Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

.NET Wrapper pro knihovnu k zařízení Phantom

Plzeň 2010 Adam Boháč

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 13.5.2010 Adam Boháč

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Petru Vaněčkovi Ph.D. za konzultace, cenné připomínky a rady.

Abstract

OpenHaptics library .NET Wrapper

The Bachelor Thesis aims to describe the process of designing .NET wrapper library for Phantom and the way of its automating so that it is possible to use the wrapper for new library versions as well. It is demonstrated how to use the .NET wrapper in the library.

The possibility of using the haptic device Phantom for counting of surface characteristics is explored and the process of designing the programme for this experiment documented.

In the following part I have described how I solved the problems that I came across and how to improve the counting so that the reproduction of gained data is as authentic as possible. The explanation of how I reproduced the counted characteristics follows.

Obsah

1 Úvod

V tomto dokumentu nastíním postup při vytváření .NET wrapperu pro knihovnu k zařízení Phantom. Popíšu, jak jeho tvorbu co nejvíce zautomatizovat, aby bylo možné wrapper použít i pro nové verze knihovny.

Na ukázkových úlohách předvedu, jak používat knihovnu s využitím .NET wrapperu.

Dále prozkoumám možnost využití haptického zařízení Phantom pro načítání charakteristiky povrchů. Zdokumentuji, jak jsem postupoval při vytváření programu pro tento experiment, jak jsem řešil problémy, na které jsem narazil, a další možná vylepšení načítání.

Cílem načítání charakteristiky je, aby reprodukce získaných dat byla co nejvěrnější. V další části vysvětlím, jakým způsobem jsem načtenou charakteristiku reprodukoval.

2 Co je to haptika a k čemu haptické zařízení slouží

Haptický znamená hmatový. Výraz pochází z řečtiny a označuje první smysl, který se vyvine u lidského plodu. Tento smysl je definován jako: „schopnost jedince vnímat svět sousedící s jeho tělem prostřednictvím svého těla“.

Haptika je také věda, která se zabývá ovládání hapticky přizpůsobených počítačových aplikací pomocí sil a jejich zpětné interpretace uživateli. K tomu se používá speciální zařízení, haptické zařízení, které umožňuje uživateli cítit a manipulovat s virtuálními třídimenzionálními objekty. Typ zpětné vazby použité k předání hmatové informace uživateli je určený typem použitého haptického zařízení. [1]

Hmatové zařízení se používá v různých odvětvích, která se neustále rozvíjejí. [1]

Patří mezi ně:

- chirurgické operace a lékařské odborné přípravy
- malování a modelování v CAD systémech
- vojenské aplikace, např. simulace vojenského a leteckého výcviku
- jednoduchá interakce s uživatelským prostředím počítačových aplikací, jako je otevírání a zavírání okna, či vybírání z menu
- v odvětvích herního průmyslu

[1]

3 Zařízení Phantom

Phantom je haptické zařízení společnosti SensAble, které umožňuje uživatelům dotýkat se, cítit a manipulovat virtuálními objekty [3]. Na Obrázku 1 je zařízení Phantom Omni, na kterém se provádělo testování funkčnosti wrapperu, ukázkových úloh a které je použito pro experiment možnosti načítání charakteristik povrchů.

Obrázek 1: Phantom Omni Haptic Device [3]

Tento Phantom se připojuje pomocí portu standartu IEEE-1394a FireWare [3].

3.1 Umístění haptického bodu zařízení

Haptický bod zařízení je umístěn na konci ramena zařízení Phantom. Tento bod se nazývá haptic interface point (HIP) a jeho umístění je stejné i pro všechna ostatní zařízení Phantom. [1]

Obrázek 2: Umístění haptického bodu zařízení [1]

3.2 Souřadný systém haptického zařízení Phantom

Souřadný systém haptického zařízení Phantom Omni je pravotočivý souřadný systém. Osa X je položená vodorovně, vpravo nabývá kladných hodnot. Osa Y je položená svisle, kladných hodnot nabývá směrem nahoru. Osa Z je tedy vodorovná a je kolmá na zařízení, kladných hodnot nabývá směrem od zařízení.

3.3 PHANTOM Omni Technical Specification

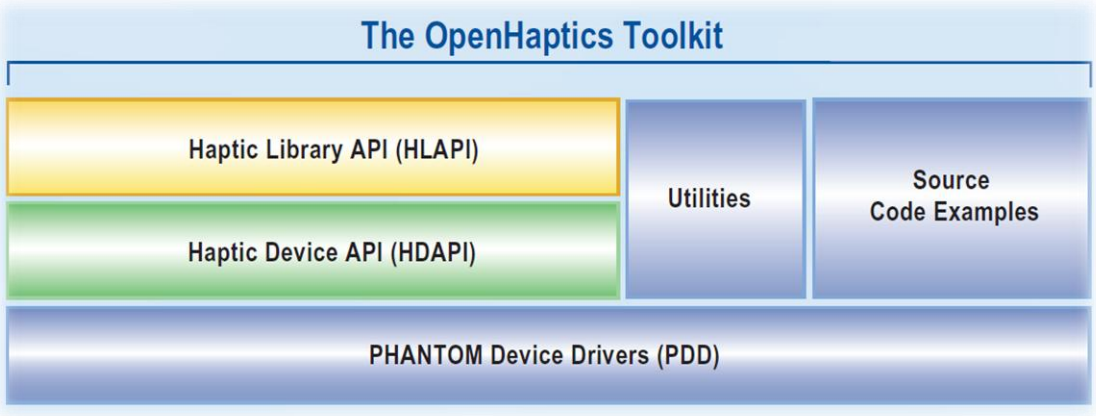
Force feedback workspace	~6.4 W x 4.8 H x 2.8 D in. > 160 W x 120 H x 70 D mm.
Footprint (Physical area device base occupies on desk)	6 5/8 W x 8 D in. ~168 W x 203 D mm.
Weight (device only)	3 lbs. 15 oz.
Range of motion	Hand movement pivoting at wrist
Nominal position resolution	> 450 dpi. ~ 0.055 mm.
Backdrive friction	< 1 oz (0.26 N)
Maximum exertable force at nominal (orthogonal arms)position	0.75 lbf. (3.3 N)
Continuous exertable force (24 hrs.)	> 0.2 lbf. (0.88 N)
Stiffness	X axis > 7.3 lbs. / in. (1.26 N / mm.) Y axis > 13.4 lbs. / in. (2.31 N / mm.) Z axis > 5.9 lbs. / in. (1.02 N / mm.)
Inertia (apparent mass at tip)	~0.101 lbm. (45 g)
Force feedback	x, y, z
Position sensing [Stylus gimbal]	x, y, z (digital encoders) [Pitch, roll, yaw (± 5% linearity potentiometers)
Interface	IEEE-1394 FireWire® port: 6-pin to 6-pin
Supported platforms	Intel or AMD-based PCs
OpenHaptics® Toolkit compatibility	Yes
Applications	Selected Types of Haptic Research, FreeForm® Modeling™ system, ClayTools™ system

[3]

4 OpenHaptics Toolkit

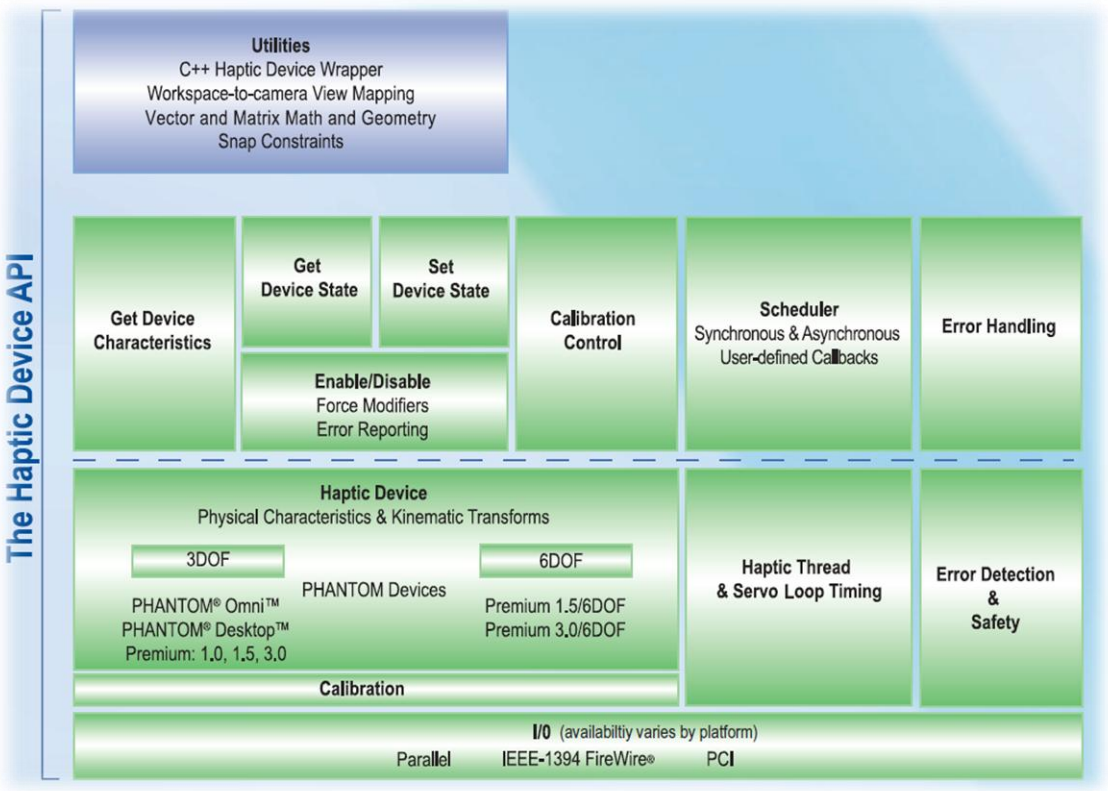
OpenHaptics Toolkit umožňuje vývojářům softwaru přidat haptiku a třídimenzionální ovládání do široké škály aplikací. Hlavně do aplikací, které jsou navržené pro 3D navrhování, modelování, lékařství, hraní, zábavu, vizualizaci a simulaci. Tato sada softwarových nástrojů je vytvořená podle OpenGL API. To ji dělá snadno pochopitelnou pro programátory grafiky a usnadňuje propojení s OpenGL aplikacemi. Použitím této sady můžou vývojáři využít existující OpenGL kód pro specifikování geometrie objektů a doplnit ho příkazy z OpenHaptics sady k simulování vlastností haptických materiálů jako je napětí a tuhost. Rozšířitelná architektura umožňuje vývojářům přidání další funkcionality k podpoře nových typů geometrických útvarů. Sada je také navržena k začlenění podpůrných knihoven jako je fyzikálně-dynamický nebo kolize detekující engine. [4]

OpenHaptics Toolkit (viz Obrázek 3) obsahuje Haptic Device API (HDAPI), Haptic Library API (HLAPI), podpůrné části (utilities), ovladače k zařízení – PHANTOM Device Drivers (PDD) a zdrojové kódy ukázkových aplikací. [4]



Obrázek 3: The OpenHaptics Toolkit [4]

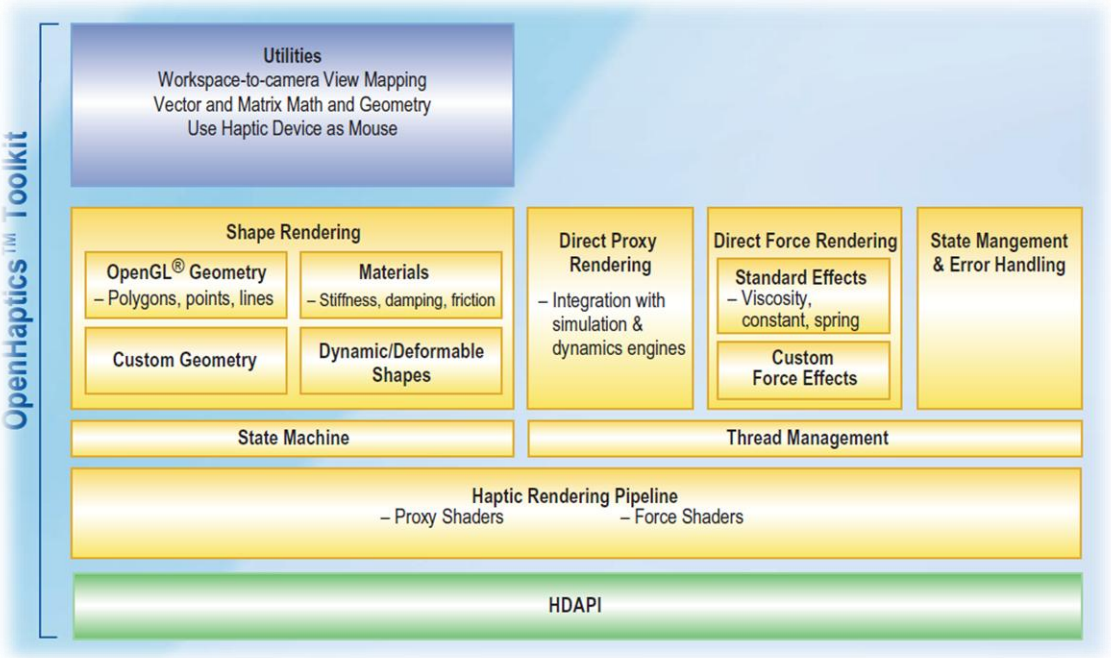
HDAPI (viz Obrázek 4) je základní nízkoúrovňová vrstva pro haptiku. Je určená pro vývojáře, kteří jsou již obeznámeni s haptickými modely a posíláním sil přímo do zařízení. To zahrnuje ty vývojáře, kteří se zajímají o výzkum v oboru haptiky, telepresence a dálkové manipulace. [5]



Obrázek 4: HDAPI - Low-level Device Access [5]

HLAPI (viz Obrázek 5) vrstva je navržena pro vysokoúrovňové programování haptiky. Je určena pro pokročilé OpenGL vývojáře, kteří jsou s programováním haptického zařízení obeznámeni méně, ale potřebují rychlé a snadné přidání haptiky do existující grafické aplikace. [1]

HLAPI je postaven na HDAPI a na úkor flexibility poskytuje vyšší úroveň kontroly haptiky, než HDAPI. HLAPI je primárně navržen pro jednoduché použití těm, kteří se dobře orientují v programování OpenGL. Například HLAPI programátor se nebude starat o otázky z nižší vrstvy jako je navrhování silových rovnic, udržování vlákna a implementování vysoce efektivní datové struktury pro vykreslování do haptiky. HLAPI respektuje tradiční grafické techniky, jako jsou v OpenGL API. Přidání haptiky nějakému objektu je poměrně triviální proces. Haptický model je podobný modelu, který je použit pro grafickou reprezentaci objektu. HLAPI také disponuje obsluhou událostí pro jednoduché propojení aplikací. [1]



Obrázek 5: HLAPI - High-level Haptic Redering [6]

5 Co je to .NET wrapper a k čemu slouží

.Net wrapper je obalová třída ke knihovně, která je napsána v jiném programovacím jazyce, v tomto případě v C/C++, pro použití v jazyce C#. Programovací jazyk C# využívá spravované paměti, to znamená, že se nemusíme starat o alokování a uvolňování paměti. To vše za nás dělá `Garbage collector` (viz kapitola 6). V programovacím jazyce C/C++ je tomu jinak, tam se musíme starat o paměť. Jelikož budeme přenášet data ze spravované paměti do nespravované paměti a naopak, musíme použít třídu `Marshal` (viz kapitola 5.1), která poskytuje potřebné metody. Aby šlo wrapper jednoduše vytvořit i pro nové verze knihoven, jsem napsal program, který kód wrapperu generuje jako svůj výstup.

5.1 Třída Marshal

Tato třída poskytuje soubor metod pro přidělování nespravované paměti a kopírování nespravovaných paměťových bloků. Dále poskytuje metody pro převádění řízených typů na neřízené, stejně jako další různorodé metody použité pro spolupráci s neřízeným kódem. [7]

Statické metody definované ve třídě `Marshal` jsou nezbytné pro práci s neřízeným kódem. Většina těchto metod je typicky použitá vývojáři, kteří chtějí udělat most mezi řízenými a neřízenými programovacími modely. Například metoda `StringToHGlobalAnsi` kopíruje ANSI znaky ze specifikovaného řetězce (v řízené paměti) do vyrovnávací paměti v haldě neřízené paměti. Také alokuje správnou velikost paměti v neřízené části. [7]

Následující kód ukazuje, jak použít různé druhy metod, které jsou definovány ve třídě `Marshal`. [7]

```
using System;
using System.Text;
using System.Runtime.InteropServices;

public struct Point
{
    public Int32 x, y;
}

public sealed class App
{
```

```
    static void Main()
    {
        // Demonstrate the use of public static fields of the Marshal
class.
        Console.WriteLine("SystemDefaultCharSize={0},
SystemMaxDBCSCharSize={1}",
            Marshal.SystemDefaultCharSize,
Marshal.SystemMaxDBCSCharSize);

        // Demonstrate the use of the SizeOf method of the Marshal
class.
        Console.WriteLine("Number of bytes needed by a Point object:
{0}",
            Marshal.SizeOf(typeof(Point)));
        Point p = new Point();
        Console.WriteLine("Number of bytes needed by a Point object:
{0}",
            Marshal.SizeOf(p));

        // Demonstrate how to call GlobalAlloc and
        // GlobalFree using the Marshal class.
        IntPtr hglobal = Marshal.AllocHGlobal(100);
        Marshal.FreeHGlobal(hglobal);

        // Demonstrate how to use the Marshal class to get the Win32
error
        // code when a Win32 method fails.
        Boolean f = CloseHandle(new IntPtr(-1));
        if (!f)
        {
            Console.WriteLine("CloseHandle call failed with an error
code of: {0}",
                  Marshal.GetLastWin32Error());
        }
    }

    // This is a platform invoke prototype. SetLastError is true,
which allows
    // the GetLastWin32Error method of the Marshal class to work
correctly.
    [DllImport("Kernel32", ExactSpelling = true, SetLastError = true)]
    static extern Boolean CloseHandle(IntPtr h);

}

// This code produces the following output.
//
// SystemDefaultCharSize=2, SystemMaxDBCSCharSize=1
// Number of bytes needed by a Point object: 8
// Number of bytes needed by a Point object: 8
// CloseHandle call failed with an error code of: 6
```

[7]

6 Garbage collector

Garbage collector se stará o automatickou správu paměti, to znamená o její alokování a uvolňování. Microsoft .NET Framework common language runtime (CLR) požaduje, aby všechny objekty byly alokovány na řízené haldě. [8]

Po spuštění runtime procesu programu Garbage collector rezervuje sousední volnou oblast paměti. Tento blok paměti je řízená halda procesu programu. Halda si udržuje ukazatel, který je po vytvoření nastaven na vlastní adresu tohoto paměťového bloku. Tento ukazatel ukazuje na místo na haldě, kde může být alokován další objekt. [8]

Jakmile aplikace vytvoří nový objekt pomocí operátoru `new`, tak Garbage collector zjistí přičtením velikosti nového objektu k ukazateli na haldě, jestli se tam objekt vejde. Pokud ukazatel ukazuje za hranici haldy, znamená to, že je plná a objekt se tam nevejde. V tento moment se musí spustit proces čistění (collection) (viz kapitola 6.1). Pokud se objekt do haldy vejde, tak je vrácena adresa nově vytvořeného objektu a generace označena na 0 (viz kapitola 6.1.2). [8]

6.1 Garbage collection

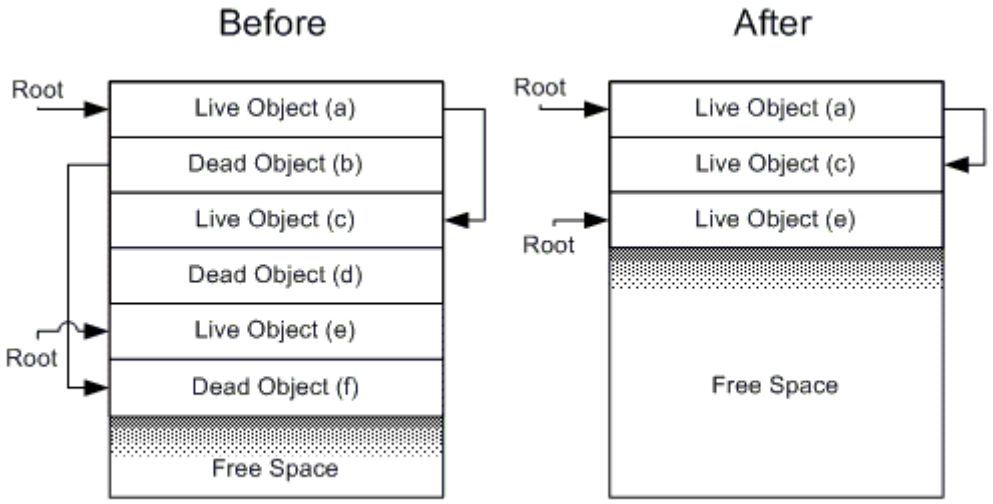
Proces čistění má několik fází:

- Hodnotící fáze, která vyhledá všechny živé objekty a vytvoří si jejich seznam.
- Přemísťovací fáze, která aktualizuje reference na objekty, které budou přesunuty z důvodu odstranění malých prázdných nevyužitelných míst mezi nimi.
- Stlačující fáze - získává místo, které je obsazováno mrtvými objekty a stlačí přežívající objekty k sobě. Stlačující fáze přesouvá objekty, které přežili čistění směrem ke generačně staršímu konci segmentu (viz Obrázek 6).

[9]

6.1.1 Úplné čistění

Při tomto typu čistění se musí pozastavit vykonávání programu a zjistit všechny kořeny (roots) v Garbage collector haldě. Tyto kořeny mají různé formy, především to jsou statické a globální proměnné, které ukazují na haldu. Od těchto kořenů začneme prohledávat. Budeme následovat každý ukazatel v každém navštíveném objektu a tím navštívíme každý živý objekt haldy. Touto cestou bude mít sběrač informaci o každém dosažitelném, neboli živém objektu. Ostatní nedosažené objekty, jsou nyní označené za nepoužívané (mrtvé). [10]



Obrázek 6: Garbage Collecting [10]

Když už byl nepřístupný objekt identifikován, tak chceme získat jeho uvolněné místo pro pozdější použití. Při pozastaveném běhu programu je bezpečné pro kolektor přesouvat objekty a upravovat ukazatele, tak aby všechny objekty byly správně propojené v jejich novém místě v paměti. Přeživší objekty jsou povýšeny na další generaci (viz kapitola 6.1.2) a běh programu může pokračovat. [10]

6.1.2 Částečné čistění

Úplné čistění je příliš drahé, než aby bylo prováděno příliš často, takže budeme využívat generace (generační algoritmus).

Za prvé uvažujme smyšlený případ, kde jsme měli mimořádné štěstí. Předpokládejme, že zde bylo nedávno provedeno úplné čištění a halda je kompaktní. Program chvíli pokračoval ve svém běhu a provedl nějaké alokace paměti. Po provedení dostatečného množství alokací, systém pro správu paměti rozhodne, že je čas pro provedení čištění.

Nyní předpokládejme, že program od svého posledního úplného čistění haldy, nezapsal žádný z generačně starších objektů. Zapisoval pouze nově alokované objekty, kterým byla generace nastavena na 0. Pomocí tohoto předpokladu jsme se dostali do výborné situace, protože můžeme proces čistění spustit masivně.

Místo standardního čistění můžeme také předpokládat, že všechny starší objekty, to jsou objekty generace 1 a vyšší, jsou stále naživu. Nebo jejich dostatečný počet je naživu. Ještě navíc žádný z nich nebyl zapisován do haldy, to znamená, že zde nejsou žádné ukazatele od starších objektů na nové objekty. Tím pádem není potřeba kontrolovat tyto generačně starší objekty. Pak tedy budeme jako obvykle prohledávat od kořenů, avšak jen do té doby, dokud nenarazíme na ukazatel ukazující na nějaký ze starších objektů.

Budeme tedy prohledávat od kořenů a všechny ukazatele, které budou ukazovat na starý objekt, budeme ignorovat. Pro ostatní kořeny, které ukazují na generaci 0, budeme provádět normální prohledávání. Budeme následovat všechny ukazatele, na které narazíme. Všechny ukazatele, které neukazují na starší objekt, budeme následovat.

Když je proces hotov a máme navštíveny všechny živé objekty z generace 0, můžeme je označit za živé či mrtvé. Po provedení čistění této části paměti zůstanou starší objekty nedotčené.

Mnoho tříd vytváří dočasné proměnné pro jejich návratové hodnoty, dočasné řetězce, různé druhy pomocných tříd jako jsou enumerátory a další pomůcky. Tudíž víme, že většina mrtvého místa je mezi mladšími objekty.

Hledání pouze objektů generace 0, nám dává jednoduchou cestu, jak získat zpět většinu mrtvého místa pomocí prohledávání jen malé části haldy, kde jsou mladé objekty.

Bohužel štěstí nemáme skoro nikdy, protože vždy jsou nějaké starší objekty, které byly změněny tak, že ukazují na nové objekty. Tudíž není rozhodně dostatečné, abychom staré objekty pouze ignorovali.

Aby algoritmus uvedený výše opravdu pracoval, potřebujeme vědět, které starší objekty byly upraveny. K zapamatování těchto pozměněných objektů, použijeme datovou strukturu nazývanou *Card Table*. Pro udržení této datové struktury aktuální, generuje kompilátor řízeného kódu takzvané zapisovací bariéry (*write barriers*). Tyto dva pojmy jsou důležité k úspěšnému řešení Garbage Collecting založeného na generacích.

Card Table může být implementován různě. Nejjednodušší je, si jeho implementaci představit jako pole bitů. Každý bit reprezentuje část haldy, řekněme 128 bytů. Pokaždé, když program zapíše objekt na nějakou adresu, musí *write barrier* kód dopočítat, který ze 128 bytových kusů byl zapsán a nastavit odpovídající bit v *Card Table*.

Pomocí tohoto mechanismu můžeme upravit původní algoritmus. Jestliže budeme provádět čistění haldy, kde je generace 0, můžeme použít algoritmus, o kterém jsem psal výše (ignorování všech ukazatelů na starší generace). Jakmile je toto hotovo, musíme najít všechny ukazatele na objekty, které se nacházejí v části paměti, která byla označena v *Card Table* jako modifikovaná. Budeme s ní zacházet jako s kořenem. Jestliže uznáme ukazatele za správné, provedeme pouze čištění paměti, kde se nacházejí objekty z generace 0.

Tento přístup nemusí být účinný, pokud je *card table* pořád plný. Prakticky je upraveno jen poměrně několik ukazatelů ze starší generace. Takže tímto přístupem provedeme podstatnou úsporu času při práci s pamětí.

[10]

7 Automatický .NET wrapper

Řešení problému, jak udělat wrapper pro knihovnu k zařízení Phantom, se skládá ze dvou projektů.

Prvním projektem je `AutomaticDotNetWrapperForPhantom`, který realizuje řešení problémů analyzováním hlavičkových souborů, podle kterých byla slinkována knihovna, pro kterou vytváříme wrapper. Dále analyzuje samotnou knihovnu, z důvodu kontroly názvů metod v ní obsažených. Tato analýza se dělá, protože většina metod má v knihovně, na rozdíl od hlavičkových souborů, před svým názvem podtržítko a za názvem má zavináč a číslo.

Druhým projektem je `HDx`, který vlastně již jen zkompiluje výstup z projektu `AutomaticDotNetWrapperForPhantom` do `dll` knihovny, která je .NET wrapper ke knihovně k zařízení Phantom.

Z důvodu větší flexibility použití, je automatický wrapper navržen pouze pro knihovnu z nižší úrovně a řádně otestován. Wrapper HDAPI je dostačující pro všechny typy aplikací používajících haptiku.

7.1 AutomaticDotNetWrapperForPhantom

Tento projekt se skládá z 5 tříd a to jsou `FileIO.cs`, `Headers.cs`, `Parser.cs`, `Program.cs` a `Typy.cs`.

7.1.1 Headers

Headers je struktura, ve které se uchovávají informace o hlavičkových souborech a jejich obsahy. Ukládá se zde celá cesta, název a obsah hlavičkového souboru.

7.1.2 FileIO

Tato třída slouží pro načítání dat z disku do operační paměti a pro ukládání z operační paměti na disk. Pro načítání dat slouží třída `public bool ReadContents()`, která podle toho, jak je nastaven `bool` operátor `justOne`, načítá buď obsahy všech hlavičkových souborů, které se nacházejí ve zvolené složce, nebo se snaží načíst obsah knihovny `hd.dll`. Knihovnu se snaží

otevřít ze složky systému System32. Pokud k ní nemáme přístup, program očekává knihovnu nakopírovanou ve stejné složce, jako je on sám. Ke zjištění nejen názvů všech hlavičkových souborů v dané složce slouží metoda `private FileInfo[] DirectoryContents()`. Jakmile si program zjistí informace o všech hlavičkových souborech, načte je do paměti a uloží je do seznamu typu `Headers`.

7.1.3 Types

Types je struktura, ve které se ukládají informace o použitých typech ve wrapperu. Například v hlavičkových souborech jsou nadefinovány typy `HDuint, HDBoolean, HDulong` a automatický wrapper je musí převést do typu, který je opravdu myšlen. Pro tyto tři výše napsané typy jsou myšleny `uint, char, ulong`.

7.1.4 Parser

Tato třída analyzuje hlavičkové soubory, pomocí kterých byla knihovna vytvořena. Dále analyzuje samotnou dll knihovnu z důvodů zpřesnění názvů metod. Vytváří zdrojový kód wrapperu ke knihovně k zařízení Phantom. Jedna ze základních metod této třídy je `private void Parsuj(Headers header)`, ve které analyzujeme po řádcích zdrojový kód každého hlavičkového souboru. Komentáře jsou zanechávány v původním tvaru. Jakmile v nějakém řádku program narazí na zdrojový kód, který není komentář ani část komentáře, kód se pošle do metody `private string CodeModifier(string code)`. V této metodě se testuje, čím řádek začíná a podle toho se kód upraví. Pokud se ve zdrojovém kódu hlavičkového souboru narazí na metodu, využije se metoda `private string MethodWrap(string[] code)`, která volání metody přeloží do jazyka C# a zároveň před metodu přidá vstupní bod do knihovny. Ke zpřesnění názvu vstupního bodu je použita metoda `private string NameInDLL(string src)`, která v načtené `dll` knihovně k zařízení Phantom najde přesné znění vstupního bodu.

Například z volání metody, která vypadala takto:

```
HDAPI void HDAPIENTRY hdBeginFrame(HHD hHD);
```

Program převedl na:

```
[DllImport(DLLName, EntryPoint = "_hdBeginFrame@4")]
public static extern void  BeginFrame(uint hHD);
```

Komentáře, které jsou napsané v hlavičkových souborech, program automaticky převádí na styl, ze kterého Visual Studio 2008 umí vygenerovat XML soubor nápovědy pro programátory.

Metoda `private void Arrange()` uspořádává vygenerovaný kód wrapperu tak, aby se dal přeložit. V podstatě přidá na začátek používané systémové služby jméno pracovního místa, které je `OpenHapticsWrapper`, chybovou strukturu `public struct ErrorInfo`, název třídy `HDx`. Na konec souboru přidává ukončující závorky tak, aby vše sedělo a wrapper šel bez problému přeložit.

7.1.5 Program

Tato třída je hlavní třída programu, obsahuje vstupní bod programu `static void Main(string[] args)`. Nastavuje se zde cesta ke složce, která obsahuje hlavičkové soubory. Pokud se povede všechny tyto `*.h` soubory načíst, jsou jejich zdrojové kódy předány třídě `Parser`, kde se analyzují a převedou do kódu wrapperu, který je následně zapsán na disk s názvem `HDx.cs`. Dále se ještě na disk zapisuje soubor `typy.txt`, který obsahuje informace o změnách typů. Ten je užitečnou informací pro programátory, kteří budou využívat vygenerovaný wrapper.

7.2 HDx

Tento projekt obsahuje výstup, vygenerovaný wrapper z projektu `AutomaticDotNetWrapperForPhantom` jako soubor `hdx.cs`. Projekt `HDx` je svázán s projektem `AutomaticDotNetWrapperForPhantom`. Když se udělá build projektu `HDx`, tak se nejprve spustí projekt `AutomaticDotNetWrapperForPhantom`, zkopíruje se jeho výstup `HDx.cs` a teprve potom se spustí samotný build, který vytvoří dll knihovnu HDx.dll, která je wrapperem ke knihovně k zařízení Phantom.

8 Ukázkové úlohy

Tyto programy jsou vytvořeny pro ukázku, že automatický .NET wrapper funguje. Jsou totiž napsány v jazyce C#, s využitím vygenerovaného wrapperu.

Na začátku programu se musí inicializovat zařízení, které budeme používat. To se dělá příkazem `hdx.InitDevice()`. Dále můžeme kontrolovat, jestli nastala nějaká chyba. To uděláme tak, že získáme chybu pomocí příkazu `ErrorInfo error = hdx.GetError()`, získanou chybovou strukturu musíme zkontrolovat, jestli obsahuje chybu příkazem `hdx.DEVICE_ERROR(error)`. Pokud nastala chyba, vrací `true`, pokud ne tak `false`. Musí se také stanovit callback funkce (funkce se zpětným voláním), což uděláme pomocí příkazu `ulong hGravityWell = hdx.ScheduleAsynchronous(...,...,...)`. Tato funkce bude poskytovat síly zařízení. Jako první argument této metody je použití delegáta `gravityWellCallback(…)`, kterému jako argument dáme callback funkci. Jako druhý argument jsou data, která posíláme do callback funkce a jako třetí argument se udává priorita vykonávání. Pomocí příkazu `hdx.Enable(hdx.FORCE_OUTPUT)` povolíme posílání sil do zařízení a konečně příkazem `hdx.StartScheduler()` celý proces spustíme. Je vhodné pravidelně kontrolovat jestli callback byl proveden, je možné to napsat následovně:

```
while (!Console.KeyAvailable)
{
   if (hdx.WaitForCompletion(hGravityWell,hdx.WAIT_CHECK_STATUS) == 0)
      {
         Console.WriteLine("Press any key to quit.");
         Console.ReadKey();
         break;
      }
}
```

Při ukončování programu je nutné, aby byla provedena ukončovací rutina. To jsou tyto tři příkazy `hdx.StopScheduler()`, `hdx.Unschedule(hGravityWell)` a `hdx.DisableDevice(hHD)`. První slouží k zastavení plánovače, druhý k vyjmutí nastavení ze zařízení a třetí příkaz uvolní zařízení pro jiné programy.

Callback funkcí může mít program více. Jsou volány každý cyklus servomechanismu Phantoma. Uvnitř této funkce se musí nacházet „Haptic Frame“, který definuje místo, kde je garantován konzistentní stav zařízení. Tento rámec je uzavřen příkazy `hdx.BeginFrame()` a `hdx.EndFrame()`. Na začátku tohoto rámu je stav zařízení aktualizován a uložen pro použití v tomto rámu. Na konci rámu je nový stav, jako jsou síly, zapsán do zařízení. Volání získávající informace jako poslední pozici poskytuje uložený stav rámce [1].

8.1 HelloHapticDevice

Tato ukázková úloha vytváří silové pole, které přitahuje zařízení k jeho středu, pokud se budeme nacházet v jeho blízkosti.

8.2 FrctionlessPlane

Tato ukázka demonstruje, jak se dělá kontakt s nekonečnou rovinou. Rovina umožňuje proražení, pokud uživatel vykoná dostatečně velikou sílu proti rovině. Pokud uživatel prorazí na druhou stranu, tak se rovině pozmění síly a tím umožní uživateli působit na ní z druhé strany.

8.3 FrictionLessSphere

V tomto příkladu je ukázáno, jak vytvořit silový kontakt s koulí.

8.4 Vibration

Tento příklad ukazuje, jak vytvořit jednoduchý sinusový vibrační efekt zařízení. Amplituda a frekvence kmitání mohou být nastavovány za běhu programu.

8.5 ErrorHandling

Tento příklad ukazuje, jak správně pracovat s chybami zařízení Phantom. Pokud nastanou chyby, jsou automaticky vraceny přístrojem. Nicméně pořád je důležité mít k dispozici informaci o detailech chyby, aby mohla být simulace obnovena. V tomto programu je pevná vodorovná rovina, se kterou se bude přerušovat kontakt, když nastane chyba. Dále je zde ukázáno několik dalších chyb, které jsou typické.

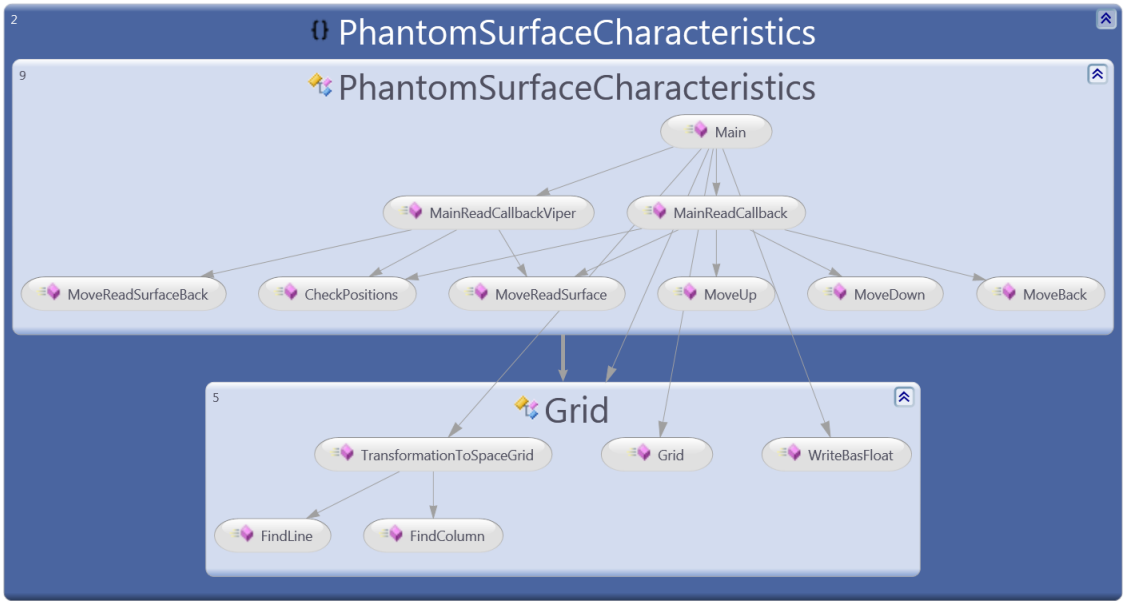
9 Načítání charakteristiky povrchů

Experiment automatického načítání charakteristiky povrchů pomocí zařízení Phantom se nachází v projektu `PhantomSurfaceCharacteristics`. Tento projekt je psán v programovacím jazyce C# a využívá .NET wrapper ke knihovně k zařízení Phantom.

Původní požadavek, kterého jsem se držel, byl, že předložíme před zařízení Phantom povrch, jehož charakteristiku chceme načíst. Phantom se bude sám pohybovat po povrchu a bude si ukládat souřadnice, na kterých se zrovna nachází.

Protože zařízení Phantom není k tomuto použití určeno, nemá možnost silového ovládání pera. Tudíž jsem musel provést zafixování pera (viz Obrázek 9).

Projekt tohoto experimentu se skládá ze dvou tříd, z pomocné třídy `Grid` a z hlavní třídy `PhantomSurfaceCharacteristics` (viz Obrázek 7).



Obrázek 7: Program pro načítání charakteristiky povrchů.

9.1 Propojení haptického zařízení s programem

V hlavní třídě ve vstupní metodě `Main` se inicializuje zařízení pomocí příkazu `hdx.InitDevice`. Dále musíme zkontrolovat, jestli inicializace proběhla v pořádku, nebo jestli nastala nějaká chyba. To můžeme udělat takto:

```
ErrorInfo error = hdx.GetError();
if (hdx.DEVICE_ERROR(error))
{
      Console.WriteLine("Failed to initialize haptic device\nPress any
            key to quit.\n");
      Console.ReadKey();
Environment.Exit(1);
}
```

Dále příkazem `hdx.Enable(hdx.FORCE_OUTPUT)` povolíme v zařízení používaní sil a pomocí příkazu `hdx.StartScheduler()` spustíme plánovač. Podobným postupem, jak již bylo ukázáno výše, provedeme kontrolu, jestli nenastala nějaká chyba.

Teď musíme vytvořit objekt typu `hdx.gravityWellCallback` a přiřadit do něj výkonnou metodu, kterou bude periodicky volat zařízení Phantom.

Z důvodu, použití neřízeného kódu uvnitř řízeného kódu, si musíme dát pozor, aby objekt, který vytváříme a přidáváme do něj výkonnou metodu, nebyl smazán Garbage Collectorem (viz kapitola 6). Nejjednodušší způsob jak tohoto dosáhnout je tento objekt nadefinovat globálně. Jestliže bychom pouze vytvořili objekt v argumentu metody, která nastavuje plánovač:

```
hdx.ScheduleAsynchronous(
    new hdx.gravityWellCallback(MainReadCallback),
    IntPtr.Zero,
    (ushort)hdx.DEFAULT_SCHEDULER_PRIORITY);
```

dojde při prvním čištění Garbage Collectorem (viz kapitola 6) ke smazání. Kdybychom objekt vytvořili ještě před posláním do metody tak, aby na něj zůstala reference, bylo by vše v pořádku. Protože překladač provádí optimalizaci kódu, výsledek bude naprosto stejný, jako když jsme objekt vytvářeli v argumentu metody. Fungující, správné a jednoduché je tento problémový objekt nadefinovat globálně, abychom na něj neztratili referenci. Lokálně ho nastavit a poslat ho do plánovače.

```
/// Schedule callback, which will then run at servoloop rates.
phantomCallback = new hdx.gravityWellCallback(MainReadCallback);
ulong Callback = hdx.ScheduleAsynchronous(phantomCallback,
IntPtr.Zero, (ushort)hdx.DEFAULT_SCHEDULER_PRIORITY);
```

Dále budeme periodicky kontrolovat, jestli callback stále běží.

```
while (true)
{
      // Periodically check if the gravity well callback has exited.
      if (hdx.WaitForCompletion(Callback, hdx.WAIT_CHECK_STATUS) == 0)
      {
            Console.WriteLine("The main scheduler callback has
                  exited.\n");
            break;
      }
}
```

Po skončení callbacku musíme zastavit plánovač a vyprázdnit naplánované úlohy. Teprve poté můžeme odpojit zařízení.

```
// Cleanup and shutdown the haptic device, cleanup all callbacks.
hdx.StopScheduler();
hdx.Unschedule(Callback);

// Disable the device.
hdx.DisableDevice(hHD);
```

Toto bylo základní nastavení, zprovoznění, naplánování a odpojení haptického zařízení. Nejednalo se však o výkonný algoritmus, ten se nachází v callback funkci (viz kapitola 9.2).

Na závěr je ve vstupní metodě důležité, poslání načtených dat (souřadnic) do transformačního mechanismu (viz kapitola 9.3) a následné uložení transformovaných dat do binárního souboru.

9.2 Načítací algoritmus - Callback funkce

Tato metoda je volána každý cyklus servomechanismu přístroje. Stará se o výpočet sil, kterými se ovládá pohyb ramena zařízení Phantom. Callback funkce také ze zařízení načítá jeho aktuální polohu, kterou si ukládá. Referenci na aktuální zařízení, se kterým pracujeme, dostaneme pomocí příkazu:

```
uint hHD = hdx.GetCurrentDevice();
```

Kód haptické callback funkce musí být uzavřený mezi příkazy `hdx.BeginFrame(hHD)` a `hdx.EndFrame(hHD)`.

Je vhodné, aby každá callback funkce kontrolovala, jestli nenastala nějaká chyba. Kontrolu můžeme provést jednoduchým dotazem:

```
if (hdx.DEVICE_ERROR(error = hdx.GetError()))
{
      uint numE = error.errorCode;
      Console.WriteLine(hdx.GetErrorString(numE));
      Console.WriteLine("Error during main scheduler callback");
}
```

Každý takovýto callback musí končit návratovou hodnotou, aby zařízení mělo informaci, že může pokračovat v provádění cyklu.

```
return hdx.CALLBACK_CONTINUE;
```

Návratová hodnota callback funkce `hdx.CALLBACK_DONE` ukončuje cyklus volání callback funkce.

9.2.1 MainReadCallback

Tento načítací algoritmus načítá charakteristiku povrchu řádku po řádce. Haptické zařízení načte řádku, která má stejnou souřadnici v ose Z, zvedne se o 10 centimetrů nahoru, posune se zpět do počáteční X-ové souřadnice, posune se o jeden milimetr v Z-ové souřadnici, sjede zpět k povrchu a načte další řádku.

Pomocí použití příkazu `hdx.GetFloatv ( hdx.CURRENT_POSITION, positionArray )` získáme ze zařízení jeho aktuální pozici v poli floatů, které má velikost 3. Toto pole si převedeme do třírozměrného vektoru, se kterým budeme dále pracovat. Následující algoritmus pomocí booleanovských proměnných rozhoduje, jaký pohyb zrovna zařízení vykonává a posílá aktuální pozici do odpovídajících metod. Tyto metody spočítají v závislosti na čase a aktuální pozici zařízení další pozici, kde by se měl Phantom nacházet. Do proměnné step je ukládána informace jakou řádku zrovna načítáme.

```
if (!reading && up)
{
      if (last)
      {
```

```
                coordinates.Add(position);
                last = false;
        }
        wellPos = MoveUp(step);//Move UP
        if (wellPos.Y >= endPosition.Y)
        {
                up = false;
                timer = 0;
        }
}
else if (!reading && !up && !down)
{
        wellPos = MoveBack(step); //Move back
        if (wellPos.X <= startPosition.X)
        {
                down = true;
                timer = 0;
                step++;
                if (!(step <= (startPosition.Z - endPosition.Z)))
                {
                        work = false;
                        end = true;
                        return hdx.CALLBACK_DONE;
                }
        }
}
else if (!reading && down)
{
        wellPos = MoveDown(step); //Move down
        if (wellPos.Y <= hightOfReading)
        {
                down = false;
                timer = 0;
                reading = true;
        }
}
else
{
        wellPos = MoveReadSurface(step); //Read surface
        if (wellPos.X > endPosition.X)
        {
                last = true;
                reading = false;
                up = true;
                timer = 0;
        }
}
```

O vypočítávání souřadnic při pohybu nahoru se stará metoda `MoveUp`:

```
static Vector3 MoveUp(int step)
{
        double[] instRatep = new double[3];
        hdx.GetDoublev(hdx.INSTANTANEOUS_UPDATE_RATE, instRatep);
        float instRate = (new Vector3((float)instRatep[0],
                (float)instRatep[1], (float)instRatep[2])).Length();
        timer += 1.0 / instRate;

        Vector3 ret = new Vector3(endPosition.X, hightOfReading +
```

```
            (float)(speedBack * timer), startPosition.Z - step);
        return ret;
}
```

Jako přírůstek času je použita převrácená hodnota okamžité frekvence. Návratová hodnota je třísložkový vektor, který udává pozici, kam se v tomto cyklu bude zařízení Phantom přesouvat. Jako X-ová hodnota návratového vektoru je vložena maximální X-ová hodnota načítacího obdélníku. Y-ová hodnota je vypočítána jako výška čtení, ke které se přičte násobek rychlosti a času. Z-ová pozice, která udává načítací řádek, je vypočítána jako posun od počáteční pozice v ose Z o číslo řádky, kterou načítáme.

Metoda `MoveBack` se stará o vypočítávání souřadnic při pohybu zpět. Návratová hodnota je třísložkový vektor. X-ová složka se počítá jako rozdíl maximální X-ové hodnoty načítacího obdélníku a násobku rychlosti pohybu s uplynulým časem tohoto pohybu. Y-ová složka je počáteční pozice v Y-ové ose zařízení. Tato hodnota je spočítána na začátku programu hned, jak jsou známy souřadnice načítacího obdélníku. K načtené Y-ové počáteční souřadnici načítacího obdélníku, což je levý vzdálenější roh od zařízení, je přičteno 100 milimetrů a tato souřadnice je označena jako počáteční pozice při načítání. Z-ová pozice je vypočítána jako posun o číslo řádky, kterou načítáme, od počáteční pozice v ose Z.

Metoda `MoveDown` se stará o vypočítávání souřadnic při pohybu dolů směrem k povrchu. X-ová hodnota návratového vektoru je nejmenší X-ová hodnota načítacího obdélníku. Y-ová hodnota se spočítá jako rozdíl počáteční pozice v Y-ové ose a násobku uplynulého času s rychlostí. Z-ová souřadnice návratového vektoru je počítána jako posun o číslo řádky, kterou načítáme, od počáteční pozice v ose Z.

Poslední metoda pro pohyb využitá v této callback funkci je metoda, která pohybuje ramenem zařízení Phantom po povrchu. Jmenuje se `MoveReadSurface`. X-ová hodnota její návratové hodnoty, která je třísložkový vektor, se spočítá jako součet X-ové složky vektoru počáteční pozice a násobku rychlosti čtení s uplynulým časem. Y-ová složka je nastavena na výšku načítání, tato výška je spočítána jako výška povrchu, od které odečteme 1 centimetr. Z-ová souřadnice je spočítána stejně jako u metod popsaných výše.

Zařízení má možnost pro každou osu nastavit různé síly. Když se používá metoda MoveReadSurface (tzn. právě se provádí sběr souřadnic), nastavuje se téměř maximální síla v ose Z. Toto nastavení pomáhá zpřesnit načítací řádky. Osa X je ponechána volnější a v ose Y působí jen velice malá síla, která se stará o to, aby rameno nezůstalo v některých místech viset ve vzduchu. Když se provádějí ostatní pohyby (nahoru, zpět a dolů), síly jsou v jednotlivých osách nastaveny na ±60% jejich maxima (viz. Kapitola 3.3). Když se při těchto pomocných pohybech v zařízení nastaví větší síly, tak z důvodu větší rychlosti než při sbírání souřadnic, dochází k rozvibrování zařízení. Toto rozvibrování generuje chybu, zařízení se restartuje a celé nastavení sil a pozice se do zařízení musí poslat znovu.

Ke konci výkonné části callback funkce musíme vypočítané souřadnice s nastavenými silami poslat do zařízení příkazem:

```
hdx.SetFloatv(hdx.CURRENT_FORCE,new float[]{force.X,force.Y,force.Z});
```

Až teď můžeme uzavřít haptické okénko příkazem `hdx.EndFrame(hHD)`.

9.2.2 MainReadCallbackViper

Tento načítací algoritmus načítá charakteristiku povrchu tak, že načte jednu řádku, posune se a zpětným pohybem načítá další řádku. Pak se opět posune a pokračuje stále stejným způsobem, dokud nemá všechny řádky načítacího obdélníku načtené. Nazývejme tento způsob načítání metodou zmije.

```
if (!reading)
{
      wellPos = MoveReadSurface(step); //Read surface normal
      if (wellPos.X > endPosition.X)
      {
            step++;
            if (!(step <= (startPosition.Z - endPosition.Z)))
            {
                  work = false;
                  end = true;
                  return hdx.CALLBACK_DONE;
            }
            last = true;
            reading = true;
            timer = 0;
      }
}
else
{
      wellPos = MoveReadSurfaceBack(step); //Read surface back
```

```
        if (wellPos.X < startPosition.X)
        {
                reading = false;
                step++;
                if (!(step <= (startPosition.Z - endPosition.Z)))
                {
                        work = false;
                        end = true;
                        return hdx.CALLBACK_DONE;
                }
                timer = 0;
        }
}
```

Základní chování callback funkce je stejné, jako je již popsáno v kapitole 9.2.1. Metoda MoveReadSurface (viz kapitola 9.2.1) načítá povrch zleva doprava.

Metoda `MoveReadSurfaceBack` načítá charakteristiku povrchu zprava doleva. Návratová hodnota je třísložkový vektor, jehož X-ová složka je spočítána jako rozdíl maximální hodnoty X-ové souřadnice načítacího obdélníku a násobku uplynulého času s rychlostí načítání. Y-ová hodnota je výška načítání (viz kapitola 9.2.1) a Z-ová souřadnice návratového vektoru je počítána jako posun od počáteční pozice v ose Z o číslo řádky.

9.3 Algoritmus pro transformování načtených dat

Načtená data jsou chaotická, takže musejí být převedena do stavu, ze kterého je možno je graficky a hapticky vykreslit. K této transformaci slouží metoda `TransformationToSpaceGrid`, která má jako svůj první argument `List` třísložkových vektorů, jako druhý argument souřadnice počátku načítání a jako třetí argument souřadnice konce načítání. Jsou to vlastně souřadnice levého bližšího rohu a pravého vzdálenějšího rohu načítacího obdélníku.

Algoritmus tedy z chaotických dat udělá prostorovou síť, která má v osách X a Z mezi body vždy jednotkovou vzdálenost. Na ose Y je výška povrchu v určitém bodě.

```
for (int j = (int)startPosition.Z; j >= (int)endPosition.Z; j--) //z-
axis
{
        lineNumber = j;
        List<Vector3> line = coordinates.FindAll(FindLine);
        for (int i = (int)startPosition.X; i <= ((int)endPosition.X);
                i++) //x-axis
        {
```

```
                columnNumber = i;
                average0.Clear();
                average0 = line.FindAll(FindColumn);
                if (average0.Count == 0)
                {
                      try
                      {
                            correctCoordinates.Add(new
                      Vector3(i,correctCoordinates[correctCoordinates.Count
                      - 1].Y,j));
                      }
                      catch (ArgumentOutOfRangeException)
                      {
                            Console.WriteLine("Could not find the first
                      coordinate of the object.");
                            correctCoordinates.Add(new Vector3((float)i,
                      startPosition.Y - 50f, (float)j));
                      }
                      continue;
                }
                double avr = 0;
                for (int l = 0; l < average0.Count; l++)
                {
                      avr += average0[l].Y;
                }
                aver.X = i;
                aver.Y = (float)(avr / average0.Count);
                aver.Z = j;
                correctCoordinates.Add(aver);
          }
}
```

V algoritmu nejdříve zaokrouhlíme Z-ovou souřadnici každého načteného bodu. Vyhledáme všechny body se stejnou Z-ovou souřadnicí a tím vlastně dostaneme řádek, který chceme dále zpracovávat. Tento řádek rozdělíme po jednotkách a pro každou celou hodnotu vyhledáme všechny body, které mají X-ovou souřadnici menší o 0.5, nebo větší o 0.5. Y-ové hodnoty těchto nalezených bodů sečteme a uděláme z nich průměr. Výsledná hodnota je Y-ová hodnota aktuálního bodu v aktuální řádce. Tento postup provedeme pro každý jednotkový bod řádky v každé nalezené řádce.

9.4 Vzniklé nepřesnosti a jejich redukce

Při načítání charakteristiky povrchu vznikají nepřesnosti. Umístění haptického bodu zařízení způsobuje, že načtené řádky nejsou úsečky, ale křivky. Řešením tohoto problému by mohlo být načítání na výšku (viz kapitola 9.4.1). Další problém, který způsobuje nepřesnost je špička pera. Volba jiné špičky nebo hrotu, třeba s malou kuličkou na konci, by mohla tento problém vyřešit. U zpětného načítání se kvůli natočení pera, rameno Phantoma zadrhává o povrch.

Jestliže zmenšíme rychlost načítání, zařízení načte více bodů, které charakterizují povrch, a tím zvýšíme přesnost. Upravením sil, které se používají pro ovládání pera při načítacím pohybu, by se také mohla zvýšit přesnost. Další možností zpřesnění je upravení algoritmu, který provádí transformaci do prostorové mřížky, aby dělal detailnější mřížku.

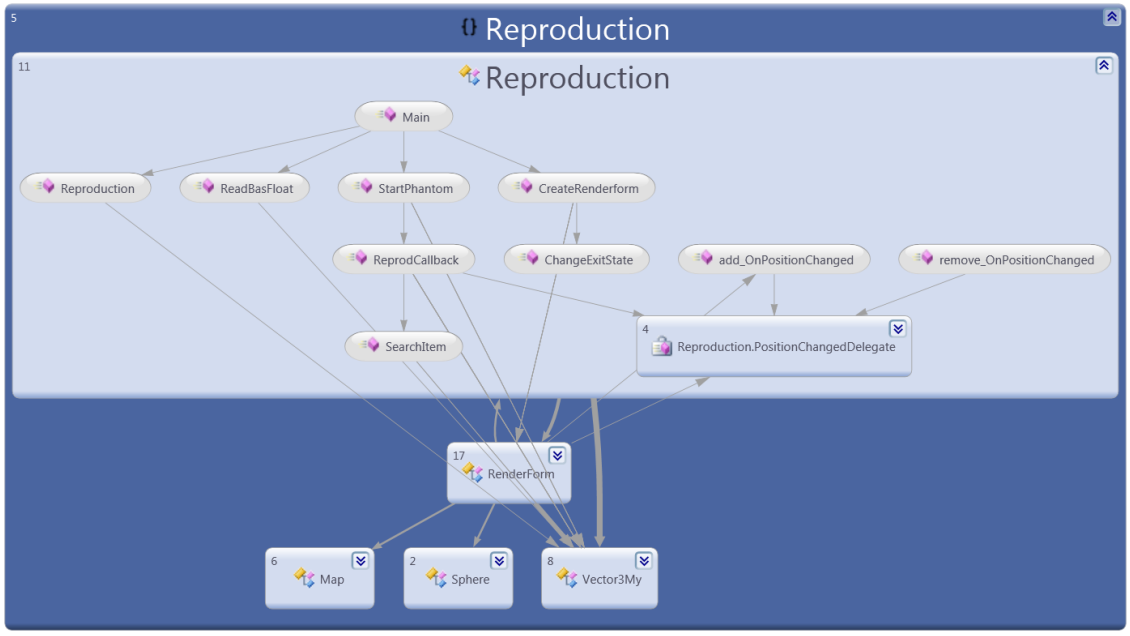
9.4.1 Možnosti jiných načítacích metod

Z důvodu zdokonalení načítání charakteristiky povrchů je možno implementovat jiné metody načítání, nebo tyto metody kombinovat.

- Bodové načítání – zařízení by vždy sjelo ze shora a našlo by výšku povrchu v určitém bodě. Tato metoda by zamezila zadrhávání pera o povrch, ale výrazně by prodloužila čas načítání.
- Načítání na výšku – povrch by byl otočen podle osy Z o 90° a načítán na výšku. Tento způsob by redukoval problém ohledně haptického bodu zařízení.
- Uživatelem řízený pohyb pera – při použití této načítací metody by přesnost načteného povrchu záležela pouze na uživateli.

10 Reprodukce načtené charakteristiky povrchů

Pro reprodukci charakteristiky načteného povrchu jsem napsal program, který se jmenuje Reproduction (viz Obrázek 8). Tento program se skládá ze dvou hlavních částí. Jedna část je zobrazení dat do haptiky a druhá část je zobrazení dat do grafiky jako výšková mapa. Grafická část programu je spuštěna ve svém vlastním vlákně.



Obrázek 8: Program pro reprodukci charakteristiky povrchů

10.1 Haptické zobrazení charakteristiky načteného povrchu

Inicializace a nastavení haptického zařízení jsou stejné, jako je popsáno v kapitole 9. Výkonný algoritmus se nachází v callback funkci. Tato metoda má k dispozici data načtené charakteristiky povrchu, která jsou přečtená z binárního souboru a aktuální polohu haptického bodu zařízení (viz kapitola 3.1). Z těchto dat pomocí bilineární interpolace vypočítáme, jaká je výška povrchu Y v souřadnicích X, Z, kde se právě nacházíme. Přičteme 10 milimetrů, abychom mohli porovnávat, jak moc se podobá reprodukovaná charakteristika charakteristice reálného povrchu.

```
lh = SearchItem((int)position.X, (int)position.Z);
ph = SearchItem((int)position.X + 1, (int)position.Z);
ld = SearchItem((int)position.X, (int)position.Z + 1);
```

```
pd = SearchItem((int)position.X + 1, (int)position.Z + 1);

float b = (pd.Y - ld.Y) * (position.X % 1) + ld.Y;
float c = (ph.Y - lh.Y) * (position.X % 1) + lh.Y;

wellPos = (b - c) * (position.Z % 1) + c;
wellPos = wellPos + 10;
```

Proměnné `lh, ph, ld` a `pd` jsou tříslożkové vektory, které udávají vždy levý horní, pravý horní, levý dolní a pravý dolní bod od pozice, pro kterou počítáme výšku povrchu.

Metoda `SearchItem` vybírá ze seznamu bodů prostorové mřížky ten správný. Pokud se tam nenachází, znamená to, že se haptický bod zařízení nachází mimo načtený povrch.

```
try
{
      if (minX >= x || x >= maxX) return new Vector3My(0, -100, 0);
      return coordinates[coordinates.Count - (int)((z - minZ) *
            lineLenght - (x - minX))];
}
catch (System.ArgumentOutOfRangeException)
{
      return new Vector3My(0, -100, 0);
}
```

Vypočítaná hodnota je výška povrchu v X, Z souřadnicích bodu, kde se zrovna nacházíme. Odečtením vypočítané hodnoty a Y-ové aktuální pozice haptického bodu získáme vzdálenost. Pro získání síly v tomto bodě vynásobíme vypočítanou vzdálenost s konstantou tuhosti. Všechny síly, které jsou záporné, vynulujeme. Takto vypočítanou sílu pošleme do zařízení:

```
/* Send the force to the device. */
hdx.SetFloatv(hdx.CURRENT_FORCE, new float[] { 0, force.Y, 0 });
```

10.2 Grafické zobrazení charakteristiky načteného povrchu

Charakteristika načteného povrchu je vykreslována jako výšková mapa. Pro vykreslení se používá `VertexBuffer` pro vrcholy a `IndexBuffer` pro jejich indexy. Pro samotné vykreslení se používá `TriangleList`.

Jsou zde implementovány dva způsoby zobrazení. První je zobrazení s texturou písku a druhé zobrazení je vrstevnicové, kde každá vrstevnice má jinou barvu.

11 Načtené vzorky

Reprodukce načtené charakteristiky do grafiky a haptiky je identická. Reálnost reprodukce závisí na tom, jak dobře probíhalo načítání.

- Dvě pravítka – Velice reálná reprodukce (viz Obrázek 10-12). Chyby při načítání dělaly šikmo položené hrany (vůči načítacímu směru).
- Jedno pravítko – Relativně reálná reprodukce (viz Obrázek 13-15). Problémy při načítání způsobovaly ostré hrany, na kterých se hrot pera zasekl a pak poskočil.
- Podložka pod myš – Jedna z nejlepších reprodukcí (viz Obrázek 16-18). Malá chyba vznikala, když pero Phantoma načítalo data šikmo nahoru (vůči načítacímu směru).
- Létající talíř (frisbee) – Špatná reprodukce (viz Obrázek 19-21) způsobená poskakováním pera po šikmých hranách. Při použití metody zmije se zobrazení ještě zhoršilo, hrot pera se zadrhával o povrch (viz Obrázek 22-23).
- Krabička na šipky – Výstupky na povrchu krabičky byly minimální, takže považuji tuto reprodukci za poměrně dobrou (viz Obrázek 24-26). Pomocí metody zmije se i zde na minimálních změnách povrchu hrot pera zadrhával (viz Obrázek 27-28).
- Víčko Nescafé – Špatná reprodukce, způsobená poskakováním pera po písmenech nápisu (viz Obrázek 29-31).
- Kancelářská sponka – Velice věrohodná reprodukce (viz Obrázek 32-34). Metoda zmije zde nešla použít z důvodu naprostého zaseknutí hrotu pera o sponku.

12 Závěr

Požadovaným úkolem bylo napsat co nejvíce automatický wrapper ke knihovně k zařízení Phantom a jeho funkčnost demonstrovat na ukázkových úlohách, prozkoumat možnost využití zařízení Phantom pro načítání charakteristiky povrchů a její následné reprodukce.

Automatické generování wrapperu je funkční. Pro nové verze knihovny by měl s minimálními úpravami fungovat bezproblémově. Na ukázkových úlohách je předvedeno, jak se jednoduše s vygenerovaným wrapperem pracuje a jak se má postupovat při vytváření programů pro haptické zařízení Phantom.

Experiment načítání charakteristik povrchů a jejich reprodukce vyšel lépe než jsem očekával. Úplně ideálně ale také nevyšel, což je pravděpodobně způsobeno tím, že zařízení není pro tento způsob použití uzpůsobené. Za hlavní nedostatek považuji uchycení pera na rameni zařízení, špičku pera a umístění haptického bodu zařízení. Zejména tyto tři aspekty způsobují nejvíce nepřesností při načítání charakteristik.

Během experimentování jsem dospěl k názoru, že zařízení je schopné načítat charakteristiky povrchů. Zmenšením rychlosti načítání, upravením načítacích sil a transformačního algoritmu načtených dat lze dosáhnout přesnější charakteristiky povrchu. Zvýšením rychlosti pomocných pohybů lze zkrátit čas, který je potřeba k získání dat. Při implementování možnosti načítání do nějaké aplikace doporučuji zaměřit se vždy jen na úzký okruh typů povrchů a využít poznatky z této práce.

Přehled zkratek

API	- (Application Programming Interface), rozhraní pro programování aplikací.
OpenGL	- (Open Graphic Library), průmyslový standart specifikující multiplatformní rozhraní pro tvorbu aplikací počítačové grafiky. [11]
ANSI	- (American National Standards Institute), americká standardizační organizace.[12] http://www.ansi.org/

Seznam použitých zdrojů

[1] SensAble Technologies. *OpenHaptics Toolkit : Programmer`s Guide*. version 3.0. USA : [s.n.], 2008. 194 s.

[2] SensAble Technologies. *OpenHaptics Toolkit : API Reference Manual*. version 3.0. USA : [s.n.], 2008. 132 s.

[3] SensAble Technologies. *SensAble technologies* [online]. 2010 [cit. 2010-02-24]. PHANTOM Omni® Haptic Device. Dostupné z WWW: <http://www.sensable.com/haptic-phantom-omni.htm>.

[4] *SensAble technologies* [online]. 2010 [cit. 2010-04-30]. OpenHaptics Toolkit. Dostupné z WWW: <http://www.sensable.com/products-openhaptics-toolkit.htm>.

[5] SensAble Technologies. *SensAble technologies* [online]. 2010 [cit. 2010-05-01]. Haptic Device API (HDAPI). Dostupné z WWW: <http://www.sensable.com/openhaptics-toolkit-hdapi.htm>.

[6] SensAble Technologies. *SensAble technologies* [online]. 2010 [cit. 2010-05-01]. Haptic Library API (HLAPI). Dostupné z WWW: <http://www.sensable.com/openhaptics-toolkit-hlapi.htm>.

[7] Microsoft Corporation. *.NET Framework Class Library* [online]. 2010 [cit. 2010-05-01]. Marshal Class. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/system.runtime.interopservices.marshal.aspx>.

[8] FAJFR, Stanislav. Architektura Microsoft .NET Framework. *Programujte* [online]. 12.6.2006, 3. díl, [cit. 2010-05-02]. Dostupný z WWW: <http://programujte.com/?akce=clanek&cl=2006060902-architektura-microsoft-net-framework-%2596-3-dil>.

[9] Microsoft Corporation. *.NET Framework 4* [online]. 2010 [cit. 2010-05-02]. Fundamentals of Garbage Collection. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ee787088.aspx>.

[10] Microsoft Corporation. *.NET Development (General) Technical Articles* [online]. 2010 [cit. 2010-05-02]. Garbage Collector Basics and Performance Hints. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ms973837.aspx#dotnetgcbasics_topic3>.

[11] Khronos Group. *OpenGL* [online]. 2010 [cit. 2010-05-03]. OpenGL Overview. Dostupné z WWW: <http://www.opengl.org/about/overview/>.

[12] *American Nation Standards Institute* [online]. 2010 [cit. 2010-05-03]. Introduction to ANSI. Dostupné z WWW: <http://www.ansi.org/about_ansi/introduction/introduction.aspx?menuid=1>.

Přílohy

A. Uživatelská dokumentace k programům

A.1 AutomaticDotNetWrapperForPhantom

Ve stejné složce jako je program, musí být složka, která se jmenuje `include` a v ní všechny hlavičkové soubory ke knihovně `hd.dll`. Program ke svému úspěšnému běhu potřebuje ještě knihovnu `hd.dll`. Pokusí se ji tedy najít ve složce operačního systému. Když se mu ji nepovede najít, ať už to je z důvodu uživatelský práv, nebo tam není, je potřeba, aby uživatel knihovnu vložil do stejné složky jako je program. Po splnění těchto podmínek stačí spustit program, který vygeneruje wrapper ke knihovně `hd.dll` k zařízení Phantom. Úspěšně vygenerovaný soubor se bude nacházet ve složce `include`, v kořenovém adresáři programu a bude se jmenovat `hdx.cs`. Generuje se ještě jeden soubor, který se jmenuje `hdtypy.txt`. V tomto souboru jsou informace o konverzi typů provedené při wrappování.

K tomuto programu je vytvořen ještě jeden projekt, který z vygenerovaného zdrojového kódu vytvoří knihovnu. Tyto dva projekty jsou v jednom Solution Microsoft Visual Studia 2008. Toto řešení je nastavené tak, že stačí přeložit projekt `HDx` a než se provede samotný překlad tohoto projektu, který vytváří dll knihovnu, spustí se překlad projektu `AutomaticDotNetWrapperForPhantom` a zkopíruje se jeho výstup do zdrojového kódu projektu `HDx`.

A.2 PhantomSurfaceCharacteristics

Po spuštění program vyzve k zadání jména výstupního souboru. Po zadání jména výstupního souboru program vyzve k zadání načítacího obdélníku. Ten uživatel zadá pomocí pera na zařízení Phantom, pomocí černého tlačítka na peru se zadává levý bližší roh načítacího obdélníku a pomocí bílého tlačítka se zadává pravý vzdálenější roh načítacího obdélníku. Po správném zadání se rameno Phantoma přesune do polohy počáteční souřadnice načítacího obdélníku a čeká, než uživatel zmáčkne mezerník. Tím se spustí proces načítání, na příkazové řádce se vypisuje, kolikátá řádka z kolika se právě

načítá. Po načtení charakteristiky povrchu program ještě chvilku zpracovává načtené souřadnice. Samozřejmě informuje o všem, co zrovna dělá. Až program uloží načtená a zpracovaná data do binárního souboru, ještě jednou vypíše jméno výstupního souboru a může být ukončen.

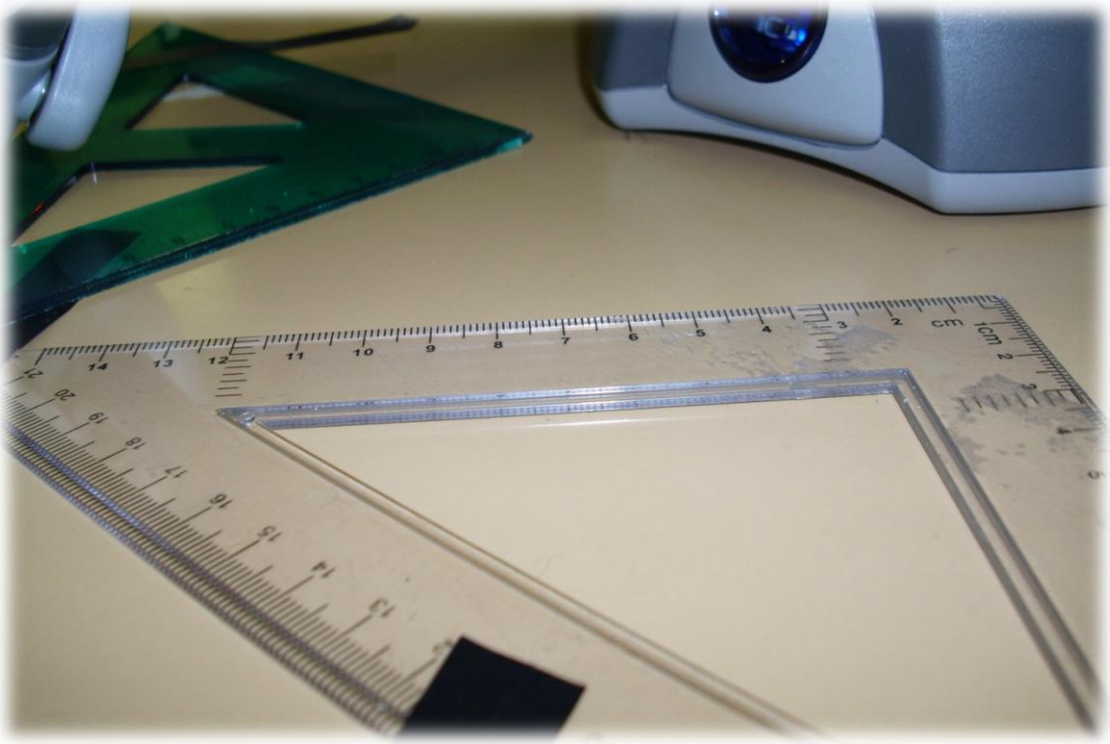
Program se kdykoliv v průběhu může ukončit pomocí klávesy `Q`. Výstupní soubor je po úspěšném dokončení běhu programu uložen v kořenovém adresáři programu.

A.3 Reproduction

Vstupní binární soubor s uloženými daty charakteristiky povrchu se zadává jako argument při spuštění programu. Program vykreslí data charakteristiky povrchu do haptiky a také je graficky vykreslí na obrazovku. Pohybováním perem haptického zařízení po povrchu bychom měli cítit zpětnou silovou vazbu. Na obrazovce je pozice haptického bodu Phantoma na povrchu označena kuličkou. Pomocí klávesy `F` můžeme změnit styl vykreslení povrchu. Jeden styl je s texturou písku a druhý styl je vrstevnicový. Pomocí myši můžeme natáčet kameru a pomocí kláves `W`, `S`, `A`, `D` s kamerou pohybovat. Klávesy `Q` a `E` přibližují a oddalují vykreslený povrch. Program se ukončí po stisknutí klávesy `Esc`.

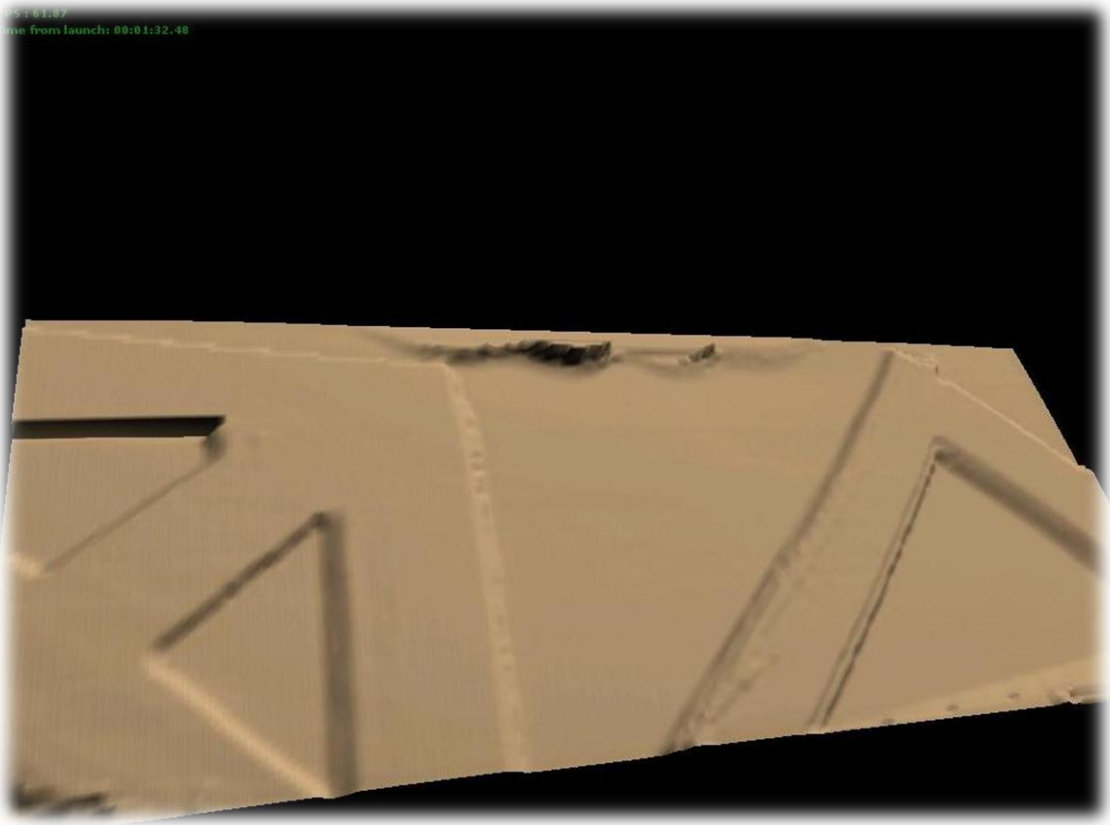
B. Obrázky

Obrázek 9: Zafixování pera zařízení Phantom Omni



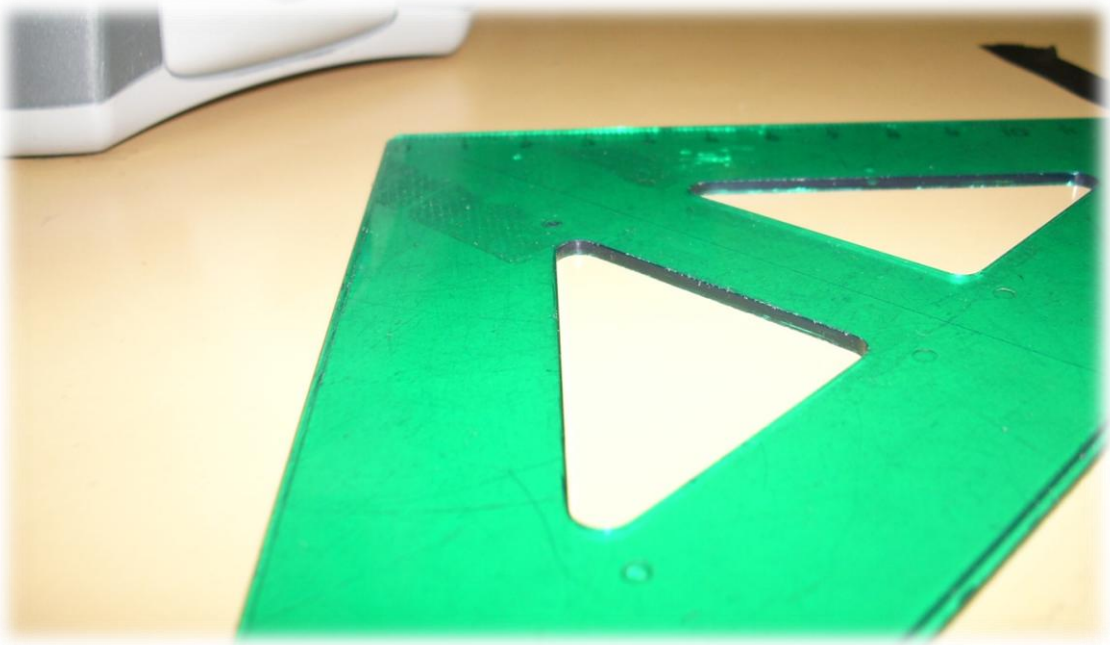
Obrázek 10: Dva trojúhelníky, reálný pohled.

Velikost snímané oblasti: 231 x 94 mm. Čas potřebný k získání dat: 40 min.



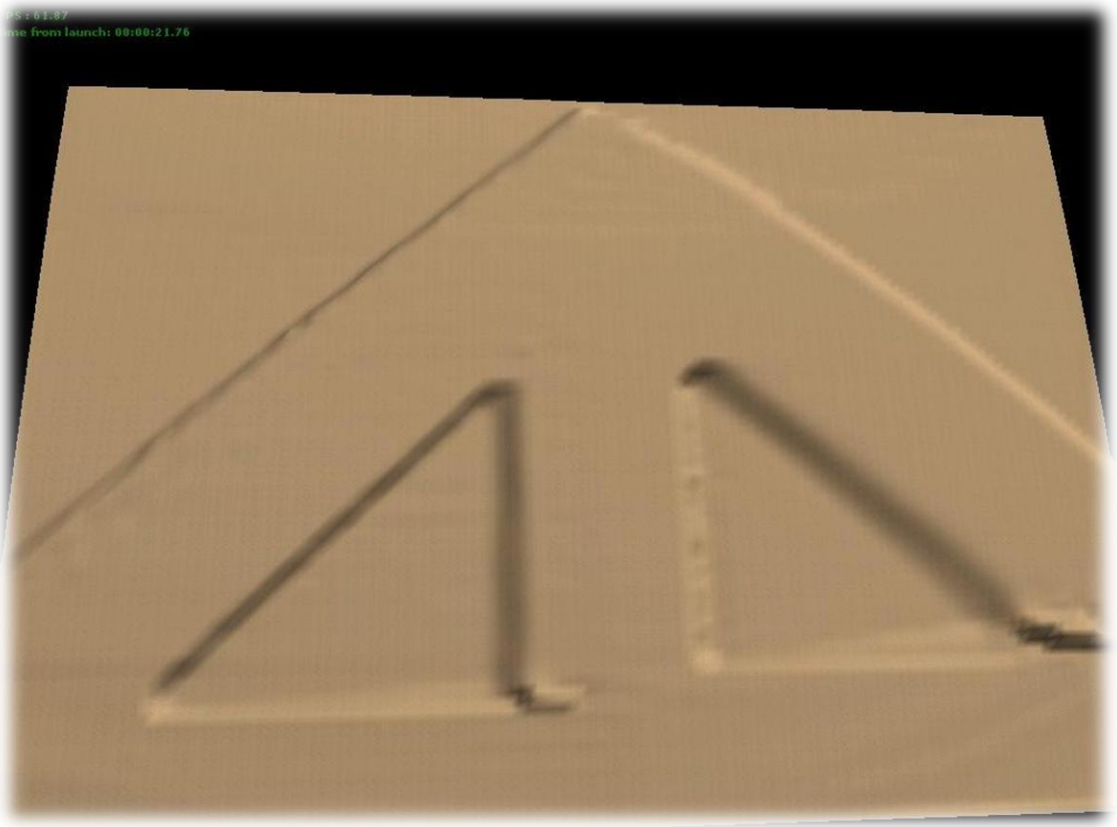
Obrázek 11: Dva trojúhelníky, pohled s texturou

Obrázek 12: Dva trojúhelníky, vrstevnicový pohled.



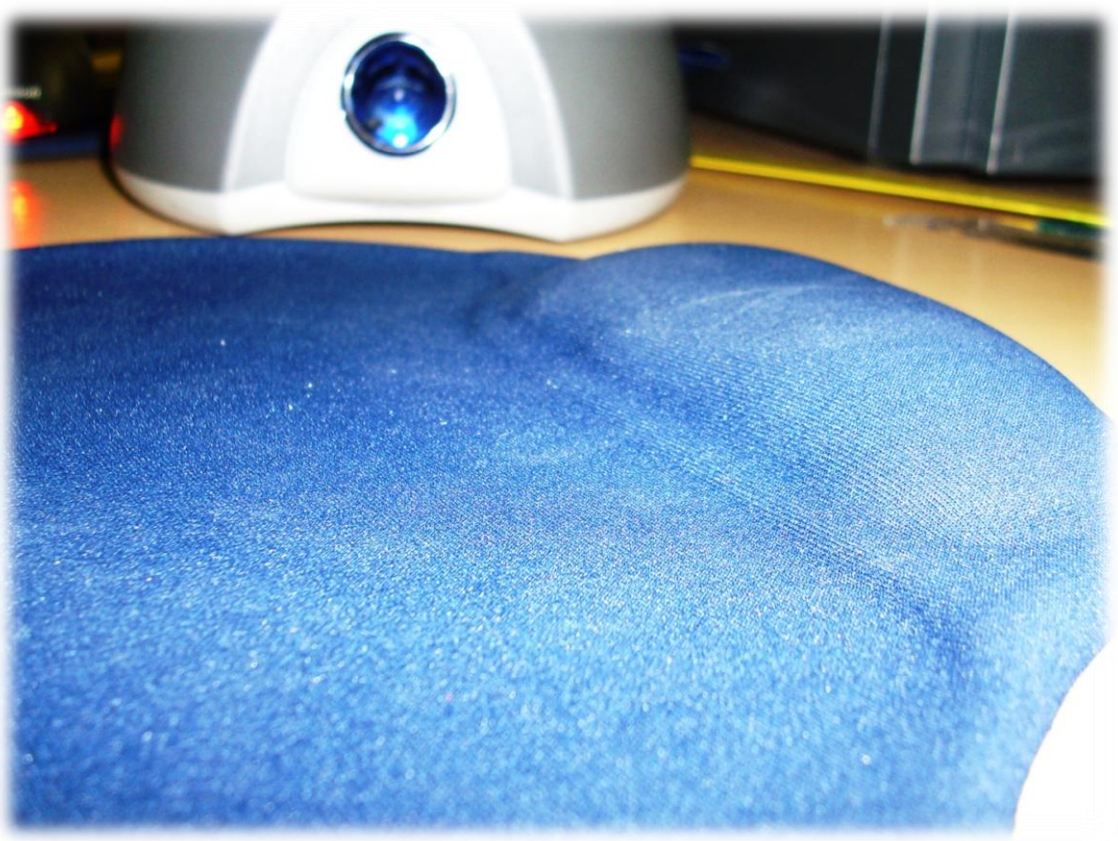
Obrázek 13: Jeden trojúhelník, reálný pohled.

Velikost snímané oblasti: 140 x 97 mm. Čas potřebný k získání dat: 24 min.



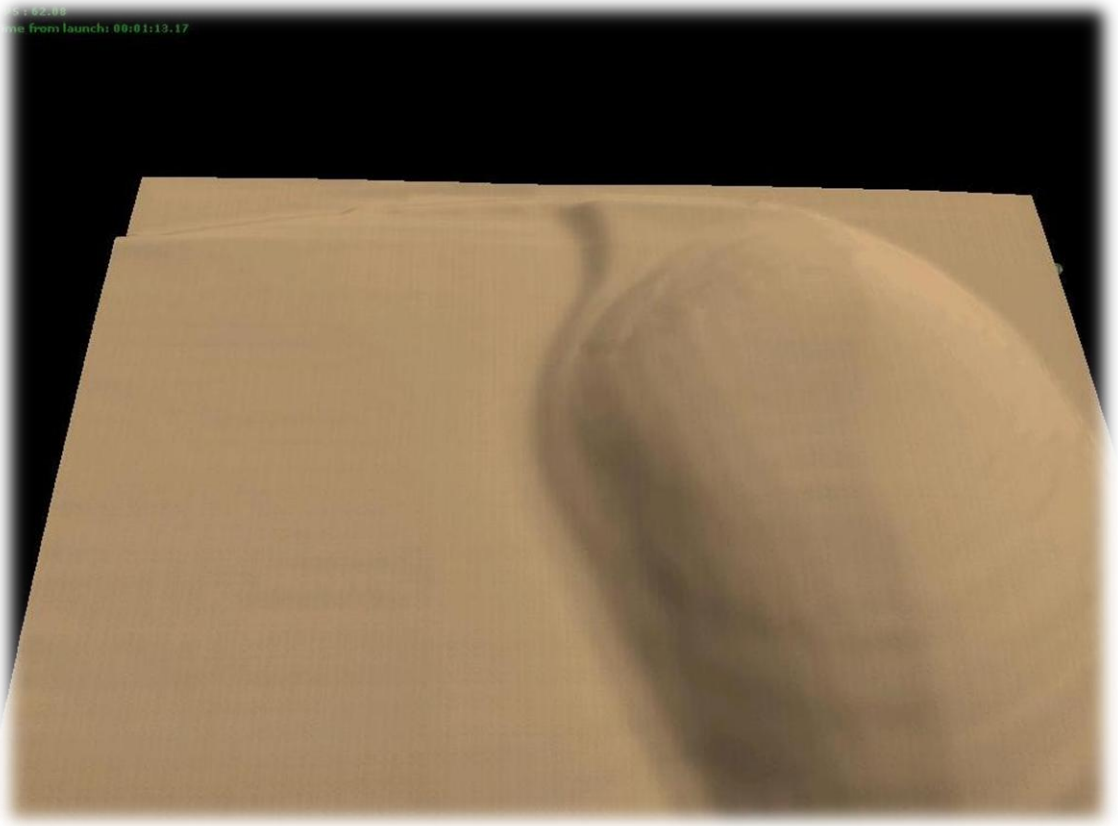
Obrázek 14: Jeden trojúhelník, pohled s texturou.

Obrázek 15: Jeden trojúhelník vrstevnicový pohled.

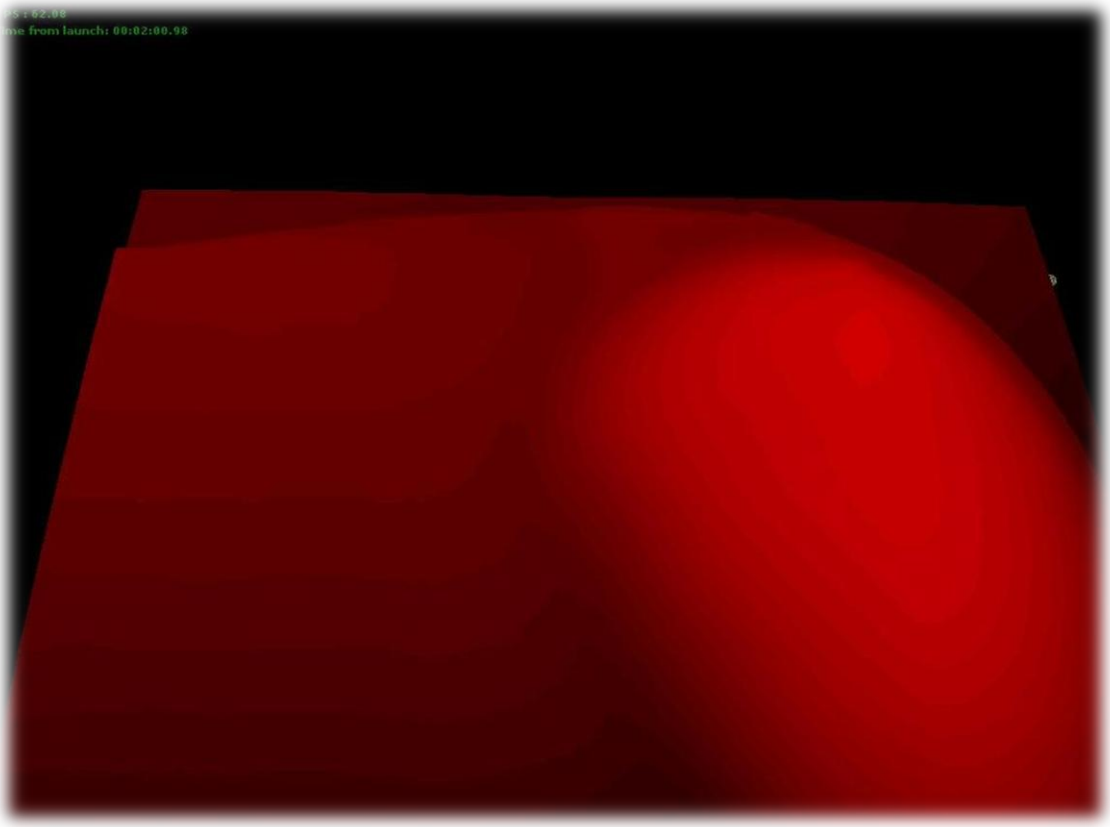


Obrázek 16: Podložka pod myš, reálný pohled.

Velikost snímané oblasti: 143 x 101 mm. Čas potřebný k získání dat: 26 min.



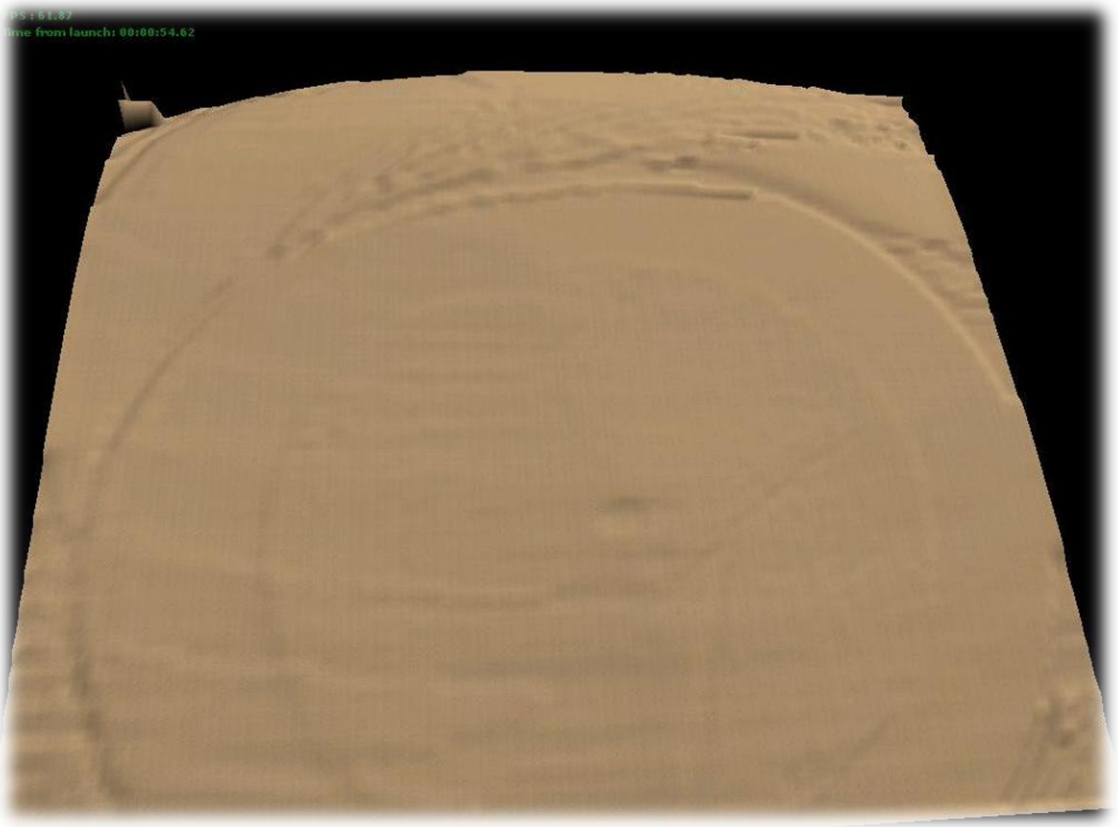
Obrázek 17: Podložka pod myš, pohled s texturou.



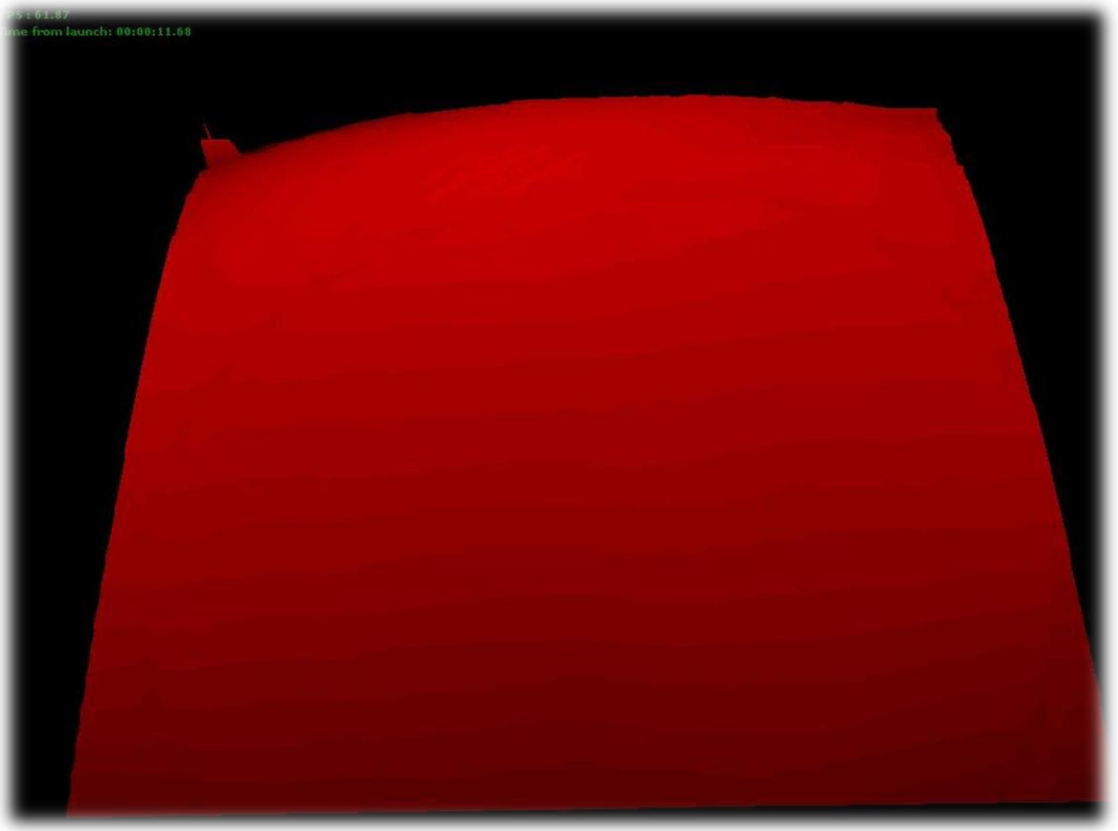
Obrázek 18: Podložka pod myš, vrstevnicový pohled.

Obrázek 19: Létající talíř (frisbee), reálný pohled.

Velikost snímané oblasti: 133 x 123 mm. Čas potřebný k získání dat: 30 min.

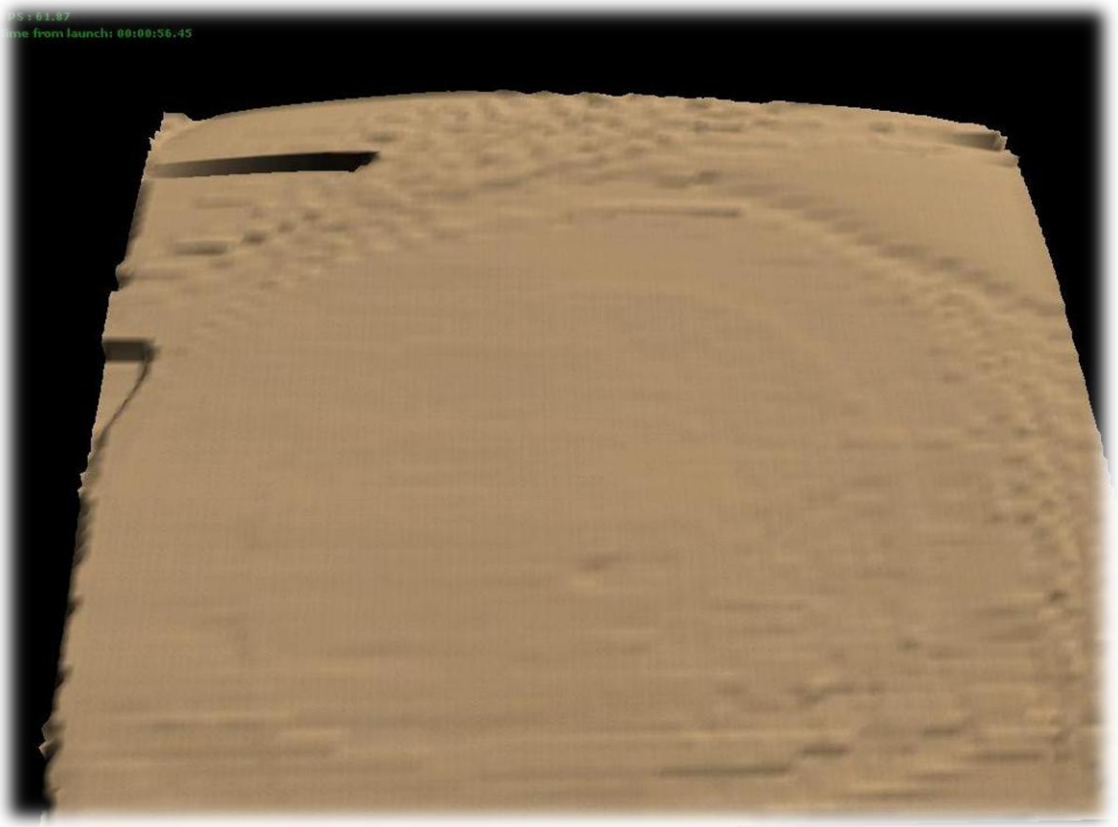


Obrázek 20: Létající talíř (frisbee), pohled s texturou.

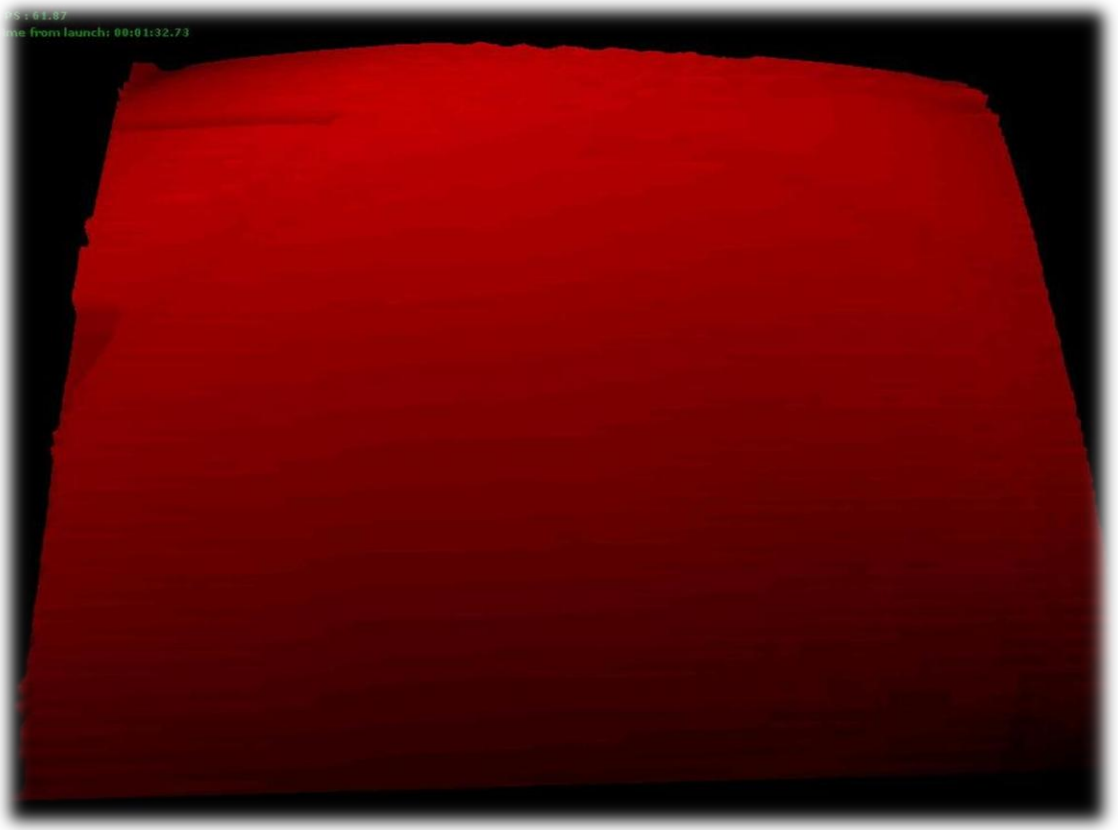


Obrázek 21: Létající talíř (frisbee), vrstevnicový pohled.

Velikost snímané oblasti: 134 x 106 mm. Čas potřebný k získání dat: 16 min.



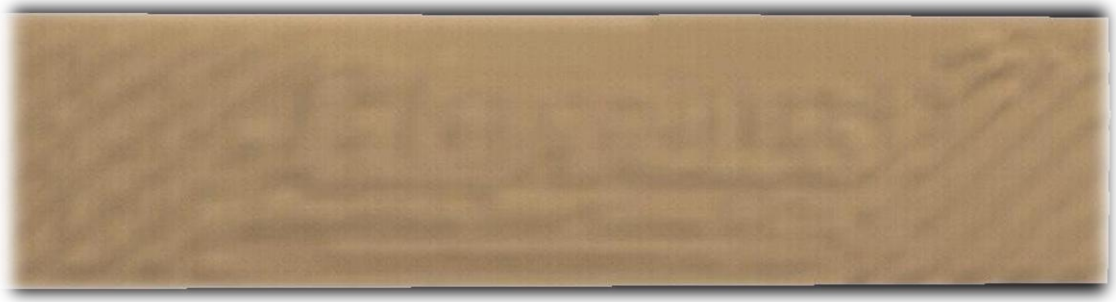
Obrázek 22: Létající talíř (frisbee), načteno metodu zmije, pohled s texturou.



Obrázek 23: Létající talíř (frisbee), načteno metodou zmije, vrstevnicový pohled.

Obrázek 24: Krabička na šipky, reálný pohled.

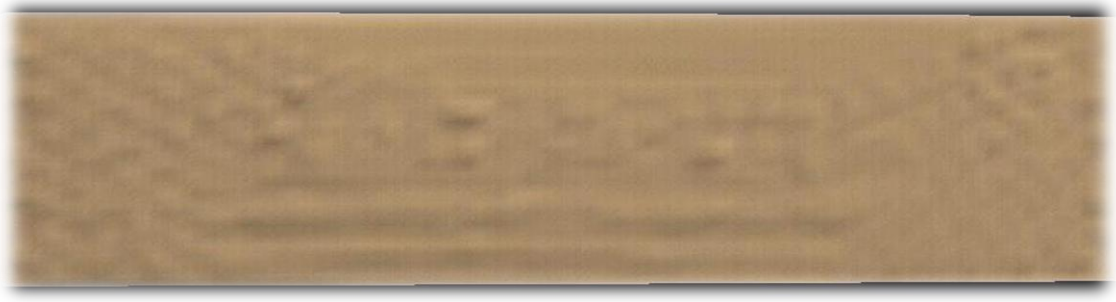
Velikost snímané oblasti: 113 x 28 mm. Čas potřebný k získání dat: 5 min.



Obrázek 25: Krabička na šipky, pohled s texturou.

Obrázek 26: Krabička na šipky, vrstevnicový pohled.

Velikost snímané oblasti: 115 x 28 mm. Čas potřebný k získání dat: 3 min.

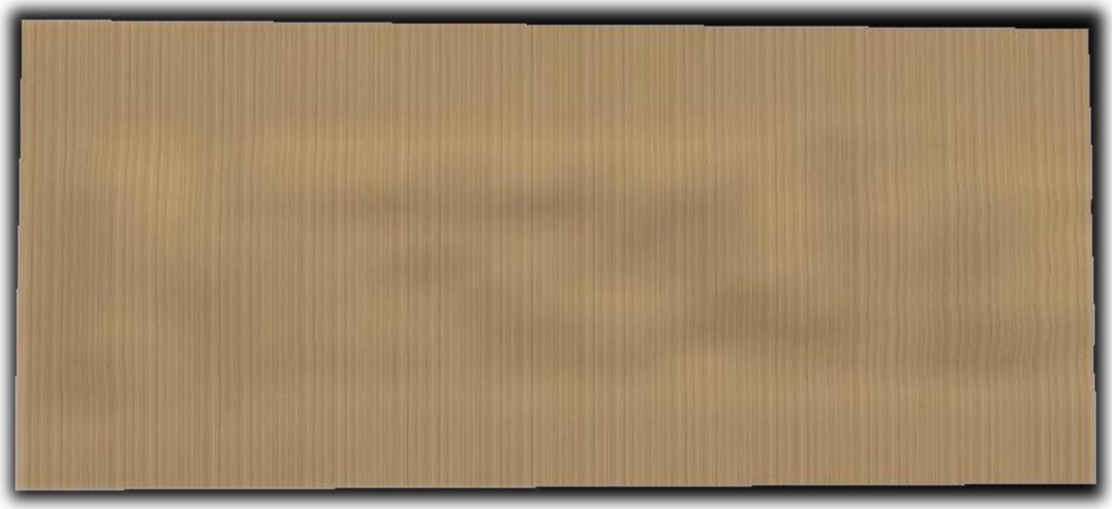


Obrázek 27: Krabička na šipky, načteno metodou zmije, pohled s texturou.

Obrázek 28: Krabička na šipky, načteno metodou zmije, vrstevnicový pohled.

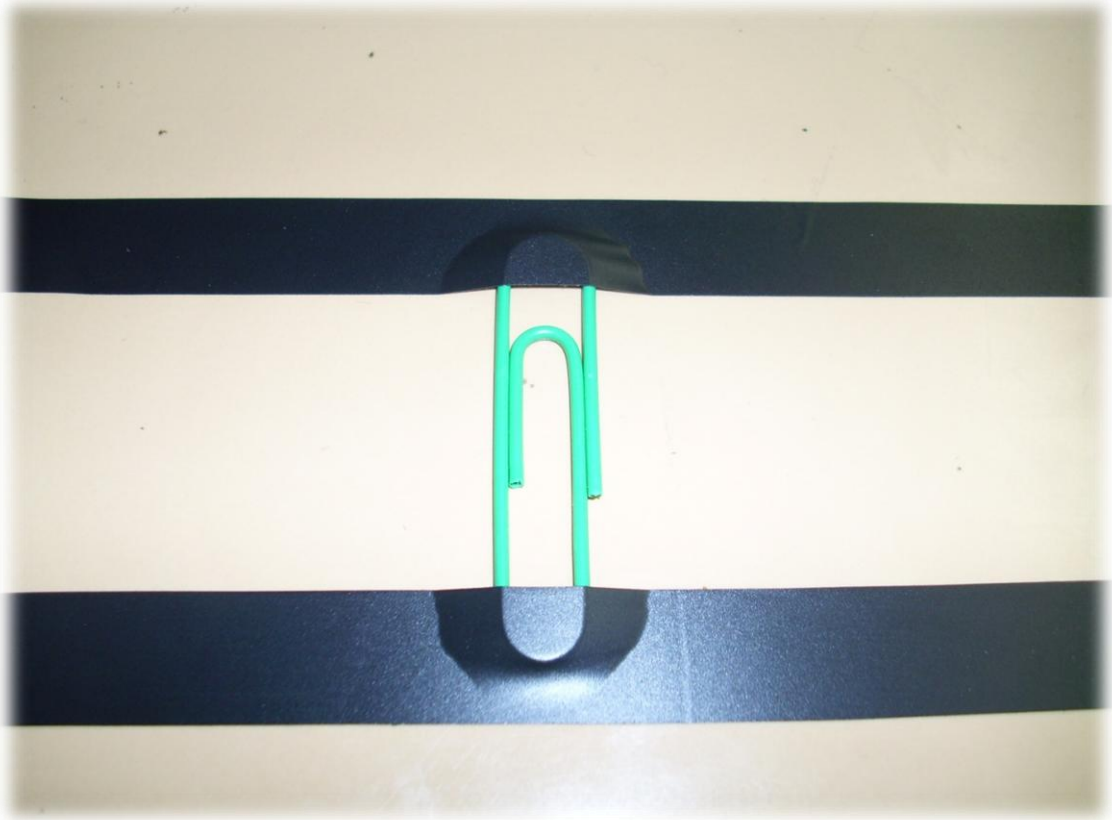
Obrázek 29: Víčko Nescafé, reálný pohled.

Velikost snímané oblasti: 36 x 16 mm. Čas potřebný k získání dat: 1 min.



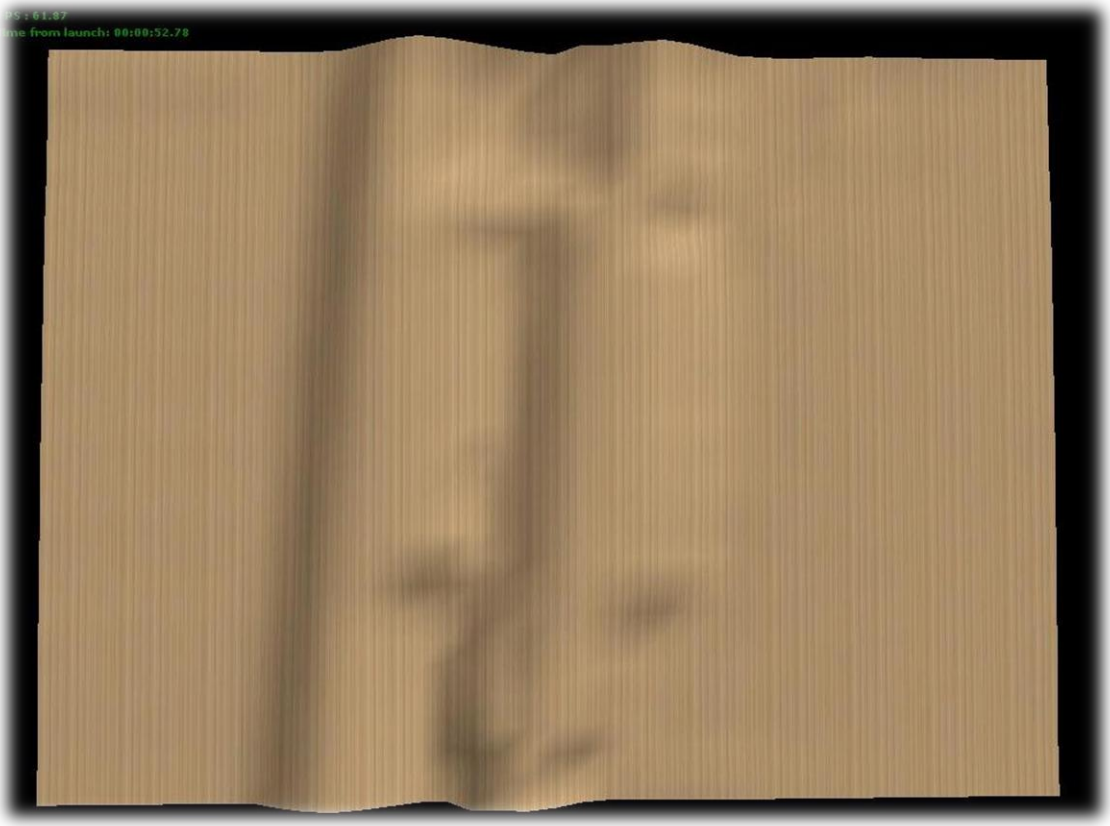
Obrázek 30: Víčko Nescafé, pohled s texturou.

Obrázek 31: Víčko Nescafé, vrstevnicový pohled.

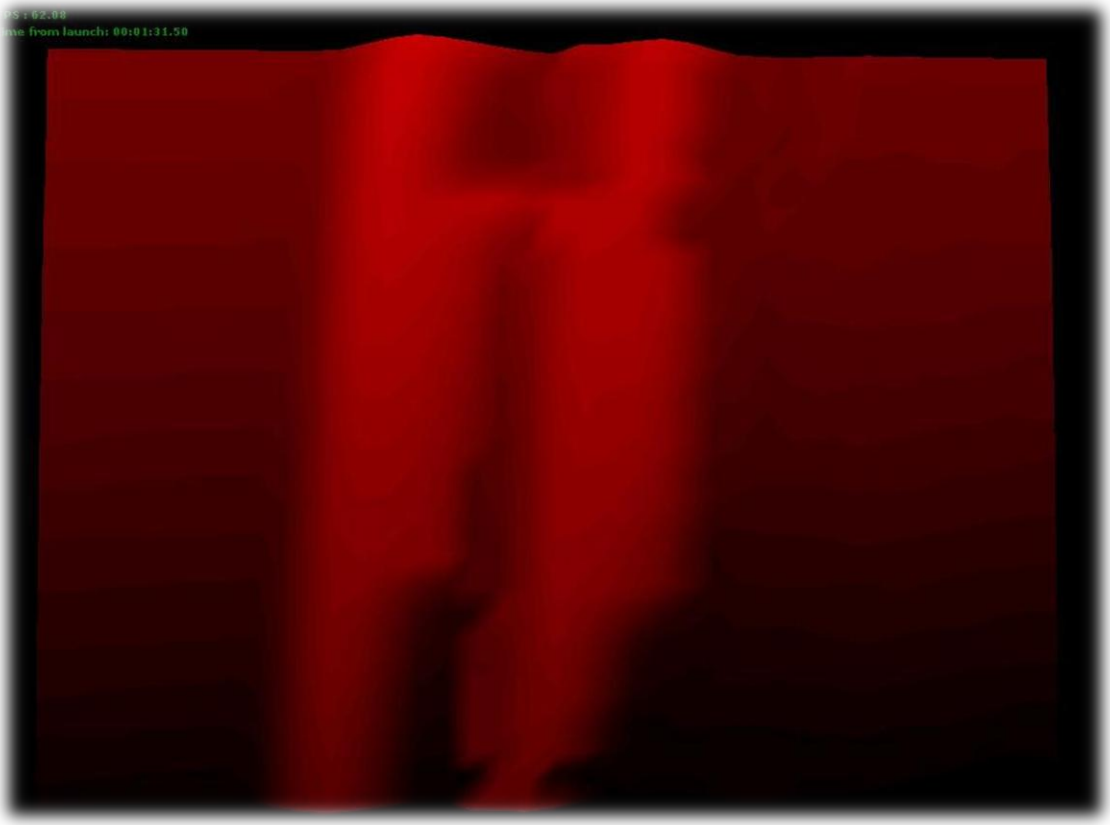


Obrázek 32: Kancelářská sponka, reálný pohled.

Velikost snímané oblasti: 39 x 29 mm. Čas potřebný k získání dat: 2 min.



Obrázek 33: Kancelářská sponka, pohled s texturou.



Obrázek 34: Kancelářská sponka, vrstevnicový pohled.

C Obsah CD

Na přiloženém médiu se nacházejí zdrojové soubory ke všem programům, zkompilované programy a tento dokument.

Na médiu se z důvodu omezené licence nenachází knihovna OpenHaptics Toolkit, ani dokumenty v ní obsažené. Tato knihovna je po registraci ke stažení na stránkách korporace SensAble http://www.sensable.com/.