Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Možnosti 2D vykreslování v prostředí Windows

Martin Jelínek, DiS.

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Katedra informačního inženýrství

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Martin Jelínek

Informatika

Název práce

Možnosti 2D vykreslování v prostředí Windows

Název anglicky

2D graphics in Windows environment

Cíle práce

Hlavním cílem práce je popsat možnosti programování 2D grafiky v prostředí MS Windows se zvláštním zaměřením na technologii Direct2D. Dílčím cílem práce je vytvořit aplikaci demonstrující vybrané technologie, která bude zároveň sloužit k ověření technické náročnosti jednotlivých přístupů.

Metodika

Metodika řešené práce ja založena na studiu a analýze odborných informačních zdrojů. Na základě syntézy zjištěných poznatků budou popsány jednotlivé technologie využitelné při programování 2D grafiky v prostředí MS Windows, se zvláštním zaměřením na Direct2D. Dále bude navržena a implementována testovací aplikace využívající vybrané technologie, která bude umožňovat jejich vzájemné srovnání z hlediska náročnosti (rychlost vykreslování atp.). Výsledky zjištěné pomocí této aplikace budou dále popsány.

Doporučený rozsah práce

35-40 stran

Doporučené zdroje informací

DirectWrite [online]. (c) 2014 [cit. 2014-06-12]. Dostupné z:
http://msdn.microsoft.com/en-us/library/windows/desktop/dd368038%28v=vs.85%29.aspx
Direct2D [online]. (c) 2014 [cit. 2014-06-12]. Dostupné z:
http://msdn.microsoft.com/en-us/library/windows/desktop/dd370990%28v=vs.85%29.aspx
GDI+ [online]. (c) 2014 [cit. 2014-06-12]. Dostupné z:
http://msdn.microsoft.com/en-us/library/ms533798%28v=vs.85%29.aspx
PRATA, Stephen. C primer plus. 6th ed. Upper Saddle River, NJ: Addison-Wesley, c2012, xii, 1420 s. ISBN 978-0-321-77640-2.
Windows GDI [online]. (c) 2014 [cit. 2014-06-12]. Dostupné z:
http://msdn.microsoft.com/en-us/library/dd145203%28v=vs.85%29.aspx
Windows Imaging Component [online]. (c) 2014 [cit. 2014-06-12]. Dostupné z:
http://msdn.microsoft.com/en-us/library/windows/desktop/ee719902%28v=vs.85%29.aspx
YUAN, Feng. Windows graphics programming: Win32 GDI and DirectDraw. Upper Saddle River, NJ: Prentice Hall PTR, xli, 1234 p. ISBN 01-308-6985-6.

Předběžný termín obhajoby

2015/06 (červen)

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Elektronicky schváleno dne 10. 11. 2014

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 10. 11. 2014

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 05. 03. 2015

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Možnosti 2D vykreslování v prostředí Windows" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 12.3.2015 _____

Poděkování

Rád bych touto cestou poděkoval vedoucímu bakalářské práce Ing. Jiřímu Brožkovi, Ph.D. za odborné vedení a cenné rady při jejím zpracování.

Možnosti 2D vykreslování v prostředí Windows

2D graphics in Windows environment

Souhrn

Práce charakterizuje GDI, GDI+ a Direct2D (s podporou WIC a DirectWrite) API jakožto existující technologie pro programování 2D grafiky v prostředí Microsoft Windows v rámci programovacího jazyka C++. Charakteristika je doplněna o příklady, které ukazují základní práci s vybranými API. V praktické části je představena tvorba testovací aplikace, na které byly srovnány jednotlivé technologie mezi sebou a poté zhodnocena jejich náročnost.

Summary

This thesis describes GDI, GDI+ and Direct2D (with the support of WIC and DirectWrite) as existing technologies for 2D graphics programming in Microsoft Windows environment and in C++ programming language. Description is complemented with examples showing fundamentals with the API's mentioned above. In the practical part is presented the creation of testing application on which were compared the particular technologies with each other and than reviewed results.

Klíčová slova: WIC, GDI, GDI+, 2D vykreslování, C++, Direct2D, WinAPI, DirectWrite

Keywords: WIC, GDI, GDI+, 2D graphics, C++, Direct2D, WinAPI, DirectWrite

Obsah

1 Úvod

Při programování na nižší úrovni abstrakce v operačních systémech Microsoft Windows využíváme především API (aplikační rozhraní pro programování) nazvané Windows API[1], ve kterém lze programovat v C/C++. Pro určité typy programů využíváme pouze příkazovou řádku. Pokud však chceme vytvářet uživatelská rozhraní nebo jakékoliv grafické možnosti (od jednoduchého vykreslování pozadí po složité zpracování obrazových formátů) a nechceme (ať už z jakéhokoliv důvodu) propojovat kód s programovacími jazyky vyšší abstrakce (C#, Java atp.), je potřeba mít přehled o možnostech nativního 2D programování. Vědět tedy jaká API vůbec existují a která jsou v určité situaci vhodnější pro použití, čímž se tato práce zabývá.

[1] dále jen WinAPI; je to API, které nabízí nejrůznější služby systému pomocí knihoven - například tvorba oken, procesů, vláken, GDI, dialogových oken, komunikace po síti atd.

2 Cíl práce a metodika

Hlavním cílem této práce je charakterizovat vybraná API pro programování 2D grafiky v prostředí Microsoft Windows. Dílčím cílem je vytvoření aplikace srovnávající vybrané technologie z hlediska technické náročnosti jednotlivých přístupů.

Pro tuto práci byly vybrány tři aktuálně existující grafická API – GDI, GDI+ a Direct2D. Direct2D spolupracuje kromě GDI/GDI+ také s WIC, DirectWrite a dalšími. Aby mohl Direct2D pracovat s textem nebo s obrazovými formáty dat (JPEG, TIFF atd.), potřebuje právě WIC a DirectWrite API, která jsou na to specializována. V teoretické části práce je uvedena základní charakteristika výše zmíněných API a poté je demonstrováno na příkladech, jak se s nimi pracuje. Direct2D příklady jsou okomentované (uvnitř i vně) a ukazují nejen základní princip práce s danou technologií, ale zároveň jsou obsahem vytvořené aplikace v praktické části.

V praktické části práce je navržena a implementována testovací aplikace, která využívá GDI, GDI+ i Direct2D a vzájemně je srovnává z hlediska náročnosti. Veškerý kód pro zmíněnou aplikaci byl napsán v programu Microsoft Visual Studio 2013. Dále jsou popsána různá úskalí při tvorbě této aplikace a jednotlivé testy, z kterých se skládá samotné testování. A posléze jsou vyhodnoceny výsledky zjištěné aplikací, která měřila čas (v mikrosekundách), variační koeficient (kvůli důvěryhodnosti průměrů) a cykly procesoru vybraných funkcí pro každou technologii. Funkce byly vybrány tak, aby mohly být objektivně porovnány mezi GDI, GDI+ i Direct2D – tj. základní funkce, která vede ke stejnému výstupu (aby nezávisela náročnost na vlastním vytvořeném algoritmu, ale pouze na daném API). Obě části práce jsou více zaměřené na nejnovější API - Direct2D.

Práce je určena pro čtenáře, kteří mají alespoň základní znalosti programovacího jazyka C++ a zároveň již mají v povědomí principy programování založené na zprávách operačního systému Windows ve WinAPI.

3 Přehled řešené problematiky

3.1 GDI

GDI je grafické rozhraní umožňující aplikacím využívat grafické prvky a formátovaný text, který se zobrazí na výstupním zařízení jako je monitor nebo tiskárna.[2]

GDI pamatuje již mnoho let (minimálně od první verze operačního systému Windows firmy Microsoft (dále jen Windows)) a tak není divu, že je tu snaha o jeho nahrazení či vylepšení, ale jak bude uvedeno v dalších kapitolách, není vše tak jednoduché. Samotné API bylo navrženo pro programovací jazyky C i C++ s tím, že nevyužívá OOP (objektově orientované programování). V různých verzích Windows je omezení na počet vytvořených GDI objektů (objekt je například font, bitmapa, štětec atd.). V aktuálních systémech jako Windows 7 či Windows 8 je stále toto omezení, ale je zvětšeno na vyšší hodnoty ve výchozím nastavení či je lze upravit pomocí registrů. Proto je dobré s objekty zbytečně neplýtvat a mazat je, když už nejsou potřeba.

V GDI lze například tvořit:

- bitmapy
- kreslící štětce, pera
- tvary a křivky (úsečky, trojúhelníky, obdélníky)
- fonty a text
- barevné výplně
- ořezy

GDI pracuje s tzv. DC (kontext zařízení). DC je struktura odkazující na určitý výstup (paměťový, klientská část okna aplikace, prvek jako je tlačítko či textové pole, celá obrazovka, tiskárna atd.). Pokud kreslíme pomocí DC patřící oknu aplikace, pak se kreslí do paměti GPU (grafický procesor / čip; GPU pamětí je myšlena RAM na grafické kartě) a pokud pomocí paměťového DC, pak je to do RAM (systémová operační paměť). Jakmile získáme HDC (handle[3] na DC neboli přístup na kreslící plochu), můžeme ho předávat různým GDI funkcím (například funkci, která kreslí kružnice) a dle toho jakého typu je DC, se na daný výstup vykreslí kružnice.

[2] Windows GDI. MICROSOFT. *MSDN - Microsoft Developer Network* [online]. © 2015 [cit. 2015-01-27]. Dostupné z: https://msdn.microsoft.com/en-us/library/dd145203%28v=vs.85%29.aspx

[3] handle je identifikátor objektu; pro handle se český překlad „rukojeť“ či „držadlo“ příliš neujal

Paměťový DC lze využít na dvojité bufferování[4] a zároveň rychlejší kreslení (jak lze vidět v praktické části této práce). Dvojité bufferování je technika vykreslování, kdy jeden buffer slouží jako nezávislá kreslící plocha a druhý, který se posílá na výstup. V Direct3D[5] a jiných API se prohazují ukazatelé na buffery a značně tím snižují šanci na nechtěné artefakty jako více scén v jednom snímku, blikání atp. Naproti tomu GDI nebylo vytvořeno za účelem vysokého výkonu a složitých grafických výpočtů jako Direct3D, které přímo počítá s technikami jako mnohonásobné bufferování či vertikální synchronizace. V GDI se dvojité bufferování řeší podobně s tím, že se neprohazují ukazatelé, ale pomocí funkce BitBlt se kopíruje plocha z jednoho nezávislého paměťového bufferu do hlavního, který patří oknu aplikace. Nevýhoda je tedy zřejmá – než BitBlt zkopíruje paměťový buffer, tak mezitím se může kdykoliv překreslit okno aplikace a vznikat zobrazení například jen půlky scény. Avšak výhodou je například možnost kreslení mimo zprávu WM_PAINT, kdy kreslení zůstává a nezmizí (nemusí se nic validovat).

Kód 1 ukazuje, jak se pracuje s GDI ve standardně zachycené zprávě WM_PAINT.

```
HDC hdc;
PAINTSTRUCT ps;
HGDIOBJ hPero, hPeroPuvodni;
hdc = BeginPaint(hwnd, &ps);
hPero = CreatePen(PS_SOLID, 12, RGB(0, 0, 0));
hPeroPuvodni = SelectObject(hdc, hPero);
MoveToEx(hdc, 20, 10, NULL);
LineTo(hdc, 70, 60);
TextOut(hdc, 50, 100, "Ahoj", 4);
SelectObject(hdc, hPeroPuvodni);
DeleteObject(hPero);
EndPaint(hwnd, &ps);
```

Kód 1 – základní práce s GDI[6]

V kódu 1 jsme nejdříve vytvořili proměnné typu HDC, PAINTSTRUCT (struktura, která obsahuje informace pro kreslení) a HGDIOBJ (je univerzální objekt, který může být typu HPEN, HBRUSH atd.). Veškeré kreslení ve WM_PAINT musí být uvnitř funkcí

[4] buffer je paměť, která slouží pro dočasné uložení dat
[5] Direct3D je API pro práci s 3D grafikou
[6] zdroj: autor

BeginPaint a EndPaint. EndPaint slouží k validaci (potvrzení), že veškerá plocha je obnovena do správného stavu. Existují i jiné funkce na získání HDC, ale ty už implicitně nevalidují. Pokud se uvnitř WM_PAINT nepoužije validující funkce, tak Windows stále posílá zprávy typu WM_PAINT (pouze pokud je potřeba překreslit určitou oblast), dokud se nevaliduje veškerá plocha a tím vznikne nekonečný cyklus zatěžující procesor na maximum. Funkce BeginPaint vrací HDC a neměla by se volat mimo WM_PAINT, tam lze využít funkci GetDC. Dále zavoláme funkci na tvorbu pera, kde specifikujeme typ čáry (souvislá, přerušovaná atd.), tloušťku a barvu. Funkce vrací daný GDI objekt, který uložíme a poté pero přiřadíme do našeho HDC. Funkce MoveToEx posouvá pozici pro začátek kreslení, LineTo nakreslí úsečku od předchozí zvolené pozice do v této funkci zvolené pozice – vše jako souřadnice X a Y. TextOut je funkce, která od zvolených souřadnic vypíše text o specifikované délce ve výchozím fontu (velikosti, typu). Font lze změnit stejným způsobem jako pero a další objekty.

V každém kreslícím kontextu máme vše na výchozích hodnotách, takže pokud chceme změnit kreslící pero, zavoláme funkci SelectObject. Ta slouží pro přiřazení určitého objektu k danému HDC a vrací nám předchozí objekt. Předchozí objekt si ukládáme z důvodu, že až nebude potřeba vytvořené pero a budeme ho chtít smazat, tak nejdříve musíme aktuální pero zase vyřadit z přiřazení a vložit původní (protože jiné nemáme). Zkrátka objekty typu pero nebo štětec nelze smazat, pokud jsou stále přiřazené pomocí funkce SelectObject.

Navzdory novým rozhraním (GDI+, Direct2D či DirectWrite) v systému GDI stále zůstává z důvodu zachování kompatibility s tiskárnami a staršími aplikacemi (i systémovými).

GDI je na Windows XP a Windows 7 (pokud běží DWM[7] a jsou přítomny ovladače GPU na bázi WDDM[8] 1.1) hardwarově akcelerované (běh určitých operací na GPU počítače, aby se zrychlil výpočet), ale jen pro některé operace (na Windows XP všechny)

[7] Desktop Window Manager; správce oken od verze Windows Vista a novější, který umožňuje použití hardwarové akcelerace
[8] Windows Display Driver Model; je to architektura grafických ovladačů GPU od verze Windows Vista a dále

jako například renderování textu, BitBlt, AlphaBlend, TransparentBlt, StretchBlt. Na Windows Vista žádná akcelerace nebyla – všechno tedy počítal procesor.[9]

3.2 GDI+

Po uvedení operačního systému Windows XP bylo GDI doplněno o GDI+, které dokáže spolupracovat s GDI (předávat si vzájemně DC či bitmapy aj.). Celé API je vytvořeno na základě OOP a je mnohem komplexnější. GDI+ umí bezmála to samé co GDI s tím, že ho doplňuje o nedostatky jako například:

- nezávislost na rozlišení výstupního zařízení
- alpha blending (míchaní barev s alfa kanálem – průhlednost)
- gradienty (přechody)
- podpora (kódování, dekódování) grafických formátů jako PNG, JPEG, TIFF a další
- antialiasing (vyhlazování hran)
- souřadnice s plovoucí desetinnou čárkou

a mnoho dalších nedostatků a vylepšení.[10]

Pro práci s GDI+ je potřeba hlavičkový soubor a knihovna (u GDI je hlavička obsažena již ve Windows.h a knihovna je automaticky linkovaná) – viz kód 2.

```
#include <gdiplus.h>
using namespace Gdiplus;
#pragma comment (lib,"Gdiplus.lib")
```

Kód 2 - GDI+ hlavičkový soubor a linkování[11]

Dále je potřeba inicializovat GDI+ pomocí následujícího kódu, který se musí zavolat ještě před použitím GDI+ objektů – viz kód 3.

[9] Comparing Direct2D and GDI Hardware Acceleration. MICROSOFT. *Windows Desktop Development – Windows Dev Center* [online]. © 2015 [cit. 2015-01-29]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/ff729480%28v=vs.85%29.aspx
[10] New Features. MICROSOFT. *MSDN - Microsoft Developer Network* [online]. © 2015 [cit. 2015-02-01]. Dostupné z: https://msdn.microsoft.com/en-us/library/ms536340%28v=vs.85%29.aspx
[11] zdroj: autor

```
  GdiplusStartupInput gdiplusStartupInput;
  ULONG_PTR gdiplusToken;
  GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);
```

Kód 3 - GDI+ inicializace[12]

Kód 4 zachycuje příklad kreslení ve standardně zachycené zprávě WM_PAINT.

```
  HDC hdc;
  PAINTSTRUCT ps;
  hdc = BeginPaint(hwnd, &ps);
  Graphics graphics(hdc);
  Pen cernePero(Color(255, 0, 0, 0));
  graphics.DrawLine(&cernePero, 0, 0, 200, 100);
  graphics.SetInterpolationMode(InterpolationModeHighQuality);
  Image fotka(L"obrazek.jpg");
  // nakreslení fotky ve dvojnásobné velikosti
  graphics.DrawImage(&fotka, 0, 0, (fotka.GetWidth())*2,
(fotka.GetHeight())*2);
  EndPaint(hwnd, &ps);
```

Kód 4 - základní práce s GDI+[13]

V kódu 4 jsou první 3 řádky stejné jako u GDI. Až od čtvrtého řádku začíná první GDI+ volání – přiřazení HDC do konstruktoru vzniklého objektu graphics, takže jakákoliv metoda objektu graphics dává svůj výstup na právě přiřazený HDC. Poté vytváříme pero a kreslíme s ním v metodě DrawLine na dané souřadnice. Pomocí metody SetInterpolationMode nastavíme nejvyšší kvalitu pro zobrazení obrázku, který bude dvakrát zvětšen oproti své původní velikosti. Načtení obrázku v jiném formátu než bitmapa je díky podpoře mnoha formátů velmi jednoduché a nemusíme vůbec řešit dekódování. Pokud s GDI+ končíme, zavoláme následující funkci – viz kód 5. Ale pozor! Musí se nejdříve smazat všechny objekty patřící GDI+.

```
  GdiplusShutdown(gdiplusToken);
```

Kód 5 - ukončení GDI+[14]

[12] zdroj: autor
[13] zdroj: autor

Bohužel GDI+ nedostalo nikdy hardwarovou akceleraci.

3.3 Direct2D

Direct2D je nové 2D grafické API, které je plně hardwarově akcelerované (běží nad Direct3D). Je objektově orientované. Bylo vytvořeno za účelem poskytnutí vysokého výkonu a vysoké kvality na výstupu. Mělo by nahradit dosavadní GDI a GDI+ s tím, že umožňuje případnou vzájemnou spolupráci. Funguje na „lehkém základě“ COM[15] a používá jej pouze jako konvenci pro správu životnosti objektů apod. – nemusíme tedy v žádném případě inicializovat či registrovat COM jako takové. Celé API se stále rozvíjí a s každou novou verzí Windows přibude i něco do Direct2D. Aktuálně se nachází ve verzi 1.2, která byla uvedena společně s Windows 8.1. Minimální požadavky pro spuštění Direct2D 1.0 na neserverových Windows jsou: Windows 7 nebo Windows Vista se Service Pack 2 a Platform Update pro Windows Vista.[16]

Díky spolupráci s WIC[17] a DirectWrite[18] Direct2D 1.0 obsahuje:

- hardwarovou akceleraci
- nouzové softwarové renderování pomocí CPU
- možnost spolupráce s GDI, GDI+, Direct3D a dalšími
- ClearType renderování textu
- antialiasing geometrických tvarů
- kreslení a výplně geometrických tvarů (úsečky, čtverce, kružnice, křivky atd.) a operace (jejich různé kombinace) mezi sebou
- práci s bitmapami a jinými formáty
- souřadnicový systém nezávislý na zařízení
- jednotlivě barevné, přechodové nebo bitmapové štětce
- renderování do mezivrstev (jako paměťové DC u GDI)

a mnoho dalšího.[19]

---

[14] zdroj: autor
[15] Component Object Model; je to metoda pro sdílení binárního kódu mezi různými aplikacemi a programovacími jazyky
[16] Direct2D. MICROSOFT. *Windows Desktop Development – Windows Dev Center* [online]. © 2015 [cit. 2015-02-07]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/dd370990%28v=vs.85%29.aspx
[17] Windows Imaging Component; je to framework založený na COM pro zpracování obrazových dat a metadat
[18] komplexní API pro zpracování textu

Direct2D používá v souřadnicovém systému speciální jednotky nazvané DIP (Device Independent Pixels), což jsou logické pixely měnící se vzhledem k nastavení DPI[20] ve Windows a tím získává onu nezávislost na zařízení. DIP se vypočítá následovně: fyzický pixel = (DIP × DPI) / 96. Někdy však může vyjít fyzický pixel jako necelé číslo, s čímž Direct2D počítá, a proto používá čísla s plovoucí řádovou čárkou všude, kde to je možné a potřebné.[21]

A nyní k samotné práci s Direct2D 1.0. Nejprve je potřeba mít správné hlavičkové soubory – viz kód 6.

```
#include <d2d1.h>
// obsahuje užitečné pomocné funkce
#include <d2d1helper.h>
#pragma comment(lib, "d2d1")
```

Kód 6 - Direct2D hlavičky a linkování[22]

Dále je dobré si definovat šablonu, která zjednoduší a zabezpečí uvolňování paměti pro jakékoliv Direct2D rozhraní – viz kód 7.

```
template<class Interface>
inline void SafeRelease(Interface **ppInterfaceToRelease)
{
           if (*ppInterfaceToRelease != NULL)
           {
                (*ppInterfaceToRelease)->Release();
                (*ppInterfaceToRelease) = NULL;
           }
}
```

Kód 7 - šablona pro bezpečné uvolňování paměti[23]

---

[19]Tom's Blog: Introducing the Microsoft Direct2D API. OLSEN, Thomas. *TechNet Blogs* [online]. 29 Oct 2008 1:42 PM [cit. 2015-02-04]. Dostupné z: http://blogs.technet.com/b/thomasolsen/archive/2008/10/29/introducing-the-microsoft-direct2d-api.aspx

[20] Dots Per Inch; počet bodů na palec – čím větší číslo (výchozí je 96 DPI), tím více se zvětší písmo globálně; všechny aplikace, které s vyšším DPI počítají, zvětší i svoji grafiku, ovládací prvky atp., aby se do nich vešel zvětšený text

[21] Direct2D and High-DPI. MICROSOFT. *Windows Desktop Development – Windows Dev Center* [online]. © 2015 [cit. 2015-02-07]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/dd756649%28v=vs.85%29.aspx

[22] zdroj: autor

Pro uvolnění paměti pak stačí vždy zadat adresu ukazatele – viz kód 8.

```
  SafeRelease(&pNejakeRozhrani);
```

Kód 8 - uvolnění paměti šablonou[24]

Používání šablony není nutnost (viz kód 9), ale zabezpečení proti běžným chybám, které se pak špatně zjišťují a zbytečně nás zdržují.

```
  pNejakeRozhrani->Release();
  pNejakeRozhrani = NULL;
```

Kód 9 - uvolnění paměti bez šablony[25]

Direct2D má obecně 2 druhy zdrojů – nezávislé na zařízení a závislé na zařízení. Tím je myšleno, kde se zdroje nacházejí. Nezávislé na zařízení jsou v systémové paměti RAM a tyto zdroje stačí vytvořit jednou a používat je po celou životnost aplikace. Závislé na zařízení se nacházejí v GPU paměti (jejich vytváření je časově drahé, proto by se měly udržovat po celou dobu běhu aplikace) a musí se znovu vytvořit pokaždé, když ztratíme kontakt s grafickou kartou (například když uživatel změní rozlišení monitoru či aktualizuje grafický ovladač), protože plocha, na kterou kreslíme a tím pádem i vytvořené bitmapy, štětce atd. se stanou neplatnými a nelze je dále používat.

Teď je potřeba vytvořit tzv. továrnu (factory), která nám poslouží k vytváření různých kreslících ploch (render target) – viz kód 10. Továrna patří mezi zdroje nezávislé na zařízení, proto ji vytvoříme ideálně někde na začátku aplikace (například při vytváření hlavního okna).

```
  // pointer dáme buď globálně, nebo ho předáváme
  ID2D1Factory *pD2DFactory = NULL;
  hr = D2D1CreateFactory(D2D1_FACTORY_TYPE_SINGLE_THREADED,
&pD2DFactory);
```

Kód 10 – tvorba Direct2D továrny[26]

---

[23] Creating a Simple Direct2D Application. MICROSOFT. *Windows Desktop Development – Windows Dev Center* [online]. © 2015 [cit. 2015-02-08]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/dd370994%28v=vs.85%29.aspx

[24] zdroj: autor

[25] zdroj: autor

[26] zdroj: autor

V D2D1CreateFactory funkci první argument značí, zdali bude továrna vytvořena s možností automatické synchronizace, pokud se k ní bude přistupovat z více vláken najednou (D2D1_FACTORY_TYPE_MULTI_THREADED) nebo bez této možnosti (D2D1_FACTORY_TYPE_SINGLE_THREADED) s tím, že je na každém, aby si vytvořil vlastní synchronizaci přístupu, pokud je potřeba.

V Direct2D kontrolujeme úspěšnost metody, funkce atd., pokud vrací tzv. HRESULT (není to handle, ale chybový kód) – viz kód 11.

```
  hr = D2D1CreateFactory(D2D1_FACTORY_TYPE_SINGLE_THREADED,
&pD2DFactory);
  // SUCCEEDED je obecné makro na zjištění úspěšnosti
  if (SUCCEEDED(hr))
  {
        // úspěšné provedení, pokračujeme dále
  }
```

Kód 11 - kontrola úspěšnosti[27]

Jakmile máme úspěšně vytvořenou továrnu, lze z ní získat například informace o aktuálně nastaveném DPI ve Windows – viz kód 12.

```
  FLOAT dpiX, dpiY;
  pD2DFactory->GetDesktopDpi(&dpiX, &dpiY);
```

Kód 12 - zjištění DPI[28]

Nebo ji použijeme na vytvoření toho nejdůležitějšího, a sice kreslící plochy. Existuje několik druhů těchto ploch (pro GDI a GDI+, Direct3D, paměťová, WIC atd.). Pozor však na to, že jsou závislé na zařízení. Pokud je potřeba spolupráce s GDI a chceme Direct2D obsah vykreslit do DC GDI, GDI+, pak lze využít CreateDCRenderTarget. Nebo naopak GDI, GDI+ obsah vykreslit do Direct2D, využijeme ID2D1GdiInteropRenderTarget. Ale nejtypičtější plochou je kreslení přímo do okna aplikace (pracuje s HWND, což je handle k oknu aplikace) – viz kód 13.

[27] zdroj: autor
[28] zdroj: autor

```
  // pointer na plochu vytvoříme jen jednou (jako továrnu)
  ID2D1HwndRenderTarget *pHWNDRT = NULL;
  pD2DFactory->CreateHwndRenderTarget(param,
D2D1::HwndRenderTargetProperties(hwnd, size,
D2D1_PRESENT_OPTIONS_NONE), &pHWNDRT);
```

Kód 13 - tvorba kreslící plochy[29]

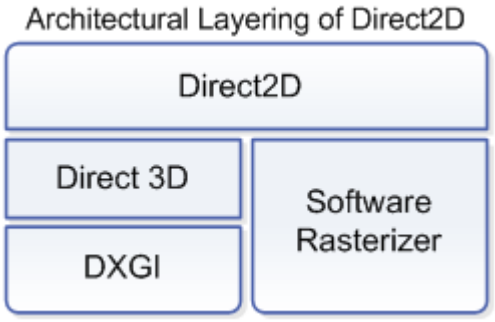
Argument param v sobě zahrnuje následující strukturu s vlastnostmi – viz kód 14.

```
  D2D1_RENDER_TARGET_PROPERTIES param =
D2D1::RenderTargetProperties(D2D1_RENDER_TARGET_TYPE_DEFAULT,
D2D1::PixelFormat(DXGI_FORMAT_B8G8R8A8_UNORM,
D2D1_ALPHA_MODE_IGNORE), 0, 0, D2D1_RENDER_TARGET_USAGE_NONE,
D2D1_FEATURE_LEVEL_DEFAULT);
```

Kód 14 - struktura D2D1_RENDER_TARGET_PROPERTIES[30]

D2D1_RENDER_TARGET_TYPE_DEFAULT znamená, že Direct2D automaticky bude používat hardwarové renderování, pokud je dostupné, jinak použije softwarové (lze definovat i konkrétně jako D2D1_RENDER_TARGET_TYPE_SOFTWARE nebo D2D1_RENDER_TARGET_TYPE_HARDWARE). Softwarové renderování znamená, že nebude použita hardwarová akcelerace na GPU, ale vše bude počítáno na CPU. Direct2D má kvalitní softwarový rasterizér, který je rychlejší než GDI+ se srovnatelnou kvalitou. Hardwarové renderování znamená, že bude použita akcelerace přes GPU, ale jen v případě přítomnosti WDDM ovladače (nezávisle na DWM).[31] Obrázek 1 zobrazuje vrstvy Direct2D.

[29] zdroj: autor
[30] zdroj: autor
[31] Comparing Direct2D and GDI Hardware Acceleration. MICROSOFT. *Windows Desktop Development – Windows Dev Center* [online]. © 2015 [cit. 2015-01-29]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/ff729480%28v=vs.85%29.aspx



Obrázek 1 - vrstvy Direct2D[32]

Dále lze struktuře nastavit formát pixelů, DPI (obě nuly znamenají, že se použije hodnota z továrny), použití plochy (například pro GDI spolupráci) a hlavně tzv. úrovně vlastností (feature level). Úrovně vlastností byly představeny poprvé v Direct3D 10.1 (Direct2D se ve verzi 1.0 váže právě na Direct3D 10.1 a ve verzi 1.1 se váže na Direct3D 11.1 a dále) a slouží ke zpětné kompatibilitě se staršími GPU. Ve verzi 10.1 je pouze možnost podporovat zpětně Direct3D 10 GPU (D2D1_FEATURE_LEVEL_10). Ale na Windows 7, kde je přítomno DirectX 11, lze podporovat i starší Direct3D 9 GPU (D2D1_FEATURE_LEVEL_9). Jestliže není přítomen správný hardware nebo ovladač, pak Direct2D poběží přes WARP[33].

V CreateHwndRenderTarget kromě param (výše popsané struktury) je ještě size a D2D1_PRESENT_OPTIONS_NONE. Size je velikost plochy, na kterou budeme kreslit a jelikož chceme výstup do okna aplikace, lze zjistit tuto velikost následovně (pokud se zadá velikost větší nebo menší než je okno, do kterého budeme kreslit, pak se automaticky zmenší či zvětší na velikost okna a může způsobit degradaci výstupu) – viz kód 15.

[32] About Direct2D. MICROSOFT. *Windows Desktop Development – Windows Dev Center* [online]. © 2015 [cit. 2015-02-08]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/dd370987%28v=vs.85%29.aspx

[33] Windows Advanced Rasterization Platform; od Direct3D 11 a dále, dokáže kompletně nahradit GPU a veškeré výpočty dělá na CPU

```
  RECT rc;
  // GDI funkce na zjištění velikosti klientské části okna
  GetClientRect(hwnd, &rc);
  // naplnění size šířkou a výškou
  D2D1_SIZE_U size = D2D1::SizeU(rc.right - rc.left, rc.bottom -
rc.top);
```

Kód 15 - zjištění velikosti klientské části okna[34]

D2D1_PRESENT_OPTIONS_NONE znamená čekání na obnovovací frekvenci monitoru. Pokud ale chceme výsledek okamžitě, lze použít D2D1_PRESENT_OPTIONS_IMMEDIATELY.

Jakmile máme volání metody CreateHwndRenderTarget úspěšné, můžeme začít vytvářet štětce, bitmapy, geometrické tvary atd. – viz kód 16.

```
  ID2D1SolidColorBrush *pBlackBrush = NULL;
  // tvorba černého štětce
  pHWNDRT->CreateSolidColorBrush(
D2D1::ColorF(D2D1::ColorF::Black), &pBlackBrush);
  pHWNDRT->BeginDraw();
  pHWNDRT->SetAntialiasMode(D2D1_ANTIALIAS_MODE_ALIASED);
  pHWNDRT->SetTransform(D2D1::Matrix3x2F::Identity());
  pHWNDRT->Clear(D2D1::ColorF(D2D1::ColorF::White));
  pHWNDRT->DrawLine(D2D1::Point2F(10.0f, 50.0f),
D2D1::Point2F(10.0f, 250.0f), pBlackBrush, 3.0);
  pHWNDRT->DrawEllipse(D2D1::Ellipse(D2D1::Point2F(150.0f,
150.0f), 60.0f, 60.0f), pBlackBrush, 3.0);
  m_pHWNDRT->EndDraw();
```

Kód 16 - základní práce s Direct2D[35]

Veškeré kreslící operace musí být uvnitř BeginDraw a EndDraw, jinak se nevykonají. Ostatní operace jako SetAntialiasMode nemusí být uvnitř. BeginDraw umožňuje dávkování příkazů (ty se nevykonávají ihned, ale dávkují se po více z důvodu efektivity) a příkazy se vykonají až ve chvíli, kdy se zavolá EndDraw, Flush nebo se zaplní buffer

[34] zdroj: autor
[35] zdroj: autor

příkazů. D2D1_ANTIALIAS_MODE_ALIASED znamená, že veškeré tvary jako úsečky, kruhy atd. se vykreslí pixelovitě nevyhlazené, proto existuje D2D1_ANTIALIAS_MODE_PER_PRIMITIVE, který vyhlazuje veškeré tvary. SetTransform s nastavením D2D1::Matrix3x2F::Identity() vymaže veškeré předchozí transformace (rotace, zkosení aj.), jelikož nastavení transformace trvá stále, dokud se nezmění. Clear vyčistí celou plochu – zde bílou barvou. DrawLine kreslí úsečku a DrawEllipse kružnici na dané souřadnice s určeným štětcem a sílou tahu.

Protože metody CreateHwndRenderTarget a CreateSolidColorBrush vytvářejí zdroje závislé na zařízení (v případě potřeby je nutno zavolat tyto metody znovu), dává se typicky tento kód a stejně tak i celý kód nacházející se mezi BeginDraw a EndDraw, pokud je ve standardně zachycené zprávě WM_PAINT, mezi BeginPaint a EndPaint (známé z GDI) případně validujeme sami na konci pomocí funkcí jako ValidateRect atp. Jedna z možností, jak to může vypadat uvnitř WM_PAINT, je vidět v kódu 17.

```
  hdc = BeginPaint(hwnd, &ps);
  // kontrola jestli existují plochy, štětce atd., případně je
znovu vytvoříme
  pHWNDRT->BeginDraw();
  // zde kreslíme do plochy pomocí štětců atd.
  pHWNDRT->EndDraw();
  // kontrola jestli máme kontakt se zařízením a vše proběhlo
v pořádku; pokud ne, smažeme všechny zdroje závislé na zařízení
  EndPaint(hwnd, &ps);
```

Kód 17 - struktura Direct2D ve WM_PAINT[36]

Jestli je zařízení dostupné či ne nám říká vrácená chyba z metody EndDraw nazvaná D2DERR_RECREATE_TARGET. Příklad kontroly této hodnoty je v kódu 18.

[36] zdroj: autor

```
hr = pHWNDRT->EndDraw();
if (hr == D2DERR_RECREATE_TARGET)
{
	hr = S_OK;
	SafeRelease(&pHWNDRT);
	SafeRelease(&pBlackBrush);
}
```

Kód 18 - kontrola chybového kódu[37]

Pokud EndDraw vrátí D2DERR_RECREATE_TARGET, změníme proměnnou zpět na bezchybovou hodnotu a všechny zdroje závislé na zařízení vymažeme. Na začátku při kontrole zdrojů zase všechny znovu vytvoříme.

Při vytváření CreateHwndRenderTarget, jak už bylo výše zmíněno, je potřeba zadat velikost plochy, na kterou se bude kreslit. Ale co se stane v případě, že uživatel změní velikost okna aplikace nebo zadáme menší či větší velikost plochy než je klientská část okna? Samozřejmě se celá plocha přepočítá na danou velikost a tím vznikne velmi nehezký výstup jako na obrázku 2.

Obrázek 2 - změna velikosti okna při konstantní velikosti kreslící plochy[38]

Abychom tomuto předešli, musíme reagovat na zprávu WM_SIZE, kterou dostáváme při změně velikosti okna aplikace nebo měnit velikost plochy vždy před kreslením. Pro změnu velikosti kreslící plochy využijeme metodu Resize – viz kód 19.

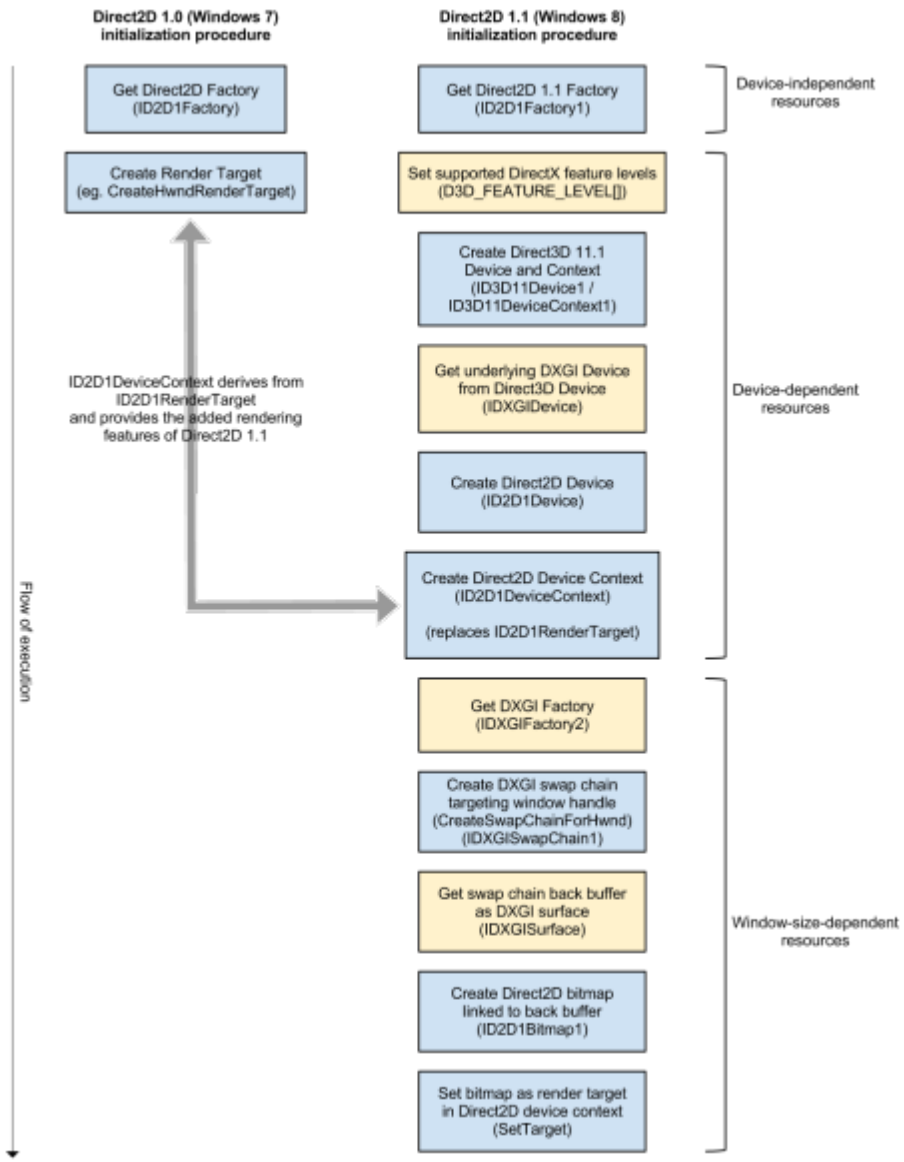
[37] zdroj: autor
[38] zdroj: vlastní aplikace

```
case WM_SIZE:
{
       if (pHWNDRT)
       {
            D2D1_SIZE_U size;
            size.width = LOWORD(lParam);
            size.height = HIWORD(lParam);
            pHWNDRT->Resize(size);
       }
}
```

Kód 19 - dynamická změna velikosti plochy

3.3.1 Direct2D 1.1

Direct2D 1.1 funguje nad Direct3D 11.1. V současné době je Direct2D ve verzi 1.2, avšak verze 1.1, která na první pohled vypadá jako menší aktualizace s novými možnostmi, vnáší do API jednu zásadní změnu. Tou změnou je začátek tvorby plochy, na kterou chceme kreslit. Samozřejmě nová hlavička (D2d1_1.h nebo D2d1_2.h), obsahující i nové funkce, je zpětně kompatibilní. To znamená, že pokud ve stávajícím zdrojovém kódu vyměníme všechny hlavičky za nové, půjde bez problému zkompilovat a bude fungovat bez rozdílu. Ale abychom zprovoznili nové funkce, je potřeba vytvořit plochu novým způsobem – viz obrázek 3.



Obrázek 3 - úvodní procedura pro práci s Direct2D 1.0 v porovnání s Direct2D 1.1[39]

Původní inicializační procedura byla v celku krátká a veškerá práce „navíc“ byla za nás udělána automaticky. Nyní jsme se ale se složitostí a také úrovní přiblížili k Direct3D. V podstatě je třeba udělat toto:

- inicializovat Direct3D
- nastavit Direct3D jako kreslící plochu okna aplikace
- získat Direct3D back buffer[40]

[39] KATY. Direct2D 1.1 Migration Guide for Windows 7 Developers. *Katy's Code* [online]. January 23, 2013 [cit. 2015-02-10]. Dostupné z: https://katyscode.wordpress.com/2013/01/23/migrating-existing-direct2d-applications-to-use-direct2d-1-1-functionality-in-windows-7

- nastavit tento buffer jako kreslící plochu Direct2D

Direct2D tedy úzce spolupracuje s Direct3D a postupně se dostává k pojmu swap chain, který je rutinou pro programátory Direct3D a neznamená nic jiného než prohození bufferů – na jeden se aktuálně kreslí a druhý se zobrazuje v naší aplikaci (bufferů může být i více, nejčastěji však dva). Po dokončení celé inicializační procedury získáme rozhraní ID2D1DeviceContext, které podporuje nové metody, ale zároveň i umí původní jako DrawEllipse, DrawBitmap, Clear, CreateSolidColorBrush atd., na které jsme již zvyklí z dřívějších rozhraní jako ID2D1HwndRenderTarget. Po dokončení kreslení, tj. zavolání metody EndDraw, se nyní nezobrazí nic v aplikaci, ale musí se ještě zavolat metoda Present1 patřící rozhraní IDXGISwapChain1.[41]

Po této vyčerpávající inicializaci je možné nyní využít nové funkce jako například:

- efekty (velké množství různých efektů jako rozmazání, změna jasu, saturace, stíny, transformace, míchání, histogramy, filtry atd., které lze použít na bitmapy, geometrické tvary, text z DirectWrite nebo dokonce i na Direct3D scény)[42]
- štětec s barvou bitmapy
- geometrické realizace (pokud chceme kreslit opakovaně stejné geometrické prvky, tak ušetříme výkon i paměť tím, že je změníme na tyto realizace)[43]
- nové algoritmy pro interpolaci obrazu
- metoda SetUnitMode pro změnu z DIP na pixely

Nové funkce, které potřebují hlavičku D2d1_2.h, jsou podporovány pouze na Windows 8.1, kdežto minimální požadavky pro D2d1_1.h jsou Windows 8 nebo Windows 7 s platform update.

---

[40] prostor v GPU paměti, kam se postupně dávají výsledky renderování a následně jako celek se vykreslí do našeho okna

[41] KATY. Direct2D 1.1 Migration Guide for Windows 7 Developers. *Katy's Code* [online]. January 23, 2013 [cit. 2015-02-10]. Dostupné z: https://katyscode.wordpress.com/2013/01/23/migrating-existing-direct2d-applications-to-use-direct2d-1-1-functionality-in-windows-7/

[42] Effects. MICROSOFT. *Windows Desktop Development – Windows Dev Center* [online]. © 2015 [cit. 2015-02-08]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/hh973240%28v=vs.85%29.aspx

[43] Improving the performance of Direct2D apps: Per-primitive caching using geometry realizations. MICROSOFT. *Windows Desktop Development – Windows Dev Center* [online]. © 2015 [cit. 2015-02-08]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/dd372260%28v=vs.85%29.aspx

3.3.2 Interoperabilita mezi GDI/GDI+ a Direct2D

V případě, kdy je potřeba spolupráce s GDI – tj. potřebujeme Direct2D obsah vykreslit do DC GDI, pak je tu možnost využít metodu CreateDCRenderTarget. Tvorba továrny a parametrů je stejné. Změní se pouze typ plochy – viz kód 20.

```
  // zde předchází tvorba továrny
  ID2D1DCRenderTarget *pDCRT = NULL;
  // toto je uvnitř WM_PAINT
  PAINTSTRUCT ps;
  HDC hdc = BeginPaint(hwnd, &ps);
  HRESULT hr = S_OK;
  // pokud plocha neexistuje, vytvoříme ji včetně jiných zdrojů
  if (!pDCRT)
  {
        // zde předchází nastavení parametrů (param)
        // tvorba plochy, která má výstup na DC GDI
        hr = pD2DFactory->CreateDCRenderTarget(&param, &pDCRT);
        if (SUCCEEDED(hr))
        {
            // vytvoření černého štětce
            hr = pDCRT->CreateSolidColorBrush(
D2D1::ColorF(D2D1::ColorF::Black), &pBlackBrush);
        }
  }
```

Kód 20 - Direct2D výstup na DC GDI[44]

Pokud vše bylo úspěšné, přiřadíme DC jako plochu a kreslíme do něj – viz kód 21.

[44] zdroj: autor

```
  if (SUCCEEDED(hr))
  {
        RECT rc;
        GetClientRect(hwnd, &rc);
        hr = pDCRT->BindDC(hdc, &rc);
        pDCRT->BeginDraw();
        pDCRT->Clear(D2D1::ColorF(D2D1::ColorF::White));
        pDCRT->DrawEllipse(D2D1::Ellipse(D2D1::Point2F(150.0f,
150.0f), 100.0f, 100.0f), pBlackBrush, 3.0);
        hr = pDCRT->EndDraw();
  }
```

Kód 21 - kreslení z Direct2D do DC GDI[45]

A nakonec vyčištění alokované paměti a validace – viz kód 22.

```
 SafeRelease(&pBlackBrush);
 SafeRelease(&pDCRT);
 EndPaint(hwnd, &ps);
```

Kód 22 - vyčištění paměti a validace[46]

Hlavní odlišností je svázání se s DC od GDI tj. metoda BindDC, jinak je kód prakticky totožný se standardním kreslením. Samozřejmostí je hardwarová akcelerace (je-li dostupná) a stejná kvalita kreslení Direct2D (například vyhlazené hrany) i přesto, že má výstup na kontext GDI.

Pokud však chceme naopak GDI, GDI+ obsah vykreslit do Direct2D plochy, využijeme k tomu ID2D1GdiInteropRenderTarget. Tvorba továrny je opět stejná, ale nyní je potřeba vytvořit 2 typy ploch – viz kód 23.

[45] zdroj: autor
[46] zdroj: autor

```
  // zde předchází tvorba továrny (pD2DFactory)
  ID2D1GdiInteropRenderTarget *pGDIRT;
  ID2D1HwndRenderTarget *pHWNDRT;
  // toto je uvnitř WM_PAINT
  HDC hD2Ddc = NULL;
  PAINTSTRUCT ps;
  HDC hdc = BeginPaint(hwnd, &ps);
  HRESULT hr = S_OK;
```

Kód 23 - příprava proměnných[47]

První plocha je klasická (přímo do okna aplikace) s tím, že nastavení parametrů se liší ve zvolení D2D1_RENDER_TARGET_USAGE_GDI_COMPATIBLE místo D2D1_RENDER_TARGET_USAGE_NONE – viz kód 24.

```
  D2D1_RENDER_TARGET_PROPERTIES param =
D2D1::RenderTargetProperties(D2D1_RENDER_TARGET_TYPE_DEFAULT,
D2D1::PixelFormat(DXGI_FORMAT_B8G8R8A8_UNORM,
D2D1_ALPHA_MODE_IGNORE), 0, 0,
D2D1_RENDER_TARGET_USAGE_GDI_COMPATIBLE,
D2D1_FEATURE_LEVEL_DEFAULT);
```

Kód 24 - nastavení parametrů[48]

V kódu 25 tuto plochu vytvoříme.

```
  // pokud plocha neexistuje, vytvoříme ji včetně jiných zdrojů
  if (!pHWNDRT)
  {
        // zde předchází nastavení parametrů (param) a velikosti
plochy (size)
        pD2DFactory->CreateHwndRenderTarget(param,
D2D1::HwndRenderTargetProperties(hwnd, size,
D2D1_PRESENT_OPTIONS_NONE), &pHWNDRT);
```

Kód 25 - vytvoření plochy[49]

[47] zdroj: autor
[48] zdroj: autor

A druhá plocha (typu ID2D1GdiInteropRenderTarget), kterou lze nyní vytvořit na základě již existující (pomocí metody QueryInterface - nelze ji tedy přímo zavolat jako například CreateDCRenderTarget), umožní udělat DC odkazující na Direct2D plochu – viz kód 26.

```
        if (SUCCEEDED(hr))
        {
            // získáme správné rozhraní pro GDI
            pHWNDRT->QueryInterface(__uuidof(
ID2D1GdiInteropRenderTarget), (void**)&pGDIRT);
            pHWNDRT->BeginDraw();
            pHWNDRT->Clear(D2D1::ColorF(D2D1::ColorF::White));
            // přiřazení DC k ploše kompatibilní s GDI
            hr = pGDIRT->GetDC(D2D1_DC_INITIALIZE_MODE_COPY,
&hD2Ddc);
        }
  }
```

Kód 26 - získání GDI kompatibilní plochy[50]

Od této chvíle už není problém kreslit z GDI do Direct2D, jak je vidět v kódu 27. Jen pozor, že metody na získání a smazání DC musí být mezi BeginDraw a EndDraw!

```
  if (SUCCEEDED(hr))
  {
        // zde pracujeme standardně s GDI (SelectObject,
DeleteObject, kreslení, text atd.) s použitím hD2Ddc
        Ellipse(hD2Ddc, 300, 50, 500, 250);
        pGDIRT->ReleaseDC(NULL);
        pHWNDRT->EndDraw();
        SafeRelease(&pHWNDRT);
        SafeRelease(&pGDIRT);
  }
  EndPaint(hwnd, &ps);
```

Kód 27 - kreslení z GDI do Direct2D[51]

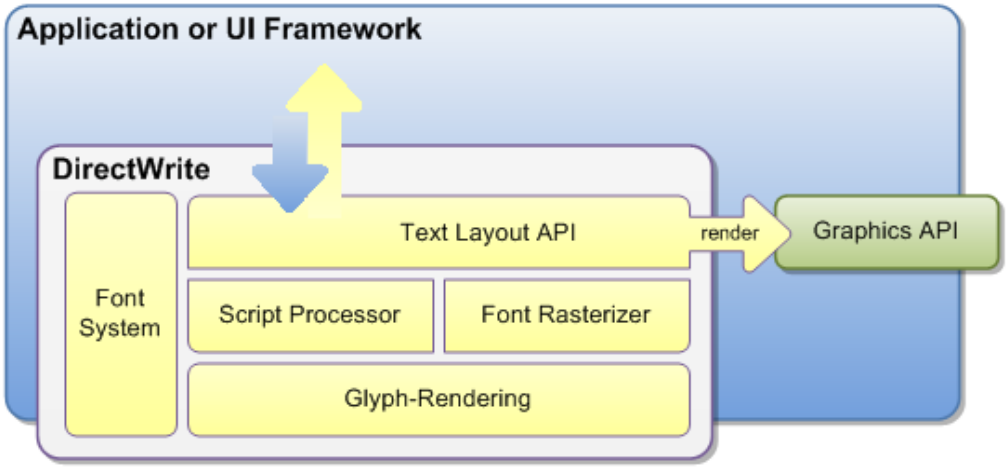
[49] zdroj: autor
[50] zdroj: autor

3.4 DirectWrite

Pokud bychom hledali nějakou funkci na psaní textu v Direct2D, hledání by bylo marné. Direct2D totiž přímo text zpracovávat neumí. Potřebuje k tomu využít spolupráce s API zvané DirectWrite. DirectWrite se specializuje především na rozložení a renderování textu a glyfů[52], ale dokáže toho mnohem více – například multiformátový text či testování pozice při najetí myší na daný úsek textu. Minimální požadavky na spuštění jsou stejné jako u Direct2D 1.0. Použitím spolu s Direct2D je hardwarově akcelerovaný. Na první pohled by měl nahradit GDI/GDI+ s Uniscribe, který sloužil hlavně na práci s mezinárodními jazyky, unikódem (unicode) a jejich rozložení.[53] Ovšem DirectWrite umí spolupracovat i s GDI a renderovat do jeho DC (vytvoří bitmapu s textem a tu si GDI zkopíruje pomocí BitBlt funkce) nebo naopak používat jeho fonty (s omezením, že vezme od GDI jen některé informace – tučnost, font, styl a vynechá velikost písma, podškrtnutí, přeškrtnutí atp.).[54] Na rozdíl od GDI vylepšuje ClearType[55] kvalitu písma o subpixelové pozicování a vyhlazování písma (antialiasing), které je nejen horizontální, ale i vertikální.[56] Na obrázku 4 je možné vidět, jak DirectWrite komunikuje s okolím a z čeho se skládá.

---

[51] zdroj: autor
[52] glyf je grafická reprezentace číslice, znaku, interpunkčního znaménka atd.
[53] Using Uniscribe. MICROSOFT. *Windows Desktop Development – Windows Dev Center* [online]. © 2015 [cit. 2015-02-15]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/dd374127%28v=vs.85%29.aspx
[54] Interoperating with GDI. MICROSOFT. *Windows Desktop Development – Windows Dev Center* [online]. © 2015 [cit. 2015-02-15]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/dd742734%28v=vs.85%29.aspx
[55] je to technologie pro vylepšení zobrazení textu zejména na LCD monitorech
[56] DirectWrite. MICROSOFT. *Windows Desktop Development – Windows Dev Center* [online]. © 2015 [cit. 2015-02-15]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/dd368038%28v=vs.85%29.aspx



Obrázek 4 - vrstvy DirectWrite[57]

Následuje ukázka základní práce s DirectWrite. Nejdříve je potřeba zahrnout hlavičku a knihovnu – viz kód 28.

```
#include <dwrite.h>
#pragma comment (lib,"dwrite.lib")
```

Kód 28 - DirectWrite hlavička a knihovna[58]

Poté si inicializujeme proměnné týkající se textu – viz kód 29.

```
const wchar_t text[] = L"Úvod do DirectWrite";
const wchar_t font[] = L"Arial";
const float velikost = 25;
```

Kód 29 - příprava textu, fontu a velikosti[59]

Následně v kódu 30 vytvoříme továrnu.

[57] Introducing DirectWrite. MICROSOFT. *Windows Desktop Development – Windows Dev Center* [online]. © 2015 [cit. 2015-02-08]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/dd371554%28v=vs.85%29.aspx
[58] zdroj: autor
[59] zdroj: autor

```
  HRESULT hr;
  IDWriteFactory *pDWriteFactory = NULL;
  hr = DWriteCreateFactory(DWRITE_FACTORY_TYPE_ISOLATED,
__uuidof(pDWriteFactory),
reinterpret_cast<IUnknown**>(&pDWriteFactory));
```

Kód 30 - tvorba továrny v DirectWrite[60]

Po továrně vytvoříme formát textu, ve kterém nastavíme tloušťku, styl, velikost atd. – viz kód 31.

```
  IDWriteTextFormat *pTextFormat = NULL;
  if (SUCCEEDED(hr))
  {
        hr = pDWriteFactory->CreateTextFormat(font, NULL,
DWRITE_FONT_WEIGHT_NORMAL, DWRITE_FONT_STYLE_NORMAL,
DWRITE_FONT_STRETCH_NORMAL, velikost, L"", &pTextFormat);
  }
```

Kód 31 – formát textu[61]

Jakmile máme vytvořený formát textu, dají se například volat metody SetTextAlignment a SetParagraphAlignment. První nastavuje pozici textu na výstupu horizontálně a druhý vertikálně. V kódu 32 je zachyceno vykreslení textu pomocí Direct2D.

```
  // zde předchází typická tvorba Direct2D továrny (pD2DFactory),
plochy (pHWNDRT) a štětce (pBlackBrush)
  pHWNDRT->SetTextAntialiasMode(D2D1_TEXT_ANTIALIAS_MODE_ALIASED);
  pHWNDRT->BeginDraw();
  pHWNDRT->Clear(D2D1::ColorF(D2D1::ColorF::White));
  // vykreslíme text
  pHWNDRT->DrawText(text, ARRAYSIZE(text) - 1, pTextFormat,
D2D1::RectF(0, 0, 1000, 180), pBlackBrush);
  hr = pHWNDRT->EndDraw();
```

Kód 32 - vykreslení DirectWrite textu[62]

[60] zdroj: autor
[61] zdroj: autor

SetTextAntialiasMode nám umožňuje měnit kvalitu výstupu textu, kdy nejkvalitnější nastavení je D2D1_TEXT_ANTIALIAS_MODE_CLEARTYPE a nejméně kvalitní je D2D1_TEXT_ANTIALIAS_MODE_ALIASED. V DrawText metodě určujeme kromě jiného obdélník, do kterého se vykreslí text. Pokud je na šířku menší, než by se vešel text, pak se postupně skládají slova (poté písmena) na druhý řádek. To lze ovlivnit metodou SetWordWrapping. Nakonec vyčistíme paměť – viz kód 33.

```
SafeRelease(&pHWNDRT);
SafeRelease(&pBlackBrush);
SafeRelease(&pTextFormat);
SafeRelease(&pDWriteFactory);
```

Kód 33 - vyčištění paměti[63]

3.5 Windows Imaging Component

Kromě spolupráce s GDI, DirectWrite a Direct3D umí Direct2D také spolupracovat s WIC. WIC je komplexní API založené na COM pro práci s obrazovými daty, čímž je například dekódování a kódování obrazových formátů (mezi něž patří například BMP, GIF, JPEG, PNG, TIFF), zpracování metadat, podpora velké škály formátu pixelů (i širokého gamutu[64]) a hloubky barev (až 48 bitů na pixel), lze implementovat i vlastní dekodéry/kodéry.[65]

Následuje ukázka základní práce s WIC – načtení TIFF obrázku, jeho dekódování na bitmapu a následné vykreslení pomocí Direct2D. Pro použití WIC je třeba hlavička – viz kód 34.

```
#include <wincodec.h>
```

Kód 34 - hlavička WIC[66]

Dále je potřeba inicializovat COM a získat WIC továrnu – viz kód 35.

---

[62] zdroj: autor
[63] zdroj: autor
[64] gamut je dosažitelná oblast barev v určitém barevném prostoru
[65] WIC API Overview. MICROSOFT. *Windows Desktop Development – Windows Dev Center* [online]. © 2015 [cit. 2015-02-08]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/ee719655%28v=vs.85%29.aspx
[66] zdroj: autor

```
  // zde předchází typická tvorba Direct2D továrny (pD2DFactory) a
HWND plochy (pHWNDRT)
  // inicializace COM
  CoInitialize(NULL);
  // ukazatel na továrnu WIC
  IWICImagingFactory *pWICFactory = NULL;
  // vytvoření WIC továrny
  HRESULT hr = CoCreateInstance(CLSID_WICImagingFactory, NULL,
CLSCTX_INPROC_SERVER, IID_PPV_ARGS(&pWICFactory));
```

Kód 35 - inicializace COM a tvorba WIC továrny[67]

V kódu 36 načteme obrázek ze souboru a dekódujeme ho na bitmapu.

```
  IWICBitmapDecoder *pDecoder = NULL;
  if (SUCCEEDED(hr))
  {
        hr = pWICFactory->CreateDecoderFromFilename(
L"obrazek.tif", NULL, GENERIC_READ, WICDecodeMetadataCacheOnLoad,
&pDecoder);
  }
```

Kód 36 - načtení a dekódování obrázku[68]

V kódu 37 načteme první snímek (například GIF, který podporuje animaci, může obsahovat i více snímků).

```
  IWICBitmapFrameDecode *pFrame = NULL;
  if (SUCCEEDED(hr))
  {
        // první snímek = 0 (nultý index)
        hr = pDecoder->GetFrame(0, &pFrame);
  }
```

Kód 37 - načtení snímku bitmapy

[67] zdroj: autor
[68] zdroj: autor

V kódu 38 vytvoříme konvertor na změnu formátu pixelů a v kódu 39 provedeme konverzi na správný formát, protože Direct2D podporuje jen určité formáty, se kterými umí pracovat.

```
  IWICFormatConverter *pConverter = NULL;
  if (SUCCEEDED(hr))
  {
        hr = pWICFactory->CreateFormatConverter(&pConverter);
  }
```

Kód 38 – tvorba konvertoru na změnu formátu pixelů[69]

```
  if (SUCCEEDED(hr))
  {
        hr = pConverter->Initialize(pFrame,
GUID_WICPixelFormat32bppBGR, WICBitmapDitherTypeNone, NULL, 0.0f,
WICBitmapPaletteTypeMedianCut);
  }
```

Kód 39 - použití konvertoru a provedení konverze[70]

Při konverzi z formátu WIC do Direct2D bitmapy, jak je výše napsáno, lze specifikovat různý formát pixelů včetně toho, jestli se má nebo nemá podporovat průhlednost (alfa kanál). Obojí závisí na nastavení parametrů v CreateHwndRenderTarget a WIC konvertoru. Různé kombinace jsou vidět na obrázku 5.

WIC format	Corresponding DXGI format	Corresponding alpha mode
GUID_WICPixelFormat8bppAlpha	DXGI_FORMAT_A8_UNORM	D2D1_ALPHA_MODE_STRAIGHT or D2D1_ALPHA_MODE_PREMULTIPLIED
GUID_WICPixelFormat32bppPRGBA	DXGI_FORMAT_R8G8B8A8_UNORM	D2D1_ALPHA_MODE_PREMULTIPLIED or D2D1_ALPHA_MODE_IGNORE
GUID_WICPixelFormat32bppBGR	DXGI_FORMAT_B8G8R8A8_UNORM	D2D1_ALPHA_MODE_IGNORE
GUID_WICPixelFormat32bppPBGRA	DXGI_FORMAT_B8G8R8A8_UNORM	D2D1_ALPHA_MODE_PREMULTIPLIED

Obrázek 5 - WIC formáty pixelů a korespondující nastavení v Direct2D[71]

[69] zdroj: autor
[70] zdroj: autor

A v tuto chvíli již můžeme vytvořit Direct2D bitmapu z WIC bitmapy – viz kód 40.

```
  ID2D1Bitmap *pBitmap = NULL;
  if (SUCCEEDED(hr))
  {
        hr = pHWNDRT->CreateBitmapFromWicBitmap(pConverter, NULL,
&pBitmap);
  }
```

Kód 40 - Direct2D bitmapa z WIC bitmapy[72]

Po vytvoření Direct2D bitmapy z WIC smažeme nepotřebné zdroje – viz kód 41.

```
  SafeRelease(&pDecoder);
  SafeRelease(&pFrame);
  SafeRelease(&pConverter);
  SafeRelease(&pWICFactory);
```

Kód 41 - smazání WIC zdrojů[73]

A poté můžeme standardně s bitmapou v Direct2D pracovat – v našem případě ji vykreslit, jak ukazuje kód 42.

[71] Supported Pixel Formats and Alpha Modes. MICROSOFT. *Windows Desktop Development – Windows Dev Center* [online]. © 2015 [cit. 2015-02-08]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/dd756766%28v=vs.85%29.aspx
[72] zdroj: autor
[73] zdroj: autor

```
  if (SUCCEEDED(hr))
  {
        pHWNDRT->BeginDraw();
        pHWNDRT->Clear(D2D1::ColorF(D2D1::ColorF::White));
        // zjistíme šířku a výšku bitmapy
        D2D1_SIZE_F size = pBitmap->GetSize();
        // určíme si pozici levého rohu bitmapy na výstupu
        D2D1_POINT_2F levyHorniRoh = D2D1::Point2F(100.f, 50.f);
        pHWNDRT->DrawBitmap(pBitmap, D2D1::RectF(levyHorniRoh.x,
levyHorniRoh.y, levyHorniRoh.x + size.width, levyHorniRoh.y +
size.height));
        hr = pHWNDRT->EndDraw();
        SafeRelease(&pBitmap);
        SafeRelease(&pHWNDRT);
  }
```

Kód 42 - vykreslení bitmapy[74]

V metodě DrawBitmap lze specifikovat i způsob interpolace[75]. Ve výchozím nastavení se použije metoda D2D1_BITMAP_INTERPOLATION_MODE_NEAREST_NEIGHBOR, která podává horší výsledky (pixelovité), ale je rychlejší, kdežto D2D1_BITMAP_INTERPOLATION_MODE_LINEAR je kvalinější za cenu nižší rychlosti. Celý WIC kód a stejně tak Direct2D plochu stačí zpracovat jen jednou, dokud nenastane chyba. Avšak oproti Direct2D zdrojům, které jsou většinou v GPU paměti, má WIC zdroje v systémové RAM a tudíž po dobu aplikace je lze mít někde „vedle“ uložené k dispozici, abychom ušetřili výkon a jen předávali do Direct2D (GPU paměti) zkonvertované bitmapy (CreateBitmapFromWicBitmap).

[74] zdroj: autor
[75] pokud se vykreslí obrázek větší nebo menší než je jeho původní velikost, pak se musí použít určitý algoritmus, který buď dopočítává chybějící pixely, nebo naopak ubírá a na něm pak záleží, jak kvalitní je výsledek

4 Vlastní práce

Dílčím cílem této práce bylo vytvoření jedné aplikace, která by v sobě zahrnovala GDI, GDI+ i Direct2D a co nejobjektivněji je porovnala (funkce, které mají společné) mezi sebou vzhledem k rychlosti vykonání jednotlivých funkcí se zaměřením více na Direct2D. Tato aplikace byla vytvořena v programovacím jazyce C++ a ve vývojovém prostředí Visual Studio 2013 od Microsoftu jako 32bitová, protože by měla podporovat co největší škálu konfigurací počítačů. Jelikož testované počítače běží na Windows 7 (Vista se příliš nerozšířila a uživatelé většinou přešli na další verzi) na platformě x86 i x86_64, bylo potřeba pracovat „pouze“ s Direct2D 1.0 verzí (stejně by nové funkce nešly objektivně porovnávat s GDI/GDI+).

Jak je na začátku zmíněno, tato práce předpokládá alespoň základní znalost principů WinAPI architektury založené na zprávách pro vytvoření uživatelského okna a přijímání zpráv od systému. Proto dále nebude rozebrána celá kostra programu, ale pouze určité úryvky kódu, které jsou důležité specificky pro tuto aplikaci.

Při vytváření hlavního okna se inicializovalo také GDI+ a továrna Direct2D, která se využívá po celou životnost aplikace. Vzhledem k faktu, že GDI a GDI+ podporují implicitně ořezy (clipping) ve smyslu kreslení na plochu aplikace, která je minimalizovaná, mimo displej, překrytá či schovaná při změně velikosti okna, bylo potřeba tomuto zamezit, aby uživatel na první pohled nemohl s oknem aplikace hýbat a znehodnocovat tímto výsledky – naneštěstí lze některé zamezení obejít třeba pomocí správce úloh. U GDI bylo potřeba zakázat změnu velikosti okna a minimalizace, což se provedlo nepoužitím ani jednoho ze stylů nastavení hlavního okna - WS_THICKFRAME, WS_SIZEBOX, WS_TILEDWINDOW, WS_OVERLAPPEDWINDOW, WS_ICONIC, WS_MINIMIZE, WS_MINIMIZEBOX. U GDI+ se musel zakázat pohyb s oknem (viz kód 43) a hlavní okno se muselo nastavit na nejvyšší v pořadí všech oken – aby ho nešlo překrýt (viz kód 44).

```
  case WM_MOVING:
  {
        GetWindowRect(hwnd, (RECT*)lParam);
  }
  break;
```

Kód 43 - zákaz pohybu okna aplikace[76]

```
  SetWindowPos(hwnd, HWND_TOPMOST, 0, 0, 0, 0, SWP_NOMOVE |
SWP_NOSIZE);
```

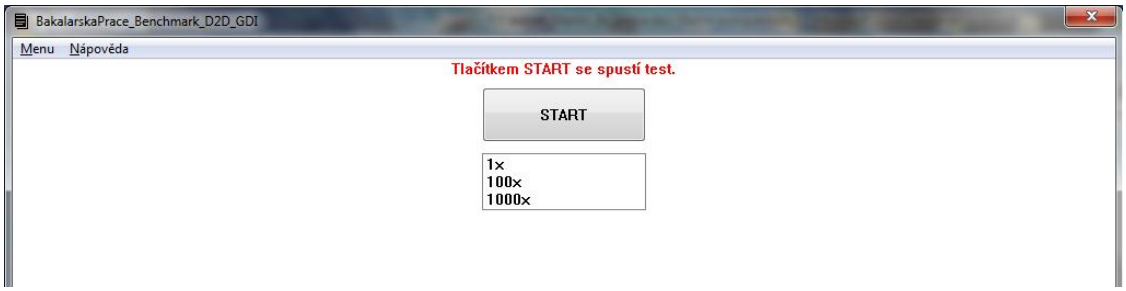
Kód 44 - nastavení okna na „vždy navrchu“

Dále se vytvořily základní prvky uživatelského rozhraní - statický text ukazující informace při testování, tlačítko na spuštění a výběr náročnosti testu – viz kód 45 a obrázek 6.

```
  case WM_CREATE:
  {
        // informační text je vycentrován nahoře
        hStext = CreateWindowExA(0, "Static", "Tlačítkem START se
spustí test.", WS_CHILD | WS_VISIBLE | SS_CENTER, 252, 0, 504, 18,
hwnd, NULL, NULL, NULL);
        // tlačítko pro spuštění testů
        hBstart = CreateWindowExA(0, "BUTTON", "START", WS_CHILD
| WS_VISIBLE | BS_PUSHBUTTON, 429, 25, 150, 50, hwnd,
(HMENU)IDB_START, NULL, NULL);
        // výběr náročnosti testů
        hList = CreateWindowExA(WS_EX_CLIENTEDGE, "Listbox", "",
WS_CHILD | WS_VISIBLE, 429, 85, 150, 60, hwnd, (HMENU)IDL_VYBER,
NULL, NULL);
        SendMessageA(hList, LB_ADDSTRING, 0, (LPARAM)"1x");
        SendMessageA(hList, LB_ADDSTRING, 0, (LPARAM)"100x");
  }
  break;
```

Kód 45 - základní prvky programu[77]

[76] zdroj: autor
[77] zdroj: autor



Obrázek 6 – výřez úvodní obrazovky aplikace[78]

Jelikož chceme, aby aplikace byla responzivní[79], využijeme na to druhé vlákno procesu. Protože celý test může trvat vzhledem k nastavení náročnosti i hardwaru počítače klidně několik minut, aplikace by nebyla schopna odpovídat kupříkladu na vypnutí procesu a musela by se ukončovat přes správce úloh. Při kliknutí na tlačítko se vykoná kód 46:

```
case IDB_START:
{
	// zde předchází zjištění výběru náročnosti testů
	// schováme výběr i tlačítko a spustíme nové vlákno
	ShowWindow(hList, SW_HIDE);
	ShowWindow(hBstart, SW_HIDE);
	DWORD tID;
	hThread = CreateThread(NULL, 0, testuj, NULL, 0, &tID);
}
break;
```

Kód 46 - odezva při kliknutí na tlačítko[80]

Druhé vlákno se stará o spouštění testů a mezi nimi čistí plochu od předchozích – viz kód 47. Tento kód se opakuje pro každý test (pochopitelně s jinými názvy a spuštěným testem). Na konci se pak zavolá to, co je v kódu 48.

[78] zdroj: vlastní aplikace
[79] ve Windows má aplikace zhruba 5 sekund na to, aby přijala zaslanou zprávu od systému, jinak se dostane do stavu „neodpovídá“ a to je velmi nepříjemné pro uživatele, kteří nevědí, co se děje – jestli se aplikace zasekla nebo jen vykonává náročný kód (časově či výkonově)
[80] zdroj: autor

```
  SetWindowTextA(hStext, "ZAČÍNÁ PRVNÍ TEST");
  Sleep(2000);
  SetWindowTextA(hStext, "PRVNÍ TEST > úsečky");
  // spuštění prvního testu
  test1();
  // vyčištění okna
  RedrawWindow(hwnd, NULL, NULL, RDW_INVALIDATE | RDW_ERASE |
RDW_UPDATENOW);
```

Kód 47 - druhé vlákno, testy[81]

```
  // za posledním testem dá druhé vlákno vědět prvnímu, že může
zobrazit výsledky
  SetWindowTextA(hStext, "HOTOVO - TESTY KOMPLETNÍ");
  zobrazVysledek = 1;
  // zde má být synchronizační kód s prvním vláknem, ale pro
jednoduchost stačí pouze vyčistit plochu
  RedrawWindow(hwnd, NULL, NULL, RDW_INVALIDATE | RDW_ERASE |
RDW_UPDATENOW);
```

Kód 48 - druhé vlákno, ukončení[82]

Pro testování byla stěžejní otázka, jakým způsobem testovat danou funkci. Nabízely se dvě možnosti:

1) za určitý časový úsek zjistit, kolikrát se daná funkce zavolala (tedy kolikrát se vykreslil daný snímek) tj. FPS (frames per second; počet snímků za sekundu)
2) předem definovat počet, kolikrát se zavolá daná funkce a zjistit, jak dlouho to trvalo

V této práci byla zvolena možnost druhá s tím, že se znalostí času lehce vypočítáme i FPS, které je zohledněno v testu. Největším „oříškem“ je vybrat vyhovující funkci, která by byla přesná, spolehlivá na všech zařízeních, nebyla náročná na výkon a zároveň měla rozlišení alespoň v mikrosekundách. Taková funkce se prakticky najít nedá, ale pouze se jí přiblížit. GetTickCount a timeGetTime mají jen milisekundy, GetSystemTimeAsFileTime má sice rozlišení po 100 nanosekundách, ale přesnost bývá na úrovni milisekund (je aktualizována typicky v rozmezí milisekund), takže nakonec byla vybrána funkce

[81] zdroj: autor
[82] zdroj: autor

QueryPerformanceCounter (dále jen QPC), která nabízí rozlišení na úrovni menší než mikrosekundy, přesnost a dobu přístupu v desítkách nanosekund, spolehlivost (tj. maximální chybu) kolem ±30 až 50 mikrosekund za 1 sekundu měření a ačkoliv se může zdát, že je docela „vyladěný“ (běží správně i na vícejádrovém procesoru), bohužel mohou nastat i případy, kdy se chová podivně a vrací nesmyslné výsledky (na testovaných počítačích problém nenastal). Jinak QPC se snaží o kvalitní výsledky i tím, že automaticky zjišťuje, jaké druhy časovačů se v systému nachází a dle toho zvolí ten nejvhodnější (TSC Registr, PM hodiny (ACPI časovač), HPET časovač (High Precision Event Timer)) a vyhýbá se TSC, který není neměnný (non-invariant; mění frekvenci za chodu) a takovým, které by byly ovlivněny moderními funkcemi měnící frekvenci procesoru.[83] QPC byl pro změření mikrosekund použit způsobem, jak ukazuje kód 49.

```
  LARGE_INTEGER qpcZac;
  LARGE_INTEGER qpcKon;
  LARGE_INTEGER qpcf;
  QueryPerformanceFrequency(&qpcf);
  QueryPerformanceCounter(&qpcZac);
  // zde je funkce, kterou chceme měřit
  QueryPerformanceCounter(&qpcKon);
  // převod na mikrosekundy a uložení časového úseku
  long long celk = ((qpcKon.QuadPart - qpcZac.QuadPart) * 1000000)
/ (qpcf.QuadPart);
```

Kód 49 - použití QPC

Pokud by se rozdíl vynásobil 1000krát, pak by to byly jen milisekundy a pokud 1000000000krát, byli bychom na úrovni nanosekund.

Dále se vypočítalo FPS dle vzorce: FPS = 1000000 / vypočítaný čas. Měřily se také cykly CPU (QueryThreadCycleTime), které se dají do určité míry použít na porovnání využití CPU. Každý test byl proveden 100krát za sebou kvůli statistickým důvodům a následně zprůměrován prostým aritmetickým průměrem. Aby se takovému průměru dalo věřit, byl proveden výpočet variačního koeficientu, který v procentech vyjadřuje, kolik procent průměru se podílí na směrodatné odchylce (ta se vyjadřuje ve stejných jednotkách

[83] Acquiring high-resolution time stamps. MICROSOFT. *Windows Desktop Development – Windows Dev Center* [online]. © 2015 [cit. 2015-02-10]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/dn553408%28v=vs.85%29.aspx

jako je průměr a udává průměrné odchýlení hodnot od průměru). Aby byl průměr ještě přesnější, bylo také provedeno ořezání prvních hodnot kvůli vysokým hodnotám na začátku, kdy se inicializují různé buffery atp.

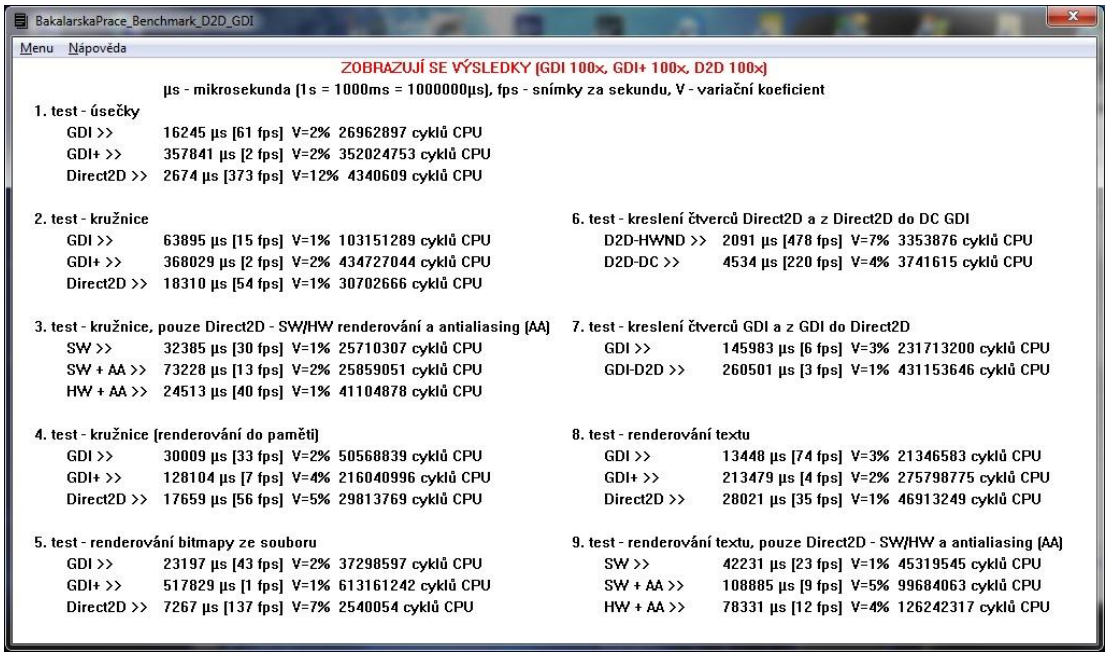
Každý test se skládá z funkcí, které byly vybrány tak, aby měly stejný (nebo téměř totožný) výstup v každém API a zvolená náročnost na začátku testování určuje počet zavolání těchto funkcí. Neměřila se inicializace, alokace zdrojů atd., ale pouze ta hlavní funkce zodpovědná za výstup. Protože všechna tři API používají hromadění příkazů a až po určitém příkazu nebo zaplnění paměti na příkazy se pošlou na hromadné vykonání z důvodu zlepšení výkonu, tak byly vždy příkazy těsně před skončením měření času vyprázdněny. Problém nastal pouze u GDI+, kde příkaz na vyprázdnění způsobil prodlevu/zaokrouhlení odpovídající vertikální synchronizaci tj. cca 16 ms u 60Hz monitoru, proto zde nebylo provedeno vyprazdňování (testy však ukázaly, že i přesto výsledky nejsou tímto ovlivněny, jako kdyby bylo vyprazdňování provedeno). U Direct2D se vyprazdňuje automaticky, pokud se zavolá EndDraw a u GDI se použilo GdiFlush.

Testů bylo vytvořeno 9 a jejich složení je následující:

1) nakreslení 5 úseček; Direct2D: DrawLine; GDI: Polyline; GDI+: DrawLine
2) nakreslení 3 kružnic; Direct2D: DrawEllipse; GDI: Ellipse; GDI+: DrawEllipse
3) pouze Direct2D - nakreslení 3 kružnic; s nastavením na CPU výpočty s/bez vyhlazování hran a GPU výpočty s vyhlazováním hran
4) nakreslení 3 kružnic do paměťové bitmapy (nezobrazí se výstup); Direct2D: CreateCompatibleRenderTarget; GDI a GDI+: CreateCompatibleDC, CreateCompatibleBitmap
5) vykreslení bitmapy ze souboru; Direct2D: WIC + CreateBitmapFromWicBitmap, DrawBitmap; GDI: LoadImage, CreateCompatibleDC, SelectObject (do nového DC vložíme načtenou bitmapu), BitBlt (z nového DC zkopírujeme do DC hlavního okna); GDI+: Bitmap, DrawImage
6) pouze Direct2D – porovnání kreslení čtverců z Direct2D do DC GDI oproti běžnému HWND; Direct2D (HWND): DrawRectangle; Direct2D (DC): CreateDCRenderTarget, BindDC, DrawRectangle
7) pouze GDI – porovnání kreslení čtverců z GDI do Direct2D oproti běžnému DC; GDI (Direct2D): QueryInterface, GetDC, Rectangle; GDI (DC): Rectangle

8) renderování textu; Direct2D: DirectWrite + DrawText; GDI: CreateFont, TextOut; GDI+: CreateFont, Font, DrawString
9) pouze Direct2D - renderování textu; s nastavením na CPU výpočty s/bez vyhlazování hran textu a GPU výpočty s vyhlazováním hran

Po dokončení testů jsou vypsány výsledky, jak je vidět na obrázku 7.



Obrázek 7 - zobrazené výsledky testů[84]

Kompletní zdrojový kód aplikace je dostupný v příloze na CD.

[84] zdroj: vlastní aplikace

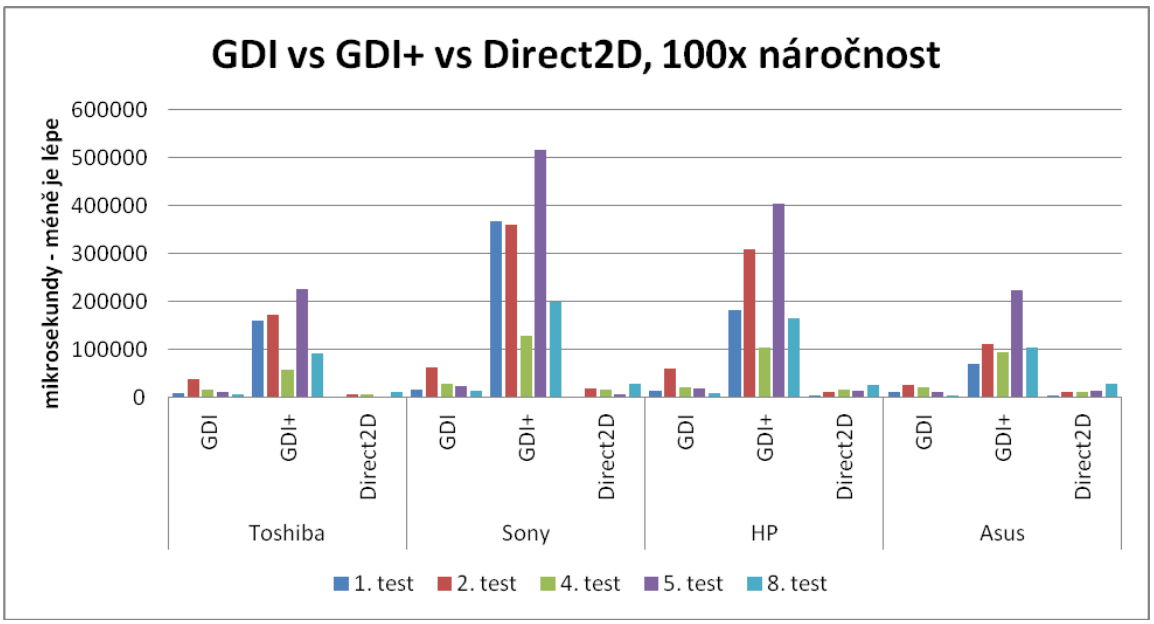
5 Zhodnocení výsledků

Pro tuto aplikaci byly záměrně zvoleny 4 následující konfigurace počítačů na otestování (aby se projevily rozdíly v testech):

1) Toshiba – QOSMIO X300 14Y - CPU: Intel T9550, GPU: Nvidia 9800M GTS (DirectX 10), Windows 7 (64bit)
2) Sony – VAIO SVE1111M1EW – CPU: AMD E2-1800, GPU: AMD HD 7300 (DirectX 11), Windows 7 (64bit)
3) HP - COMPAQ 6720s – CPU: Intel M 530, GPU: Intel GMA X3100 (DirectX 10), Windows 7 (32bit)
4) Asus – A6KM – CPU: AMD ML37, GPU: Nvidia GO 7300 (DirectX 9), Windows 7 (32bit)

Při testování byla snaha nemít na pozadí běžící úlohy a mít frekvence CPU a GPU v nešetřících stavech.

Obrázek 8 ukazuje celkové shrnutí testů, které porovnávají mezi sebou GDI, GDI+ a Direct2D.

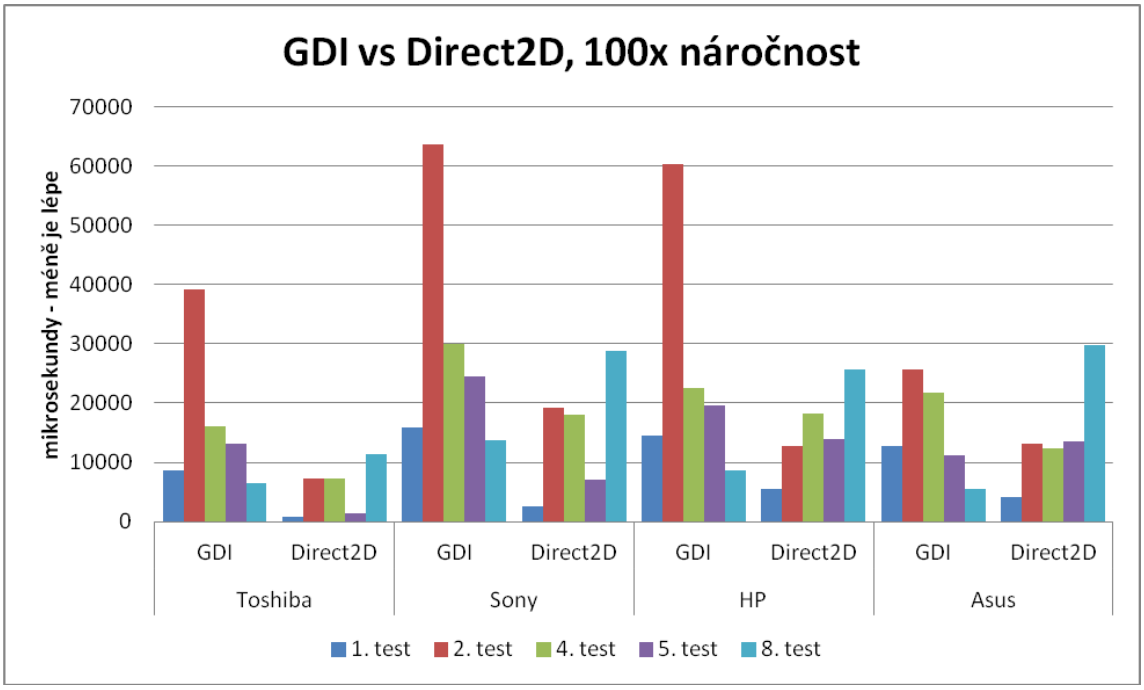


Obrázek 8 - shrnutí vybraných testů, 100x náročnost[85]

[85] zdroj: vlastní data z aplikace

Jak z obrázku 8 vyplývá, Direct2D tedy bez problému běží i s hardwarovou akcelerací na DirectX 11, 10 i 9 GPU. Zároveň je však zřejmé, že GDI+, které běží pouze na CPU, nemá šanci dohnat rychlostí GDI natož Direct2D. Také si lze všimnout jedné pozoruhodné věci - Asus, který běží na Direct3D 9 a tudíž používá ovladač na bázi WDDM 1.0 (oproti ostatním běžící na WDDM 1.1), podává výsledky v GDI+ na úrovni stejné či lepší než několikrát výkonnější Toshiba.

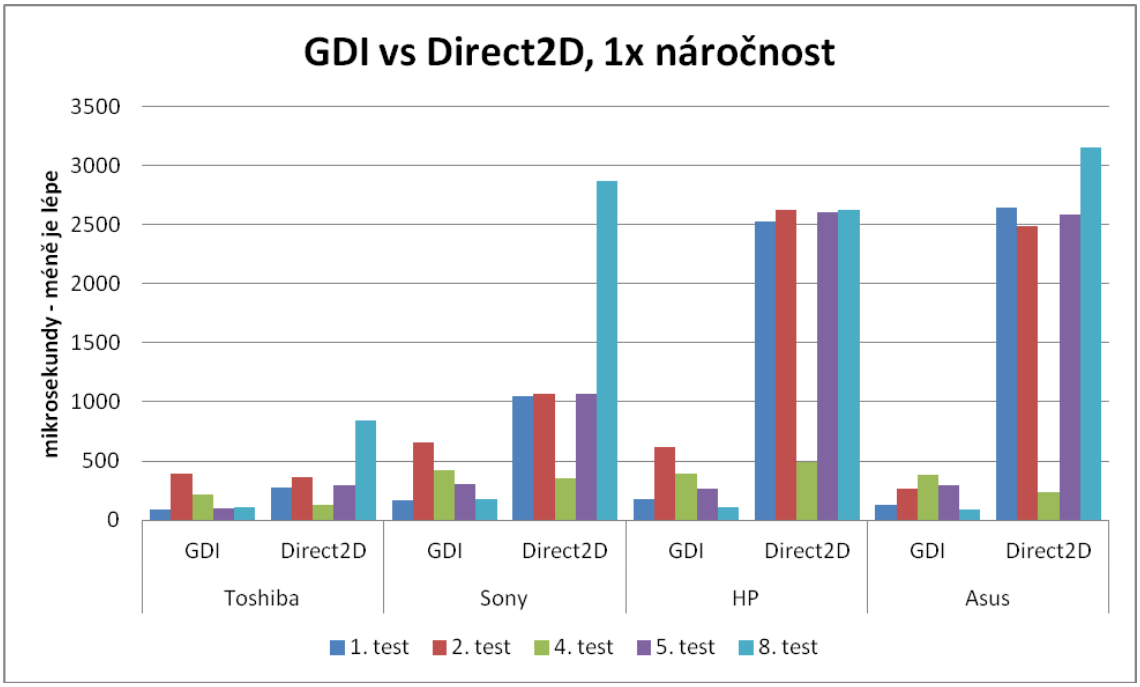
Bližší porovnání GDI a Direct2D je na obrázku 9.



Obrázek 9 - GDI a Direct2D ve vybraných testech, 100x náročnost[86]

Na obrázku 9 lze jasně vidět převahu Direct2D kromě posledního testu. Osmý test se týká vykreslování textu a Direct2D tu prohrává.

[86] zdroj: vlastní data z aplikace



Obrázek 10 - GDI a Direct2D ve vybraných testech, 1x náročnost[87]

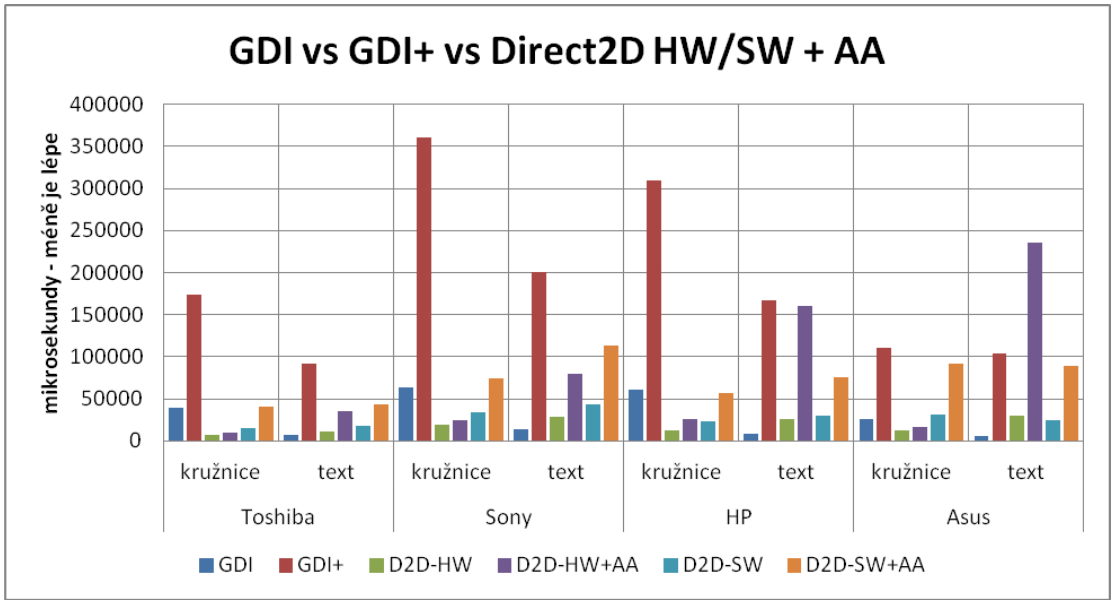
Obrázek 10 vypovídá o dalším zajímavém poznatku. Pokud se vykresluje velmi malé množství grafiky (místo stovek obrázků, kružnic atd. pouze několik), pak je GDI mnohem rychlejší než Direct2D, které potřebuje určitý minimální čas pro vykreslení. Obrázek 11 ukazuje případ, jak pomáhá GDI i GDI+ vykreslování do paměti místo přímo do okna aplikace. U Direct2D rozdíl není moc patrný, protože tam už pracujeme s pamětí GPU a není tedy velký rozdíl mezi vykreslením do viditelné oblasti okna nebo do paměti, kdežto u GDI/GDI+ pracujeme se systémovou RAM (místo s GPU pamětí, do které je časově drahé posílat data), a proto je rozdíl markantní.

```
2. test - kružnice
    GDI >>       63606 µs [15 fps]  V=8%  100901983 cyklů CPU
    GDI+ >>      361010 µs [2 fps]  V=0%  418046825 cyklů CPU
    Direct2D >>  19146 µs [52 fps]  V=4%  31643490 cyklů CPU
4. test - kružnice (renderování do paměti)
    GDI >>       29984 µs [33 fps]  V=1%  50408722 cyklů CPU
    GDI+ >>      129993 µs [7 fps]  V=2%  217930579 cyklů CPU
    Direct2D >>  17982 µs [55 fps]  V=5%  30298263 cyklů CPU
```

Obrázek 11 - renderování do okna aplikace a do paměti[88]

[87] zdroj: vlastní data z aplikace

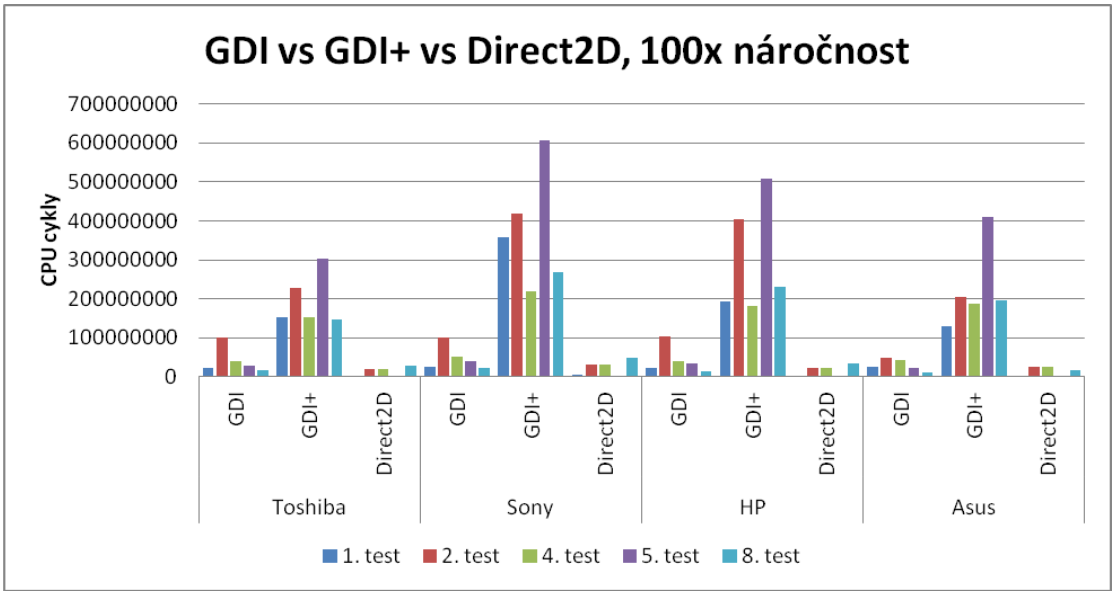
Následující obrázek 12 ukazuje, jak Direct2D v nejvyšší kvalitě vyhlazování a s výpočty na CPU překonává GDI+ (na nejnižší kvalitě).



Obrázek 12 - GDI, GDI+ a Direct2D v SW a HW módu, kružnice a text[89]

Obrázek 13 zobrazuje CPU cykly, které byly potřeba pro vykonání příslušných testů. Z obrázku je patrné, že Direct2D má kromě 8. testu, což je pochopitelné kvůli vyššímu času, nejlepší výsledky. To znamená, že skutečně je vytěžováno více GPU, zatímco odlehčuje CPU.

[88] zdroj: vlastní aplikace
[89] zdroj: vlastní data z aplikace



Obrázek 13 - GDI, GDI+, Direct2D CPU cykly[90]

Všechny výsledky z aplikace od jednotlivých konfigurací jsou dostupné v příloze na CD.

5.1 Výhody a nevýhody GDI, GDI+ a Direct2D

Pokud si to vše shrneme, dostaneme se k následujícím zjištěním. Direct2D má velký potenciál – stále se vylepšuje, je velmi výkonné díky akceleraci, bez akcelerace používá výkonný softwarový rasterizér, spolupracuje s mnoha API, ale rozhodně v plné míře nenahrazuje GDI, které se také kupodivu zdokonaluje – od Windows 7 byla vylepšena správa paměti (každé okno už nemusí mít duplikát v systémové RAM a GPU RAM) a paralelizace (současný běh více GDI aplikací dříve zdržovala celková synchronizace).[91] GDI má stále své místo v nenáročné a rychlé grafice, kde Direct2D by bylo pomalejší (minimální čas na komunikaci s GPU a časově náročné vytváření zdrojů na GPU). Oproti GDI se musí u Direct2D počítat s optimalizacemi, které jsou nutné pro zvýšení výkonu. Rozhodně by se ale dalo říct, že Direct2D nahrazuje GDI+ v rychlosti, kvalitě i funkcích. Direct2D je složitější a kódově rozsáhlejší než GDI/GDI+. GDI funguje prakticky všude, kdežto Direct2D 1.0 je omezeno na Windows Vista a novější.

[90] zdroj: vlastní data z aplikace
[91] SINOFSKY, Steven. Engineering Windows 7 Graphics Performance. MICROSOFT. *MSDN Blogs* [online]. 25 Apr 2009 3:00 AM [cit. 2015-02-24]. Dostupné z: http://blogs.msdn.com/b/e7/archive/2009/04/25/engineering-windows-7-for-graphics-performance.aspx

6 Závěr

Při psaní práce jsem narazil na problém v podobě žádné literatury, která by se týkala Direct2D nebo GDI+ v rámci programovacího jazyka C++. Přesto se však povedlo problematiku a programovací principy v těchto API nastudovat bez problému z internetových zdrojů a dokumentací přímo od Microsoftu a práci to tedy nijak na kvalitě neubírá.

Hlavním cílem této práce bylo charakterizovat vybraná API pro programování 2D grafiky v prostředí Microsoft Windows, což bylo splněno v podobě popisu každé technologie a přidáním konkrétního příkladu s tím, že čtenář se dozví, jak vlastně s daným API vůbec začít a co je k tomu potřeba.

Dílčím cílem bylo vytvoření aplikace srovnávající vybrané technologie z hlediska technické náročnosti jednotlivých přístupů. Tento cíl byl také splněn, jelikož se podařilo úspěšně spojit všechna API v jedné aplikaci a na konci testování vypsat veškeré zjištěné výsledky. Aplikace by se mohla vylepšit na zjišťování i jiných hodnot - například využití paměti GPU i CPU, ale vzhledem k velikostem dnešních pamětí nebyly konkrétně tyto informace pro práci důležité. Avšak dalo by se navázat na tuto práci porovnáním kvality výstupů jednotlivých API a analyzovat více do hloubky problematiku textu (fonty, glyfy, unikód atp.), což zde nebylo možné vzhledem k rozsahu práce.

Výsledkem této práce je zjištění, že Direct2D je většinou rychlejší než GDI a to pouze v případě, že dostane k vykreslování (zpracování) více úkonů, jinak převládá GDI, které naopak nemůže konkurovat se svojí kvalitou, ale pouze rychlostí a podporou. A také, že Direct2D je mnohonásobně rychlejší než GDI+ ve všech případech, a proto se dá konstatovat, že hardwarová akcelerace pro grafiku je velmi výhodná při masivnějších výpočtech - zvláště s dnešními výkonnými GPU.

7 Seznam použitých zdrojů

About Direct2D. MICROSOFT. Windows Desktop Development – Windows Dev Center [online]. © 2015 [cit. 2015-02-08]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/dd370987%28v=vs.85%29.aspx

Acquiring high-resolution time stamps. MICROSOFT. Windows Desktop Development – Windows Dev Center [online]. © 2015 [cit. 2015-02-10]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/dn553408%28v=vs.85%29.aspx

Comparing Direct2D and GDI Hardware Acceleration. MICROSOFT. Windows Desktop Development – Windows Dev Center [online]. © 2015 [cit. 2015-01-29]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/ff729480%28v=vs.85%29.aspx

Creating a Simple Direct2D Application. MICROSOFT. *Windows Desktop Development – Windows Dev Center* [online]. © 2015 [cit. 2015-02-08]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/dd370994%28v=vs.85%29.aspx

Direct2D and GDI Interoperability Overview. MICROSOFT. Windows Desktop Development – Windows Dev Center [online]. © 2015 [cit. 2015-02-12]. Dostupné z: https://msdn.microsoft.com/ehttps://msdn.microsoft.com/en-us/library/windows/desktop/dd370971%28v=vs.85%29.aspx

Direct2D and High-DPI. MICROSOFT. Windows Desktop Development – Windows Dev Center [online]. © 2015 [cit. 2015-02-07]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/dd756649%28v=vs.85%29.aspx

Direct2D. MICROSOFT. Windows Desktop Development – Windows Dev Center [online]. © 2015 [cit. 2015-02-07]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/dd370990%28v=vs.85%29.aspx

DirectWrite. MICROSOFT. Windows Desktop Development – Windows Dev Center [online]. © 2015 [cit. 2015-02-15]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/dd368038%28v=vs.85%29.aspx

Effects. MICROSOFT. Windows Desktop Development – Windows Dev Center [online]. © 2015 [cit. 2015-02-08]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/hh973240%28v=vs.85%29.aspx

Improving the performance of Direct2D apps: Per-primitive caching using geometry realizations. MICROSOFT. Windows Desktop Development – Windows Dev Center [online]. © 2015 [cit. 2015-02-08]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/dd372260%28v=vs.85%29.aspx

Interoperating with GDI. MICROSOFT. Windows Desktop Development – Windows Dev Center [online]. © 2015 [cit. 2015-02-15]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/dd742734%28v=vs.85%29.aspx

Introducing DirectWrite. MICROSOFT. Windows Desktop Development – Windows Dev Center [online]. © 2015 [cit. 2015-02-08]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/dd371554%28v=vs.85%29.aspx

KATY. Direct2D 1.1 Migration Guide for Windows 7 Developers. Katy's Code [online]. January 23, 2013 [cit. 2015-02-10]. Dostupné z: https://katyscode.wordpress.com/2013/01/23/migrating-existing-direct2d-applications-to-use-direct2d-1-1-functionality-in-windows-7/

New Features. MICROSOFT. MSDN - Microsoft Developer Network [online]. © 2015 [cit. 2015-02-01]. Dostupné z: https://msdn.microsoft.com/en-us/library/ms536340%28v=vs.85%29.aspx

OLSEN, Thomas. Tom's Blog: Introducing the Microsoft Direct2D API. TechNet Blogs [online]. 29 Oct 2008 1:42 PM [cit. 2015-02-04]. Dostupné z: http://blogs.technet.com/b/thomasolsen/archive/2008/10/29/introducing-the-microsoft-direct2d-api.aspx

SINOFSKY, Steven. Engineering Windows 7 Graphics Performance. MICROSOFT. *MSDN Blogs* [online]. 25 Apr 2009 3:00 AM [cit. 2015-02-24]. Dostupné z: http://blogs.msdn.com/b/e7/archive/2009/04/25/engineering-windows-7-for-graphics-performance.aspx

Supported Pixel Formats and Alpha Modes. MICROSOFT. Windows Desktop Development – Windows Dev Center [online]. © 2015 [cit. 2015-02-08]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/dd756766%28v=vs.85%29.aspx

Using Uniscribe. MICROSOFT. Windows Desktop Development – Windows Dev Center [online]. © 2015 [cit. 2015-02-15]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/dd374127%28v=vs.85%29.aspx

WIC API Overview. MICROSOFT. Windows Desktop Development – Windows Dev Center [online]. © 2015 [cit. 2015-02-08]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/ee719655%28v=vs.85%29.aspx

Windows GDI. MICROSOFT. MSDN - Microsoft Developer Network [online]. © 2015 [cit. 2015-01-27]. Dostupné z: https://msdn.microsoft.com/en-us/library/dd145203%28v=vs.85%29.aspx

8 Seznam obrázků

9 Seznam kódů