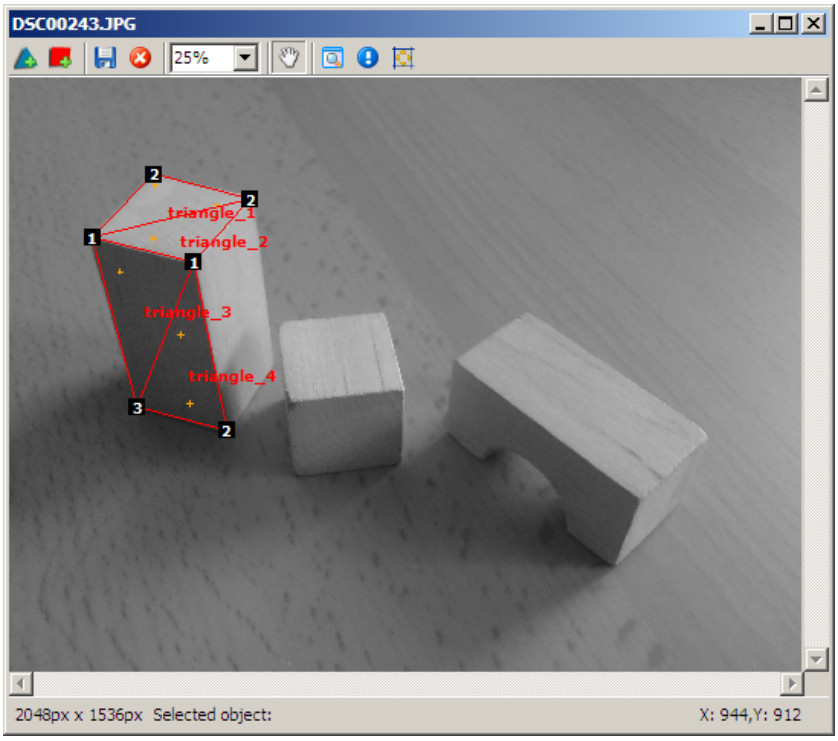
Aby měl uživatel informaci o tom, jaký objekt nebo vrchol je aktuálně vybraný, je objekt nebo vrchol graficky odlišen od ostatních. Možností změny vlastností objektu je uživateli umožněno zpřesnění polohy a jeho tvaru v případě, že je to potřeba. Samotná editace probíhá pomocí vrcholů objektu, u kterých lze interaktivně měnit jejich polohu. V tomto módu jsou uživateli zobrazeny podrobné údaje, jako kódové jméno objektu a indexy jednotlivých vrcholů. Každý objekt má vlastní kontextové menu, přes které lze nastavit dodatečné vlastnosti. Mezi ně patří index viditelnosti. Ten zajišťuje možnost určení pořadí vykreslování objektů, což je užitečné v případě, že se dva objekty překrývají. Pokud je kontextové menu vyvoláno na jednom z vrcholů objektu, lze pro tento vrchol nastavit pomocí tohoto menu jednu z předdefinovaných hodnot reprezentující vzdálenost vrcholu od kamery vůči ostatním vrcholům v rámci dané bitmapy.

- Druhý mód formuláře slouží pro definování nových objektů v bitmapě. Přepnutí mezi módy zajišťují tlačítka rozhraní určená k vytvoření nového objektu daného typu. Definice objektu spočívá v určení souřadnic vrcholů objektu pomocí levého tlačítka myši. V průběhu vytváření objektu jsou uživateli dočasně zobrazeny již určené body. Po definici dostatečného počtu potřebného pro vytvářený objekt, je tento objekt vykreslen do bitmapy a uživatel má možnost pokračovat v definici dalšího objektu. Zobrazena je pouze základní reprezentace objektu ve formě pouhých hran, aby nebyly zbytečně zakrývány detaily bitmapy.

Jelikož je sám uživatel odpovědný za to, aby co nejpřesněji určil souřadnice objektů, které si navzájem odpovídají, existuje ve formuláři několik možností jak zpřesnit zadávání vrcholů objektu. Jedním z nich je možnost změny přibližovacího faktoru pro bitmapu. Nejjednodušší způsob, jak této změny docílit, je použití skrolovacího tlačítka myši. Momentální hodnota přiblížení je zobrazována průběžně v horním menu formuláře. Lze ji taktéž zadat ručně, což umožňuje uživateli určit přesnou hodnotu přiblížení. Tato hodnota je zadávána v procentech. V případě nutnosti detailnějšího zobrazení určité oblasti bitmapy, lze použít funkci formuláře, která umožňuje definovat takovouto oblast. Poslední možný způsob zpřesnění zadávání souřadnic spočívá v použití nástroje formuláře, který průběžně zobrazuje okolí kurzoru myši při jejím pohybu nad bitmapou. Pokud je bitmapa zobrazena v takovém přiblížení, že formulář zobrazuje pouze její část, lze se po ní pohybovat pomocí pravého tlačítka myši. Pro zobrazení celé bitmapy, je k dispozici v menu tlačítko, které způsobí, že se přibližovací faktor přizpůsbí tak, aby toho bylo docíleno. Další pomůckou, jak zpřesnit zadávání obejktů, je možnost přichytávání vrcholů. V případě, že uživatel definuje objekty, které na sebe navazují, lze pomocí této funkce docílit toho, že definovaný vrchol převezme souřadnice vrcholu, v jehož blízkosti je kliknuto pravým talčítkem myši. Tato funkce je implicitně zapnuta a je nastavitelná globálně pro všechny editační formuláře.

Pokud je v bitmapě vytvořen objekt, který neodpovídá požadavkům uživatele, lze využít tlačítka menu pro smazání vybraného objektu. Veškerá editace nad danou bitmapou probíhá v lokální kopii. Změny jsou tedy promítnuty do projektového souboru až po uložení stavu formuláře, k čemuž je určeno tlačítko v menu.

Při vývoji grafického uživatelského rozhraní aplikace byl kladen důraz na možnost zajistit uživateli co nejefektivnější práci s rozhraním. Jelikož se předpokládá, že bude uživatel často přepínat mezi jednotlivými bitmapami projektu, je pro tuto akci implementována klávesová zkratka. Také většinu akcí, které jsou přístupné přes hlavní menu aplikace, lze



Obrázek 5.4: Editační forulář pro bitmapy.

vyvolat pomocí klávesových zkratek. Jejich seznam spolu s popisem lze nalézt v příloze 7.3 – *Klávesové zkratky grafického uživatelského rozhraní.*

Nastavení a parametry

Jelikož se jedná o poměrně komplexní aplikaci, má uživatel k dispozici několik nastavení. Ta jsou přístupná přes formulář nastavení, který lze vyvolat přes hlavní menu aplikace. Jedním z nich je možnost určit defaultní hodnotu části kódového jména pro grafické objekty definované pro jednotlivé bitmapy. To je následně použito při každém vytvoření objektu a doplněno o příslušný index podle pořadí. Toto kódové jméno muže být později změněno pomocí editačního panelu objektu. Protože lze přes grafické uživatelské rozhraní uložit výsledky evaluace do soborů, je zde implemntovaná možnost nastavení názvu souboru pro uložení detailního vyhodnocení a název souboru pro vyhodnocení souhrnné. Jak již bylo zmíněno dříve, lze pomocí menu editoru bitmapy vyvolat okno zobrazující přiblížené okolí bitmapy u pozice kurzoru myši umístěného nad obrázkem. Uživatel má možnost nastavit pro toto okno jeho výšku a šířku. Také lze nastavit procentuální hodnotu přiblížení tohoto okna.

Další nastavení se týká samotného procesu evaluace. V průběhu specifikace grafického uživatelského rozhraní vznikl požadavek na možnost zvolení, které bitmapy z aktuální sady v projektu mají být zahrnuty do procesu evaluace. Proto je toto nastavení implementováno v aplikaci a umožňuje uživateli určit ze seznamu bitmap ty, které mají být vyhodnoceny. Seznam bitmap v pravé části aplikace díky tomuto nastavení umožňuje filtorvat seznam bitmap tak, aby zobrazoval všechny bitmapy, bitmapy určené pro evaluaci nebo ty, které nejsou určeny pro vyhodnocování. Uživatel má také možnost určit, jaké typy objektů mají

být zahrnuty do evaluace. Aplikace umožňuje definovat dva typy objektů – trojúhelník a čtyřúhlý polygon. Implementováno je ovšem vyhodnocování pouze pomocí trojúhelníku, proto je k dipozici pouze nastavení potřebné pro evaluaci tohoto typu. Jedná se o velikost strany referenčního rovnostranného trojúhelníka a hodnotu tolerance při určování shodnosti detekovaných bodů. Popis implementace vyhodnocování za použití trojúhelníku je popsán v sekci 5.3 – *Proces evaluace*.

5.2 Konzolová aplikace

Halvním požadavkem na konzolovou aplikaci byla možnost jejího použití v automatizovaném procesu návrhu/učení algoritmů vyhledávání klíčových bodů. Z tohoto důvodu bylo nutné zajistit, aby při běhu aplikace nebyla potřeba interakce ze strany uživatele. Ke spušťení aplikace je nutné určit minimálně jeden parametr. Jedná se o umístění projektového souboru, který je vytvořen pomocí aplikace s grafickým uživatelským rozhraním. Z tohoto souboru jsou získány údaje o tom, jaké bitmapy mají být zahrnuty do procesu evaluace a jednotlivá nastavení. Spolu s projektovým projektem musí být k dipozici i soubory `.cdet` s definicí detekovaných bodů pro vyhodnocované bitmapy.

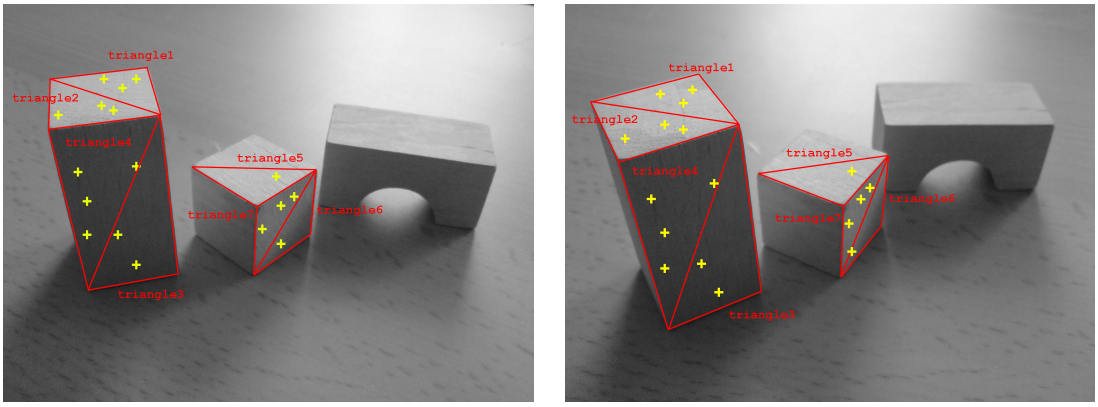
Při spuštění konzolové aplikace jsou nejdříve zpracovány parametry zadané přes konzoli a případné nastavení patřičných hodnot. Seznam všech parametrů lze nalézt v příloze 7.4 – *Parametry konzolové aplikace*. Následně jsou načtena data z projektového souboru a souborů `.cdet`. Jelikož může proces evaluace v závislosti na počtu vstupních dat trvat delší časový úsek, což by mohlo vést k domnění, že aplikace neodpovídá, je zajištěn průběžný výpis prováděných operací na standardní výstup. Tuto funkčnost zajišťuje modul pro logování, který komunikuje s modulem pro evaluaci. V případě, že dojde při procesu k chybě, je tato skutečnost zaznamenána na standardní chybový výstup. Zároveň je celý proces evaluace zaznamenáván do logovacího souboru, pro případnou pozdější kontrolu. Tento soubor je generován do stejného umístění, ze kterého je aplikace spuštěna. Implicitně nese název `evaluation.log`.

Po úspěšném dokončení evaluace jsou vygenerovány do adresáře aktuálního projektu dva výstupní soubory. Jedná se o souhrnné informace evaluace a o detailní informace o evaluaci. Souhrnné informace obsahují záznam o počtu nalezených bodů v jednotlivých objektech dvojice zkoumaných bitmap a počtu bodů, které byly označeny za shodné. Standardní název tohoto výstupního souboru je `eval_summary.txt`. Data detailní evaluace obsahují pro každý odpovídající objekt v každé dvojci bitmap seznam všech indexů detekovaných bodů s informací o tom, které body byly označeny za totožné. Tento soubor má implicitní název `eval_details.txt`.

5.3 Proces evaluace

Proces evaluace detektorů klíčových bodů za použití vyvíjeného software je rozdělen do dvou fází. První fáze představuje definici dodatečných informací potřebných pro samotný proces evaluace. Pro tento účel slouží aplikace s grafickým uživatelským rozhraním. Uživatel musí nejdříve vytvořit nový projekt. Při vytvoření projektu je specifikován jeho název a umístění. Následně je v daném umístění vytvořen projektový soubor s daným názvem. Jelikož se jedná o xml dokument, je přípona projektového souboru `.xml`.

Další akce spočívá v importování sady bitmap do projektu. Tato sada reprezentuje pohledy na tutéž scénu v různých úhlech pohledu. Pro každou bitmapu, která má být



(a) (b)

Tabulka 5.1: **Definice odpovídajících trojúhelníků v bitmapách:** (a) bitmap1, (b) bitmap2.

přidána, musí existovat odpovídající soubor se specifikací detekovaných bodů. Tento soubor má příponu `.cdet` a jeho struktura musí odpovídat specifikaci, která byla stanovena na UPMG[1]. Při procesu importování jsou zkopírovány soubory bitmap a odpovídající soubory `.cdet` do adresáře projektu.

Pro každou bitmapu v projektu má uživatel možnost definovat objekty potřebné pro evaluaci. Tyto objekty jsou identifikovány pomocí unikátního kódového názvu v rámci dané bitmapy. Uživatel definuje objekty takovým způsobem, aby ohraničovaly určitou skupinu detekovaných klíčových bodů v bitmapě a aby si objekty se stejným kódovým jménem v jednotlivých bitmapách odpovídaly. Úkolem je dodržet při definici objektů co největší možnou přesnost, aby byla do procesu evaluace zanesena co možná nejmenší chyba. Ukázku definice trojúhelníků ve dvojici bitmap lze vidět na obrázcích v tabulce 5.1. Po dokončení této fáze obsahuje projektový soubor potřebné informace pro fázi druhou – vlastní proces evaluace.

Při procesu evaluace se využívají pouze data z projektového souboru a příslušných `.cdet` souborů, neboť již není potřeba grafické reprezentace bitmap. Pro každou dvojici bitmap z projektového souboru se postupně vyhodnocují jednotlivé definované objekty. Vyhodnocení spočívá v převodu detekovaných klíčových bodů, které jsou ohraničeny objekty, do stejného souřadného systému. Příklad je uveden na obrázcích v tabulce 5.2 pro dříve zmiňované trojúhelníky. Nyní je každý bod z jedné bitmapy porovnán s bodem z bitmapy druhé. Porovnání probíhá na základě Euklidovské vzdálenosti. Pro body $A = (x_A, y_A)$ a $B = (x_B, y_B)$ je tedy vzdálenost d_{AB} rovna

$$d_{AB} = \sqrt[2]{(x_A - x_B)^2 + (y_A - y_B)^2}. \tag{5.1}$$

Nechť t je práh stanovený uživatelem určující, zda si dva zkoumané body odpovídají a d_{min} minimální vzdálenost aktuálního bodu od kteréhokoliv bodu z druhé bitmapy. Potom pro body, které si odpovídají, platí

$$d_{AB} < t \wedge d_{AB} = d_{min} \tag{5.2}$$

[1]Specifikace struktury je uvedena v příloze.

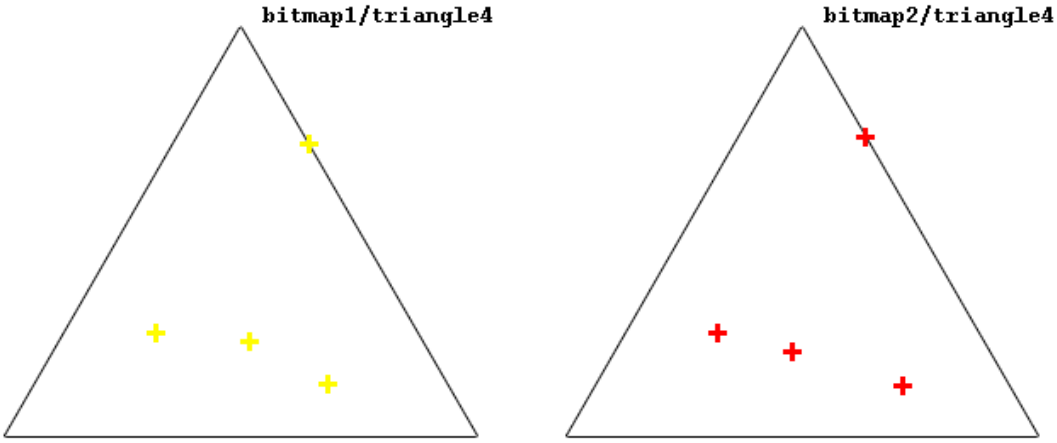
a pro body, které si neodpovídají

$$d_{AB} \geq t \vee d_{AB} \neq d_{min}. \tag{5.3}$$

Množinu dvojic bodů P, které splňují danou podmínku, lze určit vztahem

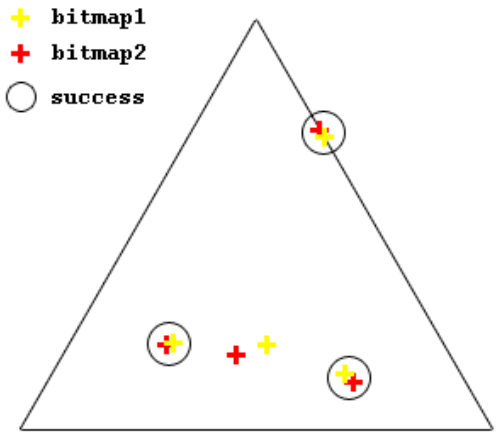
$$P = \{(A, B) \mid d_{AB} < t \wedge d_{AB} = d_{min}\}. \tag{5.4}$$

V opačném případě bodu neodpovídá žádný z nalezených bodů v druhé bitmapě, což se negativně projeví na celkovém hodnocení úpěšnosti. Takto se postupně prochází každá dvojice bitmap projektu spolu se všemi objekty v nich definovanými. Výsledek evaluace je uložen do výstupních souborů podle nastavení uživatele.



(a) (b)

Tabulka 5.2: **Porovnání vybraného trojúhelníku – triangle4:** (a) bitmap1, (b) bitmap2.



Obrázek 5.5: Společný souřadný systém.

Kapitola 6

Závěr

Cílem práce bylo seznámení se s programovacím prostředím Microsoft .NET Framework a problematikou vyhledávání klíčových bodů v obraze. Na základě těchto znalostí bylo požadováno vytvoření metodiky vyhodnocování úspěšnosti detektorů klíčových bodů v obraze a její následné použití při vývoji aplikace sloužící k evaluaci detektorů v iteračních postupech návrhu/učení algoritmů. Tento cíl byl splněn.

Jednotlivé body zadání byly splněny následujícím způsobem:

1. Seznámil jsem se s problematikou vyhledávání klíčových bodů v obraze a aplikacemi využívajícími klíčové body prostudováním dostupné literatury na toto téma. Tyto poznatky jsou shrnuty v kapitole 2 – *Microsoft .NET Framework* a kapitole 3 – *Vyhledávání klíčových bodů v obraze.*

2. Byla navržena metodika pro evaluaci úspěšnosti vyhledávání klíčových bodů v obraze. Její popis je uveden v kapitole 5 část 5.3 – *Proces evaluace.*

3. Možnosti implementace evaluačního software jsou shrnuty v kapitole 4 – *Požadavky na evaluaci detektorů klíčových bodů v obraze.*

4. Implementoval jsem konzolovou aplikaci pro evaluaci metod vyhledávání klíčových bodů v obraze a aplikaci s grafickým uživatelským rozhraním pro definici podmínek evaluace. Funkčnost byla demostrována na dodané sadě dat a výsledky vyhodnoceny.

5. Dosažené výsledky a další možnosti rozšíření jsou zahrunty v této kapitole.

Díky řešení diplomové práce jsem si osvojil způsob programování pod platformou Microsoft .NET Framework, to, jaké prostředky nabízí programátorovi, a zvládl jsem jisté techniky, které usnadňují vývoj aplikace pod touto platformou. Zároveň jsem si rozšířil znalosti v oblasti počítačového zpracování obrazu. Implementace software pro vyhodnocování úspěšnosti detektorů klíčových bodů byla provedena po dohodě s Ústavem počítačové grafiky a multimedií, jelikož nemá takovýto software k dispozici. Byl tak vytvořen prostředek pro zdokonalování algoritmů, které jsou pod tímto ústavem vyvíjeny. Je pravděpodobné, že pro interní účely firem zabývající se touto tématikou byly vyvinuty obdobné aplikace, ale ty nejsou volně k dispozici.

Při vývoji software jsem se potýkal s problémy při specifikaci grafického uživatelského prostředí. Aby bylo dosaženo co nejefektivnější práce uživatele s aplikací, bylo nutné provést několik fází návrhu ve spolupráci s uživateli. Zároveň nedostatek vstupních dat neumožňoval dostatečně optimalizovat proces evaluace.

Jelikož mě zaujala efektivita, s jakou lze v prostředí Microsoft .NET Framework pracovat, rád bych se této platformě věnoval dále a poznal další její součásti, které jsem při řešení nevyužil.

I když je vytvořená aplikace zcela funkční, je zde připraven prostor pro další rozšíření, která by uživateli umožnila zpřesnění a zároveň usnadnění zadávání dodatečných informací potřebných pro evaluaci. Bylo by vhodné například zakomponovat do aplikace s grafickým uživatelským rozhraním detektor hran, který by pomocí bodů označil hrany objektů v bitmapě, a uživatel by měl možnost zapnout přichytávání vrcholů k těmto bodům. Při vývoji aplikace byl brán ohled na možnost jednoduchého rozšíření v případě, že by byl nalezen vhodnější objekt pro porovnávání nalezených bodů. V budoucnu bych se rád pokusil rozšířit modul pro evaluaci o možnost vyhodnocování na základě čtyřúhlého polygonu a zjistil, zda tato metoda vede k přesnějšímu vyhodnocování.

Literatura

[1] ADAMS, M.; GRIFFITHS, I.: *.NET Windows Forms in a Nutshell*. CA, USA: O'Reilly & Associates, Inc., 2003, ISBN 0-596-00338-2.

[2] Bovik, A.: *Handbook of Image and Video processing*. CA, USA: Academic Press, 2000, ISBN 0-12-119790-5, 415–431 s.

[3] DUFFY, J.: *Professional .NET Framework 2.0*. IN, USA: Wiley Publishing, Inc., 2006, ISBN 0-7645-7135-4.

[4] FISHER, R.; PERKINS, S.; WALKER, A.; aj.: *Image processing learning resources explore with Java*. Edinburgh, Great Britain: The University of Edinburgh, 2004, k dispozici na http://homepages.inf.ed.ac.uk/rbf/HIPR2/hipr_top.htm, naposledy navštíveno 6. 5. 2007.

[5] JÄHNE, B.: *Digital Image Processing*. Berlin, Germany: Springer, 2005, ISBN 3-540-24035-7, 331–358 s.

[6] Kolektiv autorů: *MSDN Library*. Microsoft Corporation, 2007, k dispozici na http://msdn2.microsoft.com/en-us/library/default.aspx, naposledy navštíveno 6. 5. 2007.

[7] Kolektiv autorů: *Wikipedia, the free encyklopedia*. FL, USA: Wikimedia Foundation Inc., 2007, k nalezení na http://en.wikipedia.org/wiki/Main_Page, naposledy navštíveno 6. 5. 2007.

[8] LAM, H.; THAI, T. L.: *.NET Framework Essentials*. CA, USA: O'Reilly & Associates, Inc., třetí vydání, 2003, ISBN 0-596-00505-9.

[9] PETZOLD, C.: *Programování Microsoft Windows Forms v jazyce C#*. Brno: Computer Press, a.s., 2006, ISBN 80-251-1058-3.

[10] ROBINSON, S.; NAGEL, C.; WATSON, K.; aj.: *Professional C#*. IN, USA: Wiley Publishing, Inc., třetí vydání, 2004, ISBN 0-7645-5759-9.

[11] SCHMID, C.; MOHR, R.; BAUCKHAGE, C.: *Evaluation of Interest Point Detectors*. Montbonnot, France: INRIA, 2000, k nalezení na ftp://ftp.inrialpes.fr/pub/movi/publications/Schmid-ijcv00.ps.gz, naposledy navštíveno 14.5.2007.

[12] SYMMONDS, N.: *GDI+ Programming in C# and VB .NET*. NY, USA: Apress, 2002, ISBN 1-59059-035-X.

[13] TYNKU, A.; AJOY, R. K.: *Image processing Principles and Applications*. IN, USA: Wiley Publishing, Inc., 2005, ISBN 0-471-71998-6, 132–143 s.

Kapitola 7

Přílohy

7.1 Hiearchie vybraných tříd a jejich popis

V této části je popsána hiearchie vybraných tříd v jednotlivých projektech, jejich účel a popis nejdůležitějších vlastností a metod. V některých případech jsou uvedeny diagramy tříd pro lepší přehled.

Projekt Geometry

Tento projekt zapouzdřuje grafické objekty, které uživatel může definovat v uživatelském rozhraní pro jednotlivé bitmapy. Zároveň obsahuje implementaci třídy `GeometryProvider`, která obsahuje vlastnosti a metody týkající se geometrie.

Třída GeometryAbstractObject

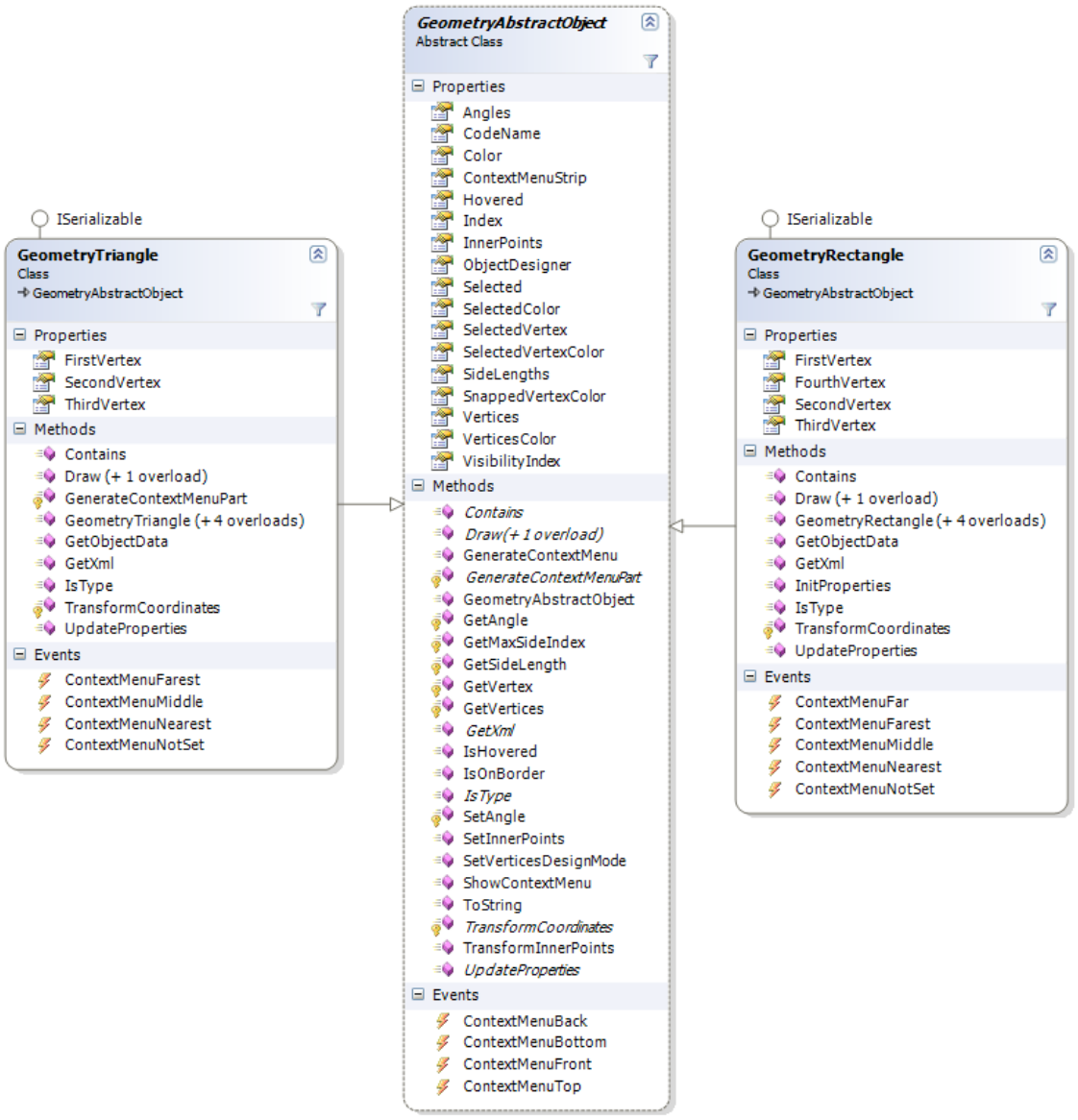
Jedná se o abstraktní třídu, která definuje společné vlastnosti a metody pro dědící třídy `GeometryRectangle` a `GeometryTriangle`, které z této třídy dědí. Reprezentuje obecný objekt, který lze v dané bitmapě vytvořit pomocí grafického uživatelského rozhraní. Každý objekt má definovanou vlastnost `CodeName`, která je unikátní v rámci jedné bitmapy a pomocí které lze objekt identifikovat. Dále je zde definována vlastnost `Index`, aby bylo možné zajistit generování unikátního kódového jména objektu. Vlastnost `VisibilityIndex` slouží k určení, v jaké vrstvě viditelnosti se objekt nachází.

Pro každý objekt je nutné před evaluací určit, které z detekovaných bodů jsou uvnitř tohoto objektu. Seznam těchto bodů je uložen ve vlastnosti třídy `InnerPoints`. Tato vlastnost je inicializována pomocí metody `SetInnerPoints` až před samotným procesem evaluace, jelikož během definování objektů není tato informace potřebná. Proces evaluace spočívá v převedení objektů do stejného souřadného systému, tudíž zde existuje abstraktní metoda `TransformCoordinates`, jejíž implementace je ponechána na dědící třídě. Zároveň třída obsahuje řadu vlastností a metod potřebných pro matematické výpočty při převodu souřadnic.

K zajištění správného zobrazování objektů je zde definována vlastnost `ObjectDesigner`, jejíž nastavení je reflektováno do veřejných vlastností třídy. Zároveň třída obsahuje vlastnost `Selected`, která určuje, zda je daný objekt vybrán, a vlastnost `Hovered`, určující, zda se kurzor myši nachází nad daným objektem. Metoda `SetVerticesDesignMode` zajišťuje nastavení grafické reprezentace všech vrcholů objektu. K určení toho, zda se kurzor myši nachází nad některou ze stran objektů, slouží metoda `IsOnBorder`. Tato metoda je zároveň

použita pro metodu `IsHovered`, která určuje, zda se kurzor myši nachází nad stranou objektu, nebo nad grafickou reprezentací některého z vrcholů objektu. Samotné vykreslování objektu u dědící třídy zajišťuje abstraktní metoda `Draw`.

Jelikož každý grafický objekt vlastní kontextové menu, je v této třídě implementována metoda `ShowContextMenu`, kteá slouží k zobrazení menu. Součástí je také abstraktní metoda `GenerateContextMenuPart`, která zajišťuje vygenerování části menu závislého na typu objektu. K určení, zda zadaný bod leží uvnitř objektu, slouží abstraktní metoda `Contains`. Hierairchii tříd a seznam všech vlastností tříd a metod lze vidět na obrázku 7.1.



Obrázek 7.1: Diagram tříd pro geometrické objekty.

Třída GeometryRectangle

Jeden z typů objektů, které je možné v bitmapě definovat je čtyřúhlý polygon reprezentovaný touto třídou. Tato třída zpřístupňuje jednodušší cestou jednotlivé vrcholy objektu pomocí vlastností `FirstVertex`, `SecondVertex`, `ThirdVertex` a `FourthVertex`. Dále zajišťuje implementaci zděděné abstraktní metody `TransformCoordinates`, `Contains` a implementaci metody `Draw`, která slouží k vykreslení objektu v závislosti na aktuálním nastavení prostředí.

Třída GeometryTriangle

Tato třída reprezentuje další z typů, které může uživatel definovat v dané bitmapě. I tato třída dědí z abstraktní třídy `GeometryAbstractObject` a zpřístupňuje přes své vlastnosti jednotlivé vrcholy objektu `FirstVertex`,`Secondvertex` a `ThirdVertex`. I zde je zajištěna implementace zděděných metod `TransformCoordinates`, `Contains` a `Draw`.

Třída GeometryAbstractPoint

Tato abstraktní třída reprezentuje obecný geometrický bod. Obsahuje vlastnosti třídy pro reprezentaci souřadnic bodu `X` a `Y` a abstraktní metodu `IsHovered`, která určuje, zda je kurzor myši nad daným bodem. Dále je zde abstraktní metoda `GetXml`, která slouží k získání xml reprezentace bodu a abstraktní metoda `IsHovered` určující, zda je kurzor myši nad souřadnicemi bodu.

Třída GeometryPoint

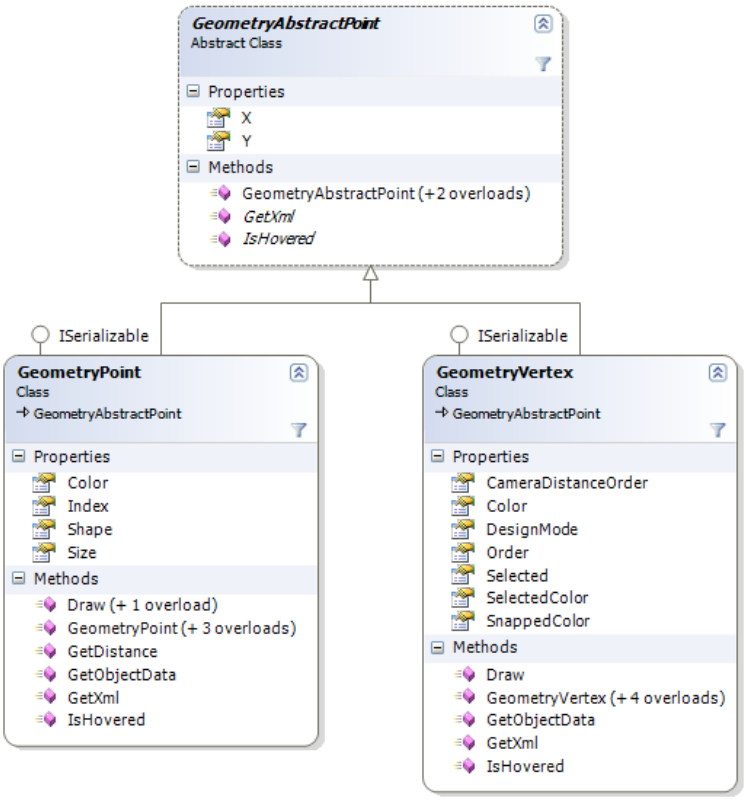
Třída `GeometryPoint` je určena pro gafickou reprezentaci detekovaných bodů vyhledávačem a pomocných bodů určujících vrcholy při vytváření grafických objektů v bitmapě. Dědí abstraktní třídu `GeometryAbstractPoint` a rozšiřuje ji o některé další vlastnosti a funkcionalitu. Součástí třídy je vlastnost `Index`, která reprezentuje index pořadí bodu. Dále obsahuje privátní proměnnou `mPointDesigner`, která obsahuje informace o vzhledu bodu. Její nastavení je opět reflektováno ve veřejných vlastnostech třídy. Třída také implementuje metodu `GetDistance`, která slouží k určení vzdálenosti aktuálního bodu od bodu zadaného.

Třída GeometryVertex

Jedná se o třídu reprezentující vrchol grafického objektu. Také tato třída dědí z abstraktní třídy `GeometryAbstractPoint`. Děděnou třídu rozšiřuje o vlastnosti `Order` určující pořadí vrcholu v objektu a `CameraDistanceOrder` určující vzdálenost vrcholu od kamery. I v této třídě se nachází privátní proměnná `mVertexDesigner` obsahující informace o designu vrcholů. Její nastavení je opět reflektováno odpovídajícími vlastnostmi třídy. Spolu s touto informací o vzhledu vrcholu je zde ještě vlastnost `DesignMode`, která slouží k nastavení vzhledu vrcholu při různých grafických módech zobrazení. Hierarchii tříd zobrazuje obrázek 7.2.

Třída GeometryProvider

Jedná se o statickou třídu zajišťující obecné metody a vlastnosti týkající se geometrie. Hlavní vlastnost této třídy je `RefferenceTriangle`, jedná se o objekt `GeometryTriangle` reprezentující referenční trojúhelník v jednotném souřadném systému, do kterého jsou



Obrázek 7.2: Diagram tříd pro geometrické body.

převáděny vyhodnocující trojúhelníky. Pro nastavení referenčního trojúhelníku slouží metoda `SetRefferenceTriangle`. Díky tomu jsou zajištěny implementační požadavky kladené na tento trojúhelník. Mezi další důležité metody implementovány v této třídě patří `GetDistance` sloužící k určení vzdálenosti dvou bodů.

Bylo nutné udržovat informaci o vzhledu jednotlivých grafických objektů. K tomuto účelu slouží třídy zobrazené na obrázku 7.6. Abstraktní třída `AbstractDesigner` seskupuje společné vlastnosti.

Projekt ProjectManager

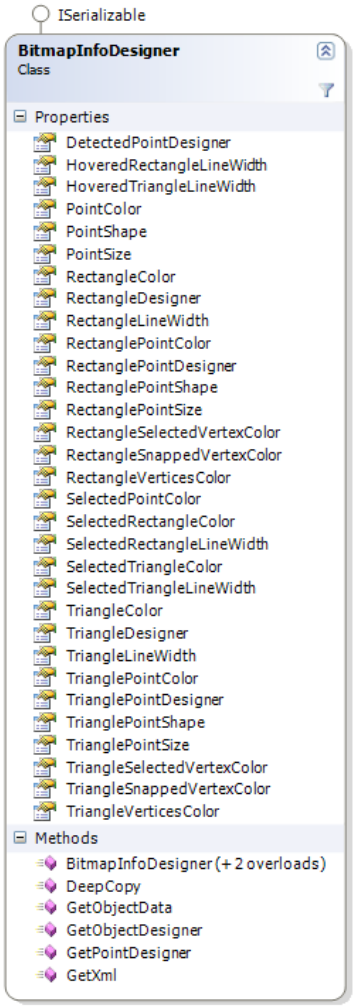
Projekt `ProjectManager` zahrnuje celkem čtyři třídy. Jedná se o třídu `BitmapInfo`, což je třída uchovávající veškeré potřebné informace o bitmapě, dále třída `EvaluatorInfo`, která obsahuje informace o vyhodnocení mezi dvojicí souborů dat daných bitmap. Třetí statická třída `EvaluationProvider` implementuje metody potřebné pro evaluaci a hlavní statická třída `ProjectProvider` slouží pro práci s projektem a uchovává informace o projektu.

Třída BitmapInfo

Jelikož je nutné spolu s bitmapou uchovávat relativně velké množství dodatečných informací, je součástí projektu tato třída. Obsahuje vlastnost `Image`, která reprezentuje odpovídající bitmapu uloženou v paměti a vlastnost `FileName` reprezentující název dané bitmapy. Dále jsou zde k dispozici vlastnosti `Width` a `Height`, které reflektují rozměry

bitmapy. Protože uživatel může v bitmapě definovat několik objektů, je zde k dispozici vlastnost třídy `Objects`, ve které jsou objekty uloženy. Pro přidání nebo odebrání objektu jsou zde implementovány metody `AddObject` a `RemoveObject`. Součástí vstupních dat jsou také body, které byly detekovány vyhodnocujícím detektorem. Jejich seznam je k dispozici prostřednictvím vlastnosti třídy `Points`, která je inicializována pomocí metody `LoadDetectorData`. Aby bylo možné určit pořadí v jakém budou jednotlivé bitmapy vyhodnocovány, je zde vlastnost třídy `Order`.

Aby měl uživatel možnost nastavit vzhled objektů bitmapy, je součástí třídy vlastnost `mDesigner`. Tato vlastnost je ovšem privátní a její jednotlivé nastavení jsou přístupná pomocí vlastností třídy, které tato nastavení reflektují. Diagram třídy je znázorněn na obrázku 7.3.



Obrázek 7.3: Diagram tříd pro designer objektu BitmapInfo.

Třída EvaluatorInfo

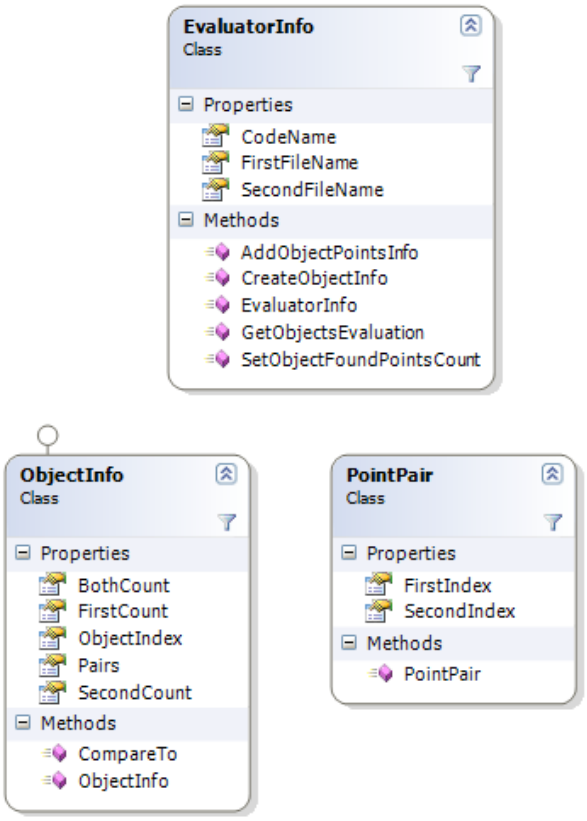
Tato třída vznikla kvůli nutnosti uchovávání dílčích informací o vyhodnocujících dvojicích sad dat v průběhu procesu evaluace. Vlastnost třídy `CodeName` uchovává kódové jméno

pro daný objekt. Aby byla známa informace o tom, ke které dvojci bitmap tento objekt patří, jsou zde dvě vlastnosti třídy `FirstFileName` a `SecondFileName` uchovávající název souboru bitmap. Zároveň třída obsahuje seznamy objektů typu `ObjectInfo` pro jednotlivé geometrické objekty.

Objekt `ObjectInfo` uchovává informaci o celkovém počtu bodů v prvním grafickém objektu `FirstCount`, ve druhém grafickém objektu `SecondCount` a o počtu bodů, které si v obou objektech odpovídají `BothCount`. Tyto informace jsou potřebné pro souhrnné vyhodnocení a nastavují se pomocí metody `SetObjectFoundPointsCount`. Další vlastnost třídy `ObjectIndex` je potřebná k určení příslušného grafického objektu. Pro detailní vyhodnocení obsahuje tato třída vlastnost `Pairs`, která obsahuje dvojice indexů bodů, které byly označeny v objektech za shodné. Pro manipulaci s objektem `ObjectInfo` jsou v třídě `EvaluatorInfo` k dispozici metody `CreateObjectInfo` a `AddObjectPointsInfo`.

Pro získání informací o vyhodnocení pro danou dvojici sad dat je k dispozici metoda `GetObjectsEvaluation`.

Na obrázku 7.4 je zachycen diagram tříd.



Obrázek 7.4: Diagram tříd používaných při vyhodnocování.

Třída EvaluationProvider

Tato statická třída seskupuje veškeré potřebné metody, které jsou použity při procesu vlastní evaluace. Obsahuje hash tabulku objektů `EvaluatorInfo`, se kterými pracuje. Metoda `EvaluateBitmapInfos` slouží k evaluaci všech bitmap obsaežených v projektu. Jednot-

livé objekty jsou vyhodnoceny pomocí metody `EvaluateObjects`. Výsledek vyhodnocení lze získat pomocí metody `GetEvaluation`.

Třída ProjectProvider

Jedná se o hlavní statickou třídu, která uchovává stavové informace týkající se aktuálního projektu a poskytuje metody pro práci s ním. Důvodem vzniku této třídy byla potřeba pracovat s projektem jak v aplikaci s grafickým uživatelským rozhraním, tak v konzolové aplikaci. Diagram třídy je zachycen na obrázku 7.5.

Každý vytvořený projekt je opatřen pro svou indentifikaci kódovým jménem, které je uloženo ve vlastnosti třídy `ProjectCodeName`. Zároveň je potřeba uchovávat informaci o umístění a názvu souboru projektu. Tyto informace jsou k dispozici prostřednictvím vlastností `ProjectFullPath`, `ProjectDirectory` a `ProjectFileName`. Aby bylo možné zjistit stavové informace projektu, je zde vlastnost `ProjectLoaded` určující, zda je projekt načten, dále pak `ProjectModified`, která určuje, zda je projekt modifikovaný.

V projektu je nutné uchovávat informaci o tom, které objekty `BitmapInfo`, nesoucí popis, obsahuje. Proto je zde přítomna privátní hash tabulka `mBitmapInfos`, která tyto informace udržuje. Jelikož bylo nutné zajistit, aby se změny dat projevily i v kontrolech grafického uživatelského rozhraní, bylo nutné implementovat přístup k této hash tabulce prostřednictvím metod. Tak lze zajistit, aby pro každou příslušnou akci byla vyvolána událost, kterou daný kontrol zpracuje. Z tohoto důvodu třída obsahuje deklarace událostí `BitmapInfoAdded`, která je vyvolána při přidání objektu `BitmapInfo` do hash tabulky, a `BitmapInfoDeleted`, která se vyskytne při odebrání objektu z hash tabulky. Mezi metody, které zajišťují tyto akce, patří `SetBitmapInfo` pro přidání nebo aktualizaci, `RemoveBitmapInfo` pro odebrání.

Vytvoření nového projektu K vytvoření projektu slouží metoda třídy `CreateProject`. Pokud daný projekt v cílovém adresáři s daným názvem již existuje, není vytvořen a uživatel je o tom informován. V opačném případě je zajištěno vytvoření projektového souboru. Tento soubor má formát xml dokumentu a obsahuje po vytvoření nového projektu již kopii skeletonu struktury, který je uložen jako privátní konstanta `XML_SKELETON` ve třídě `ProjectProvider`. Zároveň jsou inicializovány odpovídající vlastnosti této třídy.

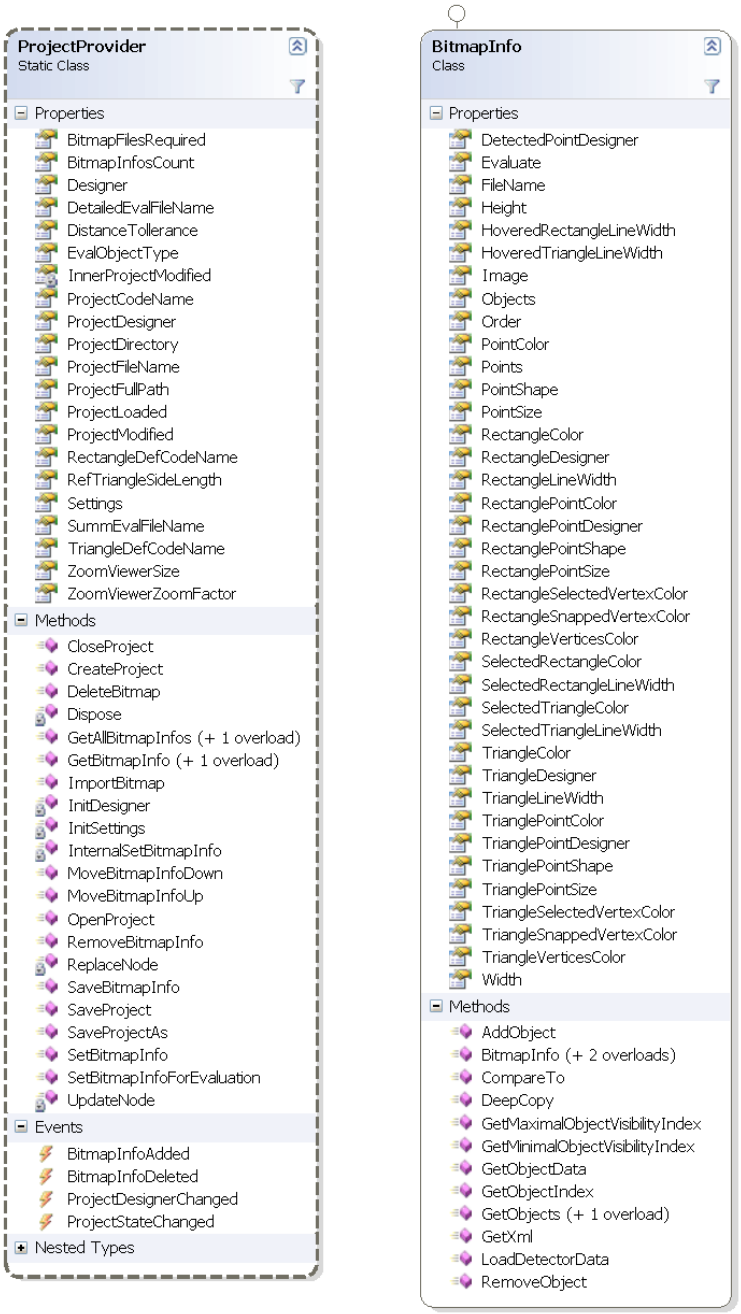
Xml skeleton nového projektu:

```
<project codename=„<project_codename>\ path=„<project_path>\>
    <settings></settings>
    <design></design>
    <bitmapinfos></bitmapinfos>
</project>
```

Uložení projektu Tuto akci zajišťuje metoda `SaveProject`. Jejím cílem je vytvořit reprezentaci aktuálního stavu projektu ve formě xml řetězce, který je následně uložen do projektového souboru. Kvůli této akci je ve většině objektů obsahujících dílčí projektové informace implementována metoda `GetXml`, pomocí které je vygenerován řetězec reprezentující xml schéma daného objektu. Při uložení projektu se tedy projdou všechny tyto relevantní objekty.

Třídy implementující metodu `GetXml`:

- `BitmapInfo`



Obrázek 7.5: Diagram tříd uchovávajících informace o projektu a jednotlivých bitmapách.

- `GeometryTriangle`
- `GeometryRectangle`
- `GeometryPoint`
- `GeometryVertex`
- `ObjectDesigner`

- `PointDesigner`

Uložení objektu BitmapInfo v projektu Kromě možnosti uložení kompletního projektu je zde implementováno uložení jednotlivých objektů `BitmapInfo` reprezentující dodatečné informace k dané bitmapě. Metoda pro tuto akci se jmenuje `SaveBitmapInfo`. Jelikož je při uložení nutné nahradit pouze část projektového souboru, a tudíž část xml dokumentu, je pro tuto potřebu implementována privátní metoda `ReplaceNode`, která volá rekurzivní metodu `UpdateNode`. Ta prochází rekurzivně xml dokument a hledá uzel s odpovídajícím názvem. V případě úspěšného nalezení je tento uzel nahrazen novým s aktuálními daty.

Otevření existujícího projektu Otevření existujícího projektu zajišťuje metoda nesoucí název `OpenProject`. Inicializuje vlastnosti třídy `ProjectProvider` na základě informací získaných z projektového souboru. Aby bylo možné zajistit vytvoření všech relevantních objektů pro aktuální projekt, bylo nutné pro tyto objekty implementovat konstruktor, který z předaného uzlu xml dokumentu vytvoří instanci tohoto objektu.

Správa chyb Při většině akcí, které jsou v aplikaci prováděny, se může vyskytnout okolnost, díky které daná akce nemůže být úspěšně provedena. Proto je nutné zajistit nějakým způsobem možnost, jak o daném problému informovat uživatele. V aplikaci je využito systému výjimek, které jsou v Microsoft .NET k dispozici. Každá výjimka s sebou nese zprávu, která popisuje, proč k dané chybě došlo. Toho lze využít k přenosu chybových hlášení pro uživatele. Proto aplikace deklaruje několik typů vlastních výjimek, které jsou v případě chyby příslušně zpracovány. Pro zobrazování chybových hlášek je využit projekt `ErrorManager`, který je popsán později v této kapitole.

Typy vlastních výjimek:

- `DataNotFoundException` označuje chybu nenalezení příslušných dat.
- `ObjectNotCreatedException` reprezentuje nevytvoření požadovaného objektu.
- `ObjectExistsException` informuje o tom, že daný objekt již existuje.

Implementace hluboké kopie objektu V některých případech použití bylo nutné zajistit u objektů vytvoření hluboké kopie, aby nebyla předávána reference na objekt a nedocházelo k jeho nechtěné modifikaci. Jelikož Microsoft .NET Framework standardně nemá k dispozici tuto možnost, bylo toto potřeba zajistit vlastní implementací. Tato funkcionalita je zajištěna pomocí serializace a deserializace. Proces serializace znamená uložení instance objektu do prezistentního úložiště, v tomto případě do paměťového streamu. Naopak proces deserializace je převod ze streamu zpět na instanci objektu. Aby bylo možné využít serializace a deserializace při vytvoření hluboké kopie objektu, musí tento objekt implementovat rozhraní `ISerializable`, které zahrnuje implementaci minimálně konstruktoru pro získání instance objektu ze streamu a metodu `GetObjectData` sloužící pro uložení do streamu.

Projekt Controls

Projekt `Controls` seskupuje vlastní kontroly, které byly implementovány v rámci aplikace z důvodu nevhodnosti a nedostatků standardních kontrolů platformy Microsoft .NET Framework. Uvedeny jsou pouze komplexnější kontroly.

Kontrol MyPictureBox

Microsoft .NET Framework obsahuje standardní kontrol `PictureBox` pro možnost zobrazování obrázků. Díky své omezené funkcionalitě a neefektivnímu zobrazování rozměrných obrázků nebyl použit. Místo toho byl implementován vlastní kontrol `MyPictureBox`, který zajišťuje efektivní vykreslování obrázku díky tomu, že je vždy vykreslována pouze viditelná oblast bitmapy. Zároveň je zajištěna možnost změny přibližovacího faktoru pomocí skrolovacího kolečka myši nebo externě pomocí vlastnosti kontrolu `ZoomFactor` nebo metod `ZoomIn` a `ZoomOut`. K upravení přibližovacího faktoru na takovou hodnotu, aby se obrázek vešel do celé plochy kontrolu, je implementována metoda `FitToScreen`.

Kontrol implementuje dva módy pro pravé tlačítko myši. První mód zajišťuje pohyb po bitmapě v případě, že je zobrazena pouze její část. Druhý mód slouží k definování části bitmapy, která má být přiblížena.

Kontrol BitmapViewer

Jedná se o kotrol reprezentující editační formulář pro jednotlivé bitmapy. Pro zobrazování využívá vlastní kontrol `MyPictureBox` spolu s jeho funkcemi. Jelikož je kontrol primárně určený pro definici objektů v obrázku, poskytuje prostřednictvím rozhraní uživateli možnost vytvářet, mazat a editovat objekty. Kontrol pracuje ve dvou módech. První mód slouží k vytváření objektů. V tomto módu lze pomocí levého kliknutí myši specifikovat vrchol vytvářeného objektu. Tyto body jsou uchovávány dočasně v privátní vlastnosti kontrolu `mPointBuffer`. V případě, že dojde k jejímu naplnění, je automaticky vytvořen objekt a lze definovat další. Druhý mód soluží k editaci již existujících objektů. To je docíleno kliknutím levého talčítka myši na jeden z vrcholů objektu. Tím dojde k jeho uchopení a lze s ním pohybovat po bitmapě. veškeré změny provedené během editace jsou uchovávány v lokální kopii objektu `BitmapInfo`. Tím je docílena možnost dodatečného uložení změn nebo návrat k původnímu stavu. Informace o tom, zda došlo ke změnám je uchovávána ve vlastnosti `ChangesMade`.

V aplikaci je potřeba zajistit, aby kontrol měl možnost komunikovat s nadřazeným MDI formulářem. Toho je docíleno pomocí vlastních událostí kontrolu, které jsou vyvolány kontrolem a následně obslouženy formulářem při každé potřebné akci. Zároveň je zde implementováno několik metod, které slouží k externímu ovládání kontrolu. Mezi nejdůležitější patří `SetSelectedObjectAndVertex` umožňující externě nastavit vybraný objekt a jeho vrchol, `SaveBitmapInfo` pro uložení aktuálního stavu editoru a `HandleKeyUp` zajišťující funkčnost klávesových zkratek.

Kontrol ZoomViewer

`ZoomViewer` je kontrol ve formě formuláře, který slouží k detailnějšímu zobrazení okolí kolem kurzoru myši nad danou bitmapu. To zajistí uživateli přesnější určení vrcholu objektů, který právě definuje. K zobrazení bitmapy využívá již zmíněný vlastní kontrol `MyPictureBox`. Kontrolu lze nastavit velikost zobrazovaného okna a přibližovací faktor. Pozici kurzoru v bitmapě zobrazuje v tomto kontrolu kříž, jehož barva je také nastavitelná.

Kontrol ImageListBox

V grafickém uživatelském rozhraní aplikace je nutné zobrazit seznam bitmap obsažených v aktuálním projektu, aby měl uživatel přehled, jaké jsou zahrnuty v aktuálním projektu.

Ze standradních kontrolů se nabízel kontrol `ListBox`, který poskytoval možnost zobrazit seznam názvů bitmap. Tento seznam by pro uživatele ovšem neměl velkou vypovídající hodnotu. Proto byl implementován vlastní kontrol `ImageListBox`, který k názvu souboru zároveň zobrazuje náhled dané bitmapy a přidává funkcionalitu kontrolu.

Pro jednotlivé položky seznamu byla implementována třída `ImageLixtBoxItem`, která dědí z třídy `ListBoxItem`. Navíc implementuje vlastnosti `Text` pro zadání názvu bitmapy, `ItemBitmap` obsahující bitmapu, ze které je generován náhled, a `Height` určenou k zadání výšky položky. Samotný kontrol `ImageListBox` dědí standardní kotrol `ListBox` využívající jeho funkčnost, ale zajišťuje vlastní vykreslování jednotlivých položek. Aby při vykreslování nedocházelo k problikávání, je použit backbuffer. Do něho je nejdříve vykreslena položka a následně je teprve kompletní zobrazena na displej. Kontrol navíc rozšiřuje funkčnost o možnost posunu položek v seznamu, aby bylo možné určit pořadí vyhodnocení jednotlivých bitmap.

Součástí kontrolu jsou i dvě metody `MoveSelectedItemUp` a `MoveSelectedItemDown`, které zajišťují možnost pohybu položek v se znamu. Metoda `SelectItem` zajišťuje možnost vybrání položky na základě názvu dané bitmapy.

Projekty Designers a Enumerations

Pro grafické objekty, které jsou vykreslovány v rámci grafického uživatelského rozhraní, bylo nutné sjednotit definici jejich vzhledu. Proto byl vytvořen projekt `Designers`, který sdružuje třídy definující vzhled jednotlivých objektů. Součástí projektu `Designers` je abstraktní třída `AbstractDesigner`, která obsahuje společné vlastnosti pro ostatní třídy pro vzhled. Ta je děděna třídami `ObjectDesigner`, což je designer pro obecný grafický objekt, dále pak `PointDesigner` definující vzhled grafického bodu a `VertexDesigner` určený pro uchování designu vrcholu objektu. Nastavení designu pro objekt `BitmapInfo` je uchováváno v `BitmapInfoDesigner`. Jednotlivé diagramy tříd zobrazuje obrázek 7.6.

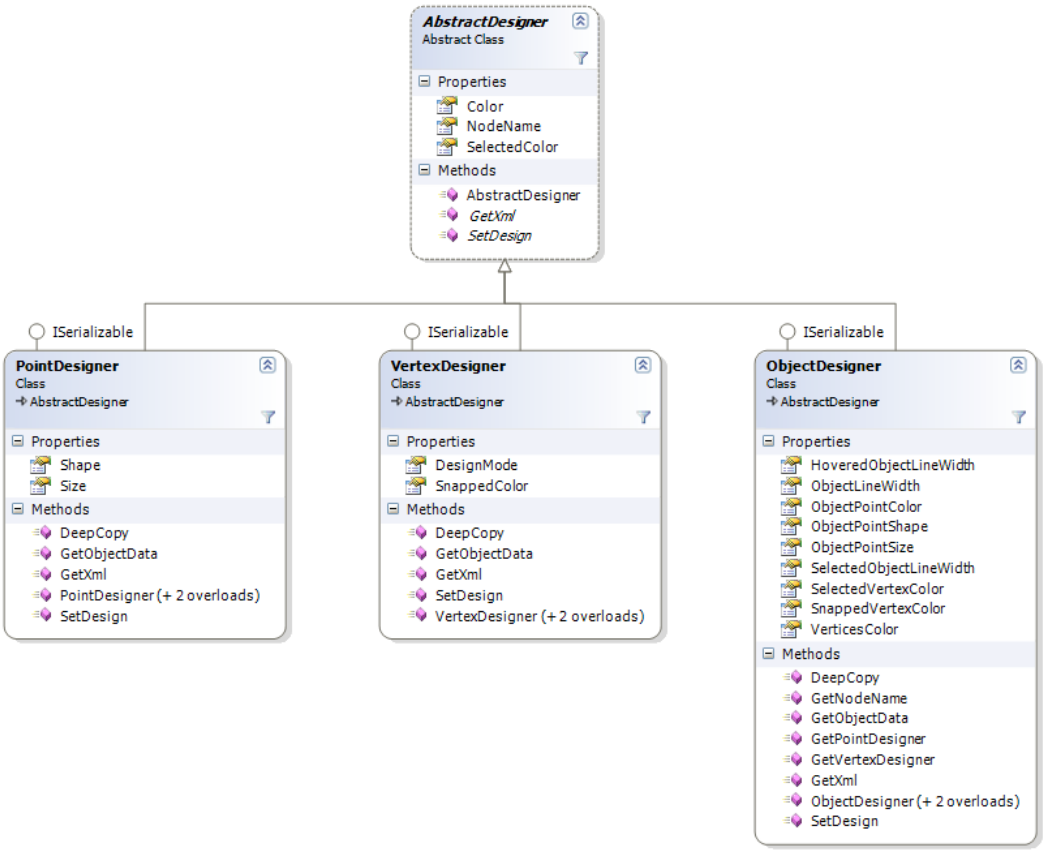
Projekt `Enumerations` obsahuje veškeré výčtové datové typy, které jsou v aplikaci použity. Dále je zde statická třída `EnumFunctions`, která implementuje metody potřebné pro převod některých těchto výčtových typů.

Seznam výčtových datových typů:

- `ObjectComparationEnum` definuje způsoby vzájemného porovnání grafických objektů.
- `ObjectTypeEnum` určuje typ grafického objektu.
- `PointShapeEnum` reprezentuje grafickou podobu zobrazovaného bodu.
- `VertexCameraDistanceEnum` je výčet určující v jaké vzdálenosti se vrchol objektu nachází od pozice kamery.
- `VertexDesignModeEnum` určuje jaký designový mód je aktuálně pro daný vrchol objektu nastaven.

Projekty ResManager a ErrorManager

`ResManager` je projekt zahrnující třídu `ResHelper`, která zajišťuje získávání textových řetězců ze standardního souboru s prostředky, který je pro každý projekt k dispozici. Tím je zajištěn snadný způsob případného převodu aplikace do jiné jazykové verze. Zároveň je tak sjednoceno umístění všech řetězců použitých v aplikaci, což umožňuje jejich jednoduchou správu.



Obrázek 7.6: Diagram tříd pro designery.

Jak již bylo zmíněno v této kapitole, všechny chybové akce, které vzniknou při práci s aplikací, jsou signalizovány prostřednictvím výjimek. Metody zajišťující uživateli zobrazování informací, které jsou v těchto výjimkách uchovány, jsou implementovány ve třídě `ErrorHelper`, která je součástí projektu `ErrorManager`.

Projekt EvaluatorConsole

Tento projekt zahrnuje implementaci konzolové aplikace a implementaci třídy textttLogProvider, která se stará o logování akcí při běhu aplikace. Konzolová aplikace využívá dříve zmíněné providery pro operace s projektovým souborem a pro vlastní evaluaci. textttLogProvider zajišťuje jak průběžné výpisy o prováděných akcích na konzoli, tak vytváření vlastního log souboru.

Projekt EvaluatorGUI

Projekt textttEvaluatorGUI představuje vlastní aplikaci s grafickým uživatelským rozhraním. Jeho součástí je kontrol reprezentující hlavní formulář aplikace. Je zde implementována veškerá jeho funkčnost.

7.2 Popis formátu výstupních dat vyhledávačů

```
N
m
u1 v1 a1 b1 c1 d1,1 d1,2 d1,3 ... d1,N
    :
    :
um vm am bm cm dm,1 dm,2 dm,3 ... dm,N
Processing\_Time\_MS

Legenda:

N                     ... délka descriptoru (feature vector, koeficienty
                          d1,1 d1,2 d1,3 ... d1,N)
m                     ... počet nalezených oblastí/bodů v obrázku
u, v                  ... střed oblasti/bodu (detekované souřadnice)
a, b, c               ... koeficienty autokovarianční matice (matice
                          druhých mometů v pořadí a1,1, a1,2, a2,2) oblasti
d                     ... koeficienty descriptoru
Processing\_Time\_MS  ... čas zpracování souboru v milisekundách
```

7.3 Klávesové zkratky grafického uživatelského rozhraní

```
Ctrl + N              ... Vytvoření nového projektu
Ctrl + O              ... Otevření projektu
Ctrl + S              ... Uložení projektu
Ctrl + Shift + S      ... Uložení projektu pod jiným názvem
Ctrl + A              ... Uložení všech otevřených editorů
Ctrl + I              ... Import bitmap do aktuálního projektu
Ctrl + C              ... Zavření aktuálního projektu
Ctrl + X              ... Ukončení aplikace
Ctrl + Shift + V      ... Zapnutí/Vypnutí přichytávání k vrcholům objektů
Ctrl + E              ... Spuštění procesu evaluace
Ctrl + T              ... Vyvolání nastavení aktuálního projektu
Ctrl + Up             ... Otevření/Vybrání následující bitmapy směrem
                          nahoru od aktuální v seznamu
Ctrl + Down           ... Otevření/Vybrání následující bitmapy směrem
                          dolu od aktuální v seznamu
Ctrl + Shift + Up     ... Posun aktuální bitmapy v seznamu směrem nahoru
Ctrl + Shift + Down   ... Posun aktuální bitmapy v seznamu směrem dolu
Ctrl + Tab            ... Přepínání mezi otevřenými editory
```

Kliknutí levého tlačítka myši na seznamu bitmap způsobí její výběr a otevření editačního formuláře. Kliknutí levého tlačítka myši spolu se stisknutou klávesou `Ctrl` pouze tuto bitmapu označí.

Klávesové zkratky pro aktivní editační formulář

```
Ctrl + T    ... Vypnutí/Zapnutí kreslení trojúhelníka
Ctrl + R    ... Vypnutí/Zapnutí kreslení polygonu
Ctrl + S    ... Uložení aktuálního stavu editoru
Del         ... Smazání vybraného objektu
Ctrl + P    ... Vypnutí/Zapnutí zobrazení detailních informací
                o detekovaných bodech
Ctrl + Z    ... Zobrazení/Skrytí detailnějšího zobrazení okolí kurzoru
Ctrl + F    ... Přizpůsobení bitmapy do plochy editoru
Ctrl + H    ... Přepnutí módu pravého tlačítka myši
Ctrl + +    ... Přiblížení bitmapy
Ctrl + -    ... Oddálení bitmapy
```

Kliknutí levého tlačítka myši nad hranou či vrcholem objektu v editoru způsobí jeho vybrání. Pomocí pravého tlačítka myši je vyvoláno kontextové menu pro editaci objektu.

7.4 Parametry konzolové aplikace

Pro každý parametr konzolové aplikace existuje jeho zkrácená a nezkrácená forma. Seznam jednotlivých parametrů je uveden v tabulce 7.1.

Název:hodnota	**Popis**
`e|eval:all|summary|detailed`	Nastavení typu evaluace
`o|objects:all|triangle|rectangle`	Nastavení typu objektů pro evaluaci
`sf|summaryfile:<název souboru>`	Výstupní soubor pro obecnou evaluaci
`df|detailedfile:<název souboru>`	Výstupní soubor pro detailní evaluaci
`lf|logfile:<název souboru>`	Souboru pro záznam průběhu evaluace
`h|help`	Nápověda
`dt|disttolerrance:<číslo>`	Tolerance pro určení odpovídajících bodů
`t|reftriangleside:<číslo>`	Délka hrany referenčního trojúhelníku

Tabulka 7.1: Parametry konzolové aplikace.