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

# Bakalářská práce

# Detekce obrysů 3D objektů pro VRUT

Plzeň 2011 Lukáš Kurz

# Prohlášení

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

V Plzni dne 11. srpna 2011

Lukáš Kurz

# Abstract

The goal of the work is to create a module for detection of the 3D objects contours in the VRUT application. Used vectors approach for contours detection is based on limiting the group of the model edges only to those which meet the conditions of contour edges. The module was implemented in the C++ language and tested on the three models. The module found the right edges and these edges meet the requirement of the contour edges. The only problem is that the visibility of the contour edges is not properly solved. This could be, however, solved by the full implementation of the Appel's algorithm which is also described in this work.

# Obsah

# 1 Úvod

Cílem této práce je do již exitující aplikace VRUT (viz [Václav Kyba(2010)]) vytvořené ve spolupráci ŠKODA AUTO a.s. a ČVUT vytvořit modul, který umí u nahraného modelu detekovat obrysy, tyto obrysy zobrazovat (pokud možno v reálném čase), a umožnit export nalezených hran do PostScript (výsledkem bude 2D vektorový obrázek).

Nejprve čtenáře seznámím s aplikací VRUT, aby získal představu o její struktuře, možnosti rozšíření jejích vlastností pomocí nových modulů a způsobu komunikace modulů pomocí zpráv, a přiblížím možnosti programovacího jazyka PostScript (viz [Ado(1999)] a [Ado(1985)]), který je použit pro výsledný export nalezených hran.

Dále jsou zde popsány některé možnosti detekce obrysů, způsob implementace nového modulu pro aplikaci VRUT a popis jeho důležitých částí, a popis výsledného souboru v jazyce PostScript.

# 2 VRUT

## 2.1 Základní charakteristiky

VRUT (Virtual Reality Universal Toolkit) je aplikace pro vizualizaci a editaci 3D dat, vytvářená ve spolupráci ŠKODA AUTO a.s. a ČVUT. Je určen k zobrazení grafických dat s podporou modulů, umožňujících rozšířit funkci hlavní aplikace.

Aplikace se dá rozdělit na dvě části – hlavní aplikace neboli jádro, které je samo o sobě z uživatelského hlediska nepoužitelné, a základní balíček modulů, které se starají například o zobrazení scény či její import a export.

## 2.2 Jádro

Jádro slouží pro správu modulů, správu grafických dat, poskytuje pomocné prvky pro správný běh aplikace a také zajišťuje správu událostí sloužící jako komunikační kanál mezi jednotlivými částmi aplikace a moduly.

Správce modulu spuštění modulu a jeho propojení s jádrem. Moduly jsou aktivovány jen na žádost v případě potřeby. Správce modulů obsahuje dílčí části, z nichž každá je určena pro obsluhu podskupiny modulů stejného nebo kompatibilního typu. Některé moduly, u nichž je větší provázanost s jádrem, vyžadují zvláštní zacházení, proto typy správců do značné míry odpovídají typům podporovaných modulů.

## 2.3 Správa a distribuce událostí

Aplikace je rozdělena na objekty, především jednotlivé moduly, které jsou na sobě zpravidla nezávislé. Tyto části však spolu potřebují komunikovat. Zvolený způsob komunikace je postaven nezasílání událostí. Výhodou tohoto způsobu je, že není potřeba znát rozhraní jednotlivých objektů, ale stačí, aby každý objekt uměl přijmout a zpracovat událost. Nevýhodou však je obtížná synchronizace, protože zasílání událostí je pouze jednosměrná operace.

Synchronizace se řeší tak, že zaslaná událost vyvolá odezvu v podobě jiné události, na kterou objekty počkají v blokujícím módu.

Události jsou zasílány centrálnímu správci událostí, který se nachází v jádru, a ten je poté distribuuje příslušným objektům. Každý z objektů může přijímat libovolný typ událostí, stačí pouze, aby se u správce událostí zaregistroval k odběru daného typu událostí. Každý nově vytvořený modul tam může ihned reagovat na stávající události.

## 2.4 Graf scény

Graf scény je hlavní úložiště grafických dat. Protože se jedná o část, ke které je potřeba přistupovat často a z různých míst, je umístěn přímo v jádře a není ho možné nahradit externím modulem. Jeho datové struktury jsou jednoznačně určené a jsou společné pro všechny moduly, které s ním pracují.

Základním prvkem grafu scény je uzel. Může obsahovat přímo grafická data a může mít libovolný počet potomků. Každý uzel má též definovanou transformační matici, která definuje umístění tohoto uzle a jeho potomků jako logického celku v lokálním měřítku. V globálním měřítku lze získat vynásobením transformačních matic od kořene stromu po daný uzel. Tato transformační matice pro globální měřítko je však automaticky pro každý uzel přepočítávána, protože se jedná o údaj, se kterým se často pracuje.

Každá operace nad grafem scény vyvolá událost, která je zaslána správci událostí. Modul či jiný zaregistrovaný příjemce může mít tedy přehled o tom, co se ve vybrané scéně odehrává. Je umožněna i opačná komunikace, tedy pokud se správci událostí odešle nějaká událost pro operaci nad scénou, je tato událost předána a provedena skrze rozhraní scény.

Uzle grafu scény se podle funkčnosti dělí na několik typů. Mezi uzly není žádná závislost na typu, je možné kombinovat libovolné typy na pozici rodiče či potomka. Typu uzlů jsou:

**Základní uzel** – SceneNode, ASSEMBLY
: Představuje logický prvek ve scéně. Je výchozím typem, ze kterého vychází všechny ostatní uzly.

**Kořenový uzel** – SceneNode, ROOT

Jiné označení pro základní uzel, který je kořenovým uzlem scény.

**Kamera** – CameraNode, CAMERA
Uzel reprezentující kameru. Přidává automaticky aktualizovanou projekční matici a informace o pohledovém jehlanu včetně obálky v podobě koule.

**Uzel s geometrií** – GeometryNode, GEOMETRY
Přidává informace o geometrii a materiálu. Geometrie je popsána primitivy, ze kterých se skládá výsledný objekt, a jejich vlastností. Každá geometrie zahrnuje kromě grafických dat i obálku v podobě AABB v lokálním měřítku.

**Uzel se světlem** – LightNode, LIGHT
Představuje zdroj světla.

**Úroveň detailů** – LODNode, LOD
Speciální logický celek, který umožňuje uchovat různou kvalitu a složitost geometrie, spolu s informací při jaké vzdálenosti kamery od obálky uzlu se daná geometrie přepne. Vždy by měl být zobrazen pouze jeden potomek.

**Přepínač** – SwitchNode, SWITCH
Funguje jako přepínač mezi jednotlivými potomky. Můžou být též aktivovány všechny nebo žádný.

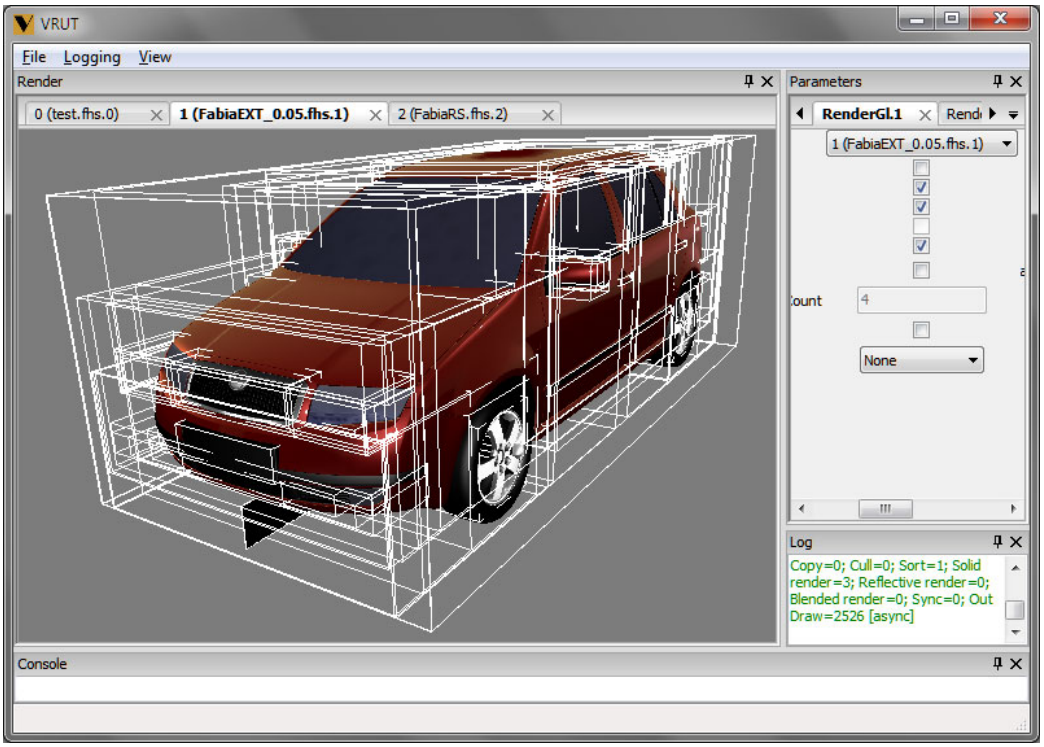
**Pozadí** – BackgroundNode, BACKGROUND
Definuje pozadí při zobrazení. Může se jednat o kulovou plochu pozadí nebo o jeden obrázek zobrazený jako pozadí.

**Zdroj zvuku** – SoundNode, SOUND SOURCE
Uzel představující zdroj světla.

## 2.5 Hierarchie obálek

Hierarchie obálek je implementována přímo v jádře. Slouží především k zefektivnění operací nad scénou, jako jsou ořezávání pohledovým jehlanem, detekce kolizí, či testy při vrhání paprsků.

Hierarchie obálek je strom, který representuje rozdělení scény na menší celky. Uzly stromu jsou pouze logickým celkem ve scéně, list představuje




Obrázek 2.1: Obálky zobrazené ve scéně

jeden uzel grafu scény s geometrickými daty. Každý uzel representuje část scény, která je vymezena určitým tělesem – obálkou. Tvar obálek je kvádr, zarovnaný s osami, tzv. AABB (Axis Aligned Bounding Box). Každá obálka potomka je plně obsažena v obálce rodiče, takže kořenový uzel představuje AABB celé scény.

Ke každé scéně je přiřazena pouze jedna hierarchie obálek a závisí výhradně na prostorové hierarchii geometrie ve scéně, nikoliv na transformační hierarchii scény. Na obrázku 2.1 jsou vidět zobrazené obálky modelu auta přímo ve scéně.

## 2.6 Moduly

Moduly umožňují aplikaci rozšiřovat o nové funkce. Moduly jsou aktivovány a spravovány příslušným správcem modulů. Pro jednotlivé moduly lze ak-

tivovat libovolné množství instancí a každé je přiřazeno vlastní vlákno, ve kterém probíhá činnost modulu.

Jak již bylo popsáno, komunikaci mezi jednotlivými moduly se zajišťuje pomocí událostí, a všechny moduly bez rozdílu mohou používat systém událostí jako příjemci i odesílatelé. Komunikace mezi moduly tak vždy probíhá skrze jádro.

Moduly jsou podle činnosti, kterou vykonávají, rozděleny do skupin. Typy se mohou lišit v úrovni povoleného přístupu k jádru a především v míře a způsobu integrace do chodu hlaví aplikace. Typy modulů jsou:

**Obecný** – Module
: Modul nezávislí na jádře. Jediné spojení s jádrem je pomocí událostí.

**Modul scény** – SceneModule
: Má přímý přístup ke správci scén, žádná jiná část jádra není přístupná.

**Modul pro import a export** – IOModule
: Vychází z modulu scény. Na straně jádra vyžaduje spolupráci při výběru správného modulu pro požadovaný formát dat.

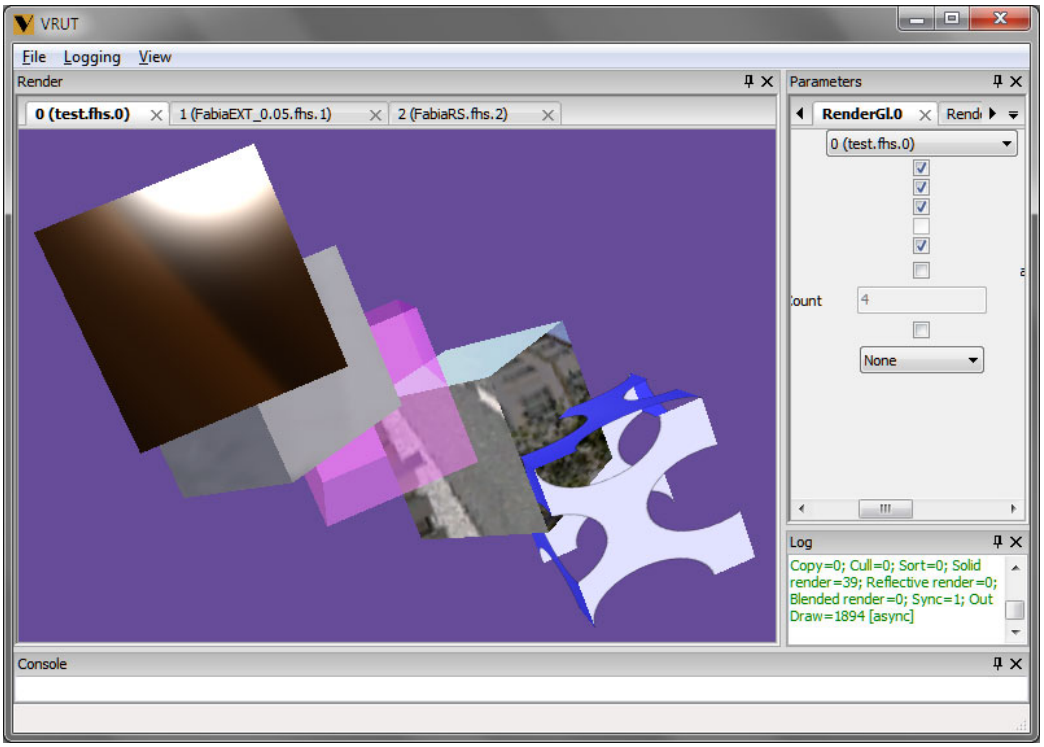
**Zobrazovací modul** – RenderModule
: Rozšiřuje modul scény. Vyžaduje vysokou míru propojenosti ze strany jádra, naopak ze strany modulu k žádnému rozšíření přístupu k jádru nedochází.

**Manipulátor** – ManipulatorModule
: Zpracovává speciální události ze vstupních zařízení.

**Manipulátor kamery** – CameraModule
: Na rozdíl od manipulátoru je více specializován na ovládání kamery.

## 2.7 Geometrie

Geometrie je součástí scény. Odkazována může být pouze uzlem s geometrií a není nijak závislá na materiálu, takže jedna geometrie muže být použita více uzly a v každém z nich lze požít jiný materiál. Na obrázku 2.2 je model krychle definovaný jednou geometrií ve scéně odkazován pětkrát, liší se však v použitém materiálu.




Obrázek 2.2: Pro jednu geometrii je použito různých materiálů

Geometrie využívá dědičnost pro možnost definice více typů geometrií, všechny typy mají svou obálku AABB a umožňují převod geometrických dat do podoby seznamu trojúhelníku.

Zatím jediným implementovaným typem geometrie je trojúhelníková representace, která je definována seznamem vrcholů a indexů do pole vrcholů, případně pak normál a souřadnic pro textury. Indexovanou trojúhelníkovou geometrii lze interpretovat jako TRI_LIST, TRI_FAN, TRI_STRIP, POLYGON, LINES, LINE_STRIP, POINTS nebo QUADS. Geometrie umožňuje libovolnou kombinaci podporované interpretace primitiv v rámci jedné geometrie.

## 2.8 Vývoj

Systém VRUT se stále ještě vyvíjí. V současné době obsahuje velké množství modulů rozšiřujících jeho možnosti. Stále však existuje funkčnost, která se ve stávajících modulech nenachází, a proto jsou potřeba nové moduly doimplementovat. Jedním z nich je právě modul pro detekci kontur a jejich následný export.

# 3 Detekce obrysů

Obrysy se v počítačové grafice používají jako jedna z metod nefotorealistického zobrazení (NPR) a metody pro jejich detekce lze na nejhrubější úrovni rozdělit na rastrové a vektorové.

## 3.1 Rastrová detekce obrysů

Pro detekci obrysů byla vyvinuta celá řada algoritmů (viz [Jiří Žára(2010)]) a pro představu cituji tři metody uvedené v tomto zdroji.

**NPR s pomocí prahování**

1. Umísti bodový zdroj světla L do polohy kamery.
2. Nastav materiál všech těles na bílý, čistě difúzní.
3. Zobraz scénu osvětlenou pouze pomocí zdroje L.
4. Převeď vykreslený obraz prahováním na černobílý.

**NPR s pomocí přivrácených a odvrácených ploch**

1. Rozděl plochy scény na přivrácené a odvrácené.
2. Vykresli přivrácené plochy bíle.
3. Posuň model blíže k rovině.
4. Vykresli odvrácené plochy černě (pomocí paměti hloubky).

**NPR s vyhledáváním změny hloubky**

1. Vyhodnoť scénu pouze pomocí paměti hloubky.
2. Načti mapu hloubek.
3. Pomocí technik hledání hran nalezni na mapě místa s výraznou změnou hloubky a vykresli je.

Přístup těchto metod je rastrový, a z tohoto důvodu jsem je zavrhl. Implementace by znamenala rasterizaci modelu, který je popsán trojúhelníkovou sítí, a jeho následnou vektorizaci, aby bylo možné výsledek exportovat jako vektorový obrázek.

## 3.2 Vektorová detekce obrysů

Vektorový přístup detekce obrysů spočívá v omezení množiny všech hran modelu pouze na ty, které splňují podmínky obrysových hran. Nejdříve je nutné z trojúhelníkové representace vytvořit seznam hran, ke kterým se uloží též informace o prvcích, které s ní sousedí (tzv. okřídlená hrana). Následně mohou být hrany rozděleny do tří skupin:

**Ostré (skutečné) hrany** – tvoří hranici ploch. S touto hranou sousedí pouze jeden trojúhelník, nebo dva trojúhelníky, které však na této hraně mají odlišné normálové vektory. Tyto hrany se nemusí při změně kamery přepočítávat a jsou vždy označeny jako obrysové.

**Potencionální obrysové hrany** – hrana, která by mohla být obrysovou hranou. Obrysová hrana vznikne mezi přivráceným a odvráceným trojúhelníkem. U této skupiny hran se musí při každé změně kamery ověřit, zda je či není hranou obrysovou.

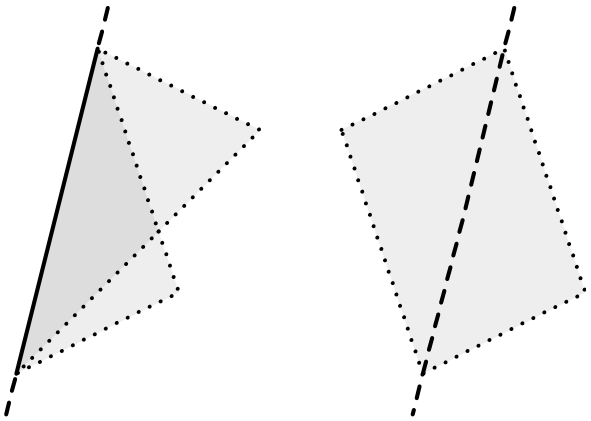
**Pomocné hrany** – hrana mezi trojúhelníky, které leží přibližně v jedné rovině. V této skupině hran se obrysové hrany nenachází.

Po tomto rozdělení musí být ve skupině potencionálních obrysových hran nalezeny ty, se kterými sousedí jeden přivrácený a jeden odvrácený trojúhelník. Určení toho, zda je trojúhelník přivrácený či odvrácený však je, v případě, že není dodrženo pořadí vrcholů, nemožné.

Pokud však převedeme vrcholy pomocí zobrazovací matice do 2D, můžeme obrysovou hranu určit na základě toho, zda třetí vrcholy (ty, které nejsou součástí hrany) přilehlých trojúhelníků leží či neleží ve stejné polorovině určené dvěma vrcholy hrany - pokud ve stejné polorovině leží, je tato hrana obrysová (Obrázek 3.1).

## 3.3 Viditelnost obrysových hran

U rastrových metod je viditelnost obrysových hran řešena pomocí Z-bufferu. U těch vektorových je však problém složitější a musí se použít některý z liniových algoritmů viditelnosti. Jako příklad zde uvedu tři algoritmy sloužící k určení viditelnosti hran (viz [Jiří Žára(2010)]).



Obrázek 3.1: Určení, zda se jedná o obrysovou hranu

### 3.3.1 Robertsův algoritmus

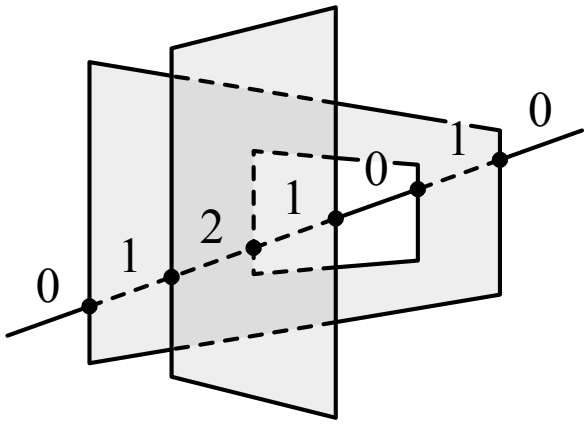
Robertsův algoritmus je určen pro řešení hranové viditelnosti ve scéně složené z konvexních ploch a těles. Postupně pro všechny hrany testuje, zda jsou zakryty jinou plochou nebo tělesem. Pokud je testované hrana zakryta jen částečně, rozdělí se hrana na zakrytou a nezakrytou část a každá nezakrytá část se stává novou hranou, kterou je třeba otestovat.

Robertsův algoritmus se stal základem mnoha dalších metod, které se liší například seřazením hran, použitím urychlovacích testů pří hledání zákrytů, apod. Většina úprav přitom zachovává hlavní myšlenku, kterou je rozdělení potencionálně viditelných hran na elementární úseky, na nichž se již viditelnost nemůže měnit. K nalezení těchto s neměnnou viditelností se používá obrysových hran, neboť pouze ty indikují případnou změnu viditelnosti.

Velkou nevýhodou tohoto algoritmu je, že zkoumání zákrytu potenciálně zakrytých elementárních úseků s nekonvexními mnohostěny je poměrně zdlouhavé.

### 3.3.2 Appelův algoritmus

Appelův algoritmus je založen na testování potencionálně viditelných hran vůči hranám obrysovým a hledají se jejich zdánlivé průsečíky (v průmětu). Pokud testovaná hrana v tomto průsečíku leží za obrysovou hranou, mění se




Obrázek 3.2: Změna koeficient zakrytí v průsečících hran

v tomto místě koeficient zakrytí – pokud za ní vstupuje, koeficient se zvyšuje, pokud vystupuje, tak se snižuje (Obrázek 3.2). Úseky, ve kterých je koeficient zakrytí roven 0, jsou viditelné.

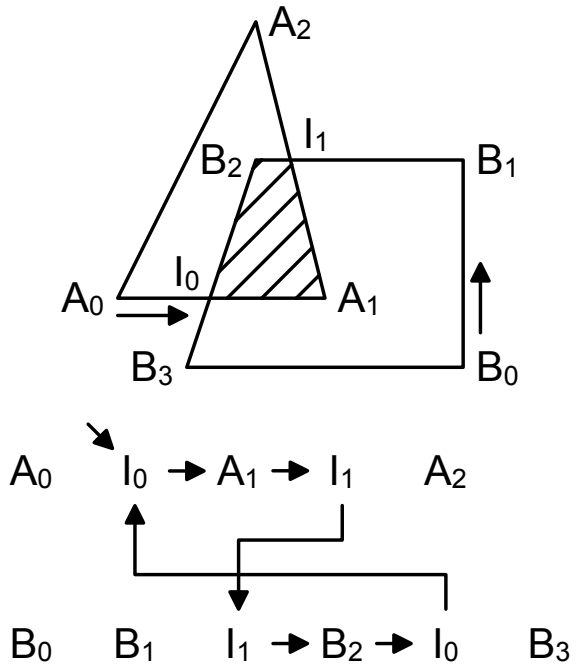
Nejprve se musí spočítat koeficient zakrytí v počátečním vrcholu, a to určením počtu polygonů, které ho zakrývají. Tyto hodnoty se pak dále nesou po hranách vycházejících z tohoto vrcholu a koeficient zakrytí se mění, jen pokud hrana protíná hranu obrysovou a vstupuje za ní.

Poměrně jednoduchý postup tohoto algoritmu je komplikován nutností ošetřování mnoha možných singulárních případům kdy např. průmět vrcholu obrysové hrany leží v průmětu testované hrany a naopak.

### 3.3.3 Weiler-Athertonův algoritmus

Weiler-Athertonův algoritmus je založen na rychlém ořezávání nekonvexních ploch vybranou blízkou plochou. Nejdříve jsou všechny plochy seřazeny podle jejich minimální vzdálenosti od pozorovatele a následně je vybrána prvního, podle které jsou rozstříhány všechny zbývající plochy. Ty jsou roztříděny do dvou dalších seznamů v závislosti na tom, zda daná část leží uvnitř nebo vně nadřazené plochy. Tento postup se dále opakuje i pro další plochy v seznamu.

Způsob ořezávání je naznačen na obrázku 3.3. Základem je nalezení všech průsečíků mezi dvěma množinami úseček v rovině. První je tvořena úsečkami mezi vrcholy $A_0 - A_1 - A_2 - A_0$, druhá mezi vrcholy $B_0 - B_1 - B_2 - B_3 - B_0$.




Obrázek 3.3: Způsob ořezávání ploch

Průsečíky obou ploch jsou body $I_0$, $I_1$, $I_3$. Algoritmus pracuje se dvěma seznamy – vrcholy první a druhé plochy. Průsečíky se zařadí mezi vrcholy první a druhé plochy. V prvního seznamu najdeme vstupní průsečík a postupujeme přes vrcholy vpřed, dokud nenarazíme na další z průsečíků. V tomto místě přejdeme do druhého seznamu na odpovídající místo, a zase postupujeme přes vrcholy vpřed až k dalšímu průsečíku. Zde se zase přesuneme do prvního seznamu a postup opakujeme tak dlouho, dokud nenarazíme na první průsečík. Všechny prošlé vrcholy a průsečíky jsou vrcholy ořezané plochy.

Tento algoritmus je příkladem postupu, řešícího viditelnost ploch. Plochy nebo jejich ořezané části, které byly označeny jako viditelné, lze dále vybarvit.

# 4 PostScript

Jedním z cílů této práce bylo umožnit export nalezených hran do formátu PostScript, a proto se v této kapitole seznámíme s tímto formátem. Použité informace jsou čerpány z [Ado(1999)], [Ado(1985)] a [Tišnovský(2006)].

## 4.1 Základní charakteristiky

PostScript, který byl vyvinut firmou Adobe v roce 1985, je programovací jazyk, soužící k popisu stránky, která se má vykreslit na tiskárně, obrazovce počítače, či jiném zobrazovacím zařízení.

K zobrazení na monitoru je zapotřebí vhodný prohlížeč, například volně šiřitelný program GhostView spolu s Ghostscriptem. Tisk může probíhat přímo na postscriptové tiskárně.

Mezi možnosti popisu stránky patří operátory pro kreslení přímek, oblouků, obdélníků a kubických křivek, nastavení vzhledu obrysových čar a výplně, nebo nastavení ořezové cesty. Barvy mohou být nastaveny více způsoby – stupně šedi, RGB, CMYK a CIE, a též je možné místo barvy použít vzory nebo stínování. Text je také považován za grafické tvary, takže s ním lze pracovat pomocí grafických operátorů. Rastrová grafika je přímo vložena do popisu stránky a existuje řada způsobů, jak reprodukovat obrázky na výstupním zařízení. PostScript též podporuje všechny kombinace lineárních transformací, jako jsou posunu, změny velikosti, otočení, zrcadlení a zkosení. Transformace se aplikují na všechny prvky stránky, včetně textu, grafických tvarů a rastrové grafiky.

PostScript používá k popsání grafické informace programovací jazyk, jehož instrukce se zaznamenávají v postfixové notaci. To znamená, že nejdříve musí být zadány operandy a potom teprve operátor. Interpretace se provádí pomocí zasobníku.

## 4.2 Syntaxe

Poststriptový soubor je soubor textový, proto leze k jeho vytvoření použít textový editor. Jednotlivé syntaktické konstrukce jsou odděleny „bílými“ znaky, které jsou `NUL`, `TAB`, `LF`, `FF`, `CR` a `SP`. Je tedy jedno, zda se jednotlivé hodnoty či operátory píší na nové řádky, nebo se k jejich oddělení použije jiný „bílý“ znak, například mezera.

Dalšími speciálními znaky jsou `(`, `)`, `<`, `>`, `[`, `]`, `{`, `}`, `/`, a `%`, které slouží pro oddělení textů, těl procedur, jmen objektů a komentářů. Komentáře začínají znakem `%` a končí znakem nové řádky.

Čísla lze zadávat následujícími způsoby:

- Celá čísla, například:
  `123 -98 43445 0 +17`
- Reálná čísla, například:
  `-.002 34.5 -3.62 123.6e10 1.0E-5 1E6 -1. 0.0`
- Čísla při určitém základu, například:
  `8#1777 16#FFFE 2#1000`

Textové řetězce se zadávají takto:

- Doslovný text je uzavřen v `(` a `)`
- Hexadecimální data jsou uzavřena v `<` a `>`
- ASCII base-85 data jsou uzavřena v `<~` a `~>`

Názvy objektů se zavádějí pomocí znaku `/`, za kterým následuje zvolené jméno. Lomítko není součástí jména.

Prvky pole se zadávají mezi znaky `[` a `]`. Konstrukce pole tedy může vypadat takto:

```
[ 123 /abc (xyz) ]
```

Těla procedur se zadávají pomocí spustitelného pole. Jeho prvky se zadávají mezi znaky `{` a `}`, například takto:

```
{add 2 div}
```

## 4.3 Základní přehled operátorů

### Matematické operátory

`add` – součet hodnot

`div` – podíl hodnot

`mod` – zbytek po dělení hodnot

`mul` – vynásobení hodnot

`sub` – rozdíl hodnot

`sqrt` – druhá odmocnina hodnoty

### Operátory pro práci se zásobníkem

`pop` – zahodí vrchní hodnotu

`dup` – duplikuje vrchní hodnotu

`clear` – zahodí celý zásobní

### Operátory pro nastavení souřadného systému

`trnaslate` – posun

`scale` – měřítko

`rotate` – natočení

### Operátory pro kreslení cesty

`newpath` – Inicializace nové cesty

`moveto, rmoveto` – přesunutí aktuálních souřadnic

`lineto, rlineto` – připojení úsečky

`arc, arcn, arct, arcto` – připojení oblouku

`cuketo, rcurveto` – připojení Beziérovy kubiky

`closepath` – uzavře cestu

### Grafické operátory

`setlinewidth` – nastavení šířku čáry

`setrgbcolor` – nastavení barevný prostor na RGB a nastaví jednotlivé barevné složky

`stroke` – vykreslení čáry po aktuální cestě

**Výstupní oprátory**

`showpage` – vykreslí a smaže aktuální stránku

## 4.4 Definice proměnných a nových příkazů

Pro definici proměnných a nových příkazů se používá příkaz `def`, který ke zvolenému jménu přiřadí hodnotu. Pokud je touto hodnotou seznam příkazů, stává se takovéto jméno novým příkazem. Seznam příkazů se zapisuje do složených závorek. Pro vykreslení přímky mezi zadanými body lze například definovat příkaz:

```
/linefromto {moveto lineto} def
```

Tento příkaz lze pak použít následujícím způsobem:

```
x1 y1 x2 y2 linefromto
```

## 4.5 Souřadný systém

Pro natočení, zkosení, posun, změnu velikosti či zrcadlení používá PostScript lineární transformace. Pro zadávání souřadnic, kterými se řídí vykreslování, se používá souřadný systém nezávislý na zobrazovacím zařízení. Základní délkovou jednotkou je jeden topologický bod, jehož délka se rovná 1/72 palce. Proto se těsně před rastrováním provádí převod těchto souřadnic do souřadného systému daného zobrazovacího zařízení.

Při inicializaci nové stránky je nastaven počátek souřadného systému do levého dolního rohu. Ne vždy takovéto nastavení vyhovuje, je však možné nastavit transformační matici a takto definovat uživatelský prostor.

Nastavení transformační matice pro počátek uživatelského prostoru na střed stránky a délkových jednotek na milimetr vypadá například takto:

```
595 2 div 842 2 div translate
72 25.4 div dup scale
```

# 5 Návrh

Při návrhu řešení jsem se nejdříve ocitl před otázkou, zda použít rastrovou nebo vektorovou detekci obrysů. Rozhodl jsem se pro vektorovou, a to proto, že výsledkem exportu obrysů má byt vektorový obrázek. V případě požití některé z rastrové metody by muselo nejdříve dojít k rasterizaci – model je popsán trojúhelníkovou sítí, a poté k převedení zpátky na vektory, které by se použily pro export.

Bylo tedy nutné vyřešit 6 základních bodů, a to:

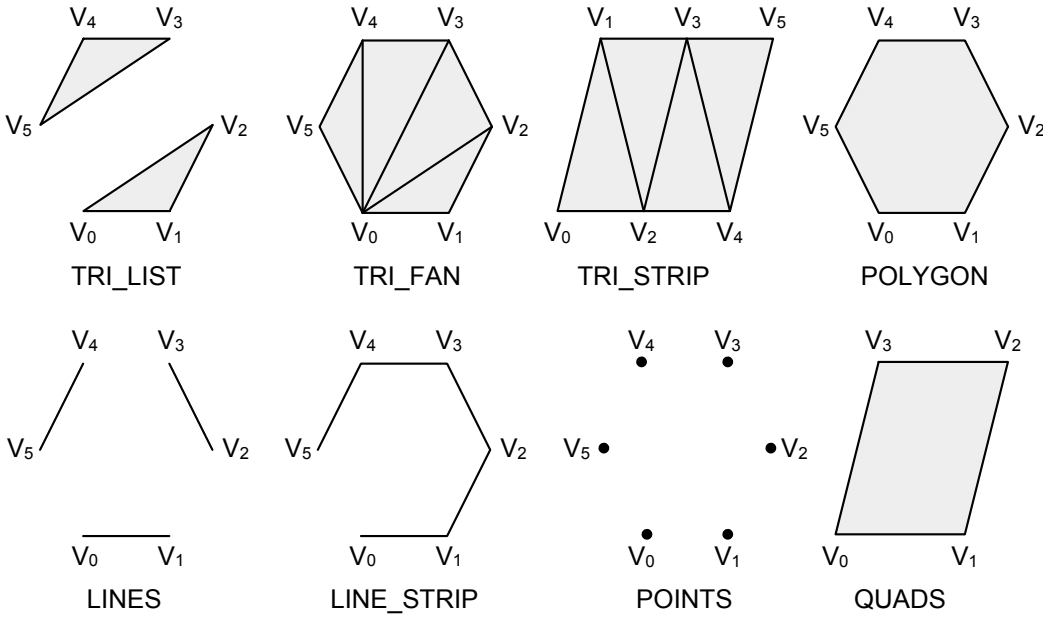
- Nalezení všech hran modelu
- Rozdělení seznamu na skutečné a potencionálně obrysové hrany
- Nalezení skutečně obrysových hran
- Sestavení lomených čar z obrysových hran
- Viditelnost obrysových hran
- Export obrysů do souboru

Při práci s reálnými daty se však vyskytl ještě jeden, a to vyřešit vznik falešných hran způsobený vícenásobným výskytem jednoho bodu v popisu geometrie.

## 5.1 Nalezení všech hran

Indexovou geometrii VRUTu lze interpretovat jako TRI_LIST, TRI_FAN, TRI_STRIP, POLYGON, LINES, LINE_STRIP, POINTS, QUADS. Proto jsem se rozhodl každou z těchto interpretací zpracovat samostatně a při postupném procházení indexového pole zařazovat hrany do seznamu na základě schémat na obrázku 5.1.

U interpretací LINES a LINE_STRIP mohou vznikat rovnou skutečné hrany, POINTS se může ze vkládání hran vynechat.




Obrázek 5.1: Přehled interpretací geometrie VRUTu

## 5.2 Nalezení skutečných hran

Nalezení skutečných hran je založeno na jednoduchém principu rozdělení seznamu všech hran do dvou na základě toho, zda je součástí jednoho nebo dvou trojúhelníků. Pokud je hrana součástí jednoho trojúhelníku, nachází se tak na okraji plochy a je to tedy skutečná hrana, která se bude vykreslovat vždy a nezávisle na parametrech kamery. Do seznamu skutečných hran jsou též zařazeny všechny hrany, které jsou součástí více než dvou trojúhelníků. Pokud je sdílena dvěma trojúhelníky, pak je třeba provést další testy, aby se rozhodla o jejím vykreslení, a je jí tedy potřeba zařadit do seznamu potencionálně obrysových hran, jejichž zobrazení je závislé na parametrech kamery.

## 5.3 Nalezení obrysových hran

Po rozdělení seznamu všech hran na skutečné a potencionálně obrysové jsou hrany z prvně jmenovaného z nich zobrazovány vždy. Problém nastává v případě druhého seznamu, ve kterém se musí hrany, které se budou zobrazovat,

nalézt v závislosti na parametrech kamery.

Všechny body se nejdříve pomocí pohledové matice převedou do souřadnic, které jsou použity při zobrazení modelu, a poté se pro každou hranu ze seznamu potencionálně obrysových ověřuje, zda její třetí a čtvrtý vrchol leží ve stejné polorovině definované přímkou určenou jejími prvními dvěma vrcholy (Obrázek 3.1). Pokud ve stejné polorovině leží, je nastaven příznak, který určí, že při vykreslení bude tato hrana zobrazena.

## 5.4 Sestavení lomených čar

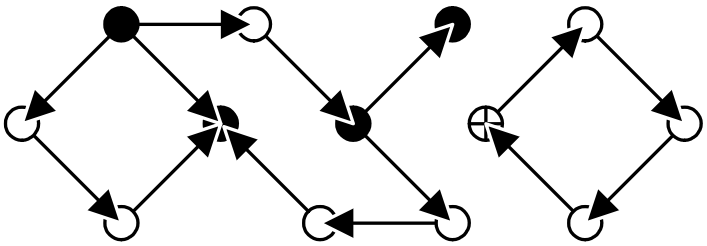
K sestavení lomených čar z jednotlivých hran jsem se rozhodl proto, aby se následně lépe řešila viditelnost hran, i z hlediska velikosti výsledného exportovaného souboru (nemusí se neustále přesouvat pozice pera).

Nejdříve si ke každému vrcholu vytvořím seznam hran, které jsou k němu navázány. Poté se postupně procházejí vrcholy, ze kterých vede jedna nebo tři a více hran.

Tyto vrcholy jsou určeny jako počátek lomené čáry a dále se postupuje ve směru ještě nepoužité hrany (prošlé hrany se označí) tak dlouho, dokud je počet hran navázaných k vrcholu roven dvěma. Takto se postupně naleznou všechny acyklické lomené čáry.

Pokud zbudou některé vrcholy se dvěma navázanými neoznačenými hranami, jsou určeny jako počátek lomené čáry a dále se postupuje ve směru ještě nepoužitý hrany tak dlouho, dokud není dosaženo počátku lomené čáry. Tímto způsobem jsou nalezeny všechny cyklické lomené čáry.

Tento mechanizmus je naznačen na obrázku 5.2. Vrcholy, ze kterých vychází jedna nebo tři a více hran, jsou znázorněny černými puntíky, a ty, ze kterých vycházejí pouze hrany dvě, jsou označeny puntíky bílými. Úsečky mezi nimi znázorňují nalezené obrysové hrany a šipky ukazují postupné vytváření lomených čar. Puntík s křížkem znázorňuje zbylý vrchol, ze kterého vychází pouze dvě hrany, který je použit jako začátek další lomené čáry.



Obrázek 5.2: Sestavení lomených čar

## 5.5 Viditelnost obrysových hran

Protože jsem se rozhodl pro vektorovou detekci obrysů, pro řešení viditelností hran připadal v úvahu Robertsův, Appelův nebo Weiler-Athertonův algoritmus.

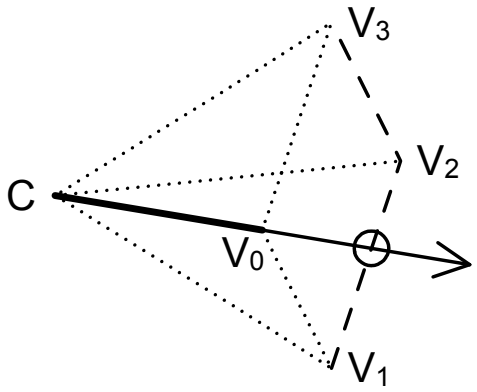
Robertsův má tu nevýhodu, že zkoumání zákrytu potencionálně zakrytých úseků hran s nekonvexními mnohostěny je zdlouhavé. Model, který jsem měl k dispozici, byl složen z více než miliónu trojúhelníků, a proto sem se tento algoritmus rozhodl zavrhnout, aby bylo možno obrysy zobrazovat v reálném čase. Weiler-Athertonův algoritmus [Kevin Weiler(1977)] řeší viditelnost ploch. Tyto plochy by nejdříve bylo nutné ze seznamu lomených čar sestrojit, a až potom tento algoritmus aplikovat. Rozhodl jsem se proto využít Appelův algoritmus [Appel(1967)], pro který popis obrysových hran stačí, neboť řeší viditelnost na základě průsečíků těchto hran.

Nejprve bylo nutné vypočíst koeficienty zakrytí v rozích geometrie. Pro tento výpočet se použije test průsečíku polopřímky začínající v daném rohu a procházející druhým bodem testované hrany s hranou všech trojúhelníků sousedících s daným rohem, které však rohem ani druhým bodem testované hrany neprocházejí. Pokud je v tomto průsečíku testovaná hrana za hranou trojúhelníku, vychází z rohu za tímto trojúhelníkem schována, a proto se její koeficient zakrytí zvýší o jedna.

Pro objasnění výpočtu koeficientů v rozích nám poslouží obrázek 5.3. Testovaný roh je zde označen písmenem C, a sousedí s ním čtyři trojúhelníky $CV_0V_1$, $CV_1V_2$, $CV_2V_3$, $CV_3V_0$. Hrana, pro kterou se provádí výpočet koeficientu zakrytí, je $CV_0$, a hrany všech čtyř trojúhelníků, které splňují podmínku, že neprocházejí rohovým bodem $C$ ani druhým bodem testované hrany $V_0$,

jsou pouze dvě – $V_1V_2$ a $V_2V_3$.

Průsečík polopřímky $CV_0$ a těchto dvou hran je pouze jeden, na obrázku označený kroužkem. Pokud se v tomto průsečíku polopřímka $CV_0$ nachází za úsečkou $V_1V_2$, je schována za trojúhelníkem $CV_1V_2$ a nutné zvýšit koeficient zakrytí této hrany o jedna, pokud se nachází před ní, tento koeficient se nemění.

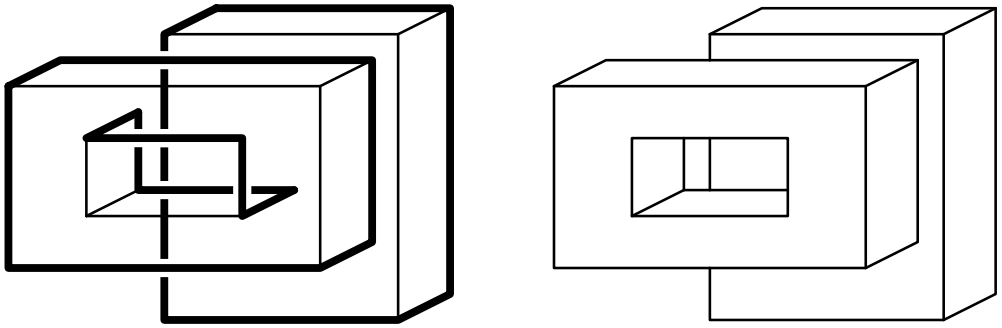



Obrázek 5.3: Výpočet koeficientu zakrytí v rozích geometrie

Dalším krokem by mělo být rozsekání sestavených lomených čar na elementární úseky, na nichž se viditelnost nemůže změnit. K nalezení těchto úseků lze s výhodou použít obrysové hrany, vyjma těch, které sdílejí trojúhelníky ležících na obou stranách dané hrany (podmnožina skutečných hran). Na obrázku 5.4 je vidět, jakým způsobem ovlivňují tyto hrany (nakresleny silnější čárou) viditelnost daných úseků.

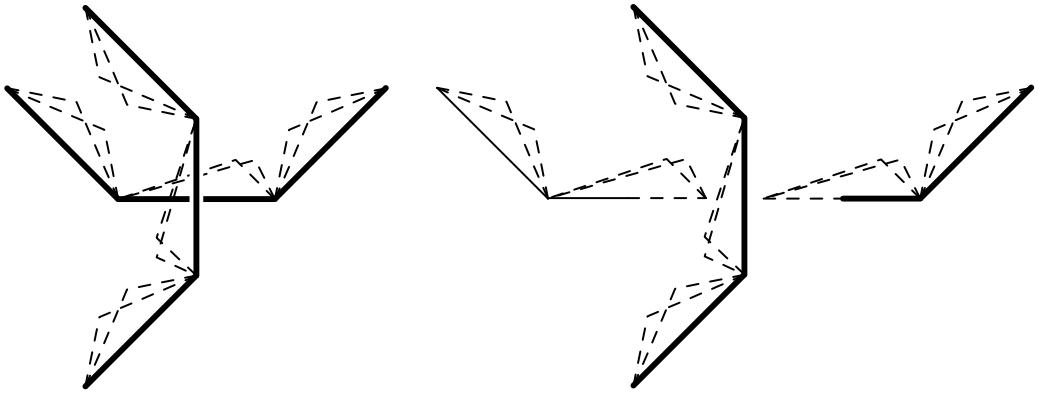
U lomené čáry se též uchovává informace o relativním koeficientu zakrytí na jejím začátku a konci. V bodě rozdělení se nastaví pro zakrytou část tento koeficient na hodnotu, která koresponduje s počtem přilehlých trojúhelníků, pro nezakrytou část se koeficient v bodě rozdělení nastaví na nulu.

Poté, co jsou všechny čáry rozděleny na úseky, ve kterých se již viditelnost změnit nemůže, a jsou nastaveny jejich relativní koeficienty zakrytí vůči ostatním navazujícím čarám, určí se lomená čára, jejíž součástí je bod s nejmenší hloubkou. Tato čára je vždy viditelná, její absolutní koeficient zakrytí se nastaví na nulu, a pomocí procházení grafem, jehož hranami jsou lomené čáry, se vypočítají absolutní koeficienty zakrytí všech lomených čar. Čára je viditelná, pokud její koeficient zakrytí je roven nule.



Obrázek 5.4: Rozdělení hran na elementární úseky s neměnnou viditelností

Rozdělení lomené čáry je nutné pouze v případě, že tato vstupuje za jinou, a v místě rozdělení se zákonitě mění koeficient zakrytí. U lomené čary se také ukládá informace o přilehlých trojúhelnících, díky které je možné rozpoznat, která část rozdělené čáry se nachází za plochou ohraničenou dělicí čárou a která mimo ni. Jak s touto informací pracovat objasňuje obrázek 5.5. Na rozděleném úseku lomené čáry se stejná informace o přilehlých trojúhelnících uchovává pro obě nové části.



Obrázek 5.5: U lomené čary se také ukládá informace o přilehlých trojúhelnících

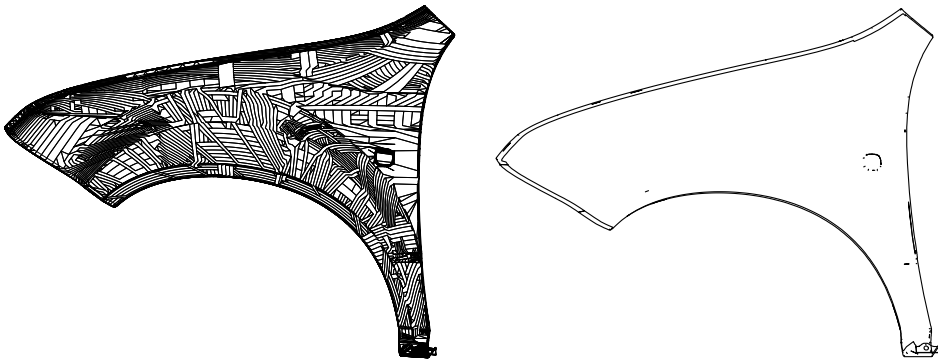
## 5.6 Export obrysů

Pro export obrysů do souboru ve formátu PostScript jsem se rozhodl využít sestavené lomené čáry. Nejdříve se přesunou souřadnice na její začátek, a poté se k cestě připojí úsečka končící v dalším jejím vrcholu. Takto se k cestě postupně připojí všechny úsečky, které tvoří lomenou čáru a za poslední z nich se cesta uzavře. Takto budou do souboru vyexportovány všechny lomené čáry tvořící obrys modelu.

Exportované hrany se nebudou nikterak ořezávat podle velikosti zobrazovaného okna, ale pro toto ořezání se použijí přímo příkazy jazyka PostScript.

## 5.7 Vznik falešných hran

Důvodem vzniku falešných hran na některých modelech je to, že jeden bod je v modelu uložen několikrát, a definice geometrie se díky tomu na něj odkazuje pomocí různých indexů. Pokud tedy seznam hran vznikal jen za pomoci indexů, a nebral v úvahu skutečnou polohu bodu a normálový vektor plochy v tomto bodě, výsledné obrysové hrany byly protkány velkým množstvím falešných hran (Obrázek 5.6).



Obrázek 5.6: Model s falešnými hranami a ten samý po jejich odstranění

K odstranění těchto falešných hran jsem se rozhodl použít hašovací tabulku, popsanou v práci [Ska(2009)]. Vedla mě k tomu její rychlost při ověřování duplicity bodů, která se odvíjí od nízké délky klastrů tabulky. Je

založena na hašovací funkci

$$Index = (int)((\alpha X + \beta Y + \gamma Z)C + 0.5)\&T$$

kde $(int)$ je konverze na datový typ `unsigned integer`, $X$, $Y$, $Z$ jsou souřadnice bodu, $\alpha$, $\beta$, $\gamma$ jsou koeficienty hašovací funkce, $C$ je koeficient, kterým se zajistí, aby se využilo plného rozsahu datového typu `unsigned integer`, $T + 1$ je velikost tabulky ($T = 2^k - 1$), & representuje modulo, které je implementováno jako logický operátor, a 0.5 je konstanta, která byla zjištěna experimentálně a pomáhá lepšímu rozložení souřadnic v tabulce.

Velice důležitá, pro velký rozptyl hašovací funkce, je volba koeficientů, jejichž hodnoty pro dosažení vynikajících výsledků jsou $\alpha = \pi$, $\beta = e$, $\gamma = \sqrt{2}$. Dalším trikem této funkce je nahrazení klasického modulo logickým operátorem &, které výpočet urychlí. Toto nahrazení si můžeme dovolit díky tomu, že velikost tabulky je vždy $2^k$.

# 6 Provedení

Programovacím jazykem pro celý projekt VRUT je C++, a proto byl použit i pro tento modul.

Vytvořený modul se nachází v adresáři `modules/drawingscene` a jeho součástí je několik tříd, které budou později podrobně popsány. Je vytvořen jako modul scény – to znamená, že má přímí přístup ke správci scén a žádná jiná část jádra není přístupná.

Modul se spouští z konzole VRUTu příkazem `runmodule DrawingScene`. Poté lze pomocí příkazu `setparam DrawingScene.sceneID [sceneID]` nastavit scénu, pro kterou bude tento modul využit a nastavením parametru `setparam DrawingScene.enabled 1` přepnout modul do módu zobrazování obrysových hran.

Po spuštění se modul zaregistruje k odběru o změnách transformací scény, geometrie a parametrů kamery. Dále jsou po spuštění modulu nebo po změně scény postupně načteny všechny geometrie z dané scény a je z nich vyexportován seznam hran a informace o sousedních plochách. Poté se hrany roztřídí no ostré (skutečné), které se nemusí při změně kamery přepočítat, a na potencionální obrysové hrany, ve kterém se to opravdu obrysové hledají při každé změně parametrů kamery.

Zvolená scéna se modifikuje tak, že se původní geometrie skryje a přidá se nová, která se bude plnit při každé změně parametrů kamery nalezenými hranami. Tak je možná zapnout klasický způsob zobrazení jen tím, že se nová geometrie pro nalezené hrany skryje a původní se naopak zviditelní.

Při ukončení modulu se uvolní všechna alokovaná paměť vytvořených objektů a modul se zavře.

## 6.1 Uživatelské rozhraní modulu

Na uživatelském rozhraní modulu se nacházejí tři ovládací prvky, pole pro zadání jména souboru a tlačítko, pomocí kterého se provede export do zvoleného souboru (Obrázek 6.1).

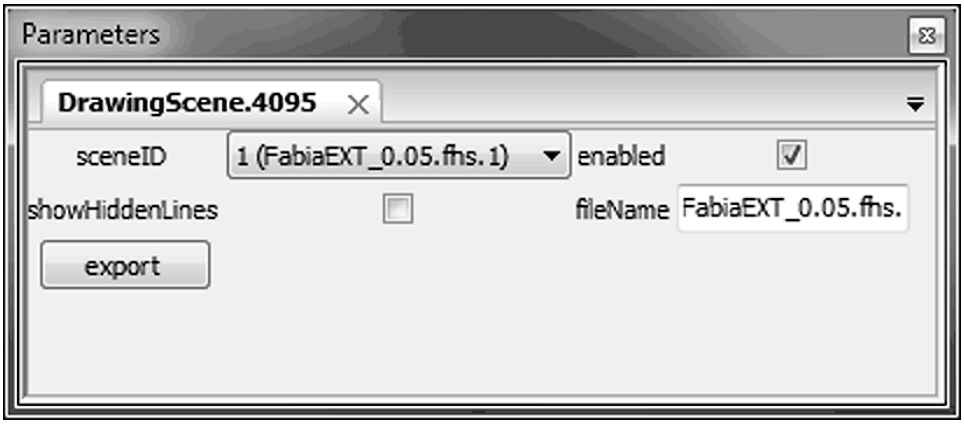
**sceneID** – slouží k výběru scény, pro kterou se bude aplikovat zobrazení obrysových hran.

**enabled** – povolí nebo zakáže zobrazení obrysových hran.

**showHiddenLines** – povolí nebo zakáže zobrazení skrytých obrysových hran.

**fileName** – slouží pro zadání jména souboru, do kterého bude proveden export.

**export** – slouží k exportu obrysových hran vybrané scény do souboru ve formátu PostScript.




Obrázek 6.1: Uživatelské rozhraní modulu

## 6.2 Popis použitých tříd

### 6.2.1 Třída `DrawingScene`

Tato třída dědí z `SceneModule` a je vstupním místem implementovaného modulu. Uchovává informace o natavení modulu, obsluhuje registrované zprávy a obsahuje funkce pro modifikaci scény a nalezení všech geometrií pro jejich následné zpracování. Jeho důležitou součástí je též obsluha GUI.

Důležité funkce třídy:

`DrawingScene` – konstruktor třídy. Jsou zde registrovány ovládací prvky uživatelského rozhraní a odběr zpráv o změně transformací scény, geometrie a parametrů kamery.

`processEvent` – obsluha zpráv zaslaných jádrem.

`processSelectScene` – tato funkce je volána při změně scény, pro kterou se mají obrysové hrany hledat. V původní scéně skryje geometrie s hranami a zobrazí původní geometrie, v nově vybrané scéně skryje původní geometrie, pokud ještě neexistují geometrie pro hrany, jsou vytvořeny, a zajistí sestavení nových seznamů hran.

`processChangeEnable` – zapíná nebo vypíná zobrazení obrysových hran.

`processExport` – tato funkce zabezpečí export vybrané scény do souboru.

`processShowHiddenLines` – zapíná a vypíná zobrazení skrytých hran.

`prepareScene` – funkce slouží k připravení scény pro zobrazení obrysových hran. Jsou zde přidány nové uzly a geometrie, která bude použita pro zobrazení viditelných a skrytých hran.

`setDrawingScene` – přepne vybranou scénu do režimu zobrazení hran a zajistí přepočet hran a jejich zobrazení.

`returnOrigScene` – přepne vybranou scénu do režimu standardního zobrazení.

`extractEdges` – postupně ve scéně projde všechny geometrie a pro každou vytvoří korespondující objekt třídy `DrawingEdges`, který následně naplní hranami z dané geometrie.

`extractGeometries` – postupně projde všechny uzly s geometriemi a pro každou vytvoří korespondující objekt třídy `DrawingGeometry`.

`extractCameras` – vyhledá všechny kamery scény a vybere kameru pro výpočet obrysových hran.

`clearGeometry` – odstraní z geometrie všechny body, indexy a definice.

`fillGeometry` – funkce slouží k naplnění geometrií pro viditelné a neviditelné hrany.

`redraw` – při změně parametrů kamery zajistí tato funkce výpočet nových obrysových hran a jejich následné zobrazení.

`clear` – uvolní modulem alokovanou paměť.

### 6.2.2 Třída `DrawingEdges`

Tato třída uchovává informace o bodech, normálách a hranách jedné geometrie scény. Obsahuje funkce pro vložení jednotlivých typů geometrie, pomocí kterých se přidávají hrany do seznamu. Dalšími funkcemi je rozdělení hran na skutečné a potencionálně obrysové a výpočet zdánlivého průsečíků dvou hran.

Důležité funkce třídy:

`DrawingEdges` – konstruktor třídy. Slouží k vytvoření hashmap potřebných pro vkládání hran.

`insertPolygon` – export všech hran z geometrie typu `POLYGON`.

`insertQuads` – export všech hran z geometrie typu `QUADS`.

`insertTriFun` – export všech hran z geometrie typu `TRI_FAN`.

`insertTriList` – export všech hran z geometrie typu `TRI_LIST`.

`insertTriStrip` – export všech hran z geometrie typu `TRI_STRIP`.

`insertLineStrip` – export všech hran z geometrie typu `LINE_STRIP`.

`insertLines` – export všech hran z geometrie typu `LINES`.

`getEdges` – funkce vrací ukazatel na seznam hran.

`getVertexMap` – funkce vrací pole obsahující seznam všech vrcholů geometrie a seznam vrcholů, které jsou s nimi spojeny hranou.

`cornerEdgeDepth` – určí viditelnost hrany vycházející z daného vrcholu.

`addEdge` – slouží k vložení hrany do seznamu. Vkládaná hrana je popsána třemi nebo čtyřmi indexy (podle toho, zda s ní sousedí jeden nebo dva trojúhelníky). První dva indexy popisují body hrany a další třetí body přilehlých trojúhelníků.

`getVertexIndex` – na vstupu této funkce je index bodu z originální geometrie. Tato funkce dohledá, zda stejný bod již nebyl použit s jiným indexem a pokud ano, vrací již použity index. Tato funkce je důležitá k odstranění vzniku falešných hran.

`insertForTest` – vložení hrany do seznamu, podle kterého bude ověřena viditelnost hrany vycházejícího z vrcholu.

`intersect` – vypočte zdánlivý průsečík dvou hran a rozhodne, která leží v popředí a která v pozadí.

### 6.2.3 Třída `DrawingGeometry`

Objekty této třídy korespondují s uzly s geometrií vybrané scény. Obsahuje funkce pro nalezení obrysových hran a jejich následné poskládání do lomených čar, řešení viditelnosti, a vykreslení hran do zvolené geometrie.

Důležité funkce třídy:

`DrawingGeometry` – konstruktor třídy. Inicializuje třídní proměnné.

`drawLineStrings` – funkce přenese nalezené lomené čary popisující obrysové hrany do geometrií sloužících pro viditelné a neviditelné hrany, čímž zajistí jejich vykreslení.

`constructLineStrings` – funkce zajistí nalezení všech obrysových hran modelu v závislosti na parametrech kamery, jejich poskládání do lomených čar a určení jejich viditelnosti.

`getGeometryName` – vrací jméno uzlu geometrie, pro kterou daný objekt vznikl.

`getVertices` – vrací seznam všech bodů použitých v popisu obrysových hran.

`getLineString` – funkce vrací ukazatel na seznam lomených čar.

`prepareEdges` – rozdělení seznamu hran na skutečné a potencionálně obrysové na základě toho, zda je k hraně přilehlý jeden nebo dva trojúhelníky.

`constructLineString` – sestrojí lomenou čáru z obrysových hran. Lomená čára je spojnice mezi body, ze kterých vede jedna nebo tři a více hran.

`getSecondVertexFromEdge` – funkce vrací druhý bod hrany.

`clearLineString` – maže seznam lomených čar a uvolní jimi alokovanou paměť.

### 6.2.4 Třída `PostScriptTools`

Tato třída obsahuje funkce sloužící k sestavení PostScript–ového souboru ze seznamu hran, jeho zápis na disk, a také funkce pro výpočet správného umístění modelu na stránku.

Důležité funkce třídy:

`PostScriptTools` – konstruktor třídy. Probíhá zde inicializace parametrů stránky.

`fileOpen` – otevře soubor, do kterého bude probíhat zápis.

`fileClose` – zavře otevřený soubor.

`setBound` – funkce slouží pro nastavení mezních hodnot, kterých nabývají lomené čáry. Tato informace je použita pro výpočet měřítka pro vložení pohledu na stánku.

`begin` – zapíše do souboru definice příkazů a nastavení stránky.

`end` – zapíše do souboru příkaz pro zobrazení stránky.

`solidLines` – zapíše do souboru informaci o tom, že následující popis lomených čar se bude týkat viditelných hran.

`hiddenLines` – zapíše do souboru informaci o tom, že následující popis lomených čar se bude týkat skrytých hran.

`writeLineStrings` – zapíše do souboru všechny lomené čáry vyhovující dané podmínce viditelnosti.

`writeGeometryName` – zapíše do souboru poznámku o názvu uzlu geometry, ze které následující popis lomených čar vznikl.

`exportGeometry` – zajistí vypsání poznámky o názvu uzlu geometrie a následný zápis popisu lomených čar do otevřeného souboru.

`write` – zapíše příslušnou informaci do souboru.

`calcScale` – výpočet měřítka podle mezních hodnot lomených čar pro vložení pohledu na stánku.

### 6.2.5 Třída `VertexHashMap`

Tuto třídu bylo potřeba implementovat kvůli vzniku falešných hran na některých modelech. Jedná se o hašovací tabulku sloužící k nalezení shodných bodů geometrie.

Důležité funkce třídy:

`VertexHashMap` – konstruktor třídy. Jako vstupní parametr má počet bodů, pro které je hašovací tabulka určena.

`insert` – funkce pro vložení daného bodu.

`find` – slouží pro zjištění, zda již daný bod existuje, a pokud ano, poskytne jeho index.

`clear` – smazání všech prvků hašovací tabulky a uvolnění alokované paměti.

`hash` – hašovací funkce.

## 6.3 Popis exportovaného souboru

Exportovaný soubor s popisem obrysových hran je ve formátu PostScript. Tento soubor můžeme rozdělit na dvě části – na definice příkazů a na data s popisem exportovaných hran. Definice se nacházejí na začátku souboru a jejich funkce je:

`/ps` **Velikost stránky** – horizontální a vertikální rozměr stránky. V definici je použit příkaz mm, sloužící pro převod mililitrů na topologické body, které se používají jako délková jednotka v PostScriptu.

`/sl` **Styl obrysových hran** – zde je definováno, jaký styl čáry se použije pro vykreslení obrysových hran.

`/hl` **Styl skrytých hran** – je zde definováno, jaký styl čáry se použije pro vykreslení skrytých hran.

`/m, /l, /s, /e` **Příkazy pro tvorbu cesty** – tyto příkazy jsou použity v datové části pro popis exportovaných hran (viz níže).

`/mm` **Převodní příkaz** – převod mililitrů na topologické body, které se používají jako délková jednotka v PostScriptu.

`/rs` **Přepočet stylů čar** – příkaz sloužící po přepočet stylů čar na jednotný formát, aby nebyly závislé na zvoleném měřítku.

`/iv` **Nastavení pohledu** – nastaví pohled s definovaným měřítkem na specifikované souřadnice.

`/cv` **Oříznutí pohledu** – ořízne pohled určený levým horním a pravým spodním rohem.

Za definicemi následuje sada příkazů pro nastavení stránky (zde je volán příkaz `ps` pro nastavení velikosti stránky), nastavení pohledu (do zásobníku je vloženo měřítko a x-ová a y-ová souřadnice, která se bude krýt s bodem [0, 0] exportovaných dat, a poté je zavolán příkaz `iv`) a oříznutí pohledu (do zásobníku se uloží x-ová a y-ová souřadnice levého horního a pravého spodního rohu, poté se zavolá příkaz `cv`).

V případě, že exportujeme i skryté hrany, bude v datové části po sobě následovat popis skrytých hran a poté popis hran viditelných. Každá z těchto částí je uvozena příkazy pro styl čáry a jeho následným přepočtem (pro skryté hrany `hl rs`, pro viditelné hrany `sl rs`).

Po těchto příkazech následuje popis hran ve formě lomených čar. Do zásobníku je vždy uložena x-ová a y-ová souřadnice bodu následována jedním z příkazů:

`/s` **První bod lomené čáry** – pomocí příkazů `newpath` a `moveto` začne cestu a nastaví počáteční souřadnice lomené čáry.

`/l` **Průběžný bod lomené čáry** – vykonáním příkazu `lineto` připojí další úsečku k cestě.

`/e` **Konečný bod lomené čáry** – příkazem `lineto` připojí k cestě poslední úsečku a příkazem `stroke` zajistí vykreslení aktuální cesty.

Na samotném konci souboru se nachází příkaz `showpage`, pomocí kterého se připravená stránka vykreslí.

# 7 Experimenty

## 7.1 Rychlost zobrazení náhledu

Čas, který zabere přepočet nového pohledu, je závislí na počtu hran modelu. V tabulce 7.1 uvádím výsledky mého testování, kterých bylo dosaženo na přenosném počítači s dvoujádrovým procesorem Intel© Core™2 Duo, operační pamětí 4GB, grafickou kartou ATI Mobility Radeon HD 3650 a 64bitovým operačním systémem.

| **Model** | **Počet hran** | **Čas [ms]** |
|---|---:|---:|
| test.fhs | 90 | 1 |
| FabiaRS.fhs | 3 628 | 5 |
| FabiaEXT_0.05.fhs | 1 240 185 | 330 |

Tabulka 7.1: Čas potřebný k přepočtu nového pohledu

Rychlost posouvání a natáčení modelu však tímto časem nejsou nikterak omezeny, jen u modelu `FabiaEXT_0.05.fhs` nedojde ke korektnímu zobrazení obrysových hran ihned po těchto úkonech.

Po té, co jsem s modulem v rámci testování pracoval, si myslím, že tato rychlost pro interaktivní práci s modely postačuje. Pokud by však čas přepočtu nového pohledu přestal dostačovat, popřípadě by se pracovalo s ještě složitějšími modely, existují metody, jak práci modulu urychlit. Nyní přepočet hran při změně pohledu běží pouze v jednom vlákně, a tak by bylo možné dosáhnout vyšší rychlosti rozložením času na více jader procesoru pomocí běhu výpočtů v několika paralelních vláknech.

## 7.2 Kvalita výstupu

Na modelu krychle (Obrázek 7.1) jsou korektně zobrazeny všechny hrany a je zde též korektně vyřešena jejich viditelnost. K řešení viditelnosti hran konvexního tělesa dostačuje pouze výpočet koeficientů zakrytí v rozích objektu.

Také na modelu `test.fhs` (Obrázek 7.2) jsou všechny nalezené hrany v pořádku. Jejich viditelnost je správně vyřešena pouze v rámci jednotlivých

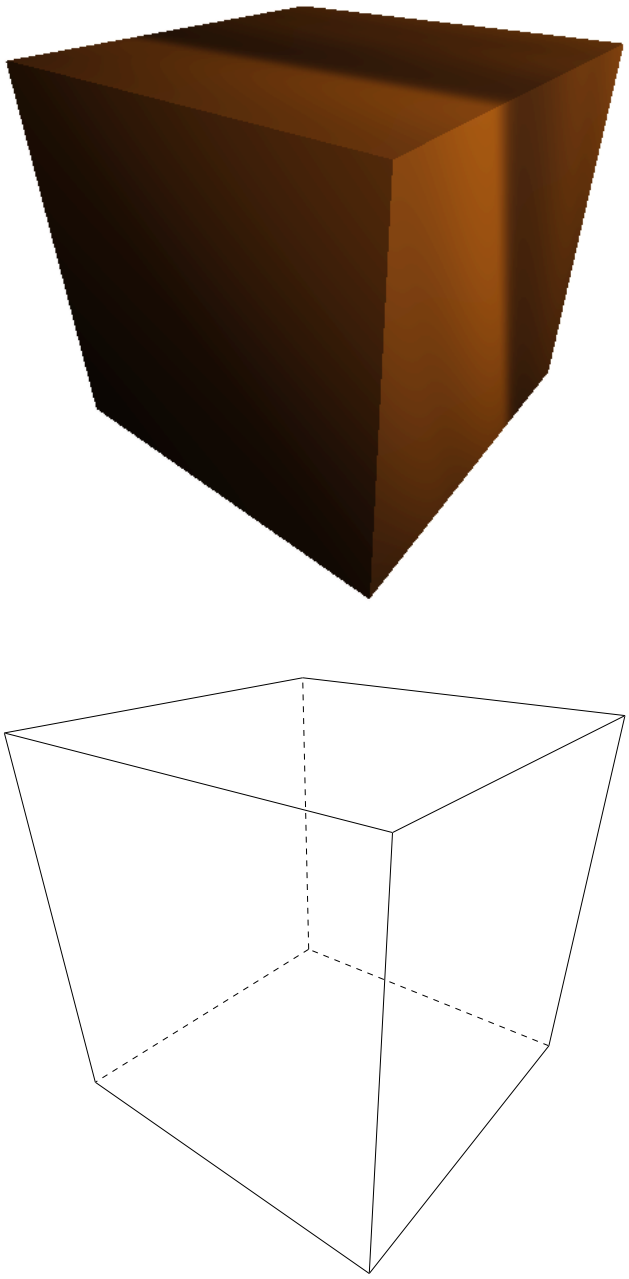
modelů krychle, ale ne v rámci celé sestavy. Nyní je implementována část Appelova algoritmu, která zaručí korektní zobrazení viditelných hran pouze v rámci jednotlivých konvexních objektů. Pro rozšíření na nekonvexní objekty a jejich sestavy by bylo potřeba doimplementovat část algoritmu řešící průsečíky jednotlivých lomených čar a jejich dělení na menší úseky, ve kterých je koeficient zakrytí konstantní (viz sekce 5.5 na straně 22).

Na obrázku 7.3 jsou vidět nalezené obrysové hrany modelu `FabiaRS.fhs`. I zde se zdají být všechny nalezené hrany opravdu těmi obrysovými. Chyby ve viditelnosti se zde však už projevují jak na úrovni jednotlivých geometrií, tak v rámci celé sestavy. I zde je důvodem neúplná implementace algoritmu řešícího viditelnost nalezených hran.

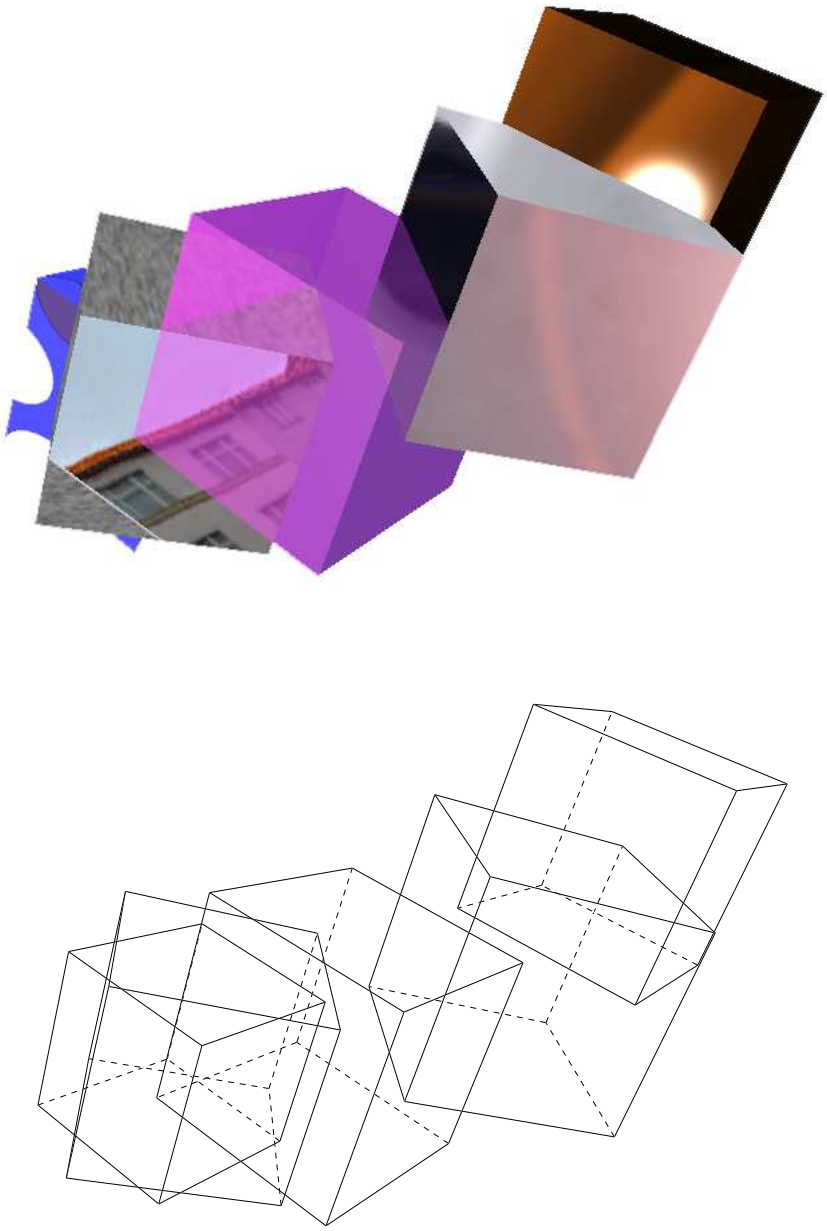
Model `FabiaEXT_0.05.fhs` (Obrázek 7.4) byl nejsložitějším z těch, na kterých byl modul testován. Obrysové hrany se hledaly ve více jak miliónu všech hran tohoto modelu. Problémy s viditelností se zde vyskytují obdobné jako v případě modelu `FabiaRS.fhs`. Přidaly se k nim však i problémy s nalezenými obrysovými hranami, které jsou ukázány na detailu kliky předních dveří (Obrázek 7.5).

Jeden z problémů je viditelný na výlisku v modelu dveří, kde se zlom opticky jeví jako hrana, ale ve skutečnosti je zde malý přechodový rádius, na kterém se v definici geometrie žádná hrana nevyskytuje.

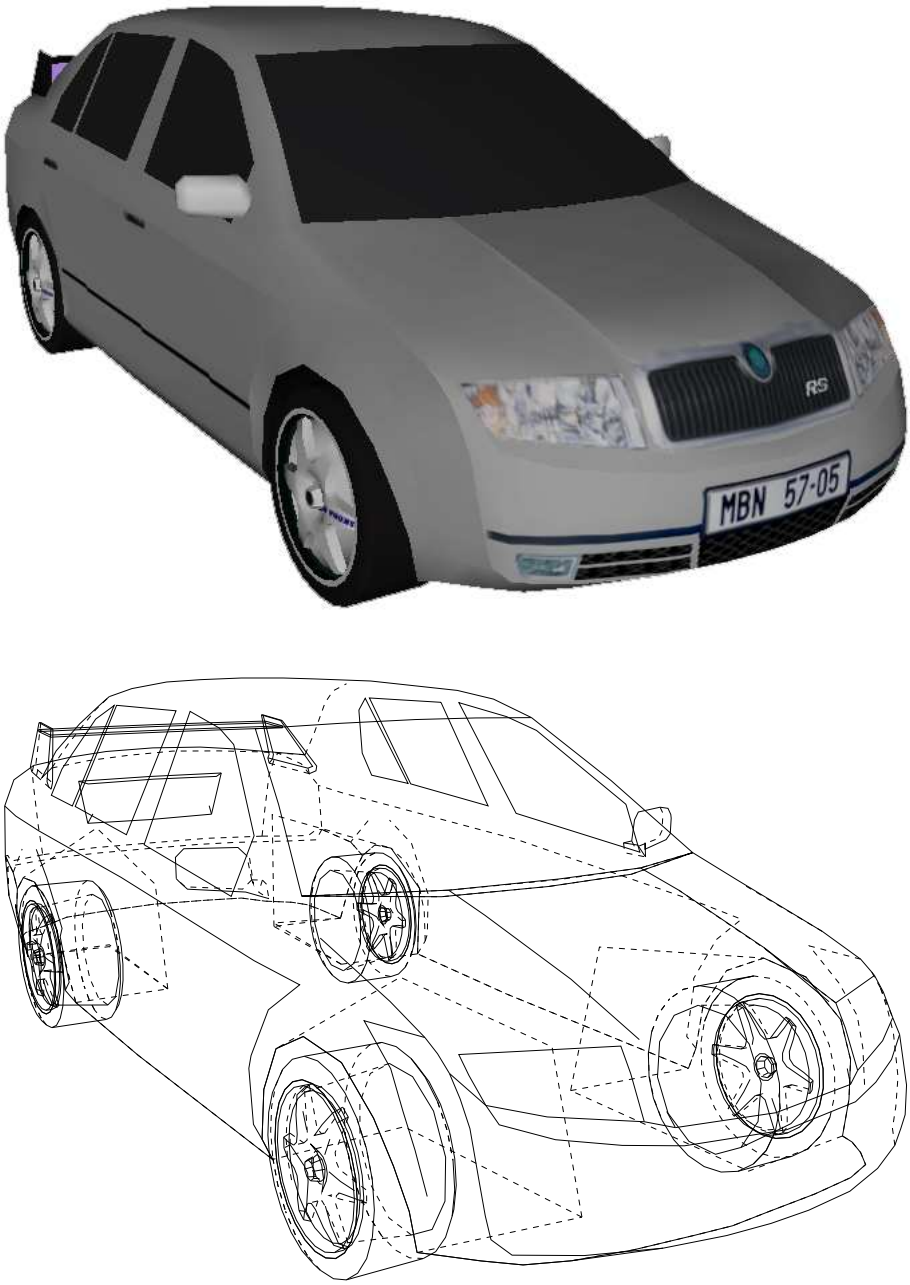
Druhým problémem jsou krátké hrany, které procházejí napříč zaoblenými hranami kliky. V těchto místech opticky žádné hrany nejsou viditelné, ale v definici geometry se zde vyskytují malé odchylky normál. Díky těmto odchylkám není možné sjednotit bod sousedních trojúhelníků, a algoritmus v tomto místě detekuje hranu.



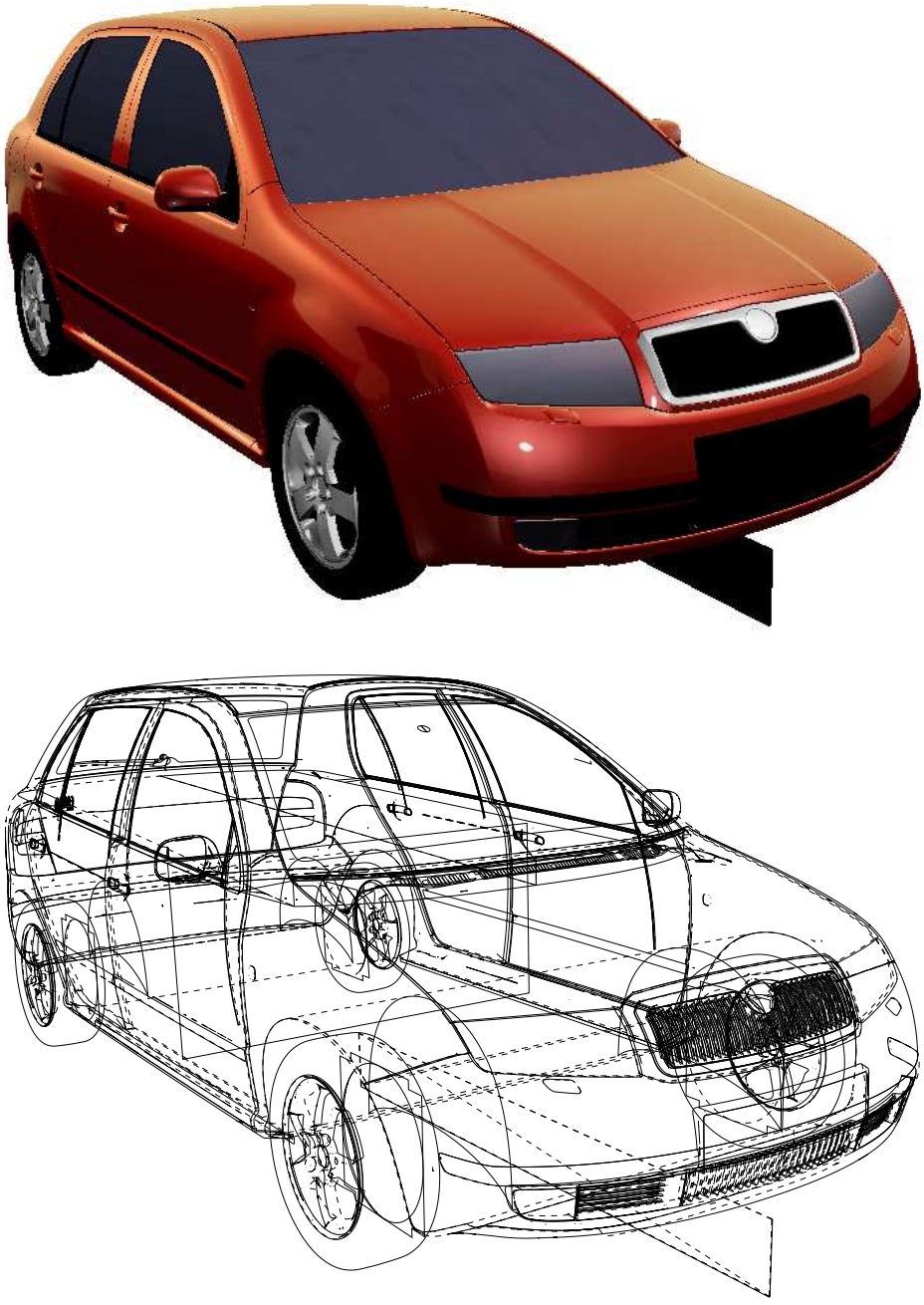
Obrázek 7.1: Model krychle



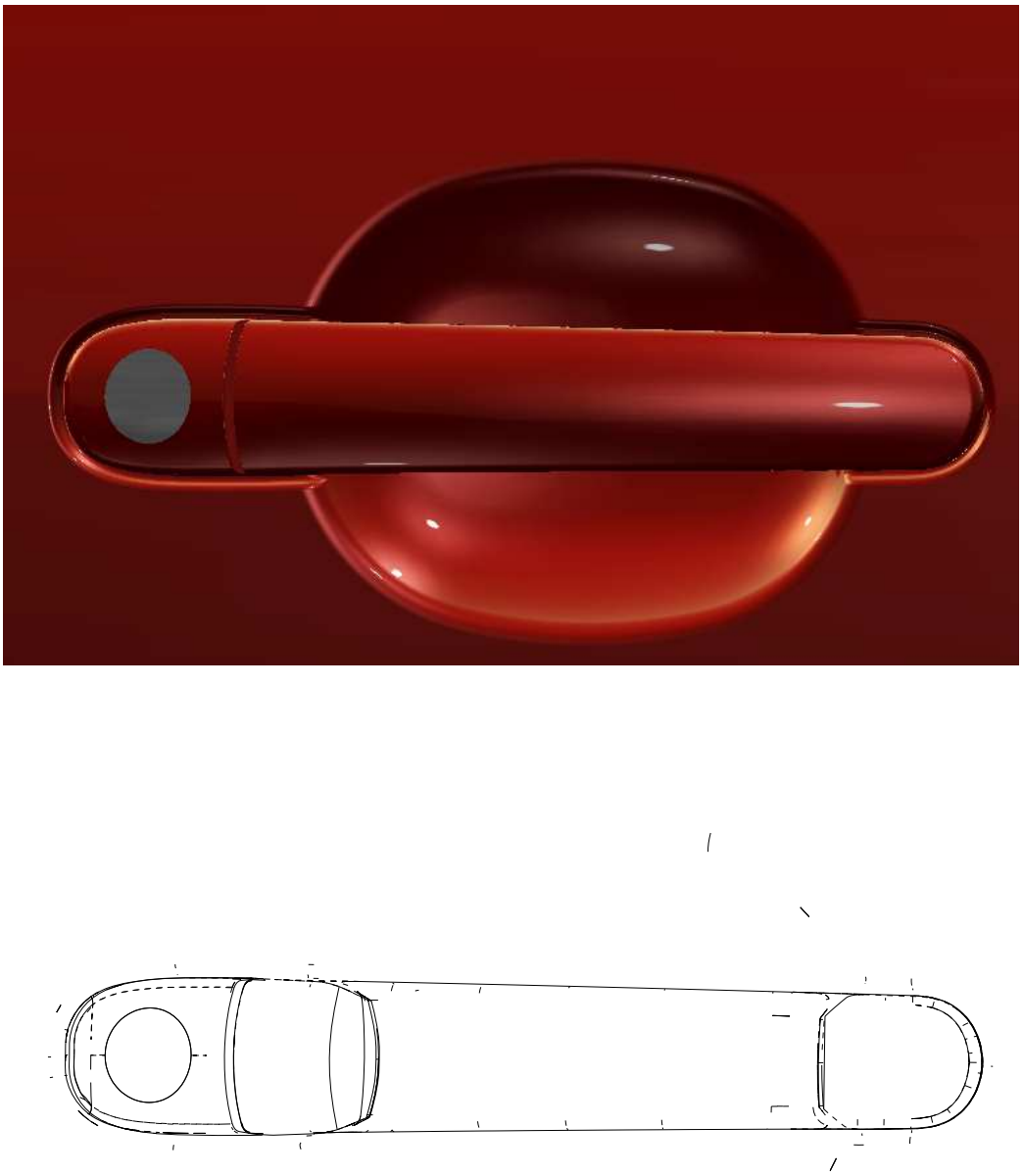
Obrázek 7.2: Model `test.fhs`




Obrázek 7.3: Model `FabiaRS.fhs`



Obrázek 7.4: Model `FabiaEXT_0.05.fhs`



Obrázek 7.5: Detail kliky předních dveří modelu `FabiaEXT_0.05.fhs`

# 8 Závěr

Cílem práce bylo vytvořit modul do aplikace VRUT, který umí zobrazit a vyexportovat obrysové hrany nahraného modelu. Při práce jsem se potýkal s nevyhovující a velmi stručnou dokumentací aplikace, a většinu důležitých informací pro tvorbu nových modulů a práci se scénou a geometrií jsem musel dohledat přímo ve zdrojových kódech jádra a již existujících modulů. Vzhledem k těmto faktům považuji práci za úspěšnou, i když se mi nepodařilo naimplementovat všechny funkce, které jsem měl v úmyslu.

Proto by ještě bylo dobré u modulu dořešit korektní výpočet koeficientů zakrytí v Appelově algoritmu použitém pro řešení viditelnosti hran, omezit přepočet a generování obrysových hran jen na ty geometrie, které se nacházejí v pohledovém jehlanu, a umožnit generování úplných a částečných řezů pomocí ořezávání hran pomocí množiny ploch.

Řešení, které bylo použito pro odstranění vícenásobných bodů při řešení problému falešných hran, by podle mého názoru bylo dobré použít i v jiných modulech. Při načítání modelu by tento jednoduchý mechanizmus snížil obsazenou paměť a při ukládání by zase omezil velikost souboru obsahujícího popis geometrie.

Cílem práce bylo také zobrazovat nalezené obrysy a to pokud možno v reálném čase. I tohoto bodu se podařilo dosáhnout a detekce hran běží na standardním HW v interaktivní rychlosti (přepočet hran nejsložitějšího testovaného modelu trvá zhruba 1/3 sekundy).

# Literatura


[Ado(1985)] *Postscript language tutorial and cookbook*. Adobe Systems Incorporated, 1985. Dostupné z: `www-cdf.fnal.gov/offline/PostScript/BLUEBOOK.PDF`.

[Ado(1999)] *PostScript language reference manual*. Adobe Systems Incorporated, 3 edition, 1999. Dostupné z: `http://www.adobe.com/products/postscript/pdfs/PLRM.pdf`.

[Appel(1967)] ARTHUR APPEL *The notion of quantitative invisibility and the machine rendering of solid*. New York : AMC Inc., 1967.

[Jiří Žára(2010)] JIŘÍ ŽÁRA, BEDŘICH BENEŠ, JIŘÍ SOCHOR, PETR FELKEL *Moderní počítačová grafika*. Holandská 8, 639 00 Brno : Computer Press, a.s., 2010. ISBN 80-251-0454-0.

[Kevin Weiler(1977)] KEVIN WEILER, PETER ATHERTON *Hidden surface removal using polygon area sorting*. 1977.

[Tišnovský(2006)] PAVEL TIŠNOVSKÝ *Seriál Grafické formáty*. Internet Info, s.r.o, 2006. Dostupné z: `http://www.root.cz/serialy/graficke-formaty/`.

[Ska(2009)] VÁCLAV SKALA, JAN HRÁDEK, MARTIN KUCHAŘ *Hash Function for Trianglular Mesh Reconstruction*. In *Recent Advances in Computers*, s. 233–238, Athény, 2009. WSEAS Press. ISBN 978-960-474-099-4.

[Václav Kyba(2010)] VÁCLAV KYBA, ANTONÍN MÍŠEK a další *Dokumentace aplikace VRUT*, 2010.

# Seznam obrázků

# Seznam tabulek