České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra řídicí techniky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Martin Kováčik**

Studijní program: Otevřená informatika (magisterský)
Obor: Počítačové inženýrství

Název tématu: **Vizualizace a simulace technologie CNC obrábění**

Pokyny pro vypracování:

Úkolem této práce je navrhnout a implementovat modul pro simulaci práce CNC stroje. Jeho vstupem bude technologický program ve vnitřním tvaru (obsahuje programovanou posloupnost úkonů stroje), na jeho základě vytvoří a zobrazí 3D model hotového opracovaného výrobku a simulaci (animaci) procesu obrábění podle daného technologického programu.

1. Seznamte se s programovým rozhraním softwaru řídicích systémů MEFI řady CNC872.
2. Navrhněte vhodný princip pro vytvoření 3D modelu obrobku na základě informací dostupných v technologickém programu.
3. Navrhněte postup vytvoření animované 3D simulace postupu obrábění.
4. Pro zobrazování 3D dat použijte knihovnu OpenGL nebo Direct3D, v případě potřeby zvolte a použijte další knihovny a prostředky.
5. Bude-li to potřeba, navrhněte rozšíření programového rozhraní řídicího systému.
6. Implementujte modul jako plug-in pro řídicí systémy MEFI řady CNC872 a demonstrujte jeho funkci pro reálné technologické programy.

Seznam odborné literatury:

Dodá vedoucí práce

Vedoucí: Ing. Jiří Hoffmann

Platnost zadání: do konce letního semestru 2011/2012

L.S.

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Boris Šimák, CSc.
děkan

V Praze dne 10. 1. 2011

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze, dne 11.5.2017 ....
podpis

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra řídicí techniky

Diplomová práce

Vizualizace a simulace technologie CNC obrábění

Bc. Martin Kováčik

Vedoucí práce: Ing. Jiří Hoffmann

Studijní program: Otevřená informatika

Obor: Počítačové inženýrství

Květen 2011

Poděkování

Chtěl bych poděkovat Ing. Jiřímu Hoffmannovi za to, že se ujal vedení této diplomové práce a za jeho pomoc na jejím zhotovení. Poděkování také patří Ing. Petru Felkelovi Ph.D., který byl ochoten pro tuto práci dělat oponenta. Velký dík taktéž patří firmě MEFI, která poskytla námět i rady nezbytné pro zhotovení této diplomové práce.

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze, dne ……. …….
podpis

Abstrakt

Tato diplomová práce se zabývá návrhem a implementací modulu, který má za úkol simulovat a poté graficky znázornit práci CNC stroje. Modul je specificky navržen pro řídicí systém MEFI řady CNC872 a tak musí splňovat všechny požadavky tohoto systému. Tento modul ke své práci využívá externí grafické jádro, důležitou součástí této práce je tedy charakteristika a popis několik takovýchto grafických jader a posléze práce s nimi. Jedná se o grafická jádra Open CASCADE a ACIS.

Abstract

This diploma thesis describes design and implementation of the module, which aims to simulate and then represented graphically CNC machine work. The module is specific designed for MEFI controller series CNC872 and must comply all requirements of the system. This module for its work using external graphics core, so an important part of this work is characterization and description of some of these graphics cores and then work with them. These are a graphic cores Open CASCADE and ACIS.

Obsah

1. Úvod ... 3
1.1. Popis problému ... 3
1.2. Specifikace cílů ... 4
1.3. Dodatečné cíle ... 5
2. Rešerše modelářů ... 7
2.1. Existující jádra ... 7
2.2. Datové modely ... 8
2.3. Licence ... 9
2.4. Výběr jádra ... 10
2.5. Experiment ... 10
2.5.1. Zadání ... 10
2.5.2. Provedení ... 10
2.5.3. Výsledek experimentu ... 11
2.6. Závěr rešerše ... 11
3. ACIS ... 13
3.1. Architektura ACIS ... 14
3.2. Topologické uspořádání ACIS ... 14
3.3. Přehled podporovaných funkcí jádrem ACIS ... 16
3.3.1. 3D Modelování ... 16
3.3.2. 3D Správa modelu ... 16
3.3.3. 3D Vizualizace modelu ... 16
3.4. Přehled rozšíření pro modelářské jádro ACIS ... 16
3.4.1. 3D Deformable Modeling ... 16
3.4.2. Advanced Covering ... 17
3.4.3. Defeaturing ... 17
3.4.4. PHL V5 ... 17
4. Open CASCADE ... 19
4.1. Topologie Open CASCADE ... 19
4.2. Návaznost na geometrii ... 20
4.2.1. Vertex ... 21
4.2.2. Edge ... 22
4.2.3. Face ... 23
5. Model objektů COM ... 27
5.1. Životní cyklus objektu COM ... 28
5.2. Jak získat ukazatel rozhraní ... 29
6. Návrh provedení simulace ... 31
6.1. Vstupní data ... 31
6.1.1. Předpřipravené bloky programu ... 31
6.1.2. Souřadnice z interpolátoru ... 32
6.2. Zobrazení nástroje ... 32

6.3. Grafická reprezentace ... 33
6.3.1. Objemová reprezentace ... 34
6.4. Shrnutí návrhu simulace ... 34
7. Napojení modulu na systém ... 35
7.1. Získávání informací ze systému ... 35
8. Práce s Open CASCADE ... 37
8.1. Inicializace ... 37
8.2. Model dráhy nástroje ... 38
8.3. Model opracované plochy nástroje ... 39
8.4. Řez ... 40
8.5. Zobrazení ... 41
8.6. Import / Export ... 42
9. Prezentace a zhodnocení výsledků ... 43
9.1. Model dráhy nástroje ... 43
9.2. Model opracované plochy nástroje ... 45
9.3. Řez ... 45
9.4. Import / Export ... 48
10. Zhodnocení výsledků ... 49
10.1. Jakým způsobem pokračovat? ... 50
10.2. Přínos této práce ... 50
11. Závěr ... 51

A. Použitá literatura ... 53
B. Seznam obrázků ... 55
C. Seznam tabulek ... 57
D. Seznam zkratek ... 59
E. Obsah přiloženého DVD ... 61

1. Úvod

Cílem této diplomové práce je naimplementovat softwarový modul do řídicího systému, který by dokázal simulovat práci stroje a výsledek zobrazit. Už z této jednoduché věty je vidět, že tento modul se musí věnovat dvěma oblastem, totiž simulaci a vizualizaci.

Vizualizace je hlavním důvodem, který vedl ke vzniku tohoto zadání. Jde v podstatě o grafickou reprezentaci toho, co stroj vyrábí, tedy zobrazení skutečného hotového výrobku. Toto je důležité především pro obsluhu systému. Pokud systém žádný podobný modul nemá, obsluha vidí pouze množství souřadnic a bez podrobnější znalosti není schopna určit, co to stroj vlastně bude obrábět. Problémem však je, jak si ukážeme dále, že této grafické vizualizace nelze dosáhnou bez toho, abychom provedli simulaci stroje, tedy napodobení skutečného procesu, který probíhá během obrábění.

Je očividné, že tyto úkoly vyžadují použití složitějších grafických algoritmů. Pokud se tedy chceme věnovat vývoji pouze tohoto modulu a nechceme se zaplést do implementace těchto algoritmů, tak musíme využít externího grafického jádra, které tyto algoritmy obsahuje již naimplementované. Takovýchto jader je k dispozici hned několik a tak je třeba z nich vybrat takové, které by nám vyhovovalo.

1.1. Popis problému

Jak jsem již uvedl výše, tvar výsledného výrobku není možné zobrazit bez toho, aniž by byla provedena alespoň přibližná simulace obrábění. Proč tomu tak je? Důvodem je typický postup podle kterého se pracuje s CNC (Computer Numeric Control) stroji.

Obrázek 1-1: Postup při práci s CNC stroji

Na obrázku 1-1 je vidět, že prvním krokem při tomto postupu je práce v CAD (Computer Aided Design) programu. V tomto kroku odborník navrhne výsledný obrobek se všemi náležitostmi - obdoba kreslícího prkna. Následuje však druhý krok, tedy využití CAM (Computer Aided Manufacturing). V tomto kroku se s pomocí znalostí o konkrétním stroji navrhnou dráhy nástroje tak, aby na konci vznikl hotový výrobek navržený v CADu. Pokud stroj dodržuje tyto dráhy přesně, tak jsou splněny všechny náležitosti a stroj vyrobí přesně to, co bylo požadováno.

Jelikož je tento navrhovaný modul určen pro řídicí systémy CNC strojů, znamená to, že se nacházíme až ve kroku třetím. Máme tedy k dispozici pouze ty samé informace, jaké má k dispozici řídicí systém - tj. výstup z CAM programu. Zpravidla již tedy nemáme přístup k dodatečným informacím nacházející se v CADu. V našem případě dostaneme k dispozici NCP (Numeric Control Part program) program.

Nyní jsem se tedy dostali k jádru problému. Na vstupu tohoto modulu nejsou v podstatě žádné informace o výsledném obrobku. Místo toho však máme k dispozici všechny potřebné informace o polotovaru, o nástroji a o tom, jakým způsobem se bude CNC stroj pohybovat. Tyto informace v podstatě stačí k tomu, abychom byli schopni výsledný obrobek zobrazit, nicméně je před samotným zobrazením obrobku nutné odsimulovat, i když třeba jen zrychleně, celou práci CNC stroje podle programu z CAMu. Odtud je tedy do zadání této práce přidána simulace obrábění CNC.

1.2. Specifikace cílů

Kromě hlavního cíle, který jsem popsal výše, se v zadání nachází šest upřesňujících bodů. I tyto body je třeba splnit. Nyní je tedy detailněji rozeberu a v hrubých rysech určím přístup, jak tyto body budu řešit.

Prvním úkolem je seznámit se s programovým rozhraním řídicího systému MEFI řady CNC872. Napojení se na toto rozhraní je prvním důležitým krokem, který je třeba splnit na této práci. Je tomu tak proto, že výsledný modul je určen právě pro tento systém a jako takový je na tomto systému zcela závislý. Systém modulu poskytuje veškeré informace, které modul potřebuje ke svému správnému chodu. Tyto informace jsou nejrůznějšího charakteru. Například informace o tom, jaké má mít rozměry a pozici pracovní plocha tohoto modulu nebo již předzpracované souřadnice pozice nástroje založené na NCP programu, atd.. Modul se naopak musí chovat tak, aby s ním byl systém schopen komunikovat, tedy musí dodržovat určité zásady. Zde se jedná především o technologii COM (Component Object Model), na které je tento systém postaven.

Dalším úkolem je navrhnout vhodný princip pro vytvoření 3D modelu obrobku na základě informací dostupných v technologickém programu (tj. NCP programu). Zde se jedná o "zrychlenou" simulaci. Pokud je naším primárním cílem vytvořit vizualizaci hotového obrobku, tak nemusíme simulaci postupu CNC stroje provést do nejmenších detailů. Nejde nám ani tak o přesnost simulace, jako spíše o její rychlost. Je tedy možné určité úkony vynechat či spojit dohromady a díky tomu docílit rychlého zhotovení výsledného modelu.

Následující úkol však pojednává o situaci přesně opačné. Zde je naším primárním cílem provést simulaci. Není zde tedy vhodné vynechávat či spojovat některé kroky dohromady. Naopak, naší prioritou v tomto úkolu je přesnost. Při simulování práce CNC stroje může být odhalena chyba. Aby uživatel modulu mohl tuto chybu přesně identifikovat, je potřeba, aby měl možnost si tuto simulaci procházet tak, jak pokračuje v čase. Odsud tedy pochází úkol pro vytvoření animace simulace.

Zvolit vhodný prostředek pro zobrazení dat je dalším vhodným krokem. Ať totiž předcházející dva kroky vyřešíme jakýmkoliv způsobem, vždy budeme potřebovat výsledek, popřípadě aktuální stav, nějak zobrazit. Protože by použití přímo OpenGL, či Direct3D bylo příliš složité, využijeme možnosti použít externí grafické modelovací jádro.

Pátý úkol je zde z důvodu, že vstupujeme do neznámých vod. V době, kdy bylo psáno zadání teto diplomové práce, nebylo ještě zřejmé, co všechno bude tento modul potřebovat ke své správné funkci a tak se mohlo lehce stát, že některé informace nebudou v systému k dispozici pro externí modul. V takovém případě je potřeba navrhnout rozšíření rozhraní systému o tyto informace, aby mohl modul pracovat. K této události naštěstí nedošlo.

Posledním úkol specifikuje, že tento modul má být navržen jako plug-in pro řídicí systém MEFI řady CNC872. Z této specifika vyplývá, jakým způsobem má být tento modul implementován. Navíc je zde uvedeno, že závěrečné testování má být provedeno na reálných technologických programech. Tento požadavek vyplývá z cíle, že tento modul má být, po svém dokončení, nasazen na reálných strojích.

1.3. Dodatečné cíle

Dodatečné cíle nejsou uvedené v zadání této diplomové práce z toho důvodu, že když vznikalo toto zadání, tak se o těchto cílech ještě nevědělo. Tyto cíle byly přidány zejména poté, kdy se o této práci dozvěděli odborníci z Výzkumného centra pro strojírenskou výrobní techniku a technologii. Tito odborníci totiž zvažovali práci na podobném modulu. Nicméně po diskuzi s nimi vyplynulo, že aby byl tento modul v praxi použitelný, tak potřebuje ještě několik dodatečných funkcí, které původní zadání neobsahovalo. Jelikož firma MEFI s tímto výzkumným centrem úzce spolupracuje, bylo rozhodnuto, že se původní zadání rozšíří i o tyto funkcionality.

Hlavním požadavkem bylo obohatit funkce tohoto modulu o import a export grafického modelu z formátů používaných v dnešních CAD aplikacích. Jde především o formáty STEP/IGES/IGS/STL a další. Důvodem tohoto požadavku byl jednoduchý fakt, že CNC stroje často obrábí ne nějaký tvar podobný grafickému primitivu (kvádr, válec...), ale obrábí odlitek, tedy libovolný netriviální tvar. Aby bylo možno něco takového znázornit, je třeba model tohoto odlitku načíst do našeho modulu a to takovým způsobem, aby na něj šly dále aplikovat operace, které budou simulovat obrábění. Export byl vyžadován z důvodu další analýzy provedené simulace v externích aplikacích.

S tím souvisí i další cíl, který zde byl vznesen a to, co možná nejpřesnější provedení simulace. Tedy aby výsledný model byl co možná nejblíže možné skutečnosti a to bez ohledu na časovou náročnost výsledné simulace.

I když tyto body nejsou v zadání této diplomové práce, snažil jsem se je zohlednit v realizaci tohoto modulu a tak, i když jim není věnována taková pozornost jako zadaným cílům, přesto se v tomto projektu vyskytují.

2. Rešerše modelářů

Základní motivací, proč sepisuji tuto kapitolu, je snaha vybrat nejvhodnější modelářské jádro. Mým cílem tedy je představit nejznámější modelářská jádra, porovnat je a vybrat mezi nimi to, které bude nejlépe vyhovovat pro tento projekt.

Nejprve představím tři nejznámější modelářská jádra. Poté porovnám jejich datové modely a licence, což jsou obě velmi důležité informace pro náš projekt. Na základě těchto informací si vyberu jedno modelovací jádro a provedu na něm malý experiment. Tento experiment bude zahrnovat, zda a jak toto jádro ovládá pro tento projekt ty nejdůležitější operace, tedy booleovské operace na dvou 3D modelech.

2.1. Existující jádra

Pro tuto rešerši jsem pracoval pouze se třemi jádry. Prvním a zřejmě nejslavnějším modelářským jádrem je jádro ACIS od firmy Spatian. Toto jádro je dnes považováno za jakousi normu v tomto oboru a tomu odpovídá i velké množství slavných firem používajících toto jádro.

Druhé jádro, které jsem zařadil do této rešerše, je jádro Open CASCADE. Toto jádro je pro nás zajímavé hned z několika důvodů. Prvním důvodem je jeho open source licence, která umožňuje nahlédnout do zdrojového kódu jádra. Druhým důvodem je, že toto jádro je za určitých podmínek poskytováno zadarmo a to i ke komerčním účelům. Třetím a taktéž velmi důležitým důvodem, je dostupná dokumentace tohoto jádra, která je na rozdíl od ostatních jader skutečně použitelná.

Posledním jádrem je jádro Parasolid, za kterým stojí firma Siemens. Toto jádro je taktéž velmi rozšířené, ale díky velmi málo uvolněným informacím je těžké toto jádro blíže popsat.

Jádro	**Firma**	**Web**	**Licence**
ACIS	Spatial	www.spatial.com	Licensed
Open CASCADE	Matra Datavision	www.opencascade.org	Open source
Parasolid	Siemens	www.parasolid.com	Licensed
SMLib	Solid Modeling Solutions	www.smlib.com	Open source
Thinkdesign kernel	Think3 Inc.	www.think3.com	Proprietary

Tabulka 2-1: Přehled modelovacích jader

Kromě těchto tří jader existuje samozřejmě celá řada dalších, která by potencionálně mohla být užitečná. Například open source knihovna SMLib či modelářské jádro Thinkdesign kernel od společnosti Think3. Tato jádra jsou však zaměřena na trochu jiný typ aplikací a tak je v této rešerši nebudu uvažovat.

Dále zde budu mezi sebou porovnávat pouze jádra ACIS, Parasolid a Open CASCADE. Jádra ACIS a Open CASCADE, především jejich topologie, jsou detailněji představeny v samostatných kapitolách.

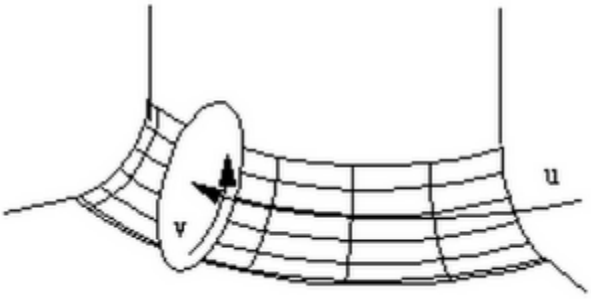
2.2. Datové modely

První, co bych měl porovnat, jsou datové modely jader Open CASCADE, Parasolid a ACIS. Právě datové modely do značné míry prozrazují, jak samotné jádro pracuje a co umí. Protože jsou však jádra ACIS a Parasolid uzavřená a je pro běžného studenta téměř nemožné se k nim blíže dostat, čerpám tedy tyto informace z blogu Romana Lygina [1], což je bývalý vývojář a Project Manager jádra Open CASCADE.

V porovnání s ACIS a Open CASCADE, Parasolid používá menší soubor primitiv. I tak samozřejmě obsahuje základní křivky a povrchy (čáry, kruhy, roviny, válce, ...), NURBS, základní "swept surfaces" (revolution, extrusion) a offsety. Parasolid definuje orientaci geometrického objektu přímo na geometrické úrovni zadáním booleanské "flag" na každou křivku nebo plochu. Původní povrchová normála je definována pomocí vektorového součinu U a V a může být potvrzena nebo rezervována pomocí booleanské flag. ACIS definuje normály pomocí speciálních hodnot některých klíčových parametrů. Ignoruje U a V parametrizace. Například koule s normálami orientovanými dovnitř je v ACIS definována jako koule se záporným poloměrem. Open CASCADE definuje orientaci jako připojení do topologie subjektu (face na surface, hrana na křivku ...).

Transformace v Parasolid mohou být použity pouze na komponenty v sestavách. ACIS používá transformace pouze na "bodies" (nejvyšší úroveň dané entity, která je skupinou pevných těles). Open CASCADE může používat transformace v libovolné úrovni od vertexu po skupinu těles. Tato flexibilita umožňuje Open CASCADE snížit nároky modelu na paměť. Například krychle může být reprezentována pomocí jediného face, který je umístěn na šest různých míst. Nevýhodou nicméně je, že je potřeba učinit zvláštní opatření, aby nebyly nechtěně změněny sdílené objekty. Na rozdíl od Open CASCADE, jak ACIS tak Parasolid, často sdílejí základní geometrii - např. velmi časté jsou faces odkazující na ten samý surface. Open CASCADE toto nedělá z důvodu, aby se zabránilo nechtěným změnám. Dalším důvodem je snížení rizik u multi-threaded aplikací.

Parasolid používá několik zajímavých geometrických typů. První z nich, který není podporován ani v ACIS ani v Open CASCADE, je průsečíková křivka (intersection curve). Průsečíková křivka je definována pomocí množiny bodů, dvou povrchů, které se vzájemně protínají a "law", které určí cíle a parametry křivky. Dalším zajímavým geometrickým typem je "rolling_ball_blend surface", který je definován dvěma podpůrnými povrchy, "páteřní křivkou" (spine curve) a poloměrem koule (ukázka na obrázku 2-1). ACIS podporuje tento typ také, nicméně místo podpůrných povrchů používá podpůrné křivky (které nazývá "Spring curves").



Obrázek 2-1: Rolling_ball_blend surface od jádra Parasolid

Parasolid nenabízí tak širokou škálu povrchů, jako ACIS (variables_blend, net, skin, tube, law, atd.). Ale i tak je jeho nabídka větší než u Open CASCADE, který je založen na standardu STEP ISO010303-42. Open CASCADE má opravdu nejmenší nabídku, snad s výjimkou Bezierových křivek a ploch a faktu, že "pcurve" mohou být jakéhokoliv typu ne jenom B-Spline jako u dvou předchozích jader.

Všechna tři jádra mají koncept pro 3D a 2D reprezentaci "face boundaries". Parasolid používá pojmy jako "fin" či "half-edge" a ACIS používá "coedge", což je entita podle "face`s loops" a jejího otočení. Ukazuje na hrany či na "pcurve". U Parasolid je zajímavé, že může mít reprezentaci pouze ve 3D nebo ve 2D, ale ne obojí, jako mají ostatní dvě jádra.

Obrázek 2-2: Loga firem stojících za vývojem zkoumaných jader

Co se týče tolerancí Parasolid stejně jako Open CASCADE používá "local tolerances", které jsou připojeny k topologickým entitám. ACIS "local tolerances" představil pouze v několika zkušebních verzích. Zatím je však nepoužívá.

Souborové formáty jednotlivých jader jsou také velmi odlišné. Parasolid používá svůj XT formát, který je, na rozdíl od ostatních jader, člověkem v podstatě nečitelný díky číslům, která jsou použita jako typové identifikátory. ACIS používá svůj formát ACIS-SAT. Oba tyto formáty se verze od verze liší a jsou problémy s kompatibilitou (zvláště u Parasolid). Na rozdíl od těchto jader Open CASCADE svůj souborový formát nezměnil již více jak 15let.

2.3. Licence

Jelikož cílem tohoto projektu není pouze teoretické zhodnocení modelářských jader, ale vytvoření fungujícího modulu, který by jednoho dne měl dosáhnout komerčního využití je otázka licence a potažmo ceny zvoleného modeláře velmi důležitá.

V tomto oddílu jednoznačně vítězí Open CASCADE díky své licenci. Licence tohoto jádra je postavena na LGPL licenci, ale je mírně upravena. V zásadě je možno říci, že toto jádro lze zdarma používat ke komerčním účelům a to pouze za podmínky, že bude uvedeno v dokumentaci výsledného produktu, že tento produkt využívá jádro Open CASCADE (v Read me, About, atd). Dále se uživatel zavazuje k poskytnutí jakýchkoliv úprav, které na tomto jádře provede a to zdarma. Je to z důvodů dalšího vývoje jádra.

Toto jsou ideální podmínky pro náš projekt, protože naším cílem není nikterak toto jádro měnit, ale na už existujícím jádře postavit naši simulaci. Další kladnou stránkou je celkem dobrá dokumentace, příklady a široká uživatelská základna, která může zdarma pomoci s vývojem našeho projektu. Na rozdíl od toho jsou jádra ACIS a Parasolid čistě komerční, tedy jsou kompletně uzavřena a to včetně jakýchkoliv příkladů či podrobnější dokumentace.

2.4. Výběr jádra

Pokud se pokusím zhodnotit všechny doposavad uvedené informace a na základě nich se rozhodnout, které jádro by bylo nejlepší pro náš projekt, tak jednoznačná odpověď je jádro ACIS. Je to jádro, které se vyvíjí již velmi dlouhou dobu a využívají ho přední firmy (např. Autodesk). Také při porovnání datových modelů musím uvést, že toto jádro mělo ve většině případů navrch.

Na druhou stranu je třeba uznat, že náš projekt potřebuje pouze základní funkce, které obsahují všechna testovaná jádra. Také se jedná o projekt, který nemá k dispozici žádné velké prostředky. Z tohoto důvodu volím jádro Open CASCADE, které je také velmi kvalitní a mělo by bez problémů zvládnout veškeré operace, které budeme potřebovat. Jeho licence je velmi kladným bodem. Dostupná dokumentace, příklady a komunita uživatelů je výhodou.

2.5. Experiment

2.5.1. Zadání

Nyní, když jsem zvolil jádro Open CASCADE je nezbytné, bych vyzkoušel, zda bude toto jádro skutečně dostatečné. Z tohoto důvodu provedu experiment, ve kterém vyzkouším nejkritičtější body vývoje tohoto projektu na jednoduchém příkladě.

Jako tyto kritické body jsem určil zaprvé zprovoznění tohoto jádra v MFC projektu, jelikož celý systém MEFI je na MFC postaven a pokud by zde byla nějaká překážka, tak by toto jádro bylo nepoužitelné. Za druhé provést booleovskou operaci na dvou tělesech ve 3D, například odečíst od sebe krychli a kouli. Pokud bychom zvládli tuto operaci, je pravděpodobné, že budeme mít vše co potřebujeme při vývoji našeho modulu.

2.5.2. Provedení

První bod tohoto experimentu se podařilo vyřešit. Ukázalo se, že Open CASCADE přednostně podporuje vývoj pomocí Java, Qt, C# a MFC. Ostatní způsoby vývoje jsou sice různými cestami možné, ale nejsou podporované a tedy otestované. Některé funkce by tak mohly při vývoji přestat fungovat. Zprovoznění MFC projektu, který by pracoval s knihovnami Open CASCADE už tak triviální nebylo, přesto byl tento problém nakonec vyřešen. Vývoj Open CASCADE pomocí MFC je tedy možný.

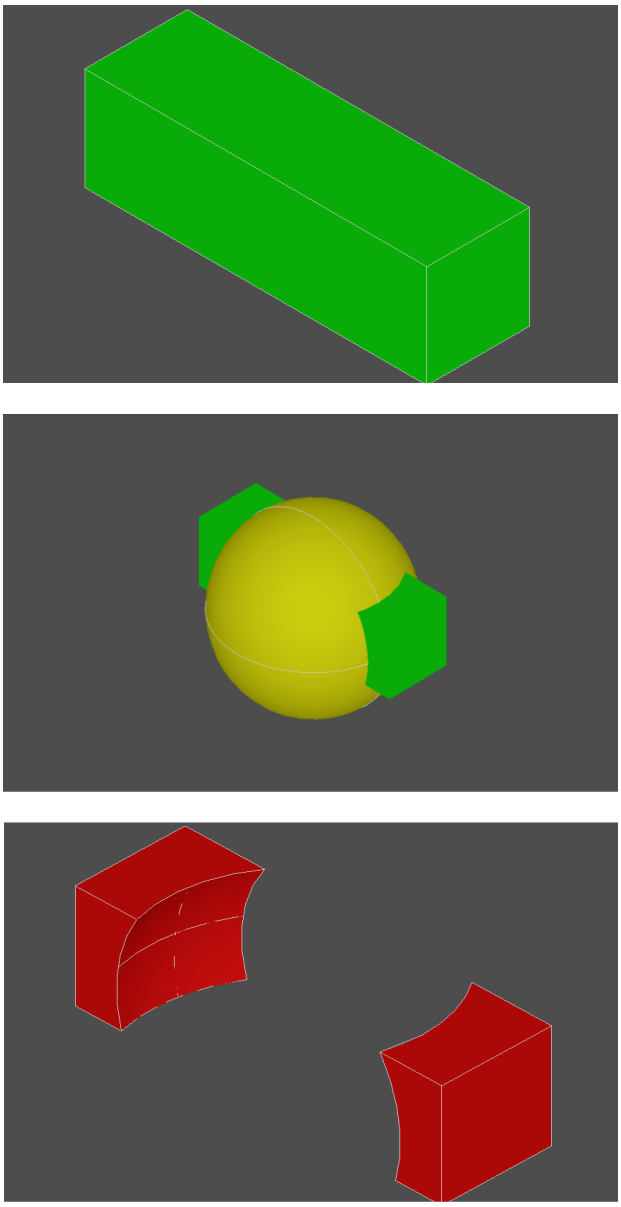
Druhým bodem bylo provedení booleovské operace. Přestože je tato operace pro náš projekt zásadní, pro modelovací jádro to je jedna se základních operací a její provedení tedy bylo velmi jednoduché. Ukázka zjednodušeného kódu:

```
//Vytvoří Krychli
TopoDS_Shape theBox = BRepPrimAPI_MakeBox(200,60,60);

//Vytvoří Kouli
TopoDS_Shape theSphere = BRepPrimAPI_MakeSphere (gp_Pnt( 100, 20,
                                                          20), 80);
//Provede odečtení Koule od Krychle
TopoDS_Shape ShapeCut = BRepAlgoAPI_Cut(theSphere,theBox);
```

2.5.3. Výsledek experimentu

Na následujících obrázcích jsou vidět výsledky těchto tří kroků:



Obrázek 2-3: Výsledek experimentu s jádrem Open CASCADE

2.6. Závěr rešerše

Když tedy zhodnotím tuto rešerši, lze říci, že jsem nalezl dostatečně kvalitní nástroj, který nám umožní dosáhnout našeho cíle, tedy vytvoření simulace obrábění CNC stroje. Tímto nástrojem je modelovací jádro Open CASCADE, které zvládne všechny požadované operace, je velmi levné a bezproblémově funguje na technologiích, které využívá i řídicí systém MEFI.

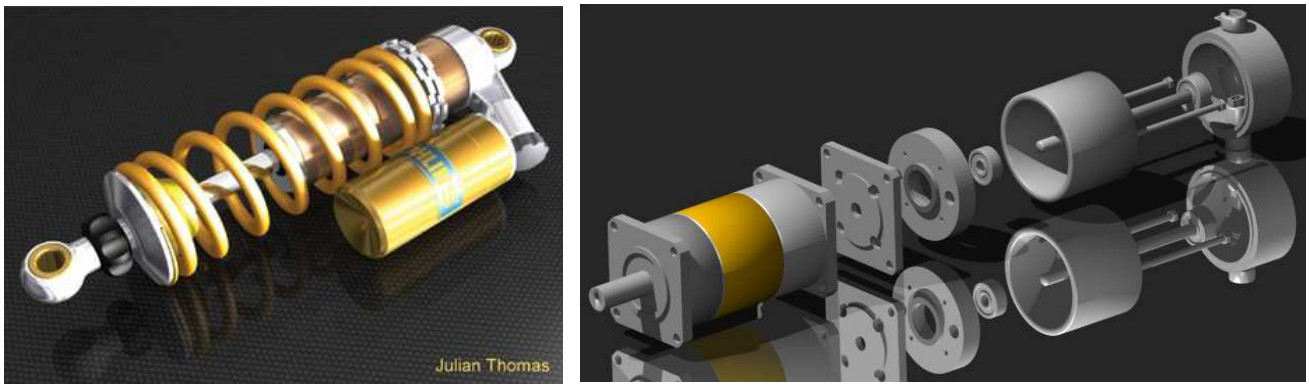
3. ACIS

Vzhledem k tomu, že se jedná o uzavřené jádro, je tedy složité získat podrobnější dokumentace. Využil jsem proto pro sepsání této kapitoly dokumentaci přímo od firmy Spiatian [2]. The 3D ACIS® Modeler je v dnešní době jedno z nejuznávanějších 3D modelářských jader (3D modelling kernel/engine). Ostatní modelářská jádra jsou z pravidla vždy porovnávána s tímto jádrem, lze ho tedy považovat za určitý standard. Toto jádro vyvinula a vlastní společnost Spatian Corporation (dříve Spatian Technology). Využívá ho celé řada firem zabývajících se vývojem softwaru pro průmysl po celém světě. Hlavní odvětví průmyslu, kde se tento software využívá, jsou: CAD, CAM, CAE (Computer-aided engineering), AEC (Engineering and Construction), CMM (Coordinate-measuring machine), ale i architektura, 3D animace a vizualizace a další. Pro všechna tato odvětví poskytuje ACIS potřebné funkce a podporu.

ACIS je objektově orientovaná architektura C++, která umožňuje 3D modelovací funkce. Je ideální pro konstrukční aplikace s hybridními modelovacími funkcemi, protože v sobě integruje drátěný model (wireframe), povrchovou reprezentaci (sheet) a objemovou reprezentaci (solid), to vše s náležitými funkcionalitami a bohatou sadou geometrických operací. Architektura ACIS se navíc skládá z jednotlivých komponent. Díky tomu je velmi flexibilní. Zákazník tedy zaplatí a dostane pouze ty komponenty, které skutečně potřebuje.

Název ACIS vznikl z řecké mytologie, kde Acis (česky: Ákis, latinsky: Acis) byl v Ovidiových Proměnách duchem řeky Acis, která protéká blízko Akionu pod sopkou Etna na Sicílii.

Jako geometrické jádro je ACIS považován až za druhou generaci systému, který byl původně znám pod názvem Romulus. Na vývoji tohoto jádra se od roku 1985 podílejí Charles Lang a Ian Brain (tvůrci původního systému Romulus a Romulus-D). V roce 1986 je tato skupina připojena k firmě Spatian Technology, patřící Dicku Sowarovi, a začíná pracovat na modelovacím jádru ACIS. První verze jádra ACIS byla vydána v roce 1989 a byla rychle licencována firmou HP pro jejich produkt ME CAD software.



Obrázek 3-1: Ukázky použití ACIS

3.1. Architektura ACIS

Architektura modelovacího jádra ACIS je postavena na Software component technology. Softwarové komponenty jsou části softwaru (například: funkce, třídy, atd.), které jsou specializované na konkrétní funkci. Tyto komponenty slouží jako součásti celkového softwarového systému nebo produktu. Tedy hotový softwarový produkt je složen z jedné nebo více těchto komponent, které jsou propojené dohromady a prodávány jako celek. Komponenty mohou být uspořádány v různých kombinacích pro různé koncové produkty.

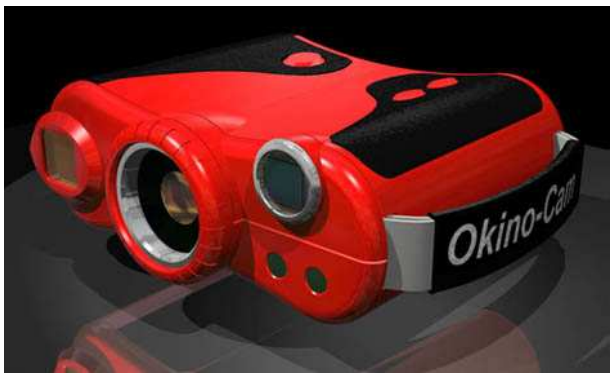
ACIS využívá právě tuto technologii, ta mu umožňuje, že koncová aplikace obsahuje právě ty komponenty, které zákazník opravdu potřebuje. Navíc je možné do koncové aplikace přidat i komponenty, které nevyvinula firma Spatian, ale jiný dodavatel. Příkladem může být renderovací komponenta. ACIS podporuje hned několik komponent, které se starají o rendering a zákazník si tedy může vybrat tu, která mu nejlépe vyhovuje.

Platforma	Operační systém	Kompiler	32-bit	64-bit
Microsoft	Windows XP Professional SP2	Visual C++ .NET 2005	X	X
Microsoft	Windows 2000 SP4	Visual C++ .NET 2003	X	
Microsoft	Windows XP Professional x64 Edition	Platform SDK for Windows Server February 2003		X
Red Hat	Enterprise Linux, Version 5	GNU C++ gcc 4.1.2	X	X
Apple	OS X 10.4 (Native BSD)	GNU C++ gcc 4.0 and XCode 2.3	X	
IBM	AIX Version 5.3	VisualAge C++ Version 8.0	X	X
HP	HP-UX 11i v1	HP aC++ Version A.03.52	X	X
SunTM	SolarisTM 10	Sun ONETM Studio 10	X	X

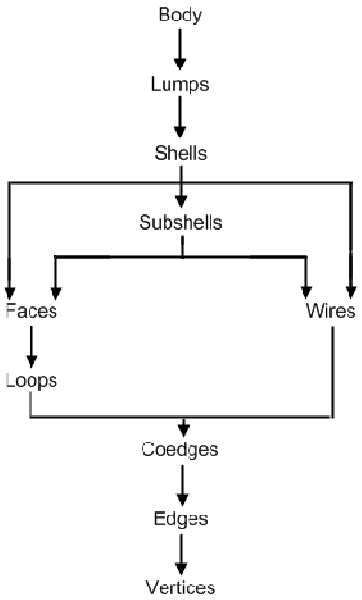
Tabulka 3-1: ACIS - Podporované operační systémy

3.2. Topologické uspořádání ACIS

Práci modelovacího jádra ACIS lze uspořádat do topologické struktury, která definuje, jak jádro pracuje s daným modelem v různých krocích. Toto uspořádání má ale řadu výjimek (například práce s objemovou reprezentací), kdy se některé kroky vynechávají a jiné zase přidávají. Celkově lze říci, že struktura uvnitř jádra a jeho práce je velmi složitá.



Obrázek 3-2: Ukázka použití ACIS



Obrázek 3-3: Vztahy mezi topologickými entitami ACIS

Element	Popis
Body	Nejvyšší úroveň objektu modelu. Skládá se z žádné nebo více *Lumps*.
Lump	Propojení 1D, 2D nebo 3D množiny bodů, které nejsou spojeny s žádnou jinou *Lumps*. Takto je ohraničeno jeden nebo více *Shell*.
Shell	Množina propojených *faces* a *wires*, které ohraničují vnější *solid* nebo vnitřní *void*.
Subshell	Volitelný rozklad *shell* z důvodů efektivity.
Face	Propojená část *surface* ohraničená množinou *loops*.
Loop	Propojená kružnice tvořena *coedges*. Obecně platí, že smyčky jsou uzavřené, nemají žádný skutečný začátek nebo konec.
Wire	Propojená množina *edges*, které nejsou připojené na žádnou *face*.
Coedge	Představuje použití *edge*, *face* nebo *wire*.
TCoedge	Specializované *coedge* - *tolerant coedge*, existuje pouze na *tolerant edge*.
Edge	Úsečka ohraničená vrcholy.
TEdge	Specializovaná *edge* - *tolerant edge*, reprezentuje *edge*, která neleží uvnitř systému polohové tolerance *face*.
Vertex	Hranice *edge*. V*ertex* odpovídá bodu ve 3D prostoru.
TVertex	Specializovaný *vertex* - *tolerant vertex*, který představuje vrchol, který neleží v rámci systému polohové tolerance *edge*.

Tabulka 3-2: Popis topologických elementů modelu ACIS

3.3. Přehled podporovaných funkcí jádrem ACIS

Jednotlivé funkce modelovacího jádra ACIS lze rozdělit do tří následujících kategorií: (Z důvodu zachování korektních názvů tento přehled píši v angličtině.)

3.3.1. 3D Modelování

- Extrude/revolve/sweep sets of 2D curves into complex surfaces or solids.
- Fillet and chamfer between faces and along edges in surface and solid models.
- Fit surfaces to a closed network of curves.
- Generate patterns of repetitive shapes.
- Hollow solids and thicken surfaces.
- Interactively bend, twist, stretch, and warp combinations of curves, surfaces, and solids.
- Intersect/subtract/unite any combination of curves, surfaces, and solids.
- Loft surfaces to fit a set of profile curves.
- Taper/offset/move surfaces in a model.

3.3.2. 3D Správa modelu

- Attach user-defined data to any level of a model.
- Track geometry and topology changes.
- Calculate mass and volume.
- Model sub-regions of a solid using cellular topology.
- Unlimited undo/redo with independent history streams.

3.3.3. 3D Vizualizace modelu

- Tessellate surface geometry into polygonal mesh representation.
- Create advanced surfacing capabilities with the optional Deformable Modeling component.
- Generate precise 2D projections with hidden line removal using optional PHL V5 component.
- Develop graphical applications

3.4. Přehled rozšíření pro modelářské jádro ACIS

3.4.1. 3D Deformable Modeling

3D Deformable Modeling je interaktivní *sculpting tool* pro tvarování 3D modelů. Pokud je zahrnut jako součást Spatian balíčku pro 3D modelování, využívá své lokální, ale i globální funkce, které mu umožňují snadnou manipulaci a úpravy free-form, B-spline a NURBS křivek a ploch.

3.4.2. Advanced Covering

Advanced Covering je funkce Deformable Modeling, která je však k dispozici jako samostatný add-on. Toto API využívá sofistikované algoritmy pro tvorbu vysoce kvalitních n-stranných povrchů, které splňují tolerance zadané uživatelem pro pozici a kontinuitu na hranicích a na volitelné vnitřní geometrii. Umožňuje povrchům, aby byly připojeny na kruh (tj. množina úseček tvořících uzavřenou smyčku), na pevné těleso či na drátěný model. Advanced Covering obsahuje i další funkce.

3.4.3. Defeaturing

Automaticky identifikuje a odstraňuje i malé prvky, které CAE (Computer-aided engineering) analytici obvykle chtějí odstranit z 3D modelu. Analytici často pracují na stejném modelu, který používají pro design a výrobu. Tyto modely však často obsahují více detailů, než je ve skutečnosti potřeba. Odstranění těchto zbytečných detailů se běžně provádí manuálně a je to drahá procedura. Defeaturing tuto proceduru zjednodušuje.

3.4.4. PHL V5

PHL V5 umožňuje rychlý vývoj 2D dokumentace a kreslící funkce pro mnohá použití, včetně strojírenství, architektury či výroby. Díky jednoduchému přístupu k osvědčené technologii Hidden Line Removal Technology (HLR).

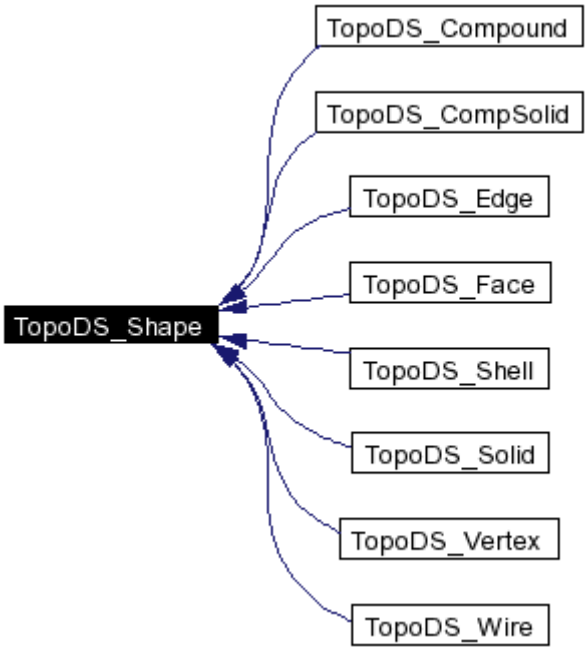
4. Open CASCADE

Jádro Open CASCADE vyšlo z naší rešerše jako nejlepší volba pro náš projekt. Z tohoto důvodu budu popisu tohoto jádra věnovat velkou pozornost. Jak je vidět z předchozích kapitol největší pozornost se při výběru jádra klade na topologii. Je to dáno tím, že z popisu topologie lze alespoň přibližně poznat, jakým způsobem jádro uvnitř vlastně funguje. Druhou velmi důležitou výhodou, kterou znalost topologie jádra přináší, je, že se tímto způsobem uživatel jádra seznámí, jak to či ono jádro používat. Pokud budu po určitém jádru totiž vyžadovat, aby se chovalo určitým způsobem a budu při tom mít na paměti topologii tohoto jádra, budu i intuitivně vědět, jaké informace po mě jádro bude vyžadovat a jak mu je mám předat.

Popis topologie jádra je však už z principu složitý úkol a tak i v tomto případě využiji informace od Romana Lygina [1]. K charakteristice tohoto jádra přistoupíme tak, že si nejdříve stručně popíšeme topologii tohoto jádra a poté si detailněji popíšeme jednotlivé části této topologie s tím, že se zaměříme na její provázanost s reálnou geometrií.

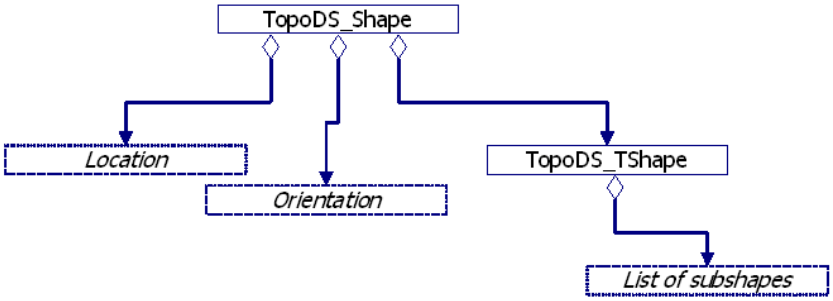
4.1. Topologie Open CASCADE

Topologie jádra Open CASCADE je postavena s ohledem na standard STEP, konkrétně pak ISO-10303-42 z roku 1997. Struktura této topologie je orientovaný graf, kde reference ukazují vždy jen jedním směrem. To znamená, že rodič (parent) odkazuje na svého potomka (children), nicméně již zde není zpětná reference. Tato abstraktní struktura je implementována jako třídy C++, které patří do balíku nazvaného TopoDS. Na obrázku 4-1 je tato struktura zobrazena.



Obrázek 4-1: Topologie Open CASCADE

Z digramu na obrázku 4-2 je vidět, že základní typ TopoDS_Shape je ovládán pomocí obsahu tří ukazatelů. Konkrétně jde o umístění (Location), orientaci (Orientation) a TopoDS_TShape.

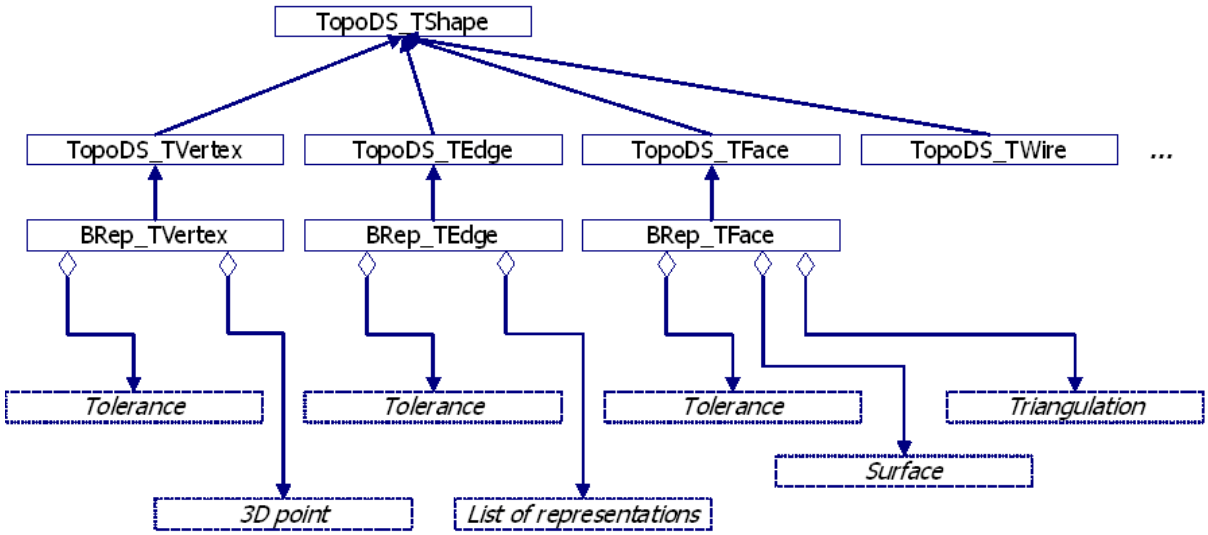


Obrázek 4-2: Diagram TopoDS_Shape

TopoDS_TShape a Location jsou použity pro sdílení dat mezi různými tvary a tím ušetří velké množství paměti. Například *edge*, která patří do dvou různých *face* má stejné Location i TopoDS_TShape. Jediné, co se mění, je Orientation (Forward pro jednu a Reversed pro druhou).

4.2. Návaznost na geometrii

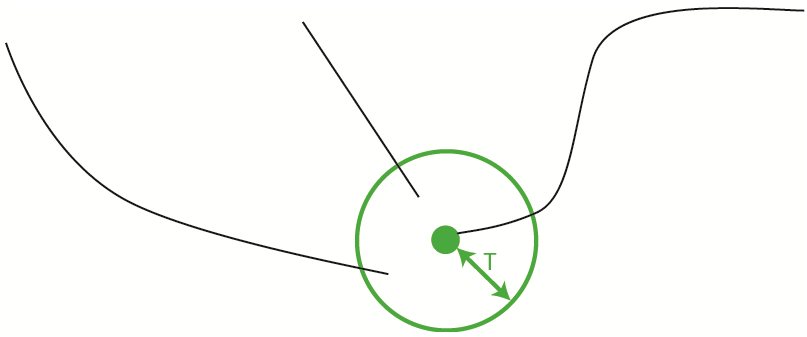
Abstraktní struktury jsou jedna věc, nicméně to opravdu zajímavé je jejich návaznost na reálnou geometrii. Toho se u Open CASCADE dosáhne děděním tříd skrz abstraktní topologii od základního balíku TopoDS. Tyto informace příjme balík BRep (tj. Boundary Reprezentition). Postup těchto informací můžeme vyčíst z obrázků 4-2 a 4-3. Z obrázku 4-3 taktéž můžeme zjistit, že pouze tři typy topologických objeků mají geometrickou reprezentaci - jsou to *vertex*, *edge* a *face*. Tyto tři typy si nyní popíšeme podrobněji.



Obrázek 4-3: Reference v topologii Open CASCADE

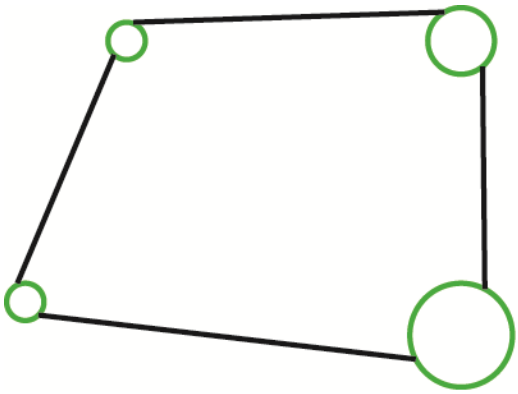
4.2.1. Vertex

Vertex (vrchol) je základní geometrickou reprezentací. Jedná se o bod ve 3D prostoru, který bývá v Open CASCADE označován jako gp_Pnt. Na obrázku 4-3 můžeme vyčíst, že vertex kromě tohoto bodu obsahu je také toleranci. Tolerance je taktéž velmi důležitý atribut, který vyjadřuje možnou nepřesnost v umístění bodu.V geometrických pojmech je tolerance T koule se středem v bodu vertexu, kde T je poloměr této koule. Tato koule musí zahrnovat konce všech *edges* napojených na tento bod.



Obrázek 4-4: Vertex s vyznačenou tolerancí

Na rozdíl od většiny ostatních geometrických knihoven, které toleranci spravují globálně, Open CASCADE spravuje toleranci jako lokální vlastnost (viz obrázek 4-3). Tento přístup pomáhá přesněji popsat výsledný model. Na obrázku 4-5 je to názorně ukázáno:

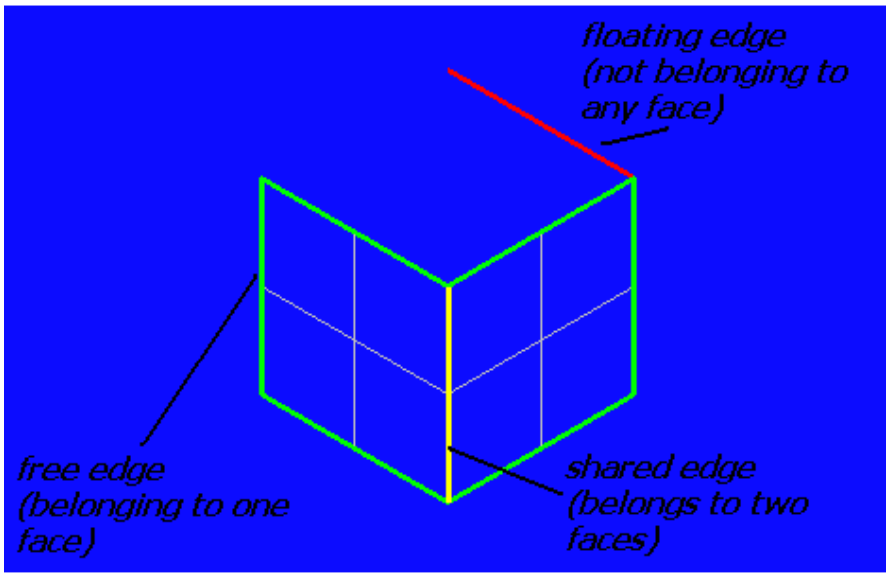


Obrázek 4-5: Ukázka využití lokální tolerance

Individuální tolerance dovolují postarat se o lokální nepřesnosti, zatímco nechávají zbytek modelu přesně definován. Obecně lze říci, že některé modelovací algoritmy jsou na nastavení tolerance velmi citlivé. Příkladem může být *Sewing* (Šití), které má za úkol propojit topologicky rozdílné *face* do jednoho *shell*. Mezera mezi *faces* může být dosti veliká a tak pokud nastavím toleranci jako příliš malou, zůstane mi po této operaci příliš mnoho nepropojených *face*. Pokud ji naopak nastavím příliš velikou, *Sewing* propojí i *face*, které jsou od sebe příliš vzdálené a výsledek opět nebude dobrý.

4.2.2. Edge

Edges neboli hrany, jsou topologické objekty, které odpovídají 2D objektu, totiž křivce. Mohou určovat hranice *face* nebo se mohou vyskytovat volně (tzv. *floating edge*) a nepatřit k žádná *face*. Jedna edge může být sdílena několika face (dvě i více). Názorně je to ukázáno na obrázku 4-6.

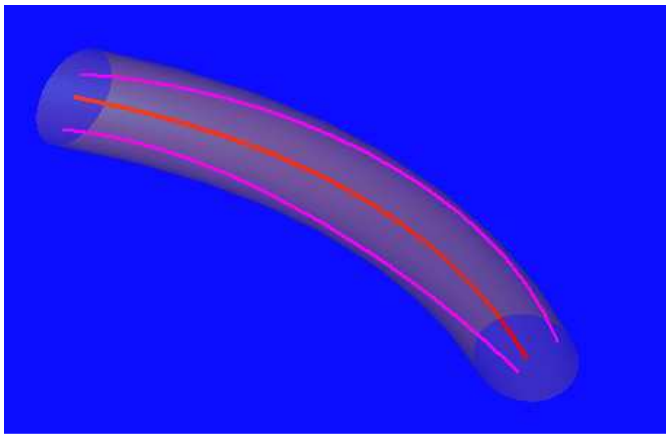


Obrázek 4-6: Edge a Face

Diagram na obrázku 4-3 ukazuje, že *edge* může mít hned několik geometrických reprezentací.

- Křivka C(t) v 3D prostoru označovaná jako Geom_Curve. Tato reprezentace se považuje za primární.
- Křivky P(t) v parametrickém 2D prostoru *surface*, který je základem každého *face*, do kterého hrana patří. Tyto *edge* jsou často nazývány *pcurve* a jsou označeny Geom2d_Curve.
- Polygonální reprezentace, jako pole bodů v 3D prostoru. Označení je Poly_Polygon3D.
- Polygonální reprezentace, jako pole indexů v pole bodů ze triangulace *face*. Označení je Poly_PlygonOnTriangulation.

I u hran je použit parametr tolerance. Zde tolerance představuje maximální odchylku mezi její 3D křivkou a jakoukoliv další reprezentací. To znamená, že geometrický význam tolerance u křivky je poloměr *Pipe* (potrubí), které je vedeno podél její 3D křivky a musí zahrnovat všechny ostatní reprezentace. Lépe je to znázorněno na obrázku 4-7:



Obrázek 4-7: Geometrický význam tolerance u křivky

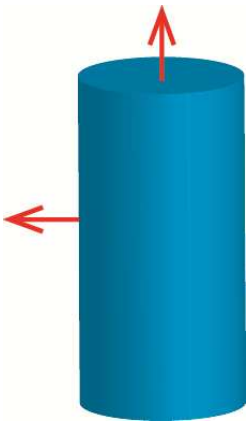
Kromě klasických reprezentací hran existuje i několik speciálních typů. Ty se využívají například u konstrukce koule či válce. Orientace má u hrany logický význam, totiž že *Forward* reprezentuje stejný směr jako má křivka, podél které hrana vznikla.

4.2.3. Face

Face je topologická entita, která popisuje hranice 3D objektu. Je popsán pomocí jeho výchozího *surface* (povrchu) a jedním nebo několik *wires* (což je spojení několik *edges*). Například válec se skládá ze tří *faces*, totiž z horní, z dolní a boční. Každá z příslušných výchozích *surface* je nekonečná, zatímco každá *face*, z nichž se válec skládá, je ohraničená pomocí příslušných *wire* a tedy konečná.

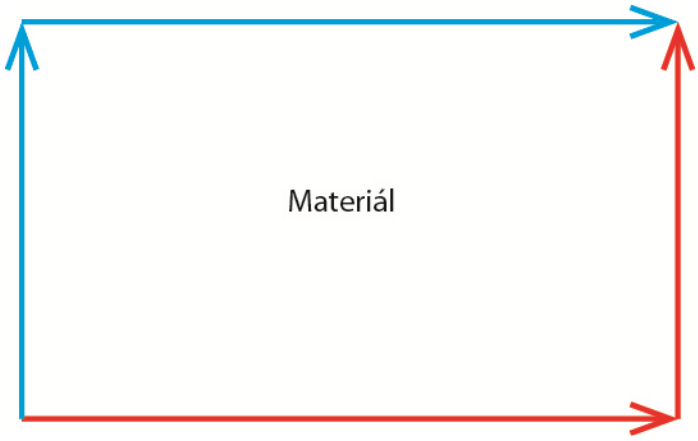
Z tohoto příkladu je vidět, že velmi podstatnou součástí *face* je *surface*. *Surface* mapuje 2D parametrický prostor {U,V} do 3D prostoru (i když je stále 2D). Lze si to představit jako ocelový plech, který se deformací změní v něco zakřiveného. Parametrický prostor může být ohraničený, neohraničený a nebo neohraničený jen v jednom směru. Například rovina (*plane*) je neohraničený prostor, NURBS je ohraničený a válcový povrch je ohraničený jen v jednom směru (ohraničen v U: od 0 do 2*PI; neohraničen ve V: $\pm\infty$).

Orientace u *face* ukazuje, jak jsou normály *face* zarovnány vzhledem k normálám jeho výchozího *surface*. Pokud je orientace *Forward*, potom normály souhlasí (jsou zarovnány stejným směrem). Pokud je *Reversed*, potom jsou normály na opačných stranách proti sobě. Normály *face* ukazují, kde se nachází materiál tělesa (tedy jeho výplň). U správně definovaného tělesa musí všechny normály *face* ukazovat z tělesa ven (viz obrázek 4-8).



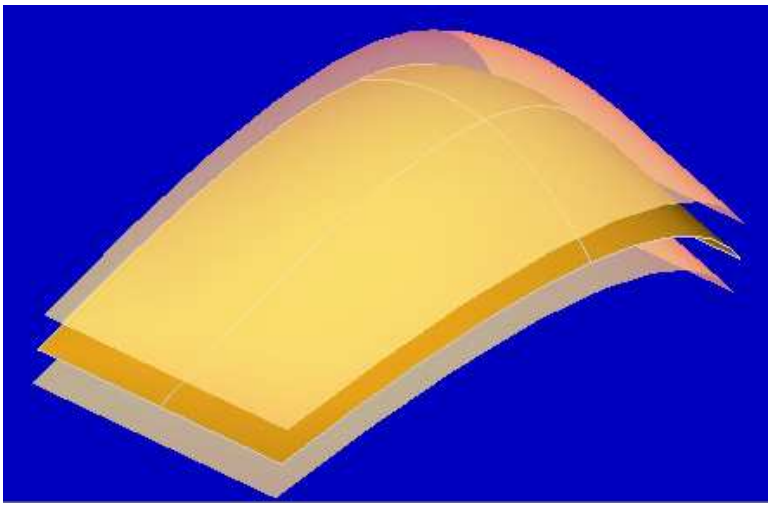
Obrázek 4-8: Orientace normál face

Materiál na *face* se definuje pomocí orientace *edges*. Strana, na kterou bude materiál mapován, je definována pomocí vektorového součinu normál *surface* (ne *face*) a *edge* s tím, že orientace *edge* určuje jak přesně. Přehledně je to zobrazeno na obrázku 4-9. Modré čáry na obrázku ukazují *Reversed edge*, zatímco červené ukazují *Forward*.



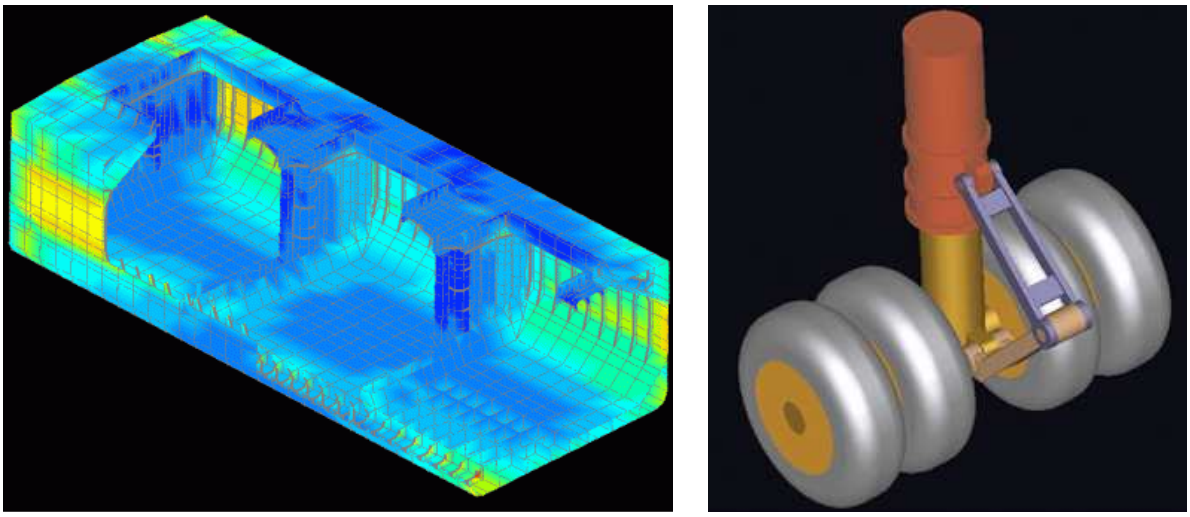
Obrázek 4-9: Výpočet umístění materiálu

Jako poslední si musíme ukázat geometrický význam tolerance u *face*. Jde v postatě o tloušťku objektu, který se vytvoří v okolí *face*. Zřejmě nejjednodušší způsob jak ji představit, je obrázek (obrázek 4-10).



Obrázek 4-10: Význam tolerance u face

O topologii či obecně o vnitřní struktuře jádra Open CASCADE by se dalo napsat ještě mnoho. Nicméně to není cílem této práce. V zásadě jsme si zde však představili základní typy či postupy, s jakými se uvnitř tohoto jádra pracuje. Těchto informací využijeme v dalších kapitolách při realizaci námi navrhovaného modulu. Na závěr několik ukázek, co jádro Open CASCADE dokáže.



Obrázek 4-11: Ukázky práce Open CASCADE

5. Model objektů COM

Cílem této kapitoly je prostudovat model objektů COM. Tato technologie je pro náš projekt důležitá, protože díky ní se náš modul napojí na řídicí systém MEFI, který tuto technologii používá. Informace jsem čerpal z knihy Programování ve Windows pomocí MFC [3]. Tato publikace obsahuje detailní informace o celé této problematice.

Pojem COM je akronymem anglického pojmenování objektově orientovaného programovacího modelu neboli modelu objektů COM (Component Object Model). Zjednodušeně se dá říci, že model COM je způsobem vytváření objektů, které jsou nezávislé na jakémkoliv programovacím jazyce. Specifikace tohoto modelu jsou velmi obsáhlé a sahají do velikých detailů. Pro tuto práci však stačí porozumět pouze základním principům.

Programátoři C++ běžně vytváří třídy pro ostatní programátory. Nedostatkem těchto tříd je jejich určení pouze pro jiné programátory C++. Programovací model COM dává návod, jak vytvářet a používat objekty v libovolném programovacím jazyce. Model COM tedy překračuje všechny jazykové bariéry a nabízí skutečně binární standard objektové architektury.

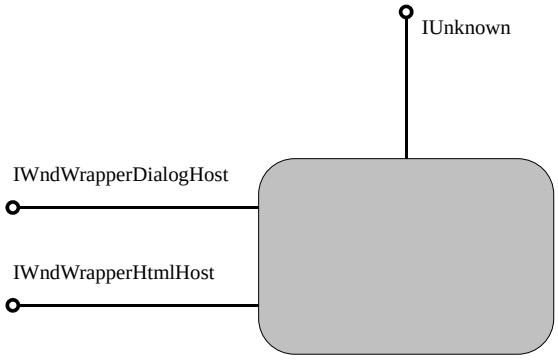
Třídy jazyka C++ obsahují členské funkce; objekty COM obsahují *metody*. Metody jsou seskupeny do příslušných *rozhraní* a jsou volány prostřednictvím *ukazatelů rozhraní*. Tato rozhraní jsou určena k sémantickému spojování skupin vzájemně souvisejících metod. Programátoři společnosti Microsoft předdefinovali více než 100 různých rozhraní použitelných v kterémkoli objektu COM. Těmto rozhraním se říká *standardní rozhraní*. Rozhraní definovaná uživatelem jsou nazývána *vlastní rozhraní*. Objekty modelu COM mohou používat standardní rozhraní, vlastní rozhraní nebo jejich kombinaci.

Každý objekt COM implementuje rozhraní *IUnknown* (Vkládání velkého písmene I na počátek názvu rozhraní je považováno za univerzální konvenci). Toto rozhraní obsahuje pouze tři metody:

Název metody	**Popis**
QueryInterface	Vrací ukazatel na další rozhraní.
AddRef	Zvýší počet odkazů na objekt.
Release	Sníží počet odkazů na objekt.

Tabulka 5-1: Popis metod, které obsahuje rozhraní IUnknown

Jedno z pravidel objektově orientovaného programovacího modelu COM říká, že ukazatelem rozhraní může uživatel zavolat jak kteroukoli z metod rozhraní IUnknown, tak libovolnou metodu, jež je pro dané rozhraní charakteristická. Všechna rozhraní tedy musí kromě svých metod podporovat také tři metody IUnknown. Většina objektů neimplementuje IUnknown jako samostatné rozhraní.



Obrázek 5-1: Prostý objekt COM

I když se rozhraní popisují popisnými názvy jako např. IWndWrapperDialogHost, ve skutečnosti jsou rozhraní identifikována nikoliv názvy, ale číselnými hodnotami. Každé rozhraní je jednoznačně označeno 128bitovou hodnotou, které se říká *identifikátor rozhraní* nebo IID. 128bitové hodnoty nabízí tolik různých čísel, že možnost, aby dva vývojáři vybrali stejný identifikátor je prakticky nulová. Proto je tedy zcela nepodstatné, že dva vývojáři na dvou koncích planety navrhnou dvě neslučitelná rozhraní nazvaná například IMath. Podstatné je, že tato rozhraní mají dva různé identifikátory.

5.1. Životní cyklus objektu COM

V C++ se běžně tvoří objekty na haldě. Takto vznikají všechny objekty, které jsou vytvořeny pomocí operátoru *new*. Tyto objekty je také třeba smazat pomocí operátoru *delete*. Model COM se v tomto ohledu od modelu C++ liší. Klient sice instance objektu vytváří, ale už je neodstraňuje. Objekty COM se totiž mažou sami.

Důvodem této změny je následující situace: Klienti A a B používají stejnou instanci objektu. Klient A vytvoří objekt, klient B se k němu připojí tím, že získá ukazatel rozhraní. Kdyby klient A, netuše, že existuje ještě nějaký klient B, vymazal objekt z paměti, klient B by zůstal s neplatným ukazatelem rozhraní, neboť ten by ukazoval do prázdna. Vzhledem k tomu, že klienti modelu COM zpravidla nevědí (a ani se o to nezajímají), zda jsou či nejsou jedinými klienty objektu, ponechává rozhraní COM vymazání objektu na něm samotném. K vymazání objektu dojde v okamžiku, kdy klesne na nulu interně uchovávaný *počet referencí* na daný objekt. Počet referencí je aktualizovanou hodnotou, která určuje počet klientů s ukazateli nastavenými na rozhraní objektu.

V C++ je počet odkazů zpravidla ukládán do členské proměnné. Počet se zvýší po každém vykonání metody *AddRef* a sníží po vykonání metody *Release*. Implementace těchto metod jsou z pravidla následovné:

```
ULONG __stdcall CComClass::AddRef()
{
	return ++m_lRef;
}

ULONG __stdcall CComClass::Release()
{
	if(--m_lRef == 0)
	{
		delete this;
		return 0;
	}
	return m_lRef;
}
```

V tomto příkladu reprezentuje *CComClass* třídu modelu COM. Proměnná m_lRef je členskou proměnnou, která uchovává počet odkazů na daný objekt.

5.2. Jak získat ukazatel rozhraní

Nyní přichází na řadu využití poslední ze tří metod IUnknown, tedy *QueryInterface*. Klient může na základě ukazatele rozhraní volat metodu QueryInterface, která umožňuje získat ukazatel na libovolné rozhraní podporované daným objektem. Ukázka takto získávaného rozhraní:

```
m_lpHost->QueryInterface(	IID_IWndWrapperDialogHost,
				(void**)&pThis->m_lpDialogHost);
```

V této ukázce získáme ukazatel na rozhraní *IID_IWndWrapperDialogHost* a naplníme jím ukazatel *m_lpDialogHost*.

6. Návrh provedení simulace

V této chvíli máme k dispozici všechny technické prostředky, abychom se mohli pustit do implementace našeho modulu. Předtím, než tak ale učiníme je třeba rozvážit, jakým způsobem budeme implementovat simulaci postupu obrábění CNC stroje. Nejlepší způsob jak začít, je popsat si, co vlastně chceme simulovat.

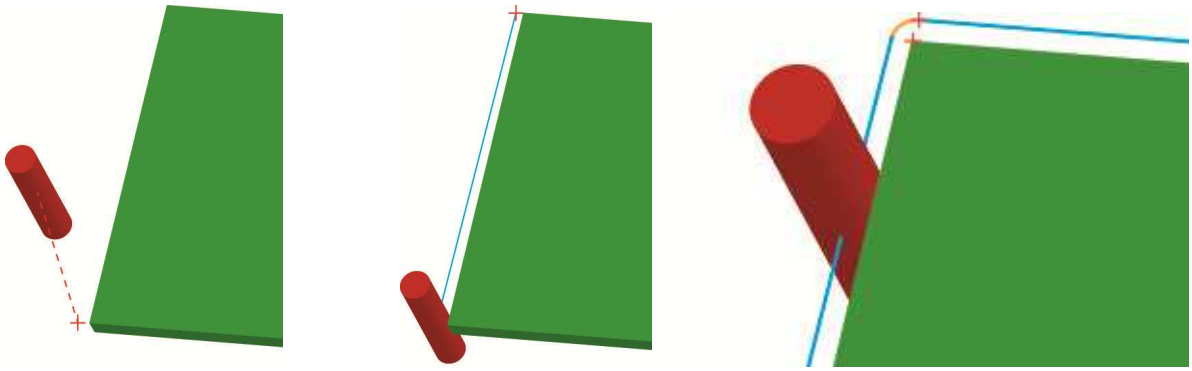
6.1. Vstupní data

Na vstupu našeho modulu mohou být dva typy vstupních dat. Prvním typem jsou systémem již předpřipravené bloky, které byly načteny z NCP programu. Druhý typ dat, který můžeme ze systému získat, jsou souřadnice na výstupu z interpolátoru systému. Oba tyto typy si nyní přiblížíme.

6.1.1. Předpřipravené bloky programu

Tyto bloky jsou již předpřipravené, tedy, již v sobě mají započítané například transformace, které systém musí dopočítávat či již mají spočítánu dráhu s poloměrovou korekcí atd.. Jediné, co s těmito daty musíme udělat, je zobrazit je.

Bloky, které do modulu tímto způsobem přicházejí, obsahují vždy informaci o nové souřadnici, o typu bloku a o nástroji, který byl na obrobení daného bloku použit. Hlavní typy takovýchto bloků jsou rychloposuv, lineární posuv a oblouk. Rychloposuv pouze ukazuje, kam se má nástroj přemístit, obsahuje tedy cílové souřadnice, po cestě však nástroj neobrábí. Lineární posuv je pohyb a obrábění nástroje po přímce. Blok opět obsahuje cílové souřadnice. Konečně oblouk značí pohyb nástroje po oblouku / kružnici. Blok kromě cílové souřadnice navíc obsahuje informace o středu kružnice. Na následujícím obrázku jsou tyto typy předvedeny. Červená čárkovaná čára značí rychloposuv. Modrá je lineární posuv a oranžová je oblouk. Červené křížky určují hodnoty obsažené v blocích.

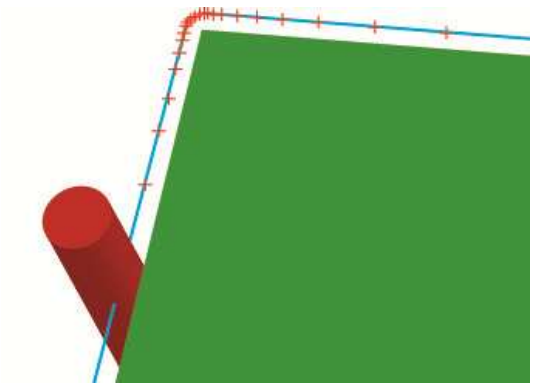


Obrázek 6-1:Ukázky typů bloků

6.1.2. Souřadnice z interpolátoru

Předpřipravené bloky jsou velmi vhodnou cestou pro rychlé provedení simulace kvůli vizualizaci výsledného obrobku. Pokud by však bylo cílem simulace místo rychlosti její přesnost, pak tyto bloky už tak vhodné nejsou. Důvodem je to, že tyto bloky jsou předpřipravovány podle programu - tedy jedná se o čistě virtuální záležitost, která nemá možnost odhalit jakoukoliv případnou chybu či nepřesnost, kterou by systém udělal při provádění programu.

Pokud bychom se chtěli této skutečnosti vyhnout, pak musíme na vstup našeho modulu přivést data, které jsou na výstupu systému - tedy údaje, které systém odesílá už přímo pohonům stroje. A právě tyto hodnoty jsou souřadnice z interpolátoru systému. Tyto souřadnice jsou však uzpůsobeny tak, aby podle nich mohl pohon stroje řídit stroj, tedy například se v nich už počítá s rychlostí, s jakou se pohon pohybuje atd.. Příklad, jak mohou takovéto souřadnice vypadat je na následujícím obrázku:

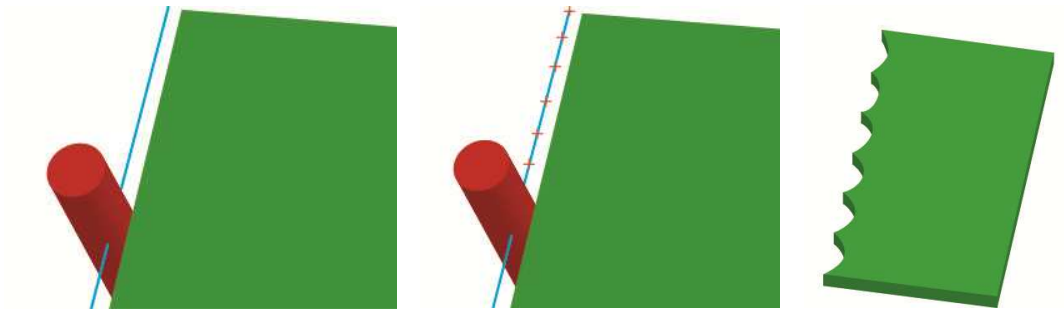


Obrázek 6-2: Souřadnice z interpolátoru

6.2. Zobrazení nástroje

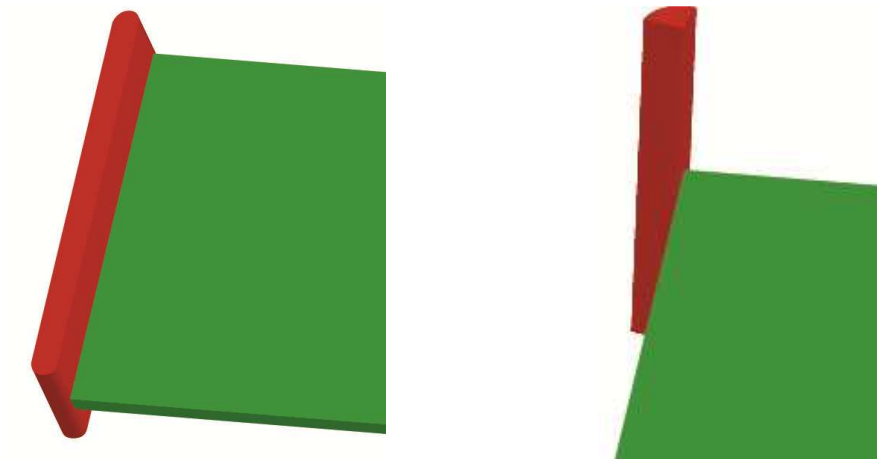
Další otázku, kterou si musíme položit před zahájením implementace, je, jakým způsobem se bude zobrazovat obráběcí nástroj. Opět zde musíme zvážit naše priority. Pokud by naším cílem byla přesnost, musíme vymodelovat nástroj přesně. Ve většině případu však stačí složitý tvar nástroje nahradit příhodným primitivem: válcová frézka či vrták - válec; kulová frézka - koule; atd..

Je však otázkou, jakým způsobem se bude tento nástroj pohybovat. Pokud bychom tento problém řešili intuitivně, tak bychom si určili nějakou přesnost jako konstantu a dráhu, například po přímce, bychom rozdrobili podle této konstanty na množinu bodů. Do každého bodu z této množiny bychom přesunuli nástroj a provedli bychom booleovskou operaci s polotovarem. Tento postup je samozřejmě možný a pokud by se konstanta nastavila dostatečně malá - výsledek by byl dostatečně přesný. Nevýhodou této metody je velké množství booleovských operací, které se při této metodě musí provést. Další nevýhodou je výsledný povrch obrobků, který bude zubatý, což neodpovídá realitě. Ilustrativní ukázka na obrázku 6-3.



Obrázek 6-3: Ilustrativní ukázka řezu po bodech

Z tohoto důvodu je lepší variantou zpracovat příchozí data po blocích. Vytvořit jakýsi model dráhy nástroje, který bude pokrývat celou plochu, kterou nástroj opracuje, od počátečního bodu bloku až ke konečnému bodu bloku. Tímto způsobem zabráníme vzniku zubů na povrchu obrobku a taktéž si ušetříme značný počet booleovských operací. Následuje ukázka, jak takovéto bloky vypadají pro lineární pohyb a pro pohyb po oblouku.



Obrázek 6-4: Pohyb nástroje po blocích

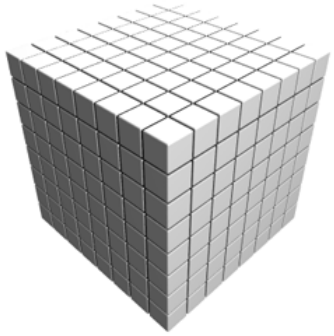
6.3. Grafická reprezentace

Ze všech informací, které jsme si zatím ukázali je zřejmé, že kritickým krokem pro naši simulaci je provedení booleovských operací. Je to dáno tím, že těchto operací musí být provedeno hodně. Složitější NCP programy čítají až desetitisíce bloků. Je tedy jasné, že grafická reprezentace musí být schopna provést značné množství booleovských operací v přijatelném čase. Tato problematika je i v dnešní tobě stále ještě předmětem vývoje. Existuje řada metod pro řešení tohoto problému (např. metoda využívající dexely [4]). Jelikož jsme ale již dříve určili, že využijeme grafického jádra Open CASCADE, musíme se omezit pouze na prostředky, které toto jádro poskytuje. V dokumentaci tohoto jádra je uvedeno, že podporuje práci s voxely. Rozhodli jsme se tedy pokračovat dále s využitím této datové struktury.

(Jak se ukázalo až během závěrečné fáze implementace - Open CASCADE práci s voxely ve skutečnosti nepodporuje, ale pouze "napodobuje" jejich chování. Dokonce se v současné době ani nepracuje na jejich vývoji. Z tohoto důvodu není ve výsledné implementaci práce s voxely zahrnuta, ale používá se pouze klasická povrchová reprezentace. Více viz kapitola 8.4)

6.3.1. Objemová reprezentace

Údaje do této části čerpám z knihy Moderní počítačová grafika [5] a z příslušné části dokumentace Open CASCADE a taktéž z článku o reali-time vytváření modelů [6]. Voxel je pojem, který vznikl, jako analogie k 2D pixelu. Jedná se o nejmenší element v 3D diskrétním prostoru. Jelikož se jedná o 3D prostor, voxely mají tvar kvádru či krychle a bývají uspořádány do pravoúhlé mřížky. V celém svém objemu má voxel konstantní hodnotu (stejně jako pixel). Jak je vidět z ilustračního obrázku 6-5.



Obrázek 6-5: Voxely - ilustrace

Inicializace takovéhoto modelu může být různá. Nám však stačí použít tzv. booleovskou inicializaci. Jedná se v podstatě o pole booleovských hodnot, kde pro každý inicializovaný voxel existuje booleovská hodnota. Tato hodnota jednoduše značí pro log. 1 voxel existuje, pro log. 0 voxel neexistuje. Když se tedy později během běhu programu provede booleovská operace například *cut* (řez), tato operace jednoduše nastaví hodnoty příslušných voxelů na log. 0. Tento návrh vychází z publikovaného článku [6], kde tímto způsobem dosáhli až real-time Volume Sculpting - tedy tvarování objemové reprezentace modelu v reálném čase.

6.4. Shrnutí návrhu simulace

Pokud tedy shrneme veškeré údaje, které jsme v této kapitole prozkoumali, můžeme si již navrhnout metodu, jak efektivně provést simulaci obrábění CNC stroje. Ze zadání této práce je jasné, že naším cílem není vytvořit přesnou simulaci obrábění, ale spíše vytvořit model, který by sloužil pro vizualizaci toho, co se právě se strojem děje. Z tohoto důvodu nám stačí načíst předpřipravené bloky programu ze systému, taktéž je pro nás výhodné využít tvorby bloků pohybu nástroje. Díky těmto volbám značně snížíme počet booleovských operací, který je nutný k vytvoření výsledného modelu. Pro realizaci této metody použijeme nejvhodnější nástroj, kterým open CASCADE disponuje - totiž voxely.

7. Napojení modulu na systém

Nyní, když už máme připravené všechny teoretické základy můžeme začít s implementací našeho modulu. Jak jsme si ukázali výše, modul ke své správné funkci vyžaduje některé údaje ze systému MEFI. Napojení se na tento systém je tedy logickým prvním krokem implementace tohoto modulu.

7.1. Získávání informací ze systému

V tomto projektu se ze systému MEFI získává celá řada informací. Příkladem může být tvorba a umístění okna aplikace, kde skoro veškeré informace obstarává systém a nikoliv tato aplikace. Nicméně pro nás zřejmě nejdůležitějšími a také i do počtu nejčetnějšími informacemi ze systému získávanými, jsou informace o programu, který se má zobrazit na obrazovce.

Poté co obsluha zvolí NCP program, který chce obrábět, systém tento program načte a přeloží do tvaru, který potřebuje pro svou funkci. Díky COM rozhraní je možné tyto již přeložené informace získat a použít pro naše účely. Není tedy třeba se starat o překlad NCP programu do prostorového souřadnicového systému. Místo toho nám systém poskytne již připravené prostorové souřadnice a popřípadě i další informace, které potřebujeme. Následující kód ukazuje získávání právě těchto informací:

```
CComPtr<ICnc>           m_lpCnc; //Zakladni rozhrani systemu
CComPtr<ICncKrnl>       m_lpCncKrnl; //Rozhraní jádra systému
CComPtr<ICncChannel>    m_lpChannel; //Rozhraní zadaného kanálu
CComPtr<ICncRTM>        m_lpCncRTM; //Rozhraní modulu reálného času

//Postupné získání jednotlivých rozhraní
VERIFY(m_lpHost->GetCnc(&m_lpCnc) && m_lpCnc);
...
VERIFY(m_lpCnc->GetCncKrnl(&m_lpCncKrnl) && m_lpCncKrnl );
...
VERIFY( m_lpCncKrnl->GetChannel(m_iChannel, &m_lpChannel) &&m_lpChannel );
...
VERIFY( m_lpChannel->GetRTM(&m_lpCncRTM) || !m_lpCncRTM );
```

Nyní jsme získali ukazatele *m_lpCncChannel* na potřebný kanál, ze kterého budeme čerpat informace, a to přes rozhraní *ICnc* a *ICncKrnl*. *CComPtr* je inteligentní ukazatel z knihovny ATL (Active Template Library), který nám velice usnadní práci s COM rozhraním. Sám se stará například o správné řízení funkcí *AddRef* a *Release*.

```
if (!m_lpRI)
{
      //Získat informace o záznamu
      BOOL bMainRec = FALSE;
      if ( m_lpCncChannel && m_lpCncChannel->
                        CreateRecordInfo(&bMainRec, &m_lpRI) && m_lpRI )
      {
```

Zde se z m_lpCncChannel pokoušíme naplnit proměnnou *m_lpRI*, která pro nás představuje veškeré informace o zvoleném programu. Pokud se tato operace nepovede, může to znamenat, že obsluha zatím žádný program nezvolila, tedy není co zobrazovat.

```
if ( m_lpRI )
{
...
      RECSTATUS RS;
      ...
      //kontrola, zda nám již systém připravil nějaké bloky pro vykreslení
      if (m_lpRI->GetStatus(&RS) && RS.dwBlocksPreparedCnt > 1
                                  && m_lpRI->MoveToFirstBlockPrepared())
      {
      ...
            RBINFO3D BI;      //informace o bloku
            if(m_lpRI->GetBlockInfo3D(&BI))
            {
            ...
```

Toto je vybraný sled kontrol, který proběhne již těsně před samotnou prací s daty vybraného programu. Pokud existuje m_lpRI znamená to, že obsluha zvolila program. Druhá podmínka kontroluje, zda pro nás systém již připravil alespoň nějaké bloky ke zpracování. Poslední naplňuje strukturu BI hodnotami aktuálního bloku, se kterými již můžeme pracovat.

Jak jsem si ukázali již dříve, NCP program je rozdělen na jednotlivé bloky, z nichž každý obsahuje určité informace a z toho jen některé potřebujeme pro vykreslení programu. m_lpRI obsahuje odkazy na všechny bloky zvoleného programu a díky metodám *m_lpRI->MoveToNextBlockPrepared()* nebo *m_lpRI->MoveToFirstBlockPrepared()* se můžeme v těchto jednotlivých blocích libovolně pohybovat. Následující tabulka ukazuje, jaké informace získáváme z jednotlivých bloků:

Typ eInterpolation	**Popis**
itNoInterpolation	Žádný pohyb.
itRapidTraverse	Rychloposuv. Získáváme prostorové souřadnice cíle posuvu.
itLinear	Lineární pohyb stroje. Získáváme prostorové souřadnice cíle pohybu.
itCircleCW	Kružnice ve směru hodinových ručiček. Získáváme souřadnice koncového bodu oblouku. Souřadnice středu kružnice. Informace o interpolační rovině a stoupání pro šroubovici.
itCircleCCW	Kružnice proti směru hodinových ručiček. Získáváme souřadnice koncového bodu oblouku. Souřadnice středu kružnice. Informace o interpolační rovině a stoupání pro šroubovici.
itThread	Závitování. My tyto informace nepoužíváme, protože nám pro závity stačí informace získané z itCircleCW a itCircleCCW.

Tabulka 7-1: Informace o programu získávané ze systému

8. Práce s Open CASCADE

V tuto chvíli se tedy dostáváme k hlavní části implementace, totiž k práci se samotným grafickým jádrem Open CASCADE. Protože jsem s tímto jádrem, před zahájením tohoto projektu, neměl žádné zkušenosti, inspiroval jsem se ve sbírce příkladů, které jsou obsaženy v distribuci tohoto jádra a na diskusních fórech, kde se k problematice vyjadřovali jeho uživatelé.

V následujícím textu si popíšeme, jakým způsobem jsem toto jádro používal v různých fázích práce tohoto modulu. Již zde nebudu rozepisovat nastavení operačního systému či vývojového prostředí, jelikož se to u každého uživatele liší, nicméně právě tato fáze je při zprovoznění Open CASCADE nejvíce problematická.

8.1. Inicializace

Při práci s Open CASCADE jsem využíval několik hlavních ukazatelů:

```
Handle_Graphic3d_WNTGraphicDevice m_pGraphicDevice;  //Grafické zařízení
Handle_V3d_View   m_pView;          //nastavuje pohled
Handle_V3d_Viewer m_pViewer;        //nastavuje prohlížeč
Handle_WNT_Window m_pWNTWindow;     //správa vytvořeného okna OCC
Handle_AIS_InteractiveContext m_pAISContext;   //interaktivní obsah v OCC
```

m_pGraphicDevice je ukazatel na třídu, která se stará o inicializaci grafického zařízeni pod Windows. Tuto inicializaci provedeme následovně:

```
m_pGraphicDevice = new Graphic3d_WNTGraphicDevice();
```

Nyní musíme nastavit , jak chceme, aby se náš obsah zobrazoval (osvětlení, atd.). Toho dosáhneme pomocí nastavení *m_pViewer*.

```
//vytvoření prohlížeče m_pViewer podle grafického zařízení m_pGraphicDevice
m_pViewer = new V3d_Viewer(m_pGraphicDevice, (short *) (""));
m_pViewer->SetDefaultLights();
m_pViewer->SetLightOn();
```

Následuje inicializace *m_pAISContext*, která reprezentuje veškeré modely na scéně a jejich nastavení (materiál, barva, některé vlastnosti atd...).

```
//vytvoření interaktivního obsahu m_pAISContext podle prohlížeče m_pViewer
m_pAISContext = new AIS_InteractiveContext(m_pViewer);
```

Nakonec si vytvoříme jeden pohled na scénu.

```
//vytvoření pohledu
m_pView = m_pViewer->CreateView();
```

Když máme všechny tyto inicializace hotové, pomocí COM získáme od systému *handle* okna a tyto informace předáme Open CASCADE. Podle těchto informací se nové okno zobrazí a my můžeme začít pracovat.

```
m_pWNTWindow = new WNT_Window( m_pGraphicDevice, m_pPic.m_hWnd);
m_pView->SetWindow(m_pWNTWindow);
```

8.2. Model dráhy nástroje

Prvním dílčím krokem při práci na tomto modulu bylo vytvoření modelu dráhy nástroje. Tento krok je logický, neboť tento modul bude s dráhou nástroje po celou dobu pracovat a nám se tak bude hodit, když budeme mít k dispozici možnost si tuto dráhu zobrazit.

Jedná se v podstatě o jednoduchou funkci, kdy pomocí rozhraní si ze systému načteme informace o jednotlivých blocích. U každého bloku pak dráhu znázorníme pomocí křivky. Budeme však barevně rozlišovat, zda se jedná a rychloposuv, klasickou dráhu nástroje (tedy dráhu s poloměrovou korekcí) či zda se jedná o hranu obrobku.

Naším prvním krokem logicky bude, že zkontrolujeme, zda pro nás systém již připravil nějaké bloky.

```
if (m_lpRI->GetStatus(&RS) && RS.dwBlocksPreparedCnt > 1
                                    && m_lpRI->MoveToFirstBlockPrepared())
{
```

Poté vložíme do smyčky náš vykreslující algoritmus.

```
do{

}while(m_lpRI->MoveToNextBlockPrepared());
```

V každém cyklu si načteme aktuální blok programu.

```
if(m_lpRI->GetBlockInfo3D(&BI))
{
```

Podle hodnot v aktuálním bloku se rozhodneme, jakým způsobem blok zpracujeme.

```
switch (BI.eInterpolation)
{
```

Tyto výše popsané kroky, jsou stejné i u dalších funkcí tohoto modulu a tak je již znovu nebudu popisovat. Pokud tedy dostaneme za úkol nakreslit přímku, tak to učiníme následovně:

```
TopoDS_Edge aEdge; //(průběžná) hrana
...
aEdge = BRepBuilderAPI_MakeEdge(aPnt_Old, aPnt_New); //hrana (odkud -> kam)
```

U kružnice:

```
gp_Dir dir;       //vektor určující orientaci kružnice
gp_Ax2 ax2;       //orientovaný bod středu kružnice
gp_Circ circ;     //kružnice / oblouk

dir = gp_Dir( 0, 0, 1);            //pro rovinu XY
ax2 = gp_Ax2( gp_aPntCenter, dir); //kromě polohy středu určuje i natočení

//tvorba kružnice (střed, poloměr)
circ = gp_Circ( ax2, gp_aPntCenter.Distance(aPnt_Old));
```

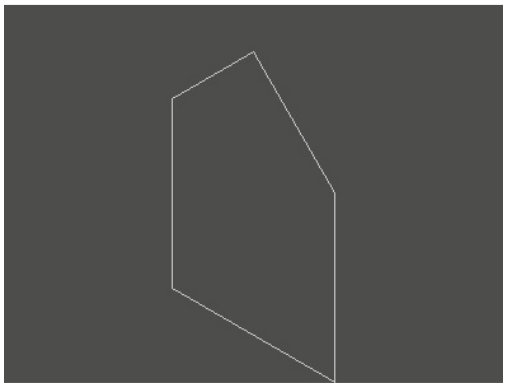
Nyní musíme rozlišit, zde se jedná o oblouk či o kružnici. Toto rozpoznáme tím, že u kružnice se nově příchozí cílový bod shoduje s počátečním. U oblouku tento nový bod určuje, kam až má oblouk sahat. V kompletní implementaci navíc musíme rozlišovat, zda je oblouk po či proti směru hodinových ručiček.

```
aEdge = BRepBuilderAPI_MakeEdge(circ);    //tvorba kružnice

//tvorba oblouku (kružnice, cíl, počátek)
aEdge = BRepBuilderAPI_MakeEdge(circ, aPnt_New, aPnt_Old);
```

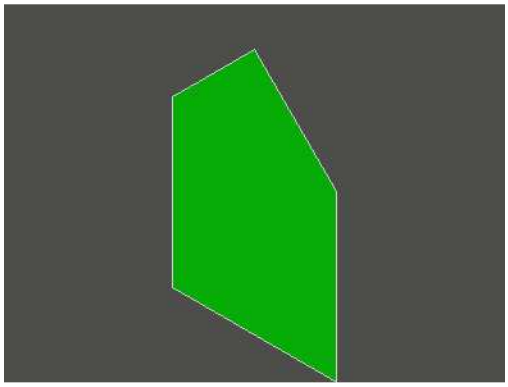
8.3. Model opracované plochy nástroje

Nyní již začneme pracovat na simulaci obrábění. V 6. kapitole jsme si určili, že tuto simulaci provedeme pomocí vymodelovaných bloků reprezentujících plochu, kterou nástroj pokryje. Abychom takovýto model mohli vytvořit, musíme postupovat podle topologie Open CASCADE (viz 4. kapitola). První, co musíme vytvořit, jsou hraniční body. Tyto body poté pospojujeme pomocí *edges* (hran). Všechny jednotlivé *edges* spojíme do jednoho celku zvaného *wires*. Na následujících obrázcích budu demonstrovat tento postup na příkladě.



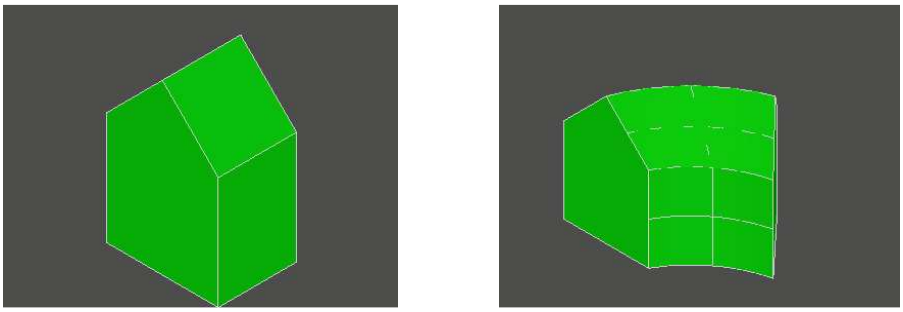
Obrázek 8-1: Příklad wire

Když budeme mít *wires* připravené, můžeme jeho pomocí vymodelovat *face*, tedy plochu ohraničenou *wires*.



Obrázek 8-2: Příklad face

V posledním kroku *face* " roztáhneme" (v angl. originále: sweeping) podél vodící *wire*. Tato vodící *wire* je vytvořená podle *edge*, která byla definována z načtené dráhy nástroje (viz kap. 8.2). Výsledek se samozřejmě bude lišit, pokud se jedná o lineární blok či o oblouk.



Obrázek 8-3: Příklad hotového modelu

Pokud známe tento teoretický základ, samotná implementace už není tak složitá. Začneme inicializací bodů, to vypadá například následovně:

```
gp_Pnt aPnt1(0, 0, 0);
```

Z bodů zformujeme *edges*:

```
TopoDS_Edge aEdge1;
aEdge1 = BRepBuilderAPI_MakeEdge(aPnt2,aPnt1); //tvorba Edge
```

Z *edge* pak *wire*:

```
TopoDS_Wire aWire1;
aWire1 = BRepBuilderAPI_MakeWire(aEdge1, aEdge2); //tvorba Wires
```

A nakonec z hraničního *wire* `aWireCompl`, utvoříme *face*:

```
TopoDS_Face aFace;
aFace = BRepBuilderAPI_MakeFace(aWireCompl); //tvorba Face
```

V tuto chvíli jsme tedy připraveni na tvorbu modelu. Ten bude utvořen pomocí metody `BRepOffsetAPI_MakePipe`, která určený *face* "roztáhne" podél zvoleného wire.

```
TopoDS_Shape aShape;    //tvar vytvořený podle pohybu nástroje
aShape = BRepOffsetAPI_MakePipe( aMainWire, aFace);
```

8.4. Řez

Na tomto místě bych měl popisovat práci Open CASCADE s voxely a booleovskými operacemi, což nakonec udělat nemohu. Důvodem je to, že i když Open CASCADE má k dispozici třídy jako `Voxel_FastConverter` (pro převod modelu z `TopoDS_Shape` na voxely) či `Voxel_BooleanOperation` (pro booleovské operace nad voxely), tyto třídy nejsou v tuto chvíli hotové a ani funkční. Kontaktoval jsem v této souvislosti i vývojový tým Open CASCADE a ti mi potvrdili, že voxely v Open CASCADE skutečně nefungují. Navíc se v době psaní teto diplomové práce na vývoji této problematiky v Open CASCADE vůbec nepracuje. Zmíněné třídy jsou tedy jen jakýsi nefunkční polotovar.

Z výše uvedeného důvodu je tedy *cut* (řez) zhotoven v tomto modulu pomocí klasické povrchové reprezentace. Jelikož jsme již v předcházejících částech implementace zhotovili všechny potřebné části, provedení samotného řezu už není nikterak náročné. `aShapeComplete` a `aShape` jsou modely typu `TopoDS_Shape`, nad kterými se operace provede. Celý postup je okomentován v následující ukázce kódu.

```
BRepAlgoAPI_Cut cut1(aShapeComplete, aShape); //nastavení řezu
cut1.Build(); //provedení řezu

if(cut1.IsDone()) //je řez hotov?
{
	aShapeComplete = cut1.Shape(); //pokud ano - uložit výsledný model
```

Nesmíme ještě zapomenout na návaznost bloků. V předchozí části jsme vytvořili jednotlivé bloky a v této části jsme je vyřezali do polotovaru. Kdybychom to však nechali takto být, tak v místě, kde na sebe dva bloky navazují, by mohli zůstat viditelné nepřesnosti. Z tohoto důvodu se ještě na závěr této sekvence v místě dotyku vytvoří válec a i ten se odečte od polotovaru.

8.5. Zobrazení

Nyní je model tedy hotov a my ho musíme zobrazit. K tomu využijeme již dopředu inicializované ukazatele, především pak `m_pAISContext.` Zobrazení se v Open CASCADE učiní jednoduše tak, že vytvoříme nový interkativní obsah, který slouží v podstatě jako jakási obálka pro náš model. Učiní se to následovně:

```
Handle(AIS_Shape) ais2;
ais2 = new AIS_Shape(aShape);
```

Do této obálky však nemusíme ukládat pouze modely typu `TopoDS_Shape`, ale například i *edge*, *wire* či *face* a další. Když máme tuto obálku vytvořenou, musíme nastavit její vlastnosti. K tomu právě využijeme `m_pAISContext.` Těchto vlastností je velké množství, zde je jen několik z nich:

```
m_pAISContext->SetDisplayMode(ais2,1,Standard_False);
m_pAISContext->SetColor(ais2,Quantity_NOC_RED,Standard_False);
m_pAISContext->SetMaterial(ais2,Graphic3d_NOM_PLASTIC,Standard_False);
m_pAISContext->Display(ais2,Standard_False);
m_pAISContext->SetCurrentObject(ais2,Standard_False);
```

A na závěr dáme pokyn aktuálnímu pohledu, aby vycentroval svou kameru tak, aby zabírala celou scénu:

```
m_pView->FitAll();
```

8.6. Import / Export

V dodatečných cílech této práce jsem se zmínil (kap. 1.3) o potřebě importovat a exportovat modely z a do tohoto modulu. Zde se ukázala veliká výhoda použití grafického jádra. Exportovat model do nějakého formátu je totiž složitá operace a sama o sobě by dala na téma další diplomové práce. My však využíváme grafické jádro a to celou tuto proceduru udělá za nás.

Export vybraného modelu do STEPu pak vypadá následovně (`aSequence->Value(i)` udává námi vybraný model):

```
STEPControl_StepModelType aValue = STEPControl_AsIs;
STEPControl_Writer aWriter;

if(!Interface_Static::SetIVal("write.precision.mode",1))
      return S_OK;

IFSelect_ReturnStatus status;

for (Standard_Integer i = 1; i <= aSequence->Length(); i++)
{
      status = aWriter.Transfer(aSequence->Value(i), aValue); //export STEP
      if ( status != IFSelect_RetDone ) return S_OK;
}

status = aWriter.Write(aFileName);
```

Načtení modelu je ještě jednodužší:

```
CImportExport::ReadSTEP(m_pAISContext);

m_pView->FitAll();
```

9. Prezentace a zhodnocení výsledků

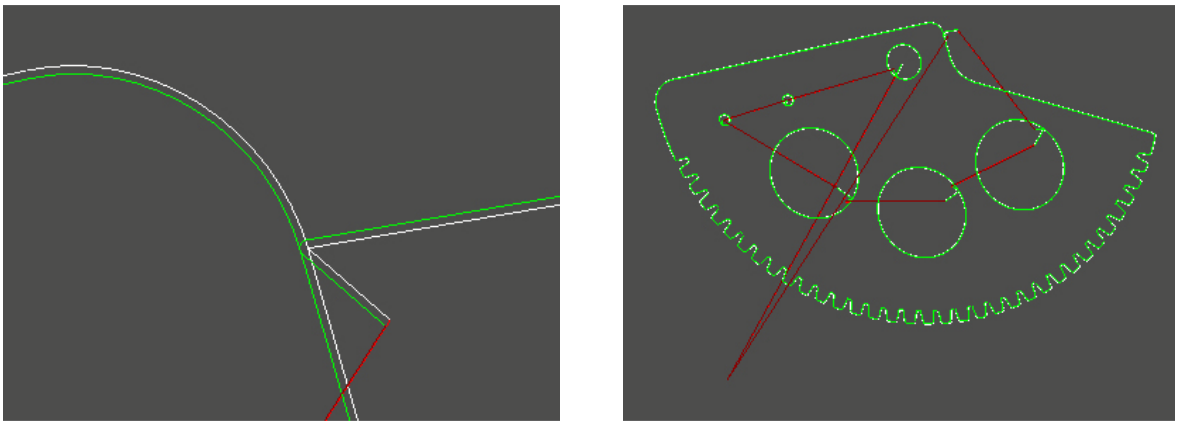
A jsme hotovi. V tuto chvíli máme modul plně implementovaný. Je tedy na čase, abychom si představili výsledky práce, tedy to, co tento modul dokáže. Předvedeme si schopnost tohoto modulu vykreslit model dráhy nástroje, model plochy, kterou nástroj obrobí, výsledky importu a exportu a nakonec samozřejmě výsledky řezu. Protože hlavním cílem tohoto modulu jsou výsledky řezů, otestujeme je i ohledně časové náročnosti. Další vedlejší funkce jako jsou funkce posuvu / rotace / zoomu či funkce související s návazností na systém prezentovat nebudeme. Veškeré testování proběhlo na stejné konfiguraci počítače. Obrázky v této kapitole jsou sejmuty z tohoto počítače. Použité testovací programy jsou uloženy na přiloženém DVD.

Komponenta	**Hodnota**
CPU	Intel Pentium 4 3.00GHz HT
RAM	512MB
OS	Windows XP SP3
Grafická karta	NVIDIA GeForce4 MX 4000
Verze Open CASCADE	6.3.0
Verze systému MEFI CNC 872	0.9 Rev. 768
Vývojové prostředí	Microsoft Visual Studio 2008

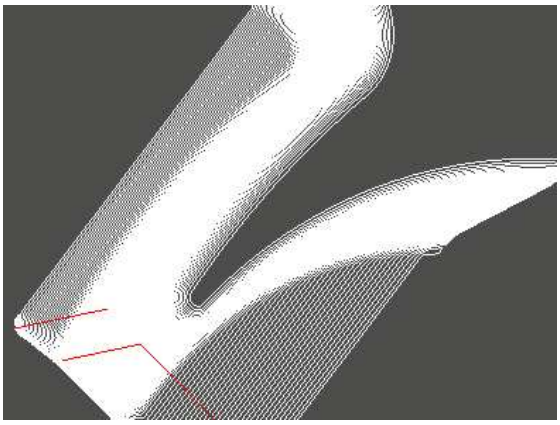
Tabulka 9-1: Relevantní konfigurace PC

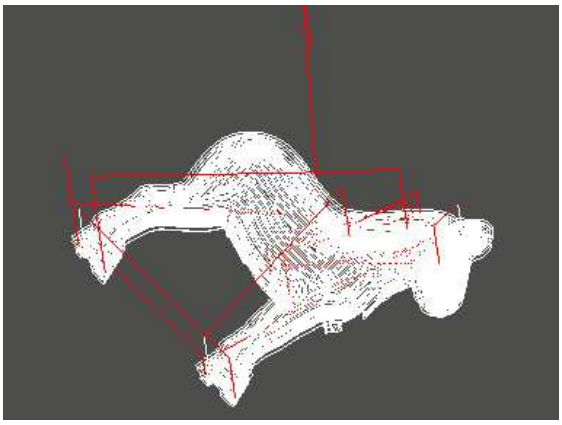
9.1. Model dráhy nástroje

Model dráhy nástroje, tedy pomocí křivky znázorněný pohyb nástroje. Na obrázcích bílá křivka reprezentuje dráhu bez poloměrové korekce. Zelená reprezentuje dráhu s touto korekcí. Červená pak rychloposuv. Poloměrovou korekcí rozumíme posunutí nástroje o jeho poloměr (o tento výpočet se stará systém). Zelená křivka je tedy křivka, která indikuje pohyb středu nástroje. Bílá křivka indikuje hranu výsledného obrobku.



Obrázek 9-1: Zubatka





Obrázek 9-2: Forma 203 a Kůň CEN182

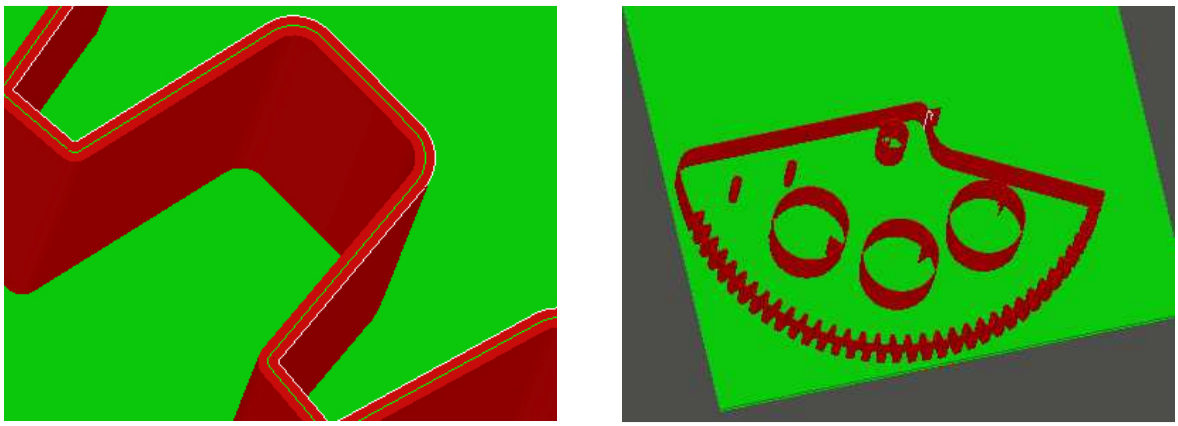
	Hodnota
Zubatka	
Zdroj:	Zubatka.NCP
Počet bloků:	242
Průměrná doba vykreslení:	0,02s
Vykreslení současným modulem:	46,6s
Forma 203	
Zdroj:	203.NCP
Počet bloků:	22 983
Průměrná doba vykreslení:	51,5s
Vykreslení současným modulem:	48min 58s
Kůň CEN182	
Zdroj:	CEN182.NCP
Počet bloků:	48 361
Průměrná doba vykreslení:	5min 31s
Vykreslení současným modulem:	1h 49min 43s

(pozn.: Údaj "Vykreslení současným modulem" je doba od načtení programu do úplného zobrazení programu)

Tabulka 9-2: Údaje o zobrazovaných programech

9.2. Model opracované plochy nástroje

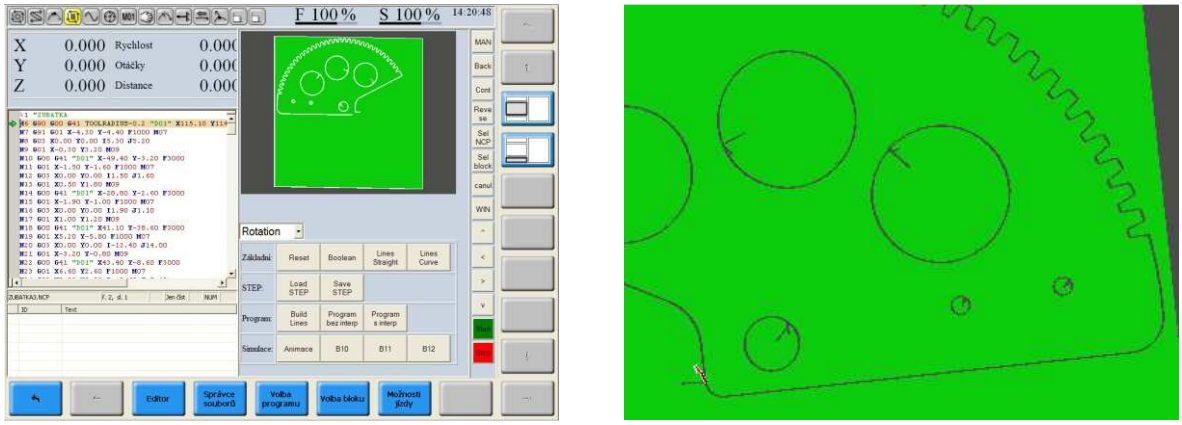
Model opracované plochy nástroje je model, který se v této formě běžně nepoužívá. Jde pouze o vnitřní meziprodukt během práce na vytvoření kompletního vyřezaného modelu. Z tohoto důvodu ho zde uvádím jen spíše pro ilustraci a demonstruji ho pouze na jednom programu - konkrétně pak Zubatka.NCP.

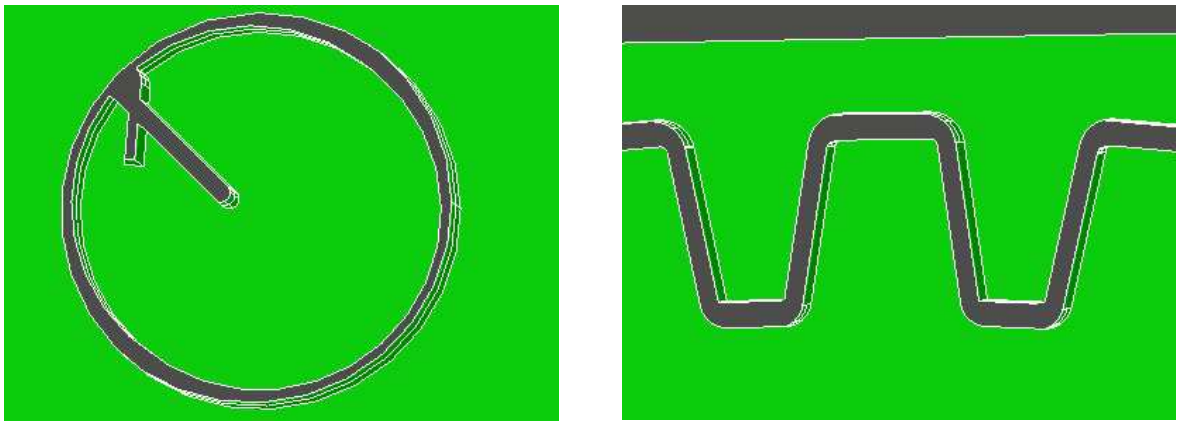


Obrázek 9-3: Model opracované plochy nástroje

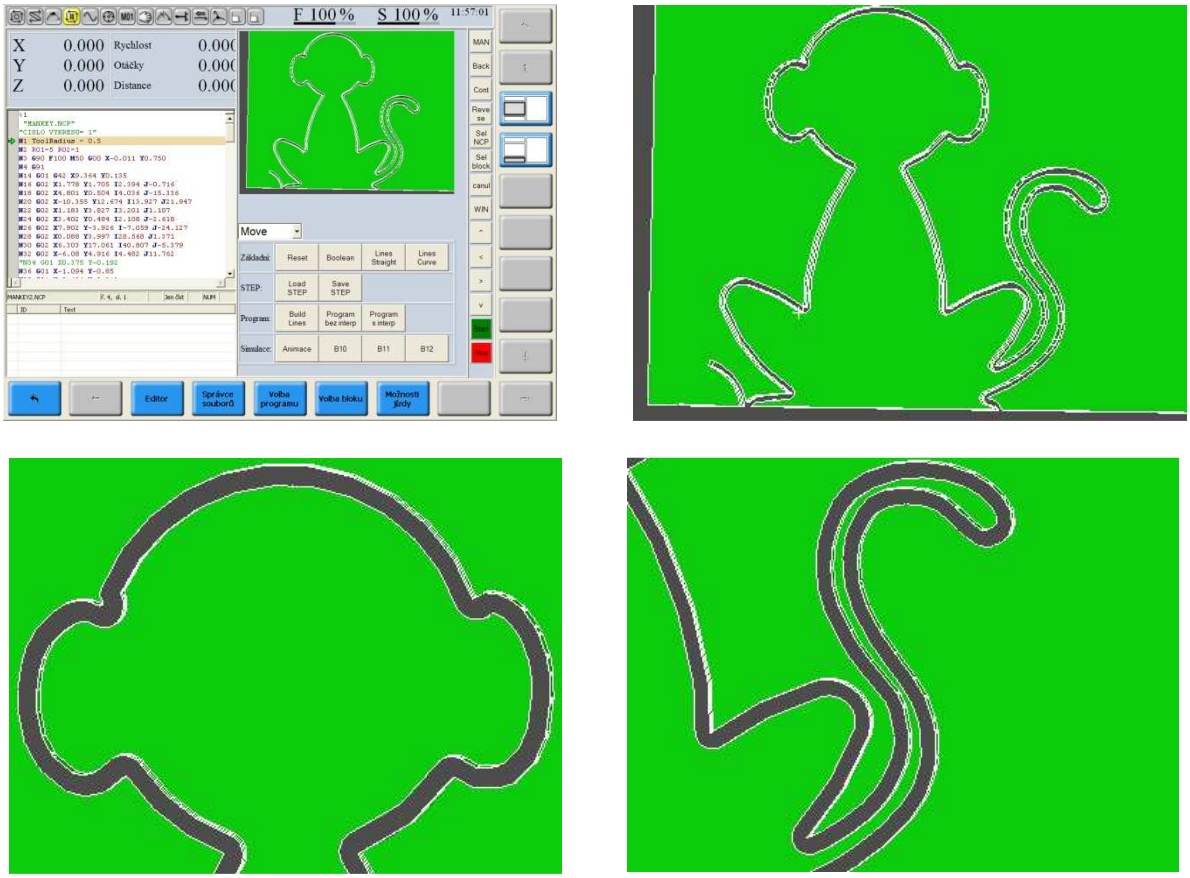
9.3. Řez

Nyní si předvedeme výsledky práce modulu při řezech na různých reálných programech. Je zde vždy zobrazena celá plocha systému bez jakýchkoliv změn, náhled na program a případně několik detailů na zajímavé časti programu, které mohou odhalit přesnost řezů. V závěru této části jsou uvedeny číselné údaje o velikosti programu a časové náročnosti tvorby výsledného modelu.

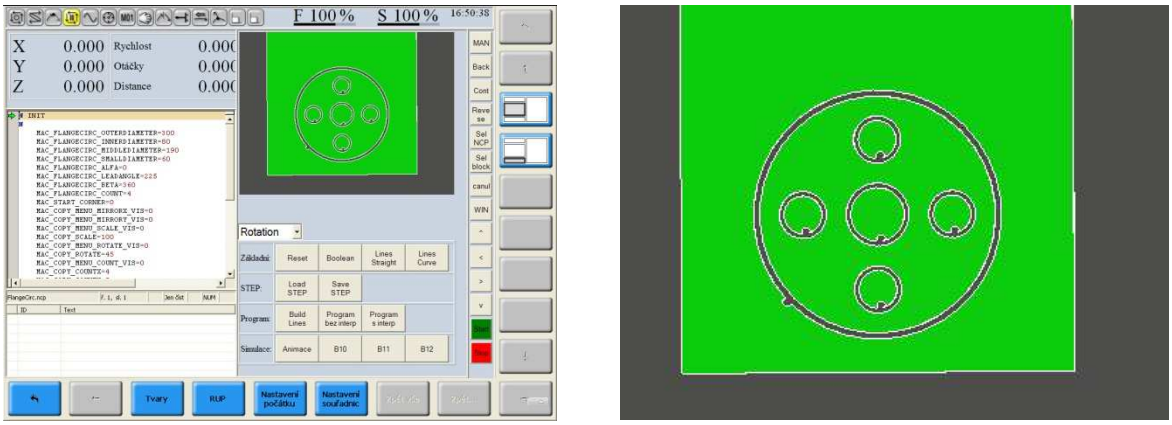




Obrázek 9-4: Výsledky - Zubatka



Obrázek 9-5: Výsledky - Monkey



Obrázek 9-6: Výsledky - Pevný tvar

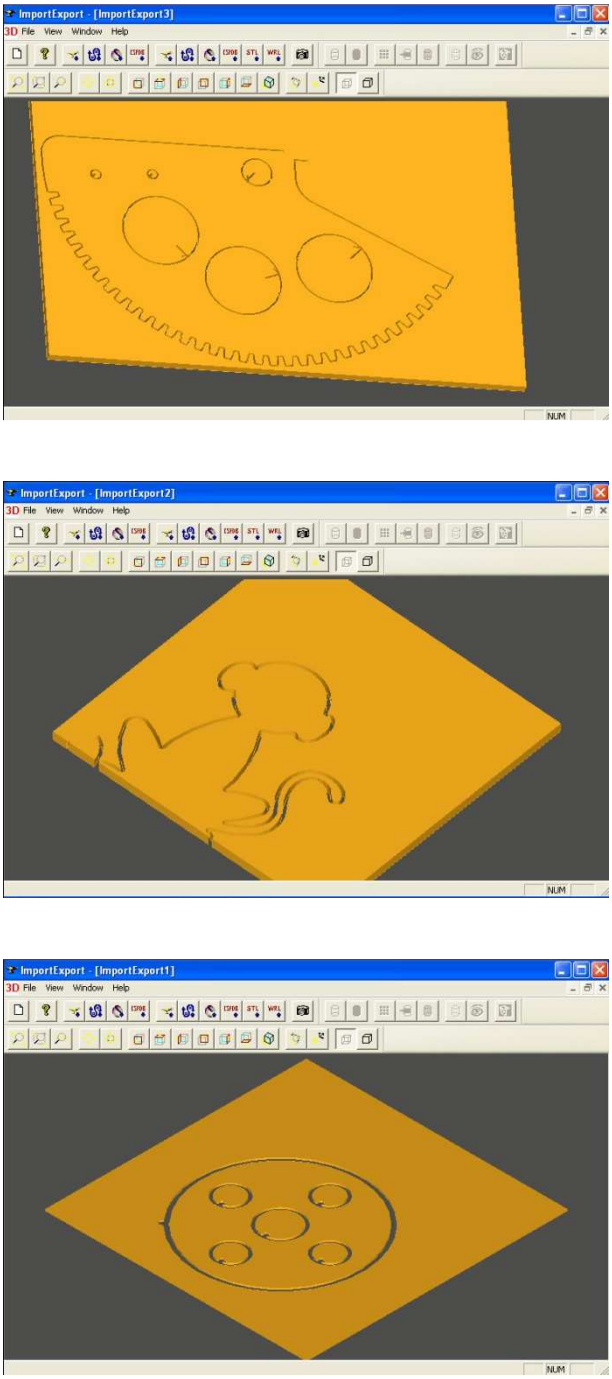
	Hodnota
Zubatka	
Zdroj:	Zubatka.NCP
Počet bloků:	242
Průměrná doba vykreslení:	42min
Monkey	
Zdroj:	Monkey.NCP
Počet bloků:	143
Průměrná doba vykreslení:	4min 2s
Pevný tvar	
Zdroj:	- (Pevný tvar je generován přímo systémem)
Počet bloků:	24
Průměrná doba vykreslení:	31s

Tabulka 9-3: Údaje o provedených řezech

Z výše uvedených hodnot je jasné, že nasazovat tento modul na složitější programy není možné. Taktéž je vidět, že pokusit se animovat proces, jehož výpočet trvá až 42min ztrácí praktický význam. Z tohoto důvodu tyto funkce do výsledného modulu nezahrnuji. Nicméně pokud by byl v budoucnu použit postup řezu s menší časovou náročností, implementace těchto funkcionalit by nebyla nikterak složitá.

9.4. Import / Export

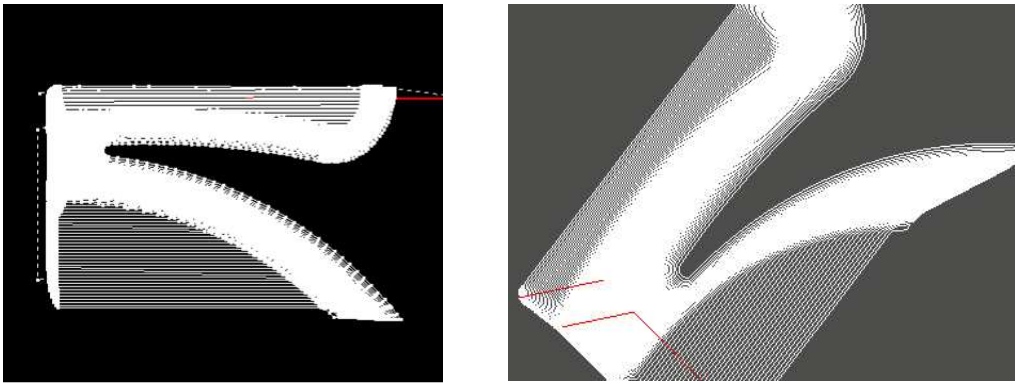
Tato část je výsledkem práce na Dodatečných cílech (viz kap. 1.3). Jelikož je tato funkcionalita plně funkční, všechny programy, na kterých jsem zhotovil řez, jsem vyexportoval do STEPu. Abychom mohli výsledek porovnat, zobrazil jsem v externím programu výsledné exporty. Tyto soubory jsou opět k dispozici na DVD (zubatka.step; monkey.step; priruba.step).



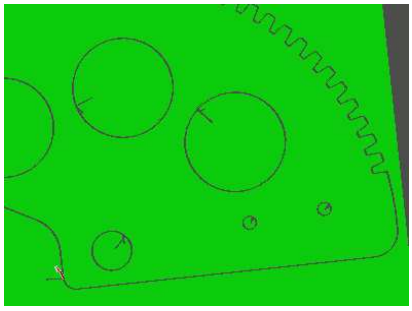
Obrázek 9-7: Výsledky - Export do STEP

10. Zhodnocení výsledků

V předchozí kapitole jsme prezentovali to, co tento modul dokáže. Nyní je třeba kriticky zhodnotit výsledky jeho práce. První prezentovanou funkcionalitou bylo vykreslení modelu dráhy nástroje. Tato funkcionalita fungovala velmi dobře. Z prezentovaných obrázků je vidět, že tato funkce dokázala pěkným způsobem zobrazit i velmi obsáhlé programy a to v přijatelném čase. Je složité porovnat její časovou náročnost se stávajícím modulem, neboť ten je implementován způsobem, aby nezatěžoval systém svou činností, tedy pracuje s výpočetním výkonem systému pouze v případě, když mu dá systém pokyn, že tento výkon nepotřebuje. Navíc má stávající modul nastavené některé omezující podmínky, jako jsou maximální počet zobrazených bloků během prováděné rotace atd.. Tyto vlastnosti náš modul nemá a tak není možné je mezi sebou relevantně porovnat. Z obrázku detailu 9-1 je vidět, že segmenty dráhy na sebe pěkně navazují. Navíc, implementace této funkcionality nebyla nikterak složitá. Tuto funkci lze tedy považovat za úspěšnou a potencionálně využitelnou v systému MEFI.



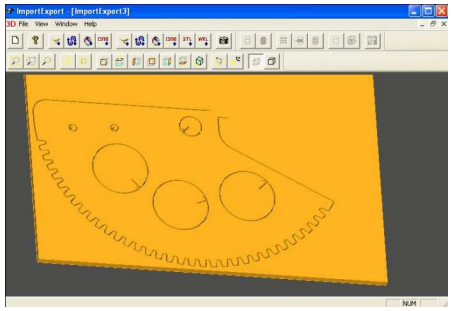
Obrázek 10-1: Porovnání náhledů programu



Naším hlavním cílem však bylo dokázat vytvořit model, který by reprezentoval již hotový obrobek. Z výsledků lze usoudit, že tohoto cíle se podařilo dosáhnout pouze částečně. Pomocí grafického jádra a klasických booleovských operací jsme dokázali vytvořit modely jednodušších programů. Tyto modely vypadají po grafické stránce hezky a dostatečně věrně reprezentují zvolený program. Nicméně vytvoření i těchto jednodušších modelů trvalo značné množství času - pro praktické použití příliš dlouhé. Z naměřených časových údajů lze také rozpoznat, že doba vytvoření těchto modelů je závislá na výsledném tvaru modelu. Například vytvoření Zubatky s 242 bloky trvalo celých 42min, ale Monkey s pouhou polovinou bloků (143) se modeloval za něco málo přes 4min. Obě tyto vlastnosti modulu brání v jeho praktickém použití.

Hlavním důvodem této časové náročnosti modulu je použití klasické povrchové reprezentace. Pokud bychom dodrželi teoretický návrh, který jsme si připravili v 6. kapitole, měli bychom, podle publikací v této kapitole uvedených, být schopni výše zmíněnou funkci vykonávat dokonce v reálném čase. Jádro Open CASCADE nám v této možnosti zabránilo (viz kap. 8.4).

Poslední oblastí je export či import modelů do grafického formátu STEP. I v tomto případě můžeme uvést, že modul fungoval dobře. Exportovali jsme námi vytvořené modely do STEPu a úspěšně jsme je načetli v externím programu. Je nutné dodat, že tato funkce by nebyla možná bez použití grafického jádra. Kdybychom veškeré funkce v tomto modulu museli implementovat sami, rozhodně bychom tuto funkcionalitu nenaimplementovali v tak krátkém časovém horizontu. Jelikož, jak jsme již ukázali v kapitole 1.3, je tato funkce nepostradatelná pro úplnou funkčnost tohoto modulu při reálném nasazení, je tedy také jedním z velkých argumentů, proč používat grafická jádra při vývoji tohoto modulu i nadále.



10.1. Jakým způsobem pokračovat?

Jakým směrem tedy pokračovat při vývoji tohoto modulu? Ukázalo se, že využití grafického jádra při jeho vývoji je prospěšné. Modul díky tomu získá celou řadu funkcionalit, které by jiným způsobem získal jedině až po dlouhém vývoji. Avšak námi zvolené jádro Open CASCADE se ukázalo pro tento typ aplikace jako nevhodné. Proto bych doporučil pokračovat ve vývoji na tomto modulu s jiným grafickým jádrem. Aby se budoucí vývoj již nesetkal s typem problémů, kterým jsme čelili při práci na tomto projektu, doporučil bych pro příště použít některou z komerčních variant. Ideální by zřejmě v tomto směru bylo jádro ACIS, jelikož několik projektů, které měly podobný cíl jako my a vyřešily ho úspěšně, právě toto jádro použily k řešení (např.: http://www.machineworks.com). Je tedy velmi pravděpodobné, že jádro ACIS má všechny potřebné funkce.

Nicméně ani grafické jádro Open CASCADE není vhodné zcela přestat používat. Toto jádro ukázalo, že velmi obstojně dokáže zobrazovat dráhu nástroje a taktéž zvládá import a export do řady externích grafických formátů. Toto jádro by tedy bylo do budoucna možné využít právě k těmto jeho funkcím. Mohlo by například nahradit stávající modul systému, který se stará o náhled programů a rozšířit některé jeho funkce. Nespornou výhodou tohoto jádra je taktéž to, že je poskytováno zdarma.

10.2. Přínos této práce

Tato práce ukázala hned několik velmi zajímavých aspektů. V prvé řadě ukázala, jak vypadá vývoj, který je postaven na využití externího grafického jádra. Takovýto vývoj může být velmi efektivní. I firma, která se vývojem v oblasti počítačové grafiky primárně vůbec nezabývá, může touto metodou získat grafický modul na vysoké úrovni a to za relativně krátký čas vývoje.

Tato práce také ukázala potřebu pro mezioborovou spolupráci v této oblasti. Hlavními uživateli tohoto modulu jsou totiž odborníci z oboru strojírenství. To oni udávají co a jakým způsobem by rádi viděli zobrazené na monitoru systému. Vývoj sytému zase patří čistě do oboru řídicí techniky. Tito odborníci zase zásobují tento modul potřebnými informacemi. Samotná implementace modulu však zasahuje do oboru počítačové grafiky. Protože do této problematiky zasahuje tolik oborů, existuje pouze málo firem, které se tomuto vývoji skutečně věnují. Po dobrých řešeních v této problematice tedy existuje značná poptávka a je zde i možnost dalšího uplatnění.

11. Závěr

V závěru bych rád řekl, že problematika, které se tato práce věnovala, je dosti rozsáhlá. Sahá od porozumění a použití pokročilých programátorských metod, přes základní pochopení funkcí řídicího systému CNC stroje až po výzkum v oblasti počítačové grafiky. Přesto, že konečná podoba modulu nedosahuje v plném rozsahu cílů zadání, tato práce byla přínosem a vývoj vyvíjeného modulu posunula o značný kus kupředu. Navíc prozkoumala celou řadu možností a různých cest, kterými se může budoucí vývoj, nejenom tohoto konkrétního modulu, ubírat.

Že se tato práce věnovala problematice, která není pouze teoretická, ale jedná se o skutečný problém z reálné praxe, který komerční firmy mají zájem aby byl vyřešen, svědčí i skutečnost zájmu ze strany Výzkumného centra pro strojírenskou výrobní techniku a technologii při ČVUT. Bez mého jakéhokoliv osobního přičinění mě tato skupina oslovila a požádala o spolupráci, neboť i oni sami hledali řešení tohoto problému a má metoda využití grafického jádra se jim zamlouvala. Je tedy vidět, že tato problematika je v současné době předmětem vývoje.

Za sebe bych rád uvedl, že práce na tomto projektu mě velice zaujala a bavila. Byla pro mě také velmi obohacující, neboť většina problematiky, se kterou jsem se v ní setkal, byla pro mě nová. Že s výsledky této práce byly spokojeni i její zadavatelé svědčí i nabídka, kterou jsem dostal, na další spolupráci při vývoji tohoto modulu.

A. Použitá literatura

[1] LYGIN, Roman. *OPEN CASCADE NOTES* [online]. 2009-02-04 [cit. 2011-05-09]. Topology and Geometry in Open CASCADE. Dostupné z WWW: <http://opencascade.blogspot.com/2009/02/topology-and-geometry-in-open-cascade.html>.

[2] *Spatian : Product Documentation* [online]. 2010-05-07 [cit. 2011-05-09]. Dostupné z WWW: <http://doc.spatial.com/index2.php>.

[3] PROSISE, Jeff. *Programování ve Windows pomocí MFC*. Bogdan Kiszka. Vydání druhé. Praha : Computer Press, 2002. 1135 s. ISBN 80-7226-309-9, K0373.

[4] Konig, A.H.; Groller, E.; , "Real time simulation and visualization of NC milling processes for inhomogeneous materials on low-end graphics hardware," *Computer Graphics International, 1998. Proceedings* , vol., no., pp.338-349, 22-26 Jun 1998 doi: 10.1109/CGI.1998.694284
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=694284&isnumber=15133

[5] ŽÁRA, Jiří, et al. *Moderní počítačová grafika*. Vydání první. Brno : Computer Press, 2004. 609 s. ISBN 80-251-0454-0, K1143.

[6] Sidney W. Wang and Arie E. Kaufman. 1995. Volume sculpting. In *Proceedings of the 1995 symposium on Interactive 3D graphics* (I3D '95). ACM, New York, NY, USA, 151-ff.. DOI=10.1145/199404.199430 http://doi.acm.org/10.1145/199404.199430

B. Seznam obrázků

Obrázek 1-1: Postup při práci s CNC stroji ... 3
Obrázek 2-1: Rolling_ball_blend surface od jádra Parasolid ... 8
Obrázek 2-2: Loga firem stojících za vývojem zkoumaných jader ... 9
Obrázek 2-3: Výsledek experimentu s jádrem Open CASCADE ... 11
Obrázek 3-1: Ukázky použití ACIS ... 13
Obrázek 3-2: Ukázka použití ACIS ... 14
Obrázek 3-3: Vztahy mezi topologickými entitami ACIS ... 15
Obrázek 4-1: Topologie Open CASCADE ... 19
Obrázek 4-2: Diagram TopoDS_Shape ... 20
Obrázek 4-3: Reference v topologii Open CASCADE ... 20
Obrázek 4-4: Vertex s vyznačenou tolerancí ... 21
Obrázek 4-5: Ukázka využití lokální tolerance ... 21
Obrázek 4-6: Edge a Face ... 22
Obrázek 4-7: Geometrický význam tolerance u křivky ... 23
Obrázek 4-8: Orientace normál face ... 24
Obrázek 4-9: Výpočet umístění materiálu ... 24
Obrázek 4-10: Význam tolerance u face ... 25
Obrázek 4-11: Ukázky práce Open CASCADE ... 25
Obrázek 5-1: Prostý objekt COM ... 28
Obrázek 6-1:Ukázky typů bloků ... 31
Obrázek 6-2: Souřadnice z interpolátoru ... 32
Obrázek 6-3: Ilustrativní ukázka řezu po bodech ... 33
Obrázek 6-4: Pohyb nástroje po blocích ... 33
Obrázek 6-5: Voxely - ilustrace ... 34
Obrázek 8-1: Příklad wire ... 39
Obrázek 8-2: Příklad face ... 39
Obrázek 8-3: Příklad hotového modelu ... 40
Obrázek 9-1: Zubatka ... 43
Obrázek 9-2: Forma 203 a Kůň CEN182 ... 44
Obrázek 9-3: Model opracované plochy nástroje ... 45
Obrázek 9-4: Výsledky - Zubatka ... 46
Obrázek 9-5: Výsledky - Monkey ... 46
Obrázek 9-6: Výsledky - Pevný tvar ... 47
Obrázek 9-7: Výsledky - Export do STEP ... 48
Obrázek 10-1: Porovnání náhledů programu ... 49

C. Seznam tabulek

Tabulka 2-1: Přehled modelovacích jader 7
Tabulka 3-1: ACIS - Podporované operační systémy 14
Tabulka 3-2: Popis topologických elementů modelu ACIS 15
Tabulka 5-1: Popis metod, které obsahuje rozhraní IUnknown 27
Tabulka 7-1: Informace o programu získávané ze systému 36
Tabulka 9-1: Relevantní konfigurace PC 43
Tabulka 9-2: Údaje o zobrazovaných programech 44
Tabulka 9-3: Údaje o provedených řezech 47

D. Seznam zkratek

CNC	Computer Numeric Control
CAD	Computer Aided Design
CAM	Computer Aided Manufacturing
NCP	Numeric Control Part program
COM	Component Object Model
CAE	Computer Aided Engineering
AEC	Engineering and Construction
CMM	Coordinate-Measuring Machine
IID	Identifikátor rozhraní COM
ATL	Active Template Library

E. Obsah přiloženého DVD

/Dip	Dokument obsahující tuto práci ve zdrojovém formátu + PDF
/Project	Obsahuje celý projekt spustitelný ve VS2008
/Code	Vybrané zdrojové kódy z projektu
/NCP	NCP programy, které byly použity k testování
/STEP	Exporty z tohoto modulu do formátu STEP
/Doc	Několik důležitých článků ([4], [6], dokumentace Open CASCADE o voxelech)
/OCC	Instalace Open CASCADE verze 6.3.0