

Tomori, Z. – Demjénová, E.

**Programovanie modulov
na číslicové spracovanie obrazov**

Košice

2006

Obsah

1	Úvod.....	7
2	Hostiteľský program Ellipse	8
2.1	Modulárna štruktúra Ellipse	8
2.2	MDI (Multiple Document Architecture), aktívne okno	9
2.3	Slider - ovládanie stacku pomocou posuvného ovládača.....	9
2.4	Prostriedky na grafickú editáciu (kresliace prostriedky).....	10
2.5	Kalibrácia vzdialeností a optickej hustoty.	11
2.6	Okno výsledkov.....	12
2.7	Organizácia adresárov a podadresárov v Ellipse.....	12
2.8	Ovládanie plug-in modulov.....	12
2.9	Príklad 1 - Inštalácia a test základných funkcií programu Ellipse	13
3	Plug-in moduly a ich komunikácia s Ellipse.....	14
3.1	Základné komunikačné funkcie plug-in modulu.....	14
3.2	Príklad 2: Hello1 - Vytvárame prvý modul („Hello world“)	14
3.3	Základné triedy plug-in modulu	17
3.3.1	Trieda CPluginDlg	17
3.3.2	CDlg	18
3.3.3	CPluginView – pohľad z modulu na dokument	18
3.4	Príklad 3: Basics – modul demonštrujúci základné triedy	19
4	Plugin Wizard a jeho používanie	24
4.1	Čo je Plugin Wizard	24
4.2	Príklad 4: Hello2 - „Hello World“ pomocou Plugin Wizardu	25
5	Bitmapy typu DDB a DIB	27
5.1	Bitmapa DIB (Device Independent Bitmap).....	27
5.2	Triedy ATL::CImage a CImg na podporu práce s bitmapou.	28
5.3	Príklad 5: CImgDemo - demonštrácia použitia triedy CImg	29
6	Kreslenie do okna obrazu, overlay, objekty, polygóny.....	31
6.1	Príklad 6: DrawDemo - kreslenie z modulu a funkcia AddPluginDraw	31
6.2	Objekty v overlay a ich reprezentácia polygónmi.....	32
6.3	Príklad 7: OverlayPolyDemo - vykresľovanie pomocou OverlaPoly	34
7	Trackery, kreslenie základných geometrických objektov	36
7.1	Trieda CTracker	36
7.2	Príklad 8: TrackerDrawDemo - používanie Trackera na kreslenie.....	37
7.3	Príklad 9: TrackerEditDemo - editovanie objektu pomocou CTracker:	39
8	Vyššia forma komunikácie medzi Ellipse a modulom pomocou Plugin Interface.	43
8.1	Zoznam funkcií Plugin Interface (PIN).....	44
8.2	Využitie PluginInterface.	46
8.2.1	Príklad 10: PINDocDemo - vytvorenie PIN pomocou existujúceho dokumentu 46	
8.2.2	Príklad 11: PINImgDemo - vytvorenie PIN pomocou existujúceho obrazu a vytvorenie nového dokumentu	47
9	Zobrazovanie číselných výsledkov	49
9.1	Okno výsledkov (Result Window) a tabuľka výsledkov	49
9.2	CResultTable - programátorská podpora tabuľky výsledkov	49
9.3	Príklad 12: ResultTableDemo – používanie tabuľky výsledkov.....	50
10	Literatúra	53
11	Knižnica elGraph.....	54

11.1	Funkcie špecifické pre triedu CImg.....	55
	CImg.....	55
	operator =	55
	IsAlpha	55
	GetPixelIndex.....	56
	IsGrayscale	56
	GetDimensions	56
	GetPitch.....	57
	GetBits.....	57
	SetGrayColorTable.....	57
11.2	Funkcie spoločné pre triedy CImg a ATL::CImage.....	58
	~CImg(void).....	58
	Create	58
	GetWidth.....	58
	GetHeight	58
	GetBPP.....	58
	IsIndexed	58
	GetColorTabl.....	58
	SetColorTable.....	58
	GetPixel.....	59
	SetPixel.....	59
	Load.....	59
	Save	59
	Draw	59
	GetMaxColorTableEntries	59
11.3	Trieda COverlayPoly.....	60
	COverlayPoly	60
	Add.....	60
	GetSize	60
	SetSize.....	61
	operator =	61
	operator [].....	61
	SetAt.....	62
	RemoveAt.....	62
	RemoveAll	62
	InWhichPol.....	62
	PtInWhichPol	63
	LabInWhichPoly	63
11.4	Trieda CPolygon a jej špecifické funkcie	64
	CPolygon.....	64
	SetLabel.....	64
	SetClass	64
	GetLabel.....	65
	GetClass	65
	PtInPolygon.....	65
	DPtoLP	66
	LPtoDP	66
	Shift	66
	Area	66
	Length.....	67

GetBoundingRectLP	67
GetRefreshRectDP	67
InvalidatePolygon.....	68
Overlapped	68
CalculateAttributes	68
Profile	69
ReducePoints	69
ReduceLines	70
11.5 Trieda CPolygon a jej funkcie zdedené od CArray.....	70
operator =	70
operator []	70
~CPolygon.....	70
SetSize	71
GetSize	71
SetAt.....	71
InsertAt.....	71
RemoveAt.....	71
RemoveAll	71
Add.....	71
GetData.....	71
11.6 Trieda CTracker	72
CTracker.....	72
~CTracker.....	72
MovePolygon	72
MoveVertex.....	73
InsertVertex.....	73
RemoveVertex.....	74
HitTest.....	74
TrackRubberBand	75
11.7 Triedy odvodené od CTracker.....	75
CTrackerPoint	75
CTrackerLine	76
CTrackerAngle	76
CTrackerPoly	76
11.8 CResultTable a zapisovanie výsledkov do okna výsledkov.....	77
CResultTable.....	77
Create	77
Add	77
Replace	78
SetMask.....	78
GetRow.....	78
GetHeadItem	79
GetString	79
GetSize	80
GetCompressedTabSize	80
Empty	80
RemoveAll	80
RemoveAt.....	81
12 Podrobný popis funkcií Plugin Interface.....	82
12.1 Funkcie pre čítanie z PIN	82

GetPINVersion	82
GetPINSize	82
GetViewZoomVal	82
GetViewTrackPoly	83
GetViewCanAddPoly	83
GetViewCurClass	83
GetViewDrawingTools	84
GetViewDrawnPolygons	84
GetCalDensLUT	85
GetCalDensN	85
GetCalDensCalibrated	85
GetCalDistConstXY	86
GetCalDistConstZ	86
GetCalDistUnits	86
GetCalDistCalibratedXY	87
GetCalDistCalibratedZ	87
GetSlidrInvert	87
GetSlidrEnableInvert	87
GetSlidrMinSel	88
GetSlidrMaxSel	88
GetSlidrPos	88
GetSlidrLastPos	89
GetSlidrLabMin	89
GetSlidrLabMax	89
GetStackImages	90
GetStackCurImage	90
GetStackOverlayPolys	90
GetStackCurOverlay	91
GetStackCaptions	91
GetStackXYSize	91
GetStackZSize	92
GetStackModifiedImg	92
GetUserData	92
GetPathModule	93
GetPathImages	93
GetPathEllipse	93
GetResultTable	94
12.2 Funkcie pre zápis prostredníctvom PIN	94
SetViewCanAddPoly	94
SetViewDrawingTools	95
SetViewDrawnPolygons	95
SetCalDensCalibrated	95
SetCalDistConstXY	96
SetCalDistConstZ	96
SetCalDistUnits	96
SetCalDistCalibratedXY	97
SetCalDistCalibratedZ	97
SetSlidrInvert	97
SetSlidrEnableInvert	98
SetSlidrMinSel	98

SetSlidrMaxSel.....	98
SetSlidrPos	99
SetSlidrLastPos	99
SetSlidrLabMin	99
SetSlidrLabMax.....	100
SetStackCaptions.....	100
SetStackModifiedImg.....	100
SetUserData.....	101
SetPIN	101
12.3 Špeciálne funkcie PIN.....	101
CPluginInterface(CDocument* pDoc);.....	102
CPluginInterface(CImg** pDib,...).....	102
~CPluginInterface	102
AddNewImages	103

1 Úvod

Táto učebnica je určená všetkým, ktorí majú základné programátorské znalosti v prostredí MS Visual C++ a potrebujú riešiť špecifický problém z oblasti číslícového spracovania obrazu bez toho, aby museli programovať tzv. obslužné a servisné činnosti. Vďaka hosťateľskému programu na spracovanie obrazov *Ellipse*, je možné sa koncentrovať iba na naprogramovanie samotnej špecifickej aplikácie v tvare plug-in modulu. Programátor pritom využíva služby *Ellipse* nielen na načítanie a základné rutinné operácie s obrazmi, ale vďaka podpore knižnice *elGraph* dostane k dispozícii aj základné stavebné prvky, na ktorých je postavený program *Ellipse*. Filozofia plug-in modulov je pomerne rozšírená a u väčšiny programov na spracovanie obrazu dosť podobná. To umožňuje čitateľovi uplatniť tu získané skúsenosti aj pri inom prostredí než je *Ellipse*.

Z hľadiska spôsobu výkladu bol našim vzorom prístup, ktorý zvolil Kruglinský vo svojej svetoznámej učebnici programovania Visual C++ [1]. Ten sa, na rozdiel od mnohých iných autorov, nesnažil vnútiť čitateľovi systematický prístup k pochopeniu celého vysvetľovaného príkladu, ale učí ho pomocou *Application Wizardu* a *Class Wizardu* generovať základný kód, pričom na miestach kódu označených TO DO (treba urobiť) čitateľ doňho „vpisuje“ svoje časti kódu. V snahe uplatniť podobnú filozofiu sme aj my vyvinuli tzv. „*Ellipse Plug-in Wizard*“, ktorý na základe určitých vzorových príkladov generuje základný kód, do ktorého potom programátor vkladá svoje funkcie a modifikácie kódu na spracovanie obrazu. To samozrejme predpokladá znalosť programovania v C++ a knižnice MFC. Doporučeným prostredím je „Visual Studio .NET“ pri použití jazyka C++, v ktorom boli kompilované všetky vzorové príklady.

Netajíme sa tým, že čitateľ dostáva „kuchársku knihu“ s jediným cieľom: umožniť mu vo veľmi krátkom čase vytvoriť plne funkčnú aplikáciu z oblasti spracovania obrazu.

Všetky softwarové produkty potrebné k vytvoreniu funkčnej aplikácie, včítane zdrojových textov príkladov, sú k dispozícii na príslušnej stránke <http://www.ellipse.sk>. Pre užívateľov demo verzie (ktorá dovoľuje načítať iba limitovaný počet predvolených obrázkov), ako aj pre registrovaných užívateľov komerčnej verzie, sa tak ponúka príležitosť vyskúšať si, ako jednoducho je možné s touto podporou vytvoriť plne funkčnú špecializovanú aplikáciu.

Túto učebnicu je možné rozdeliť zhruba na tri časti. V prvých desiatich kapitolách sú vysvetľované základné pojmy, ktorých ovládanie je potrebné k vývoju vlastnej aplikácie. Tieto vysvetľovanie týchto pojmov je sprevádzané triviálnymi príkladmi s grafickým odlíšením novo vytvoreného kódu. Aj napriek minimálnemu rozsahu kódu (často iba niekoľko riadkov v univerzálnej funkcii *DoTest*) je výsledná aplikácia plne funkčná a použiteľná.

Druhá časť učebnice už používa komplikovanejšie príklady, ktorých úlohou nie je iba jednoduché vysvetlenie pojmov, ale slúžia priamo ako vzorové príklady (Templates) pre vlastné aplikácie. Vývoj aplikácie je potom založený na výbere najvhodnejšieho vzorového príkladu, ktorý slúži ako zdroj pre generovanie nového projektu pomocou *Plugin Wizardu*. V tretej časti knihy sú podrobne popísané jednotlivé funkcie *PluginInterface* a ostatných tried, ktorých znalosť je podmienkou úspešného vývoja modulov.

2 Hostiteľský program *Ellipse*

Názov programu „*Ellipse*“ je v skutočnosti skratkou slov **E**asy-to-**L**earn **L**aboratory **I**mage **P**rocessing **S**ystem and **E**ditor. Je produktom firmy ViDiTo Systems, pričom autorský kolektív tejto knihy významne prispel k vývoju programu a svojimi návrhmi ovplyvnil jeho vlastnosti.

Program *Ellipse* v demo verzii sa nachádza na Web stránke <http://www.ellipse.sk>. Demo verzia je totožná s plnohodnotnou verzou, pokiaľ ide o ovládanie a funkcie, dovoľí však načítať iba vopred definované obrázky, ktoré sú súčasťou inštalácie. Súčasťou inštalácie *Ellipse* je aj rozsiahla užívateľská príručka v anglickom jazyku, a preto predpokladáme, že čitateľ sa už oboznámil z jeho ovládaním a základnými vlastnosťami. Podobne predpokladáme aspoň základnú znalosť architektúry Dokument/Pohľad (*Doc/View*), ktorá je dostatočne zrozumiteľne popísaná v [1]. Preto budú v ďalších podkapitolách spomenuté iba tie črty programu, ktoré sú dôležité z hľadiska nášho cieľa – písania vlastných plug-in modulov.

2.1 Modulárna štruktúra *Ellipse*

Zásady štruktúrovaného a modulárneho programovania patria už dlhšiu dobu medzi základné „hygienické“ návyky programátora, a preto mlčky predpokladáme že čitatelia tejto knihy ich majú v dostatočnej miere osvojené. Aj v našich programoch sa snažíme striktné dodržiavať objektovo – orientovaný prístup. Táto zásada je ešte posilnená tým, že základné stavebné kamene programu (objekty a im patriace funkcie) musia tvoriť logický celok relatívne nezávislý od ostatných objektov, s ktorými komunikujú definovaným spôsobom.

V tejto knihe ako aj v súvisiacich programoch sa snažíme o ešte vyššiu formu modularity a to nielen v zdrojovom kóde, ale aj vo výsledných preložených produktoch. Vhodnou platformou sú „Dynamicky linkovateľné knižnice“ (DLL), ktoré predstavujú moduly, ktoré môžu (ale nemusia) byť pripojené k programu počas jeho behu. To znamená, že po spustení základného programu *Ellipse* a načítaní vstupného obrazu sa operátor rozhodne, ktorý modul potrebuje a odštartuje ho prostredníctvom menu. Ten sa skopíruje do pamäte, vykoná nad vstupným obrázkom nejakú špecifickú operáciu a po jej ukončení opäť uvoľní alokovanú pamäť. Takýto modul sa volá **Plug-in** (pripojiteľný). Ich výhodou je ich nezávislosť od hostiteľského programu (s výnimkou definovaného rozhrania) a tiež od ostatných modulov. Táto nezávislosť má okrem známych obecných pozitív modulárneho programovania aj jeden dôležitý psychologický dôsledok. Autor modulu je jeho vlastníkom, môže rozhodovať o forme jeho uplatnenia (freeware, shareware, výmena, predaj). Na druhej strane je autor neanonymne zodpovedný za jeho správnu funkciu, čo vedie k zodpovednosti pri programovaní. Koncepcia Plug-in modulov nie je nová, sú často používané ako doplnky rôznych softwarových produktov. V oblasti programov na spracovanie obrazu sa táto koncepcia uplatnila veľmi výrazne a stala sa ich pevnou súčasťou (ImagePro, Photoshop, ImageTool, IPTK a pod.).

2.2 MDI (Multiple Document Architecture), aktívne okno

MDI je taká architektúra programu pod Windows, ktorá umožňuje súčasne zobrazovať viac dokumentov, každý v inom dcérskom okne aplikácie. Príkladom takejto architektúry je napr. program MS Word. Iba jedno z jeho okien môže byť označené ako aktívne, tohto okna sa týkajú všetky príkazy menu, udalosti, príkazy z panelu nástrojov. Neaktívne okno môžeme kliknutím aktivizovať, prejaví sa to odlišnou farbou titulkového pruhu okna.

Pre programátora plug-in modulov je dôležité vedieť, že modul je zviazaný s tým dokumentom, ktorý je aktívny v okamihu štartu modulu a to až do uzavretia modulu alebo okna. Takýto dokument budeme v ďalšom texte volať **vstupný dokument modulu**. Teda všetko, čo bude modul vykonávať, sa prejaví iba v tomto okne a to aj v prípade, keď bol pri otvorení modulu aktivizovaný iný dokument.

V našom programe na spracovanie obrazov je dokumentom obraz a preto aj v ďalšom texte budeme pod **dokumentom** rozumieť **obraz** a naopak. Prvým krokom po štarte je otvorenie dokumentu a to buď načítaním z obrazového súboru (*File | Load*), cez skener - TWAIN interface (*File | Acquire*), alebo pomocou plug-in modulu kategórie *Input*.

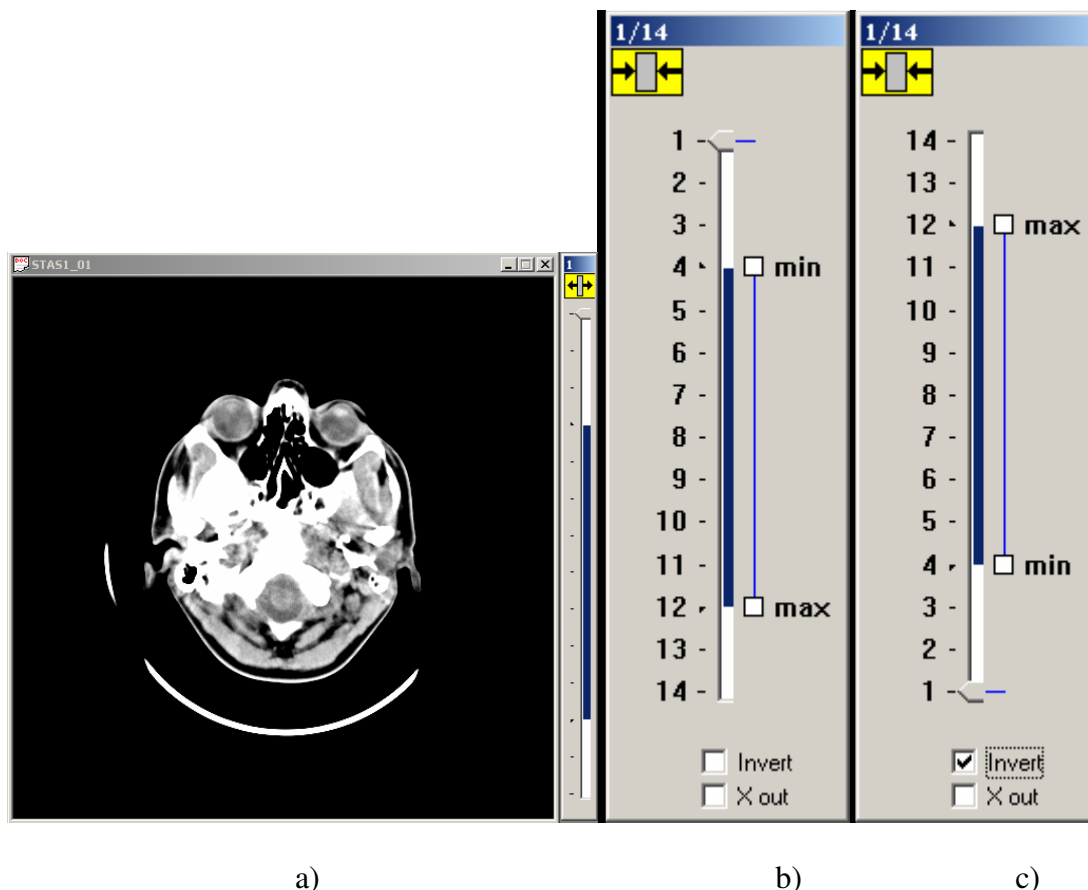
Každé obrazové okno je vybavené vlastnosťami „zoom“ a „scroll“. Súradnice v rámci obrazu i vypočítané parametre sú invariantné voči zväčšeniu a posunutiu okna skrolovacou lištou. Program pracuje nielen s individuálnymi obrázkami, ale aj so sériami obrazov reprezentujúcich určitú časovú, alebo priestorovú postupnosť (napr. sériové rezy trojrozmerným objektom).

2.3 Slider - ovládanie stacku pomocou posuvného ovládača

Pripomeňme si, ako Ellipse zobrazuje stack obrazov rovnakých rozmerov. Stack môže v danom okamihu zobrazovať iba jeden jeho obrázok, ktorý je vybraný pomocou posuvného ovládača (slidera), ktorý je tesne pripojený k pravému okraju obrázku (viď Obr. 1). Pri posúvaní obrázku (uchopením za lištu pomocou myši) ostáva slider pripojený k pravej časti obrázku a kopíruje jeho pohyb. Slider a obrázok môžeme od seba odpojiť tak, že slider uchopíme myšou za lištu a odtiahneme. Dvojitým kliknutím na lištu slidera sa tento automaticky opäť „nalepí“ na pravý okraj.

Slider môže byť zobrazovaný v dvoch **módoch**. V úzkom móde (Obr. 1a) prispôbi svoju veľkosť rozmerom obrázku a zaberá minimum šírky. Kliknutím na farebné tlačítko so šípkami v hornej časti slidera sa prepne do tzv. „širokého“ módu (Obr. 1b). V ňom sú jednotlivé hladiny číselne označené. Zaškrtnutím checkboxu **Invert** dosiahneme, že prvý obrázok série bude zobrazovaný v polohe slidera celkom hore (Obr. 1c). V neinvertovanom zobrazení to bude naopak spodná poloha. Tým vieme prispôbiť spôsob zobrazovania fyzikálnej predstave (napr. zaostrovanie do hĺbky je orientované smerom dole). Posunutím štvorčekov s označením „min“ a „max“ vo zvislom smere vieme nastaviť tzv. výberovú oblasť „**Selection**“, ktorá je v slideri označená modrou farbou. Zaškrtnutím **Xout** zabezpečíme také zobrazovanie obrázkov slidera, že keď sa dostaneme mimo oblasť Selection, budú obrázky zobrazované preškrtnuté výrazným krížom cez uhlopriečky obrazu. Drobná horizontálna čiara, ktorá je na obrázkoch b) a c) spojená so sliderom, je **ukazovateľ poslednej pozície**. Slúži, podobne ako záložka v knihe na to, že keď sliderom pohneme a chceme ho potom vrátiť do pôvodnej polohy, ukazovateľ poslednej pozície nám uľahčí orientáciu.

Ovládanie stacku pomocou slidera v Ellipse je jednoduché. Cez PluginInterface má aj modul umožnené nielen čítať informáciu o nastavení slidera, ale aj priamo z modulu programovo ovládať jeho polohu a nastavenia.



Obr. 1

Ovládanie stacku obrazov pomocou posuvného ovládača (slidera)

- séria obrazov z CT zobrazená v zmenšenej mierke. K pravému okraju obrázku je pripojený „úzky“ slider prispôsobený veľkosťou výške obrázku.
- Slider v „širokom“ móde zobrazenia. Posunutím bielych štvorčekov označených „min“ a „max“ vo vertikálnom smere určíme veľkosť výberovej oblasti „Selection“ .
- Nastavenie invertovaného módu zobrazenia, keď prvý obrázok série zodpovedá polohe slidera celkom hore. Pomocou Xout vieme zobrazovať obrázky mimo rozsahu Selection ako preškrtnuté výraznou kresbou v tvare kríža.

2.4 Prostriedky na grafickú editáciu (kresliace prostriedky)

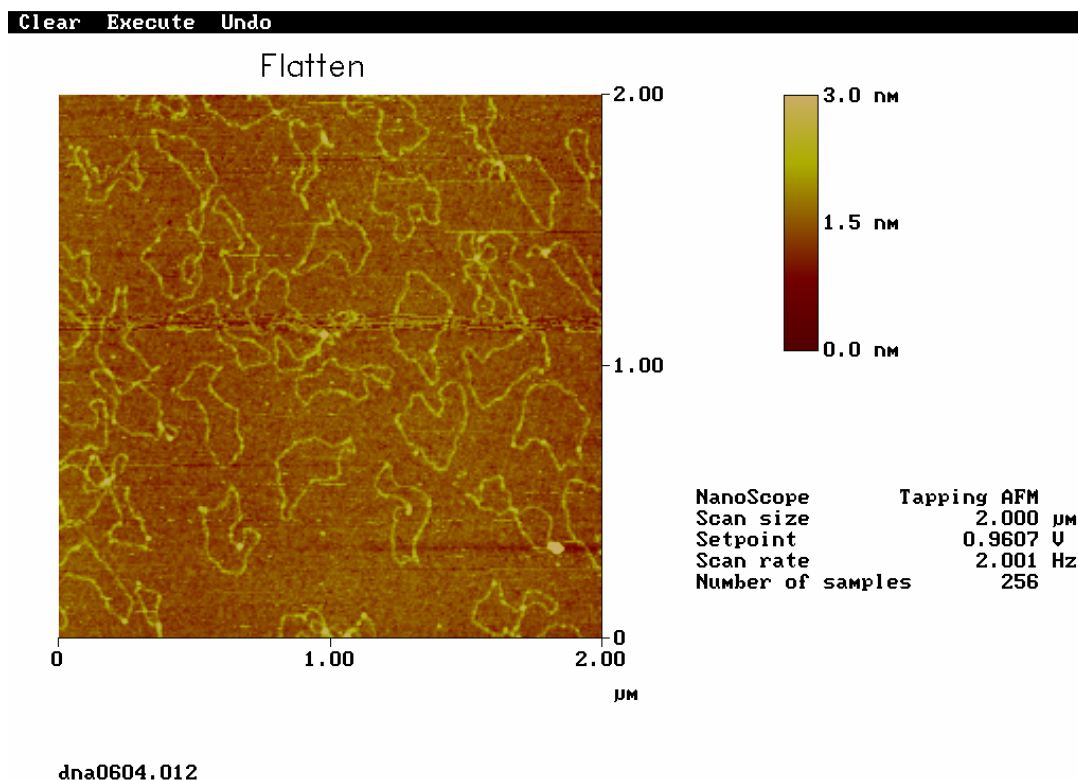
Ellipse má kresliace prostriedky ktorými je možné kresliť priamo v obrazovom okne (bod, úsečka, uhol, multi-úsečka, polygón definovaný postupnosťou vrcholov, polygón definovaný kreslením voľnou rukou). Každý z nakreslených objektov je uchovaný ako nezávislá súčasť (OverlayPoly) okna a jeho koordináty sa dajú získať za účelom spracovania.

V tzv. editovacom režime sa uplatňuje podobná filozofia „aktívneho prvku“ ako pri dokumentoch, čiže z nakreslených objektov jeden môže byť aktívny a budú sa ho teda týkať príkazy menu (napr. vymazanie). Analogicky, objekt je tvorený bodmi, z ktorých jeden môže kliknutím označiť ako aktívny a bude sa ho týkať príkaz z menu (vymazať, presunúť a pod.)

2.5 Kalibrácia vzdialeností a optickej hustoty.

U nekalibrovaného systému predstavuje základnú jednotku vzdialenosti *pixel*, ktorého hodnota predstavuje u šedotónových obrazov *jas* a u farebných obrázkov *farbu*. Tieto jednotky sú závislé na parametroch konkrétneho snímacieho zariadenia a preto ich nie je možné vzájomne porovnávať.

Pretože ale často zobrazuje obrázok reálnu scénu so známymi fyzikálnymi veličinami, je možné pomocou nich obrázok kalibrovať. Ellipse podporuje 2 základné typy kalibrácie a to **priestorovú kalibráciu** (vzdialenosti v smeroch osí X, Y, Z) a **kalibráciu optickej hustoty**. Obr. 2 obsahuje údaje nevyhnutné pre oba typy kalibrácie a to skutočný rozmer obrázky v mikrometroch (spodný a pravý okraj obrázku) a tiež „tabuľku“, ktorou sa transformuje optická hustota resp. jas pixelu na inú fyzikálnu veličinu (výška objektu v nm – vid’ kalibračný prúžok vpravo). Pokiaľ by sme miesto jedného obrázku mali k dispozícii ich sériu predstavujúcu napr. sériové rezy telesom, vieme určiť vzdialenosť v smere osi Z ako vzdialenosť medzi jednotlivými hladinami, resp. hrúbku rezu. Detailný popis kalibrácie priestorovej kalibrácie i kalibrácie optickej hustoty je uvedený v „help“ súbore Ellipse.



Obr. 2

Kalibrácia vzdialeností a optickej hustoty

Z hľadiska programového modulu je dôležité mať informáciu o tom, či je systém priestorovo kalibrovaný, a ak áno, tak aké sú kalibračné konštanty v smere osi XY a Z a tiež aké reálne jednotky predstavujú číselné údaje (mm, km, a pod.). V prípade kalibrovania optickej hustoty potrebuje modul poznať prepočítavaciu tabuľku (tzv. Lookup Table – LUT) medzi jasom

a reálnou fyzikálnou veličinou. Všetky tieto údaje môže modul zistiť pomocou `PluginInterface` (viď kap. 8).

2.6 Okno výsledkov

Ellipse používa na zobrazovanie výsledkov analýzy tzv. Okno výsledkov (Result Window), ktoré môžeme otvoriť príkazom `File | New ResultsWnd` za predpokladu, že máme otvorený nejaký obrázok (dokument). Zobrazuje jednoduchý tabuľkový procesor, ktorý používa aj funkcie `Copy`, `Paste` a `Export`. Ak je toto okno otvorené, pre každý nakreslený objekt sa počíta cca 18 charakteristických parametrov, ktoré sa zobrazujú interaktívne, to znamená, že napr. pri zmene tvaru objektu sa prepočítajú aktuálne parametre a výsledky sa okamžite zobrazia v príslušnom riadku Okna výsledkov.

2.7 Organizácia adresárov a podadresárov v Ellipse

Ako pri každom inštalovanom programe, aj pri Ellipse je možné zvoliť si cieľový adresár. Predpokladajme, že zvolíme predvolené hodnoty, potom sa nám program nainštaluje do adresára `C:/Program Files/Ellipse`. V tomto adresári sú okrem `Ellipse.exe`, `Ellipse.hlp` a niektorých pomocných dll knižníc, aj podadresáre **Images** a **Plugins**, ktorých funkcia je zrejmá. Tu doporučujeme umiestniť aj adresár **SDK** (Software Development Kit), doňho adresár **Ellipse**, doňho podadresáre s predponou „Cat_“ reprezentujúce jednotlivé kategórie modulov (napr. **Cat_Templates**). Až v ňom je potom umiestnený podadresár obsahujúci zdrojový kód modulu. Táto štruktúra síce nie je povinná, vrelo ju ale doporučujeme, nakoľko z nej vychádzajú mnohé nastavenia využívané napr. `Plugin Wizardom`, ktoré by sa inak museli zložito prestavovať.

2.8 Ovládanie plug-in modulov.

Plug-in modul vykonáva špecifickú činnosť, v závislosti od spôsobu akým bol naprogramovaný. Táto činnosť môže mať nasledujúci charakter:

- a) **Generovanie obrazu.** Vstupom nie je nič, výstupom je obraz. Plug-in moduly tohto typu patria všetky do kategórie „**Input**“, a odlišujú sa od ostatných tým, že ich je možné spúšťať hneď po štarte *Ellipse* a pomocou nich generovať nový dokument. Dokážu tak nahradiť funkcie `File | Load` na načítanie obrazu zo súboru a `File | Acquire` na snímanie obrazu skenerom pomocou TWAIN interface. V tejto kategórii sú napr. aj moduly, ktoré načítavajú obraz z Web kamery, DV kamery, neštandardných obrazových súborov, alebo ich generujú analytickými metódami.
- b) **Spracovanie obrazu.** Vstupom je vstupný dokument (obraz), výstupom opäť obraz, ktorý sa generuje buď miesto pôvodného vstupného obrazu, alebo ako nový dokument v samostatnom okne.
- c) **Analýza obrazu.** Vstupom je opäť obraz, výstupom číselné hodnoty, ktoré ho nejakým spôsobom charakterizujú. Tieto číselné údaje sa zapisujú zvyčajne do Okna výsledkov, alebo do textového súboru.

Existuje špeciálny podadresár s názvom **Plugins**, určený iba pre plug-in moduly a nepovinné súbory, ktoré s nimi bezprostredne súvisia (help súbor a ikona). Keď sa napr. modul volá `MyModule.dll`, súbor `MyModule.hlp` bude help súbor, ktorý je možné zavolať iba z modulu `MyModule.dll` a obrazový súbor `MyModule.bmp` predstavuje ikonu, ktorú je možné umiestniť na lištu Ellipse a spúšťať modul kliknutím na ikonu. Modul síce môže byť umiestnený v ľubovoľnom adresári, vtedy je však nutné ho spúšťať pomocou funkcie **Browse and Run**. Naproti tomu, umiestnenie modulu do podadresára *Plugins* zobrazuje všetky moduly zatriedené do kategórií a podľa abecedy, čo umožní pohodlný výber a spustenie.

2.9 Príklad 1 - Inštalácia a test základných funkcií programu Ellipse

- 1) Nainštalujte Ellipse z pripraveného CD spustením Setup.exe. Pokiaľ použijete default nastavenia, vytvorí sa adresár *c:/Program Files/Ellipse* v ktorom je spustiteľný súbor *Ellipse.exe*. Okrem toho sa tam vytvorí podadresár „*Images*“ ktorý obsahuje testovacie obrázky pre prácu s demo verziou ako aj podadresár *Plugins*, kde budú umiestňované všetky plug-in moduly viditeľné z menu.
- 2) Pri inštalácii sa vytvorí aj podadresár SDK (Software Development Kit), obsahujúci súbory pre prácu s projektom vo Visual C++ (práca s SDK bude obsahom nasledujúcich cvičení).
- 3) Odštartujte Ellipse.exe a pomocou funkcie *Load | Single Image* a načítajte obr. „*Blobs.tif*“ nachádzajúci sa v podadresári *Images*.
- 4) Nastavte v *Setup | Object Analysis* parametre, ktoré chceme počítať.
- 5) Odštartujte modul prahovania *Plug-ins | Ellipse | Segmentation | Threshold*. Vyberte sliderom najvhodnejší prah a stlačte *OK*.
- 6) Označte *Result Window* kliknutím ako aktívne okno a uložte jeho obsah do textového súboru (*File | Save As*). Spustíte nejaký externý tabuľkový procesor (napr. MS Excel) a načítajte doň údaje z uloženého textového súboru.
- 7) Zavrite a opätovne otvorte obrázok „*Blobs.tif*“ Tým sa zmaže výsledok segmentácie prahovaním. Kliknite na ikonu *Freehand* a snažte sa ručne obkresliť hranice objektov na obrázku. Kreslenie sa ukončí ak sa kurzor dostane do blízkosti počiatočného bodu čo znamená uzavretie krivky, alebo ak ju uzavrie užívateľ dvojitém kliknutím. Pri kreslení využite funkciu *zoom* a *scroll*.
- 8) Použite obdobný postup pomocou funkcie *Polygon*, kde sa kliknutím zadávajú jednotlivé vrcholu polygónu.
- 9) Sledujte údaje v *Result Window*. Pomocou príslušnej ikony prejdite do *EDIT* módu. Aktivujte kliknutím niektorý z nakreslených polygónov, skúste presunúť, vymazať a vložiť (pravým tlačítkom myši) niektorý vrchol. Sledujte odozvu v príslušnom riadku *Result Window*.
- 10) Označte niekoľko riadkov v *Result Window* a to tak, že označíte prvú bunku kliknutím a poslednú kliknutím so *SHIFT* tlačítkom. Označená oblasť je odlíšená farbou. Uložte označené údaje do clipboardu (*Edit | Copy*)
- 11) Vložte údaje z clipboardu do externého tabuľkového procesora, v ktorom už sú uložené výsledky segmentácie prahovaním (bod 7). Porovnajte hodnoty parametrov tých istých objektov získaných a) prahovaním, b) kreslením pomocou funkcie *Freehand* c) kreslením pomocou funkcie *Polygon* (zadávanie vrcholov).

3 Plug-in moduly a ich komunikácia s Ellipse

Každý plug-in modul pre Ellipse je vlastne samostatnou DLL knižnicou, čiže súborom nazvaným napr. „*MyModule.dll*“, pričom názov „*MyModule*“ by mal vystihovať funkciu modulu. Navyše by mal byť unikátny, keďže všetky moduly majú byť umiestnené v spoločnom podadresári „*Plugins*“.

Pripomeňme si najprv na jednoduchom príklade, ako vo Windows prebieha tzv. explicitné linkovanie a volanie funkcie *MyFunc(WPARAM wp, LPARAM lp)*, ktorá je súčasťou knižnice *MyLibrary.dll*. Po načítaní knižnice do pamäte zistíme adresu funkcie *MyFunc*. Potom pomocou získaného smerníka môžeme funkciu zavolať, pričom parametre nech sú napr. *wp=0* a *lp* prenáša smerník na získanú kresliacu plochu pDC.

```
.  
.   
CDC* pDC = GetDC();  
typedef int (MYFUNC)(WPARAM, LPARAM);  
HINSTANCE m_hMyDLL = AfxLoadLibrary("MyLibrary.dll");  
MYFUNC* pMyFunc = (MYFUNC*)GetProcAddress(m_hMyDLL, "MyFunc");  
pMyFunc(0, (LPARAM)pDC);  
.   
.   
AfxFreeLibrary(m_hMyDLL);
```

3.1 Základné komunikačné funkcie plug-in modulu

Základná komunikácia medzi Ellipse a plug-in modulom prebieha pomocou štyroch globálnych funkcií, ktoré musia byť štandardne súčasťou každého plug-in modulu.

PluginOpen	obsluha modulu po otvorení
PluginClose	záverečná obsluha modulu po jeho uzavretí
PluginMessages	obsluha posielania správ z Ellipse do modulu
PluginDraw	výzva, aby modul pridal svoju kresbu ku kresbe Ellipse

Otvorenie modulu je vlastne uskutočnené tak, že sa zistí adresa funkcie *PluginOpen* vo zvolenom module a tá sa odštartuje. Uzavretie modulu prebieha analogicky – modul vyšle požiadavku na uzavretie do Ellipse a tá vzápätí zavolá funkciu *PluginClose*. Medzi týmito dvoma udalosťami (čiže keď je modul otvorený) každé prekreslenie obrazu je nasledované volaním funkcie *PluginDraw*, ktoré vyzve modul na nakreslenie svojich dát. Okrem toho je to funkcia *PluginMessages*, ktorá posiela do otvoreného modulu niektoré správy na spracovanie.

3.2 Príklad 2: Hello1 - Vytvárame prvý modul („Hello world“)

Prikročme teda ku generovaniu notoricky známeho klasického príkladu, „Hello world“, ktorým začína množstvo učebníc programovania. Tento príklad v našom prípade znamená, že vytvoríme modul s názvom *Hello1.dll*, ktorý zavoláme z Ellipse a on nám vypíše spomínaný text vo zvláštnom okne. Po zavretí tohto okna modul ukončí svoju činnosť. Z kategorizácie

modulov v časti 2.8 vyplýva, že jedinou kategóriou, ktorá dovoľuje spustenie modulu bez predchádzajúceho načítania dokumentu je kategória *Input*. Preto aj náš modul zaradíme do tejto kategórie (hoci jeho výstupom nebude generovaný obraz, ako sa to obvykle od modulov tejto kategórie očakáva).

Používame prostredie MS Visual Studio.Net a jazyka C++, pričom vychádzame zo základného „solution“ súboru *Ellipse.sln* ktorý je na priloženom CD alebo na domácom Webe Ellipse. Podobne, ako to je pravidlom v knihe [1], aj my budeme zobrazovať primárny zdrojový kód bez podfarbenia, časti kódu, ktoré postupne k primárnemu kódu pridávame sú označené šedo tieňovaným pozadím. Takto označených častí sa potom týkajú komentáre v ďalšom texte.

- a) Spustíme Visual Studio .Net a otvoríme *Ellipse.sln*. Necháme zobrazovať Solution Explorer v okne vľavo. Na začiatku toto „solution“ neobsahuje žiadny projekt čo Solution Explorer signalizuje nápisom **Solution ‘ellipse‘ (0 projects)**.
- b) Klikneme na spomínaný nápis pravým tlačidlom myši, a vyberiem položku **Add** a potom vyberieme **New Project**. Zvoľme jazyk Visual C++ a z ponúknutých možností zvoľme MFC DLL. Treba tiež zadať názov *Hello1* a cieľový podadresár (*Cat_Input*) udávajúci kategóriu. V súlade s doporučeným umiestnením adresárov, popísanom v časti 2.7, bude potom plná cesta k cieľovému podadresáru:
C:/Program Files/ Ellipse/SDK/Ellipse/Cat_Input
- c) Po zadaní Názvu projektu a cieľového podadresára sa objaví ďalšie okno s predvolenou ponukou „Create a regular DLL (MFC shared)“. My ju však nezvolíme, ale prepneme na Application kde sa objavia 3 možnosti pre MFC DLL. Vyberieme poslednú (MFC Extension DLL). Ostatné políčka necháme nezaškrtnuté a potvrdíme generovanie projektu *Hello1*.
- d) Vygenerovaný projekt pozostáva z niekoľkých súborov, za pozornosť však stojí iba súbor *Hello1.cpp*, ktorý obsahuje zdrojový kód. Tento je totožný s kódom na nasledujúcom výpise, avšak bez šedo podfarbených častí. Obsahuje vlastne iba funkciu *DllMain*, ďalej makro *AFX_EXTENSION_MODULE* a tiež štandardné hlavičkové súbory a obsluhu *ifdef _DEBUG*. Súbor *Hello1.cpp*, z ktorého sme z priestorových dôvodov odstránili väčšinu komentárov, je v nasledovnom výpise (šedo tieňované časti sa týkajú kódu pridaného a komentovaného neskôr).
- e) Dáme zobrazit' vlastnosti vygenerovaného projektu a vyberieme položku **Linker** a v rámci nej položku **General**. Zmeníme obsah položky **Output file** nasledovne:
Output File = ../../../../plugins/Hello1.dll.
Tým zabezpečíme, aby sa kompiláciou a linkovaním projektu vytvoril modul priamo v podadresári *Plugins*.
- f) Prepne Solution Explorer tak, aby zobrazoval Resource View. Jediná položka, ktorú momentálne náš projekt obsahuje je **Version**. V rámci nej nastavíme `FileDescription = Ellipse_207 | Input | Hello1`. Tým definujeme, s ktorou verziou Ellipse je modul kompatibilný, do akej kategórie patrí a nakoniec, pod akým názvom sa bude objavovať v menu (môže, ale nemusí byť totožný s názvom modulu). Vyplnenie ostatných položiek ako číslo verzie, autora, copyright a pod. je síce nepovinné, ale veľmi doporučované.
- g) Skompilujeme a linkujeme projekt *Hello1* (napr. funkciou *Built* pomocou pravého tlačítka myši). V podadresári *Plugins* by sa mal objaviť nový súbor *Hello1.dll*. Pokiaľ máme v projekte predvolenú funkciu *Enable Incremental Linking*, objaví sa tam okrem neho aj súbor *Hello1.ink*. Jeho generovanie môžeme zrušiť zakázaním inkrementálneho linkovania v projekte.

- h) Vygenerovali sme teda najjednoduchšiu možnú knižnicu DLL, ktorú však zatiaľ nemôžeme nazvať modulom, pretože neobsahuje triedky na komunikáciu s hosťovským programom Ellipse. To bude úlohou šedo podfarbeného kódu, ktorý pridáme k primárne generovanému kódu. Tento predstavuje pridanie štyroch „povinných“ funkcií spomínaných v časti 3, ktoré musí každý modul obsahovať. Ich pridanie k doteraz vytvorenému kódu reprezentuje šedo podfarbená časť vo výpise *Hello1.cpp*. Všimnime si, že iba funkcia *PluginOpen* obsahuje príkaz na generovanie okna s nadpisom „Hello World“, ostatné tri funkcie nerobia nič (avšak musia tam byť, nakoľko sú z *Ellipse* volané).
- i) Odštartujeme *Ellipse* v režime Debug. Pri prvom štarte budeme pravdepodobne požiadaní o zadanie cesty k súboru *Ellipse.exe*, čo si vieme zjednodušiť pomocou tlačítka BROWSE. Zvoľme v menu položku *Plug-in* a potom *Ellipse*. Objaví sa nám zoznam modulov z kategórie *Input*, ktoré sa nachádzajú v adresári *Plugins*. Vyberieme modul *Hello1*, malo by sa objaviť okno s nápisom „Hello World“, po jeho zavretí činnosť modulu končí.

Hello1.cpp

```
#include "stdafx.h"
#include <afxdll.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

extern "C" __declspec(dllexport) int PluginOpen(WPARAM wp, LPARAM lp)
{
    AfxMessageBox("\nHello World");
    return(1);
}
extern "C" __declspec(dllexport) int PluginClose(WPARAM wp, LPARAM lp)
{
    return(0);
}
extern "C" __declspec(dllexport) int PluginMessages(WPARAM wp, LPARAM lp)
{
    return(0);
}
extern "C" __declspec(dllexport) int PluginDraw(WPARAM wp, LPARAM lp)
{
    return(0);
}

static AFX_EXTENSION_MODULE Hello1DLL = { NULL, NULL };

extern "C" int APIENTRY
DllMain(HINSTANCE hInstance, DWORD dwReason, LPVOID lpReserved)
{
    // Remove this if you use lpReserved
    UNREFERENCED_PARAMETER(lpReserved);

    if (dwReason == DLL_PROCESS_ATTACH)
    {
        TRACE0("Hello1.DLL Initializing!\n");

        // Extension DLL one-time initialization
        if (!AfxInitExtensionModule(Hello1DLL, hInstance))
            return 0;
    }
}
```

```

        new CDynLinkLibrary(Hello1DLL);
    }
    else if (dwReason == DLL_PROCESS_DETACH)
    {
        TRACE0("Hello1.DLL Terminating!\n");
        AfxTermExtensionModule(Hello1DLL);
    }
    return 1;    // ok
}

```

3.3 Základné triedy plug-in modulu

Predchádzajúci príklad demonštroval, že aj pomocou štyroch základných komunikačných funkcií je možné navrhnúť modul, ktorý plní veľmi jednoduché funkcie. Je však zrejmé, že akákoľvek zložitejšia činnosť bude vyžadovať, aby modul mal stanovenú takú štruktúru, ktorá umožní jeho ľahké ovládanie a hlavne rozširovanie.

Keďže Ellipse i plug-in moduly sú založené na používaní objektovo orientovaného programovania, bude každý plug-in modul, zložitejší od predchádzajúceho príkladu, používať nasledovné základné triedy :

CPluginDlg	zabezpečí najzákladnejšie funkcie dialógového okna
CDlg	odvodená od CPluginDlg, zabezpečí rozšírené ovládanie modulu pomocou tlačítok a obslužných funkcií
CPluginView	pohľad na dokument (obraz) v prostredí modulu

Dalo by sa zjednodušiť povedať, že trieda CPluginDlg zabezpečí základný servis modulu, a trieda CDlg, ktorá je od CDlgPlugin odvodená, umožní doprogramovanie vlastných ovládacích prvkov a zodpovedajúcich obslužných funkcií užívateľom. Trieda CPluginView je trieda, ktorá je „pohľadom modulu na obraz“, resp. predstaviteľom obrazového okna v module. Rozoberieme si teraz vlastnosti jednotlivých tried podrobnejšie

3.3.1 Trieda CPluginDlg

Zabezpečuje základný servis modulu, o programovanie ktorých sa preto nemusí starať programátor modulu. Služi ako rodičovská trieda triedy *CDlg*, ktorej programátorská realizácia je, naopak, plne pod správou užívateľa.

Pod základným servisom rozumieme napr. obsluhu tlačidiel *OK* alebo *Cancel* tak, aby sa správali v súlade s pravidlami pre nemoďálny dialóg, čiže aby nereagovali na stlačenie kláves ENTER a ESC.

Deklarácia triedy vyzerá nasledovne:

```

class AFX_EXT_CLASS CPluginDlg : public CDialog
{
    DECLARE_DYNAMIC(CPluginDlg)
public:
    CPluginDlg(CWnd* pParent = NULL);
    virtual ~CPluginDlg();
    CScrollView* GetView(int nView = viewEllipse);
    BOOL Create(int id);
    void OnOK();
    void OnCancel();

    CDocument* m_pDoc;

```

```

enum { IDD = IDD_PLUGINDLG };
protected:
    virtual void DoDataExchange(CDataExchange* pDX);
    DECLARE_MESSAGE_MAP()
public:
    virtual void AddPluginDraw(LPARAM lp);
    virtual void UpdateImage( LPRECT lpR = NULL, BOOL Erase = TRUE){
        GetView()->InvalidateRect(lpR, Erase);
    };
};

```

3.3.2 CDlg

Dialógovému oknu modulu zodpovedá trieda *CDlg*, pričom každému programátorovi musí byť známy mechanizmus ako po pridaní napr. tlačidla do dialógového okna zabezpečiť, aby sa do triedy *CDlg* pridala funkcia zabezpečujúca obsluhu udalosti kliknutia na tlačidlo. Za normálnych okolností by sme postupovali tak, že by sme vytvorili dialógové okno v Resource editore a potom nechali *Class Wizard* aby generoval triedu *CDlg*. Takáto trieda by bola štandardne odvodená od triedy *CDialog*. My ju však prepíšeme tak, že bude odvodená od triedy *CPluginDlg*.

Vďaka takémuto mechanizmu dedičnosti trieda *CDlg* zabezpečí nasledovné:

- Konštruktor automaticky naplní hodnotu *m_pDoc* smerníkom na dokument, ktorý bol aktívny počas štartu modulu (vstupný dokument).
- Funkcia *Create(ID)* zabezpečí vytvorenie triedy *CDlg*, ktorá zodpovedá vytvorenému dialógovému oknu s identifikátorom ID.
- Funkcia *GetView()* vráti smerník na okno dokumentu, ktoré bolo aktívne pri štarte modulu (vstupný dokument). Túto funkciu často potrebujeme pri programovaní modulu.
- Funkcie *OnOK()* a *OnCancel()* reagujú na správy IDOK a IDCANCEL a zabezpečujú správne uzatvorenie nemoďálneho dialógu tým, že pošlú Ellipse správu so žiadosťou o uzavretie modulu. Okrem toho analyzujú, či správy IDOK a IDCANCEL prišli po stlačení tlačítka myši alebo po stlačení kláves ENTER alebo ESC. Funkcie nechávajú správy z klávesnice bez odozvy, pretože nechceme, aby sa nemoďálny dialóg zatváral klávesnicou, ako je to obvyklé u moďálnych dialógov. (Túto situáciu by inak musel programátor ošetriť v triede *CDlg*). Pokiaľ požadujeme, aby uvedené funkcie vykonávali aj inú než len nevyhnutne nutnú činnosť, musíme ich deklarovať v samotnej triede *CDlg* a napísať vlastný obslužný kód ktorý musí byť ukončený volaním rodičovskej funkcie, napr. *CPluginDlg::OnOK()*

3.3.3 CPluginView – pohľad z modulu na dokument

Všetky moduly, okrem tých z kategórie *Input*, majú za úlohu buď spracovať, alebo analyzovať vstupný obraz. S tým súvisí aj potreba reagovať na udalosti, ktoré sa týkajú obrazu (kliknutie na určitom mieste obrazu, pohyb myši a pod.). Za normálnych okolností Ellipse spracováva udalosti, týkajúce sa dokumentu vlastnými obslužnými funkciami, ktoré sú ale programátorovi plug-in modulu neznáme a nedostupné.

Ellipse preto obsahuje mechanizmus, ktorý počas doby, keď je otvorený modul, zastaví spracovávanie týchto udalostí, a miesto toho ich posielajú do modulu na spracovanie. Keďže udalosti tohto typu sa štandardne spracovávajú v triede typu *CWnd*, bude aj naša trieda *CPluginView* tohto typu.

3.4 Príklad 3: Basics – modul demonštrujúci základné triedy

Kým príklad *Hello1* demonštroval použitie základných komunikačných funkcií, v tomto príklade bude demonštrovaná činnosť základných tried *CDlg* a *CPluginView*. Aj keď modul nevykonáva žiadnu konkrétnu činnosť, má kľúčový význam, keďže predstavuje „polotovár“, alebo „štartovaciu platformu“ s ktorej bude vychádzať konštrukcia ďalších modulov. Preto je na konci tejto kapitoly uvedený kompletný výpis zdrojového textu. Je tiež uvedený spôsob konštrukcie tohto modulu analogickým spôsobom, ako v príklade *Hello1*. Pre netrpezlivého čitateľa, ktorému by sa tento postup mohol zdať zdĺhavý, radšej vopred predznamenujeme, že v ďalších častiach už prevezme túto rutinnú činnosť pomocný program, tzv. *Plugin Wizard*. Pokiaľ by jeho znechutenie nad pomalým postupom malo dosiahnuť kritickú mieru, je možno rovno prejsť na kapitolu: **Plugin Wizard a jeho používanie**. Z hľadiska pochopenia činnosti modulu ale nezaškodí, si tento kód vytvoriť iba za pomoci tradičných prostriedkov.

Postup vytvorenia modulu bude nasledovný:

- Vytvorenie nového projektu *Basics* z kategórie *Input* a to analogickým spôsobom, ako sme vytvorili *Hello1*.
- modifikácia štyroch globálnych funkcií (*PluginOpen*, *PluginClose*, *PluginMessages*, *PluginDraw*) – vid' podfarbený text v súbore *Basics.cpp*. Všimnime si rozdiel oproti podfarbenému textu v súbore *Hello1.cpp*. Do *Basics.cpp* tiež pridáme deklaráciu smerníka budúceho dialógu *CDlg* m_pDlg* s príslušným hlavičkovým súborom.
- V Resource editore pridáme do projektu nový dialóg. Ten bude štandardne obsahovať iba dve tlačidlá *OK* a *Cancel*. Priradíme dialógu identifikátor *IDD_DLG* a názov (zvyčajne totožný s názvom modulu).
- Priradíme dialógu triedu. Robí sa to zvyčajne kliknutím pravého tlačítka myši na dialóg nasledované výberom položky *Add class*. Vyplníme názov triedy *CDlg* a ako rodičovskú triedu zadáme *CDialog* (v skutočnosti by to mala byť *CPluginDlg*, tú však v ponuke nemáme a preto použijeme zatiaľ *CDialog* a neskôr to ručne prepíšeme na *CPluginDlg*).
- Klikneme na projekt *Basics* a pomocou pravého tlačítka myši pridáme novú triedu *CPluginView*, ktorá je typu MFC a dedí vlastnosti od *CWnd*. To nám vygeneruje súbory *PluginView.h* a *PluginView.cpp*, ktoré tak isto upravíme podľa dole uvedeného výpisu.

Tento postup by mal vygenerovať zdrojový kód rozdelený do štyroch súborov. Ich kód by mal byť totožný s textom uvedeným vo výpise na konci tejto kapitoly, avšak bez podfarbených častí. Podfarbené časti, ako obvykle, znamenajú vložený alebo modifikovaný kód, ktorý si v ďalšej časti vysvetlíme.

Aby sme mohli zdrojový kód úspešne skompilovať a linkovať, je treba uskutočniť ešte jeden krok a síce sprístupniť v projekte knižnicu *elGraph*, ktorá obsahuje triedu *CPluginDlg*. To znamená, že nastavíme cestu k hlavičkovým súborom knižníc *elGraph* a IPP a tiež cestu k samotnej knižnici *elGraph.lib*. Urobíme to nasledovne:

- Vo vlastnostiach projektu *Basics* položku *C/C++ | General | Additional include directories* nastavíme na:
`../../../../elGraph, ../../include/IPP`
a tiež v položke *Linker | Input* nastavíme na:
`../../../../lib/elGraph.lib`.
- Skompilujeme a linkujeme projekt. Spustíme Eclipse a odštartujeme plug-in *Basics*. Ten by mal otvoriť dialógové okno s dvoma tlačidlami *OK* a *Cancel*.

Vysvetlime si teraz význam podfarbených častí kódu, ktoré predstavujú vložený alebo modifikovaný text oproti kódu, ktorý sme generovali hore uvedeným postupom.

Basics.cpp obsahuje statickú triedu *CDlg* a príslušný hlavičkový súbor. Okrem toho obsahuje 4 základné komunikačné funkcie. Funkcia *PluginOpen* je dostane ako parameter *lp* ukazovateľ na okno obrazu, ktorý vzápätí využije ako parameter pri konštrukcii triedy *CDlg*, ktorú hneď aj vytvorí pomocou funkcie *Create* a vzápätí zobrazí. Funkcia *PluginDraw* vždy keď je z *Ellipse* zavolaná tak zavolá funkciu *AddPluginDraw* triedy *CDlg*. Táto funkcia je virtuálnou funkciou rodičovskej triedy *CPluginDlg*, ktorá nevykonáva žiadnu činnosť. To v praxi znamená, že keď potrebujeme aby modul pridal vlastnú kresbu do obrázku, definujeme v triede vlastnú kresliacu funkciu *AddPluginDraw*, ktorá kreslenie vykoná, v opačnom prípade túto funkciu v triede *CDlg* nedefinujeme a tým pádom bude volaná miesto nej rovnomenná funkcia rodičovskej triedy, ktorá neurobí nič. Funkcia *PluginMessages* je dôležitá funkcia volaná z *Ellipse* vždy, keď je modul otvorený a teda spracovanie udalosti týkajúce sa okna dokumentu nevykonáva *Ellipse*, ale modul. Táto funkcia vlastne pošle správu, ktorú prijme od *Ellipse* ďalej do modulu *CPluginView*. Funkcia *PluginClose* je volaná z *Ellipse* vtedy, keď modul signalizuje ukončenie činnosti (stlačenie *OK* alebo *Cancel*), vtedy sa *Ellipse* musí postarať o vymazanie deštrukciu dialógu *CDlg*.

Dlg.h je hlavičkový súbor triedy *CDlg* a obsahuje aj deklaráciu smerníka na triedu *CPluginView*, ako aj prototyp funkcie *OnInitDialog*.

Dlg.cpp obsahuje predovšetkým modifikovaný konštruktor, ktorý má rodičovskú triedu *CPluginDlg* a nie *CDialog*, ako to ponúkol Class Wizard generujúci kód triedy na základe Resource. Túto zmenu je treba uskutočniť ručne. Konštruktor potom inicializuje konštrukciu triedy *CPluginView*, ktoré je vo funkcii *OnInitDialog* kreovaná (*Create*) a jej okno zobrazené (*ShowWindow*). Deštruktor triedy sa postará aj o vymazanie triedy *CPluginView*, ktorú vytvoril.

PluginView.h a **PluginView.cpp** obsahujú vložený kód týkajúci sa funkcie *Create* a tiež premennú *m_pDlg*, ktorá uchováva smerník na triedu *CDlg*.

Basics.cpp

```
#include "stdafx.h"
#include <afxdlx.h>
#include "Dlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

static AFX_EXTENSION_MODULE Templ01DLL = { NULL, NULL };
static CDlg* m_pDlg;

extern "C" __declspec(dllexport) int PluginOpen(WPARAM wp, LPARAM lp)
{
    CView* pImage = (CView*)lp;
    m_pDlg = new CDlg(pImage);
    m_pDlg->Create(CDlg::IDD);
    m_pDlg->ShowWindow(SW_SHOW);
}
```

```

        return(1);
    }

extern "C" __declspec(dllexport) int PluginDraw(WPARAM wp, LPARAM lp)
{
    m_pDlg->AddPluginDraw(lp);
    return(0);
}

extern "C" __declspec(dllexport) int PluginMessages(WPARAM wp, LPARAM lp)
{
    MSG* pMsg = (MSG*) lp;
    m_pDlg->m_pPluginView->SendMessage(pMsg->message,
                                     pMsg->wParam, pMsg->lParam);
    return(0);
}

extern "C" __declspec(dllexport) int PluginClose(WPARAM wp, LPARAM lp)
{
    m_pDlg->DestroyWindow();
    delete m_pDlg;
    return(0);
}

extern "C" int APIENTRY
DllMain(HINSTANCE hInstance, DWORD dwReason, LPVOID lpReserved)
{
    UNREFERENCED_PARAMETER(lpReserved);
    if (dwReason == DLL_PROCESS_ATTACH)
    {
        TRACE0("Templ01.DLL Initializing!\n");
        if (!AfxInitExtensionModule(Templ01DLL, hInstance))
            return 0;
        new CDynLinkLibrary(Templ01DLL);
    }
    else if (dwReason == DLL_PROCESS_DETACH)
    {
        TRACE0("Templ01.DLL Terminating!\n");
        AfxTermExtensionModule(Templ01DLL);
    }
    return 1;    // ok
}

```

Dlg.h

```

#pragma once
#include "resource.h"
#include "elGraph.h"
#include "PluginView.h"

class CDialog;
class CDlg : public CPluginDlg
{
    DECLARE_DYNAMIC(CDlg)

public:
    CDlg(CWnd* pParent = NULL);
    virtual ~CDlg();
    CPluginView* m_pPluginView;

```

```

        BOOL OnInitDialog(void);
        enum { IDD = IDD_DLG };

protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
support
        DECLARE_MESSAGE_MAP()

};

```

Dlg.cpp

```

#include "stdafx.h"
#include "Dlg.h"
#include "resource.h"
#include "PluginView.h"

IMPLEMENT_DYNAMIC(CDlg, CDialog)
CDlg::CDlg(CWnd* pParent /*=NULL*/) : CPluginDlg(pParent)
{
    m_pPluginView = new CPluginView(this);
}

CDlg::~CDlg()
{
    m_pPluginView->DestroyWindow();
    delete m_pPluginView;
}

void CDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CDlg, CDialog)
END_MESSAGE_MAP()

BOOL CDlg::OnInitDialog(void)
{
    m_pPluginView->Create();
    m_pPluginView->ShowWindow(SW_HIDE);
    return 0;
}

```

PluginView.h

```

#pragma once
#include "Dlg.h"

class CDlg;
class CPluginView : public CWnd
{
    DECLARE_DYNAMIC(CPluginView)

public:
    CPluginView(CDlg* pParent);
    virtual ~CPluginView();
}

```

```
        BOOL Create();
        CDlg* m_pDlg;

protected:
        DECLARE_MESSAGE_MAP()
};
```

PluginView.cpp

```
#include "stdafx.h"
#include "Basics.h"
#include "PluginView.h"
#include "Dlg.h"

IMPLEMENT_DYNAMIC(CPluginView, CWnd)
CPluginView::CPluginView(CDlg* pParent):CWnd()
{
    m_pDlg = pParent;
}

CPluginView::~CPluginView()
{
}

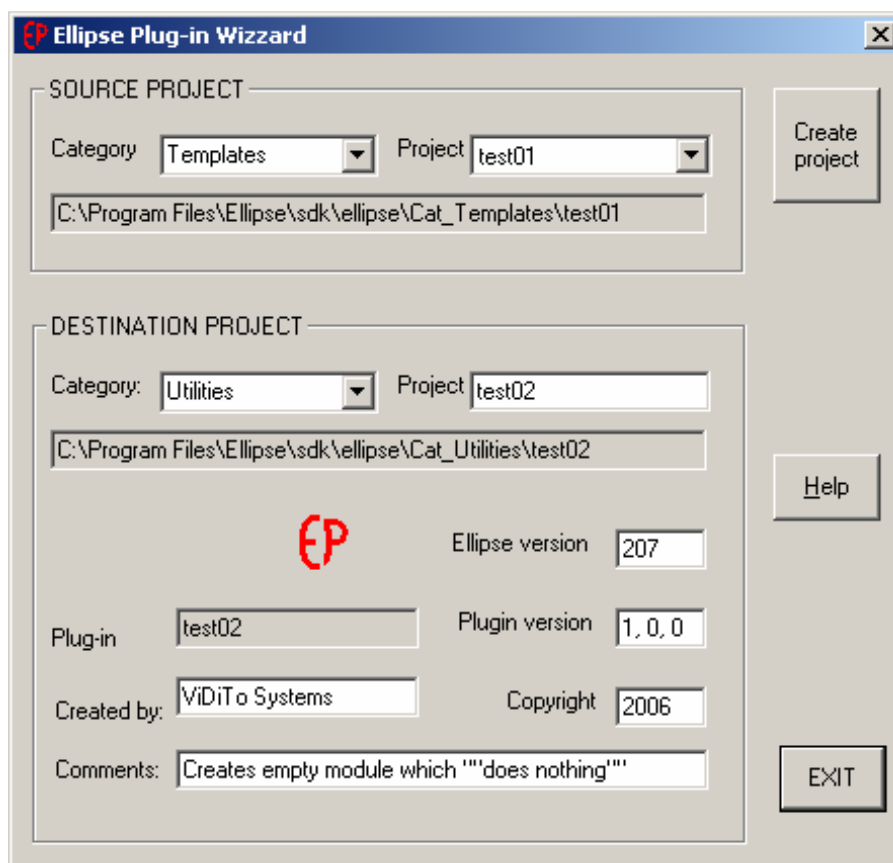
BEGIN_MESSAGE_MAP(CPluginView, CWnd)
END_MESSAGE_MAP()

BOOL CPluginView::Create()
{
    BOOL ret = CWnd::Create(NULL, "", WS_CHILD | WS_VISIBLE, CRect(0, 0,
        1, 1), (CWnd*)m_pDlg, 1234);
    return ret;
}
```

4 Plugin Wizard a jeho používanie

4.1 Čo je Plugin Wizard

Plug-in Wizard programu *Ellipse* má podobný cieľ ako napr. *Class Wizard*, alebo *Application Wizard* programu *Visual C++* a tým je odbremeniť programátora od rutinej a málo tvorivej práce pri ktorom sa vytvára základný kód aplikácie. Neznamená to, že milovníkom klasického programátorského štýlu je táto pomôcka vnucovaná. Ako to bolo ukázané v predošlých častiach, všetko, čo vygeneruje Wizard sa dá bez problémov naprogramovať aj „ručne“ za predpokladu dobrej znalosti rozhrania. Wizard však šetrí čas, ktorý by inak musel programátor stráviť štúdiom, písaním a ladením základného kódu. V neposlednom rade určitý „vnútený“ štýl zjednodušuje komunikáciu medzi jednotlivými autormi modulov a čitateľnosť cudzích programov.



Obr. 3

Základné dialógové okno Plugin Wizardu

Náš *Plugin Wizard* v podstate iba napodobuje to, čo robí väčšina programátorov postavených pred konkrétnu úlohu – snaží sa nájsť čo najpodobnejší funkčný program so zdrojovým kódom (vlastný, alebo cudzí) a modifikovať ho vo svoj prospech. Funkciu vzorových

príkladov v našom prípade tvoria jednotlivé príklady. Okrem toho umožní Plugin Wizard generovať nový projekt pomocou iného funkčného projektu .

Po spustení PluginWizard.exe sa objaví na obrazovke rovnaké okno ako je na Obr. 3. Ovládacie prvky v dialógovom okne sú rozdelené do dvoch základných skupín podľa toho, či patria do zdrojového projektu (SOURCE PROJECT), alebo cieľového projektu (DESTINATION PROJECT).

Jednotlivé položky dialógového okna majú nasledovný význam:

SOURCE PROJECT

Category kategória zdrojového projektu

Project názov zdrojového projektu

Obe položky je možné vybrať zo zoznamu existujúcich projektov a kategórii zobrazených pomocou ovládača typu ComboBox. Pod týmito ovládačmi je zobrazená celá cesta k adresáru, kde je uložený zdrojový projekt.

DESTINATION PROJECT

Category kategória cieľového projektu

Project názov cieľového projektu

Aj tu je možné vybrať kategóriu spomedzi existujúcich kategórii pomocou ComboBoxu. Navyše je ale v Combo Boxe aj položka **New Category** s možnosťou pridať do zoznamu novú kategóriu. Názov cieľového projektu je treba zadať v samostatnom editovacom okne. Aj pri cieľových projektoch sa vypisuje kompletná cesta k príslušnému adresáru.

Okrem spomínaných položiek sú k dispozícii aj editovacie okná, kde je možné upresniť prednastavené údaje týkajúce sa cieľového projektu:

Plug-in – názov modulu. Je totožný s názvom projektu, avšak s príponou dll. Tento názov je možné ešte zmeniť priamo pomocou Resource editora v položke *FileDescription* (viď príklad v časti 3.2).

Ellipse version - udáva s ktorou verziou Ellipse je modul kompatibilný. Niektoré staršie moduly fungujú aj pod novými verziami Ellipse, upozornia však pri štarte na možnú nekompatibilitu.

Plugin version – udáva verziu modulu. Môže, ale nemusí zodpovedať verzii Ellipse

Created by poskytuje možnosť uviesť výrobcu modulu, **Copyright** zasa jeho ochranu proti kopírovaniu. **Comments** by mal stručne charakterizovať činnosť modulu. Všetky spomínané údaje je možné vynechať a zadať ich priamo pomocou Resource editora.

Samotné generovanie zdrojového kódu nového projektu nastane po stlačení **Create project**.

4.2 Príklad 4: Hello2 - „Hello World“ pomocou Plugin Wizardu

Pretože *Basics* je minimálna funkčná verzia modulu, ktorá je „štartovacou plochou“. Preto nemá zmysel ju generovať, nakoľko sama slúži ako predloha na generovanie iných projektov, a preto sa budeme na tento príklad v ďalších častiach často odvolávať. Môžeme si tu ukázať, ako sa dá pomocou *PluginWizardu* a *Basics* generovať iný projekt ktorý sa od neho iba minimálne líši. Okrem „povinných“ tlačítok *OK* a *Cancel* bude náš príklad obsahovať navyše tlačítko „DO TEST“, ktorého stlačenie vyvolá otvorenie okna s nápisom „Hello World“.

- Spustíte Plugin Wizard. Zvoľte ako vzorový príklad *Basics* z kategórie *Input*, ktorý sme vytvorili v predošlých častiach a ktorého kompletný zdrojový text je uvedený na konci predošlej kapitoly.

- Zvoľte názov výstupného projektu napr. *Hello2*. Pretože chceme aby sa modul dal spustiť bez načítania dokumentu, ponecháme ho v kategórii *Input*. Vyplňte aj ostatné údaje v okne *Plugin Wizardu*. Nechajte generovať projekt *Hello2*.
- Spustíte Visual C++ a otvorte Solution Ellipse, podobne ako v časti 3.2. Pokiaľ Solution obsahuje nejaké projekty (napr. z minulých príkladov), vymažte ich. Načítajte projekt *Hello2* uložený v adresári *Cat_Input*. Zdrojový kód by mal byť totožný s tým z príkladu *Basics*.
- Pomocou *Resource Editoru* otvorte dialógové okno *IDD_DLG*, ktoré obsahuje iba tlačítka OK a Cancel. Pomocou lišty nástrojov pridajte tlačítko, pomenujte ho napr. *IDC_DOTEST* a priradte mu nápis „DO TEST“. Priradte mu obslužnú funkciu *OnBnClickedDotest*. Zdrojový kód tejto, zatiaľ prázdnej, funkcie bude pridaný na koniec súboru *Dlg.cpp* a deklarácia funkcie zasa do hlavičkového súboru *Dlg.h*. Vložte do vnútra tejto funkcie riadok s textom:
`AfxMessageBox(„Hello world“);` .
Výsledný zdrojový kód pridanej funkcie *OnBnClickedDotest* je nižšie zobrazený podfarbeným spôsobom.
- Skompilujte a linkujte projekt, spustíte *Ellipse* a načítajte ľubovoľný obraz zo súboru a potom spustíte modul *Hello2* z kategórie *Input*. Objaví sa dialógové okno, ktoré už bude obsahovať tlačítko s nadpisom *SHOW MESSAGE*, ktorého stlačenie generuje okno s nápisom „Hello World“

```
void CDlg::OnBnClickedDotest()
{
    AfxMessageBox("Hello World");
}
```

5 Bitmapy typu DDB a DIB

Operačný systém Windows používa na vnútornú reprezentáciu obrazu v pamäti formát, ktorému hovoríme „bitmapa“. Používajú sa dve verzie tohto formátu v závislosti od požiadaviek na rýchlosť spracovania, resp. na univerzálnosť a to DDB (Device Dependent Bitmap) a DIB (Device Independent Bitmap). [1].

DDB (Device Dependent Bitmap) je formát závislý na zariadení a používa sa hlavne v aplikáciách, kde je potrebné rýchle zobrazovanie. Rýchlosť zobrazovania je založená na tom, že táto bitmapa je už od začiatku prispôbena danému zobrazovaciemu zariadeniu (device context). Vďaka tomu môžeme vytvoriť bitmapu kompatibilnú so zobrazovacím zariadením a veľmi rýchlymi kopírovacími funkciami typu BitBlt (Bit Block Transfer) preniesť na kresliacu plochu obrazovky obraz pripravený v pamäti. My sa v ďalšej časti sústreďíme na druhú z nich (DIB), nakoľko táto je v moduloch používaná najčastejšie a pre jednoduchosť ju budeme v ďalšom texte často stotožňovať s pojmom bitmapa.

5.1 Bitmapa DIB (Device Independent Bitmap)

Jej nezávislosť od technických prostriedkov je možná vďaka tomu, že obsahuje kompletnú informáciu o obraze, vďaka čomu je zobraziteľná na rôznych technických zariadeniach. Keďže podrobný popis štruktúry DIB je súčasťou dokumentácie Windows, obmedzíme sa len na základné informácie o bitmape, ktoré budeme potrebovať na úspešné programovanie modulu.

Bitmapa načítaná do operačnej pamäte pozostáva z a) hlavičky *Bitmap Info Header*, b) tabuľky farieb *Palette* (môže byť vynechaná) a c) bloku pamäte *Bits* predstavujúceho jednotlivé pixely snímané po riadkoch. Pokiaľ je bitmapa uložená v súbore (zvyčajne s príponou .BMP), predchádza tomu ešte zvláštna hlavička s informáciami o súbore.

Hlavička (Bitmap info header) obsahuje informáciu o rozmeroch obrázku (*Width, Height*), počte bitov potrebných na kódovanie 1 pixelu (*BPP* - Bits per Pixel) a niektoré ďalšie údaje, ktorých význam momentálne nepotrebujeme poznať. Údaj *BPP* je určujúcim pre rozdelenie bitmáp na *indexované* a tzv. „*true color*“ bitmápy. Indexovaná bitmapa (pre *BPP*=8 alebo menej) je taká, kde údaje v poli *Bits* nepredstavujú priamo farbu, ale poradové číslo v tabuľke farieb. Takže napr. ak *BPP*=8, každý bajt v poli *Bits* predstavuje jednu z 256 farieb tabuľky.

Tabuľka farieb (*Palette*) predstavuje postupnosť 32 bitových údajov (tzv. RGBQUAD), kde tri najmenej významné bajty kódujú jednotlivé farebné zložky v poradí R (červená), G (zelená), B (modrá), pričom najvýznamnejší bajt je nulový tak napr. hodnota 0x000000ff predstavuje čisto červenú farbu, 0x0080FFFF zasa farbu žltú. V prípade bitmápy typu „*true color*“ je tabuľka farieb nemá význam, nakoľko farbu pixelu určujú priamo údaje v poli *Bits*.

Pole Bits predstavuje u indexovaných bitmáp poradové číslo v tabuľke farieb a v prípade „*true color*“ priamo farbu pixelu. Takže pre *BPP*=24 sú po sebe idúce bajty združované do trojíc (B, G, R). V prípade *BPP*=32 sú bajty združované do štvoric (A, B, G, R), kde R, G, B sú jednotlivé farebné zložky a A je buď nevyužitá, alebo predstavuje tzv. *Alfa kanál*, ktorý môže obsahovať informáciu o transparentnosti pixelu a pod. Takto kódované pixely sú ukladané postupne po riadkoch, pričom dĺžka riadku v bajtoch nezodpovedá priamo šírke obrazu násobenej počtom bajtov na pixel ($Width * BPP / 8$), ale je zaokrúhľená na najbližšiu vyššiu 32-bitovú hodnotu, ktorá sa nazýva *Pitch*. Táto hodnota (musí byť bezo zvyšku

deliteľná 4) predstavuje vzdialenosť v bajtoch medzi jednotlivými riadkami. Takže napr. ak $Width=256$ a $BPP=8$, potom $Pitch=256$. Ak ale napr. $Width=257$ a $BPP=24$, potom $Pitch=772$.

Bitmapa je zvyčajne zobrazovaná tak, že prvý riadok (začiatok poľa *Bits*) sa zobrazuje na spodku obrazu a posledný na jeho vrchu. To v rozpore so súradným systémom obrazovky, kde počiatok predstavuje ľavý horný roh obrazovky (okna). Preto ak potrebujeme napr. zistiť hodnotu pixelu pod kliknutým bodom $p(x,y)$, musíme y -ovú súradnicu upraviť nasledovne:

$p.y = (H-1) - p.y$, kde H je výška obrazu

5.2 Triedy *ATL::CImage* a *CImg* na podporu práce s bitmapou.

Bohužiaľ, pomerne dlho nebola DIB implementovaná do knižnice MFC od Microsoftu, čo viedlo jednotlivých vývojárov k implementovaniu vlastných tried (jedna takáto trieda *CDib* je popísaná v [1] a slúžila za základ starších verzií *Ellipse*). Až v poslednom čase je v rámci rozhrania GDI+ k dispozícii trieda *CImage*, ktorá je základom nových verzií *Ellipse* a preto si ju v ďalšom popíšeme podrobnejšie.

Trieda *CImage* je súčasťou knižnice ATL a preto je dobrým zvykom používať v texte popis *ATL::CImage*. Obsahuje všetky základné funkcie potrebné na získanie údajov popísaných v predošlej časti. V prostredí Visual Studio je k tejto triede rozsiahla dokumentácia. Práca s triedou *ATL::CImage* je veľmi pohodlná. Ako je to v prostredí Windows časté, aj v tomto prípade pozostáva vytvorenie objektu z niekoľkých fáz, a to najprv použitie konštruktora, potom použitie funkcie *Create* so zadaním rozmerov a *BPP*. Tým sa vytvorí „prázdna bitmapa“, ktorá má alokované potrebné miesto v pamäti avšak aj tabuľka farieb, aj samotné dáta v poli *Bits* sú iniciované na nulu a treba ich naplniť potrebným obsahom.

Trieda *CImg* je našim vlastným rozšírením triedy *ATL::CImage* pričom bolo našou snahou modifikácie minimalizovať a ponechať iba tie, ktoré podľa nášho názoru zjednodušujú prácu pri plnom rešpektovaní vlastností triedy *ATL::CImage*. V rámci filozofie objektovo orientovaného programovania je trieda *CImg* priamo odvodená od *ATL::CImage* ako je to vidieť z jej deklarácie

```
class AFX_EXT_CLASS CImg : public ATL::CImage
```

Vďaka tomu je možné používať v triede *CImg* všetky funkcie triedy *ATL::CImage*. Iba niektoré funkcie sme prepísali pretože, podľa nášho názoru, zjednodušujú prácu a zabezpečujú lepšiu kompatibilitu s doteraz napísaným kódom. Pribudol tzv. *Copy* konštruktor a tiež operátor priradenia, ktoré v rodičovskej triede chýbajú a pritom veľmi zjednodušujú prácu. Keďže sme sa v praxi zatiaľ nestretli s inou bitmapou než takou, ktorá má prvý riadok na spodku obrázku, fixovali sme novú triedu na tento prípad. To znamená, že modifikovaná funkcia *GetPitch()* vráti vždy kladnú hodnotu a funkcia *GetBits()* vráti vždy ukazovateľ na miesto bezprostredne nasledujúce po tabuľke farieb (ak existuje), čiže na prvý (spodný) riadok bitmapy. Okrem toho sme triedu *CImg* doplnili o niektoré jednoduché funkcie napr. *SetColorTable*, ktorá v indexovanej bitmape nastaví lineárnu šedotónovú paletu v rozsahu 0-255. V nasledujúcom výpise je zoznam najpoužívanejších funkcií triedy *CImg*. Ich podrobnejší popis je v časti 11.1 (funkcie špecifické pre *CImg*) a 11.2

(funkcie spoločné pre *CImg* a *ATL::CImage*). Pri spoločných funkciách je väčšinou iba odkaz na originálnu dokumentáciu *ATL::CImage*.

CImg

Funkcie špecifické pre CImg

CImg	default konštruktor
CImg(const CImg& Img);	copy konštruktor
CImg(LPVOID lpMemory);	konštruktor na základe DIB v pamäti
~CImg(void);	deštruktor
operator=	operátor priradenia
IsAlpha();	vráti informáciu, či obrázok obsahuje alfa kanál
GetPixelIndex	zistí index v tabuľke farieb pre príslušný bod
IsGrayscale	vráti informáciu, či obrázok je šedotónový
GetDimensions	vráti rozmery obrázku v tvare CSize
GetPitch	vráti hodnotu Pitch-vzdialenosť medzi riadkami v bajtoch.
GetBits	ukazovateľ na začiatok poľa Bits
SetGrayColorTable	vytvorí tabuľku farieb pre šedotónový obraz

Funkcie spoločné pre CImg a ATL::CImage

Create	kreovanie bitmapy zadaním rozmerov a BPP.
GetWidth	vráti šírku obrazu v pixeloch
GetHeight	vráti výšku obrazu v pixeloch
GetBPP	vráti počet bitov/pixel (BPP zvyčajne 8, 24, 32)
IsIndexed	udáva typ bitmapy (Indexovaná alebo True Color)
GetColorTable	Skopíruje tabuľku farieb do pripraveného buffera
SetColorTable	Naplní tabuľku farieb bitmapy hodnotami z bufferu
GetPixel	vráti farebnú hodnotu pixelu o súradniciach (x,y)
SetPixel	nastaví farbu pixelu majúceho súradnice (x,y).
Load	načíta bitmapu zo súboru *.BMP (pozná aj iné formáty)
Save	uloží bitmapu do súboru vo zvolenom formáte
Draw	vykreslenie bitmapy do zadanej kresliacej plochy (DC)
GetMaxColorTableEntries	maximálny počet hodnôt v tabuľke farieb

5.3 Príklad 5: CImgDemo - demonštrácia použitia triedy CImg

Tento jednoduchý príklad *CImgDemo* ukazuje, ako je možné zmeniť hodnotu pixelov v bitmape použitím triedy *CImg*. Nech je úlohou modulu vytvoriť v spodnej polovici obrazu zrkadlový obraz jeho hornej polovice. Na tento účel vygenerujeme nový projekt použitím predchádzajúceho vzorového príkladu. Pretože zatiaľ sme sa pri vysvetľovaní nedostali k tomu, ako modul získava informácie o vstupnom obraze načítanom a zobrazovanom pomocou *Ellipse*, budeme v tomto prípade vstupný obraz načítavať zo súboru a výstupný obraz sa uloží v rovnakom formáte a pod rovnakým názvom, len s príponou *_mod*.

- Vygenerujte nový projekt *CImgDemo* zaradený do kategórie *Input*, pričom ako vzor sa použije projekt *Hello2* z kategórie *Input*.
- modifikujte zdrojový kód funkcie *OnBnClickedDotest()* v súlade so zadanou úlohou.
- Overte činnosť modulu po stlačení tlačítka DO TEST na šedotónových i farebných obrázkoch.

```

void CDlg::OnBnClickedDotest()
{
    const char szFilter[] =
        "Windows or OS/2 Bitmap(*.bmp)|*.bmp|
        Tagged Image Format(*.tif)|*.tif|
        Image Files (*.bmp, *.tif,*.jpg)|*.bmp;*.tif;*.jpg||";

    CFileDialog dlg(true, "*.bmp", "blobs", 0, szFilter);
    dlg.DoModal();
    CString Path = dlg.GetPathName();

    CImg Img;
    Img.Load(Path);
    int W = Img.GetWidth();
    int H = Img.GetHeight();
    int P = Img.GetPitch();

    for(int i=0; i<H/2; i++){
        BYTE* pSrcRow = (BYTE*)Img.GetBits() + i*P;
        BYTE* pDestRow = (BYTE*)Img.GetBits() + (H-1-i)*P;
        CopyMemory(pDestRow, pSrcRow, abs(P));
    }

    Path.Replace(".", "_mod.");
    Img.Save(Path);
    Img.ReleaseGDIPlus();
}

```

6 Kreslenie do okna obrazu, overlay, objekty, polygóny

Dôležitou súčasťou všetkých vyspelejších programov na spracovanie obrazu je možnosť nedeštruktívne kresliť do okna obrazu. Môžeme si to predstaviť ako priesvitnú fóliu (**overlay**), ktorú položíme na fotografiu a vykreslíme na nej napr. obrisy objektov na fotografii. Fóliu môžeme kedykoľvek mazať, odstrániť, alebo naopak pridať ďalšiu fóliu, na ktorej budú napr. inou farbou vykreslené obrisy iného typu objektov. Pritom obsah fotografie (dáta bitmapy) nebudú nijakým spôsobom zmenené, keďže sa mení iba hodnota pixelov kópie bitmapy na kresliacej ploche obrazovky (device contents DC). V prípade stacku obrazov musí mať každý obraz vlastný overlay. Overlay môže mať najrôznejšiu podobu. Môže to byť napr. pomocný obrázok, ktorý sa „priloží“ na obraz, ale tak isto to môže byť text, alebo kontúra definovaná polygónom. V prípade 32 bitových obrazov sa na tento účel s výhodou využíva *Alfa kanál*, využívajúci voľný bajt 4-bajtovej hodnoty (zvyšnými tromi sú kódovane zložky R, G, B).

Predpokladáme, že čitateľovi je známy mechanizmus, akým aplikácia prekresľuje jednotlivé okná (pohľady). Stručne povedané, okno má svoju funkciu `OnDraw`, v ktorej je naprogramované, čo sa má do okna nakresliť. Táto funkcia je volaná vždy, keď vznikne požiadavka na prekreslenie okna či už od systému, alebo od užívateľa. Systém automaticky vysiela túto požiadavku vtedy, keď zaregistruje, že sa okno pohlo, maximalizovalo, rozťahlo a pod. Programátor zasa má možnosť zadať požiadavku na prekreslenie okna napr. pomocou funkcií `RedrawWindow()`, alebo `Invalidate()`.

6.1 Príklad 6: DrawDemo - kreslenie z modulu a funkcia AddPluginDraw

V predošlých častiach bolo naznačené, že pomocou funkcie `AddPluginDraw` dokáže modul kresliť do obrazu načítaného a zobrazovaného pomocou `Ellipse`. Presnejšie povedané, modul pridá svoju kresbu k tomu, čo práve nakreslila `Ellipse`. Nasledovný príklad demonštruje nakreslenie čiary, ktorá vychádza z ľavého horného rohu a ide pod uhlom 45 stupňov smerom doľava a dole. Dĺžka čiary je na začiatku 10 pixelov a predlžuje sa každým kliknutím na tlačítko DO TEST. Je zrejmé, že nadišiel čas opustiť tvorbu modulov iba z kategórie `Input`, ktoré nepotrebovali vstupný obrázok. Tentoraz budeme kresliť priamo do vstupného obrázku z `Ellipse`.

- Pomocou `Plugin Wizardu` vytvoríme novú kategóriu `Test` a generujeme nový projekt `DrawDemo`. Ako vstupný projekt použijeme opäť príklad `Hello2` z kategórie `Input`.
- Do súboru `Dlg.h` pridáme členskú premennú `m_Dist`, ktorá určuje dĺžku nakreslenej čiary. V súbore `Dlg.cpp` uskutočníme úpravy, ktoré zobrazuje podfarbený text

Treba pripomenúť, že funkcia `AddPluginDraw`, ktorá kreslí čiaru určitej dĺžky, nemusela byť doteraz nikde definovaná vďaka tomu, že jej úlohu plnila virtuálna funkcia rodičovskej triedy `CPluginDlg`. Tá nerobila nič (pretože sme doteraz neopotrebovali nič kresliť z modulu), jedinou jej úlohou bolo „byť volaný“ z komunikačnej funkcie `PluginDraw`. Každé kliknutie tlačítka DO TEST a teda aj volanie funkcie `OnBnClickedDotest()`, inkrementuje premennú `m_Dist` a teda aj dĺžku kreslenej čiary. Dôležité je tiež volanie funkcie `GetView()`, vďaka ktorej môžeme po každom kliknutí testovacieho tlačítka vyslať do `Ellipse` príkaz na

prekreslenie obsahu okna (bez toho by sme narastanie čiary videli iba vtedy, ak by sme umelo vyvolali podobný príkaz od systému, teda napr. by sme zmenšili a vzápätí zväčšili okno.

```
BOOL CDlg::OnInitDialog(void)
{
    m_Dist = 1;
    m_pPluginView->Create();
    m_pPluginView->ShowWindow(SW_HIDE);
    return 0;
}

void CDlg::OnBnClickedDotest()
{
    CView* pView = GetView();
    m_Dist++;
    pView->Invalidate();
}

void CDlg::AddPluginDraw(LPARAM lParam)
{
    CDC* pDC = (CDC*)lParam;
    pDC->MoveTo(0,0);
    pDC->LineTo(10*m_Dist,10*m_Dist);
}
```

6.2 Objekty v overlay a ich reprezentácia polygónmi

Objekt, ktorý bol vykresľovaný v predchádzajúcom príklade, bol veľmi jednoduchý, keďže bol vlastne úsečkou idúcou z počiatku súradného systému po 45 stupňovým uhlom do určitej vzdialenosti. Je zrejmé, že pre vykresľovanie viacerých objektov v overlay potrebujeme dômyselnejšiu formu, ako ich uchovávať a naraz všetky vykresliť.

V takomto prípade je rozumné nahradiť všeobecný pojem overlay špecifickým pojmom **OverlayPoly**, čo je sada kresieb rôznych objektov, ktoré sú reprezentované pomocou **polygónov**, pričom kresliace prostriedky (farba a hrúbka pera, štetec) sú definované v *Ellipse options*. V budúcnosti plánujeme okrem *OverlayPoly* použiť aj iné typy overlay (*OverlayText*, *OverlayImg*).

Polygón definujeme ako postupnosť bodov ktoré reprezentujú uzavretú obrysovú kontúru, ale môžeme ho použiť aj na reprezentáciu objektov ako sú lomená čiara (multi-line), úsečka (line), ramená uhla (angle), alebo aj samostatný bod (point). Aj keď všetky spomínané typy geometrických útvarov sú reprezentované tou istou triedou *CPolygon*, odlišenie typu je možné na základe vnútornej premennej *m_Type*, ktorá sa nastaví napr. na základe použitého kresliaceho prostriedku. Táto premenná je typu *public* a môže sa nastaviť priamo programom. Pri použití niektorého kresliaceho prostriedku sa *m_Type* nastaví automaticky. Okrem toho je každý polygón charakterizovaný triedou (*Class*) umožňujúcou zlučovať polygóny do skupín, ktoré sú potom vykresľované rovnakou farbou a tiež číselnou nálepkou (*Label*), ktorá sa zobrazí vedľa polygónu. Hodnota *Label* sa zvyčajne inicializuje v poradí, v akom sú polygóny vytvárané. Aj keď *Label* nezodpovedá jednoznačne indexu v poli polygónov, predsa tento mechanizmus zabezpečuje, aby na jednej hladine *OverlayPoly* neboli dva polygóny s rovnakým *Labelom*. Pri vykresľovaní sa interpoluje úsečka medzi bodmi polygónu, čo umožní nielen jeho plnohodnotné vykreslenie, ale aj výpočet pozície interpolovaných bodov za účelom výpočtu parametrov.

OverlayPoly teda slúži na uchovávanie objektov v tvare polygónu a preto bola definovaná trieda *COverlayPoly*, ktorá túto filozofiu programátorsky zastrešuje.

Trieda COverlayPoly

Je to jednoduchá trieda ktorá uchováva pole polygónov, teda objektov typu *CPolygon*. Pokiaľ užívateľ kliknutím označí určitý polygón ako „aktívny“, bude tento pri zobrazovaní zvýraznený, pričom trieda *COverlayPoly* poskytuje informáciu o tom ktorý polygón je práve aktívny.

COverlayPoly

COverlayPoly	konštruktory (default konštruktor a copy konštruktor)
Add	umožní pridať do OverlayPoly nový polygon
GetSize	vráti počet polygónov v OverlayPoly
operator=	umožní vytvoriť nový OverlayPoly zápisom Ovr2 = Ovr1
operator[]	prístup k polygónu v poli (napr. NewPoly = Ovrly[10])
InWhichPol	určí index polygónu, ktorý úplne prekryje daný polygón
PtInWhichPol	určí index polygónu vnútri ktorého sa nachádza daný bod
LabInWhichPoly	určí index polygónu s daným Labelom

CPolygon

CPolygon	konštruktory (default, copy, polygon tvaru obdĺžnika)
GetLabel/SetLabel	funkcie na zistenie, resp. nastavenie hodnoty <i>Label</i>
GetClass/SetClass	funkcie na zistenie, resp. nastavenie hodnoty <i>Class</i>
PtInPolygon	určí, či sa daný bod nachádza vnútri polygónu
DPToLP	prevod zo súradníc zariadenia na logické súradnice
LPtoDP	prevod z logických súradníc na súradnice zariadenia
Shift	posunie polygón o určitú vzdialenosť v smere (x,y)
Area	veľkosť plochy (v pixeloch) vnútri uzavretého polygónu
Length	vypočíta obvod (súčet vzdialeností medzi bodmi)
GetBoundingRectLP	nájde obdĺžnik, ktorý kompletne ohraničí daný polygón
GetRefreshRectDP	nájde obdĺžnik vhodný na refresh (ohraničí polygón+nápis)
InvalidatePolygon	zabezpečí refresh nakresleného polygónu
Overlapped	určí, či dva polygóny majú spoločný prienik
CalculateAttributes	spočíta viacero preddefinovaných parametrov polygónu
Profile	nájde hodnoty pixelov pozdĺž polygónu (aj interpolovaná)
ReducePoints	zredukuje počet bodov ponechá iba každý n-ty
ReduceLines	zredukuje počet bodov - vynechá body na priamke

funkcie totožné s CArray

operator =	operátor priradenia, umožní zápis CPolygon Poly2=Poly1
operator[]	prístup k prvku poľa (napr. CPoint P2 = Poly[10])
~CPolygon	deštruktor polygónu
SetSize	nastaví veľkosť poľa
GetSize	určí veľkosť (počet bodov) reprezentujúcich polygón
SetAt	vloží bod na miesto poľa dané hodnotou Idx
InsertAt	vloží bod za miesto poľa dané hodnotou Idx (pridá nové)
RemoveAt	odstráni prvok poľa na určitom mieste danom hodnotou Idx
RemoveAll	odstráni všetky vrcholy polygónu (vynuluje pole)
Add	pridá prvok na koniec poľa
GetData	vráti smerník na začiatok poľa prvkov v tvare CPoint*

6.3 Príklad 7: OverlayPolyDemo - vykresľovanie pomocou OverlaPoly

V tomto príklade nakreslíme do obrázku objekty zložitejšieho tvaru (hviezdičky, štvorce, kruhy, trojuholníky) a to tak, že v každom riadku sa vykreslia rovnaké objekty v pravidelných vzdialenostiach od seba. Každý nasledujúci riadok bude obsahovať objekty iného typu, takže celkovo sa zobrazia 4 riadky a v každom 4 rovnaké objekty.

- Vygenerujeme projekt *OverlayPolyDemo* v kategórii *Test* a to použitím *Plugin Wizardu*, kde ako vzor slúži predošlý príklad *DrawDemo* z kategórie *Test*.
- Pri modifikácii zdrojového kódu budeme modifikovať iba súbory týkajúce sa triedy *CDlg*. Do *Dlg.h* pridáme premennú *m_OverlayPoly* triedy *COverlayPoly*, ktorá bude archivovať jednotlivé objekty. Vymažeme premennú *m_Dist*. Podstatne viac budeme modifikovať funkcie obsiahnuté v súbore *Dlg.cpp* tak, ako sú zobrazené v podfarbenom texte

Dlg.cpp

```
void CDlg::OnBnClickedDotest()
{
    CRect Rect;
    CView* pView = GetView();
    pView->GetWindowRect(Rect);
    CSize step(Rect.Width()/4, Rect.Height()/4);

    int R = 5;
    int r = (3*R)/4;
    int a = R/2;

    int Pt[4][8][2] = {
        {{R,R},{0,R},{-R,R},{-R,0},{-R,-R},{0,-R},{R,-R},{R,0}},
        {{r,r},{0,R},{-r,r},{-R,0},{-r,-r},{0,-R},{r,-r},{R,0}},
        {{R,R},{0,a},{-R,R},{-a,0},{-R,-R},{0,-a},{R,-R},{a,0}},
        {{R,R},{R,R},{0,R},{0,R},{-R,R},{-R,R},{0,-R},{0,-}},
    };

    CPolygon Obj[4];
    for(int j=0; j<4; j++){
        for(int i=0; i<8; i++){
            Obj[j].Add(CPoint(0,0)+CSize(Pt[j][i][0],Pt[j][i][1]));
        }
        Obj[j].Shift(CSize(step.cx/2, step.cy/2 + j*step.cy));
    }

    m_OverlayPoly.RemoveAll();
    for(int row=0; row<4; row++){
        for(int col=0; col<4; col++){
            CPolygon poly = Obj[row];
            poly.Shift(CSize(col*step.cx, 0));
            m_OverlayPoly.Add(poly);
        }
    }
    pView->Invalidate();
}

void CDlg::AddPluginDraw(LPARAM lParam)
{
    CDC* pDC = (CDC*)lParam;
```

```
for(int i=0; i<m_OverlayPoly.GetSize(); i++)
    pDC->Polygon(m_OverlayPoly[i].GetData(),
m_OverlayPoly[i].GetSize());
}
```

Funkcia programu je programátorsky realizovaná nasledovne: Na uloženie objektov, ktoré sa budú vykresľovať slúži premenná *m_OverlayPoly* triedy *COverlayPoly*. V nej sú uložené všetky polygóny, ktoré potom funkcia *AddPluginDraw* jednoducho v slučke vykreslí. Takže kľúčovou úlohou sa stáva správne uloženie polygónov do *m_OverlayPoly*, ktoré sa realizuje vo funkcii *OnBnClickedDotest()*. V nej sa najprv definujú súradnice jednotlivých vrcholov polygónu predstavujúceho základný objekt umiestnený do stredu súradného systému. Premenné *R*, *r* a *a* pomáhajú definovať tieto súradnice v relatívnej mierke vzhľadom na veľkosť *R*. Pretože pomocou funkcie *GetView()* získame smerník na pohľad, ktorý je vlastne oknom obsahujúcim obraz, vieme zistiť rozmer tohto okna a vypočítať hodnotu *step* reprezentujúcu vzdialenosť medzi objektami v smeroch X a Y. Po vypočítaní týchto hodnôt je naplnenie *m_OverlayPoly* triviálnou úlohou. Každý základný objekt je v smere osi Y posunutý o hodnoty *step.cy*. V slučke sa potom každý takýto objekt posunie o hodnotu *step.cx* v smere osi X a pridá do *m_OverlayPoly*.

7 Trackery, kreslenie základných geometrických objektov

V predošlej časti bolo uvedené, že základným geometrickým útvarom, ktorý používa *OverlayPoly* je mnohoúhelník (polygón) reprezentovaný ako postupnosť bodov – vrcholov. Za polygón považujeme aj útvary s redukovaným počtom vrcholov ako **bod** (1 vrchol), **úsečka** (2), **uhol** (3), **obdĺžnik** (4), **multi-úsečka** (neuzavretý polygón ktorému chýba spojnica posledného a prvého vrcholu). **Polygón** je možné určiť buď zadaním všetkých vrcholov alebo kreslením voľnou rukou (**Freehand**).

Je jasné, že kreslenie každého takéhoto útvaru vyžaduje odlišnú stratégiu ovládania tlačítok a pohybu myši, napr. bod je určený kliknutím, obdĺžnik stlačením ľavého tlačítka myši v prvom vrchole, posúvaním myši a uvoľnením tlačítka v pozícii kde sa nachádza druhý bod uhlopriečky želaného obdĺžnika (technika Drag and Drop). Aj napriek rozličnej stratégii kreslenia majú všetky tieto techniky spoločnú filozofiu a tak je prirodzené použitie mechanizmu dedičnosti, keď pri realizácii sa vytvorila základná trieda *CTracker* a jej jednotlivé varianty (*CTrackerPoint*, *CTrackerLine*, *CTrackerRectangle*, *CTrackerPoly*). Väčšina týchto tried obsahuje funkciu *TrackRubberBand*, ktorá definuje pomery počas kreslenia daného útvaru (napr. pri kreslení obdĺžnika sa po zadaní prvého vrcholu a počas pohybu myši kreslí „gumová čiara“ ktorá ukazuje aktuálny stav. Po uvoľnení ľavého tlačítka myši sa táto dočasná kresba zmení na trvalú s definovanou farbou a hrúbkou čiary. Používanie trackerov zjednodušuje kreslenie natoľko, že programátor nemusí ošetrovať jednotlivé kliknutia myši, stačí keď po vytvorení trackera zavolá funkciu *TrackRubberBand*, ktorá už zabezpečí obsluhu jednotlivých udalostí.

7.1 Trieda CTracker

CTracker

Funkcie

CTracker	2 konštruktory: a) kreslenie polygónu b) úprava polygónu
~CTracker	deštruktor
MovePolygon	posunutie existujúceho polygónu
MoveVertex	posunutie určitého vrchola polygónu
InsertVertex	vloženie nového vrchola do polygónu
RemoveVertex	odstránenie určitého vrchola polygónu
TrackRubberBand	funkcia zabezpečujúca kreslenie polygónov rôzneho typu
HitTest	určí, ktorá časť existujúceho polygónu bola kliknutá (hitNothing, hitLabel, hitInside, hitVertex, hitEdge)

verejné premenné

m_polygon	premenná typu CPolygon obsahujúca nakreslený objekt
int m_nHit;	udáva index kliknutého vrcholu (za predpokladu, že funkcia HitTest vráti hodnotu hitVertex.
struct CMode	mod trackera daný hrúbkou a farbou čiary pri kreslení {BOOL m_bRubber; COLORREF m_col; int m_nWidth;} m_mode;

odvodené triedy

<i>CTrackerPoint</i>	Tracker vhodný na kreslenie bodov (funkcia Track)
<i>CTrackerLine</i>	Tracker vhodný na kreslenie úsečiek (TrackRubberBand)
<i>CTrackerRectangle</i>	Tracker vhodný na kreslenie obdĺžnikov (TrackRubberBand)
<i>CTrackerPoly</i>	Tracker vhodný na kreslenie polygónov (funkcie TrackRubberBand alebo TrackFreehand)

Popis konštruktorov a funkcií.

Existujú 2 konštruktory triedy *CTracker*. Obidva dostávajú ako parameter smerník na triedu *CView*, čiže na okno, v ktorom sa tracker vykresľuje resp. zobrazuje. Tak isto obidva konštruktory obsahujú ako parameter mód, v ktorom sú vykresľované. Tento mód udáva štruktúra *CMode*, ktorá obsahuje boolovskú premennú *m_Rubber* udávajúcu, či pri kreslení je použitá tzv. „gumová čiara“ alebo farba *m_col*. Parametrom *m_nWidth* je nastavená šírka pera. Štruktúra sa využíva len pri zadávaní nového polygónu. Pri editovaní sa používa len gumová čiara.

Rozdiely medzi konštruktorami sú v účele ich použitia. Prvý z nich (s dvoma parametrami) slúži výhradne na kreslenie, druhý na editovanie už nakresleného polygónu, ktorý sa konštruktoru poskytne ako referencia *CPolygon&* na už nakreslený existujúci polygón. Počiatočný resp. výsledný polygón je uložený v tomto polygóne.

MovePolygon – Funkcia na posun polygónu. Počas činnosti funkcie sú správy presmerované a spracované v tele funkcie. Parametrom je referencia na existujúci polygón, kde po vykonaní posunu je uložený výsledný polygón. Druhý parameter je bod typu *CPoint*, voči ktorému je posun vypočítaný. Počas posunu je polygón prekreslený „gumovou čiarou“. Návrátová hodnota *false* indikuje chybu pri vykonaní operácií, napr. nepodarilo sa zistiť smerník na *mainframe*, alebo na *statusbar*.

MoveVertex – Funkcia na posun jedného vrcholu polygónu. Index daného vrcholu je jednoznačne identifikovaný členskou premennou *m_nHit*, ktorý môže byť nastavený buď funkciou *HitTest*, alebo priamo programátorom. Počas činnosti funkcie sú správy presmerované a spracované v tele funkcie. Parametrom je referencia na polygón, kde je po vykonaní posunu uložený aj výsledný polygón. Druhý parameter je bod *CPoint*, voči ktorému je posun vypočítaný. Počas posunu je polygón prekreslený „gumovou čiarou“. Návrátová hodnota *false* indikuje chybu pri vykonaní operácií, napr. nepodarilo sa zistiť smerník na *mainframe*, alebo na *statusbar*.

InsertVertex – vloží bod *pt* do polygónu *polygon*. Miesto vloženia je jednoznačne určené parametrom *m_nHit*, ktorý môže byť nastavený buď priamo programátorom, alebo funkciou *HitTest*.

RemoveVertex – odstráni vrchol polygónu na mieste špecifikovanom pomocou členskej premennej *m_nHit*.

HitTest – funkcia testuje, kde sa nachádza bod *pt* vzhľadom na *polygon*, pričom návratová hodnota môže byť *hitNothing*, *hitLabel*, *hitVertex*, *hitEdge*, *hitInside*, čiže bod nie je v blízkosti polygónu, bod je blízko k značke/vrcholu/hrane polygónu, resp. sa nachádza vo vnútri polygónu (len u uzavretých polygónov). Ak bod leží blízko k vrcholu resp. hrane, potom členská premenná *m_nHit* bude indexom daného vrcholu, resp. koncového bodu hrany. Pomocou parametra *nRectSize* sa dá určiť okolie bodu v pixeloch, v ktorom pracuje *HitTest* (predvolená hodnota je 10).

7.2 Príklad 8: TrackerDrawDemo - používanie Trackera na kreslenie

Tento príklad demonštruje, ako môžeme nakresliť nový polygón tzv. „voľnou rukou“ (Freehand).

- Vygenerujeme projekt *TrackerDrawDemo* v kategórii Test a to použitím Plugin Wizardu, kde ako vzor slúži predošlý príklad *OverlayPolyDemo* z kategórie Test.

- Pri modifikácii zdrojového kódu budeme modifikovať iba jedinú funkciu – *OnBnClickedDotest* v súbore *Dlg.cpp* tak, ako sú zobrazené v podfarbenom texte

Dlg.cpp

```
void CDlg::OnBnClickedDotest()
{
    CView* pView = GetView();
    CClientDC dc(pView);
    pView->OnPrepareDC(&dc);

    CTracker::CMode mode;
    mode.m_bRubber = false;
    mode.m_nWidth = 1;
    mode.m_col = RGB(255, 0, 0);
    CTrackerPoly tracker(pView, &mode);

    tracker.TrackFreehand();

    tracker.m_polygon.DPtoLP(&dc);
    m_OverlayPoly.Add(tracker.m_polygon);
    pView->Invalidate();
}
```

Po nakreslení polygónu trackerom sa pomocou funkcie *Invalidate()* zavolá funkcia *AddPluginDraw()*, ktorá vyfarbí vnútro nakresleného polygónu na bielo. Ak by sme chceli zadať polygón kliknutím na jednotlivé vrcholy, namiesto funkcie *TrackFreehand* by sme použili *TrackRubberband*.

Sada tried trackerov odvodených od *CTracker* podporuje okrem kreslenia mnohoúhelníkov aj kreslenie bodu, úsečky, uhla a obdĺžnika. Ich použitie je jednoduché, stačí konštruovať príslušnú triedu odvodenú od *CTracker* (v poradí: *CTrackerPoint*, *CTrackerLine*, *CTrackerAngle*, resp. *CTrackerRect*) a zavolať ich členskú funkciu *TrackRubberband*. Tak napr. trieda *CTrackerAngle* s jedinou verejnou funkciou *TrackRubberBand*, slúži na kreslenie uhlov, kde parameter *pPoint* je smerník na prvý bod. Výsledný uhol je uložený v členskej premennej *m_polygon*.

CTrackerAngle

```
class AFX_EXT_CLASS CTrackerAngle : public CTracker
{
public:
    BOOL TrackRubberBand(CPoint* pPoint);
    CTrackerAngle(CView* pView, CTracker::CMode* mode = NULL);
    CTrackerAngle(CView* pView, const CPolygon& poly, CTracker::CMode*
        mode = NULL);
    virtual ~CTrackerAngle();
}
```

Analogicky sú deklarované aj ostatné triedy odvodené od *CTracker*.

7.3 Príklad 9: TrackerEditDemo - editovanie objektu pomocou CTracker:

Ako už bolo v úvode kapitoly spomínané, trieda *CTracker* je natoľko univerzálna, že môže slúžiť nielen na kreslenie polygónov, ale aj na editovanie už nakreslených polygónov. Funkciu na ktorú budeme tracker používať definujeme pomocou konštruktora – ak je parametrom konštruktora *CPolygon*, znamená to, že sme tracker skonštruovali za účelom editovania.

Nasledovný príklad demonštruje odstránenie jedného vrcholu objektu *m_polygon* kliknutím pravého tlačítka myši na vrchol, ktorý chceme odstrániť. Potrebujeme teda zaregistrovať súradnicu kliknutého bodu a zavolať obslužnú funkciu (v danom prípade obsluha kliknutia pravým tlačítkom myši). Je zrejmé, že tu už nevystačíme so smerníkom na okno pohľadu získaným pomocou *GetView()*. Nadišiel čas využiť doteraz „odpočívajúcu“ triedu *CPluginView*, ktorá je „reprezentantom“ okna obrazu v module a sú do nej posielené všetky správy, ktoré by za normálnych okolností (t.j. keď nie je otvorený žiadny modul), obslúžil program Ellipse.

- Vygenerujeme projekt *TrackerEditDemo* v kategórii *Test*, kde ako vzor slúži predošlý príklad *TrackerDrawDemo* z kategórie *Test*.
- Modifikujeme zdrojový kód funkcie *OnBnClickedDotest()* tak, že tentoraz tracker volá miest funkcie *TrackFreehand* funkciu *TrackRubberBand*. Tým dosiahneme zmenu štýlu kreslenia tak, že každý vrchol polygónu je definovaný kliknutím, kreslenie sa ukončí dvojitém kliknutím. Farba a hrúbka čiary trackera je definovaná rovnako ako v predošlom príklade.
- Modifikujeme funkciu *AddPluginDraw* tak, aby nevyfarbovala vnútro polygónu, a aby vykreslila polygón rovnakým štýlom, ako to robí tracker (farba, hrúbka čiary), pričom sa vykreslí aj úsečka spájajúca prvý a posledný bod.
- Pri modifikácii zdrojového kódu pridáme do triedy *CPluginView* obsluhu kliknutia pravého tlačidla myši. Vnútri funkcie *OnRButtonDown* definujeme tracker pomocou konštruktora, ktorý ako parameter obsahuje polygón. Tým je definovaná jeho schopnosť editovať spomínaný polygón. Pokiaľ tracker zistí, že kliknutý bod sa nachádza v blízkosti niektorého vrcholu polygónu, zavolá funkciu slúžiacu na vymazanie identifikovaného vrcholu.

Obsluha programu je jednoduchá. Po načítaní obrázku a kliknutí DO TEST sa očakáva nakreslenie polygónu zadaním jednotlivých bodov. Keďže každé použitie funkcie *DoTest* vynuluje *m_OverlayPoly*, zabezpečí to, aby mohol byť vždy nakreslený iba jeden polygón. Kliknutím pravého tlačidla myši do blízkosti niektorého vrcholu tento vrchol odstránime, modifikovaný polygón sa okamžite zobrazí.

Pretože tento príklad obsahuje viacero modifikácií súborov *Dlg.cpp* a *PluginView.cpp*, uvádzame preto ich kompletný výpis, pričom modifikované časti textu sú podfarbené.

Dlg.cpp

```
#include "stdafx.h"
#include "Dlg.h"
#include "resource.h"
#include "PluginView.h"
#include "..\dlg.h"
```

```

IMPLEMENT_DYNAMIC(CDlg, CDialog)
CDlg::CDlg(CWnd* pParent /*=NULL*/) : CPluginDlg(pParent)
{
    m_pPluginView = new CPluginView(this);
}

CDlg::~~CDlg()
{
    m_pPluginView->DestroyWindow();
    delete m_pPluginView;
}

void CDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CDlg, CDialog)
    ON_BN_CLICKED(IDC_DOTEST, OnBnClickedDotest)
END_MESSAGE_MAP()

BOOL CDlg::OnInitDialog(void)
{
    m_pPluginView->Create();
    m_pPluginView->ShowWindow(SW_HIDE);
    return 0;
}

void CDlg::OnBnClickedDotest()
{
    CView* pView = GetView();
    CClientDC dc(pView);
    pView->OnPrepareDC(&dc);

    CTracker::CMode mode;
    mode.m_bRubber = false;
    mode.m_nWidth = 1;
    mode.m_col = RGB(255, 0, 0);
    CTrackerPoly tracker(pView, &mode);
    tracker.TrackRubberBand();
    tracker.m_polygon.DPtoLP(&dc);
    m_OverlayPoly.RemoveAll();
    m_OverlayPoly.Add(tracker.m_polygon);
    pView->Invalidate();
}

void CDlg::AddPluginDraw(LPARAM lParam)
{
    CDC* pDC = (CDC*)lParam;
    if(m_OverlayPoly.GetSize() != 1)
        return;

    if(m_OverlayPoly[0].GetSize() == 0)
        return;

    CPen RedPen(PS_SOLID, 1, RGB(255,0,0));
    CPen* pOldPen = pDC->SelectObject(&RedPen);

    pDC->MoveTo(m_OverlayPoly[0][0]);
    for(int i=0; i<m_OverlayPoly[0].GetSize(); i++)
        pDC->LineTo(m_OverlayPoly[0][i]);
}

```

```
pDC->LineTo(m_OverlayPoly[0][0]);
pDC->SelectObject(pOldPen);
}
```

PluginView.cpp

```
#include "stdafx.h"
#include "PluginView.h"
#include "Dlg.h"
#include "..\pluginview.h"

IMPLEMENT_DYNAMIC(CPluginView, CWnd)
CPluginView::CPluginView(CDlg* pParent):CWnd()
{
    m_pDlg = pParent;
}

CPluginView::~CPluginView()
{
}

BEGIN_MESSAGE_MAP(CPluginView, CWnd)
    ON_WM_RBUTTONDOWN()
END_MESSAGE_MAP()

BOOL CPluginView::Create()
{
    BOOL ret = CWnd::Create(NULL, "", WS_CHILD | WS_VISIBLE, CRect(0, 0,
        1, 1), (CWnd*)m_pDlg, 1234);
    return ret;
}

void CPluginView::OnRButtonDown(UINT nFlags, CPoint point)
{
    CView* pView = m_pDlg->GetView();

    CClientDC dc(pView);
    pView->OnPrepareDC(&dc);
    CPoint lp = point;
    dc.DPtoLP(&lp);

    CTracker tracker(pView, m_pDlg->m_OverlayPoly[0]);
    int nMode = tracker.HitTest(m_pDlg->m_OverlayPoly[0], lp);
    if(nMode == CTracker::hitVertex)
        tracker.RemoveVertex(m_pDlg->m_OverlayPoly[0]);
    pView->Invalidate();

    CWnd::OnRButtonDown(nFlags, point);
}
```

Drobnými modifikáciami funkcie `OnRButtonDown` môžeme výrazne modifikovať vykonávanú činnosť.

Pokiaľ by sme riadok : `tracker.RemoveVertex(&m_polygon);`
nahradili riadkom : `tracker.InsertVertex(m_polygon, lp);`

znamenaloby to zaradenie nového vrcholu do polygónu, pričom pozícia by bola daná kliknutým miestom.

Naproti tomu riadok: `tracker.MovePolygon(&m_polygon, lp);`
znamená posun polygónu bez použitia funkcie *HitTest*.

8 Vyššia forma komunikácie medzi Ellipse a modulom pomocou Plugin Interface.

V kapitole 3 boli popísané najjednoduchšie formy komunikácie medzi Ellipse a plug-in modulom. Ellipse dokáže pomocou štyroch základných komunikačných funkcií otvoriť a zatvoriť modul, odovzdať správy napr. od myši na spracovanie modulu, ktorý je práve otvorený. Poskytne tiež modulu smerník na kresliacu plochu okna obsahujúceho obrázok a tým umožní tvorcovi modulu, aby pomocou funkcie AddPluginDraw doprogramoval svoju obsluhu kreslenia.

Bohužiaľ, takáto forma komunikácie má veľmi obmedzené možnosti. Ellipse totiž obsahuje viaceré premenné, ktorých využitie v module by bolo veľmi užitočné. Každému je ale asi zrejmé, že sprístupnenie rôznych premenných, ktoré sa vyskytujú vo viacerých rôznych triedach Ellipse by nebolo účelné, ba dokonca by to bolo veľmi nebezpečné pre samotnú činnosť Ellipse. Preto sa vytvorila zvláštna trieda (CPluginInterface), ktorá združuje všetky funkcie, pomocou ktorých môže získať modul prístup k premenným Ellipse.

Väčšina funkcií sprostredkujúca komunikáciu s Ellipse je typu „Get“, teda vráti kópiu požadovanej premennej, čím znemožní jej prepísanie. Niektoré funkcie „Get“ vrátia smerník na vnútorné premenné Ellipse, ich spracovanie je tak plne na zodpovednosť užívateľa. Okrem toho existuje aj viacero funkcií typu „Set“, ktoré slúžia priamo na zápis určitých hodnôt do premenných hostiteľského programu Ellipse. V ďalšom texte budeme často používať skratku PIN vo význame Plugin Interface. Jednotlivé funkcie PIN sú:

8.1 Zoznam funkcií Plugin Interface (PIN).

CPluginInterface - funkcie na čítanie z PIN

// General	
GetPINVersion	číslo verzie aktuálneho PIN
GetPINSize	veľkosť PIN (kvôli kontrole compatibility verzií)
// View	
GetViewZoomVal	aktuálny koeficient ZOOMu (=1.0 pre zobrazenie 1:1)
GetViewTrackPoly	smerník na posledný polygón vytvorený trackerom
GetViewCanAddPoly	stav premennej povoľujúcej trvalé zobrazovanie Trackerom nakresleného polygónu
GetViewCurClass	číslo prednastavenej triedy (Combo box na lište)
GetViewDrawingTools	pole určujúce stav ikon kresliacich prostriedkov
GetViewDrawnPolygons	stav premennej povoľujúcej zobrazenie OverlayPoly
// Density Calibration	
GetCalDensLUT	smerník na kalibračnú tabuľku optickej hustoty LUT
GetCalDensN	veľkosť LUT (v Ellipse je štandardne 256)
GetCalDensCalibrated	stav premennej, či bola optická hustota kalibrovaná
// Distance Calibration	
GetCalDistConstXY	vráti kalibračnú konštantu v smere osí X a Y
GetCalDistConstZ	vráti kalibračnú konštantu v smere osí Z
GetCalDistUnits	vráti popis kalibrovaných jednotiek (napr. „mm“)
GetCalDistCalibratedXY	informácia či bol obraz kalibrovaný v smere XY
GetCalDistCalibratedZ	informácia či bol obraz kalibrovaný v smere Z
// Stack slider	
GetSldrInvert	mód zobrazovania slidera (invertovaný alebo nie)
GetSldrEnableInvert	stav premennej povoľujúcej invertovanie slidera
GetSldrMinSel	minimálna hodnota SELECTION slidera
GetSldrMaxSel	maximálna hodnota SELECTION slidera
GetSldrPos	aktuálna pozícia slidera
GetSldrLastPos	stav premennej na zapamätanie pozície (zarážka)
GetSldrLabMin	reťazec znakov popisujúcich minimálnu SELECTION
GetSldrLabMax	reťazec znakov popisujúcich maximálnu SELECTION
// Stack of images	
GetStackImages	smerník na začiatok poľa obrazov (CImg**)
GetStackCurImage	smerník na práve zobrazovaný obraz v stacku (CImg*)
GetStackOverlayPolys	smerník na začiatok poľa overlayov (COverlayPoly**)
GetStackCurOverlay	smerník na práve zobrazovaný overlay (COverlayPoly*)
GetStackCaptions	smerník na názvy obrazov v stacku (CString*)
GetStackXYSize	veľkosť XY jednotlivých obrazov v stacku (CSize)
GetStackZSize	veľkosť v smere Z (=počet obrazov v stacku)
GetStackModifiedImg	stav premennej udávajúcej, či boli obrazy menené
// user data	
GetUserData	smerník na blok vyhradený pre dáta, ktoré majú ostať aj po uzavretí modulu (void*)
// Strings;	
GetPathModule	reťazec udávajúci cestu k práve otvorenému modulu
GetPathImages	reťazec udávajúci cestu k práve otvorenému obrazu
GetPathEllipse	reťazec udávajúci cestu k „Ellipse.exe“
// ResultTable	
GetResultTable	smerník na tabuľku výsledkov CResultTable*

CPluginInterface - funkcie pre zápis prostredníctvom PIN

Funkcia	Návratová hodnota funkcie
// View	
SetViewCanAddPoly	nastavenie premennej povoľujúcej prídanie nakresleného polygónu do OverlayPoly
SetViewDrawingTools	nastavenie počtu ikon kresliacich prostriedkov
SetViewDrawnPolygons	nastavenie premennej na zobrazenie OverlayPoly
// Density Calibration	
SetCalDensCalibrated	nastavenie premennej, o kalibrácii optickej hustoty
// Distance Calibration	
SetCalDistConstXY	nastaví kalibračnú konštantu v smere osí X a Y
SetCalDistConstZ	nastaví kalibračnú konštantu v smere osí Z
SetCalDistUnits	definuje popis kalibrovaných jednotiek (napr. „mm“)
SetCalDistCalibratedXY	nastaví informáciu o kalibrovaní v smere XY
SetCalDistCalibratedZ	nastaví informáciu o kalibrovaní v smere Z
// Stack slider	
SetSlidrInvert	nastaví mód zobrazovania slidera (Invert/NonInvert)
SetSlidrEnableInvert	povolí, alebo zakáže invertovanie slidera
SetSlidrMinSel	nastaví minimálnu hodnotu SELECTION slidera
SetSlidrMaxSel	nastaví maximálnu hodnotu SELECTION slidera
SetSlidrPos	nastaví novú pozíciu slidera
SetSlidrLastPos	nastaví hodnotu „zarážky“ slidera
SetSlidrLabMin	popíše minimálnu hodnotu SELECTION slidera
SetSlidrLabMax	popíše maximálnu hodnotu SELECTION slidera
// Stack of images	
SetStackCaptions	umožní zmeniť názov učitého obrázku v stacku
SetStackModifiedImg	nastaví premennú informujúcu o zmene obrazov
// User data	
SetUserData	uloží smerník na užívateľské data
SetPIN	prepíše naraz všetky data z PIN do dokumentu

CPluginInterface - špeciálne funkcie

```
CPluginInterface(CDocument* pDoc);           // spojenie s dokumentom
CPluginInterface(CImg** pDib, int n=1, COverlayPoly** pOvr=NULL, CString*
    strCaption=NULL);           // spojenie s obrazom modulu
~CPluginInterface;           // deštruktor
AddNewImages           // pridá obrázky ku akt. dokumentu
```

8.2 Využitie PluginInterface.

PluginInterface predstavuje užívateľské rozhranie medzi modulom a dokumentom. Existujú dva základné typy *PluginInterface* líšiace sa konštruktorom a to podľa ich účelu:

- *PluginInterface* vytvorený pomocou existujúceho dokumentu. Použijeme konštruktor:

```
CPluginInterface PIN(m_pDoc);
```

Používa sa vtedy, keď potrebujeme v nejakej funkcii modulu sprístupniť momentálny stav premenných v tom dokumente *Ellipse*, ktorý bol aktívny v okamihu štartu modulu (vstupný dokument). Konštrukcia PIN sa uskutočňuje zvyčajne na začiatku funkcie čím poskytuje práve vykonávanej funkcii najaktuálnejšie dáta dokumentu. Podobným spôsobom sa používa aj známa funkcia *GetDocument()*, ktorá sa tiež zvyčajne vyskytuje na začiatku funkcií programov vo Windows, ktoré používajú architektúru *Doc/View*. Hypoteticky by sme mohli skonštruovať PIN iba raz pri štarte modulu a uložiť objekt napr. pod názvom *m_PIN*, bolo by to však šetrenie na nesprávnom mieste, keďže *m_PIN* by neodrážal zmeny dokumentu po štarte modulu (napr. zmenu polohy *StackSlidera* užívateľom). Naproti tomu, použite archivovaného smerníka na dokument *m_pDoc*, ktorý sme uložili pri štarte modulu chybou nie je. Ako už bolo spomínané, celá činnosť modulu sa má týkať vstupného dokumentu (obrazu). Ak teda otvoríme modul a potom kliknutím označíme iný obraz, modul bude stále spracovávať údaje z pôvodného dokumentu.

- *PluginInterface* vytvorený pomocou obrazu za účelom tvorby nového dokumentu. Používa sa zvyčajne vtedy, keď potrebujeme v module vytvoriť vlastný obraz a potrebujeme v *Ellipse* vytvoriť nový dokument, ktorý by tento obraz reprezentoval. Použijeme konštruktor, ktorý ako parameter používa existujúci obraz, resp. stack obrazov (vtedy udávané aj počet obrazov v stacku N). Po konštrukcii PIN môže nasledovať volanie funkcie, ktorá na základe PIN vytvorí v *Ellipse* nový dokument.

```
CPluginInterface PIN(&pImg, N);
```

Po vytvorení dokumentu v *Ellipse* splnil PIN svoju úlohu a nedoporučuje sa používať ho v rámci aktuálnej funkcie na žiadne ďalšie úlohy. Pretože pri vytváraní dokumentu si *Ellipse* vytvorila vlastnú kópiu stacku, nesmieme tiež zabudnúť nami vytvorený stack v module vymazať.

8.2.1 Príklad 10: PINDocDemo - vytvorenie PIN pomocou existujúceho dokumentu

Na demonštráciu tohto typu použitia PIN využijeme príklad 5.3, ktorý demonštroval triedu *CImg*, avšak ešte nebol známy spôsob ako získať smerník na aktuálny dokument a preto sa načítaval zo súboru.

Príklad v tejto podkapitole vykonáva rovnakú činnosť ako spomínaný príklad 5.3, avšak využijeme smerník na vstupný dokument ktorý poskytuje PIN. To znamená vstupný obraz modul skonvertuje tak, že v spodnej polovici sa vytvorí zrkadlový obraz riadkov z vrchnej polovice.

- Vygenerujeme projekt *PINDocDemo* v kategórii Test a to použitím Plugin Wizardu, kde ako vzor slúži príklad *CImgDemo* z kategórie Test.

- Pri modifikácii zdrojového kódu budeme modifikovať iba jedinú funkciu – *OnBnClickedDotest* v súbore *Dlg.cpp* tak, ako sú zobrazené v podfarbenom texte

```
void CDlg::OnBnClickedDotest()
{
    CPluginInterface PIN(m_pDoc);
    CImg* pImg = PIN.GetStackImage();

    int W = pImg->GetWidth();
    int H = pImg->GetHeight();
    int P = pImg->GetPitch();

    for(int i=0; i<H/2; i++){
        BYTE* pSrcRow = (BYTE*)pImg->GetBits() + i*P;
        BYTE* pDestRow = (BYTE*)pImg->GetBits() + (H-1-i)*P;
        CopyMemory(pDestRow, pSrcRow, abs(P));
    }
    GetView()->Invalidate();
}
```

Rozdiel oproti príkladu z kapitoly 5.3 je iba v tom, že obraz *Img* nezískame načítaním zo súboru ale priamo získame smerník *pImg* na vstupný obraz modulu (podfarbený text). Miesto zápisu do výstupného súboru, prebieha konverzia priamo vo vstupnom obrázku, čo sa aj po stlačení tlačítka DO TEST vykreslí.

8.2.2 Príklad 11: PINImgDemo - vytvorenie PIN pomocou existujúceho obrazu a vytvorenie nového dokumentu

Tento príklad demonštruje druhú alternatívu, keď sa pomocou PIN vytvorí spojenie so stackom, ktorý bol vytvorený modulom a ten potom umožní Ellipse vytvoriť nový dokument. Modul má za úlohu vytvoriť stack pozostávajúci z piatich šedotónových obrazov rozmerov 256x256 pixelov, pričom každý ďalší obraz v stacku má byť homogénne vyplnený šedou farbou. Stupeň šedi na prvej hladine je 100, na každej ďalšej hladine má byť o 20 jednotiek svetlejší, než na predchádzajúcej. Takýto stack obrazov má Ellipse zaevidovať ako nový dokument, čiže vytvorí sa pre stack nové okno vedľa vstupného dokumentu.

- Vygenerujeme projekt *PINImgDemo* v kategórii *Test* a to použitím Plugin Wizardu, kde ako vzor slúži príklad *PINImgDemo* z kategórie *Test*.
- Pri modifikácii zdrojového kódu budeme modifikovať iba jedinú funkciu – *OnBnClickedDotest* v súbore *Dlg.cpp* tak, ako sú zobrazené v nasledujúcom podfarbenom texte.

Dlg.cpp

```
void CDlg::OnBnClickedDotest()
{
    int GrayLevel = 100;
    const int N = 5;
    CImg* pImg = new CImg[N];
    for(int i=0; i<N; i++){
        pImg[i].Create(256, 256, 8);
        pImg[i].SetGrayColorTable();
        int H = pImg[i].GetHeight();
        int P = pImg[i].GetPitch();
        GrayLevel = GrayLevel + 20;
        memset(pImg[i].GetBits(), GrayLevel, P*H);
    }
}
```

```
}  
  
CPluginInterface PIN(&pImg,N);  
delete[] pImg;  
}
```

Funkcia pracuje nasledovne: Najprv vytvoríme pole piatich smerníkov na budúce obrazy, potom v slučke každému smerníku priradíme konkrétny obsah – obraz 256x256x8 bits, čím vytvoríme v pamäti potrebné miesto pre pixely a (pretože sa jedná o 8-bitový indexovaný obraz), tak aj pre tabuľku farieb. Tabuľku farieb naplníme pomocou funkcie SetGrayColorTable() a na miesto určené funkciou GetBits() kde začína blok pixelov skopírujeme konštantnú hodnotu danú premennou m_GrayLevel.

Potom k vytvorenému stacku skonštruujeme Plugin Interface (PIN), pričom ako parameter sa zadáva adresa kde sa nachádza smerník na prvý obraz a tiež počet obrazov. Tento PIN pri svojej konštrukcii súčasne ktorá požiada Ellipse o zaevidovanie nového dokumentu. Pretože pri vytváraní dokumentu si Ellipse vytvorila vlastnú kópiu stacku, nesmieme zabudnúť nami vytvorený stack v module vymazať.

9 Zobrazovanie číselných výsledkov

V časti 2.8 bolo uvedené, že výstupom modulov pre analýzu obrazu je vektor číselných údajov, ktoré charakterizujú výsledok analýzy (napr. plocha a obvod jednotlivých detekovaných objektov) a ktoré je treba prezentovať v prehľadnej podobe. Najbežnejšou formou prezentácie tohto typu je tabuľka výsledkov zobrazená vo zvláštnom okne, tzv. „Okne výsledkov“ (*ResultWindow*). Programátorskú podporu tabuľky výsledkov zabezpečuje trieda *CResultTable*. Tieto pojmy sú spresnené v ďalších podkapitolách.

9.1 Okno výsledkov (*Result Window*) a tabuľka výsledkov

Okno výsledkov je zvláštne okno, ktoré sa dá otvoriť iba v *Ellipse*. Otvorením okna výsledkov cez menu *File|New|ResultWindow* vznikne nový *Pohľad* na vstupný dokument. Ako je už užívateľom *Ellipse* známe, tento pohľad automaticky reaguje na akékoľvek zmeny v dokumente, čiže napr. ak nakreslíme nový objekt do obrazu, automaticky sa do okna výsledkov pridá nový riadok s vypočítanými parametrami nakresleného objektu. Táto interaktívna spolupráca obrazu a tabuľky výsledkov bola naprogramovaná v prostredí *Ellipse* a preto je nie je možné priamo využívať vo vlastných moduloch. Ako bude ukázané v tejto kapitole, aj z modulu dokážeme využívať tabuľku, zapisovať do nej výsledky vypočítane modulom a aj naprogramovať podobné interaktívne správanie ako v *Ellipse*.

Tabuľka výsledkov slúži na prehľadné a jednoduché zobrazovanie číselných údajov vypočítaných modulom. Najjednoduchšie je využiť tabuľku, ktorá už je v *Ellipse*. Aj keď táto plní v *Ellipse* svoju špecifickú úlohu, v okamihu otvorenia modulu je k dispozícii tomuto modulu. Pozostáva z hlavičky (head), formátovacieho riadku (form), masky (mask) a poľa s číselnými údajmi (params).

Vzhľadom na čo najjednoduchšiu manipuláciu s tabuľkou údajové bunky akceptujú iba čísla typu *double*. Pokiaľ chceme tabuľkou zobraziť celé číslo, musíme ho pred zadaním do tabuľky konvertovať na *double* a formátovacím poľom nastaviť jeho zobrazenie na nulový počet desatinných miest. Zvláštnu úlohu ma pole *Mask*, ktoré povoľuje alebo zabraňuje zobrazeniu príslušného stĺpca po jeho naplnení (rozmer tabuľky sa automaticky prispôsobí vynechanému stĺpcu).

9.2 *CResultTable* - programátorská podpora tabuľky výsledkov

Trieda *CResultTable* je dispozíciou na vytváranie tabuľky zobrazovaných výsledkov. Úlohou tvorcu modulu je naplniť ju vhodnými dátami, o samotné zobrazovanie sa už automaticky postará *CResultWindow*. K dispozícii sú nasledovné funkcie:

CResultTable	konštruktor. V module sa nepoužíva, keďže <i>Ellipse</i> poskytne modulu smerník na svoju tabuľku <i>pResultTable</i>
Create	vytvorí tabuľku na základe počtu stĺpcov, head a form
Add	pridá na koniec tabuľky nový riadok určený poľom dát
Replace	nahradí riadok tabuľky <i>Idx</i> novým riadkom <i>row</i>
SetMask	vytvorí masku určujúcu zobrazovanie jednotlivých stĺpcov
GetSize	Vráti rozmery tabuľky (počet riadkov a stĺpcov)
GetCompressedTabSize	Rozmery maskovanej tabuľky (viditeľné riadky a stĺpce)

Empty	Zistí, či je tabuľka prázdna (počet dátových riadkov = 0)
GetRow	vráti smerník na pole čísel v požadovanom riadku
GetHeadItem	vráti reťazec popisujúci požadovaný stĺpec hlavičky
GetString	vráti reťazec znakov obsahujúci všetky čísla (môžu byť oddelené tabelátorom) medzi dvoma pozíciami v tabuľke
RemoveAll	vynuluje tabuľku (bude mať nulový počet riadkov)
RemoveAt	vymaže daný riadok tabuľky z pamäte

S tabuľkou sa manipuluje v zásade po riadkoch, ktoré sú reprezentované jednorozmerným poľom určitej dĺžky (totožnej s počtom stĺpcov tabuľky). **Head** je pole reťazcov predstavujúcich nápisy v prvom riadku tabuľky. **Form** je tak isto pole formátovacích reťazcov (podobných ako sú napr. vo funkcii *printf* jazyka C). Ich funkciou je definovať formát zobrazovaných premenných. Keďže jediným povoleným typom premennej v tabuľke je double, jediným spôsobom ako zobrazit' celočíselnú premennú je zobrazit' ju s nulovým počtom desatinných miest. Pomocou spomínaných dvoch poľí použitých ako parameter funkcie **Create** vieme definovať tabuľku, ku ktorej môžeme pridávať (**Add**) nové riadky, alebo modifikovať (**Replace**) riadok na určitom mieste tabuľky.

Mask je pole celých čísel, ktoré môže zabrániť alebo povoliť zobrazovanie určitých stĺpcov tabuľky, bez toho aby sa musela predefinovať samotná tabuľka.

9.3 Príklad 12: *ResultTableDemo* – používanie tabuľky výsledkov.

Na demonštrovanie základných možností práce s tabuľkou výsledkov uvidíme nasledovný príklad. Opäť budeme vychádzať z jedného z predošlých príkladov (*TrackerEditDemo*), pri ktorom sme po stlačení DO TEST mohli nakresliť do vstupného obrázku mnohoholník a kliknutím pomocou pravého tlačítka myši do blízkosti niektorého vrcholu sme ho mohli odstrániť. Spomínaný príklad rozšírime o možnosť zobrazovať v otvorenom okne výsledkov aktuálne údaje o ploche a obvode nakresleného útvaru. Budeme to robiť tak, že po každom použití DO TEST a nakreslení polygónu sa do tabuľky pridá nový riadok. Naopak, keď použijeme pravé tlačidlo myši na vymazanie vrcholu posledne nakresleného polygónu, nebude sa riadok pridávať, ale bude sa modifikovať posledný riadok tabuľky tak, aby údaje zodpovedali zmenenému tvaru polygónu.

- Vygenerujeme projekt *ResultTableDemo* v kategórii *Test* a to použitím Plugin Wizardu, kde ako vzor slúži príklad *TrackerEditDemo* z kategórie *Test*.
- Pri modifikácii zdrojového kódu budeme modifikovať funkcie ku ktorým bol pridaný podfarbený text.

Dlg.cpp

```

BOOL CDlg::OnInitDialog(void)
{
    m_pPluginView->Create();
    m_pPluginView->ShowWindow(SW_HIDE);

    CPluginInterface PIN(m_pDoc);
    const char head[][15] = {"No", "Area", "Perimeter"};
    const char form[][10] = {"10.0", "10.2", "10.2"};
    int mask[3] = {1,1,1};
    PIN.GetResultTable()->Create(3, head, form);

```

```

PIN.GetResultTable()->SetMask(mask);
CView* pResultView = GetView(viewResults);
if(pResultView)
    pResultView->Invalidate();

    return 0;
}

void CDlg::OnBnClickedDotest()
{
    CView* pView = GetView();
    CClientDC dc(pView);
    pView->OnPrepareDC(&dc);

    CTracker::CMode mode;
    mode.m_bRubber = false;
    mode.m_nWidth = 1;
    mode.m_col = RGB(255, 0, 0);
    CTrackerPoly tracker(pView, &mode);
    tracker.TrackRubberBand();
    tracker.m_polygon.DPtoLP(&dc);
    m_OverlayPoly.RemoveAll();
    m_OverlayPoly.Add(tracker.m_polygon);
    pView->Invalidate();
    m_pPluginView->UpdateResultWindow(true);
}

```

PluginView.h

```

#pragma once
#include "Dlg.h"

class CDlg;
class CPluginView : public CWnd
{
    DECLARE_DYNAMIC(CPluginView)
public:
    CPluginView(CDlg* pParent);
    virtual ~CPluginView();

    BOOL Create();
    CDlg* m_pDlg;
protected:
    DECLARE_MESSAGE_MAP()
public:
    afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
    void UpdateResultWindow(bool bAddMode);
};

```

PluginView.cpp

```

.
.
.
void CPluginView::OnRButtonDown(UINT nFlags, CPoint point)
{
    CView* pView = m_pDlg->GetView();

```

```

CClientDC dc(pView);
pView->OnPrepareDC(&dc);
CPoint lp = point;
dc.DPtoLP(&lp);

CTracker tracker(pView, m_pDlg->m_OverlayPoly[0]);
int nMode = tracker.HitTest(m_pDlg->m_OverlayPoly[0], lp);
if(nMode == CTracker::hitVertex)
    tracker.RemoveVertex(m_pDlg->m_OverlayPoly[0]);
pView->Invalidate();
UpdateResultWindow(false);
CWnd::OnRButtonDown(nFlags, point);
}

void CPluginView::UpdateResultWindow(bool bAddMode)
{
    double Params[3];
    CPolygon poly = m_pDlg->m_OverlayPoly[0];
    Params[0] = 1;
    Params[1] = (double)poly.Area();
    Params[2] = (double)poly.Length();

    CPluginInterface PIN(m_pDlg->m_pDoc);
    CResultTable* pTable = PIN.GetResultTable();
    if(bAddMode)
        pTable->Add(Params);
    else{
        int LastRow = pTable->GetSize().cy-1;
        pTable->Replace(Params, LastRow);
    }

    m_pDlg->GetView(viewResults)->Invalidate();
}

```

Tabuľku definujeme vo funkcii *OnInitDialog* v súbore *Dlg.cpp* a to pomocou polí reťazcov *Head* a *Form*, ktoré sa použijú ako parametre funkcie *Create*. Najdôležitejšou zmenou je pridanie funkcie *UpdateResultWindow* do triedy *CPluginView*. Táto funkcia definuje pole troch parametrov, z ktorých nultý prvok a je vždy rovný 1, a preto je zobrazovaný vždy celočíselne. Prvý a druhý prvok poľa predstavujú aktuálnu plochu a obvod posledne nakresleného polygónu (uloženého ako jediný (nultý) polygón v poli *OverlayPoly*). Parameter funkcie určuje, či sa bude to tabuľky výsledkov pridávať vypočítaný riadok (po nakreslení nového), alebo sa bude modifikovať posledný riadok (po vymazaní vrcholu posledne nakresleného polygónu).

10 Literatúra

- [1] Kruglinski, D.J. - Shepherd, G. - Wingo, S.: Programujeme v Microsoft Visual C++, Computer Press, Praha, 2000.

11 Knižnica *elGraph*

Táto knižnica obsahuje väčšinu tried ktoré sme doteraz spomínali ktoré značne zjednodušujú písanie plug-in modulov pre Ellipse. Triedy obsiahnuté v tejto knižnici sú programátorom plug-in modulov priamo prístupné, nakoľko knižnica je štandardne pripojená ku všetkým projektom, ktoré sú generované Plugin Wizardom. Štúdiom hlavičkových súborov jednotlivých tried je možné pomerne presne pochopiť funkciu jednotlivých tried. To spolu s touto publikáciou a zdrojovými kódmi viacerých modulov, ktoré sú voľne dostupné umožňuje pomerne rýchlo dosiahnuť stupeň znalostí potrebný na vývoj vlastných modulov na načítanie, analýzu alebo spracovanie obrazov.

Triedy obsiahnuté v knižnici *elGraph*

Knižnica obsahuje nasledovné triedy ktoré už boli v predošlej časti podrobne popísané a ktorých znalosť je pre písanie modulov nevyhnutná:

CImg	rozširuje možnosti triedy ATL::CImage
COverlayPoly	tvorí nedeštruktívnu kresbu na obraze pomocou polygonov
CPolygon	útvár definovaný postupnosťou súradníc hraničných bodov
CPluginInterface	umožňuje transfer údajov medzi modulom a dokumentom
CResultTable	slúži na zobrazovanie tabuľky výsledkov
CTracker	obsluha kreslenia útvarov, slúži ako rodičovská trieda pre CTrackerPoint, CTrackerLine, CTrackerAngle, CTrackerRectangle, CTrackerPoly

Okrem toho sú v knižnici viaceré triedy, ktoré nie sú z hľadiska programovania modulov nutné, ale uľahčujú písanie kódu. Ich podrobnejší popis je mimo rámca tejto publikácie, odporúčame preštudovať hlavičkové súbory tried, alebo zdrojový kód príkladov, v ktorých sa dané triedy používajú. Sú to triedy:

CColorButton	dovoľuje vytvoriť tlačitko s farebným povrchom
CDlgInput	pomocný dialóg na zadávanie numerických údajov
CDlgInputString	pomocný dialóg na zadávanie textových údajov
CFit	umožní fitovať sadu bodov (x,y) rôznymi funkciami
CHelper	viacero pomocných funkcií (prevody string.číslo apod.)
CPluginDlg	slúži ako rodičovská trieda triedy CDlg modulu
CProgressBar	umožní zobrazit "Progress Bar" napr. pri dlhých výpočtoch

Jednotlivé funkcie tried obsiahnutých v knižnici *elGraph* budú v ďalšej časti popísané podrobnejšie v samostatných podkapitolách tejto kapitoly. Výnimkou je trieda *CPluginInterface*, ktorej je venovaná samostatná kapitola.

11.1 Funkcie špecifické pre triedu CImg.

CImg

konštruktory objektu typu CImg

```
1) CImg::CImg() throw();  
2) CImg::CImg(const CImg& img);  
3) CImg::CImg(LPVOID lpMemory);
```

Parametre

img obraz typu CImg, z ktorého sa vytvorí kópia
lpMemory blok pamäte reprezentujúci štruktúru DIB

Poznámka

- 1) Default konštruktor, ktorý skonštruuje objekt CImg, ale na jeho úplné vytvorenie a alokovanie miesta pre dáta musí byť nasledovaný niektorou z funkcií *Create*, *Load*, alebo *Attach*.
- 2) Kopy konštruktor, ktorý vytvorí úplnú kópiu obrazu *img*.
- 3) Konštruktor, ktorý vytvorí objekt typu *CImg* na základe bloku pamäte, ktorý obsahuje štruktúru DIB - hlavička, tabuľka farieb (ak existuje), dátová časť predstavujúca pixely.

Pozri tiež

Prehľad CImg | Členské funkcie | ATL::CImage

operator =

operátor priradenia

```
CImg& operator=(const CImg& img);
```

Návratová hodnota

premenná typu *CImg*, do ktorej sa skopíruje premenná na pravej strane operátora = .

Poznámka

Operátor priradenia je užitočným rozšírením možností triedy *ATL::CImage*, ktorý dovoľuje jednoduchý zápis typu *CImg img2 = img1*;

Pozri tiež

Prehľad CImg | Členské funkcie

IsAlpha

zistí, či obraz je 32 bitový s alfa kanálom

```
BOOL IsAlpha();
```

Návratová hodnota

TRUE ak obraz má alfa kanál, FALSE v opačnom prípade

Poznámka

Pri 32 bitových obrázkoch sa môže štvrtý byte využiť ako tzv. alfa kanál, ktorý môže obsahovať informáciu o transparentnosti daného pixelu.

Pozri tiež

Prehľad CImg | Členské funkcie

GetPixelIndex

u indexovaných obrázkov vráti index pixelu o súradniciach (x,y).

```
BOOL GetPixelIndex(int x, int y, BYTE *idx) const;
```

Návratová hodnota

FALSE ak bod je mimo obrazu alebo obraz nie je indexovaný, TRUE v opačnom prípade

Parametre

x, y súradnice vyšetřovaného bodu
idx adresa premennej, do ktorej sa zistený index uloží

Pozri tiež

Prehľad CImg | Členské funkcie | ATL::CImage::GetPixel()

IsGrayscale

zistí, či obrázok je šedotónový

```
BOOL IsGrayscale() const;
```

Návratová hodnota

TRUE ak obraz je šedotónový, FALSE v opačnom prípade

Poznámka

Šedotónový je taký obraz, ktorý je indexovaný a pre všetky zložky tabuľky farieb platí R=G=B.

Pozri tiež

Prehľad CImg | Členské funkcie | SetGrayColorTable()

GetDimensions

zistí, či obrázok je šedotónový

```
CSize GetDimensions();
```

Návratová hodnota

Rozmery obrázku v tvare premennej *CSize*.

Poznámka

Funkcia vráti naraz oba rozmery obrázku (šírku, výšku), v podobe premennej typu *CSize*

Pozri tiež

Prehľad CImg | Členské funkcie

GetPitch

zistí, či obrázok je šedotónový

```
int GetPitch() const;
```

Návratová hodnota

Šírka obrázku zaokrúhlená na celý násobok 32 bitov (tzv. pitch).

Poznámka

Pitch predstavuje v podstate vzdialenosť medzi susednými riadkami v bytoch. Na rozdiel od funkcie *ATL::CImage::GetPitch()* je prvý riadok vždy zobrazovaný na spodku bitmapy a hodnota pitch je vždy kladná.

Pozri tiež

Prehľad CImg | Členské funkcie | *ATL::CImage::GetPitch()*

GetBits

zistí, či obrázok je šedotónový

```
BYTE* GetBits () const;
```

Návratová hodnota

Ukazovateľ na prvý byte poľa pixelov.

Poznámka

Na rozdiel od funkcie *ATL::CImage::GetPitch()* je prvý riadok vždy zobrazovaný na spodku bitmapy a funkcia *GetBits* ukazuje na pozíciu v pamäti tesne za tabuľkou farieb (indexované obrázky), resp. tesne za hlavičkou (neindexované).

Pozri tiež

Prehľad CImg | Členské funkcie | *ATL::CImage::GetBits ()*

SetGrayColorTable

nastaví šedotónovú tabuľku farieb

```
bool SetGrayColorTable();
```

Návratová hodnota

false, ak obrázok nie je indexovaný, true v opačnom prípade

Poznámka

Šedotónový je taký obraz, ktorý je indexovaný a pre všetky zložky tabuľky farieb platí R=G=B.

Pozri tiež

Prehľad CImg | Členské funkcie | IsGrayScale()

11.2 Funkcie spoločné pre triedy CImg a ATL::CImage.

~CImg(void)

deštruktor

Poznámka: (totožné s ATL::CImage::~~CImage)

Create

kreovanie bitmapy zadaním rozmerov a BPP.

Poznámka: (totožné s ATL::CImage::Create)

GetWidth

vráti šírku obrazu v pixeloch

Poznámka: (totožné s ATL::CImage::GetWidth)

GetHeight

vráti výšku obrazu v pixeloch

Poznámka: (totožné s ATL::CImage::GetHeight)

GetBPP

vráti počet bitov/pixel (BPP zvyčajne 8, 24, 32)

Poznámka: (totožné s ATL::CImage::GetBPP)

IsIndexed

udáva typ bitmapy (Indexovaná alebo True Color)

Poznámka: (totožné s ATL::CImage::IsIndexed)

GetColorTabl

Skopíruje tabuľku farieb do pripraveného bufferu

Poznámka: (totožné s ATL::CImage::GetColorTable)

SetColorTable

Naplní tabuľku farieb bitmapy hodnotami z bufferu

Poznámka: (totožné s ATL::CImage::SetColorTable)

GetPixel

vráti farebnú hodnotu pixelu o súradniciach (x,y)

Poznámka: (totožné s ATL::CImage::GetPixel)

SetPixel

nastaví farbu pixelu majúceho súradnice (x,y). Táto funkcia má niekoľko modifikácií.

Poznámka: (totožné s ATL::CImage::SetPixel)

Load

načíta bitmapu zo súboru *.BMP (pozná aj iné formáty)

Poznámka: (totožné s ATL::CImage::Load)

Save

uloží bitmapu do súboru vo zvolenom formáte

Poznámka: (totožné s ATL::CImage::Save)

Draw

vykreslenie bitmapy do zadanej kresliacej plochy (DC)

Poznámka: (totožné s ATL::CImage::Draw)

GetMaxColorTableEntries

maximálny počet hodnôt v tabuľke farieb

Poznámka: (viď ATL::CImage::GetMaxColorTableEntries)

11.3 Trieda *COverlayPoly*

COverlayPoly

konštruktory

```
1) COverlayPoly();  
2) COverlayPoly(const COverlayPoly& poly);
```

Poznámka

- 1) Default konštruktor – vytvorí prázdny overlay, do ktorého môžeme pridávať polygóny
- 2) Copy konštruktor – vytvorí nový objekt triedy *COverlayPoly* ako kópiu parametra.

Pozri tiež

Prehľad *COverlayPoly* | Členské funkcie |

Add

umožní pridať do *OverlayPoly* nový polygon

```
int Add(CPolygon& polygon, bool AvoidSameLabels=false);
```

Návratová hodnota

index pridaného prvku v poli

Parametre

polygon objekt typu *CPolygon*, ktorý pridávame do *COverlayPoly*
AvoidSameLabels ak =true, nedovolí pridať polygón s rovnakou značkou

Poznámka

Ak *AvoidSameLabels* = true a pridávame polygón, ktorého label už v *OverlayPoly* existuje, zmení značku polygónu na najmenšiu možnú, ktorá ešte v poli nie je.

Pozri tiež

Prehľad *COverlayPoly* | Členské funkcie | *CArray::Add*

GetSize

vráti počet polygónov v *OverlayPoly*

```
int GetSize() const;
```

Návratová hodnota

počet prvkov typu *CPolygon* v poli

Pozri tiež

Prehľad *COverlayPoly* | Členské funkcie | *CArray::GetSize()*

SetSize

nastaví veľkosť poľa, alokuje potrebnú pamäť

```
void SetSize(int NewSize, int GrowBy=-1) const;
```

Parametre

NewSize nová veľkosť (počet prvkov) poľa
GrowBy prírastok veľkosti poľa, ak je potrebné pridať prvok

Poznámka

Funkcia sa správa rovnako ako *CArray::SetSize()*

Pozri tiež

Prehľad COverlayPoly | Členské funkcie | CArray::SetSize()

operator =

operátor priradenia

```
COverlayPoly& operator=(COverlayPoly& ovr);
```

Návratová hodnota

objekt typu COverlayPoly

Parametre

ovr objekt typu *COverlayPoly*

Poznámka

umožní vytvoriť nový OverlayPoly zápisom $Ovr2 = Ovr1$

Pozri tiež

Prehľad COverlayPoly | Členské funkcie | CArray::operator=

operator []

operátor prístupu k prvkom poľa

```
CPolygon& operator[](int Idx);
```

Návratová hodnota

objekt typu *CPolygon* reprezentujúci prvok poľa na mieste idx

Parametre

idx poradové číslo objektu typu *CPolygon* v poli *COverlayPoly*

Poznámka

umožní jednoduchý zápis typu *CPolygon NewPoly = Overlay[10]*

Pozri tiež

Prehľad COverlayPoly | Členské funkcie | CArray::operator[]

SetAt

nastaví prvok *COverlayPoly* na miesto dané indexom

```
void SetAt((int idx, const CPolygon& newPoly);
```

Parametre

idx	poradové číslo objektu
newPoly	objekt typu CPolygon, ktorý sa má nastaviť na dané miesto

Poznámka

Funkcia sa správa rovnako ako *CArray::SetAt*

Pozri tiež

Prehľad *COverlayPoly* | Členské funkcie | *CArray::SetAt*

RemoveAt

odstráni prvok poľa *COverlayPoly* na mieste dané indexom

```
void RemoveAt(int idx, int Count=1);
```

Parametre

idx	poradové číslo objektu
Count	počet objektov, ktoré sa majú odstrániť

Poznámka

Funkcia sa správa rovnako ako *CArray::RemoveAt*

Pozri tiež

Prehľad *COverlayPoly* | Členské funkcie | *CArray::RemoveAt*

RemoveAll

odstráni všetky prvky poľa *COverlayPoly*

```
void RemoveAll();
```

Poznámka

Funkcia sa správa rovnako ako *CArray::RemoveAll*

Pozri tiež

Prehľad *COverlayPoly* | Členské funkcie | *CArray::RemoveAll*

InWhichPol

nájde index prvého polygónu s ktorým má vstupný polygón nenulový prienik

```
int InWhichPol(CPolygon& pol);
```

Návratová hodnota

index prvého polygónu s ktorým má vstupný polygón nenulový prienik

Parametre

pol vstupný objekt typu *CPolygon*

Poznámka

ak nenájde prienik, vráti zápornú hodnotu

Pozri tiež

Prehľad COverlayPoly | Členské funkcie

PtInWhichPol

nájde index prvého polygónu v ktorom sa nachádza daný bod

```
int PtInWhichPol(CPoint p);
```

Návratová hodnota

index prvého polygónu v ktorom sa nachádza daný bod

Parametre

p vyšetovaný bod

Poznámka

ak nenájde, vráti zápornú hodnotu

Pozri tiež

Prehľad COverlayPoly | Členské funkcie

LabInWhichPoly

nájde index prvého polygónu v ktorom sa nachádza daný bod

```
int LabInWhichPoly(int nLabel);
```

Návratová hodnota

index prvého polygónu ktorý má daný label

Parametre

int hľadaný label

Poznámka

ak nenájde, vráti zápornú hodnotu

Pozri tiež

Prehľad COverlayPoly | Členské funkcie

11.4 Trieda CPolygon a jej špecifické funkcie

CPolygon

konštruktory triedy

```
1) CPolygon();  
2) CPolygon(const CPolygon& polygon);  
3) CPolygon(const CRect &r, int Class=0, int lab=0);
```

Poznámka

- 1) Default konštruktor – po vytvorení je prázdny, môžeme mu pridávať body
- 2) Copy konštruktor – vytvorí kópiu vstupného polygónu
- 3) Konštruktor, ktorý vytvorí zo vstupného obdĺžnika polygón typu RECT s danou triedou *Class* a Labelom *lab*

Pri vytváraní polygónu je potrebné nastaviť premennú *m_Type* podľa geometrického útvaru, ktorý polygon reprezentuje. Predvolené sú typy: TYPE_NOTHING=0, TYPE_POINT=1, TYPE_LINE=2, TYPE_ANGLE=3, TYPE_MULTILINE=4, TYPE_RECT=5, TYPE_POLY=6, TYPE_SPLINE=7.

Použitím kresliacich prostriedkov sa správny typ polygónu nastaví automaticky. Pri programovaní je správne nastavenie *m_Type* plne v kompetencii autora, ktorý musí ošetriť prípady ako napr. že 5-uholník bude typu TYPE_LINE (úsečka).

SetLabel

nastaví *Label* príslušného polygónu

```
bool SetLabel(int nNewLab);
```

Návratová hodnota

true, ak je *nNewLab* v povolenom rozsahu, inak false

Parametre

nNewLab Label, ktorý chceme nastaviť

Poznámka

povolený rozsah pre hodnotu Label je 16 bitové kladné číslo

Pozri tiež

Prehľad CPolygon | Členské funkcie | GetLabel

SetClass

nastaví triedu *Class* príslušného polygónu

```
bool SetClass(int nNewClass);
```

Návratová hodnota

true, ak je *nNewClass* v povolenom rozsahu, inak false

Parametre

nNewClass číslo triedy, ktorú chceme nastaviť

Poznámka

povolený rozsah pre hodnotu *Class* je 16 bitové kladné číslo.

Pozri tiež

Prehľad COverlayPoly | Členské funkcie | GetClass

GetLabel

zistí *Label* príslušného polygónu

```
int GetLabel();
```

Návratová hodnota

Hodnota *Label* príslušného polygónu

Pozri tiež

Prehľad CPolygon | Členské funkcie | SetLabel

GetClass

Zistí triedu *Class* príslušného polygónu

```
int GetClass();
```

Návratová hodnota

Hodnota triedy *Class* príslušného polygónu

Pozri tiež

Prehľad COverlayPoly | Členské funkcie | SetClass

PtInPolygon

Zistí, či daný bod leží vnútri polygónu

```
BOOL PtInPolygon(CPoint point);
```

Návratová hodnota

TRUE, keď bod leží v vnútri polygónu, inak FALSE

Poznámka

Ak polygón nemá aspoň 3 vrcholy, nevie vytvoriť plochu a vracia vždy FALSE

Pozri tiež

Prehľad COverlayPoly | Členské funkcie

DPToLP

Transformuje polygón zo súradnice zariadenia na logické súradnice

```
void DPToLP(CDC* pDC);
```

Parametre

pDC smerník na kresliacu plochu typu Device Contents

Poznámka

Funkcia je ekvivalentom funkcie *DPToLP* aplikovanej na jednotlivé vrcholy polygónu

Pozri tiež

Prehľad COverlayPoly | Členské funkcie | CDC

LPtoDP

Transformuje polygón z logických súradníc na súradnice zariadenia

```
void LPtoDP(CDC* pDC);
```

Parametre

pDC smerník na kresliacu plochu typu Device Contents

Poznámka

Funkcia je ekvivalentom funkcie *LPtoDP* aplikovanej na jednotlivé vrcholy polygónu

Pozri tiež

Prehľad COverlayPoly | Členské funkcie | CDC

Shift

Posunie polygón o súradnice (cx, cy) dané veľkosťou typu CSize

```
void Shift(CSize sz);
```

Parametre

sz premenná typu CSize udávajúca posunutie v smeroch (cx,cy)

Poznámka

Funkcia je ekvivalentom posunutia jednotlivých vrcholov polygónu

Pozri tiež

Prehľad COverlayPoly | Členské funkcie

Area

vypočíta plochu uzavretého polygónu

```
long Area();
```

Návratová hodnota

plocha uzavretého polygónu v pixeloch

Pozri tiež

Prehľad COverlayPoly | Členské funkcie

Length

vypočíta obvod uzavretého polygónu

```
long Length();
```

Návratová hodnota

Obvod uzavretého polygónu v pixeloch

Pozri tiež

Prehľad COverlayPoly | Členské funkcie

GetBoundingRectLP

nastaví obdĺžnik, ktorý kompletne ohraničí daný polygón

```
void GetBoundingRectLP(CRect& r);
```

Parametre

r obdĺžnik, ktorého veľkosť nastaví funkcia tak, aby ohraničil celý polygón

Poznámka

Obdĺžnik je nastavený v logických súradniciach (LP)

Pozri tiež

Prehľad COverlayPoly | Členské funkcie

GetRefreshRectDP

nastaví obdĺžnik, ktorý kompletne ohraničí daný polygón aj s labelom

```
void GetRefreshRectDP(CRect& r, CDC* pDC);
```

Parametre

r obdĺžnik, ktorého veľkosť nastaví funkcia tak, aby ohraničil celý polygón
pDC smerník na kresliacu plochu

Poznámka

Obdĺžnik je nastavený v logických súradniciach (DP) a obsahuje aj textové označenie čísla polygónu (label). Získaný obdĺžnik je možné priamo použiť ako parameter funkcie InvalidateRect(r).

Pozri tiež

Prehľad COverlayPoly | Členské funkcie | InvalidateRect

InvalidatePolygon

prekreslí daný polygón aj s labelom (analogicky ako InvalidateRect)

```
void InvalidatePolygon(CView* pView);
```

Parametre

pView smerník na pohľad okna, v ktorom sa vykresľuje polygón

Poznámka

Má podobný účinok, ako dvojica funkcií *GetRefreshRectDP* + *InvalidateRect*, avšak vykoná to jediným príkazom.

Pozri tiež

Prehľad COverlayPoly | Členské funkcie | GetRefreshRectDP

Overlapped

zistí, či dva polygóny majú spoločný prienik

```
bool Overlapped(CPolygon& poly);
```

Návratová hodnota

true, ak existuje spoločný prienik dvoch polygónov

Parametre

poly polygón zadaný ako parameter

Poznámka

Oba polygóny musia mať minimálne 3 vrcholy (tvoriť plochu)

Pozri tiež

Prehľad COverlayPoly | Členské funkcie

CalculateAttributes

vypočíta viaceré predvolené atribúty polygónu v kalibrovaných hodnotách

```
BOOL CalculateAttributes(CImg* pDib, double dConst, double* LUT, bool  
bCalculate = true);
```

Návratová hodnota

TRUE ak sa podarí výpočet atribútov, FALSE v opačnom prípade.

Parametre

pDib smerník na obrázok v okne, v ktorom je nakreslený polygón

dConst	kalibračná konštanta vzdialenosti v smeroch XY.
LUT	kalibračná tabuľka hodnôt optickej hustoty
bCalculate	premenná ktorá umožní blokovat' počítanie

Poznámka

Polygón je overlayom obrázku CDib. Funkcia spočíta viaceré parametre polygónu na základe pixelov obrázku, ktoré sú vnútri polygónu sú to parametre: LABEL, CLASS, AREA, PERIMETER, ROUNDNESS, FERETDIAMETER, MAJORAXISLENGTH, MAJORAXISANGLE, COMPACTNESS, CENTROIDX, CENTROIDY, GRAYCENTROIDX, GRAYCENTROIDY, INTEGRATEDDENSITY, MINGRAYLEVEL, MEANGRAYLEVEL, MEDIANGRAYLEVEL, MAXGRAYLEVEL, MODEGRAYLEVEL, STANDARDDEVIATION.

Počet atribútov (NUMOFATTRIBUTES) = 20

Výsledok sa uloží do poľa: `double m_pdAttributes[NUMOFATTRIBUTES];`

Pozri tiež

Prehľad COverlayPoly | Členské funkcie

Profile

poskytne hodnotu bodov pozdĺž polygónov včítane interpolovaných.

```
bool Profile(CArray<CLabPoint, CLabPoint&& LabptArr, CDib* pDib);
```

Návratová hodnota

true, ak sa podaril výpočet, inak false (napr. ak polygón je typu Bod).

Parametre

LabptArr pole *CLabPoints* predstavujúce profil pozdĺž polygónu
pDib obraz pod polygónom. Pixely pozdĺž polygónu predstavujú profil.

Poznámka

Profil je hodnota pixelov obrazu *pDib*, ktoré ležia na mieste obrysu polygónu. Polygón je definovaný ako postupnosť vrcholov spojených úsečkami. Na získanie jednotlivých súradníc bodov úsečiek je medzi bodmi polygónu potrebné vykonať interpoláciu a do profilu zaradiť aj hodnoty interpolovaných bodov.

Každý interpolovaný bod má odlišné súradnice a obsahuje aj značku

```
struct CLabPoint {
    CPoint p;
    int val;
};
```

Pozri tiež

Prehľad COverlayPoly | Členské funkcie |

ReducePoints

zredukuje počet bodov polygónu ponechaním každého n-tého bodu.

```
void ReducePoints(int n = 2);
```

Parametre

n faktor redukcie

Poznámka

Funkcia ponechá iba každý n-tý vrchol polygónu, ostatné vymaže. Používa sa hlavne pri polygónoch získaných kreslením typu Freehand, ktorého výsledkom je zbytočne veľké množstvo bodov spomaľujúcich výpočty.

Pozri tiež

Prehľad COverlayPoly | Členské funkcie | ReduceLines

ReduceLines

zredukuje počet bodov polygónu vynechaním redundantných bodov ležiacich na priamke.

```
void ReduceLines();
```

Poznámka

Funkcia vynechá body ležiace na priamke. To znamená, ak z trojice po sebe idúcich bodov leží stredný na spojnici dvoch krajných, je možné ho vynechať. Vynechané body sú nahradzované lineárne interpolovanými či už pri vykresľovaní, alebo napr. pri výpočte profilu.

Pozri tiež

Prehľad COverlayPoly | Členské funkcie | Reduce | Profile

11.5 Trieda CPolygon a jej funkcie zdedené od CArray

operator =

operátor priradenia. Dovoľuje zápis napr. CPolygon poly2 = poly1)

Poznámka: (totožné s CArray::operator =)

operator []

operátor prístupu k prvkom poľa. Dovoľuje zápis napr. CPoint p = poly[10];

Poznámka: (totožné s CArray::operator [])

~CPolygon

deštruktor polygónu

Poznámka: (totožné s CArray::~~CArray)

SetSize

nastaví veľkosť poľa bodov

Poznámka: (totožné s CArray::SetSize)

GetSize

vráti veľkosť poľa, t.j. počet vrcholov polygónu

Poznámka: (totožné s CArray::GetSize)

SetAt

nastaví hodnotu určitého prvku poľa (súradnice vrcholu polygónu)

Poznámka: (totožné s CArray::SetAt)

InsertAt

vloží nový vrchol polygónu p na miesto dané hodnotou idx

Poznámka: (totožné s CArray::InsertAt)

RemoveAt

odstráni vrchol polygónu daný hodnotou idx

Poznámka: (totožné s CArray::RemoveAt)

RemoveAll

odstráni všetky prvky poľa (vrcholy polygónu)

Poznámka: (totožné s CArray::RemoveAll)

Add

pridá nový bod p na koniec poľa

Poznámka: (totožné s CArray::Add)

GetData

vráti smerník CPoint* na prvky poľa

Poznámka: (totožné s CArray::GetData)

11.6 Trieda CTracker

CTracker

konštruktory pre kreslenie polygónu a pre jeho editovanie

```
1) CTracker(CView* pView, CMode* pMode = NULL);  
2) CTracker(CView* pView, const CPolygon& poly, CMode* pMode = NULL);
```

Parametre

pView smerník na pohľad (okno), kde sa kreslený polygón zobrazuje
pMode smerník na štruktúru definujúcu mód operácie (viď poznámka)
poly polygón, ktorý sa chystáme editovať pomocou konštruktora2

Poznámka

- 1) Prvý konštruktor slúži na kreslenie. Užívateľ odštartuje vhodný kresiaci prostriedok a definuje polygón buď kreslením voľnou rukou (Freehand), alebo klikaním jednotlivých vrcholov mnohoúhelníka. Výsledok uloží do verejnej členskej premennej *m_polygon*.
- 2) Druhý konštruktor slúži na editovanie už existujúceho mnohoúhelníka. Môžeme použiť funkcie na pridanie, odstránenie vrcholov alebo presun vrcholov, alebo celého mnohoúhelníka.

Pokiaľ postačí defaultný mód kreslenia trackera (čierna čiara hrúbky 1, kreslenie typu *RubberBand*), nemusíme definovať mód. Pokiaľ chceme vlastný vzhľad, definujeme a naplníme štruktúru *m_mode*, ktorej smerník zadáme ako parameter.

```
struct CMode {  
    BOOL m_bRubber;  
    COLORREF m_col;  
    int m_nWidth;  
} m_mode;
```

Pozri tiež

Prehľad CTracker | Členské funkcie |

~CTracker

deštruktor trackera

```
~CTracker();
```

MovePolygon

funkcia na posuv celého polygónu

```
bool MovePolygon(CPolygon& polygon, CPoint pt);
```

Návratová hodnota

false indikuje chybu pri vykonaní operácií, napr. nepodarilo sa zistiť smerník na mainframe, alebo na statusbar, true znamená správne vykonanie operácie

Parametre

polygon objekt typu CPolygon, ktorý editujeme (posúvame)
pt bod voči ktorému posúvame

Poznámka

Počas činnosti funkcie sú správy presmerované a spracované v tele funkcie. Parametrom je referencia na existujúci polygón, kde po vykonaní posunu je uložený výsledný polygón. Druhý parameter je bod typu *CPoint*, voči ktorému je posun vypočítaný. Počas posunu je polygón prekreslený „gumovou čiarou“.

Pozri tiež

Prehľad CTracker | Členské funkcie |

MoveVertex

funkcia na posuv jedného vrcholu polygónu

```
bool MoveVertex(CPolygon& polygon, CPoint pt);
```

Návratová hodnota

false indikuje chybu pri vykonaní operácií, napr. nepodarilo sa zistiť smerník na mainframe, alebo na statusbar, true znamená správne vykonanie operácie

Parametre

polygon polygón, ktorý editujeme (posúvame jeho vrchol)
pt bod voči ktorému posúvame

Poznámka

Index daného vrcholu je jednoznačne identifikovaný členskou premennou *m_nHit*, ktorý môže byť nastavený buď funkciou *HitTest*, alebo priamo programátorom. Počas činnosti funkcie sú správy presmerované a spracované v tele funkcie. Parametrom je referencia na polygón, kde je po vykonaní posunu uložený aj výsledný polygón. Druhý parameter je bod *CPoint*, voči ktorému je posun vypočítaný. Počas posunu je polygón prekreslený „gumovou čiarou“.

Pozri tiež

Prehľad CTracker | Členské funkcie |

InsertVertex

vloží bod *pt* do polygónu *polygon*. funkcia na vloženie jedného vrcholu do polygónu

```
void InsertVertex(CPolygon& polygon, CPoint pt);
```

Parametre

polygon polygón, ktorý editujeme (posúvame jeho vrchol)
pt bod voči ktorému posúvame

Poznámka

Miesto vloženia je jednoznačne určené parametrom *m_nHit*, ktorý môže byť nastavený buď priamo programátorom, alebo funkciou *HitTest*.

Index daného vrcholu je jednoznačne identifikovaný členskou premennou *m_nHit*, ktorý môže byť nastavený buď funkciou *HitTest*, alebo priamo programátorom. Počas činnosti funkcie sú správy presmerované a spracované v tele funkcie. Parametrom je referencia na polygón, kde je po vykonaní posunu uložený aj výsledný polygón. Druhý parameter je bod *CPoint*, voči ktorému je posun vypočítaný. Počas posunu je polygón prekreslený „gumovou čiarou“.

Pozri tiež

Prehľad CTracker | Členské funkcie |

RemoveVertex

odstráni vrchol polygónu na mieste špecifikovanom pomocou členskej premennej *m_nHit*.

```
void RemoveVertex(CPolygon& polygon);
```

Parametre

polygon polygón, ktorý editujeme

Poznámka

Index daného vrcholu je jednoznačne identifikovaný členskou premennou *m_nHit*, ktorý môže byť nastavený buď funkciou *HitTest*, alebo priamo programátorom. Počas činnosti funkcie sú správy presmerované a spracované v tele funkcie. Parametrom je referencia na polygón, kde je po vykonaní posunu uložený aj výsledný polygón.

Pozri tiež

Prehľad CTracker | Členské funkcie |

HitTest

testuje, kde sa nachádza bod *pt* vzhľadom na *polygon*.

```
int HitTest(CPolygon& polygon, CPoint lp, int nRectSize);
```

Návratová hodnota

určuje, aká časť polygónu bola kliknutá. Môže to byť *hitNothing*, *hitLabel*, *hitVertex*, *hitEdge*, *hitInside*, čiže bod nie je v blízkosti polygónu, bod je blízko k značke/vrcholu/hrane polygónu, resp. sa nachádza vo vnútri polygónu (len u uzavretých polygónov).

Parametre

polygon polygón, ktorý editujeme
pt kliknutý bod ktorého polohu voči polygónu testujeme
nRectSize definuje okolie bodu v pixeloch, v ktorom pracuje *HitTest* (predvolená hodnota je 10).

Poznámka

Ak bod leží blízko k vrcholu resp. hrane, potom členská premenná *m_nHit* bude indexom daného vrcholu resp. koncového bodu hrany. Počas činnosti funkcie sú správy presmerované a spracované v tele funkcie.

Pozri tiež

Prehľad CTracker | Členské funkcie |

TrackRubberBand

zabezpečuje samotné kreslenie polygónov rozličného typu,.

```
BOOL CTracker::TrackRubberBand(CPoint *pPoint);
```

Návratová hodnota

TRUE ak bola operácia úspešne vykonaná, inak FALSE

Parametre

pPoint smerník na kliknutý bod

Poznámka

V triedach odvodených od *CTracker* môže mať táto funkcia rozličnú podobu a aj modifikované názvy (*Track*, *TrackFreehand*) – pozri jednotlivé odvodené triedy

Pozri tiež

Prehľad CTracker | Členské funkcie |

11.7 Triedy odvodené od CTracker

sú určené na kreslenie špecifických objektov typu CPolygon (bod, úsečka, uhol, polygón zadávaním vrcholov, polygón kreslený voľnou rukou). Konštruktory a väčšina funkcií je prevzatá z rodičovskej triedy CTracker a má identický spôsob používania. Odlišnosti sú iba vo funkcii *TrackRubberBand*, ktorá má pri každom prostriedku špecifický spôsob ovládania, ako je to uvedené v poznámke pri každej triede. Z každej funkcie *TrackRubberBand* je možné kedykoľvek vyskočiť stlačením klávesy ESC.

CTrackerPoint

Poznámka

Na zadávanie bodov slúži funkcia *Track*. Po každom kliknutí ľavým tlačítkom myši sa do premennej *m_Polygon* uloží polygón pozostávajúci z jediného bodu.

Pozri tiež

Prehľad CTracker | Členské funkcie | CTracker

CTrackerLine

Poznámka

Pri kreslení čiary sa používa funkcia *TrackRubberband*. Pri stlačení ľavom tlačítke myši sa ťahá “gumová čiara” od kliknutej pozície po aktuálnu polohu kurzora. Druhý bod úsečky je definovaný ako miesto, kde sa uvoľní stlačené pravé tlačítko myši.

Pozri tiež

Prehľad CTracker | Členské funkcie | CTracker

CTrackerAngle

Poznámka

Funkcia *TrackRubberBand* si zapamätá pozíciu pri prvom kliknutí ľavého tlačítka myši a považuje ju za vrchol uhla. Ďalšie dve kliknutia definujú polohu koncových bodov jednotlivých ramien uhla.

Pozri tiež

Prehľad CTracker | Členské funkcie | CTracker

CTrackerPoly

Poznámka

Na kreslenie mnohoúhelníka sú k dispozícii dve funkcie odvodené od *TrackRubberBand*.

- 1) *TrackRubberBand* funguje tak, že každý vrchol mnohoúhelníka je definovaný samostatným kliknutím ľavého tlačítka myši. Pokiaľ sa kurzorom vrátíme do blízkosti východzieho bodu, je polygón automaticky uzavretý. Podobne sa polygón uzavrie pri dvojitém kliknutí ľavým tlačítkom myši.
- 2) *TrackFreehand* je funkcia podporujúca kreslenie voľnou rukou. Po kliknutí prvého bodu môžeme ľavé tlačítko myši pustiť, pričom sa bude zaznamenávať každý bod predstavujúci novú polohu kurzora. Kreslenie sa ukončí návratom kurzora do blízkosti štartovacieho bodu, alebo dvojitém kliknutím

Pozri tiež

Prehľad CTracker | Členské funkcie | CTracker

11.8 CResultTable a zapisovanie výsledkov do okna výsledkov

CResultTable

konštruktor tabuľky výsledkov

```
CResultTable();
```

Poznámka

V module sa nikdy nepoužíva konštruktor nepoužíva, nakoľko tabuľka už je vytvorená v *Ellipse*, stačí zistiť smerník *pResultTable* na ňu (cez PIN) a vytvoriť vlastnú formu tabuľky funkciou *Create*.

Pozri tiež

Prehľad CResultTable | Členské funkcie | Create

Create

vytvorenie vlastnej tabuľky výsledkov

```
1) void Create();  
2) void Create(int width, const char Head[][CellHeadLength], const char  
Format[][FormatStringLength]);
```

Parametre

width	počet stĺpcov tabuľky
Head[]	pole reťazcov znakov popisujúcich jednotlivé stĺpce tabuľky
Format[]	pole reťazcov znakov popisujúcich formát výstupu

Poznámka

- 1) Vytvorí defaultnú tabuľku výsledkov o šírke 5 stĺpcov označených A B C D E, pričom formát zobrazovania je celé číslo.
- 2) Vytvorí vlastnú tabuľku obsahujúcu „width“ stĺpcov. Ich popis je určuje pole reťazcov *Head* a formát pole reťazcov *Format*.

Pozri tiež

Prehľad CResultTable | Členské funkcie |

Add

pridá na koniec tabuľky nový riadok určený poľom dát

```
void Add(double* row);
```

Parametre

row	pole čísel typu double predstavujúcich údaje v jednotlivých stĺpcoch riadku
-----	---

Poznámka

Keďže číselné údaje sú vždy typu `double`, zobrazovaniu celočíselných výsledkov musí predchádzať ich konverzia na `double` a formát zobrazovania musí byť upravený na nula desatinných miest.

Pozri tiež

Prehľad `CResultTable` | Členské funkcie |

Replace

nahradí riadok tabuľky s poradovým číslom *index* novým riadkom *row*.

```
void CResultTable::Replace(double* row, int index);
```

Parametre

`row` pole čísel typu `double` predstavujúcich údaje v jednotlivých stĺpcoch riadku
`index` poradové číslo riadku

Poznámka

Pozri tiež

Prehľad `CResultTable` | Členské funkcie |

SetMask

priradí tabuľke masku, ktorá takto zobrazí iba tie stĺpce, kde je maska nenulová

```
void CResultTable::SetMask(int *pMask);
```

Parametre

`pMask` smerník na pole čísel maskujúcich jednotlivé stĺpce tabuľky

Poznámka

Ak máme napr. masku `pMask = { 1,0,0,1,0 }`, bude sa zobrazovať tabuľka tak, akoby mala iba 2 stĺpce. Pritom všetky údaje v tabuľke ostanú zachované, aj keď sa nezobrazujú.

Pozri tiež

Prehľad `CResultTable` | Členské funkcie |

GetRow

vráti smerník na pole čísel v požadovanom riadku

```
double* GetRow(int row);
```

Návratová hodnota

smerník na pole čísel v požadovanom riadku

Parametre

`row` poradové číslo riadku, ktorého dáta chceme zobrazit'

Poznámka

Funkcia vráti všetky údaje v riadku bez ohľadu na to, či príslušný stĺpec je maskovaný, alebo nie

Pozri tiež

Prehľad CResultSet | Členské funkcie | SetMask

GetHeadItem

vráti reťazec popisujúci požadovaný stĺpec hlavičky

```
char* GetHeadItem(int index);
```

Návratová hodnota

smerník na reťazec znakov popisujúci daný stĺpec

Parametre

index poradové číslo stĺpca, ktorého popis chceme zobrazit'

Poznámka

Funkcia nezohľadňuje maskovanie, počítajú sa aj neviditeľné stĺpce

Pozri tiež

Prehľad CResultSet | Členské funkcie | SetMask

GetString

vráti reťazec znakov obsahujúci všetky čísla medzi dvoma pozíciami v tabuľke

```
bool GetString(CPoint FirstCell, CPoint LastCell, CString &BigS, bool AppendTab=false);
```

Návratová hodnota

true ak operácia prebehla bezchybne, false v opačnom prípade (napr. nesprávne zadané bunky tabuľky)

Parametre

FirstCell prvá bunka výberu vo formáte (riadok, stĺpec)
LastCell posledná bunka výberu vo formáte (riadok, stĺpec)
BigS string kumulujúci výsledky jednotlivých buniek s tabelátorom medzi číslami, ak je nastavené AppendTab. Riadky sú oddelené znakom konca riadku '\n'
AppendTab udáva, či sa majú oddeľovať reťazce tabelátorom

Poznámka

Funkcia nezohľadňuje maskovanie, počítajú sa aj neviditeľné stĺpce

Pozri tiež

Prehľad CResultSet | Členské funkcie | SetMask

GetSize

vráti rozmery tabuľky (počet stĺpcov a počet riadkov)

```
CSize GetSize();
```

Návratová hodnota

Rozmery tabuľky v tvare *CSize*, kde *cx* udáva počet stĺpcov a *cy* počet riadkov (mimo hlavičky). Nezohľadňuje maskovanie stĺpcov.

Pozri tiež

Prehľad CResultTable | Členské funkcie | SetMask

GetCompressedTabSize

vráti rozmery maskovanej tabuľky (počet viditeľných stĺpcov a počet riadkov)

```
CSize GetCompressedTabSize();
```

Návratová hodnota

Rozmery tabuľky v tvare *CSize*, kde *cx* udáva počet viditeľných stĺpcov a *cy* počet riadkov (mimo hlavičky)

Pozri tiež

Prehľad CResultTable | Členské funkcie | SetMask

Empty

udáva, či je tabuľka prázdna (nulový počet dátových stĺpcov)

```
bool Empty();
```

Návratová hodnota

true ak je tabuľka prázdna, false inak

Poznámka

Prázdna tabuľka je taká, ktorá má nulový počet riadkov (okrem hlavičky).

Pozri tiež

Prehľad CResultTable | Členské funkcie | SetMask

RemoveAll

vynuluje tabuľku

```
void RemoveAll();
```

Poznámka

Vynulovanie tabuľky znamená odstránenie všetkých riadkov z pamäti a nastavenie veľkosti poľa riadkov na nulu

Pozri tiež

Prehľad CResultSet | Členské funkcie | SetMask

RemoveAt

odstráni určitý riadok tabuľky

```
void RemoveAt(int idx);
```

Parametre

idx poradové číslo riadku, ktorý treba odstrániť

Poznámka

Odstránenie riadku tabuľky znamená vynulovanie pamäte alokovanej riadkom a aplikovanie funkcie RemoveAt na pole riadkov.

Pozri tiež

Prehľad CResultSet | Členské funkcie | CArray::RemoveAt

12 Podrobný popis funkcií Plugin Interface

12.1 Funkcie pre čítanie z PIN

GetPINVersion

Vráti číslo verzie aktuálneho PIN

```
int GetPINVersion();
```

Návratová hodnota

číslo verzie aktuálneho PIN

Poznámka

Umožní kontrolu verzie PIN. Každá ďalšia verzia PIN môže obsahovať ďalšie rozširujúce funkcie, pričom je vždy zachovávaná kompatibilita smerom nadol.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | CPluginInterface:: GetPINSize

GetPINSize

Vráti veľkosť PIN v bajtoch.

```
int GetPinSize();
```

Návratová hodnota

veľkosť PIN v bajtoch

Poznámka

Používa sa pri kontrole kompatibility verzií.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | CPluginInterface:: GetPINVersion

GetViewZoomVal

Vráti aktuálny koeficient hodnoty ZOOM

```
double GetViewZoomVal ();
```

Návratová hodnota

aktuálny koeficient ZOOMu, ktorý je nastavený v Ellipse.

Poznámka

koeficient Zoom = 1.0 pre zobrazenie 1:1. Koeficient sa dá nastaviť iba „ručne“ na lište Ellipse, z modulu sa dá iba čítať, nie nastavovať.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie

GetViewTrackPoly

Vráti smerník na posledný polygón nakreslený pomocou nástrojov Ellipse.

```
CPolygon* GetViewTrackPoly();
```

Návratová hodnota

smerník na posledný polygón nakreslený pomocou nástrojov Ellipse.

Poznámka

Pri použití ľubovoľného kresliaceho nástroja v Ellipse (ikona na lište) je posledne nakreslený polygón dočasne uložený a pri každom ďalšom použití prekresľovaný. Táto funkcia vráti smerník na posledne nakreslený polygón.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | CTracker

GetViewCanAddPoly

Vráti stav premennej povoľujúcej alebo zabraňujúcej pridanie posledne nakresleného polygónu do OverlayPoly.

```
bool GetViewCanAddPoly();
```

Návratová hodnota

stav premennej povoľujúcej alebo zabraňujúcej pridanie posledne nakresleného polygónu do OverlayPoly a tým jeho zobrazovanie v Ellipse.

Poznámka

Po každom použití kresliaceho nástroja *Ellipse* (ikona na lište) sa výsledný polygón vždy zapíše do dočasnej premennej typu CPolygon. V *Ellipse* existuje vnútorná boolovská premenná, ktorú vieme nastaviť pomocou funkcie *SetViewCanAddPoly*. Táto premenná povoľuje, aby sa po každom kreslení nakreslený polygón pridal do poľa polygónov tvoriacich *OverlayPoly*, ktoré sú zobrazované. Pomocou funkcie *GetViewCanAddPoly* vieme zistiť stav spomínanej premennej.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | COverlayPoly | SetViewCanAddPoly

GetViewCurClass

Vráti číslo prednastavenej triedy, ktorú udáva aj Combo box na lište

```
int GetViewCurClass();
```

Návratová hodnota

číslo prednastavenej triedy

Poznámka

Prednastavená trieda je zobrazená pomocou Combo boxu na lište. Každý objekt nakreslený pomocou kresliacich prostriedkov je vždy typu *CPolygon* a bude automaticky zaradený do prednastavenej triedy. Polygóny zaradené do rozličných tried sú zobrazované rozličnou farbou určenou v *Ellipse Settings*. Číslo triedy sa dá nastaviť iba „ručne“ na lište *Ellipse*, z modulu sa dá iba čítať, nie nastavovať.

Pozri tiež

Prehľad *CPluginInterface* | Členské funkcie | *CPolygon* | *Ellipse Settings*

GetViewDrawingTools

Vráti pole boolovských premenných určujúce stav ikon kresliacich prostriedkov

```
BOOL* GetViewDrawingTools();
```

Návratová hodnota

pole boolovských premenných určujúce stav ikon kresliacich prostriedkov

Poznámka

Pomocou funkcie *SetViewDrawingTools* je možné z modulu povoliť, alebo znefunkčniť ikonu určitého kresliaceho prostriedku na lište. Nefunkčný prostriedok sa prejaví zašedenou ikonou aj zodpovedajúcim príkazom menu. Funkcia *GetViewDrawingTools* vráti pole boolovských premenných, ktoré vyjadrujú funkčnosť jednotlivých tlačítok tak, ako sú usporiadané na lište v smere zľava doprava (napr. nultý prvok poľa predstavuje ikonu na kreslenie bodu).

Pozri tiež

Prehľad *CPluginInterface* | Členské funkcie | *COverlayPoly* | *SetViewDrawingTools*

GetViewDrawnPolygons

Vráti stav premennej povoľujúcej zobrazenie *OverlayPoly*

```
bool GetViewDrawnPolygons();
```

Návratová hodnota

stav premennej povoľujúcej zobrazenie *OverlayPoly*

Poznámka

V *Ellipse* je vnútorná premenná typu *COverlayPoly*, ktorá slúži na archivovanie polygónov za účelom ich zobrazenia ako overlay obrázku. Samotné zobrazenie je možné povoliť, alebo mu zabrániť nastavením vnútornej premennej pomocou funkcie *SetViewDrawPolygons*. Funkcia *GetViewDrawPolygons* vráti aktuálny stav tejto premennej.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | SetViewDrawPolygons

GetCalDensLUT

Vráti smerník na kalibračnú tabuľku optickej hustoty LUT

```
double* GetCalDensLUT();
```

Návratová hodnota

smerník na kalibračnú tabuľku optickej hustoty LUT

Poznámka

Ellipse dokáže kalibrovať hodnoty jasů v šedotónových, alebo indexovaných paletovaných obrázkoch, t.j. každý pixel predstavuje iba 1 hodnotu. LUT je pole hodnôt typu double, ktoré udáva prepočet medzi hodnotou pixelu (poradové číslo v tabuľke) a reálnou fyzikálnou veličinou (hodnota tabuľky na tomto mieste). Dĺžku tabuľky je možné získať pomocou funkcie *GetCalDensN()*

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Kalibrácia | GetCalDensN

GetCalDensN

Vráti veľkosť LUT

```
int GetCalDensN();
```

Návratová hodnota

veľkosť LUT (v Ellipse je štandardne 256)

Poznámka

V Ellipse sa používa štandardná dĺžka tabuľky 256 hodnôt. Pretože v budúcnosti nie je vylúčený prechod na kalibrovanie 16 bitových pixelov, doporučujeme v záujme kompatibility používať na zisťovanie dĺžky tabuľky popisovanú funkciu miesto konštanty 256.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Kalibrácia | GetCalDensLUT

GetCalDensCalibrated

Vráti stav premennej, či bola optická hustota kalibrovaná

```
bool GetCalDensCalibrated();
```

Návratová hodnota

stav premennej, či bola optická hustota kalibrovaná

Pozri tiež

GetCalDistConstXY

vráti kalibračnú konštantu v smere osí X a Y

```
double GetCalDistConstXY();
```

Návratová hodnota

kalibračná konštantu v smere osí X a Y

Poznámka

Ellipse predpokladá rovnakú kalibračnú konštantu v smeroch osí X a Y.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Kalibrácia

GetCalDistConstZ

vráti kalibračnú konštantu v smere osi Z

```
double GetCalDistConstZ();
```

Návratová hodnota

kalibračná konštantu v smere osi Z

Poznámka

Pokiaľ používame kalibrovaný systém v smere osí XY a nenastavíme kalibračnú konštantu v smere osi Z, táto bude automaticky nastavená na hodnotu 1.0.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Kalibrácia

GetCalDistUnits

vráti popis kalibrovaných jednotiek (napr. „mm“)

```
char* GetCalDistUnits();
```

Návratová hodnota

popis kalibrovaných jednotiek

Poznámka

Priestorová kalibrácia v Ellipse dovoľuje nastaviť nasledovné jednotky dĺžky: „nm“, „um“, „mm“, „cm“, „m“, „km“. V prípade nekalibrovaných hodnôt ostáva jednotka „pix“.

Pozri tiež

GetCalDistCalibratedXY

Vráti informáciu či bol obraz kalibrovaný v smere XY

```
bool GetCalDistCalibratedXY();
```

Návratová hodnota

informácia či bol obraz kalibrovaný v smere XY

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Kalibrácia | GetCalDistCalibratedZ

GetCalDistCalibratedZ

Vráti informáciu či bol obraz kalibrovaný v smere Z

```
bool GetCalDistCalibratedZ();
```

Návratová hodnota

informácia či bol obraz kalibrovaný v smere Z

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Kalibrácia | GetCalDistCalibratedXY

GetSlidrInvert

Vráti mód zobrazovania slidera (invertovaný alebo nie)

```
BOOL GetSlidrInvert();
```

Návratová hodnota

mód zobrazovania slidera (invertovaný alebo nie)

Poznámka

Pokiaľ nastavíme invertované zobrazovanie, bude prvý obrázok série zobrazený pri polohe slidera celkom hore.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Slider | SetSlidrInvert

GetSlidrEnableInvert

Vráti stav premennej povoľujúcej invertovanie slidera

```
BOOL GetSlidrEnableInvert();
```

Návratová hodnota

stav premennej povoľujúcej invertovanie slidera

Poznámka

V niektorých moduloch je nežiaduce mať možnosť invertovať slider a preto existuje v Ellipse premenná, ktorá znemožní invertovanie tým, že znefunkční príslušný checkbox slidera.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Slider | SetSlidrEnableInvert

GetSlidrMinSel

Vráti minimálnu hodnotu SELECTION slidera

```
int GetSlidrMinSel();
```

Návratová hodnota

minimálna hodnota SELECTION slidera

Poznámka

Minimálna je tá poloha, ktorá predstavuje nižšie číslo hladiny (to je nezávislé od toho, či zobrazujeme invertovane, alebo nie).

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Slider | SetSlidrMinSel

GetSlidrMaxSel

Vráti maximálnu hodnotu SELECTION slidera

```
int GetSlidrMaxSel();
```

Návratová hodnota

maximálna hodnota SELECTION slidera

Poznámka

Maximálna je tá poloha, ktorá predstavuje vyššie číslo hladiny (to je nezávislé od toho, či zobrazujeme invertované, alebo nie).

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Slider | SetSlidrMaxSel

GetSlidrPos

Vráti aktuálnu pozíciu slidera

```
int GetSlidrPos();
```

Návratová hodnota

aktuálna pozícia slidera

Poznámka

Poloha slidera je vždy rovnaká bez ohľadu na to či máme invertovaný mód alebo nie

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Slider | SetSldrPos

GetSldrLastPos

Vráti polohu záložky – t.j. premennej na zapamätanie určitej (poslednej) pozície.

```
int GetSldrLastPos();
```

Návratová hodnota

stav záložky - premennej na zapamätanie pozície.

Poznámka

Záložka sa nenastavuje automaticky, ale algoritmus musí polohu, ktorú si chce zapamätať, nastaviť pomocou funkcie SetSldrLastPos. Táto poloha je zobrazovaná v širokom móde malou vodorovnou čiarkou.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Slider | SetSldrLastPos

GetSldrLabMin

Vráti reťazec znakov popisujúcich minimálnu SELECTION

```
char* GetSldrLabMin();
```

Návratová hodnota

reťazec znakov popisujúcich minimálnu SELECTION

Poznámka

Ako predvolená hodnota je v Ellipse nastavený reťazec znakov „min“. Pomocou funkcie SetSldrLabMin je možné označiť minimálnu polohu iným názvom v rozsahu 4 znakov.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Slider | SetSldrLabMin

GetSldrLabMax

Vráti reťazec znakov popisujúcich maximálnu SELECTION

```
char* GetSldrLabMax();
```

Návratová hodnota

reťazec znakov popisujúcich maximálnu SELECTION

Poznámka

Ako predvolená hodnota je v *Ellipse* nastavený reťazec znakov „max“. Pomocou funkcie *SetSliderLabMax* je možné označiť maximálnu polohu iným názvom v rozsahu 4 znakov.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Slider | SetSliderLabMax

GetStackImages

Vráti smerník na začiatok poľa obrazov (CImg**)

```
CImg** GetStackImages();
```

Návratová hodnota

smerník na začiatok poľa obrazov (CImg**)

Poznámka

Začiatok poľa obrazov predstavuje smerník naobraz s indexom „0“ a zodpovedá polohe minimálnej polohy slidera (dole v neinvertovanom, hore v invertovanom mode). Pomocou indexu sa dá získať smerník na ľubovoľný obrázok stacku.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | CImg

GetStackCurImage

Vráti smerník na práve zobrazovaný obraz v stacku(CImg*)

```
CImg* GetStackCurImage();
```

Návratová hodnota

smerník na práve zobrazovaný obraz v stacku(CImg*)

Poznámka

Zjednodušuje získanie smerníka na aktuálny obraz keďže automaticky vypočíta index podľa pozície slidera a tento index použije na prístup k aktuálnemu obrázku v stacku.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | CImg

GetStackOverlayPolys

Vráti smerník na začiatok poľa overlayov (COverlayPoly**)

```
COverlayPoly** GetStackOverlayPolys();
```

Návratová hodnota

smerník na začiatok poľa overlayov (COverlayPoly**)

Poznámka

Začiatok poľa overlayov typu *COverlayPoly* predstavuje smerník na *OverlayPoly* s indexom „0“ a zodpovedá polohe minimálnej polohe slidera (dole v neinvertovanom, hore v invertovanom mode). Pomocou indexu sa dá získať smerník na ľubovoľný *OverlayPoly* v stacku.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | COverlayPoly

GetStackCurOverlay

Vráti smerník na práve zobrazovaný overlay typu *COverlayPoly**

```
COverlayPoly* GetStackCurOverlay();
```

Návratová hodnota

smerník na práve zobrazovaný overlay typu *COverlayPoly**

Poznámka

Zjednodušuje získanie smerníka na aktuálny overlay, keďže automaticky vypočíta index podľa pozície slidera a tento index použije na prístup k aktuálnemu overlay (typu *COverlayPoly*) v stacku.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | COverlayPoly

GetStackCaptions

Vráti smerník na názvy obrazov v stacku (*CString**)

```
CString* GetStackCaptions();
```

Návratová hodnota

smerník na názvy obrazov v stacku (*CString**)

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | CImg

GetStackXYSize

Vráti veľkosť obrázku v tvare *CSize* v smere osí X a Y

```
CSize GetStackXYSize();
```

Návratová hodnota

veľkosť obrázku v tvare *CSize* v smere osí X a Y

Poznámka

V skutočnosti sa testuje iba rozmer prvého obrázku v stacku (s indexom 0), predpokladajúc, že všetky obrázky v stacku majú rovnaké rozmery.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | CImg

GetStackZSize

Vráti veľkosť v smere Z, čo je totožné s počtom obrazov v stacku

```
int GetStackZSize();
```

Návratová hodnota

veľkosť v smere Z, čo je totožné s počtom obrazov v stacku

Poznámka

V prípade že nemáme stack, ale iba jeden obraz je táto hodnota 1.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | CImg

GetStackModifiedImg

Vráti stav premennej udávajúcej, či boli obrázky menené

```
bool GetStackModifiedImg();
```

Návratová hodnota

stav premennej udávajúcej, či boli obrázky menené

Poznámka

Existuje premenná, ktorá zaregistruje, že sa pomocou *Ellipse* uskutočnili na obrazoch nejaké zmeny. Je užitočné poznať aj v module stav tejto premennej a tiež mať možnosť ho meniť (pomocou funkcie *SetStackModifiedImg*).

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | SetStackModifiedImg

GetUserData

Vráti smerník na blok vyhradený pre dáta, ktoré majú ostať aj po uzavretí modulu (void*)

```
void* GetUserData();
```

Návratová hodnota

smerník na blok vyhradený pre dáta, ktoré majú ostať aj po uzavretí modulu (void*)

Poznámka

Po ukončení činnosti modulu preberá riadenie opäť *Ellipse* a zrušia sa aj všetky premenné alokované v module. Jedinou výnimkou je blok dát vyhradený užívateľovi modulu, ktorý ostane alokovaný aj po zavretí modulu (tzv. *User data*) a umožňuje tak výmenu dát medzi modulmi. Tieto dáta sa zmažú až pri deštrukcii príslušného dokumentu. Pretypovanie dát do vhodného tvaru je plne v režii programátora modulu.

Pozri tiež

Prehľad CPluginInterface Členské funkcie | SetUserData

GetPathModule

Vráti reťazec udávajúci cestu k práve otvorenému modulu

```
char* GetPathModule();
```

Návratová hodnota

reťazec udávajúci cestu k práve otvorenému modulu

Pozri tiež

Prehľad CPluginInterface | Členské funkcie

GetPathImages

Vráti reťazec udávajúci cestu k práve otvorenému obrazu

```
char* GetPathImages();
```

Návratová hodnota

reťazec udávajúci cestu k práve otvorenému obrazu

Pozri tiež

Prehľad CPluginInterface | Členské funkcie |

GetPathEllipse

Vráti reťazec udávajúci cestu k „Ellipse.exe“

```
char* GetPathEllipse();
```

Návratová hodnota

reťazec udávajúci cestu k „Ellipse.exe“

Poznámka

Cesta, ktorá je predvolená pri inštalácii je *C:/Program Files/Ellipse/*. Užívateľ môže túto cestu pri inštalácii zmeniť, kvôli predvoleným nastaveniam v projekte sa to ale nedoporučuje. (viď kapitola 2.7.)

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Organizácia adresárov v Ellipse

GetResultTable

Vráti smerník na tabuľku výsledkov *CResultTable**

```
CResultTable* GetResultTable();
```

Návratová hodnota

smerník na tabuľku výsledkov *CResultTable**

Poznámka

Získaný smerník dovoľuje konštruovať vlastnú tabuľku výsledkov, avšak zobrazovanú pomocou Okna výsledkov *Ellipse* (viď kap. 2.6)

Pozri tiež

Prehľad *CPluginInterface* | Členské funkcie | *CResultTable*

12.2 Funkcie pre zápis prostredníctvom PIN

SetViewCanAddPoly

nastavenie premennej povoľujúcej alebo zabraňujúcej prídanie posledne nakresleného polygónu do *OverlayPoly* a tým jeho zobrazovanie v *Ellipse*.

```
void SetViewCanAddPoly(bool Can, bool bSave = true);
```

Parametre:

Can nastavovaná hodnota (true, false)
bSave udáva, či sa má nastavenie vykonať okamžite, alebo až príkazom *SetPIN()*

Poznámka

Po každom použití kresliaceho nástroja *Ellipse* (ikona na lište) sa výsledný polygón vždy zapíše do dočasnej premennej typu *CPolygon*. V *Ellipse* existuje vnútorná boolovská premenná, ktorú vieme nastaviť pomocou funkcie *SetViewCanAddPoly*. Táto premenná povoľuje, aby sa po každom kreslení nakreslený polygón pridal do poľa polygónov tvoriacich *OverlayPoly*, ktoré sú zobrazované. Pomocou funkcie *GetViewCanAddPoly* vieme zistiť stav spomínanej premennej.

Pozri tiež

Prehľad *CPluginInterface* | Členské funkcie | *COverlayPoly* | *GetViewCanAddPoly*

SetViewDrawingTools

nastavenie poľa ikon kresliacich prostriedkov

```
void SetViewDrawingTools(int which, BOOL View, bool bSave = true);
```

Parametre:

which udáva poradové číslo nastavovaného nástroja
View nastavovaná hodnota (TRUE, FALSE)
bSave udáva, či sa má nastavenie vykonať okamžite, alebo až príkazom SetPIN()

Návratová hodnota

pole boolovských premenných určujúce stav ikon kresliacich prostriedkov

Poznámka

Pomocou tejto funkcie je možné z modulu povoliť, alebo znefunkčnit' ikonu určitého kresliaceho prostriedku na lište, v poradí ako sú usporiadané na v smere zľava doprava (napr. nultý prvok poľa predstavuje ikonu na kreslenie bodu). Nefunkčný prostriedok sa prejaví zašedenou ikonou aj zodpovedajúcim príkazom menu.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | COverlayPoly | GetViewDrawingTools

SetViewDrawnPolygons

nastavenie premennej povoľujúcej zobrazenie OverlayPoly

```
void SetViewDrawnPolygons(bool View, bool bSave = true);
```

Parametre:

View nastavovaná hodnota (true, false)
bSave udáva, či sa má nastavenie vykonať okamžite, alebo až príkazom SetPIN()

Poznámka

V Ellipse je vnútorná premenná typu *COverlayPoly*, ktorá slúži na archivovanie polygónov za účelom ich zobrazenia ako overlay obrázku. Samotné zobrazenie je možné povoliť, alebo mu zabrániť nastavením vnútornej premennej pomocou funkcie *SetViewDrawPolygons*. Funkcia *GetViewDrawPolygons* vráti aktuálny stav tejto premennej.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | GetViewDrawPolygons

SetCalDensCalibrated

nastavenie premennej, o kalibrácii optickej hustoty

```
void SetCalDensCalibrated(bool Calibrated, bool bSave);
```

Parametre:

Calibrated nastavovaná hodnota (true, false)

bSave udáva, či sa má nastavenie vykonať okamžite, alebo až príkazom *SetPIN()*

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | GetCalDensCalibrated

SetCalDistConstXY

nastaví kalibračnú konštantu v smere osí X a Y

```
void SetCalDistConstXY(double ConstXY, bool bSave = true);
```

Parametre:

ConstXY hodnota kalibračnej konštanty

bSave udáva, či sa má nastavenie vykonať okamžite, alebo až príkazom *SetPIN()*

Poznámka

Ellipse predpokladá rovnakú kalibračnú konštantu v smeroch osí X a Y.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Kalibrácia | GetCalDistConstXY

SetCalDistConstZ

nastaví kalibračnú konštantu v smere osí Z

```
void SetCalDistConstZ(double ConstZ, bool bSave = true);
```

Parametre:

ConstZ hodnota kalibračnej konštanty

bSave udáva, či sa má nastavenie vykonať okamžite, alebo až príkazom *SetPIN()*

Poznámka

Pokiaľ používame kalibrovaný systém v smere osí XY a nenastavíme kalibračnú konštantu v smere osi Z, táto bude automaticky nastavená na hodnotu 1.0.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Kalibrácia | GetCalDistConstZ

SetCalDistUnits

definuje popis kalibrovaných jednotiek (napr. „mm“)

```
void SetCalDistUnits(const char* Units, bool bSave = true);
```

Parametre:

Units reťazec znakov použitý na popis jednotiek

bSave udáva, či sa má nastavenie vykonať okamžite, alebo až príkazom *SetPIN()*

Poznámka

Priestorová kalibrácia v Ellipse dovoľuje nastaviť nasledovné jednotky dĺžky: „nm“, „um“, „mm“, „cm“, „m“, „km“. V prípade nekalibrovaných hodnôt ostáva jednotka „pix“.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Kalibrácia | GetCalDistUnits

SetCalDistCalibratedXY

nastaví informáciu o tom, či bolo vykonané kalibrovanie v smere XY

```
void SetCalDistCalibratedXY(bool CalibratedXY, bool bSave = true);
```

Parametre:

CalibratedXY nastavovaná hodnota (true, false)

bSave udáva, či sa má nastavenie vykonať okamžite, alebo až príkazom *SetPIN()*

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Kalibrácia | GetCalDistCalibratedXY

SetCalDistCalibratedZ

nastaví informáciu o tom, či bolo vykonané kalibrovanie v smere Z

```
void SetCalDistCalibratedZ(bool CalibratedZ, bool bSave = true);
```

Parametre:

CalibratedZ nastavovaná hodnota (true, false)

bSave udáva, či sa má nastavenie vykonať okamžite, alebo až príkazom *SetPIN()*

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Kalibrácia | GetCalDistCalibratedZ

SetSldrInvert

nastaví mód zobrazovania slidera (Invert/NonInvert)

```
void SetSldrInvert(BOOL Invert, bool bSave = true);
```

Parametre:

Invert nastavovaná hodnota (true, false)

bool bSave udáva, či sa má nastavenie vykonať okamžite, alebo až príkazom *SetPIN()*

Poznámka

Pokiaľ nastavíme invertované zobrazovanie, bude prvý obrázok série zobrazený pri polohe slidera celkom hore.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Slider | GetSldrInvert

SetSlidrEnableInvert

povolí, alebo zakáže invertovanie slidera

```
void SetSlidrEnableInvert(BOOL EnableInvert, bool bSave = true);
```

Parametre:

EnableInvert nastavovaná hodnota (true, false)

bool bSave udáva, či sa má nastavenie vykonať okamžite, alebo až príkazom *SetPIN()*

Poznámka

V niektorých moduloch je nežiaduce mať možnosť invertovať slider a preto existuje v *Ellipse* premenná, ktorá znemožní invertovanie tým, že znefunkční príslušný checkbox slidera.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Slider | GetSlidrEnableInvert

SetSlidrMinSel

nastaví minimálnu hodnotu SELECTION slidera

```
void SetSlidrMinSel(int MinSel, bool bSave = true);
```

Parametre:

MinSel hodnota, ktorá sa má nastaviť

bool bSave udáva, či sa má nastavenie vykonať okamžite, alebo až príkazom *SetPIN()*

Poznámka

Minimálna je tá poloha, ktorá predstavuje nižšie číslo hladiny (to je nezávislé od toho, či zobrazujeme invertované, alebo nie).

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Slider | GetSlidrMinSel

SetSlidrMaxSel

nastaví maximálnu hodnotu SELECTION slidera

```
void SetSlidrMaxSel(int MaxSel, bool bSave = true);
```

Parametre:

MaxSel hodnota, ktorá sa má nastaviť

bool bSave udáva, či sa má nastavenie vykonať okamžite, alebo až príkazom *SetPIN()*

Poznámka

Maximálna je tá poloha, ktorá predstavuje vyššie číslo hladiny (to je nezávislé od toho, či zobrazujeme invertované, alebo nie).

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Slider | GetSlidrMaxSel

SetSldrPos

nastaví novú pozíciu slidera

```
void SetSldrPos(int Pos, bool bSave = true);
```

Parametre:

Pos hodnota novej pozície slidera
bool bSave udáva, či sa má nastavenie vykonať okamžite, alebo až príkazom *SetPIN()*

Poznámka

Poloha slidera je vždy rovnaká bez ohľadu na to či máme invertovaný mód alebo nie

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Slider | GetSldrPos

SetSldrLastPos

nastaví hodnotu „záložky“ slidera

```
void SetSldrLastPos(int LastPos, bool bSave);
```

Parametre:

LastPos hodnota záložky slidera
bool bSave udáva, či sa má nastavenie vykonať okamžite, alebo až príkazom *SetPIN()*

Poznámka

Záložka sa nenastavuje automaticky, ale algoritmus musí polohu, ktorú si chce zapamätať, nastaviť pomocou funkcie *SetSldrLastPos*. Táto poloha je zobrazovaná v širokom móde malou vodorovnou čiarkou.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Slider | GetSldrLastPos

SetSldrLabMin

popíše minimálnu hodnotu SELECTION slidera

```
void SetSldrLabMin(const char* LabMin, bool bSave = true);
```

Parametre:

LabMin reťazec znakov použitých na popis
bool bSave udáva, či sa má nastavenie vykonať okamžite, alebo až príkazom *SetPIN()*

Poznámka

Ako predvolená hodnota je v Ellipse nastavený reťazec znakov „min“. Pomocou funkcie *SetSldrLabMin* je možné označiť minimálnu polohu iným názvom v rozsahu 4 znakov.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Slider | GetSldrLabMin

SetSldrLabMax

popíše maximálnu hodnotu SELECTION slidera

```
void SetSldrLabMax(const char* LabMax, bool bSave);
```

Parametre:

LabMax reťazec znakov použitých na popis
bool bSave udáva, či sa má nastavenie vykonať okamžite, alebo až príkazom *SetPIN()*

Poznámka

Ako predvolená hodnota je v Ellipse nastavený reťazec znakov „max“. Pomocou funkcie *SetSldrLabMax* je možné označiť maximálnu polohu iným názvom v rozsahu 4 znakov.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | Slider | GetSldrLabMax

SetStackCaptions

umožní zmeniť názov učitého obrázku v stacku

```
void SetStackCaptions(int which, CString Caption, bool bSave = true);
```

Parametre:

which index popisovaného obrázku
Caption reťazec znakov použitých na popis
bool bSave udáva, či sa má nastavenie vykonať okamžite, alebo až príkazom *SetPIN()*

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | CImg | GetStackCaptions

SetStackModifiedImg

nastaví premennú informujúcu o zmene obrazov

```
void SetStackModifiedImg(bool ModifiedImg, bool bSave = true);
```

Parametre

ModifiedImg nastaví premennú reagujúcu na zmeny v obrázku
bool bSave udáva, či sa má nastavenie vykonať okamžite, alebo až príkazom *SetPIN()*

Poznámka

Existuje premenná, ktorá zaregistruje, že sa pomocou Ellipse uskutočnili na obrazoch nejaké zmeny. Je užitočné poznať aj v module stav tejto premennej a tiež mať možnosť ho meniť (pomocou funkcie *SetStackModifiedImg*).

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | GetStackModifiedImg

SetUserData

uloží užívateľské dáta na určené miesto

```
void SetUserData(void* pUserData, bool bSave = true)
```

Parametre

pUserData smerník na blok dát vytvorených užívateľom, trvajúcich aj po uzavretí modulu
bool bSave udáva, či sa má nastavenie vykonať okamžite, alebo až príkazom SetPIN()

Poznámka

Po ukončení činnosti modulu preberá riadenie opäť *Ellipse* a zrušia sa aj všetky premenné alokované v module. Jedinou výnimkou je blok dát vyhradený užívateľovi modulu, ktorý ostane alokovaný aj po zavretí modulu (tzv. *User data*) a umožňuje tak výmenu dát medzi modulmi. Tieto dáta sa zmažú až pri deštrukcii príslušného dokumentu. Pretypovanie dát do vhodného tvaru je plne v réžii programátora modulu.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie | GetUserData

SetPIN

zapíše naraz všetky zmeny z PIN do dokumentu

```
void SetPIN ();
```

Poznámka

Všetky funkcie pre zápis do PIN (funkcie Set...) majú ako parameter *bSave* s prednastavenou hodnotou *true*, čo znamená, že zmena sa okamžite zapíše do príslušného dokumentu. Pokiaľ zavoláme funkciu *Set...* s parametrom *bSave = false*, zmena sa zapíše iba do vnútornej premennej v PIN a až po zavolaní funkcie *SetPIN()* sa skopíruje do dokumentu. Túto verziu využívame vtedy, keď potrebujeme nastaviť viacej premenných naraz. Najprv voláme viacero funkcií *Set...* a potom zmeny všetky naraz skopírujeme do dokumentu, čo je oveľa úspornejšie.

Pozri tiež

Prehľad CPluginInterface | Členské funkcie

12.3 Špeciálne funkcie PIN

CPluginInterface(CDocument* pDoc);

konštruktor vytvárajúci PIN na základe existujúceho dokumentu

```
CPluginInterface(CDocument* pDoc);
```

Parametre

pDoc smerník na dokument, ku ktorému vytvárame interface

Poznámka

Vytvorenie PIN na základe existujúceho dokumentu sa požíva vtedy, keď modul potrebuje získať prístup ku aktuálnym dátam dokumentu (existujúcemu obrazu) a na základe nich vykonať spracovanie obrazu.

Pozri tiež

Prehľad CPluginInterface Členské funkcie | Využitie PluginInterface

CPluginInterface(CImg pDib,...)**

konštruktor vytvárajúci PIN na základe existujúceho obrazu

```
CPluginInterface(CImg** pImg, int n=1);
```

Parametre

pImg smerník na obraz (alebo stack) vytvorený v module
n počet obrázkov v prípade, že sa jedná o stack

Poznámka

Vytvorenie PIN na základe existujúceho obrazu sa požíva vtedy, keď modul vygeneruje nový stack (resp. obraz) a potrebuje, aby ho Ellipse zaregistrovala ako nový dokument. Súčasne s konštrukciou sa vytvorí a inicializuje aj nový dokument, ktorý je možné ešte ďalej nastavovať jednotlivými funkciami *PluginInterface* z kategórie *Set*.

Pozri tiež

Prehľad CPluginInterface Členské funkcie | Využitie PluginInterface

~CPluginInterface

deštruktor PIN

```
~CPluginInterface(void);
```

Poznámka

Deštruktor vymaže všetky vytvorené premenné.

Pozri tiež

Prehľad CPluginInterface Členské funkcie | Využitie PluginInterface

AddNewImages

Pridá obrázky ku aktívnemu dokumentu

```
void AddNewImages(CImg** pImg, int N=1);
```

Parametre

pImg smerník na obraz (alebo prvý obraz v stack) vytvorený v module
N počet obrazov v stacku

Pozri tiež

Prehľad CPluginInterface Členské funkcie | Využitie PluginInterface
