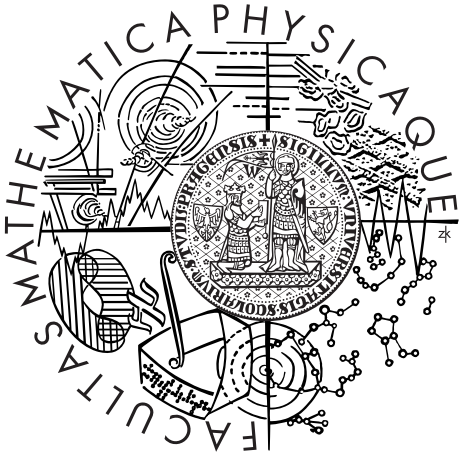
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Jiří Václavík

CSG modelování pro polygonální objekty

Kabinet software a výuky informatiky

Vedoucí diplomové práce: Ing. Jaroslav Křivánek, Ph.D.

Studijní program: Informatika

Studijní obor: softwarové systémy

Praha 2011

Rád bych poděkoval vedoucímu práce Ing. Jaroslavu Křivánkovi, Ph.D., za cenné rady, vstřícnost, trpělivost a poskytnuté materiály. Dále bych chtěl poděkovat svým rodičům a bratrovi Janovi za neustálou podporu nejen během psaní této práce, ale i v průběhu celého studia.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Praze dne ..... Podpis autora

Název práce: CSG modelování pro polygonální objekty

Autor: Jiří Václavík

Katedra: Kabinet software a výuky informatiky

Vedoucí diplomové práce: Ing. Jaroslav Křivánek, Ph.D., Kabinet software a výuky informatiky

Abstrakt: Tato práce se zabývá efektivní a robustní technikou provádění booleovských operací nad polygonálními modely. Plné robustnosti je dosaženo ve vnitřní reprezentaci založené na rovinách a BSP stromech (binary space partitioning trees), v které lze operace provádět přesně v pouhé aritmetice s pevnou přesností. Potřebné konverze z obvyklé reprezentace do vnitřní a zpět včetně jejich důsledků jsou podrobně analyzovány. Výkon metody je optimalizován lokalizačním schématem v podobě adaptivního oktantového stromu. Vzniklá implementace RazeCSG je experimentálně srovnána s implementacemi používanými v praxi Carve a Maya, které nejsou plně robustní. U velkých modelů vykazuje RazeCSG v nejhorším případě jen dvakrát nižší výkon než Carve a je nejméně 130krát rychlejší než Maya.

Klíčová slova: počítačová grafika, 3D, konstruktivní geometrie těles, CSG, efektivní a robustní booleovské operace

Title: CSG modeling for polygonal objects

Author: Jiří Václavík

Department: Department of Software and Computer Science Education

Supervisor: Ing. Jaroslav Křivánek, Ph.D., Department of Software and Computer Science Education

Abstract: This work deals with an efficient and robust technique of performing Boolean operations on polygonal models. Full robustness is achieved within an internal representation based on planes and BSP (binary space partitioning) trees, in which operations can be carried out exactly in mere fixed precision arithmetic. Necessary conversions from the usual representation to the inner one and back, including their consequences are analyzed in detail. The performance of the method is optimized by a localization scheme in the form of an adaptive octree. The resulting implementation RazeCSG is experimentally compared with implementations used in practice Carve and Maya, which are not fully robust. For large models, RazeCSG shows only twice lower performance in the worst case than Carve, and is at least 130 times faster than Maya.

Keywords: computer graphics, 3D, constructive solid geometry, CSG, efficient and robust Boolean operations

Obsah

1. Úvod

Konstruktivní geometrie těles (angl. constructive solid geometry, CSG) je důležitá technika používaná při modelování trojrozměrných objektů v počítačové grafice. Tato technika spočívá v postupném kombinování dvou (jednodušších) objektů pomocí regularizovaných booleovských množinových operací[1] sjednocení, průnik a rozdíl. Takto postupně vznikají složitější formace objektů.

Existuje mnoho softwaru, který tyto booleovské operace podporuje. Za plně přesnou a robustní implementaci se považuje balíček 3D Nef Polyhedra z knihovny CGAL (Computational Geometry Algorithms Library) [8]. Přesnosti je dosaženo pomocí aritmetiky s libovolnou přesností. Nevýhodou implementace však jsou velmi vysoké časové a paměťové nároky výpočtu, které ji činí pro zpracování velkých modelů v praxi nepoužitelnou.

Vedle toho velká část softwaru dosahuje lepší prakticky použitelné efektivity, nicméně za cenu jisté míry nestability a nepřesnosti (např. knihovna Carve [9]).

Cílem práce je implementovat knihovnu, která podporuje výše zmíněné booleovské operace nad polygonálními modely s důrazem na rychlost a stabilitu při práci s velkými objekty.

Kapitola 2 stručně pojednává o související literatuře a vybraných implementacích. Kapitola 3 se zabývá analýzou a návrhem řešení. Kapitola 4 popisuje implementaci, která vznikla na základě navrženého řešení. Kapitola 5 experimentálně srovnává výslednou implementaci s vybranými programy. Kapitola 5 shrnuje celou práci a uzavírá ji zhodnocením splnění stanovených cílů a navržením budoucí práce. V příloze A lze nalézt obsah disku přiloženého k této práci. Zbytek této kapitoly pokračuje vymezením důležitých pojmů.

[1] Regularizovaná booleovská množinová operace je definována jako uzávěr vnitřku (obyčejné) booleovské množinové operace. Tento typ operace se zavádí z toho důvodu, že jejím výsledkem je opět těleso či případně prázdná množina, což u (obyčejné) booleovské operace platit nemusí.

1.1 Vymezení pojmů

Hraniční reprezentace (boundary representation, B-rep) – těleso je reprezentováno pomocí své hranice (povrchu).

Polygonální model/objekt/těleso – často používaný druh hraniční reprezentace, kdy je hranice tělesa zachycena polygonálními stěnami. Orientace stěn (pořadí jejich vrcholů) jednoznačně určuje vnitřek a vnějšek tělesa. Obvykle se používá konvence, kdy jsou stěny při pohledu z vnějšku tělesa orientované proti směru hodinových ručiček.

Manifold (2-varieta) – polygonální objekt lze označit za manifold, pokud se žádné jeho dvě stěny nedotýkají pouze vrcholem (ale alespoň jednou hranou) a každá hrana je sdílena nejvýše dvěma stěnami.

BSP strom (binary space partitioning tree) - úplný binární strom, který rekurzivně rozděluje prostor na poloprostory.

2. Rešerše

První část této kapitoly stručně popisuje důležité poznatky z literatury, z kterých bude čerpáno při analýze a návrhu řešení v kapitole 3. V druhé části kapitoly jsou zmíněny dvě významné implementace z oblasti konstruktivní geometrie těles.

2.1 Literatura

2.1.1 Booleovské operace

Za historicky zásadní dílo na poli konstruktivní geometrie polygonálních těles lze považovat práci autorů Laidlaw, Trumbore a Hughes [6]. Ačkoliv základní myšlenky jejich práce už byly známé dříve, tito autoři byli zřejmě první, kteří algoritmy popsali dostatečně detailně na to, aby mohly být implementovány. Jejich metoda pracuje s hraniční reprezentací modelů a základním geometrickým primitivem jsou body v prostoru. Práce obsahuje vyčerpávající analýzu všech potřebných typů protínání geometrických útvarů. Jedná se o řadu obtížných případů, z kterých plyne nevýhoda metody v podobě vysoké algoritmické složitosti.

Metoda používá běžnou nepřesnou aritmetiku v plovoucí řádové čárce. Slabší míry robustnosti je dosaženo zavedením konstantní tolerance (epsilon), kdy jsou čísla zatížená chybou výpočtu považovaná za sobě rovná, pokud se liší méně než o tuto toleranci. Pokud by tato metoda pracovala v aritmetice s libovolnou přesností, mohla by být plně přesná a robustní, avšak za cenu nepřijatelného výkonu [1].

Novou alternativu ke klasické hraniční reprezentaci polygonálních těles nabídli Naylor, Amanatides a Thibault [4, 3]. Zjistili totiž, že mnohostěny lze reprezentovat pomocí BSP stromů (binary space partitioning trees) a booleovské operace nad nimi provádět spojováním těchto stromů. Výhodou této metody je nižší algoritmická složitost než u předchozí práce, protože spojování BSP stromů není zatížené řešením tolika různých možných situací jako u protínání polygonů.

Sugihara a Iri [7] přišli s myšlenkou nahradit dosud používané základní geometrické primitivum bod za rovinu. Provádění booleovských operací pak nevyžaduje žádnou konstrukci nových geometrických informací (tj. rovin), protože stačí pouze

kombinovat informace stávající. Díky absenci konstrukce je možné přesně vyhodnocovat geometrické predikáty v aritmetice s předem známou pevnou přesností a dosáhnout tak robustnosti s výrazně vyšší efektivitou než v případě aritmetiky s libovolnou přesností.

Bernstein a Fussell [2] spojili tyto dvě dřívější alternativní myšlenky, tj. reprezentaci objektů pomocí BSP stromů a rovinu jako základní geometrické primitivum. Vzniklý systém se v rámci této své přirozené reprezentace vyznačuje jak robustností, přesností a efektivitou, tak i nízkou algoritmickou složitostí. Problém však nastává s konverzemi do klasické hraniční reprezentace a zpět, které jsou nepřesné a pomalé. Autoři provedli experimenty, z kterých vyplývá, že jejich systém dosahuje při iterativních výpočtech 16–28krát vyššího výkonu než plně robustní knihovna CGAL a jen dvakrát nižšího výkonu než nerobustní implementace v aplikaci Autodesk Maya. Tato měření však nezahrnují časově náročné konverze mezi reprezentacemi, které mohou trvat řádově sekundy až minuty.

Na předchozí práci [2] navázali Campen a Kobbelt [1]. Jejich cílem bylo zvýšit celkovou efektivitu metody od začátku do konce (včetně pomalých vstupních a výstupních konverzí) a odstranit různé nedostatky metody (např. nepřesnost vstupní konverze). Pro zvýšení efektivity práce zavádí lokalizační schéma, které je realizované adaptivním oktantovým stromem. Nezbytné výpočty jsou omezeny jen do těch buněk oktantového stromu, kde potenciálně dochází k průniku vstupních objektů. Uvnitř těchto kritických buněk je vstupní geometrie konvertována do reprezentace založené na rovinách a BSP stromech. To dovoluje provádět všechny výpočty přesně v pouhé aritmetice s pevnou přesností. Výsledkem je přesná a robustní implementace, která má vyšší výkon a menší spotřebu paměti než předchozí metody. Toto tvrzení autoři potvrzují experimentálním srovnáním jejich implementace s knihovnou CGAL, kdy naměřili 2,5–13krát vyšší výkon.

2.1.2 Přesná aritmetika

Shewchuk [5] představil velmi rychlé algoritmy pro aritmetiku s libovolnou přesností. Vedle teorie dal autor veřejně k dispozici i zdrojový kód knihovny, která tyto algoritmy implementuje[1]. Tato knihovna má podle jejího autora silnou výhodu

[1] `http://www.cs.cmu.edu/~quake/robust.html`

nad ostatními softwarovými technikami ve výpočtech, kde se pracuje s hodnotami s rozšířenou, ale malou přesností. Knihovna na rozdíl od většiny ostatních knihoven neukládá hodnotu jako sekvenci číslic, ale jako sumu obyčejných čísel v plovoucí řádové čárce. Algoritmy zajišťují, aby se řády číslic jednotlivých složek sumy vzájemně nepřekrývaly. Výhodu tohoto přístupu lze ukázat na příkladu operace součtu $2^{300} + 2^{-300}$. Její výsledek je knihovna schopná uložit do dvou slov (obyčejných čísel v plovoucí řádové čárce). Při klasickém způsobu uložení je však potřeba nejméně 601 bitů a výkon dalších výpočtů s touto hodnotou bude nepříznivě ovlivněn její délkou.

2.2 Software

2.2.1 Knihovna CGAL

Knihovna CGAL (Computational Geometry Algorithms Library) [8] je rozsáhlá kolekce spolehlivých algoritmů výpočetní geometrie skládající se z mnoha balíčků. Z hlediska této práce je zajímavý balíček 3D Nef Polyhedra, který se zabývá konstruktivní geometrií těles. Tento balíček v současnosti představuje nejpokročilejší implementaci v této oblasti. Podporovány jsou booleovské operace doplněk, sjednocení, průnik, rozdíl a symetrický rozdíl a topologické operace vnitřek, uzávěr a hranice. Reprezentované objekty nemusí být jen uzavřená tělesa, ale mohou mít i otevřené hranice.

Při výpočtech je využívána aritmetika s libovolnou přesností, proto lze implementaci charakterizovat jako plně přesnou a robustní. V důsledku této aritmetiky a vnitřně použité datové struktury je však implementace velmi časově i paměťově neefektivní. Podle experimentů v [1] operace nad dvěma modely po 200 tisících stěnách trvá řádově minuty a spotřebuje asi 5 GB paměti.

2.2.2 Knihovna Carve

Knihovna Carve [9, 10] je specializovaná přímo na booleovské množinové operace nad polygonálními modely těles. Podporovány jsou operace sjednocení, průnik, rozdíl a symetrický rozdíl.

Na rozdíl od knihovny CGAL nepoužívá knihovna Carve aritmetiku s libovolnou přesností, ale spoléhá na nativní čísla v plovoucí řádové čárce s dvojitou přesností. Podle jejího autora je proto knihovna robustnější než knihovny používající jednoduchou přesnost a významně rychlejší než knihovny závislé na libovolné přesnosti. Knihovnu však rozhodně nelze považovat za plně robustní. Z jejího zdrojového kódu je patrné použití konstantní tolerance (epsilon) při porovnávání dvou hodnot podobně jako v [6], čímž je dosaženo pouze slabší míry robustnosti. Výchozí hodnotu tolerance dovoluje knihovna uživateli předefinovat.

Knihovna Carve neklade na vstupní modely těles mnoho omezení. Stěny musí být planární a nesmí protínat samy sebe. Mohou však mít libovolný počet hran a nemusí být konvexní. Vstupní modely smí obsahovat uzavřené i otevřené povrchy a mohou se skládat z více disjunktních, vnořených nebo dotýkajících se povrchů. Povrchy však nesmí protínat samy sebe a vnořené uzavřené povrchy musí být konzistentně orientované.

Výpočet je optimalizován systémem adaptivního dělení prostoru. Knihovna by proto měla zvládat velké modely.

Carve také podporuje interpolaci libovolných dat přiřazených k vrcholům stěn (např. souřadnice textur).

3. Analýza a návrh řešení

Řešení analyzované v této kapitole přímo vychází z metody [1] a částečně také z metody [2], přičemž první zmíněná metoda na druhou zmíněnou navazuje, zásadně ji vylepšuje a odstraňuje její nedostatky.

Na rozdíl od [1] se tato práce omezuje pouze na booleovské operace nad správně orientovanými modely, které neprotínají samy sebe ([1] se navíc zabývá konstrukcí vnějšího obalu modelu, který protíná sám sebe). Další odlišností je způsob vnitřní reprezentace vstupních polygonálních modelů. Zatímco v [1] se k tomu využívá tzv. půlhrana (angl. halfedge), v této práci postačí indexy do seznamu vrcholů definující stěnu, protože v rámci provedené analýzy nebyly objeveny významné důvody pro nasazení algoritmicky složitější půlhrany.

Pro některé části metody jsou v této práci rozvinuty vlastní postupy v případě, že v literatuře nejsou dostatečně rozebrané. K některým postupům v literatuře je také navržena vlastní alternativa, pokud se jeví pro tuto práci jako výhodnější. Tyto skutečnosti budou průběžně zmiňovány na příslušných místech dále v této kapitole.

Metoda byla zvolena po pečlivém prostudování různých přístupů zejména z poslední doby. Důvodem pro její zvolení byl především fakt, že metoda nabízí nejen přesnost a robustnost, ale i vyšší efektivitu než předchozí metody tohoto typu a tedy naplňuje důležitý bod zadání práce. Bylo přihlédnuto i k tomu, že jádro metody není algoritmicky tak složité jako u klasických algoritmů, které musí přímo řešit mnoho různých situací vzájemného protínání polygonů, což činí implementaci těchto algoritmů náchylnější k chybám.

3.1 Základ metody

Základní filozofie metody je čerpána z [2]. Na rozdíl od většiny klasických metod budou základním prostředkem pro reprezentaci geometrie nikoliv souřadnice bodů v prostoru, ale koeficienty rovnice roviny ($ax + by + cz + d = 0$). Každý koeficient bude za účelem vysokého výkonu uložený do nativního číselného typu s pevně danou přesností v plovoucí řádové čárce. Body v prostoru budou určeny jako

průsečíky trojic rovin.

Použití rovin místo bodů je klíčové pro výkon a robustnost celé metody. Při provádění booleovské operace se totiž v reprezentaci založené na rovinách (na rozdíl od té založené na bodech) lze úplně vyhnout konstrukci nových geometrických informací [7]. Na základě toho můžeme celou metodu vystavět pouze na predikátech, které jsou zároveň rychlé i přesné. Tyto predikáty se dají přesně vyhodnocovat v pouhé aritmetice s předem známou pevnou přesností, protože hloubka libovolného aritmetického výrazu predikátů je omezená konstantou (kvůli absenci konstrukce nových geometrických informací).

Vyhodnocování predikátů lze dále zrychlit filtrovacími technikami z [5]. Popíšeme si princip jednoduché, ale účinné jednostupňové filtrovací techniky. Výraz predikátu nejprve vyhodnotíme v nejrychlejší dostupné (a tedy nepřesné) aritmetice. K vypočtené hodnotě také odhadneme maximální chybu, tj. odchylku od správné hodnoty. Pokud je vypočtená hodnota natolik vzdálená od mezní hodnoty predikátu, že ani maximální chyba nepřesného výpočtu nemůže vést k nesprávnému závěru predikátu, prohlásíme tento závěr za definitivní. V opačném případě výpočet opakujeme v přesné aritmetice (s využitím velmi efektivních algoritmů z [5]) a definitivní závěr určíme na jeho základě.

3.1.1 Predikáty

Algoritmus se opírá o čtyři predikáty, jejichž vstupem jsou koeficienty rovnic několika rovin. Potřebné predikáty jsou v [2] definovány takto:

Shodnost. *Dvě roviny p a q jsou shodné, právě když jsou všechny subdeterminanty řádu dva následující matice nulové:*

$$\begin{pmatrix} p_a & p_b & p_c & p_d \\ q_a & q_b & q_c & q_d \end{pmatrix}$$

Shodná orientace. *Pokud je známo, že p a q jsou shodné, pak p a q jsou shodně orientované, právě když jsou všechny součiny $p_aq_a, p_bq_b, p_cq_c, p_dq_d$ nezáporné.*

Platnost bodu. *Bod (p, q, r) je dobře definovaný (tj. platný), právě když je*

následující determinant nenulový:

$$\begin{vmatrix} p_a & p_b & p_c \\ q_a & q_b & q_c \\ r_a & r_b & r_c \end{vmatrix}$$

Orientace. *Je-li bod (p, q, r) platný, pak tento bod leží za rovinou s, na rovině s, resp. před rovinou s, právě když je následující výraz záporný, nulový, resp. kladný:*

$$\begin{vmatrix} p_a & p_b & p_c \\ q_a & q_b & q_c \\ r_a & r_b & r_c \end{vmatrix} \cdot \begin{vmatrix} p_a & p_b & p_c & p_d \\ q_a & q_b & q_c & q_d \\ r_a & r_b & r_c & r_d \\ s_a & s_b & s_c & s_d \end{vmatrix}$$

3.1.2 Konvexní polygon

Použijeme reprezentaci konvexního polygonu založenou na rovinách podle vzoru [2]. Takový polygon se skládá z podpůrné roviny s a ze seznamu ohraničujících rovin $\{b_i\}_{i \in \mathbb{Z}_n}$, které definují jeho hrany (v pořadí proti směru hodinových ručiček). Podpůrná rovina prochází všemi vrcholy polygonu. Každá ohraničující rovina prochází dvěma vrcholy odpovídající hrany a libovolným bodem ležícím mimo podpůrnou rovinu. Vrcholy polygonu definují trojice rovin: $v_i = (s, b_{i-1}, b_i)$.

Všimněme si, že takto nelze reprezentovat hrany, které svírají přímý úhel (180 stupňů), protože příslušná trojice rovin by se neprotnula v bodě, ale v přímce. Reprezentovatelné polygony tedy musí být ryze konvexní.

3.1.3 Konstrukce konvexního polygonu z roviny

Operace konstrukce konvexního polygonu pochází z [2]. Vstupem je rovina h a výstupem konvexní polygon, který představuje rovinu h po ořezání velmi velkou krychlí. Tato krychle je osově zarovnaná a ohraničená neměnnými rovinami $x^+, x^-, y^+, y^-, z^+, z^-$. Krychle musí být dostatečně velká, aby obklopovala zpracovávané modely. Používá se stále stejná krychle.

Nejprve zvolíme čtyři z ohraničujících rovin krychle. Vybranými rovinami ohraničíme rovinu h, čímž získáme počáteční polygon. Volba rovin závisí na tom,

která souřadnice normálového vektoru roviny h je dominantní. Zvolíme ty roviny, které nejsou spojené s dominantní souřadnicí. Např. je-li dominantní souřadnice h_z, zvolíme roviny x^+, y^+, x^-, y^-. Pořadí rovin udává znaménko dominantní souřadnice.

Zvolené roviny představující počáteční ohraničující roviny polygonu. Podpůrnou rovinou je rovina h. Tento počáteční polygon následně postupně ořežeme oběma dosud nevyužitými ohraničujícími rovinami krychle pomocí níže popsané rutiny (viz 3.1.4). Tím zajistíme, že polygon bude ležet uvnitř krychle. Tento polygon je výsledkem operace.

3.1.4 Ořezání konvexního polygonu rovinou

Algoritmus ořezání konvexního polygonu rovinou pochází opět z [2]. Pomocí dvou doplňkových operací ořezání lze polygon rozdělit na část před rovinou a za ní.

Vstupem je konvexní polygon $(s, \{b_i\}_{i\in\mathbb{Z}_n})$ a řezná rovina h. Výstupem je část polygonu, která leží v kladném poloprostoru řezné roviny h, nebo prázdný výsledek, pokud taková část neexistuje.

Nejjednodušší případ nastává, pokud je podpůrná rovina polygonu s shodná s řeznou rovinou h. V případě shodné orientace těchto rovin je jako výsledek vrácen nezměněný vstupní polygon, jinak je vrácen prázdný výsledek.

V obecném případě se bude výsledný polygon skládat z podpůrné roviny s a výběru z ohraničujících rovin vstupního polygonu a řezné roviny h. Tento výběr je na začátku prázdný. Zůstane-li prázdný až do skončení algoritmu, znamená to, že žádná část vstupu se nenachází před řeznou rovinou.

Postupně zpracujeme všechny ohraničující roviny vstupního polygonu podle jejich pořadí. Pro každou rovinu b_i určíme, zda do výstupního výběru (ohraničujících rovin) přidat rovinu b_i, řeznou rovinu h či obě (a v jakém pořadí). Toto rozhodnutí závisí na orientaci tří specifických vrcholů vzhledem k řezné rovině h (viz 3.1.1). Dva z těchto vrcholů tvoří hranu odpovídající rovině b_i: $v_i = (s, b_{i-1}, b_i)$ a $v_{i+1} = (s, b_i, b_{i+1})$. Třetí vrchol je jejich přímý předchůdce $v_{i-1} = (s, b_{i-2}, b_{i-1})$. Celkem existuje 27 možností orientace těchto tří vrcholů vzhledem k řezné rovině (pro každý vrchol tři možnosti). Jak přesně má vypadat výstup pro jednotlivé možnosti, lze nalézt v [2].

3.2 Konverze modelu mezi reprezentacemi

Algoritmy konverze mezi reprezentacemi založenými na bodech a rovinách nejsou matematicky nijak komplikované. Ovšem v reprezentaci s omezenou přesností vyvstává problém, jak tyto konverze provádět tak, aby zkonvertovaný model přesně (případně co nejpřesněji) odpovídal původnímu modelu.

V případě konverze do reprezentace založené na rovinách provedené nepřesně je ovlivněna nejen geometrie, ale i topologie modelu. Přesné provedení této konverze znamená přesně proložit všechny stěny a hrany modelu rovinami. Uvidíme, že toho lze dosáhnout, pokud daný model splňuje určité podmínky (viz níže). Pro opačný případ bude ukázán postup, jak mírným snížením přesnosti geometrie modelu (vstupní kvantizace) dosáhnout toho, aby model takové podmínky splňoval a mohl tedy následně být zkonvertován přesně (alespoň se zachováním topologie).

Při zpětné konverzi do reprezentace založené na bodech je zachována topologie modelu, ale snížena přesnost jeho geometrie (výstupní kvantizace). Lze dosáhnout pouze omezené přesnosti souřadnic vrcholů, kterou nabízí cílová reprezentace.

3.2.1 Podmínky přesného proložení stěn modelu rovinami

Při konverzi popsané v [2] jsou stěny a hrany modelu prokládány rovinami nepřesně. V důsledku toho dochází k ovlivnění geometrie i topologie modelu, kdy z každého sdíleného vrcholu se stupněm větším než tři může vzniknout více blízkých vrcholů. Vzniklý model proto může obsahovat netěsnosti či vzájemně se protínající stěny. Aby byl model použitelný pro další zpracování, musí na něj nejprve být aplikována speciální opravná metoda.

Tento nedostatek konverze je napraven v [1] pečlivou analýzou přesné konverze a z ní vyplývajících požadavků na model. Cílem je zachovat topologii i geometrii modelu, pokud je to současně možné, nebo alespoň samotnou topologii modelu. Tímto přístupem se zabývá následující text.

Jako koeficienty rovnice roviny a, b, c lze vzít souřadnice normálového vektoru roviny, která stěnou prochází. Normálový vektor se rovná vektorovému součinu vektorů dvou různých hran stěny. Koeficient d se pak dá z rovnice roviny ($ax +$

$by + cz + d = 0$) jednoduše dopočítat po dosazení souřadnic libovolného vrcholu stěny. Polygonální stěny je nutné triangulovat (rozdělit na trojúhelníky), protože kvůli omezené přesnosti reprezentace modelů na počítači je obecně nevyhnutelné, že všechny vrcholy každé stěny neleží přesně v jedné rovině.

Vzhledem k omezené přesnosti koeficientů rovnice rovin (viz 3.1) je zřejmé, že řádové rozmezí významných číslic souřadnic vrcholů musí být omezené konstantou, aby vůbec bylo možné stěny modelu rovinami přesně proložit. Ona konstanta je pro každý model specifická. Pokud tento požadavek nebude splněný, pak přesnost koeficientů rovnice roviny nemusí pro přesné vystihnutí stěn stačit.

Jaký je tedy konkrétní požadavek na maximální podporovanou přesnost konkrétního modelu? Nechť V označuje maximální počet číslic potřebný pro vyjádření kterékoli souřadnice vrcholů modelu a E maximální počet číslic potřebný pro vyjádření kterékoli souřadnice vektorů hran modelu (obojí v pevné řádové čárce). Z postupu výpočtu koeficientů vyplývá, že koeficienty a, b, c (vektorový součin hran) mohou mít maximálně $2E+1$ číslic[1] a koeficient $d = -(ax+by+cz)$ maximálně $(2E+1)+V+2 = 2E+V+3$ číslic. Pro uložení každého koeficientu roviny je tedy potřeba maximálně $2E+V+3$ číslic. Jelikož platí, že $E \leq V+1$ (hrana s největší možnou souřadnicí), lze předchozí výraz přepsat jako $3V+5$. Koeficienty roviny tedy v nejhorším případě vyžadují přibližně **trojnásobnou přesnost** vzhledem k výchozí přesnosti souřadnic vrcholů modelu. Konkrétní postup tohoto odvození v [1] se liší od výše uvedeného postupu (podle autora subjektivně intuitivnějšího), nicméně výsledky se shodují.

3.2.2 Vstupní kvantizace modelu

Běžně používaný hardware však tak velkou přesnost, která byla odvozena výše pro nejhorší případ, nativně nepodporuje. Uložení koeficientů do nativního typu je přitom klíčové pro výkon systému.

Problém lze vyřešit **kvantizací modelu na nižší přesnost** než výchozí, tj. snížením přesnosti (zaokrouhlením) souřadnic všech vrcholů vstupního modelu [1]. Kvantizace nenaruší topologii modelu, ale pouze do určité míry znepřesní je-

[1] V pevné řádové čárce platí pro dvě čísla vyjádřená na P a Q číslic, kde $P \geq Q$: jejich součin má max. $P+Q$ číslic a jejich součet či rozdíl max. $P+1$ číslic.

ho geometrii, nicméně je třeba si uvědomit, že každý model uložený v počítači je už kvantizovaný na nějakou základní přesnost. Míra snížení přesnosti souřadnic u konkrétního modelu záleží zejména na maximální relativní délce hran vzhledem k rozměru modelu (přesněji na max. počtu číslic potřebném pro vyjádření kterékoli souřadnice vektorů hran). Delší hrany vyžadují větší úbytek přesnosti, zatímco kratší hrany úbytek menší.

Jaká je nejnižší přesnost, na kterou musí být libovolný model kvantizován v nejhorším případě? Z odvození v 3.2.1 vyplývá, že přibližně třetina přesnosti koeficientů rovnice roviny. Jsou-li koeficienty uložené s dvojitou přesností podle normy IEEE 754 (53 číslic), pak musí být model v nejhorším případě kvantizován na 16 číslic. Výchozí přesnost modelů bývá většinou jednoduchá, tj. 24 číslic. Při použití dvojité přesnosti pro koeficienty tedy model v nejhorším případě **ztratí třetinu výchozí přesnosti**.

Jednoduché předzpracování vstupního modelu může probíhat tak, že každý model je vždy kvantizován na přesnost právě 16 číslic. Tím je zaručena dostatečná přesnost koeficientů rovnice roviny, ale na druhou stranu je geometrie většiny modelů zkreslena více než nezbytně.

Lepší řešení je zvolit optimální přesnost jednotlivě pro konkrétní model. Hledat se začíná postupně od výchozí přesnosti modelu směrem dolů. V každém cyklu se pro každou stěnu modelu dočasně sníží přesnost souřadnic jejích vrcholů, vypočtou se z nich koeficienty rovnice roviny a otestuje se, jestli souřadnice rovnici přesně vyhovují. (Kvantizaci nelze provádět pro každou stěnu jednotlivě, protože by tím došlo k rozpojení sdílených vrcholů a narušení topologie modelu.) Pokud testy dopadnou úspěšně pro všechny stěny, je model kvantizován na danou přesnost. Jinak se pokračuje s přesností nižší o jednu číslici.

Příliš agresivní kvantizace může způsobit zdegenerování některých stěn (např. v důsledku sloučení blízkých vrcholů). Kvůli nutnosti triangulovat model během předzpracování se omezíme jen na jednodušší případ, a to degeneraci trojúhelníků. Trojúhelníková stěna může zdegenerovat na úsečku nebo bod. Takto zdegenerované trojúhelníky nepředstavují zásadní problém pro další zpracování modelu. Z modelu je lze jednoduše odstranit, protože nadále nedefinují hranici objektu. Tuto roli převzaly sousední trojúhelníky.

Ve vzácných případech však může dokonce dojít k převrácení orientace stěn, což je daleko závažnější. Tímto problémem se zabývá část 6.1.3.

3.2.3 Konverze stěny do reprezentace založené na rovinách

Na základě předchozích částí (3.2.1 a 3.2.2) můžeme předpokládat nezdegenerovanou trojúhelníkovou stěnu. Následující postup z [1] je však aplikovatelný i na nezdegenerovaný, ryze konvexní a přesně planární polygon.

Podpůrnou rovinu získáme vektorovým součinem dvou hran a dopočtením koeficientu d (viz 3.2.1). Každou ohraničující rovinu získáme z dvou vrcholů příslušné hrany a třetího bodu, který leží mimo podpůrnou rovinu. Je důležité tento třetí bod kvantizovat na stejnou přesnost, kterou mají ostatní vrcholy modelu. Souřadnice třetího bodu můžeme získat např. minimálním posunutím souřadnic jednoho z vrcholů ve směru normálového vektoru trojúhelníku tak, aby bod po kvantizaci (na přesnost ostatních vrcholů) ležel mimo podpůrnou rovinu.

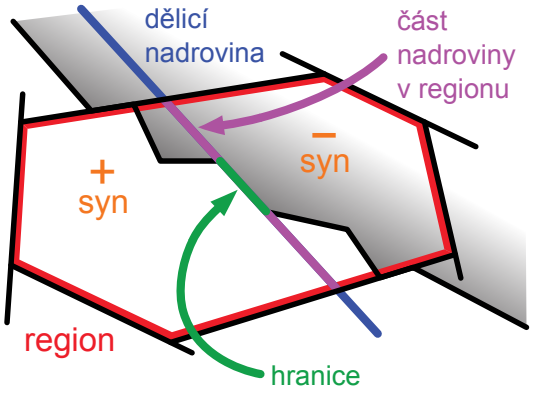
Vrcholy vzniklé stěny reprezentované příslušnými trojicemi rovin jsou dobře definované z hlediska predikátu platnosti bodu (viz 3.1.1).

3.2.4 Konverze stěny do reprezentace založené na bodech

Vrcholy stěny reprezentované pomocí rovin se nachází v průsečíku podpůrné roviny stěny a dvou po sobě jdoucích ohraničujících rovin. Souřadnice průsečíku jsou řešením soustavy tří lineárních rovnic o třech neznámých.

Soustavu lze vyřešit např. Cramerovým pravidlem. Výhodou tohoto způsobu je, že můžeme znovu využít rutiny pro výpočet determinantů použité dříve u predikátů. Každou souřadnici průsečíku tedy vyjádříme jako podíl dvou determinantů. Abychom mohli souřadnice určit s plnou přesností výstupní reprezentace, musíme výpočty determinantů provádět v přesné aritmetice algoritmy z [5]. Tyto algoritmy bude nutné rozšířit o dělení s přesností výstupní reprezentace.

Všimněme si, že determinant soustavy rovnic (dělitel ve vyjádřeném řešení soustavy) odpovídá determinantu použitém v predikátu platnosti bodu (viz 3.1.1). Řešení soustavy tedy existuje (nenulový dělitel), právě když platí predikát platnosti bodu.



Obrázek 3.1: Části vnitřního uzlu BSP stromu (buňkový diagram) *(převzato z [2])*

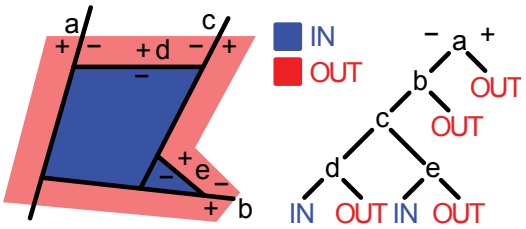
Takto získaná reprezentace však postrádá informace o vzájemném propojení vrcholů. Řešením tohoto problému se zabývá část 3.5.1.

3.3 BSP stromy

Booleovské operace budeme provádět podle vzoru v [1] a [2] nad objekty v reprezentaci založené na rovinách a BSP stromech (binary space partitioning trees). Myšlenka reprezentace mnohostěnu, který neprotíná sám sebe, pomocí BSP stromu pochází z [4]. Další důležité algoritmy (např. spojení dvou BSP stromů) byly popsány v [3]. Geometrie v těchto pracích však byla založená na bodech, nikoliv na rovinách. Tyto algoritmy byly mírně přizpůsobeny pro reprezentaci založenou na rovinách v [2] a [1].

BSP strom je úplný binární strom, který představuje rekurzivní dělení prostoru (s libovolnou dimenzí) na poloprostory. Podprostor, který přísluší nějakému uzlu stromu, označíme jako **region** tohoto uzlu. Regionem stromu budeme rozumět region jeho kořenu. Počáteční region stromu je předem daný (např. celý prostor).

Každý vnitřní uzel stromu obsahuje jednu orientovanou dělicí nadrovinu, která dělí jeho region na dva poloprostory. Poloprostor před nadrovinou (resp. za ní) se označuje jako kladný (resp. záporný) poloprostor. Tyto poloprostory představují regiony synů uzlu. Syny lze podle této logiky označovat jako kladné a záporné. Hranice (stěny) objektu reprezentovaného stromem leží uvnitř regionů vnitřních uzlů stromu na jejich dělicích nadrovinách. Viz obr. 3.1.



Obrázek 3.2: BSP strom reprezentující modrý objekt *(převzato z [2])*

Regiony listů stromu (dále nerozdělené) se nazývají **buňky**. Buňky jsou konvexní, pokud byl konvexní počáteční region. Region uzlu je definován jako průnik poloprostorů, do kterých vstoupíme na cestě od kořene stromu k tomuto uzlu. **Triviální strom** obsahuje pouze jeden uzel a představuje jedinou buňku.

Aby BSP strom mohl reprezentovat mnohostěn, je nutné do listů stromu přidat informaci o tom, zda se buňka nachází uvnitř tělesa (IN), nebo mimo něj (OUT) (viz obr. 3.2). Reprezentovaný mnohostěn se pak skládá právě z buněk označených IN.

3.3.1 Konstrukce stromu

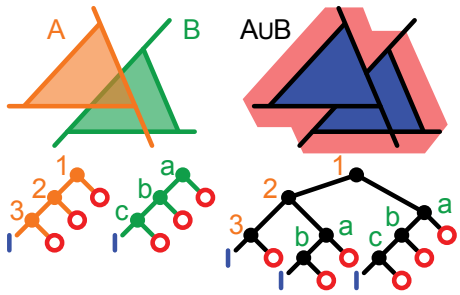
BSP strom reprezentující polygonální objekt lze zkonstruovat postupným vkládáním všech stěn objektu do původně prázdného (triviálního) stromu (buňka OUT) [4]. Proces vkládání stěny se provádí rekurzivně. Stěna je jako první zpracována v kořenu stromu.

Splývá-li stěna s nadrovinou vnitřního uzlu, pak vkládání stěny ihned končí.

Zasahuje-li stěna současně do prostoru před nadrovinou vnitřního uzlu i za ní, musí být rozdělena podle této nadroviny na dvě části (viz 3.1.4). Stěna, resp. její části jsou pak rekurzivně zpracovány v těch synovských uzlech, do kterých spadají.

V případě, kdy dojdeme až k listu, je tento list (bez ohledu na označení) nahrazen novým uzlem. Tento uzel bude obsahovat nadrovinu vkládané stěny a oba jeho synové budou listy – kladný označený OUT a záporný IN.

Ve vnitřních uzlech lze navíc volitelně uchovávat seznam vložených stěn (či odkazů na ně). Toho lze využít např. při zpětné extrakci stěn ze stromu za účelem určení původu extrahovaných stěn. Stěny se přidávají pouze do uzlů, kde došlo



Obrázek 3.3: Výsledek operace sjednocení *(převzato z [2])*

k ukončení rekurze. Tedy do vnitřních uzlů, kde stěna splývá s nadrovinou uzlu, a na místo původních listů později nahrazených novým uzlem.

3.3.2 Spojení dvou stromů

Algoritmus 1 Spojení dvou stromů T_1 a T_2

1: **if** T_1 nebo T_2 je triviální strom (buňka) **then**
2: Spojíme strom s buňkou (T_1 a T_2), čímž dostaneme strom T_3.
3: **return** T_3
4: **else**
5: Rozdělíme strom T_2 nadrovinou v kořenu stromu T_1 algoritmem 2, čímž dostaneme rozdělení T_2 na dva stromy: T_2^+ v kladném a T_2^- v záporném poloprostoru nadroviny.
6: Spojíme záporný synovský podstrom T_1 se stromem T_2^- algoritmem 1, čímž dostaneme strom T^-.
7: Spojíme kladný synovský podstrom T_1 se stromem T_2^+ algoritmem 1, čímž dostaneme strom T^+.
8: Vytvoříme nový strom T_3: v kořenu bude nadrovina z kořenu T_1, záporný synovský podstrom bude T^- a kladný T^+.
9: **return** T_3
10: **end if**

Spojením dvou BSP stromů lze provést booleovskou operaci nad objekty, které tyto stromy reprezentují (viz obr. 3.3). Vstupem jsou stromy T_1, T_2 a typ booleovské operace (např. průnik). Výsledný strom označíme T_3. Popíšeme si algoritmy pro spojování BSP stromů z [3].

Podstatně jednodušeji než spojování obecných stromů probíhá spojování, pokud je T_1 nebo T_2 (či oba) triviální strom (buňka). Za těchto okolností je výsledkem operace přímo jeden z těchto stromů nebo doplněk jednoho z nich, a to podle konkrétní situace a typu booleovské operace. Doplněk BSP stromu se konstruuje jednoduše znegováním označení buněk v listech, tj. záměnou označení IN za OUT a obráceně. Např. při operaci sjednocení (resp. průnik) je výsledkem triviální strom, pokud symbolizuje buňku označenou IN (resp. OUT); v opačném případě (OUT, resp. IN) je výsledkem druhý strom. Při operaci rozdíl je výsledkem buď strom T_1, nebo doplněk stromu T_2 v závislosti na tom, který strom je triviální a jaký druh buňky představuje.

Při spojování netriviálních BSP stromů budeme potřebovat operaci dělení stromu nadrovinou. Tato operace představuje nejsložitější část celého algoritmu spojování stromů. Princip operace bude vysvětlen níže. Jeden ze stromů rozdělíme dělicí nadrovinou z kořenu druhého stromu na dvě části (stromy). Přitom si můžeme vybrat, který ze stromů bude rozdělen. V tomto popisu budeme dělit strom T_2.

Kořen výsledného stromu T_3 bude obsahovat dělicí nadrovinu z kořenu nedělených stromu T_1. Pokud v uzlech navíc uchováváme vložené stěny, pak bude kořen stromu T_3 obsahovat jednak stěny z kořenu stromu T_1 a jednak stěny z toho uzlu děleného stromu T_2, jehož nadrovina je shodná s nadrovinou kořenu stromu T_3 (takový uzel se totiž neobjeví ani v jednom z rozdělení stromu T_2).

Stromy vzniklé rozdělením stromu T_2 jsou definované na stejných regionech jako synovské podstromy stromu T_1. Tím vznikly dva nové podproblémy, které jsou formálně identické původnímu problému. Jejich rekurzivním vyřešením získáme synovské podstromy kořenu T_3.

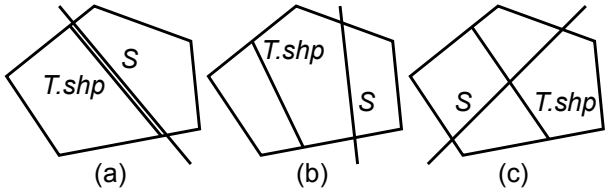
Celý postup spojení dvou stromů shrnuje algoritmus 1.

Nyní popíšeme, jak probíhá operace dělení stromu T v daném regionu nadrovinou S. Nejjednodušší případ nastává, dělíme-li triviální strom T (tedy buňku). Výsledkem dělení jsou tehdy dvě kopie stromu.

Pokud je T netriviální strom, budeme se zabývat vzájemnou polohou té části dělicí nadroviny kořenu stromu T a té části nadroviny S, které se nachází uvnitř regionu stromu T. Obr. 3.4 ukazuje možná uspořádání seskupená podle podob-

Algoritmus 2 Dělení stromu T nadrovinou S na části před nadrovinou a za ní

1: **if** T je triviální strom **then**
2: **return** (T, T)
3: **else if** nadrovina v kořenu T je shodná s nadrovinou S **then**
4: **if** nadrovina v kořenu T je shodně orientovaná s nadrovinou S **then**
5: **return** (záporný podstrom T, kladný podstrom T)
6: **else**
7: **return** (kladný podstrom T, záporný podstrom T)
8: **end if**
9: **else**
10: Určíme vzájemnou polohu polygonálních reprezentací nadroviny z kořenu T a nadroviny S v regionu T.
11: **if** S zasahuje jen do regionu jednoho synovského podstromu T **then**
12: Existují 4 podobné možnosti. Popíšeme jen tu, kdy nadrovina S zasahuje do kladného synovského podstromu T a polygonální reprezentace nadroviny z kořenu T leží v záporném poloprostoru nadroviny S.
13: Algoritmem 2 rozdělíme zasažený kladný synovský podstrom T nadrovinou S, čímž dostaneme stromy (T_+^{S-}, T_+^{S+}).
14: Vytvoříme nový strom T^{S-}: v kořenu bude nadrovina z kořenu T, záporný synovský podstrom bude záporný synovský podstrom T a kladný T_+^{S-}.
15: **return** (T^{S-}, T_+^{S+})
16: **else**
17: Algoritmem 2 rozdělíme záporný synovský podstrom T nadrovinou S, čímž dostaneme stromy (T_-^{S-}, T_-^{S+}).
18: Algoritmem 2 rozdělíme kladný synovský podstrom T nadrovinou S, čímž dostaneme stromy (T_+^{S-}, T_+^{S+}).
19: Vytvoříme nový strom T^{S-}: v kořenu bude nadrovina z kořenu T, záporný synovský podstrom bude T_-^{S-} a kladný T_+^{S-}.
20: Vytvoříme nový strom T^{S+}: v kořenu bude nadrovina z kořenu T, záporný synovský podstrom bude T_-^{S+} a kladný T_+^{S+}.
21: **return** (T^{S-}, T^{S+})
22: **end if**
23: **end if**



Obrázek 3.4: Možné situace při dělení stromu T nadrovinou S. $T.shp$ (sub-hyperplane) označuje část nadroviny kořenu stromu T omezenou na jeho region. V regionu stromu T platí, že $T.shp$ a S: (a) jsou shodné (stejně či opačně orientované), (b) neprotínají se (čtyři možná uspořádání), (c) protínají se. Celkem sedm možných uspořádání *(převzato z [2])*.

nosti do tří případů – (a), (b), (c). Dohromady existuje sedm uspořádání.

V případě shodnosti obou nadrovin (a) jsou výsledkem přímo oba synovské podstromy stromu T. Jinak nadrovina S protíná jen region jednoho (b), resp. regiony obou synovských podstromů stromu T (c). Abychom rozlišili druh uspořádání, určíme polohu polygonálních reprezentací nadrovin uvnitř regionu.

Nejprve získáme polygonální reprezentace těchto nadrovin (viz 3.1.3) a pak je omezíme na region stromu T, tj. ořežeme je nadrovinami, které tento region definují (viz 3.1.4). Vzájemnou polohu těchto polygonů určíme pomocí predikátu orientace (viz 3.1.1).

Následně stejným způsobem rekurzivně rozdělíme synovské podstromy, do jejichž regionu zasahuje nadrovina S. V případě (b) tedy rozdělíme jen jeden synovský podstrom a v případě (c) oba. Vzniklá rozdělení vhodně zkombinujeme do jednoho rozdělení před nadrovinou S a druhého za ní, čímž dostaneme požadované dělení stromu T nadrovinou S.

Celý postup dělení stromu nadrovinou shrnuje algoritmus 2.

Časová složitost spojování dvou stromů v nejhorším případě je $\Theta(n^2)$, kde n je počet uzlů stromu [3].

3.3.3 Extrakce stěn ze stromu

Popíšeme si extrakci stěn přímo z BSP stromu nezávisle na původních stěnách z [3]. Stejný způsob byl vybrán ve vzorové metodě [1]. (Existuje i alternativní způsob, který konstruuje nové stěny ze stěn původně vložených [4, 3]. Jeho nevýhodou

Algoritmus 3 Rozklad polygonu p v podstromu uzlu u na množinu fragmentů s přidaným označením IN/OUT

1: **if** p je prázdný **then**
2: **return** $\emptyset$
3: **end if**
4: **if** u je list označený IN **then**
5: **return** $\{(p, IN)\}$, tj. polygon s přidaným označením
6: **else if** u je list označený OUT **then**
7: **return** $\{(p, OUT)\}$, tj. polygon s přidaným označením
8: **else**
9: Rozdělíme p na fragment p^+ před nadrovinou uzlu u a na fragment p^- za ní (viz 3.1.4).
10: Algoritmem 3 rozložíme p^+ v kladném synovském podstromu uzlu u, čímž získáme množinu fragmentů P^+ s přidaným označením.
11: Algoritmem 3 rozložíme p^- v záporném synovském podstromu uzlu u, čímž získáme množinu fragmentů P^- s přidaným označením.
12: **return** $P^+ \cup P^-$
13: **end if**

Algoritmus 4 Extrakce stěn z vnitřního uzlu u stromu

1: Získáme polygonální reprezentaci p nadroviny uzlu u (viz 3.1.3).
2: Ořezáme p nadrovinami (viz 3.1.4), které definují region uzlu u, čímž získáme polygonální reprezentaci nadroviny uzlu uvnitř jeho regionu.
3: Rozložíme p v záporném synovském podstromu uzlu u algoritmem 3, čímž získáme jeho fragmenty P s prvním označením IN/OUT.
4: $R := \emptyset$
5: **for all** $q \in P$ **do**
6: Rozložíme q v kladném synovském podstromu uzlu u algoritmem 3, čímž získáme jeho fragmenty Q s druhým označením IN/OUT.
7: $R := R \cup Q$
8: **end for**
9: Vyřadíme z R fragmenty, u kterých se shoduje první i druhé označení, tj. IN-IN a OUT-OUT.
10: Obrátíme orientaci fragmentů z R označených OUT-IN.
11: **return** R

je, že stěny extrahované v rámci jednoho uzlu nemusí být disjunktní.)

Základní pozorování zní takto: hranici objektu (tj. stěny) reprezentovaného BSP stromem tvoří ty části dělicích nadrovin vnitřních uzlů stromu, které oddělují buňku uvnitř objektu IN na jedné straně nadroviny od buňky mimo objekt OUT na její druhé straně. Pro získání celé hranice objektu je nutné extrahovat stěny ze všech vnitřních uzlů stromu.

Abychom mohli extrahovat stěny z jednoho vnitřního uzlu, budeme potřebovat polygonální reprezentaci té části nadroviny uzlu uvnitř regionu tohoto uzlu. To tedy znamená polygonální reprezentaci celé nadroviny uzlu (viz 3.1.3) ořezat nadrovinami (viz 3.1.4), které region uzlu definují.

Získaný polygon reprezentující část nadroviny uzlu následně zpracujeme v jednom ze synovských podstromů (lze si vybrat v kterém) tohoto uzlu. Cílem takového zpracování je získání všech fragmentů polygonu, které doputovaly až do listů podstromu, rozdělených do dvou kategorií IN a OUT podle označení listů, kde fragmenty skončily. Polygon je při průchodu podstromem fragmentován stejně jako v případě vkládání stěny do stromu (podstrom však zůstává bez úprav).

Vzniklé označené fragmenty polygonu jsou pak stejným způsobem jednotlivě zpracovány v druhém synovském podstromu uzlu. Tam fragmenty získají druhé označení kategorie.

Stěny objektu pak představují jen ty fragmenty, jejichž první a druhé označení kategorie se neshoduje (tedy IN-OUT a OUT-IN). Zbývající fragmenty se shodným označením pouze oddělují buňky uvnitř (IN-IN) či vně objektu (OUT-OUT).

Fragmenty, které byly v kladném synovském podstromu označené IN a v záporném OUT, ve skutečnosti představují stěny s opačnou orientací vzhledem k nadrovině uzlu, z jejíž polygonální reprezentace vznikly. Orientaci těchto fragmentů je proto nutné obrátit.

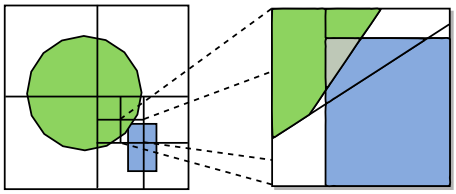
Celý postup extrakce stěn z vnitřního uzlu stromu shrnuje algoritmus 4.

3.3.4 Identifikace původních stěn

Při extrakci stěn přímo z BSP stromu bohužel ztrácíme informaci o jejich původu. Tato informace přitom může být nutná pro další fázi zpracování (např. propojení vrcholů či interpolaci dat spojených se stěnami). Představíme vlastní postup, protože v literatuře žádný nebyl nalezen. Pro identifikaci původu stěn využijeme odkazy na stěny vložené do uzlů stromu při jeho konstrukci (viz 3.3.1).

V rámci jednoho uzlu stromu budeme pro každou vloženou původní stěnu postupně z extrahovaných stěn vyřezávat části (viz 3.1.4), které jsou společné s touto původní stěnou. Úspěšným vyříznutím společné části dostaneme hledanou vazbu fragmentu extrahované stěny na původní stěnu. S neprovázanými fragmenty extrahovaných stěn pokračujeme pro další původní stěnu. Po zpracování všech původních stěn by neměly zbýt žádné fragmenty, protože všechny extrahované stěny musí být zahrnuté v původních stěnách.

Z popisu se může zdát, že tento algoritmus způsobí přílišnou fragmentaci získaných stěn. Ve skutečnosti to však není tak hrozivé. Počet stěn vložených do téhož uzlu stromu je totiž větší než jedna jen velmi vzácně. Takové stěny musí ležet na přesně shodných nadrovinách, což není příliš pravděpodobné. Pokud byla do uzlu vložena pouze jedna stěna, pak lze původ extrahovaných stěn určit snadno a jednoznačně naprosto bez fragmentace.



Obrázek 3.5: Rovinná ilustrace základního konceptu: oktantový strom je zjemněn na průniky. BSP stromy jsou vnořeny do takto zasažených buněk za účelem lokálního provedení topologických změn *(převzato z [1])*

3.4 Lokalizační schéma

Nevýhodou provádění booleovské operace pomocí algoritmů pro BSP stromy je nízký výkon při zpracování větších modelů. Jen algoritmus pro spojení dvou stromů má v nejhorším případě kvadratickou složitost vzhledem k počtu uzlů. Nejpomalejší fází se podle experimentů zdá extrakce stěn z BSP stromu. Celkově tyto algoritmy přestávají být v praxi použitelné už při zpracování modelů o velikosti v řádu 10 tisíc stěn, kdy se doba výpočtu pohybuje v řádu minut.

Tento problém s výkonem byl vyřešen v [1] zavedením lokalizačního schématu s využitím **oktantového stromu** (octree). Základní koncept zachycuje obrázek 3.5. Následující text popisuje tuto metodu z [1].

Lokalizační schéma slouží k rozmělnění jedné velké globální operace na mnoho malých lokálních operací, kde je počet naráz zpracovávaných stěn významně snížený. Tyto lokální operace se odehrávají uvnitř disjunktních konvexních prostorových buněk, jejichž sjednocení zahrnuje všechny potenciální křivky průniku vstupních modelů. Takovéto buňky se nazývají **kritické buňky**.

Hranice buněk by neměly procházet vrcholy vstupních modelů, jinak by totiž mohlo docházet k nežádoucím zvláštním případům, které by bylo nutné ošetřit. Část 3.4.1 se zabývá tím, jak toho docílit.

Zkonstruované kritické buňky by ideálně měly obsahovat přibližně konstantní počet vstupních stěn. Operace provedená v rámci jedné buňky pak trvá konstantní dobu. Tím lze v mnoha případech globální časovou složitost metody snížit na $O(\sqrt{n})$ (kde n je celkový počet vstupních stěn) plus čas potřebný pro konstrukci kritických buněk a závěrečného spojení lokálních výsledků, protože počet kritických buněk u polygonálních modelů s rovnoměrným rozlišením obvykle bývá

$O(\sqrt{n})$ [1].

Zvolený způsob lokalizace pomocí oktantového stromu se řadí spíše mezi jednodušší techniky. Počet takto zkonstruovaných kritických buněk proto nemusí být nejmenší možný. Nicméně použití pokročilejších technik (např. flexibilnější k-dimenzionální strom) by bylo spíše příliš drahé, aniž by to bylo převáženo zlepšením v počtu buněk [1].

Na začátku je jediná buňka (osově zarovnaný kvádr), která svými hranicemi obklopuje oba vstupní modely a obsahuje všechny jejich stěny. Oktantový strom má tehdy jen jediný uzel. Další zpracování probíhá rekurzivně, kdy dochází k dělení buněk, dokud obsahují stěny z obou vstupních modelů (tedy potenciálně obsahují protínající se stěny) v celkovém počtu převyšujícím jediný parametr metody m. Při dělení se buňka rozpadne na osm (téměř) symetrických osově zarovnaných kvádrů (oktantů).

Parametr m ovlivňuje počet takto zkonstruovaných kritických buněk. Lze s ním tedy škálovat výkon celé metody. Na jedné straně by parametr měl vyjadřovat co nejnižší počet kvůli časové složitosti algoritmů, které proběhnou v rámci kritických buněk (viz 3.3). Na straně druhé však příliš nízké hodnoty povedou k vzniku příliš velkého množství kritických buněk a nadměrné režii s tím spojené. Jako optimální hodnota vedoucí k minimální době výpočtu se podle většiny experimentů provedených v [1] jeví $m = 17$.

Odkazy na stěny obsažené v buňce jsou uložené v příslušném uzlu oktantového stromu. Při dělení buňky jsou odkazy předány nově vzniklým synovským uzlům. Z předaných stěn jsou však vybrány pouze ty, které novou buňku potenciálně protínají. Tento test průniku se z důvodu efektivity provádí velmi jednoduše jako test průniku dvou kvádrů (každá stěna je obalena osově zarovnaným kvádrem). Test vždy spolehlivě odhalí skutečný průnik, ale může tak někdy nesprávně vyhodnotit i neskutečný průnik. Takové selhání testu však korektnost metody neovlivní.

Po dokončení dělení stromu nás budou zajímat kritické buňky v listech stromu, tedy buňky obsahující potenciální průniky vstupních modelů. Stěny, které jsou obsažené v těchto kritických buňkách, je třeba rozdělit na části uvnitř těchto buněk a mimo ně. Nejprve tyto stěny převedeme z reprezentace založené na

bodech do reprezentace založené na rovinách (viz 3.2.3), kde bude rozdělování probíhat.

Stěny obsažené v každé kritické buňce ořezáme jejími hraničními rovinami (viz 3.1.4) a takto ořezané stěny nazveme **kritické stěny** buňky. Přitom se může stát, že z některých stěn po ořezání nic nezbude. Takovéto stěny z buňky vyřadíme, protože ji ve skutečnosti vůbec neprotínají. Po dokončení ořezávání znovu vyhodnotíme, zda je buňka nadále kritická, tj. zda obsahuje stěny nejen z jednoho, ale z obou modelů (tedy potenciální průnik). Tím potenciálně klesne celkový počet kritických buněk a zvýší se efektivita metody.

Stěny, které zůstaly obsažené v kritických buňkách, také omezíme do nekritického regionu. Každou z těchto stěn postupně ořezáme na fragmenty vně každé kritické buňky, do které zasahuje. Takto získané stěny nazveme **polokritické stěny**.

V každé kritické buňce izolovaně provedeme nad jejími kritickými stěnami (ořezanými na její vnitřek) jednu lokální booleovskou operaci s využitím BSP stromů (viz 3.3). Provádění těchto lokálních operací lze díky izolovanosti kritických stěn různých buněk paralelizovat.

Výsledný model začneme skládat ze stěn, které vznikly provedením booleovské operace nad kritickými stěnami uvnitř kritických buněk. Dále k nim přidáme polokritické stěny. Oba tyto druhy stěn převedeme zpět do reprezentace založené na bodech (viz 3.2.4). A nakonec ještě doplníme zbývající stěny vstupních modelů (v reprezentaci založené na bodech), které vůbec nebyly obsažené v kritických buňkách.

3.4.1 Dělicí bod oktantového stromu

Buňky oktantového stromu je nutné umisťovat tak, aby na nich neležely žádné vrcholy vstupních modelů, protože se tím zabrání vzniku nežádoucích zvláštních případů (např. stěny ležící přesně v hranici buněk by se mohly dostat do výsledku, i když by tam nepatřily).

Bylo nutné vymyslet vlastní postup, protože v [1] není řešení tohoto problému uvedeno. Využijeme toho, že vstupní modely musí být při předzpracování kvantizovány na nějakou přesnost, která je nižší než přesnost vnitřně použitelného

nativního číselného typu (viz 3.2.2). Souřadnicím vrcholů modelů se pak můžeme vyhnout, pokud zajistíme, aby souřadnice dělicího bodu obsahovaly nenulovou číslici těsně pod nejméně významnou číslicí souřadnic modelů. Dělicí bod tedy volíme přibližně uprostřed buňky s případným mírným posunutím jednotlivých souřadnic o přidanou číslici.

Z postupu vyplývá omezení na řádový rozdíl velikosti vstupních modelů, aby byla přidávaná číslice reprezentovatelná. V případě vstupních modelů kvantizovaných na plnou jednoduchou přesností (24 číslic) a vnitřního typu pro vypočtené souřadnice s dvojitou přesností (53 číslic) smí být řádový rozdíl nejvýše $2^{53-24-1} = 2^{28} \approx 10^8$. Takto velký rozdíl nepředstavuje pro praktické použití významnou překážku.

3.5 Dokončení výsledku

Výstup z lokalizačního schématu však ještě neodpovídá konečnému výsledku. Chybí v něm propojení sdílených vrcholů, některé jeho povrchy mohou být špatně orientované nebo dokonce nežádoucí atd. Pro získání konečného výsledku musí být tyto skutečnosti napraveny. Tím se zabývá tato část.

3.5.1 Propojení sdílených vrcholů

Stěny extrahované z BSP stromů uvnitř kritických buněk postrádají informaci o vzájemném propojení vrcholů (konektivitě). Tyto údaje musíme doplnit speciálním postupem. Popíšeme dva způsoby – jeden z literatury a druhý vlastní.

První způsob popsaný v [1] je založený na přesném testu shodnosti dvou vrcholů pomocí predikátu orientace (tj. zda vrchol určený třemi rovinami leží přesně na jiných třech rovinách, které definují druhý vrchol). Uvnitř kritických buněk je vzhledem k předpokládanému malému počtu vrcholů proveden test každého vrcholu s každým. Mimo kritické buňky se s ohledem na výkon takto nepostupuje. Využívají se odkazy na původní stěny získané při extrakci z BSP stromu (viz 3.3.4) a navzájem testují se pouze vrcholy v rámci stěn, které mají stejnou původní stěnu. Na původní stěnu se lze omezit díky zvláštním požadavkům na dělení oktantového stromu (viz 3.4.1).

Druhý způsob výpočetně náročné predikáty orientace nepoužívá. Dva vrcholy se považují za shodné, pokud se shodují jejich souřadnice vypočtené s omezenou přesností nativního číselného typu. Při tomto způsobu je velmi důležité, aby vypočtené souřadnice identických vrcholů (avšak reprezentovaných různými trojicemi rovin) byly vypočteny a zaokrouhleny vždy na totožnou hodnotu. Určování shodných vrcholů lze urychlit použitím asociativního pole a operaci provádět např. v logaritmickém či konstantním čase.

Druhý způsob není tak přesný jako první v tom ohledu, že velmi blízké vrcholy může považovat za identické. (U prvního je míra posuzované přesnosti souřadnic nekonečná, zatímco u druhého konečná.) Nicméně takovéto blízké vrcholy po výstupní kvantizaci stejně získají totožné souřadnice. Hrany a stěny tvořené takovými vrcholy pak budou degenerované, a proto je bude žádoucí odstranit. V tomto ohledu se zdá druhý způsob praktičtější, protože takovýmto vrcholům bude přiřazen stejný index.

Během následné implementace budou vyzkoušeny oba způsoby a bude vybrána efektivnější varianta (viz 4.2.5).

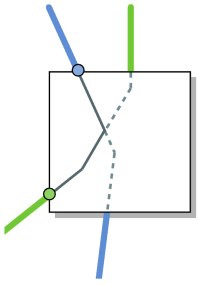
3.5.2 Orientace stěn mimo kritické buňky

Stěny extrahované z BSP stromů v kritických buňkách po provedení booleovské operace jsou orientované správně. Pro stěny mimo kritické buňky to však platit nemusí. K změně orientace dochází při operaci rozdíl. Řešení spočívá v převrácení stěn mimo kritické buňky, které pochází z odčítaného modelu.

3.5.3 Nežádoucí povrchy

Nepříjemným důsledkem použití lokalizačního schématu je skutečnost, že v nekritickém regionu mohou vzniknout otevřené povrchy, které nepatří do konečného výsledku (viz obr. 3.6). Chybějící části, které způsobily odpojení těchto povrchů, se původně nacházely v kritických buňkách a byly odstraněny provedením booleovské operace uvnitř buněk. Za hranice kritických buněk se tato změna pochopitelně nerozšířila.

Opět bylo nutné vymyslet vlastní řešení, protože v [1] není popsáno. Nejprve rozdělme stěny vstupních modelů do souvislých komponent na základě sdílených



Obrázek 3.6: Nežádoucí povrchy uvnitř kritické buňky byly náležitě odstraněny lokálním provedením booleovské operace. Odpojené části mimo buňku však přetrvaly *(převzato z [1])*.

vrcholů. Tedy pokud dvě stěny z jednoho modelu sdílí společný vrchol, pak patří do stejné komponenty. Jako **kritické komponenty** označíme ty, jejichž některé stěny jsou obsažené v kritických buňkách.

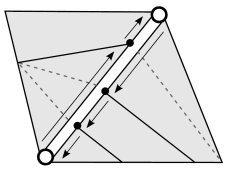
Nyní si všimněme, že vzniklé nežádoucí povrchy tvoří pouze stěny z kritických komponent modelů mimo kritické stěny. Problém lze vyřešit odstraněním těch stěn, které původně ve vstupních modelech patřily do kritických komponent, ale po provedení booleovské operace v spojeném modelu už do kritických komponent nenáleží. Tyto stěny se tedy oddělily do vlastních komponent, v kterých nejsou kritické stěny obsažené. Po odstranění povrchů v modelu zbudou nepoužité vrcholy.

3.5.4 Neprotínající se komponenty

Lokalizační schéma pracuje pouze s komponentami objektů, které se protínají. Komponenty, které se neprotínají, jsou naprosto ignorovány, jako kdyby se v modelech vůbec nevyskytovaly.

Správně by tyto komponenty ve výsledku měly buď být beze změny, s převrácenou orientací stěn, anebo nebýt vůbec, přičemž záleží na jejich vnoření do jiných komponent a na typu operace. Pro určení vnoření komponent je v [1] navrženo použít metodu vrhání paprsku. Vysvětlení konkrétního postupu v rámci použitého specifického řešení však chybí.

Pro jednoduchost lze zjišťování vnoření opomenout a předpokládat situaci, která dává pro použitou operaci větší smysl. Např. při operaci rozdíl se dá spí-



Obrázek 3.7: Cyklus topologicky otevřených hran s T-vrcholy *(převzato z [1])*

še předpokládat, že komponenty druhého objektu jsou vnořené; jinak by totiž výsledku operace odpovídal přímo první objekt.

Plnohodnotné řešení tohoto problému zůstává vzhledem k jeho nízké významnosti zatím otevřené.

3.5.5 T-vrcholy

Výsledný model, který vznikl po aplikaci této metody, bude velmi pravděpodobně obsahovat tzv. **T-vrcholy**. T-vrchol si lze představit jako nesymetrické rozdělení dvojice opačně orientovaných hran, kdy tento vrchol dělí pouze jednu z hran místo obou. Tyto vrcholy v modelu vznikají zejména v důsledku lokalizace oktantovým stromem, ale i použití BSP stromů.

Složitý problém T-vrcholů je v [1] rozebraný bohužel jen povrchně v několika větách. Jejich hlavní myšlenky jsou shrnuty ve zbytku tohoto odstavce. T-vrcholy lze bez ovlivnění geometrie opravit zavedením dodatečného degenerovaného rohu s úhlem 180 stupňů na příslušné místo do protilehlé stěny. Tyto vrcholy jsou obsažené v cyklech topologicky otevřených orientovaných hran delších než dvě hrany, přičemž hrany v každém cyklu jsou kolineární, tj. leží na jediné přímce (viz obr. 3.7). Pro detekci kolineárních hran je použita znegovaná forma predikátu platnosti bodu (viz 3.1.1), která určuje, zda se tři roviny procházející společným bodem protínají ve společné přímce. Relativní pořadí T-vrcholů na obou stranách přímky je určováno pomocí predikátu orientace (viz 3.1.1). Pro každou hranu cyklu jsou sebrány a seřazeny protější T-vrcholy, které do ní mají být vloženy.

Na základě těchto myšlenek navrhneme konkrétní podobu algoritmu. Budeme se zabývat i modely, které nejsou manifold.

Nejprve určíme výběr orientovaných hran modelu, v kterém budeme cykly hledat. U každé hrany si zapamatujeme stěnu, kam patří, abychom do ní později mohli vkládat opravné vrcholy. Víme-li, že je model manifold, pak lze výběr

omezit jen na hrany, ke kterým v modelu neexistují opačně orientované hrany.

V obecném případě, kdy model nemusí být manifold, do výběru zařadíme hrany ze všech stěn kritických komponent, tedy potenciálně i hrany stěn, které nebyly lokalizačním schématem fragmentovány. Takové stěny je nutné převést do reprezentace založené na rovinách. Při sestavování výběru hran se však vyhneme vkládání duplicitních identických hran, protože duplicita hran by mohla vést k exponenciální časové složitosti hledání cyklů.

Orientované cykly hran budeme detekovat metodou zpětného vyhledávání (backtracking). Jako první hranu nově tvořeného cyklu vezmeme libovolnou hranu dosud nepoužitou v žádném nalezeném cyklu. Dále se k první hraně budeme snažit přidat další hrany, abychom utvořili orientovaný cyklus s více než dvěma hranami, který začíná i končí v tomtéž vrcholu. V úvahu přichází hrany, které začínají přesně v (zatím) posledním vrcholu tvořeného cyklu a jsou přesně kolineární (rovnoběžné) se stávajícími. Taková hrana se však nesmí vyskytovat v právě tvořeném cyklu ani v žádných dosud nalezených.

Používáme-li však nepřesnou identifikaci shodných vrcholů s omezenou přesností (viz 3.5.1), musíme absolutní totožnost vrcholů se shodnými indexy ještě ověřit predikátem orientace. Při tomto přístupu je také třeba počítat s tím, že se v cyklech mohou vyskytovat i hrany, které se jeví jako zdegenerované do bodu (počáteční i koncový vrchol má stejný index).

Časová složitost zpětného vyhledávání je sice obecně exponenciální, nicméně díky omezení na kolineární hrany a vyloučení duplicitních hran bude složitost pouze lineární vzhledem k počtu hran a kvadratická k délce každé cesty složené z kolineárních hran.

Nyní nastává fáze oprav nalezených cyklů. Postupně zpracujeme každý cyklus. Pro každou hranu daného cyklu najdeme protější T-vrcholy, které je do hrany třeba vložit. Těmito vrcholy jsou myšleny všechny vrcholy zpracovávaného cyklu, které leží uvnitř mezi koncovými vrcholy dané hrany. Při vkládání vrcholu do hrany bychom si měli dát pozor na to, že tato hrana mohla být v příslušné stěně mezitím rozdělena jinými T-vrcholy.

Pro správné ošetření objektů, které nejsou manifold, musíme protější vrcholy vkládat nejen do dané hrany cyklu, ale i do příslušné opačně orientované hrany

(existuje-li) a do případných hran k nim duplicitních, které se do výběru nedostaly.

Pro určení relativního pořadí T-vrcholů na přímce cyklu je v [1] navrženo použít predikát orientace. Používáme-li však nepřesnou identifikaci shodných vrcholů s omezenou přesností (viz 3.5.1), můžeme porovnávat přímo vypočtené souřadnice vrcholů.

Stěny výsledného modelu je nakonec vhodné triangulovat, aby se předešlo možným problémům při dalším zpracování. Vložením opravných vrcholů totiž stěny přestanou být ryze konvexní. Teoreticky jsou sice stále konvexní, ale prakticky po provedení výstupní kvantizace mohou některé stěny být dokonce mírně konkávní.

3.6 Interpolace dat přiřazených k vrcholům stěn

Pouhé provedení booleovské operace tak, že stěny výsledného modelu obsahují jen souřadnice vrcholů, většinou není pro použití v praxi dostačující. Modely totiž často obsahují ještě další informace jako např. barvu, souřadnice textur či normály. Tato data by proto neměla být ignorována, ale vhodně přenesena (interpolována) do výsledku. Řešení tohoto problému je nad rámec vzorové metody [1], proto použijeme vlastní způsob lineární interpolace dat.

Abychom mohli data interpolovat, potřebujeme znát původ výsledných stěn, tj. mít u každé výsledné stěny odkaz na odpovídající stěnu vstupních modelů (viz 3.3.4).

Využijeme toho, že stěny vstupních modelů musely být při předzpracování rozděleny na trojúhelníky (viz 3.2.1). Interpolace přiřazených dat z vzorového trojúhelníku je snazší než v případě polygonu. Omezení na trojúhelník neplatí pro výslednou interpolovanou stěnu, která může být polygonální.

Data pro každý vrchol budeme interpolovat lineárně. Nejprve parametricky vyjádříme souřadnice daného vrcholu vzhledem k vrcholům původního trojúhelníku. Potom pomocí zjištěných parametrů a dat vrcholů tohoto trojúhelníku vypočteme interpolovanou hodnotu pro daný vrchol.

Následující rovnice zachycuje souřadnice bodu $\vec{p}$ v rovině trojúhelníku se sou-

řadnicemi vrcholů $\vec{p_0}, \vec{p_1}, \vec{p_2}$ v závislosti na parametrech t_1, t_2:

$$\vec{p} = \vec{p_0} + t_1 \cdot (\vec{p_1} - \vec{p_0}) + t_2 \cdot (\vec{p_2} - \vec{p_0})$$

Z rovnice chceme získat parametrické vyjádření t_1, t_2 známého bodu $\vec{p}$ (vrchol interpolované stěny) vzhledem k známým vrcholům $\vec{p_0}, \vec{p_1}, \vec{p_2}$ vzorového trojúhelníku. Protože se nacházíme v trojrozměrnému prostoru, dostáváme z předchozí vektorové rovnice jednu obyčejnou rovnici pro každou ze souřadnic x, y, z – celkem tři rovnice o dvou neznámých t_1, t_2. Můžeme si tedy vybrat, které dvě použijeme pro výpočet neznámých. To si lze představit tak, že stěnu promítneme do jedné z rovin xy, xz, yz. Abychom se vyhnuli dělení nulou a zajistili dobrou numerickou stabilitu, vyloučíme rovnici s dominantní souřadnicí normálového vektoru vzorového trojúhelníku. Např. je-li dominantní souřadnicí x, pracujeme pouze s rovnicemi pro souřadnice y a z.

Pro hledanou interpolaci dat $\vec{d}$ pro vrchol $\vec{p}$ platí obdoba předchozí rovnice:

$$\vec{d} = \vec{d_0} + t_1 \cdot (\vec{d_1} - \vec{d_0}) + t_2 \cdot (\vec{d_2} - \vec{d_0}),$$

přičemž $\vec{d_0}, \vec{d_1}, \vec{d_2}$ jsou n-složkové vektory dat popořadě pro vrcholy $\vec{p_0}, \vec{p_1}, \vec{p_2}$.

Má-li interpolovaná stěna převrácenou orientaci vzhledem k původní stěně (např. u operace rozdíl), může být nutné některé typy interpolovaných dat dále upravit (např. normálové vektory).

Při interpolačních výpočtech nemusí být na rozdíl od booleovských operací nutné používat přesnou aritmetiku s libovolnou přesností. Na základě vypočtených hodnot se totiž neprovádí žádné rozhodnutí. Výpočty ani nijak neovlivňují geometrii či topologii modelu. Pro tato data obvykle postačuje obyčejná přesnost.

3.7 Shrnutí základních kroků metody

Jednotlivé fáze celé metody navržené v této kapitole na základě metody [1] lze shrnout do těchto bodů:

1. triangulace stěn vstupních modelů
2. kvantizace vstupních modelů (viz 3.2.2)

3. aplikace lokalizačního schématu (viz 3.4)

   (a) zjemnění oktantového stromu na oblasti potenciálního průniku vstupních modelů

   (b) identifikace kritických buněk stromu a stěn, které jsou v nich skutečně obsažené

   (c) konverze těchto stěn do reprezentace založené na rovinách (viz 3.2.3)

   (d) rozdělení těchto stěn na části uvnitř kritických buněk (kritické stěny) a mimo ně (polokritické stěny) (viz 3.1.4)

   (e) lokální provedení booleovské operace nad kritickými stěnami v každé kritické buňce pomocí BSP stromů (viz 3.3)

   (f) konverze polokritických stěn a stěn vzniklých z kritických stěn v předchozím bodě zpět do reprezentace založené na bodech (viz 3.2.4)

   (g) vložení těchto zkonvertovaných stěn a zbývajících stěn vstupních modelů, které nebyly obsažené v kritických buňkách, do výsledného modelu

4. dokončení výsledku (viz 3.5)

   (a) propojení sdílených vrcholů (viz 3.5.1) – tento krok je (raději než samostatně) vhodné provádět současně s konverzí stěn do reprezentace založené na bodech a vkládáním stěn do výsledného modelu

   (b) zajištění správné orientace stěn mimo kritické buňky (viz 3.5.2)

   (c) odstranění nežádoucích odpojených povrchů, které mohou být mimo kritické buňky (viz 3.5.3)

   (d) ošetření neprotínajících se komponent (viz 3.5.4)

   (e) oprava T-vrcholů (viz 3.5.5)

   (f) (volitelně) odstranění degenerovaných hran a stěn

5. interpolace dat přiřazených k vrcholům stěn (viz 3.6)

6. (volitelně) triangulace stěn

4. Implementace

Tato kapitola popisuje implementaci řešení, které bylo navrženo v předchozí kapitole. Implementace byla vytvářena ve vývojovém prostředí Microsoft Visual C++ Express 2010. Vzniklé řešení se jmenuje RazeCSG.

4.1 Struktura řešení

Řešení se skládá z projektů `core` a `merge`. Jejich zdrojové a projektové soubory se nachází ve stejnojmenných podadresářích adresáře `src`. Soubory převzaté z vnějších zdrojů jsou vyčleněné do adresáře `src/external`.

Projekty jsou nastavené tak, aby jejich zdrojové soubory byly oddělené od souborů generovaných překladačem. Proto se generované soubory po překladu budou nalézat ve vlastních adresářích (na stejné úrovni jako adresář `src`): `obj` (přechodné soubory) a `bin` (výsledné soubory s přeložený kódem).

Převážnou část implementace tvoří projekt `core`. Jedná se o výpočetní jádro ve formě statické knihovny. Knihovnou se zabývá část 4.2.

Na této knihovně závisí projekt `merge`, který se skládá z jediného stejnojmenného modulu a představuje velmi jednoduchou nadstavbu knihovny v podobě konzolové aplikace. Tato aplikace provádí booleovské operace na dvěma polygonálními modely ve formátu Wavefront OBJ [12] s využitím knihovny `core`. Knihovna podporuje pouze podmnožinu specifikace tohoto formátu, která postačuje pro práci s polygonálními modely. Přiřazení materiálu k stěnám modelu a souřadnic textur a normál k jejich vrcholům je rovněž podporováno a přenášeno do výsledku.

4.2 Knihovna core

Projekt `core` se skládá z 10 modulů, přičemž první popisovaný je převzatý z vnějšího zdroje a ostatní jsou autorovým původní dílem. Následuje popis modulů.

4.2.1 Převzatý modul adaptive_precision_arithmetic.cpp

Převzatý modul `adaptive_precision_arithmetic.cpp` se nachází odděleně od ostatních zdrojových souborů v adresáři `src/external/shewchuk`. Tento modul pochází z veřejně dostupné knihovny pro adaptivní aritmetiku s libovolnou přesností, která je součástí práce [5]. Původní knihovna byla upravena pro potřeby této práce. Byla přizpůsobena jazyku C++, byly z ní odstraněny nepotřebné funkce a byla do ní přidána funkce `expansion_division_estimate` pro dělení, jejíž vstupem jsou čísla s rozšířenou přesností a výstupem číslo s přesností číselného typu `double`. Tato funkce je nutná pro přesný výpočet průsečíku tří rovin (pomocí Cramerova pravidla).

Modul je nutné inicializovat funkcí `exactinit`, která provede potřebné nastavení řídicího slova FPU (floating-point unit) a vypočte konstanty pro určování chyb při filtrovacích technikách.

4.2.2 Modul primitives.cpp

Modul `primitives.cpp` obsahuje třídy představující základní geometrická primitiva: `Point` (bod), `Plane` (rovina), `Vector` (vektor) a `BoundingBox` (ohraničující kvádr).

Koeficienty rovnice rovin jsou uložené s nejvyšší nativní přesností, kterou vývojové prostředí nabízí, tj. s dvojitou přesností v číselném typu `double`. Pro souřadnice vrcholů je definovaný zvláštní typ `CoordFloatType`, což dovoluje snadnou záměnu. Typ aktuálně odpovídá typu `double`, aby mohly být zpracovány i modely s větším řádovým rozdílem (viz 3.4.1).

Pro úsporu paměti nejsou v datových strukturách uložené přímo koeficienty roviny, ale ukazatele na ně. Často je však potřeba k dané rovině získat ještě rovinu opačně orientovanou. Třída `Plane` proto obsahuje ukazatel na párovou rovinu. Ta je alokována, jen pokud je potřeba, a to automaticky při prvním přístupu přes metodu `getNegatedPlane`. K správnému odalokování rovin včetně párových slouží statická metoda `deleteAllocatedPlanes`.

4.2.3 Modul determinants.cpp

Modul `determinants.cpp` slouží k výpočtu determinantů a levých horních subdeterminantů matic, jejichž řádky tvoří koeficienty rovnic rovin. Funkce mají jak přesnou (s příponou `-Exact`), tak rychlou variantu (`-Fast`) a některé také filtrovanou variantu (`-Filtered`), což znamená, že znaménko výsledku je správné, ale hodnota nemusí být přesná. Byla implementována jednostupňová filtrovací technika z [5], která odhaduje chyby pomocí konstant vypočtených při inicializaci modulu `adaptive_precision_arithmetic`.

4.2.4 Modul predicates.cpp

Modul obsahuje čtyři funkce, které představují geometrické predikáty pracující s rovinami (viz 3.1.1): `coincidence` (shodnost), `coincidentOrientation` (shodná orientace), `pointValidity` (platnost bodu) a `orientation` (orientace bodu k rovině).

4.2.5 Modul point_based_model.cpp

Modul `point_based_model.cpp` obsahuje třídu `PointBasedModel`, která představuje model v reprezentaci založené na bodech. Její metoda `quantizeTriangulatedFaces` provede vstupní kvantizaci modelu na optimální přesnost. Přitom jsou nastaveny některé atributy třídy: přesnost kvantizace `coordBitPrecision` a interval $<$`lowerCoordLimit`, `upperCoordLimit`), do kterého po kvantizaci modelu spadají všechny jeho nenulové souřadnice vrcholů. Metody `readFromObjFile` a `writeToObjFile` umožňují model načíst a zapsat ve formátu Wavefront OBJ [12].

Metoda `putVertex` slouží k vkládání vrcholů, přičemž zajistí propojení sdílených vrcholů. Během vývoje byly implementovány oba způsoby propojení sdílených vrcholů popsané v části 3.5.1. Tedy plně přesný způsob, který testuje shodnost pomocí predikátu orientace, a alternativní způsob, který porovnává jen souřadnice vypočtené s přesností nativního typu.

Implementace plně přesného způsobu vykazovala znatelně nižší výkon oproti alternativě i po optimalizaci, kdy byl test výpočetně náročným predikátem orien-

tace redukován jen na případy potenciální shody vrcholů (na základě heuristiky alternativního způsobu). Vzhledem k nízkému výkonu přesného způsobu od něj bylo upuštěno a byl vybrán alternativní způsob.

Třída `PointBasedFace` představuje stěnu v reprezentaci založené na bodech. Atribute `model` je odkaz na model stěny a atribut `originalFace` odkaz na původní stěnu, který je nastaven při provádění booleovské operace u stěn výstupního modelu a ukazuje na odpovídající stěnu vstupního modelu.

4.2.6 Modul plane_based_model.cpp

Modul `plane_based_model.cpp` obsahuje třídu `PlaneBasedModel`, která symbolizuje model v reprezentaci založené na rovinách. Jeho stěnu představuje třída `PlaneBasedFace`. Její metody `convertFromPointBasedFace` a `convertToPointBasedFace` provádí konverze z reprezentace založené na bodech a zpět. Atribut `originalFace` je odkaz na původní stěnu v reprezentaci založené na bodech.

4.2.7 Modul bsp_tree_based_model.cpp

Modul `bsp_tree_based_model.cpp` obsahuje třídu `BspTreeBasedModel`, která představuje model reprezentovaný BSP stromem. Její metody `convertFromPlaneBasedModel` a `convertToPlaneBasedModel` provádí konverze z reprezentace založené na rovinách a zpět. Spojení modelů (a tedy provedení booleovské operace) provádí metoda `merge`.

Uzel BSP stromu odpovídá třídě `BspTreeNode`. Její atribut `originalFaces` obsahuje odkazy na původní stěny (v reprezentaci založené na rovinách) vložené do uzlu při vytváření stromu.

4.2.8 Modul octree_model.cpp

Modul `octree_model.cpp` obsahuje třídu `OctreeModel`, která symbolizuje lokalizační schéma v podobě oktantového stromu nad dvěma objekty typu `PointBasedModel`. Třída `OctreeNode` definuje uzel stromu a třída `OctreeCell` jeho buňku. Modul zavádí konstantu kritického počtu stěn `CRITICAL_FACE_COUNT`, která udává nejvyšší povolený počet stěn v buňkách s potenciálním průnikem.

K nalezení vhodného dělicího bodu buněk slouží metoda `OctreeModel::computeDivisionPoint`. Metoda `OctreeCell::merge` provádí jednu lokální booleovskou operaci v rámci buňky.

4.2.9 Modul boolean_operation_context.cpp

Modul `boolean_operation_context.cpp` řídí provádění celé booleovské operace nad dvěma triangulovanými a kvantizovanými modely typu `PointBasedModel` s využitím výše uvedených modulů. Provádění operace je spojené s mnoha daty, která jsou zabalená do třídy `BooleanOperationContext`, aby mohla být snadno sdílena mezi jednotlivými částmi modulu. Tím je umožněno různé instance třídy použít pro paralelní provedení operací nad různými modely. Řízení celé operace probíhá v její metodě `merge`.

4.2.10 Modul razecsg.cpp

Modul `razecsg.cpp` představuje rozhraní knihovny `core`. Před prvním použitím je nutné knihovnu inicializovat funkcí `initialize`. Booleovské operace nad modely v klasické reprezentaci založené na bodech (typ `PointBasedModel` či zde zavedený alias `Model`) pak lze provádět funkcí `mergeModels`, která modely připraví pro zpracování v nové instanci třídy `BooleanOperationContext`.

5. Výsledky

V této kapitole je provedeno experimentální srovnání vzniklé implementace RazeCSG s vybranými implementacemi používanými v praxi – knihovnou Carve [10] a příslušným nástrojem v aplikaci Autodesk Maya [11]. Na přiloženém CD lze nalézt dávkové soubory pro zopakování experimentů provedených v této kapitole a také příslušné vstupní a výstupní modely (viz příloha A).

5.1 Způsob testování

Knihovny RazeCSG a Carve byly přeloženy ve stejném prostředí Microsoft Visual C++ 2010 Express s plnými optimalizacemi a testovány v 64bitové verzi. Aplikace Maya byla testována v 32bitové verzi, protože při předběžných pokusech byla schopná zpracovat více modelů a také vykazovala lepší výkon než její 64bitová verze.

Testování probíhalo na počítači s konfigurací uvedenou v tabulce 5.1. Experimenty byly prováděny opakovaně a prezentovány jsou nejlepší naměřené hodnoty.

Náplní většiny experimentů bylo vykonání booleovské operace nad daným polygonálním modelem a modelem, který z něj vznikl otočením o pravý úhel kolem jedné souřadnicové osy posunuté do středu modelu (případně posunutím o polovinu velikosti modelu ve směru jedné souřadnicové osy). S modely bylo manipulováno v programu MeshLab [15]. Všechny vstupní i výstupní modely byly ve formátu Wavefront OBJ [12]. Testované implementace generovaly netriangulované výstupní modely.

Testování knihoven RazeCSG a Carve bylo spouštěno pomocí skriptů v dávkových souborech. Doba výpočtu byla brána přímo z výpisu měření těchto knihoven.

Počítač	Dell Studio XPS 1640
Procesor	Intel Core2 Duo P8700 @ 2,53 GHz
Operační paměť	4 GB RAM
Operační systém	Microsoft Windows 7 Ultimate (64bitový)

Tabulka 5.1: Konfigurace počítače použitého pro testování

Aplikace Maya byla testována ručně a čas byl měřen příkazem `dgtimer`. Naměřená doba výpočtu u všech testovaných programů vždy zahrnovala všechny potřebné kroky kromě diskových operací (import a export modelů). Spotřebovaná paměť byla sledována v systémovém správci úloh.

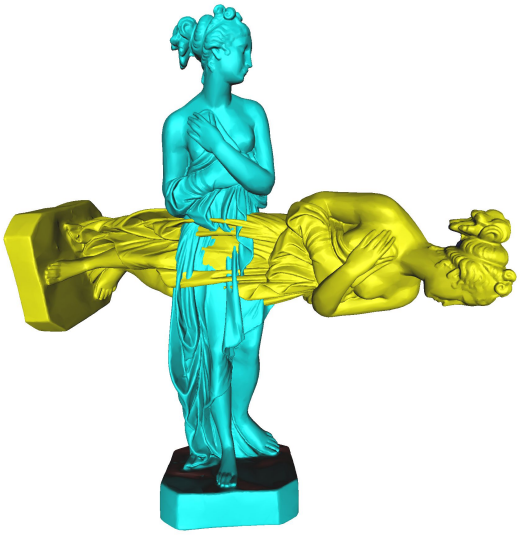
Každý výstupní model vytvořený knihovnou RazeCSG byl porovnán s příslušným výstupem srovnávaných implementací (po dodatečné triangulaci) pomocí programu Metro [16]. Tento program počítá Hausdorffovu vzdálenost dvou modelů (tj. největší vzdálenost bodu z povrchu jednoho modelu k nejbližšímu bodu z povrchu druhého modelu). Topologie výstupních modelů (počet obsažených manifoldů a jejich uzavřenost) byla kontrolována knihovnou Carve [10].

5.2 Test výkonu

Cílem první části experimentů bylo prověřit výkon implementací při práci se středně velkými až velkými modely. Testy byly provedeny s modely Ifigenie [13] o velikosti od 25 000 do 800 000 stěn při operaci sjednocení. Výsledky těchto experimentů obsahuje tab. 5.2. Výsledný model jednoho z experimentů zobrazuje obr. 5.1. Dobu vybraných fází výpočtu knihovny RazeCSG zachycuje tab. 5.3.

V další sérii experimentů byl testován výkon při operacích sjednocení, průnik a rozdíl nad modely z [14]: toroid o velikosti 573 440 stěn (viz tab. 5.4 a obr. 5.2), pásovec o velikosti 345 944 stěn (viz tab. 5.5 a obr. 5.3) a golfový míček o velikosti 245 760 stěn (viz tab. 5.6 a obr. 5.4). Poslední experiment tohoto typu proběhl s malým modelem zdeformovaného toru (ukázkový model z [15]) o velikosti 2 880 stěn pokrytým texturou. Cílem bylo zejména ukázat schopnost knihovny RazeCSG pracovat se souřadnicemi textur přiřazenými k vrcholům stěn. Viz tab. 5.7 a obr. 5.5.

Odpovídající výstupní modely srovnávaných implementací obsahovaly shodný počet manifoldů, přičemž všechny byly uzavřené (tzn. bez T-vrcholů). Maximální naměřená Hausdorffova vzdálenost odpovídajících modelů vztažená k jejich velikosti (tj. vydělená délkou diagonály obalujícího kvádru modelů) byla řádově 2^{-19}. Příčinou této malé chyby je vstupní kvantizace modelů knihovny RazeCSG.



Obrázek 5.1: Výsledný model jednoho z experimentů s modelem Ifigenie [13]

Počet stěn	25 000	50 000	100 000	200 000	400 000	800 000
Doba výpočtu v sekundách:						
RazeCSG	3,8	5,7	9,0	14,2	23,3	38,9
Carve	1,1	2,2	4,4	9,1	18,6	41,5
Maya	8,6	53,1	208,9	775,8	3056,9	–
Spotřebovaná paměť v MB:						
RazeCSG	100	173	312	572	1 081	2 069
Carve	123	234	453	892	1 757	3 358
Maya	455	557	751	979	1 625	–
Počet stěn výsledného modelu:						
RazeCSG	64 270	116 318	214 667	407 125	783 978	1 529 483
Carve	46 710	92 169	182 063	361 966	723 264	1 453 111
Maya	46 710	92 168	182 060	361 958	723 258	–

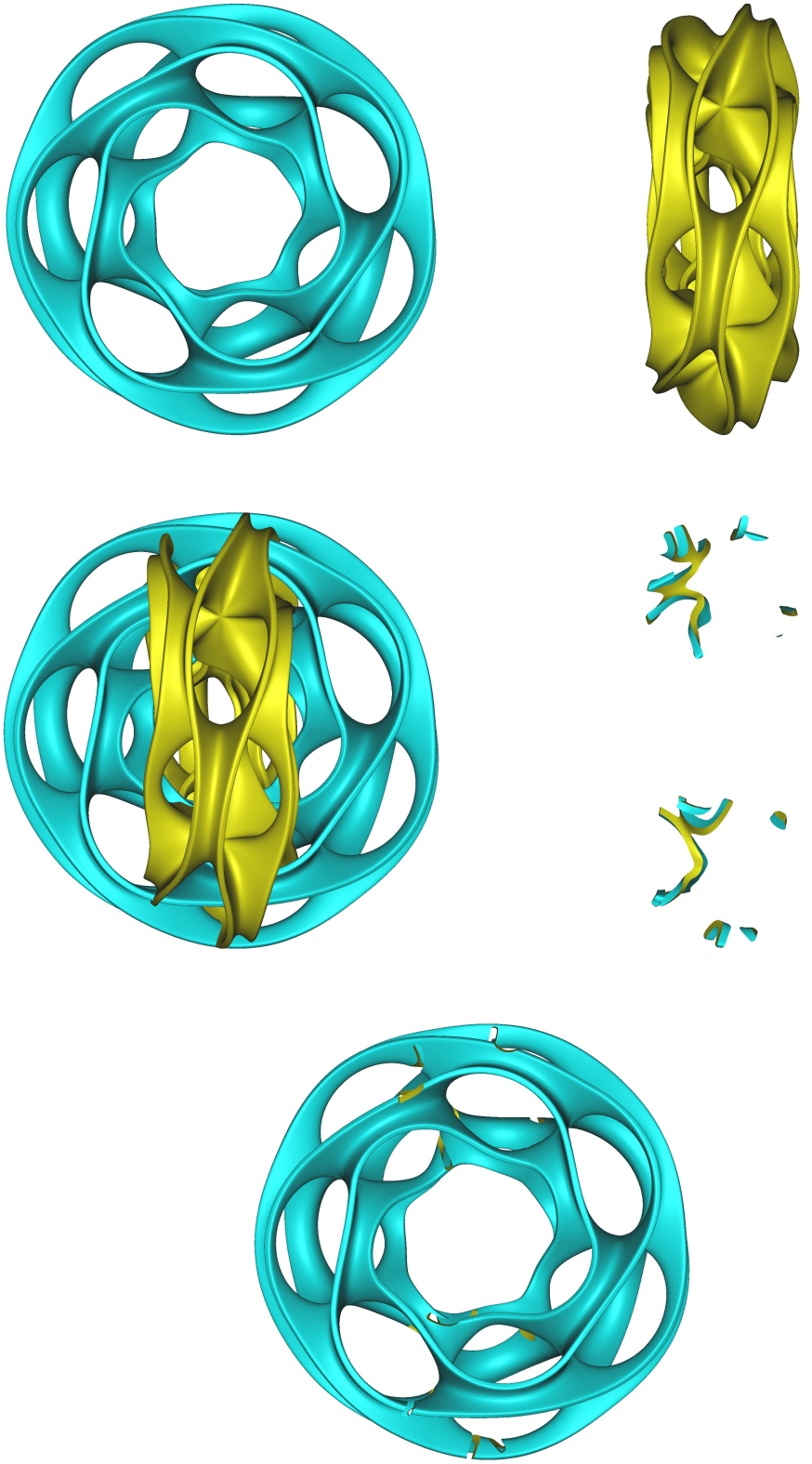
Tabulka 5.2: Výsledky experimentů s modely Ifigenie [13] o velikosti od 25 000 do 800 000 stěn. Náplní testů bylo sjednocení daného modelu s jeho otočením o pravý úhel. Srovnány jsou implementace RazeCSG, Carve a Maya. Všechny testy proběhly úspěšně s výjimkou testu s největším modelem, při kterém se zhroutila Maya.

Zjemnění oktantového stromu na průnik modelů	2,0
Identifikace kritických buněk a získání kritických stěn (včetně konverze)	0,7
Získání polokritických stěn	2,9
Provedení booleovských operací v kritických buňkách	5,3
Vložení všech stěn do výsledného modelu (včetně zpětné konverze, propojení sdílených vrcholů a registrace hran pro hledání T-vrcholů)	12,0
Odstranění nežádoucích odpojených povrchů (včetně identifikace kritických komponent modelů)	4,0
Oprava T-vrcholů	8,1

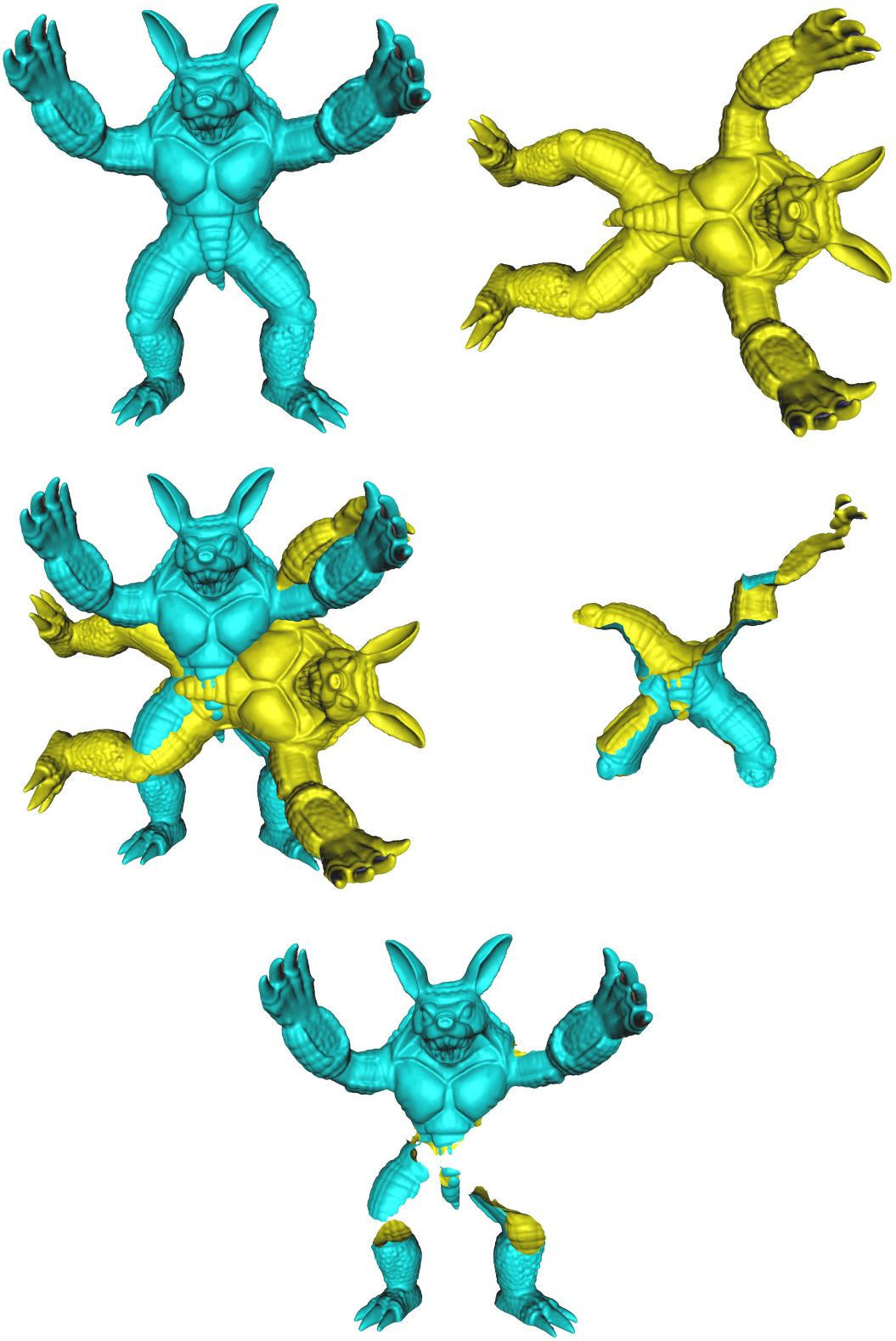
Tabulka 5.3: Doba vybraných fází výpočtu knihovny RazeCSG (viz 3.7) při experimentu s modelem Ifigenie [13] o velikosti 800 000 stěn (v sekundách)

Operace	Sjednocení	Průnik	Rozdíl
Doba výpočtu v sekundách:			
RazeCSG	36,8	33,0	34,7
Carve	24,8	16,1	20,2
Spotřebovaná paměť v MB:			
RazeCSG	1 665	1 651	1 668
Carve	2 686	1 365	2 029
Počet stěn výsledného modelu:			
RazeCSG	1 262 303	163 630	712 052
Carve	1 122 232	39 596	581 196

Tabulka 5.4: Výsledky experimentů s modelem toroidu z [14] o velikosti 573 440 stěn. Náplní testů bylo sjednocení, průnik a rozdíl modelu a jeho otočení o pravý úhel. Srovnány jsou pouze implementace RazeCSG a Carve. Maya byla vyřazena, protože skončila s chybou, že operaci nelze provést.



Obrázek 5.2: Experiment s modelem toroidu z [14]: dva vstupní modely (nahoře) a tři výstupní modely operací sjednocení, průnik (uprostřed) a rozdíl (dole)



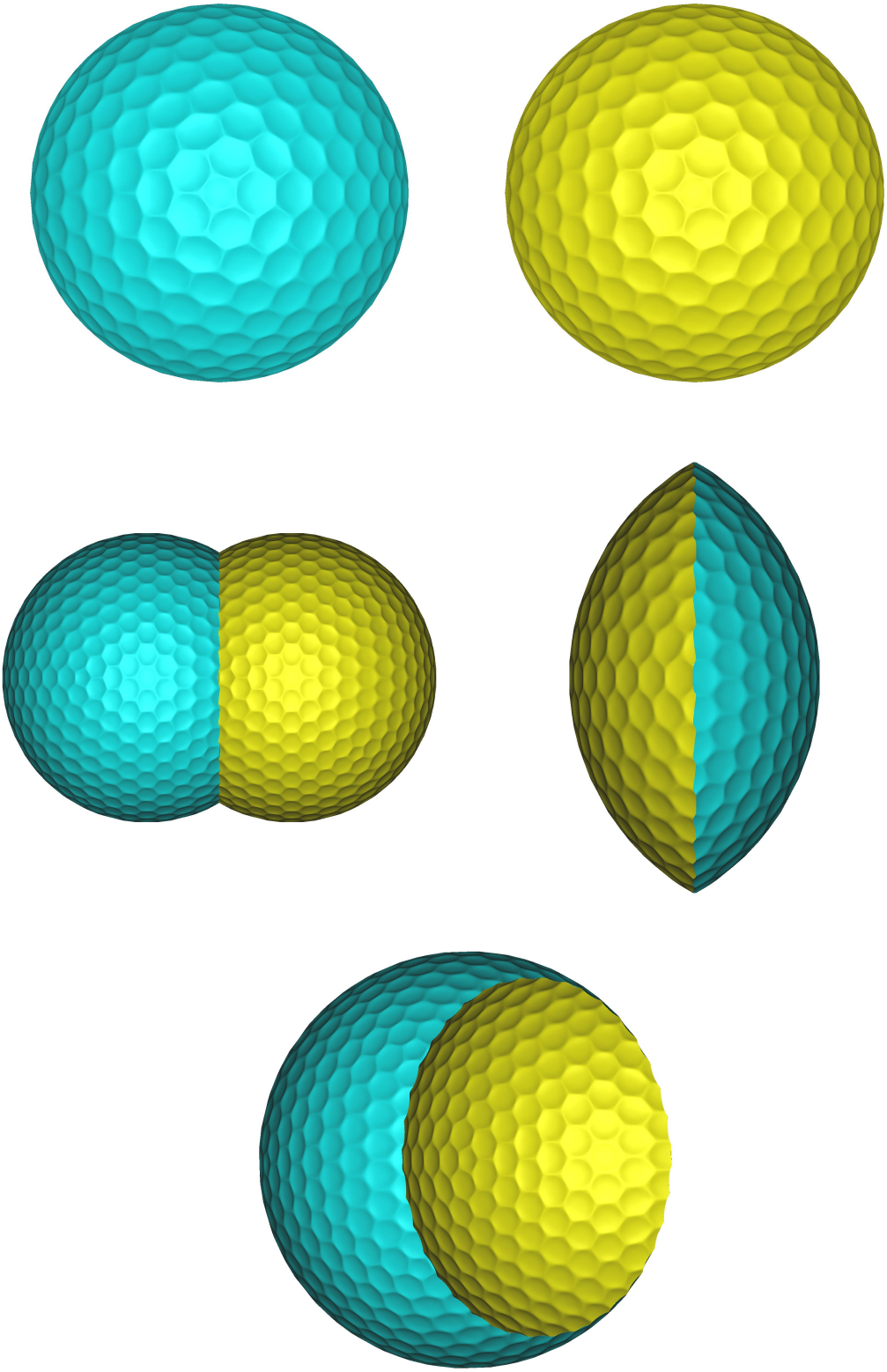
Obrázek 5.3: Experiment s modelem pásovce z [14]: dva vstupní modely (nahoře) a tři výstupní modely operací sjednocení, průnik (uprostřed) a rozdíl (dole)

Operace	Sjednocení	Průnik	Rozdíl
Doba výpočtu v sekundách:			
RazeCSG	22,3	20,6	21,3
Carve	12,9	10,5	11,4
Spotřebovaná paměť v MB:			
RazeCSG	1 003	999	1 007
Carve	1 329	938	1 122
Počet stěn výsledného modelu:			
RazeCSG	643 564	226 257	422 969
Carve	556 020	144 629	339 282

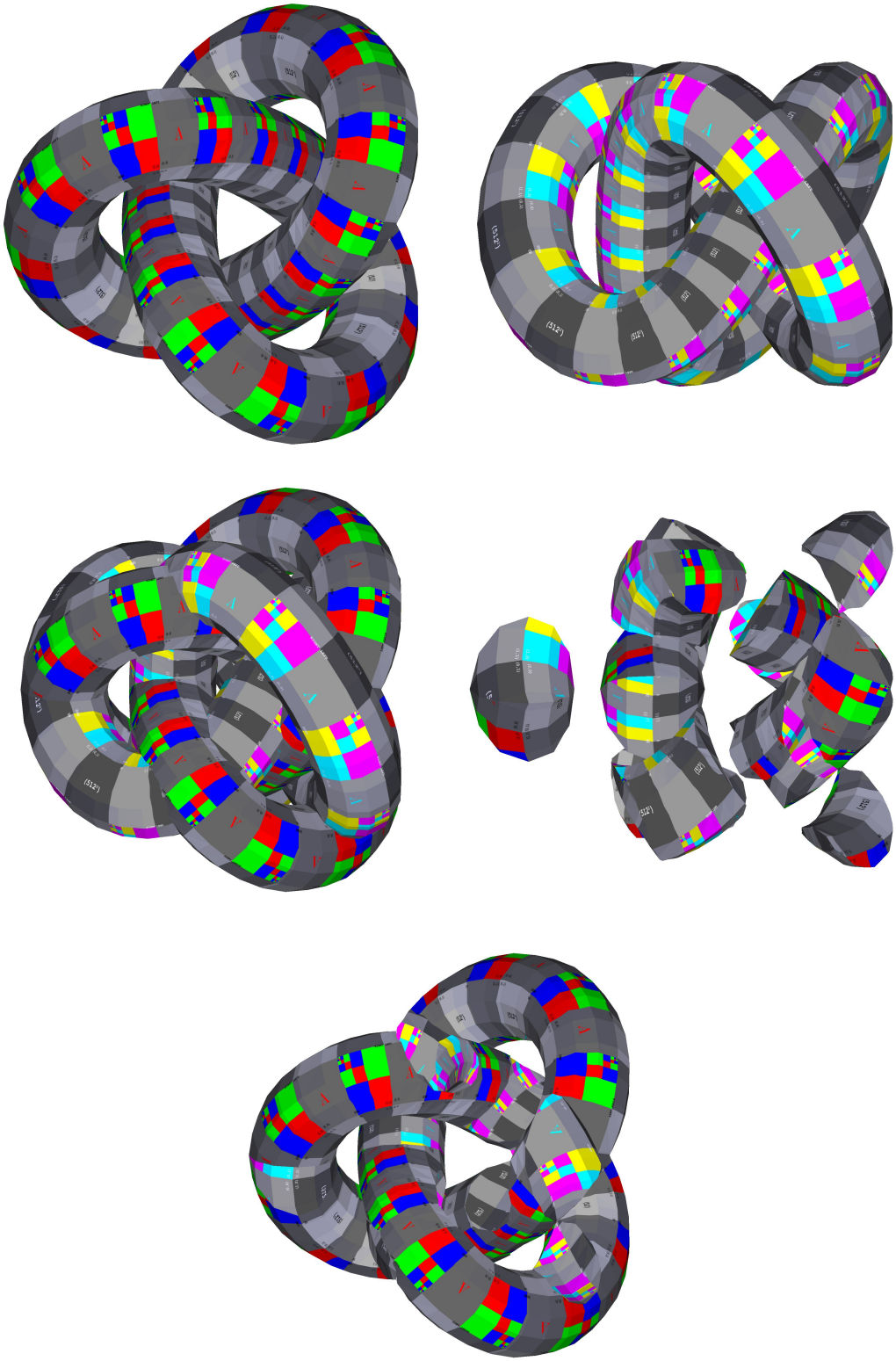
Tabulka 5.5: Výsledky experimentů s modelem pásovce z [14] o velikosti 345 944 stěn. Náplní testů bylo sjednocení, průnik a rozdíl modelu a jeho otočení o pravý úhel. Srovnány jsou pouze implementace RazeCSG a Carve. Maya byla vyřazena, protože po asi 40 minutách výpočtu vrátila prázdný výsledek.

Operace	Sjednocení	Průnik	Rozdíl
Doba výpočtu v sekundách:			
RazeCSG	8,5	8,6	8,8
Carve	7,1	5,6	6,3
Spotřebovaná paměť v MB:			
RazeCSG	637	641	641
Carve	878	646	761
Počet stěn výsledného modelu:			
RazeCSG	390 056	146 596	268 318
Carve	371 380	122 920	247 150

Tabulka 5.6: Výsledky experimentů s modelem golfového míčku z [14] o velikosti 245 760 stěn. Náplní testů bylo sjednocení, průnik a rozdíl modelu a jeho posunutí ve směru souřadnicové osy. Srovnány jsou pouze implementace RazeCSG a Carve. Maya byla vyřazena, protože vracela prázdný výsledek.



Obrázek 5.4: Experiment s modelem golfového míčku z [14]: dva vstupní modely (nahoře) a tři výstupní modely operací sjednocení, průnik (uprostřed) a rozdíl (dole)



Obrázek 5.5: Experiment s modelem zdeformovaného toru z [15]: dva vstupní modely (nahoře) a tři výstupní modely operací sjednocení, průnik (uprostřed) a rozdíl (dole)

Operace	Sjednocení	Průnik	Rozdíl
Doba výpočtu v sekundách:			
RazeCSG	2,8	2,6	2,7
Carve	0,2	0,2	0,2
Spotřebovaná paměť v MB:			
RazeCSG	45	42	45
Carve	21	18	18
Počet stěn výsledného modelu:			
RazeCSG	20 970	16 304	18 544
Carve	4 328	2 960	3 644

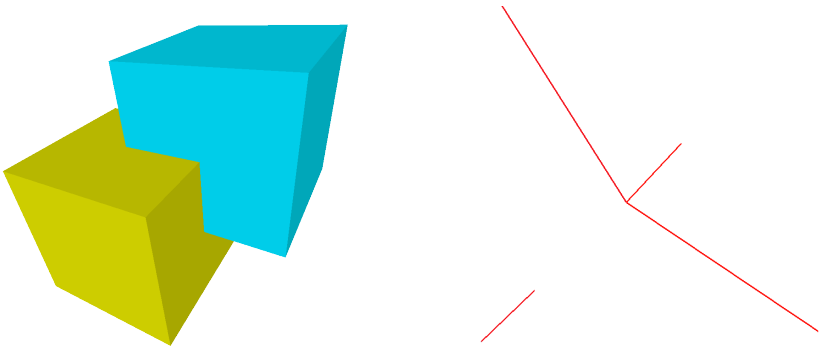
Tabulka 5.7: Výsledky experimentů s modelem zdeformovaného toru z [15] o velikosti 2 880 stěn pokrytým texturou. Náplní testů bylo sjednocení, průnik a rozdíl modelu a jeho otočení o pravý úhel s odlišnou texturu. Srovnány jsou pouze implementace RazeCSG a Carve. Maya byla vyřazena, protože vracela prázdný výsledek.

5.3 Test robustnosti

Během testů výkonu v předchozí části se prokázalo, že z testovaných implementací je nejméně robustní Maya, která nebyla schopná dopočítat výsledek u více než poloviny provedených experimentů.

Knihovny RazeCSG a Carve uspěly u všech předchozích experimentů, a tak prokázaly, že jsou významně robustnější. Knihovnu Carve však nelze považovat za plně robustní, protože pracuje pouze v rychlé a nepřesné nativní aritmetice procesoru. Chyby v rozhodování plynoucí z nepřesnosti výpočtů jsou částečně omezeny zavedením konstantní tolerance při porovnávání hodnot. Testovaná verze knihovny používá toleranci $\epsilon = 2^{-26} \approx 1,5 \cdot 10^{-8}$ (viz konstanta `DEF_EPSILON` zdrojového souboru `carve.cpp` z [10]).

Nedokonalost tohoto přístupu lze ukázat např. při libovolné booleovské operaci nad dvěma osově zarovnanými krychlemi definovanými souřadnicovými intervaly: první krychle od $[0, 0, 0]$ do $[1, 1, 1]$ a druhá od $[0 + \delta, 0 + \delta, 0 + \delta]$ do $[1 + \delta, 1 + \delta, 1 + \delta]$ pro např. $\delta = 10^{-8}$. Knihovna Carve nebyla takové operace



Obrázek 5.6: Výsledek operace sjednocení dvou krychlí s délkou hrany 2^{-26} jednotek: správný výstup knihovny RazeCSG (vlevo) a chybný výstup knihovny Carve v podobě stěn zdegenerovaných do úseček (vpravo)

schopná provést a střídavě končila s dvěma různými chybovými hlášeními. Zřejmě navíc závisí na náhodném prvku. Naopak knihovna RazeCSG při těchto operacích uspěla. Lze však oprávněně namítnout, že knihovna byla zvýhodněna, protože po vstupní kvantizaci na 24 bitů ve skutečnosti pracovala s identickými krychlemi. Pro korektní srovnání je potřeba najít jiný příklad (bez vlivu kvantizace).

Takovým příkladem jsou opět dvě osově zarovnané krychle, ale nyní definované intervaly: první od $[0, 0, 0]$ do $[2^{-26}, 2^{-26}, 2^{-26}]$ a druhá od $[-2^{-27}, -2^{-27}, -2^{-27}]$ do $[2^{-27}, 2^{-27}, 2^{-27}]$. Knihovna Carve nedokázala s těmito krychlemi správně provést žádnou booleovskou operaci. Výsledek buď obsahoval stěny zdegenerované do úseček, nebo byl prázdný. Naopak knihovna RazeCSG s nimi správně provedla libovolnou booleovskou operaci. Tyto krychle přitom tentokrát nebyly kvantizací knihovny RazeCSG vůbec nijak zasaženy (kvantizováno bylo opět na 24 bitů a souřadnice vrcholů by zůstaly beze změny i po kvantizaci na 1 bit). Viz obr. 5.6.

5.4 Analýza výsledků

Knihovny RazeCSG a Carve vykazují u velkých modelů Ifigenie podobný výkon. U největšího testovaného modelu o velikosti 800 000 stěn je RazeCSG dokonce mírně rychlejší. V rámci experimentů s dalšími velkými modely je RazeCSG v nejhorším případě jen dvakrát pomalejší než Carve. Výpočty ve vnitřní reprezentaci knihovny RazeCSG přitom probíhají na rozdíl od knihovny Carve plně

přesně a robustně.

RazeCSG ve většině experimentů spotřebuje nejméně paměti z testovaných implementací. U největšího modelu vyžaduje jen asi 60 % paměti ve srovnání s Carve.

Nevýhodou knihovny RazeCSG je, že výsledné modely mají vyšší počet stěn než výsledky ostatních implementací. Při experimentech s modely Ifigenie míra tohoto jevu klesá v závislosti na zvyšující se velikosti vstupu. U největšího modelu je výsledný počet stěn jen asi o 5 % vyšší než u knihovny Carve.

Při experimentech nejhůře obstál nástroj v aplikaci Maya. Jeho doba výpočtu roste velmi strmě vzhledem k zvyšující se velikosti vstupu. U modelu se 400 000 stěnami výpočet trval téměř hodinu, tedy asi 130krát déle než u knihovny RazeCSG. Maya navíc projevila nestabilitu, když nebyla schopná vypočítat správný výsledek u více než poloviny experimentů včetně testů s nejmenším (2 880 stěn) a největším modelem (800 000 stěn).

6. Závěr

Cílem práce bylo implementovat knihovnu, která podporuje nejpoužívanější booleovské operace nad polygonálními modely. Důraz byl přitom kladen na rychlost a stabilitu knihovny při práci s velkými objekty.

Po prostudování literatury byla vybrána metoda [1] jako vzorová, protože nejlépe vyhovovala stanoveným požadavkům. Řešení založené na této metodě bylo podrobně analyzováno a navrženo v kapitole 3. Nedostatečně rozebrané části metody v literatuře byly rozvinuty vlastními postupy (např. dělicí bod oktantového stromu či oprava T-vrcholů). K některým postupům byla také navržena z hlediska této práce potenciálně výhodnější alternativa (např. propojení sdílených vrcholů).

Stabilitu metody zaručuje použitá vnitřní reprezentace založená na rovinách a BSP stromech, v které lze operace provádět plně přesně a robustně a zároveň efektivně v pouhé aritmetice s předem známou pevnou přesností (tedy bez pomalé aritmetiky s libovolnou přesností). Výkon metody je optimalizován lokalizačním schématem, které je realizované adaptivním oktantovým stromem.

Výsledkem práce je implementace nazvaná RazeCSG, popsaná v kapitole 4. Experimenty s implementacemi provedené v kapitole 5 potvrzují, že stanovený cíl práce byl splněn. Knihovna RazeCSG vykazovala u velkých modelů v nejhorším případě jen dvakrát nižší výkon než v praxi používaná knihovna Carve, která však není plně robustní. Při experimentu s největším modelem o velikosti 800 000 stěn dokonce knihovna RazeCSG knihovnu Carve v rychlosti mírně překonala. Knihovna RazeCSG byla celkově výrazně rychlejší (u velkých modelů alespoň 130krát) než nástroj v aplikaci Autodesk Maya, který je z testovaných implementací robustní nejméně. U velkých modelů spotřebovala knihovna RazeCSG většinou znatelně méně paměti než ostatní. Knihovna také dokáže zpracovat modely, k jejichž vrcholům jsou přiřazené souřadnice textur a normál, a tato data přenést do výsledku.

6.1 Budoucí práce

6.1.1 Optimalizace výkonu

Vyhodnocování predikátů, které tvoří základ metody, je v současné implementaci optimalizováno jednoduchou jednostupňovou filtrovací technikou. Nasazení složitějšího vícestupňového filtrování z [5] by pravděpodobně přineslo další zvýšení výkonu, protože vyhodnocování predikátů trvá převážnou část celkové doby výpočtu [1].

Lokální booleovské operace jsou uvnitř kritických buněk prováděny sériově. Vzhledem k izolovanosti kritických stěn různých buněk lze tyto operace paralelizovat a získat tím vyšší výkon na víceprocesorových systémech.

6.1.2 Snížení fragmentace stěn

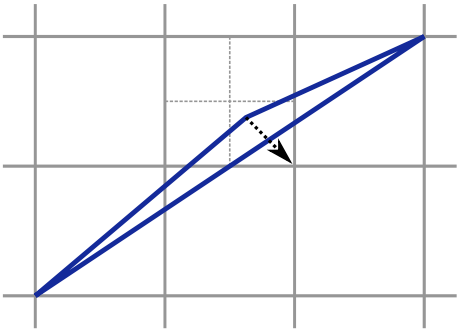
Nevýhodou této metody je velký stupeň fragmentace stěn modelu. Fragmentace je způsobena zejména používáním BSP stromů. Ačkoli je tento negativní jev silně redukován a omezen na regiony průniku lokalizačním schématem, stále může být vhodné fragmentaci dále snížit [1].

Proto je v [1] navrženo použít konkrétní proceduru na zjednodušení polygonálního modelu. Tato procedura spojuje pouze přesně koplanární stěny (ležící v jedné rovině), aniž by byla narušena přesnost výsledku. K tomu můžeme dodat, že stěny lze také spojovat v rámci jejich stejného odkazu na původní stěnu, protože takto vybrané stěny jsou jistě koplanární.

Je třeba dát pozor na to, že spojené stěny nutně nemusí být konvexní, což výrazně komplikuje jejich triangulaci.

6.1.3 Ošetření převrácení orientace stěn při kvantizaci

Po vstupní i výstupní kvantizaci modelu může kvůli zanesení geometrické chyby dojít ve vzácných případech k tomu, že model začne v mikroskopickém měřítku protínat sám sebe [1]. Častým požadavkem metod včetně této přitom je, aby vstupní model sám sebe neprotínal. Tento problém však není výlučně spjatý s touto metodou, ale může se vyskytovat u kterékoli metody, kde je výstupní model



Obrázek 6.1: Příklad převrácení orientace téměř degenerovaného trojúhelníku během kvantizace na přesnost znázorněnou mřížkou

nakonec uložen s konečnou přesností.

Příčinou problému je převrácení orientace téměř degenerovaných trojúhelníkových stěn v důsledku zaokrouhlení souřadnic vrcholů (viz obr. 6.1), pokud si stěny modelu představíme jako triangulované. Převrácení orientace stěny lze detekovat na základě záporného znaménka skalárního součinu normálových vektorů stěny před kvantizací a po ní. Výskyt problému během kvantizace tedy dokážeme snadno určit.

V rámci této metody je vzácnost výskytu tohoto problému ještě umocněna lokalizačním schématem. Převrácené trojúhelníky by se musely vyskytnout uvnitř kritických buněk, aby mohly výsledek negativně ovlivnit. Výsledný model by v takovém případě mohl obsahovat otvory a/nebo protínající se stěny.

Abychom zajistili vždy plně korektní výsledek, musíme v případě zjištění výskytu tohoto problému zajistit opravu modelu některou z řady existujících metod (např. [17] či [18]).

Seznam použité literatury

[1] Campen, M.; Kobbelt, L. Exact and Robust (Self-)Intersections for Polygonal Meshes. In *Computer Graphics Forum*. Volume 29, Number 2. [Oxford] : Blackwell Publishing, 2010. s. 397–406.

[2] Bernstein, G.; Fussell, D. Fast, Exact, Linear Booleans. In *Computer Graphics Forum*. Volume 28, Number 5. [Oxford] : Blackwell Publishing, 2009. s. 1269–1278.

[3] Naylor, B.; Amanatides, J.; Thibault, W. Merging BSP Trees Yields Polyhedral Set Operations. In *SIGGRAPH Computer Graphics*. Volume 24, Number 4. [New York] : ACM, 1990. s. 115–124.

[4] Thibault, W. C.; Naylor, B. F. Set Operations on Polyhedra Using Binary Space Partitioning Trees. In *SIGGRAPH Computer Graphics*. Volume 21, Number 4. [New York] : ACM, 1987. s. 153–162.

[5] Shewchuk, J. R. Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. In *Discrete & Computational Geometry*. Volume 18, Number 3. New York : Springer, 1997. s. 305–363.

[6] Laidlaw, D. H.; Trumbore, W. B.; Hughes, J. F. Constructive Solid Geometry for Polyhedral Objects. In *SIGGRAPH Computer Graphics*. Volume 20, Number 4. [New York] : ACM, 1986. s. 161–170.

[7] Sugihara, K.; Iri, M. A solid modelling system free from topological inconsistency. In *Journal of Information Processing*. Volume 12, Number 4. Tokyo : Information Processing Society of Japan, 1989. 380–393.

[8] *CGAL – Computational Geometry Algorithms Library* [online]. 2011 [cit. 2011-11-15]. Dostupné z WWW: <`http://www.cgal.org`>.

[9] *Carve CSG – a fast and robust constructive solid geometry library* [online]. 2011 [cit. 2011-11-15]. Dostupné z WWW: <`http://carve-csg.com`>.

[10] Sargeant T. *Carve* [počítačový program]. Ver. 1.0.0. [s.l.], 2009 [cit. 2011-06-01]. Dostupné z WWW: <http://code.google.com/p/carve/downloads/list>.

[11] *Maya* [počítačový program]. Ver. 2012. San Rafael (California, USA) : Autodesk, 2011 [cit. 2011-11-21]. Dostupné z WWW: <http://usa.autodesk.com/maya/trial>.

[12] *Object Files (.obj)* [online]. 2011 [cit. 2011-11-21]. Dostupné z WWW: <http://paulbourke.net/dataformats/obj>.

[13] *Iphigenie meshes* [online]. 2009 [cit. 2011-10-11]. Dostupné z WWW: <http://www.graphics.rwth-aachen.de/uploads/media/iphigenie-meshes.zip>.

[14] *Suggestive Contour Gallery* [online]. 2011 [cit. 2011-12-03]. Dostupné z WWW: <http://gfx.cs.princeton.edu/proj/sugcon/models>.

[15] *MeshLab* [počítačový program]. Ver. 1.3.0b. Pisa (Italy) : Visual Computing Lab – ISTI – CNR, 2010 [cit. 2011-01-24]. Dostupné z WWW: <http://meshlab.sourceforge.net>.

[16] Cignoni, P.; Rocchini C.; Scopigno R. Metro: measuring error on simplified surfaces. In *Computer Graphics Forum*. Volume 17, Number 2. [Oxford] : Blackwell Publishers, 1998. s. 167–174. Dostupné z WWW: <http://vcg.sf.net>.

[17] Bischoff, S.; Pavic, D.; Kobbelt, L. Automatic Restoration of Polygon Models. In *ACM Transactions on Graphics*. Volume 24, Number 4. [New York] : ACM, 2005. s. 1332–1352.

[18] Murali, T. M.; Funkhouser, T. A. Consistent solid and boundary representations from arbitrary polygonal data. In *Proceedings of the 1997 symposium on Interactive 3D graphics*. [New York] : ACM, 1997. s. 155–162.

Seznam tabulek

Seznam použitých zkratek

BSP – Binary space partitioning

CGAL – Computational Geometry Algorithms Library

CSG – Constructive solid geometry

FPU – Floating-point unit

RAM – Random access memory

A. Obsah přiloženého CD

Přiložený disk obsahuje tyto adresáře:

- `bin` – Řešení RazeCSG přeložené do spustitelné podoby.
- `models` – Testovací modely ve formátu OBJ a dávkové soubory použité při experimentech v kapitole 5.
- `results` – Výsledné modely ve formátu OBJ vygenerované RazeCSG při experimentech v kapitole 5.
- `src` – Zdrojové kódy RazeCSG.
- `text` – Text této práce ve formátu PDF a jeho zdrojový kód pro LaTeX.