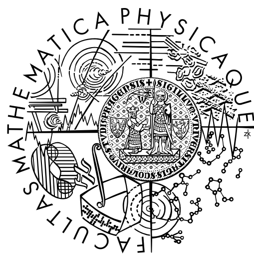


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

David Král

Rekonstrukce 3D modelu z kolmých průmětů

Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce: doc. RNDr. Elena Šikudová, Ph.D.
Studijní program: Informatika

Praha 2024

Prohlašuji, že jsem tuto bakalářskou 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 má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V .... dne ....

Podpis autora

Děkuji doc. RNDr. Eleně Šikudové, Ph.D. za cenné rady a nápady, které přispěly k vylepšení aplikace. Rovněž děkuji své rodině za trpělivost během psaní práce. A v neposlední řadě děkuji také Mgr. Vojtěchu Tázlarovi za jeho pomoc v počátcích této práce.

Název práce: Rekonstrukce 3D modelu z kolmých průmětů

Autor: David Král

Katedra: Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce: doc. RNDr. Elena Šikudová, Ph.D., Katedra softwaru a výuky informatiky

Abstrakt: 3D CAD systémy usnadňují tvorbu a úpravu 3D modelů. Přesto se stále používají také technické výkresy a 2D projekce. Rekonstrukce 3D modelu z těchto projekcí však vyžaduje manuální práci uživatele. Cílem této práce je vytvořit aplikaci, která z obrázků kolmých průmětů dokáže 3D model zrekonstruovat. V průběhu práce navrhneme algoritmus pro generování trojúhelníkové sítě z binárních voxelových dat. Navržený algoritmus generuje hladký povrch, aniž by vyžadoval informace o normálách. Algoritmus také přizpůsobuje velikost a počet generovaných trojúhelníků rekonstruovanému povrchu. Na rovných plochách modelu jich generuje minimální počet, zatímco detaily jsou zachovány a reprezentovány větším počtem trojúhelníků.

Klíčová slova: Kolmý průmět, Mračno bodů, 3D model

Title: 3D Model Reconstruction from Orthogonal Projection

Author: David Král

Department: The Department of Software and Computer Science Education

Supervisor: doc. RNDr. Elena Šikudová, Ph.D., The Department of Software and Computer Science Education

Abstract: 3D CAD systems facilitate the creation and modification of 3D models, but technical drawings and 2D projections are still used. However, the reconstruction of a 3D model from these projections requires manual work by the user. The goal of this thesis is to develop an application that can reconstruct a 3D model from images of orthogonal projections. We propose an algorithm for triangular mesh generation from binary voxel data. The proposed algorithm generates a smooth surface without requiring normal information. The algorithm also adapts the size and number of generated triangles to the reconstructed surface. It generates a minimal number of triangles on the flat surfaces of the model, while the details are preserved and represented through a larger number of triangles.

Keywords: Orthogonal projection, Point cloud, 3D model

Obsah

Úvod **6**

1 Reprezentace 3D modelů **7**
1.1 Objemové reprezentace . . . 7
1.2 Povrchové reprezentace . . . 7

2 Stávající algoritmy **9**
2.1 Marching Cubes . . . 9
2.1.1 Marching Tetrahedra . . . 10
2.1.2 Extended Marching Cubes . . . 11
2.2 Surface Nets . . . 12
2.3 Dual Contouring . . . 14
2.3.1 Adaptive Dual Contouring . . . 15

3 Rekonstrukce modelu **17**
3.1 Předzpracování vstupních obrázků . . . 17
3.2 Generování trojúhelníkové sítě . . . 18
3.3 Zjednodušení trojúhelníkové sítě . . . 19

4 Výsledky **21**

5 Dokumentace **24**
5.1 Sestavení a spuštění aplikace . . . 24
5.2 Konzolová aplikace . . . 25
5.3 Aplikace s grafickým uživatelským rozhraním . . . 26
5.4 Programátorská dokumentace . . . 29
5.4.1 OrthoTo3D . . . 29
5.4.2 ConsoleApp . . . 29
5.4.3 AvaloniaApp a AvaloniaApp.Desktop . . . 30
5.4.4 StlEncoder a ObjEncoder . . . 30
5.4.5 OrthoTo3DTests . . . 30
5.5 Zásuvné moduly . . . 30
5.5.1 Implementace zásuvného modulu . . . 31

Závěr **32**

Literatura **33**

Seznam obrázků **34**

Úvod

3D CAD (z anglického computer-aided design) systémy usnadňují tvorbu a modifikaci 3D modelů. I přes rozšíření těchto programů se stále používají také technické výkresy, proto CAD programy umožňují také generování 2D projekcí. Rekonstrukce 3D modelu z těchto projekcí však stále vyžaduje manuální práci uživatele. A právě rekonstrukcí trojrozměrného modelu z jeho kolmých průmětů se budeme v této práci zabývat.

3D model můžeme pomocí metody fotogrametrie zrekonstruovat z několika fotografií objektu pořízených z různých úhlů. Během rekonstrukce jsou ve snímcích detekovány odpovídající body, které jsou následně umisťovány do 3D prostoru. K tomu je zapotřebí odhadnout pozice kamer při pořízení snímků.

Cílem této práce je vytvoření aplikace, která 3D model z obrázků zrekonstruuje. Rekonstrukci 3D modelu si však usnadníme, neboť model budeme rekonstruovat z obrázků kolmých průmětů. Budeme tak přesně znát pozice kamer. Dále budeme u obrázků kolmých průmětů předpokládat jednobarevné pozadí, což nám umožní jednoduché rozlišení objektu a pozadí.

Fotogrametrie je výpočetně náročná metoda. Od naší aplikace budeme očekávat rychlejší rekonstrukci, které bychom díky omezením na kolmé průměty měli dosáhnout. Snímky kolmých průmětů navíc na úspěšnou rekonstrukci pomocí fotogrametrie nemusejí stačit.

V této práci se nejdříve seznámíme s používanými způsoby reprezentací 3D modelů. Dále si představíme některé stávající algoritmy rekonstrukce povrchu, kterými se následně inspirujeme při návrhu vlastního algoritmu. Tento algoritmus si popíšeme a zhodnotíme výslednou kvalitu zrekonstruovaných modelů. Uživatelskou a programátorskou dokumentaci k přiložené aplikaci naleznete v kapitole 5.

1 Reprezentace 3D modelů

3D modely můžeme v počítači reprezentovat několika způsoby, které lze rozdělit do dvou kategorií: *objemové reprezentace* a *povrchové reprezentace*. Objemové reprezentace obsahují přímé informace o tom, které body prostoru jsou součástí objektu. Povrchové reprezentace pak obsahují pouze informace o povrchu modelovaných objektů.

1.1 Objemové reprezentace

Mezi objemové reprezentace se řadí několik odlišných metod reprezentací modelu. Nás bude v této práci zajímat především reprezentace pomocí výčtu voxelů. Voxel (z anglického *volume element*) je jednotka prostoru, který byl rozdělen trojrozměrnou mřížkou na buňky shodné velikosti. Jedná se tedy o obdobu pixelu v trojrozměrné grafice. Ve voxelech může být kromě souřadnic uložena také informace o materiálu, ze kterého je modelovaný objekt složen. Toho se využívá například v počítačových hrách se zničitelným prostředím pro reprezentaci terénu. V medicíně jsou pak voxely používány v počítačové tomografii, magnetické rezonanci a dalších zobrazovacích technikách, kde je potřeba rozlišit různé typy tkání. Tyto lékařské zobrazovací přístroje mohou mít na jednotlivých osách rozdílné rozlišení, voxel tak nutně nemusí být krychlí.

Další používanou metodou je reprezentace pomocí CSG (z anglického *Constructive Solid Geometry*). Tato metoda využívá množinových operací pro složení objektu z elementárních geometrických těles. Výhodou CSG je vysoká matematická přesnost reprezentace objektu.

Nevýhodou objemových reprezentací obecně je pak obtížnější zobrazení modelu, proto existuje množství algoritmů pro převod takto reprezentovaných modelů na modely reprezentované povrchově. Některé algoritmy si představíme v kapitole 2. Další možností je zobrazení modelu pomocí vrhání paprsku (*ray casting*). Jedná se však o výpočetně a časově náročnější metodu.

1.2 Povrchové reprezentace

První povrchovou reprezentací, kterou si představíme, je *mračno bodů*. Mračno bodů je množina bodů v prostoru, které se nacházejí na povrchu modelovaného objektu. Tyto body však mezi sebou nejsou nijak propojeny. Body mračna mohou nést další informace například o barvě nebo o normále na povrch objektu. Mračno bodů bývá výstupem lidaru a dalších 3D skenerů.

Pro uložení mračna bodů můžeme použít *oktantový strom*. Jedná se o strom, jehož jednotlivé vrcholy odpovídají buňkám prostoru a každý jeho vnitřní vrchol má právě 8 synů. Datová struktura tak prostor rekurzivně dělí na 8 stejně velkých částí, dokud velikost buňky nedosáhne předem definovaných rozměrů. V listech stromu je pak uložen seznam vrcholů, které se nacházejí uvnitř buňky prostoru, jíž vrchol reprezentuje. Tato struktura umožňuje efektivní hledání vrcholů, které tak zabere čas $\mathcal{O}(\log_8 n + m)$, kde n je počet buněk oktantového stromu a m je počet vrcholů v listu stromu.

Asi nejrozšířenější metodou reprezentace 3D modelů je *polygonová*, nebo *trojúhelníková síť*. Pro označení polygonové sítě se používá také anglický termín *mesh*. Tato síť je složena z mnohoúhelníkových, resp. trojúhelníkových stěn, pomocí nichž je aproximován povrch modelovaného objektu. Důvodem rozšíření této reprezentace je především optimalizace hardwaru na vykreslování takto reprezentovaných modelů.

Polygonovou síť lze v paměti jednoduše uložit jako seznam vrcholů a stěn. Vrcholy obsahují informace o svých souřadnicích. Každá stěna pak obsahuje odkazy na vrcholy, z nichž je tvořena. Odkazy na vrcholy mohou mít formu referencí, nebo indexů do seznamu vrcholů. Tato reprezentace polygonové sítě se používá pro předávání modelů grafické kartě nebo jejich ukládání na disk.

Algoritmy pracující s polygonovými sítěmi však často potřebují najít stěny, jejichž součástí je daný vrchol. Aby nám toto hledání nezabralo lineární čas vůči počtu stěn v celém modelu, můžeme ke každému vrcholu přidat odkazy na přilehlé stěny. Používají se však i další sofistikovanější datové struktury, které usnadňují operace s polygonovou sítí.

Jednou z nich je tzv. *okřídlená hrana* (*winged edge*). Tato datová struktura je tvořena seznamem hran, v němž má každá hrana odkaz na své koncové vrcholy, přilehlé stěny a sousední hrany přilehlých stěn. Nevýhodou této struktury je vyšší paměťová náročnost.

2 Stávající algoritmy

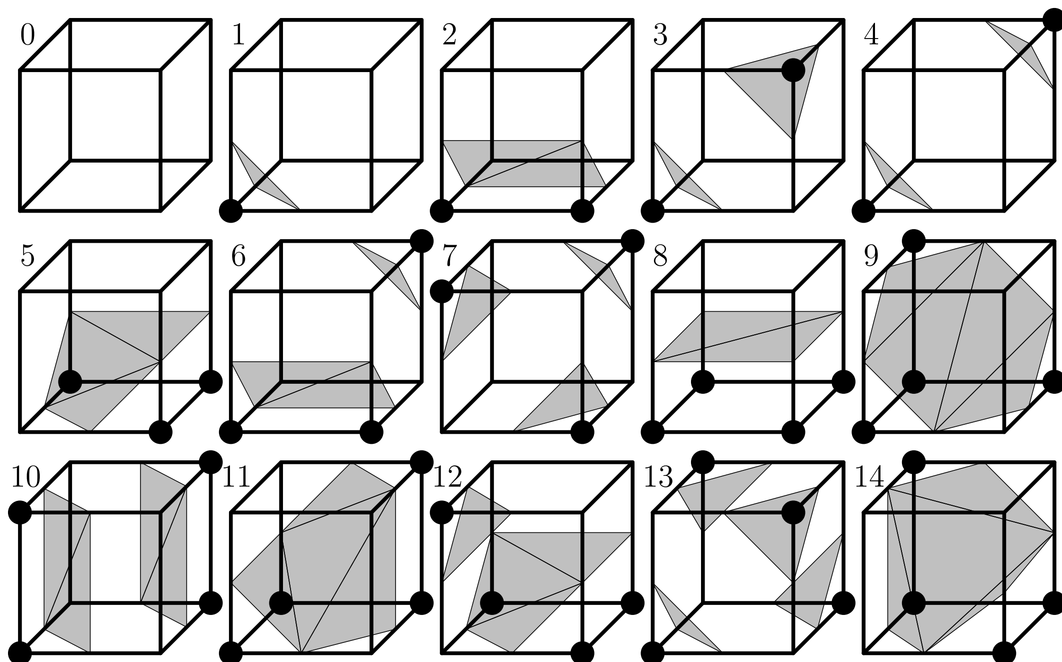
V této kapitole si představíme některé ze stávajících algoritmů, které se používají pro generování trojúhelníkové sítě z objemových reprezentací modelu. Důležité pro nás budou především myšlenky použité při řešení různých problémů a nedostatků, které mohou při generování trojúhelníkové sítě nastat.

2.1 Marching Cubes

Jedním z nejznámějších je algoritmus *Marching Cubes* (Lorensen; Cline, 1987). Tento algoritmus byl původně vyvinut za účelem generování trojúhelníkových sítí z 3D lékařských dat pořízených např. pomocí počítačové tomografie. Postupem času si však našel cestu i do dalších odvětví.

Algoritmus pracuje nad *vzorkovaným skalárním polem*. Hodnoty tohoto pole můžeme jednoduše uložit do jednotlivých voxelů. Cílem algoritmu je vygenerovat trojúhelníkovou síť reprezentující tzv. *izopovrch*, tedy plochu spojující body v prostoru, v nichž nabývá skalární pole určité konstantní hodnoty. Tuto hodnotu můžeme bez újmy na obecnosti označit jako nulovou, stačí ji odečíst od všech bodů vzorkovaného skalárního pole. Vně objektu, jehož povrch generujeme, budou ležet kladné hodnoty, zatímco uvnitř objektu budou ležet hodnoty záporné. Absolutní hodnoty nám pak budou udávat vzdálenost k povrchu.

Algoritmus získal své jméno po pomyslných krychlích, které jsou tvořeny 8 voxely v jejich rozích. Každý z 8 rohů krychle leží buď uvnitř, nebo vně objektu. Pokud některé rohy krychle leží uvnitř objektu, zatímco jiné vně, pak buňkou musí procházet povrch. Povrchem prochází právě ty hrany krychle, jejichž koncové vrcholy obsahují hodnoty s opačným znaménkem.

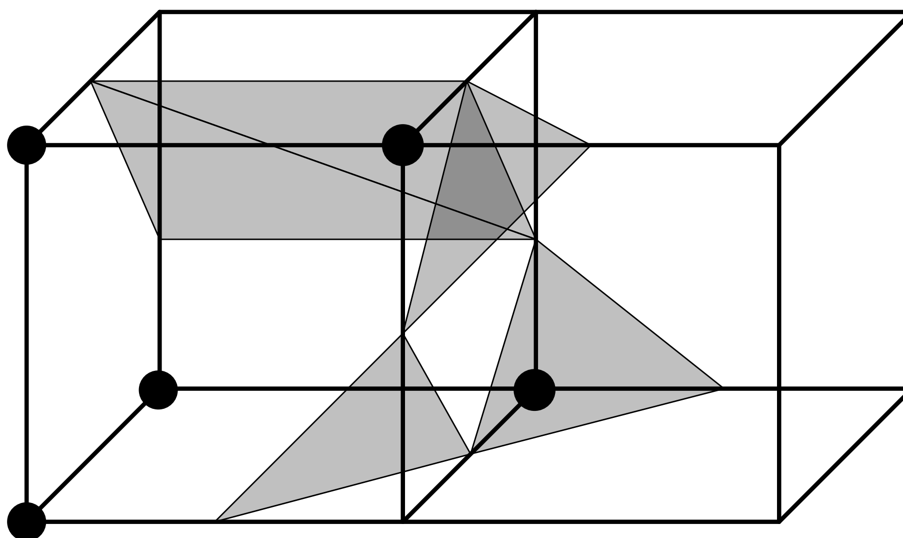


Obrázek 2.1 Základní konfigurace buněk v algoritmu *Marching Cubes*. Tyto konfigurace určují podobu části generované trojúhelníkové sítě.

Celkem existuje $2^8 = 256$ různých způsobů, jak může povrch buňkou procházet. Některé z těchto konfigurací jsou však až na orientaci či překlopení ekvivalentní. Dále si povšimněme, že i při záměně vnitřku a vnějšku objektu bude povrch procházet buňkou stále stejným způsobem. Těchto poznatků můžeme využít ke snížení počtu základních konfigurací na 15. Základní konfigurace si můžeme prohlédnout na obrázku 2.1.

Během samotného generování trojúhelníkové sítě algoritmus iteruje přes jednotlivé buňky. Dle znamének hodnot uložených ve voxelech v rozích buněk pak zvolí jednu z 256 konfigurací a dle ní vygeneruje část trojúhelníkové sítě. Přesnou pozici vrcholů generovaných trojúhelníků na hranách buňky určíme pomocí lineární interpolace hodnot v koncových bodech hrany.

Nevýhodou algoritmu v jeho původní podobě je však nejednoznačnost některých konfigurací buněk, které lze triangulovat více způsoby. Například při triangulaci konfigurace 10 (viz obrázek 2.1) můžeme tmavé voxely, které představují modelovaný objekt, mezi sebou propojit, nebo je od sebe oddělit. Autoři u těchto nejednoznačných konfigurací vždy zvolili jednu z možností, která se tak stala základní konfigurací. Při troše smůly se však mohou ve výsledném povrchu objevit trhliny, jak vidíme na obrázku 2.2. Tento nedostatek byl však již opraven přidáním několika dalších základních konfigurací buněk.



Obrázek 2.2 Při triangulaci sousedních buněk např. dle konfigurací 3 a inverzní 6 na sebe části povrchu nenavazují. V generovaném povrchu tak mohou vzniknout trhliny.

Pokud generujeme trojúhelníkovou síť nad binárními daty, umístíme vrcholy generovaných trojúhelníků doprostřed hran, kterými povrch prochází.

Generování trojúhelníků v jednotlivých buňkách je na sobě navzájem nezávislé. Algoritmus *Marching Cubes* tak můžeme snadno paralelizovat.

2.1.1 Marching Tetrahedra

Algoritmus *Marching Cubes* byl po svém uvedení chráněn patentem, což vedlo ke vzniku volně dostupného algoritmu *Marching Tetrahedra* (Doi; Koide, 1991). *Marching Tetrahedra* je obdobou algoritmu *Marching Cubes*. Na rozdíl od něj však není základní buňkou krychle, ta je totiž dále rozdělena na 6 čtyřstěnů.

Ke triangulaci čtyřstěnů je pak opět využita tabulka konfigurací, která je však tentokrát menší. Přesné pozice vrcholů generovaných trojúhelníků určíme stejně jako v algoritmu *Marching Cubes* lineární interpolací hodnot koncových vrcholů hran, na nichž vrcholy leží.

Tento algoritmus netrpí nejednoznačností konfigurací jako původní algoritmus. V generovaném povrchu tak nevznikají trhliny.

2.1.2 Extended Marching Cubes

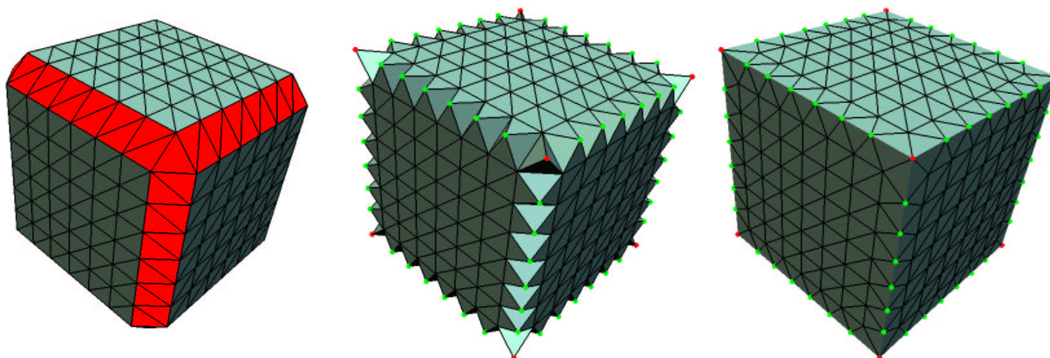
Algoritmus *Marching Cubes* umisťuje vrcholy trojúhelníků na hrany krychlí, které tvoří jednotlivé buňky prostoru. Nevýhodou tohoto přístupu je nemožnost generování ostrých hran a rohů, které se na těchto hranách nenacházejí (viz obrázek 2.7). Algoritmus *Extended Marching Cubes* (Kobbelt et al., 2001) si klade za cíl tento problém vyřešit.

Oproti původnímu algoritmu nyní potřebujeme navíc znát normály na izopovrch v bodech skalárního vzorkovaného pole. Algoritmus pak během rekonstrukce povrchu rozlišuje buňky, které obsahují ostrou hranu nebo roh, od buněk, které je neobsahují. K vzájemnému rozlišení těchto buněk navrhují autoři algoritmu spočítat maximální úhel mezi normálami na povrch v bodech, v nichž rekonstruovaný povrch protíná hranu buňky. Tyto normály můžeme jednoduše získat lineární interpolací normál v koncových bodech hrany. Pokud je maximální úhel mezi normálami nižší než předem definovaná prahová hodnota, pak buňka neobsahuje žádné ostré prvky. U konfigurací buněk, které obsahují více něž jednu komponentu souvislosti, provedeme tento test pro každou komponentu zvlášť.

Buňky, v nichž se nenachází žádná ostrá hrana nebo roh, jsou triangulovány stejným způsobem jako v původním algoritmu *Marching Cubes*. Do buněk a každé jejich komponenty souvislosti, která obsahuje nějaký ostrý prvek, je potřeba přidat nový vrchol. Pozici tohoto vrcholu určíme pomocí metody nejmenších čtverců tak, aby ležel v průniku rovin určených body na hranách buňky, jimiž prochází povrch, a normálami v nich.

Po přidání nového vrcholu je komponenta souvislosti triangulována. Generované trojúhelníky tvoří tzv. *vějíř trojúhelníků* (*triangle fan*) se středem v nově přidaném vrcholu. Při generování části trojúhelníkové sítě uvnitř buňky nebereme, stejně jako v původním *Marching Cubes*, v úvahu okolní buňky. Následkem toho je, že hrany vygenerované trojúhelníkové sítě nerespektují přítomnost ostrých hran modelu (viz obrázek 2.3).

Proto musíme po vygenerování trojúhelníkové sítě provést ještě závěrečné úpravy. Během nich budeme používat operaci *překlopení hrany* (*edge flipping*). Jedná se o operaci na dvojici přilehlých trojúhelníků, při níž nahradíme jejich společnou hranu hranou, která spolu propojí zbylé dva vrcholy. Pomocí této operace překlopíme hrany všude tam, kde tím propojíme dva vrcholy, které leží na ostrém prvku rekonstruovaného povrchu, tedy ty vrcholy, které byly algoritmem přidány dovnitř buňky. Výslednou trojúhelníkovou síť si můžeme prohlédnout na obrázku 2.3.



Obrázek 2.3 Fáze algoritmu *Extended Marching Cubes*: Algoritmus *Marching Cubes* generuje trojúhelníkovou síť bez ostrých prvků. Nejdříve identifikujeme buňky obsahující ostré hrany nebo rohy (vlevo). Do těchto buněk přidáme nové vrcholy (uprostřed). Po překlopení hran získáme výslednou trojúhelníkovou síť s ostrými hranami (vpravo). (Převzato z *Feature sensitive surface extraction from volume data* Kobbelt et al. (2001))

2.2 Surface Nets

Dalším algoritmem, který si v této práci představíme, je algoritmus *Surface Nets* (Gibson, 1998). Jedná se o algoritmus, který na vstupu očekává pouze binární data, přesto však generuje hladký povrch.

Pro generování trojúhelníkové sítě z binárních dat můžeme využít také algoritmus *Marching Cubes*, který jsme si již představili. Při jeho použití však výsledný povrch může obsahovat terasovité artefakty. Umisťováním vrcholů pouze do středů hran buněk totiž nemůžeme znázornit rovinu, která mřížkou prochází pod mírným úhlem. Výsledkem je tak několik odstupňovaných terasovitých ploch (viz obrázek 2.4). Tyto terasovité artefakty plně neodstraní ani následné použití lokálního vyhlazovacího filtru. Při použití vyhlazovacího filtru s velkým poloměrem navíc ztrácíme drobné detaily, jejichž zachování může být například v lékařství klíčové.

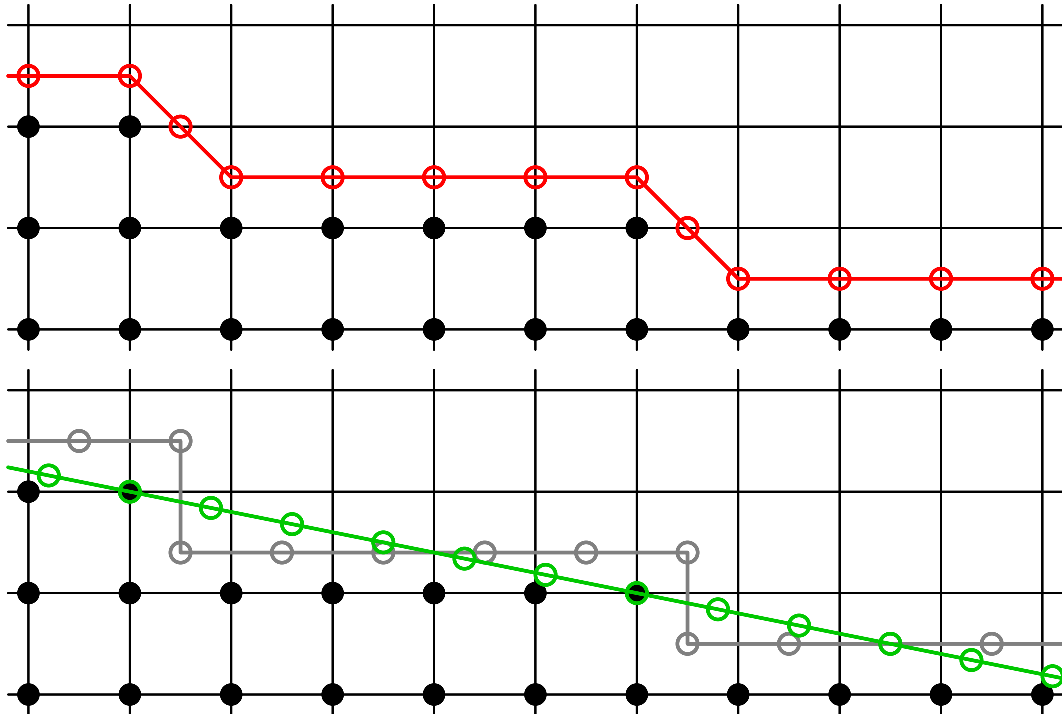
Algoritmus *Surface Nets* je iterativním algoritmem. Nejprve vytvoříme síť vrcholů, po které algoritmus získal své jméno, a následně ji budeme postupně vyhlazovat. Během vyhlazování však musíme omezit pohyb vrcholů sítě, abychom nepřišli o ostré hrany a další detaily modelu.

Prostor je rozdělen na krychlové buňky stejně jako v algoritmu *Marching Cubes*. Do středu každé buňky, kterou prochází povrch modelovaného objektu, umístíme vrchol sítě. Jedná se o ty buňky, v jejichž rozích se nacházejí voxely s různou binární hodnotou. Síť pak vytvoříme propojením vrcholů v přilehlých buňkách. Tato síť však není tvořena trojúhelníky, ale čtverci.

Nyní již můžeme přejít k samotnému vyhlazování. Nejprve si definujeme energii vrcholu v jako:

$$\sum_{e \in E_v} |e|^2,$$

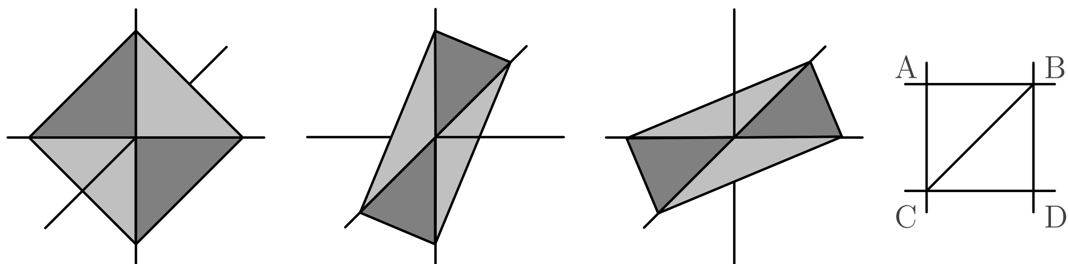
kde E_v je množina hran vedoucí z vrcholu v. Během jednotlivých iterací algoritmu budeme postupně procházet vrcholy sítě a přesouvat je na pozici, kde minimalizujeme jejich energii. Pohyb vrcholu sítě však omezíme pouze na buňku, v níž se vrchol nachází. Díky tomu se síť po několika iteracích nesmrští do jediného bodu a tvar modelu, včetně jeho ostrých hran, tak bude zachován. Počet iterací algoritmu může být dán předem, nebo určen dynamicky v závislosti na chování



Obrázek 2.4 Povrch generovaný algoritmem *Marching Cubes* nad binárními daty s patrnými terasovitými artefakty (červeně). Síť vrcholů algoritmu *Surface Nets* před vyhlazováním (šedě) a po rozvolnění s minimální energií (zeleně).

sítě. Gibsonová dále ve své práci navrhuje použití alternativních definic energií vrcholů pro přizpůsobení podoby generovaného povrchu.

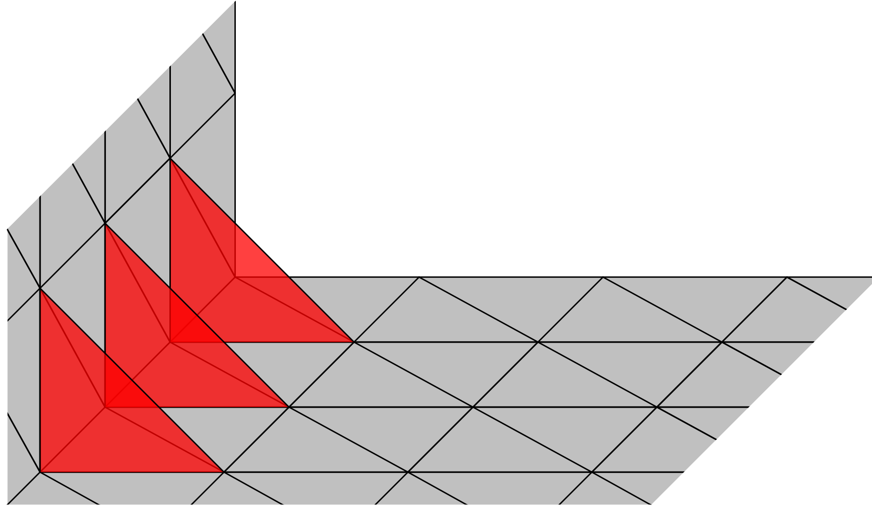
Na závěr nám ještě zbývá síť triangulovat. Na obrázku 2.5 vidíme, že každý vrchol může být propojen s až 6 sousedními, s nimiž pak tvoří až 12 trojúhelníků. Během triangulace tedy budeme generovat trojúhelníky, pro něž se v přilehlých buňkách nacházejí potřebné vrcholy. Tento přímočarý postup však generuje překrývající se trojúhelníky. Při zpracování vrcholů A a D chceme vygenerovat trojúhelníky ABC a BCD, při zpracování vrcholů B a C by však již trojúhelníky ACD a ABD být generovány neměli (viz obrázek 2.5). Proto budeme při generování brát v úvahu pouze 6 z 12 trojúhelníků, například ty tmavé.



Obrázek 2.5 Triangulace sítě algoritmu *Surface Nets*. Každý vrchol může být propojen s až 6 sousedy, s nimiž pak tvoří 12 trojúhelníků. Při triangulaci je však používána pouze polovina z nich.

Zde si dovolím poznamenat, že postup triangulace popsaný v *Constrained elastic surface nets* (Gibson, 1998) vede ke generování trojúhelníků navíc (viz obrázek 2.6). Tyto trojúhelníky, které se nacházejí na hranách, by ve výsledné

trojúhelníkové síti neměli být přítomny. Proto trojúhelník vygenerujeme pouze pokud sousedi právě zpracovávaného vrcholu, kteří spolu s ním trojúhelník tvoří, mají ještě dalšího společného souseda.



Obrázek 2.6 Červené trojúhelníky by při závěrečné triangulaci neměli být vygenerovány. V postupu popsaném v *Constrained elastic surface nets* (Gibson, 1998) však generovány jsou.

2.3 Dual Contouring

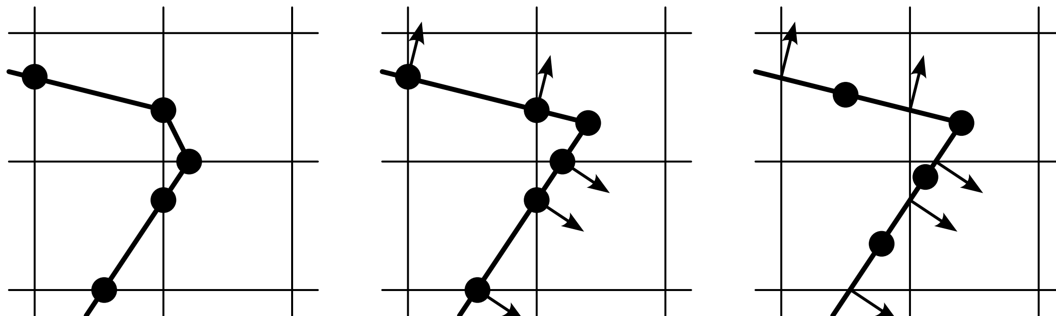
Algoritmus *Dual Contouring* (Ju et al., 2002) je dalším z algoritmů, který stejně jako algoritmus *Extended Marching Cubes* řeší nedostatky původního *Marching Cubes* ohledně generování ostrých hran. Nevýhodou všech v této práci dříve prezentovaných algoritmů, však je, že i rozlehlé rovné plochy dláždí malými trojúhelníky, jejichž velikost odpovídá velikosti buněk. Algoritmy *Marching Cubes* a *Surface Nets* byly vyvinuty za účelem rekonstrukce povrchu z dat pořízených lékařskými zobrazovacími přístroji a pro minimalizaci počtu generovaných trojúhelníků tak nebyl žádný důvod.

Algoritmus *Dual Contouring* byl naopak již od svého vzniku určen pro použití v počítačových hrách, kde má počet vykreslovaných trojúhelníků značný dopad na výkon. Ve své adaptivní verzi algoritmus generuje pouze takový počet trojúhelníků, který je potřebný pro zachování požadované úrovně detailů. Rozsáhlé rovné plochy modelu tak reprezentuje malým počtem velkých trojúhelníků, zatímco detaily znázorňuje více malými trojúhelníky.

Nejprve si algoritmus *Dual Contouring* představíme v jeho neadaptivní verzi. V ní je prostor rozdělen pravidelnou mřížkou stejně jako u předchozích algoritmů. Algoritmus generuje trojúhelníkovou síť z tzv. *hermitovských dat*. Jedná se o data, která ke své práci vyžadoval již algoritmus *Extended Marching Cubes*, tedy informace o průsečících rekonstruovaného povrchu s mřížkou a normálách v nich. Během rekonstrukce povrchu budeme iterovat přes jednotlivé buňky mřížky a v buňkách, jimiž prochází povrch, vygenerujeme vrchol sítě. Tento vrchol umístíme do průniku rovin určených hermitovskými daty na hranách buňky. Všechny

roviny se nemusí protínat v jediném bodě, proto použijeme metodu nejmenších čtverců, tak abychom nepřesnost umístění vrcholu minimalizovali.

Algoritmus *Dual Contouring* řadíme k duálním metodám, které neumisťují vrcholy na hrany mřížky, ale dovnitř buněk. V obrázku 2.7 vidíme, že nám tento přístup umožňuje ve výsledném modelu reprezentovat ostré hrany.



Obrázek 2.7 Algoritmus *Marching Cubes* umisťuje vrcholy na hrany mřížky a nemůže tak reprezentovat ostrou hranu (vlevo). Algoritmus *Extended Marching Cubes* rozpozná buňky, které obsahují ostré prvky, a vygeneruje v nich další vrcholy (uprostřed). Algoritmus *Dual Contouring* pak umisťuje všechny vrcholy dovnitř buněk (vpravo). Algoritmy *Extended Marching Cubes* a *Dual Contouring* vyžadují informace o normálách na povrch v místech, kde rekonstruovaný povrch prochází mřížkou.

Následně pro každou hranu mřížky, kterou prochází povrch, vygenerujeme část povrchu. Každá taková hrana je součástí čtyř sousedních buněk, v nichž se nacházejí vrcholy vygenerované v předchozím kroku. Propojením vrcholů získáme čtyřúhelník, který tak musíme při generování trojúhelníkové sítě ještě rozdělit na dva trojúhelníky.

2.3.1 Adaptive Dual Contouring

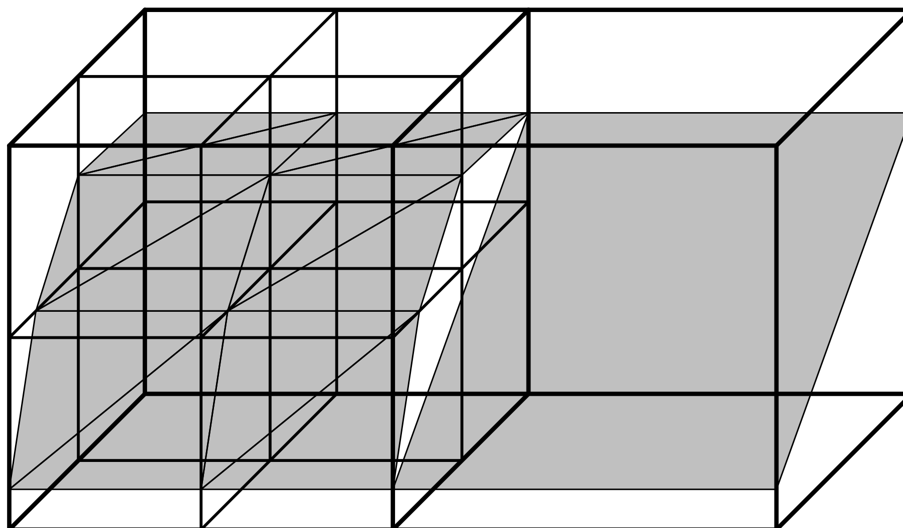
Ve své adaptivní verzi se algoritmus *Dual Contouring* přizpůsobuje tvaru povrchu a požadované úrovni detailů. Velikost a počet generovaných trojúhelníků se tak v různých částech modelu může lišit. Výsledkem je, že při zachování stejné úrovně detailů bude vygenerovaný model obsahovat znatelně menší počet trojúhelníků.

V adaptivní verzi není prostor na buňky rozdělen pravidelnou mřížkou, ale pomocí *oktantového stromu*, kde každý jeho uzel odpovídá nějaké části prostoru. Celý modelovaný prostor je na počátku obsažen v jediné buňce, která je následně rekurzivně dělena na 8 stejně velkých částí, pokud je odchylka v ní potencionálně umístěného vrcholu příliš velká. Po zastavení rekurzivního dělení odpovídají listy stromu buňkám, se kterými tento algoritmus dále pracuje.

Zbytek algoritmu zůstává stále stejný. Nejprve tedy vygenerujeme vrcholy uvnitř buněk a následně je budeme mezi sebou propojovat. Oproti neadaptivní verzi algoritmu, kde byla každá hrana součástí 4 buněk, může nyní hrana na rozmezí buněk rozdílné velikosti náležet také pouze 3 buňkám. Tento případ však není potřeba nijak složitě ošetřovat. Stačí jednoduše vygenerovat jediný trojúhelník namísto dvou.

Adaptivní mřížku lze použít i u primárních metod, tedy algoritmů, které umisťují vrcholy na hrany mřížky. Při přímočaré implementaci se však mohou v ge-

nerovaném povrchu objevit trhliny, jak vidíme v obrázku 2.8. Jsou však známy algoritmy, které tyto trhliny umí zacelit. Jedním z nich je například algoritmus *Adaptive Marching Cubes* (Shu et al., 1995).



Obrázek 2.8 Při použití adaptivní mřížky u primárních metod mohou ve výsledném povrchu vzniknout trhliny na rozhraní buněk rozdílných velikostí.

3 Rekonstrukce modelu

Nyní, když jsme si představili různé reprezentace 3D modelů a stávající algoritmy pro generování trojúhelníkové sítě, se již zaměříme na samotnou rekonstrukci 3D modelu z jeho kolmých průmětů.

Algoritmus rekonstrukce je založen na jednoduché myšlence: rekonstruovaný objekt odpovídá průniku lineárních extruzí jeho kolmých průmětů. To si můžeme představit jako protažení jednotlivých kolmých průmětů do hloubky. Z pixelů, které v kolmých průmětech představují rekonstruovaný objekt, se tak stanou voxely. A průnikem těchto sad voxelů pak získáme výsledný model. Povrch tedy budeme rekonstruovat z binárních dat, kde je voxel na pozici (x, y, z) definován takto:

$$V[x,y,z] = \begin{cases} -1, & \text{jsou-li pixely na pozicích } (z,x) \text{ horního, } (z,y) \text{ pravého} \\ & \text{a } (x,y) \text{ čelního průmětu pixely objektu,} \\ 1, & \text{jinak.} \end{cases}$$

Pro rekonstrukci povrchu z binárních dat můžeme použít algoritmus *Surface Nets*. Nevýhodou tohoto algoritmu však je, že generuje velké množství trojúhelníků i na rovných plochách modelu. Ostatní algoritmy, které jsme si v této práci představili, pak vyžadují více informací o rekonstruovaném povrchu. Tyto algoritmy pro rekonstrukci modelu v přiložené aplikaci nepoužijeme, ačkoliv existují metody pro odhad potřebných průsečíků s mřížkou a normál v nich.

V této práci navrhneme nový algoritmus pro rekonstrukci povrchu z binárních objemových dat. Naším cílem je vygenerovat hladký povrch bez terasovitých artefaktů, algoritmus by však měl zachovat ostré hrany modelu. Dále bychom chtěli, aby algoritmus negeneroval zbytečně vysoké množství trojúhelníků. To vše by měl algoritmus zvládnout v rozumném čase. Jedná se o problémy, které řeší také stávající algoritmy, jimž jsme se věnovali v kapitole 2. My se tak jimi můžeme při návrhu inspirovat.

3.1 Předzpracování vstupních obrázků

Ze vstupních obrázků nejprve odebereme pomocí algoritmu *Flood Fill* jednobarevné pozadí. Tím od sebe pomocí průhlednosti, kterou v obrázcích uložíme pomocí alfa kanálu, odlišíme pixely objektu a pozadí. Konkrétně je v přiložené aplikaci implementována optimalizovaná verze algoritmu, kterou ve svém článku *A seed fill algorithm* navrhl Paul Heckbert (Heckbert, 1990).

Tato verze snižuje průměrný počet přístupů k jednotlivým pixelům vyplňované oblasti a šetří tak čas při porovnávání jejich barev. V naivní rekurzivní implementaci algoritmu může být každý pixel při vyplňování do 4 směrů navštíven až 4krát. V této verzi algoritmu se průměrný počet přístupů k pixelům blíží jedné. Toho je dosaženo vyplňováním oblasti po sekcích jednotlivých řádků a zapamatováním si směru vyplňování.

Pro vyšší kvalitu výsledného modelu jsou k potlačení šumu ve vstupních obrázcích použity morfologické operace. Morfologické otevření — eroze následovaná

dilatací — odstraní osamocené pixely, které v obrázku zůstaly po odebrání pozadí. Morfologické uzavření — dilatace následovaná erozí — naopak zaplní drobné mezery v průmětech rekonstruovaného objektu.

Na závěr je obrázek ještě oříznut podle rekonstruovaného objektu. Díky tomu jsou souřadnice jednotlivých projekcí zarovnány, což usnadní následnou rekonstrukci.

3.2 Generování trojúhelníkové sítě

Při generování trojúhelníkové sítě budeme iterovat přes jednotlivé voxely a pro voxely, jež leží na povrchu modelu, vygenerujeme část trojúhelníkové sítě. Voxely, které si nyní budeme představovat jako krychle, leží na povrchu, pokud některými svými stěnami přiléhají k voxelům s jinou binární hodnotou. Pro každou takovou stěnu do trojúhelníkové sítě přidáme 2 trojúhelníky na 4 vrcholech. Tento algoritmus je velmi podobný generování iniciální sítě vrcholů v algoritmu *Surface Nets*. Čtvercová síť je však rovnou triangulována. Časová složitost algoritmu 1 je kubická vůči rozměrům vstupních obrázků.

Algoritmus 1 Zrekonstruuje trojúhelníkovou síť z rastrových obrázků kolmých průmětů. Síť odpovídá povrchu krychlí reprezentujících jednotlivé voxely a není nijak vyhlazena.

```
 1: function MeshFromImages(top, right, front)
 2:                      ▷ Obrázky horního, pravého a čelního kolmého průmětů
 3:   width ← min(front.width, top.height)                    ▷ Šířka modelu
 4:   height ← min(front.height, right.height)                ▷ Výška modelu
 5:   length ← min(right.width, top.width)                    ▷ Délka modelu
 6:   mesh ← prázdná trojúhelníková síť
 7:
 8:   for x ← 0 to width do
 9:     for y ← 0 to height do
10:       if front[x, y] je pixel objektu then
11:         for z ← 0 to length do
12:           if top[z, x] je objekt and right[z, y] je objekt then
13:             if top[z, x − 1] or front[x − 1, y] není objekt then
14:               Do sítě mesh přidáme levou stěnu voxelu (x, y, z)
15:             if top[z, x + 1] or front[x + 1, y] není objekt then
16:               Do sítě mesh přidáme pravou stěnu voxelu (x, y, z)
17:             if right[z,y − 1] or front[x, y − 1] není objekt then
18:               Do sítě mesh přidáme spodní stěnu voxelu (x, y, z)
19:             if right[z,y + 1] or front[x, y + 1] není objekt then
20:               Do sítě mesh přidáme horní stěnu voxelu (x, y, z)
21:             if top[z − 1, x] or right[z − 1, y] není objekt then
22:               Do sítě mesh přidáme zadní stěnu voxelu (x, y, z)
23:             if top[z + 1, x] or right[z + 1, y] není objekt then
24:               Do sítě mesh přidáme přední stěnu voxelu (x, y, z)
25:   return mesh
```

Během vývoje aplikace jsem zkoušel v této fázi generovat pouze mračno bodů. Toto mračno bylo následně zjednodušeno a až na závěr triangulováno. Jelikož s sebou však mračno bodů nenese žádné informace o sousednosti, je triangulace v závěru rekonstrukce složitější. Trojúhelníková síť je navíc využita v druhé fázi algoritmu.

Při implementaci algoritmu si musíme dát pozor na převod mezi souřadným systémem obrázků a systémem souřadnic rekonstruovaného 3D modelu. U obrázků je počátek soustavy souřadnic umístěn zpravidla v levém horním rohu, ze kterého roste osa x směrem doprava a osa y směrem dolů. V případě prostorových souřadnic je pak používáno několik různých souřadných systémů. V této práci jsme si zvolili konvenci používanou standardem OpenGL: osa x míří doprava, osa y vzhůru a osa z vpřed.

3.3 Zjednodušení trojúhelníkové sítě

Pro dosažení adaptivní úrovně detailů použijeme postup inspirovaný algoritmem *Dual Contouring*. Tento algoritmus rekurzivně podrozděloval buňku obsahující celý modelovaný prostor, dokud nedosáhl požadované přesnosti. Uvnitř buněk pak generoval vrcholy trojúhelníkové sítě, které byly následně triangulovány. Algoritmus navržený v této práci naopak začíná s prostorem, který je rozdělen až na jednotlivé voxely. Buňky prostoru pak postupně slučuje za předpokladu, že jejich sloučením nepřijdeme o detaily modelu.

Algoritmus na vstupu očekává trojúhelníkovou síť, jejíž vrcholy jsou uloženy v oktantovém stromu. Na jeho postupném zjednodušování je algoritmus založen. V oktantovém stromu, který je implementován v přiložené aplikaci, navíc každý list obsahuje pouze jediný vrchol. Také budeme po použité reprezentaci trojúhelníkové sítě požadovat, aby hledání stěn, které obsahují daný vrchol, bylo efektivní. Během adaptivního zjednodušování algoritmus trojúhelníkovou síť také vyhladí.

Postupně budeme procházet vnitřní uzly oktantového stromu po hladinách od listů směrem ke kořeni a pro každý uzel se pokusíme najít vrchol, který by vhodně reprezentoval povrch procházející jeho buňkou. Pokud takový vrchol nalezneme, nahradíme tento vnitřní uzel listem a všechny vrcholy původního uzlu sloučíme s nalezeným vrcholem.

Při hledání vhodného vrcholu postupně vyzkoušíme všechny vrcholy uvnitř buňky. Pokud se v buňce nalézá ostrý prvek, některý z těchto vrcholů na něm bude s velkou pravděpodobností ležet. Podobnost povrchu před a po potencionálním zjednodušení budeme měřit pomocí *Hausdorffovy vzdálenosti*. Zjednodušený povrch je podobný původnímu, pokud každý jeho bod je blízko nějakému bodu původního povrchu a naopak. Tato metrika bere právě toto v úvahu. Jednodušší metriky nedávaly při vývoji aplikace dobré výsledky.

Pokud je tato vzdálenost menší než předem definovaná prahová hodnota, buňku zjednodušíme. V přiložené aplikaci jsme zvolili velikost poloviny úhlopříčky pixelu, algoritmus dává pro hodnotu $\sqrt{2}/2$ dobré výsledky.

Pokud listy oktantového stromu obsahují pouze jediný vrchol, tak je počet vrcholů ve vnitřních uzlech, které se pokoušíme zjednodušit, omezen na 8 vrcholů. Zjednodušujeme totiž pouze vnitřní uzly, jejichž přímí potomci jsou listy. Časová složitost algoritmu 2 je tak lineární vůči počtu vnitřních uzlů oktantového stromu. To je také $\mathcal{O}(n)$, kde n je počet vrcholů v trojúhelníkové síti.

Algoritmus 2 Zjednoduší trojúhelníkovou síť při zachování detailů modelu.

```
 1: function SimplifyMesh(mesh)
 2:     octree ← mesh.vertices          ▷ Vrcholy jsou uloženy v oktantovém stromu
 3:
 4:     for each vnitřní uzel node in octree do          ▷ Od nejhlubších po kořen
 5:         if některý syn uzlu node je vnitřní uzel then
 6:             continue              ▷ Už na nižší úrovni jsme vrchol nenalezli,
                                         nebudeme se tak o hledání pokoušet znovu
 7:
 8:         bestVertex ← průměrný vrchol uzlu node
 9:         bestDist ← HausdorffDist(node, bestVertex)
10:
11:         for each vrchol vertex in node do
12:             dist ← HausdorffDist(node, vertex)
13:             if dist < bestDist then
14:                 bestVertex ← vertex
15:                 bestDist ← dist
16:
17:         if bestDist ≤ √2/2 then
18:             uzel node nahradíme novým listem s jediným vrcholem bestVertex
19:
20:     return mesh
```

Algoritmus 3 Vrátí Hausdorffovu vzdálenost mezi povrchem před a po zjednodušení uzlu. Při zjednodušování jsou všechny vrcholy uzlu *node* sloučeny s vrcholem *vertex*.

```
 1: function HausdorffDist(node, vertex)
 2:     maxDist ← −∞
 3:     oldVertices ← vrcholy uzlu node a středy hran, které z nich vedou
 4:     oldTriangles ← trojúhelníky, jejichž vrchol leží v uzlu node
 5:     newVertices ← vertex a středy hran, které z něj po zjednodušení povedou
 6:     newTriangles ← trojúhelníky obsahující po zjednodušení vrchol vertex
 7:
 8:     for each vertex in oldVertices do
 9:         minDist ← min_{face in newTriangles}(vzdálenost vertex od face)
10:         if minDist > maxDist then
11:             maxDist ← minDist
12:
13:     for each vertex in newVertices do
14:         minDist ← min_{face in oldTriangles}(vzdálenost vertex od face)
15:         if minDist > maxDist then
16:             maxDist ← minDist
17:
18:     return maxDist
```

4 Výsledky

V této kapitole zhodnotíme kvalitu zrekonstruovaných modelů. Pro rekonstrukci modelů z kolmých průmětů byla použita přiložená aplikace, v níž je implementován algoritmus popsaný v kapitole 3. Obrázky modelů v této kapitole byly pořízeny pomocí programu Blender (verze 3.5.0).

Při promítání objektu dochází k redukci informací o jeho podobě a více různých objektů tak může mít shodné kolmé průměty. Z tohoto důvodu nemusí zrekonstruovaný model přesně odpovídat modelu, jehož kolmé průměty byly pořízeny. Povšimnout si toho můžeme například u rekonstrukce koule, jejíž výsledný model vidíme na obrázku 4.4.

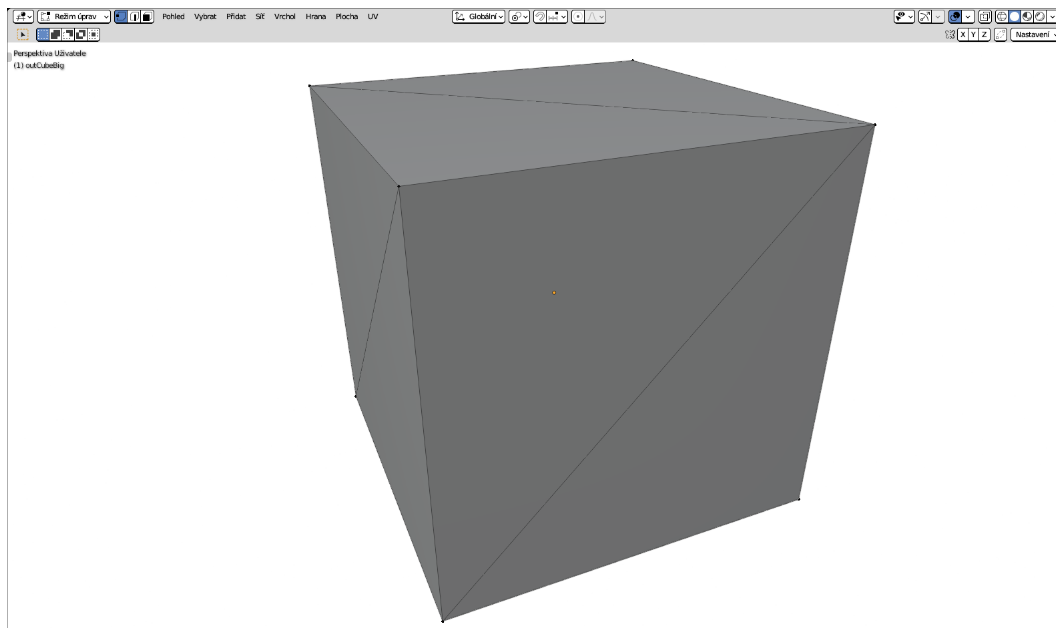
Z obrázků dále vidíme, že adaptivní mřížka použitá při rekonstrukci se úspěšně přizpůsobuje povrchu a rovné plochy modelu tak algoritmus dláždí minimem trojúhelníků. Krychle na obrázku 4.1 složená z pouhých 12 trojúhelníků je dobrým příkladem. Na obrázcích 4.2 a 4.3 však vidíme, že není potřeba, aby tyto plochy byly zarovnány s mřížkou, která byla při rekonstrukci použita. Na obrázku 4.3 si můžeme povšimnout, že algoritmus menším počtem trojúhelníků úspěšně aproximuje také oblé plochy modelu.

Největším nedostatkem algoritmu je generování velkého počtu vrcholů na hranách modelu. Toto můžeme pozorovat na obrázcích 4.2 a 4.3. Tento jev je zapříčiněn tím, že algoritmus během zjednodušování oktantového stromu nenalezl vrchol, kterým by aproximoval část povrchu. Buňky oktantového stromu tak nebyly sloučeny a vrcholy v nich nebyly nahrazeny jediným reprezentantem. Tento nedostatek můžeme napravit lepším výběrem kandidátů na aproximující vrchol, nebo rozvolněním maximální povolené vzdálenosti mezi povrchem před a povrchem po zjednodušení. Počet zvažovaných vrcholů však může mít značný dopad na dobu běhu algoritmu.

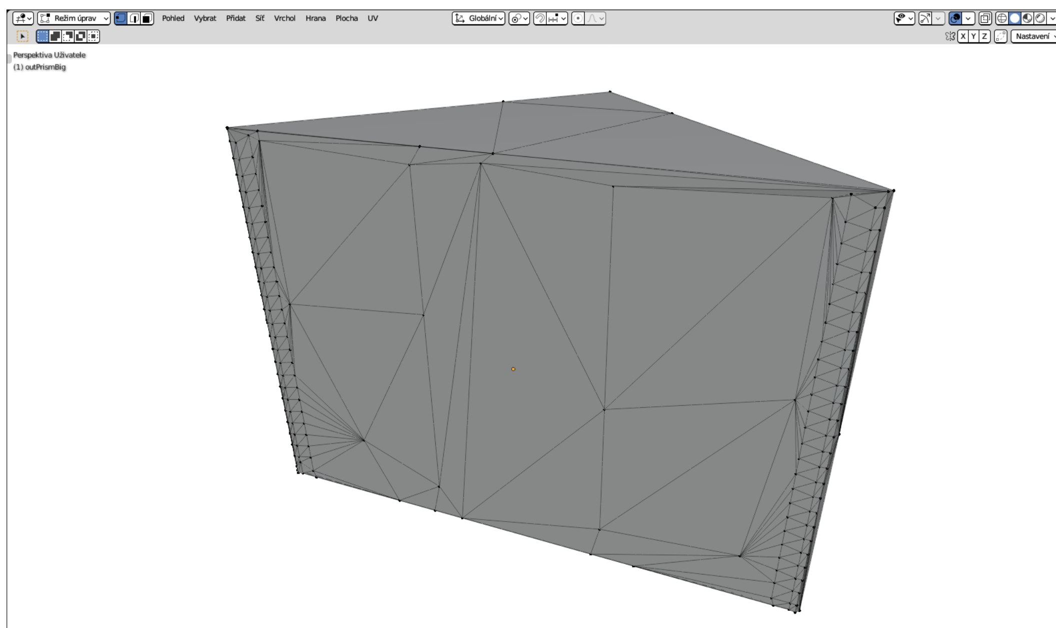
Na obrázku 4.4 dále vidíme, že trojúhelníky nerespektují hranu, která směřuje ze středu obrázku do pravého dolního rohu. Pro zvýšení kvality trojúhelníkové sítě by tak bylo potřeba na některých hranách provést operaci *prohození hrany* (*Edge flipping*). Kritérium pro výběr hran, na kterých by tato operace měla být provedena, však bohužel není tak jednoduché jako u algoritmu *Extended Marching Cubes* (Kobbelt et al., 2001).

Dalším nedostatkem algoritmu je v ojedinělých případech vytvoření záhybů povrchu a generování překrývajících se trojúhelníků. Záhyb můžeme vidět v pravé části obrázku 4.3. Algoritmus také generuje dlouhé a velmi úzké trojúhelníky. Těchto trojúhelníků se můžeme zbavit zhroucením jejich krátkých stran při post-processingu modelu. Během operace *zhroucení hrany* (*edge collapse*) jsou koncové vrcholy této hrany sloučeny v jediný a přilehlé stěny jsou odstraněny.

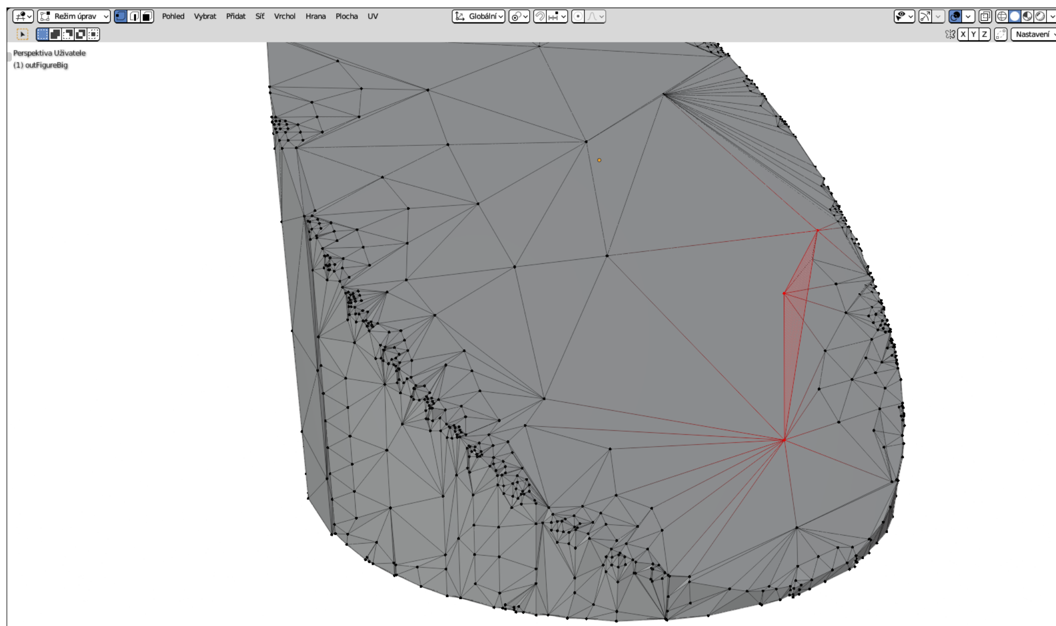
Algoritmus dosahuje i přes své nedostatky více než uspokojivých výsledků. Výsledný model je vyhlazen, přestože algoritmu stačí pouze binární data.



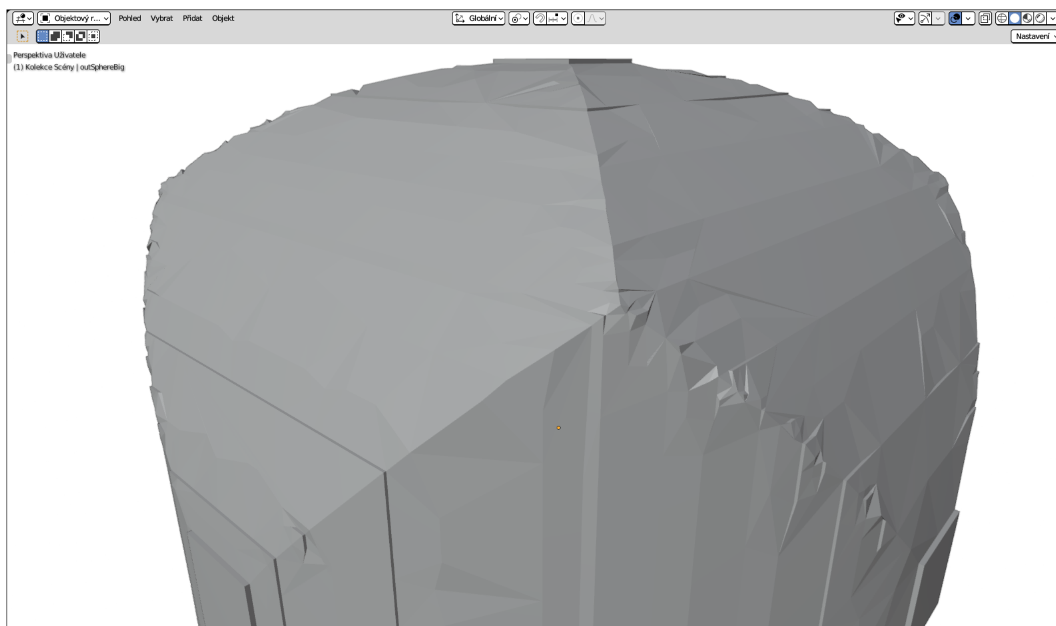
Obrázek 4.1 Krychle zrekonstruovaná z kolmých průmětů je tvořena minimem trojúhelníků.



Obrázek 4.2 Hranol zrekonstruovaný z kolmých průmětů. Stěny jsou hladké nezávisle na orientaci vůči mřížce. Algoritmus však generuje úzké trojúhelníky.



Obrázek 4.3 Algoritmus zvládá rekonstruovat také oblé plochy. Nedostatkem jsou neostré hrany, na nichž leží velké množství vrcholů. Povšimněte si také záhybu povrchu v pravé části.



Obrázek 4.4 Model plně neodpovídá kouli, jejíž kolmé průniky byly pro rekonstrukci použity. Hrana směřující do pravého dolního rohu není trojúhelníky respektována.

5 Dokumentace

Aplikace je napsána v jazyce `C#` a cílí na .NET 8.0. Aplikace dále využívá následující knihovny: ImageSharp pro práci s obrázky, Silk.NET poskytující propojení s OpenGL API a Avalonia UI, pomocí které je vytvořeno grafické uživatelské rozhraní.

Aplikace je multiplatformní a podporuje moderní verze všech hlavních desktopových operačních systémů: Windows, Linux a MacOS. Pro účely této práce byla aplikace testována na systémech Windows 10, Windows 11 a Fedora Linux 39 (Xfce).

Mezi podporovanými formáty knihovny ImageSharp jsou všechny běžně používané formáty rastrových obrázků: `png`, `jpeg`, `gif`, `bmp`, `pbm`, `tga`, `tiff`, `webp` a `qoi`. Pro rekonstrukci 3D modelu tedy můžeme použít obrázky s kolmými průměty v libovolném z těchto formátů.

Zrekonstruovaný model pak můžeme uložit do formátů: *polygon file format* (`ply`), *stereolitography file format* (`stl`) a *wavefront object file* (`obj`). Při vývoji aplikace však bylo myšleno na jednoduché rozšíření podpory ukládání 3D modelů do dalších formátů. Podporu pro ně lze dodat pomocí zásuvných modulů.

5.1 Sestavení a spuštění aplikace

Nejdříve se ujistěte, že na svém počítači máte nainstalován .NET SDK minimální verze 8.0, samotná verze .NET Runtime k sestavení aplikace nestačí. Zobrazit instalovanou verzi můžete pomocí následujícího příkazu:

```
dotnet --info
```

Pokud nemáte nainstalovánu požadovanou minimální verzi .NET SDK, stáhněte a nainstalujte ji. Jednotlivé verze .NET SDK jsou dostupné ke stažení například na stránkách společnosti Microsoft (`https://dotnet.microsoft.com/en-us/download/dotnet/8.0`).

Rozbalte archiv se zdrojovými soubory přiložené aplikace. Veškeré cesty v této dokumentaci jsou uvedeny jako relativní vůči složce, do níž je archiv rozbalen. Příkazy příkazové řádky spouštějte také z této složky. Aplikace závisí na několika externích knihovnách, pro jejichž správu je používán správce balíčků NuGet. Pro stažení externích závislostí použijte tento příkaz:

```
dotnet restore
```

Pomocí následujícího příkazu pak sestavte projekty řešení. Řešení můžete sestavit také v konfiguraci `Debug`. Tato konfigurace je však určena pouze pro ladění aplikace.

```
dotnet build --configuration Release
```

Konzolovou aplikaci spustíte následujícím příkazem, pokud používáte systém Windows:

```
ConsoleApp\bin\Release\net8.0\ConsoleApp.exe
```

A tímto příkazem, pokud používáte Linux:

```
ConsoleApp/bin/Release/net8.0/ConsoleApp
```

Konzolová aplikace dále očekává několik argumentů. Jejich použití si představíme v následující sekci.

Po sestavení spustíte aplikaci s grafickým uživatelským rozhraním pomocí následujícího příkazu:

```
dotnet run --configuration Release
        --project AvaloniaApp.Desktop/AvaloniaApp.Desktop.csproj
```

Aplikaci můžete alternativně spustit také prostřednictvím spustitelného souboru. Tento soubor je umístěn na cestě: `AvaloniaApp.Desktop/bin/Release/net8.0/AvaloniaApp.Desktop.exe`. Také můžete vytvořit zástupce spustitelného souboru a umístit jej na svou plochu.

Ve složce `Data` se nachází testovací obrázky kolmých průmětů, s nimiž můžete rekonstrukci vyzkoušet.

5.2 Konzolová aplikace

Konzolová aplikace na vstupu očekává volbu formátu 3D modelu a cesty k obrázkům s horním, pravým a čelním průmětem v tomto pořadí. U obrázků s kolmými průměty je očekáváno jednobarevné pozadí sahající až k okrajům obrázků. Žádná část rekonstruovaného objektu by se okraje obrázku neměla dotýkat. Aplikace nejprve toto pozadí odebere a následně zrekonstruuje 3D model promítaného objektu. Model je pak uložen zvoleným enkodérem do některého z formátů 3D modelů.

Pokud je rekonstruován objekt s otvorem skrz, který v kolmém průmětu nesahá k okraji obrázku, odeberte pozadí ve vašem oblíbeném programu pro úpravu obrázků. Pixely tohoto otvoru v něm nahraďte průhlednou barvou.

Pro zobrazení nápovědy s popisem jednotlivých argumentů aplikaci předejte argument `--help`.

```
usage: OrthoTo3D [options] <encoder> <imgTop> <imgRight> <imgFront>

arguments:
    <encoder>          3D model format encoder to use.
    <imgTop>           File name of top projection image.
    <imgRight>         File name of right projection image.
    <imgFront>         File name of front projection image.

options:
    -e, --encoders     Display available 3D model format encoders
                           and exit.
    -h, --help         Display this help end exit.
    -n, --noise        Radius of a noise filter in pixels. Filter
                           with a greater radius removes more noise
                           from input images, but small details
                           may be lost.
    -o, --output       Output file name.
    -q, --quiet        Do not print progress messages.
    -t, --tolerance    Color similarity tolerance in the range from 0.0
```

```
                              to 1.0. Image pixels are considered similar
                              if their color difference is less than
                              this tolerance. Pixels are compared
                              when the background of an image is removed
                              or when an object is detected
                              on a transparent background.
      -v, --version      Display version and exit.
```

Při spuštění aplikace s argumentem `--encoders` aplikace zobrazí seznam dostupných enkodérů, které lze použít pro uložení zrekonstruovaného modelu do různých formátů. Dostupné jsou následující enkodéry, pokud aplikace používá pouze zásuvné moduly, které jsou přiloženy k této práci:

```
    obj    Wavefront Object File (obj)
    ply    Polygon File Format (ply)
    stl    Stereolitography File Format (stl)
```

Při volbě formátu 3D modelu použijte jako první povinný argument aplikace klíč, který se nachází v levé části tohoto seznamu. Vpravo je pak vypsán název a přípona formátu souboru, do nějž enkodér model ukládá.

Pomocí volitelných argumentů `--noise` a `--color-tolerance` můžete přizpůsobit rekonstrukci modelu různé kvalitě vstupních obrázků. Volitelný argument `--noise` určuje celočíselný poloměr filtru šumu, který k odstranění šumu využívá morfologické operace. Vyšší hodnoty však není doporučeno používat, neboť při jejich použití mohou být ze vstupních obrázků kromě šumu odstraněny také detaily. Pokud se ve vstupních obrázcích šum nenachází a nechcete jej tak odstraňovat, předejte aplikaci tento argument s hodnotou 0.

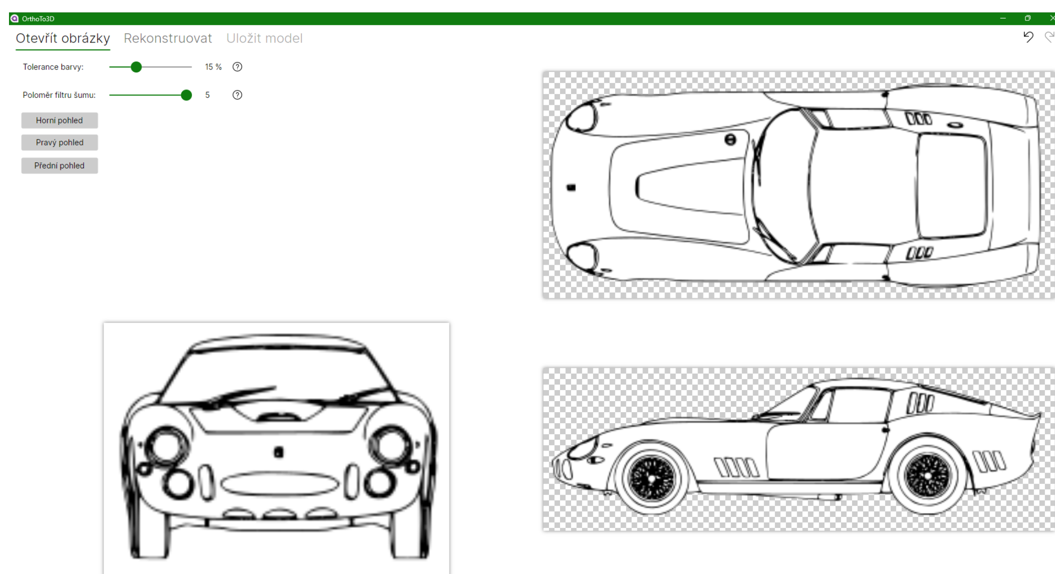
Volitelný argument `--color-tolerance` pak určuje míru tolerance podobnosti barev na škále 0.0–1.0. Za podobné jsou považovány barvy, jejichž relativní Euklidovská vzdálenost je menší než daná tolerance. Při počítání Euklidovské vzdálenosti barev jsou použity všechny jejich složky, tedy červený, zelený, modrý i alfa kanál. Absolutní Euklidovská vzdálenost je pak vydělena maximální vzdáleností mezi barvami. Při použití nulové míry tolerance jsou za podobné považovány pouze barvy shodné ve všech svých složkách. Při použití druhé krajní hodnoty jsou pak za podobné považovány všechny barvy. Hodnoty vyšší než 0.5 tak není doporučeno používat.

Podobnost barev je použita při odebírání pozadí, které se skládá z pixelů s podobnými barvami. Při rekonstrukci modelu pak algoritmus pomocí podobnosti barev rozlišuje pixely pozadí od pixelů objektu. Za pixely pozadí považuje pixely, jejichž barva je podobná průhledné černé.

5.3 Aplikace s grafickým uživatelským rozhraním

Pro rekonstrukci 3D modelu můžete použít také aplikaci s grafickým uživatelským rozhraním. Výhodou oproti konzolové aplikaci je větší kontrola nad odebíráním pozadí ze vstupních kolmých průmětů a možnost prohlédnutí zrekonstruovaného modelu. Jedná se také o intuitivnější volbu pro běžné uživatele.

Aplikace obsahuje světlý a tmavý motiv. Použitý motiv aplikace je zvolen v závislosti na nastavení systému. Veškerý text aplikace je přeložen do anglického a českého jazyka. Výchozím jazykem aplikace je angličtina. Čeština je použita, pokud je nastavena jako jazyk operačního systému.

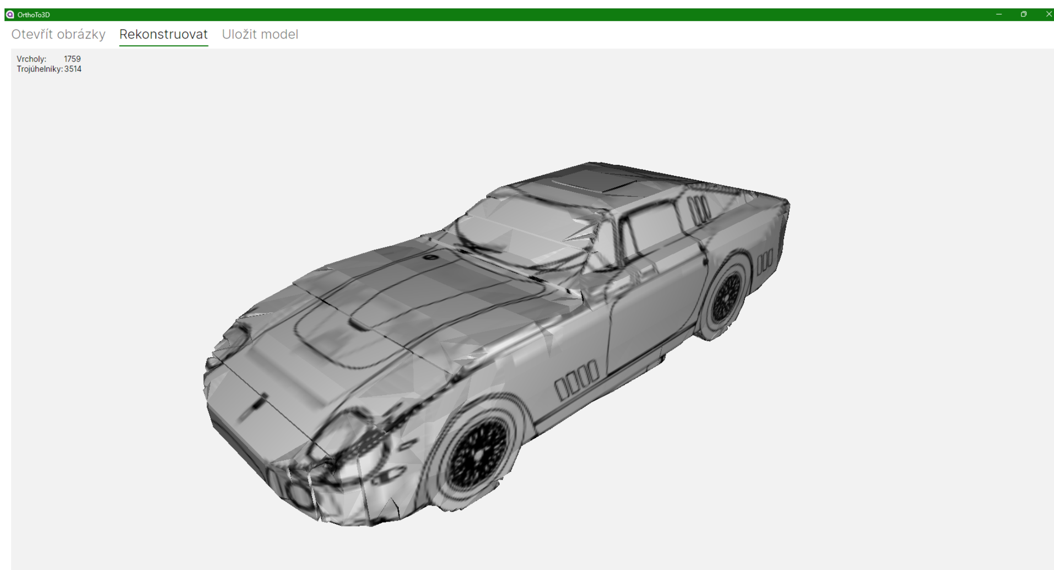


Obrázek 5.1 Obrazovka aplikace pro výběr kolmých průmětů.

Po spuštění aplikace je zobrazena obrazovka pro výběr kolmých průmětů. Tuto obrazovku vidíte na obrázku 5.1. Po kliknutí na tlačítka „Horní pohled“, „Pravý pohled“ a „Přední pohled“, která se nachází v levé horní části obrazovky, jsou otevřeny systémové dialogy pro výběr obrázků. Vyberte tedy kolmé průměty objektu, jehož model chcete rekonstruovat. Při pokusu o načtení obrázku v nepodporovaném formátu nebo jinak poškozeného obrázku je u spodního okraje obrazovky zobrazena chybová zpráva.

Po otevření se zvolené obrázky kolmých průmětů zobrazí v okně aplikace. Kliknutím myši odeberte z těchto obrázků pozadí. Pro vrácení chybně odebrané části obrázku můžete použít tlačítko „Zpět“ v pravém horním rohu obrazovky. Také lze použít klávesovou zkratku `Ctrl+Z`. Pokud pak budete chtít vrácenou akci provést znovu, použijte tlačítko „Znovu“ nebo klávesovou zkratku `Ctrl+Shift+Z`. Podobně jako v konzolové aplikaci, můžete i zde nastavit hranici podobnosti barev a poloměr filtru šumu. Efekt různých hodnot tolerance barev můžete pozorovat okamžitě při odebírání pozadí z obrázků. Pro odebrání různých částí pozadí také můžete použít rozdílné úrovně tolerance.

Po odebrání všech částí pozadí z obrázků s kolmými průměty klikněte na kartu „Rekonstruovat“. Tímto je zahájena rekonstrukce 3D modelu. Po jejím dokončení je výsledný model zobrazen, jak vidíme na obrázku 5.2. Zrekonstruovaný model si můžete prohlédnout. K jeho otáčení použijte levé tlačítko a k posunu modelu pak pravé tlačítko myši. Posouvat modelem můžete také pomocí levého tlačítka myši za současného držení klávesy `Shift`. Pro přiblížení, či oddálení modelu použijte kolečko myši. Pokud preferujete ovládání kamery pomocí klávesnice, můžete ji také použít. Pro rotaci modelu podržte některou z kurzorových šipek ↑, ↓, ← a → nebo kláves `W`, `S`, `A` a `D`. Při současném držení těchto kláves a klávesy `Shift` pak budete moci modelem posouvat. Pro přiblížení použijte klávesy `+` nebo

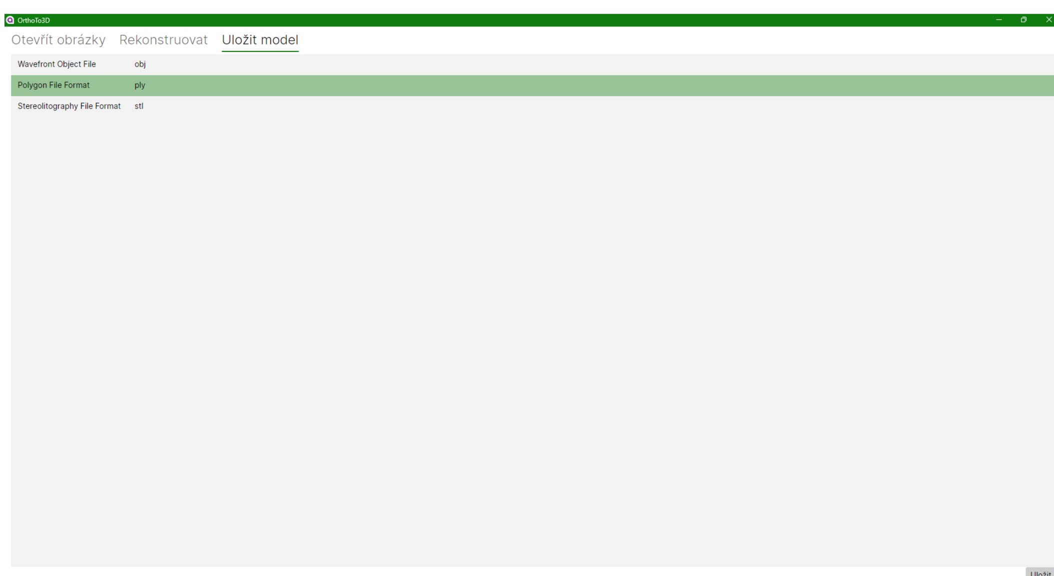


Obrázek 5.2 Prohlížeč zrekonstruovaného modelu.

`Page Up` a pro oddálení pak klávesu `-`, nebo `Page Down`. Klávesou `R` resetujete kameru do její úvodní pozice.

V levém horním rohu obrazovky se nacházejí informace o počtu vrcholů a trojúhelníků v trojúhelníkové síti zrekonstruovaného modelu. Pro stínování modelu je použito Phongovo stínování. Na model je nanesena textura, která pochází z obrázků s kolmými průměty. Tato trojice textur je interpolována dle orientace trojúhelníků, na něž je nanesena.

Pokud jste s výsledkem rekonstrukce spokojeni můžete model uložit do některého z dostupných formátů. Po kliknutí na kartu „Uložit model“ se zobrazí seznam s dostupnými formáty 3D modelů. Tuto obrazovku aplikace vidíme na obrázku 5.3. Po výběru formátu, do nějž chcete model uložit, klikněte na tlačítko „Uložit“. V systémovém dialogu pak zvolte místo pro uložení modelu.



Obrázek 5.3 Výběr formátu pro uložení modelu.

Na počítačích s operačním systémem Windows v kombinaci s AMD grafikou můžete zaznamenat nestabilitu aplikace během prohlížení zrekonstruovaného modelu. Aktualizace AMD OpenGL ovladače by měla tento problém vyřešit.

5.4 Programátorská dokumentace

Celé řešení je složeno z několika projektů, které oddělují aplikační logiku od uživatelského rozhraní. Zdrojové soubory každého z projektů se nacházejí ve stejnojmenných složkách.

5.4.1 OrthoTo3D

Projekt `OrthoTo3D` je knihovnou zajišťující klíčovou funkcionalitu aplikace. Důležitými datovými strukturami tohoto projektu jsou trojúhelníková síť, reprezentovaná třídou `Mesh`, a oktantový strom, který reprezentuje třída `Octree`. Třída `Octree` je navržena za použití návrhového vzoru *Composite*. Uzly oktantového stromu jsou reprezentovány třídami `InnerNode` a `LeafNode`. Tyto třídy implementují společný interface `IOctreeNode`. Jednotlivé uzly stromu delegují práci na své potomky bez ohledu na jejich skutečný typ.

Algoritmus popsaný v kapitole 3 je implementován v souborech, které se nacházejí ve složce `OrthoTo3D/Reconstruction`. Při sestavení aplikace v konfiguraci `Debug` je rekonstruovaný model ukládán po zjednodušení každé další úrovně oktantového stromu. Tyto ladící modely naleznete ve složce se spustitelným souborem aplikace, tedy složce `ConsoleApp/bin/Debug/net8.0/`, nebo složce `AvaloniaApp.Desktop/bin/Debug/net8.0/`.

Aplikace byla navrhována s ohledem na konvence použitých knihoven. Operace *flood fill*, *eroze* a *dilatace* jsou implementovány pomocí extension metod na interfacu `IImageProcessingContext`. To umožňuje jejich použití prostřednictvím metod `Mutate` a `Clone` stejně jako u ostatních operací s obrázky knihovny ImageSharp. Tyto extension metody pak interně používají procesory, které jim umožňují pracovat s konkrétními formáty pixelů. Zdrojové soubory operací s obrázky se nachází ve složce `OrthoTo3D/ImageProcessing/`.

Složka `OrthoTo3D/Encoders/` pak obsahuje definici interfacu `IEncoder` a enkodéru formátu *polygon file format* (`ply`). Tento enkodér je použit pro ukládání modelů během ladění algoritmu použitého pro rekonstrukci a není tak na rozdíl od ostatních enkodérů implementován jako zásuvný modul. Interface `IEncoder` musí implementovat všechny enkodéry použité aplikací.

Programátorskou dokumentaci můžete vygenerovat např. pomocí nástroje Doxygen.

5.4.2 ConsoleApp

V projektu `ConsoleApp` se nachází zdrojový kód konzolové aplikace. Argumenty aplikace jsou zpracovány v metodě `ParseArgs`. Po jejich zpracování je provedena samotná rekonstrukce modelu. Návratové kódy jsou definovány v souboru `ConsoleApp/Program.cs`.

5.4.3 AvaloniaApp a AvaloniaApp.Desktop

Grafické uživatelské rozhraní je vytvořeno pomocí frameworku Avalonia UI. Aplikace využívající Avalonii bývají složeny z multiplatformního jádra a projektů specializovaných na jednotlivé platformy.

Hlavní část uživatelského rozhraní této aplikace je definována v knihovně `AvaloniaApp`. Projekt `AvaloniaApp.Desktop` pak cílí na desktopové operační systémy Windows, Linux a MacOS. V budoucnu může být aplikace rozšířena o podporu webových prohlížečů. Zatím však není `OpenGlControl` ve webových projektech podporován.

Ve složce `AvaloniaApp/Views/` se nacházejí soubory, které určují podobu jednotlivých obrazovek aplikace. Podobně jako ve frameworku WPF, kterým se Avalonia inspiruje, je vzhled aplikace definován v XAML souborech, zatímco logika pomocí jazyka C#. Model, jenž obsahuje data využívaná v aplikaci napříč obrazovkami, je umístěn ve složce `AvaloniaApp/ViewModels/`.

Ve složce `AvaloniaApp/Controls/` jsou definovány vlastní ovládací prvky aplikace, aktuálně se zde nachází jediný, a to v podsložce `AvaloniaApp/Controls/Viewer/`. Jedná se o prohlížeč 3D modelu založený na OpenGL. Jeho základem je třída `Viewer`, která dědí od třídy `OpenGlControlBase`. Kamera, textury a OpenGL buffery a shadery jsou definovány ve vlastních třídách. Samotné OpenGL shadery napsané v jazyce GLSL a použité při vykreslování modelu se pak nacházejí v souborech `shader.vert`, `shader.geom` a `shader.frag`.

Ve složce `AvaloniaApp/Converters/` se nachází převodníky mezi různými typy. Aplikace k reprezentaci obrázků interně používá knihovnu ImageSharp a její třídu `SixLabors.ImageSharp.Image`. Před zobrazením tak musí být obrázek pomocí tohoto převodníku převeden na instanci třídy `Avalonia.Controls.Image`. Složka `AvaloniaApp/Lang/` obsahuje zdrojové soubory lokalizace textu. Ve složce `AvaloniaApp/Resources/` jsou definovány barvy, ikony a další zdroje používané aplikací. Definice vlastních stylů některých ovládacích prvků se pak nacházejí ve složce `AvaloniaApp/Styles/`.

5.4.4 StlEncoder a ObjEncoder

Enkodéry do formátů *stereolitography file format* (`stl`) a *wavefront object file* (`obj`) jsou implemenovány v projektech `StlEncoder` a `ObjEncoder`. Jedná se o zásuvné moduly, kterým se budeme více věnovat v sekci 5.5.

5.4.5 OrthoTo3DTests

V tomto projektu jsou umístěny testy ke třídě `OrthoTo3D.Octree`. Pro jejich spuštění použijte následující příkaz:

```
dotnet test
```

5.5 Zásuvné moduly

Prostřednictvím zásuvných modulů lze dodat podporu pro ukládání 3D modelů do dalších formátů bez nutnosti opětovného překladu aplikace. Pro rozšíření

aplikace umístěte `dll` soubor zásuvného modulu do složky `Plugins`, která se nachází ve složce se spustitelným souborem aplikace. Při sestavení aplikace dle této dokumentace se jedná o složku `ConsoleApp/bin/Release/net8.0/Plugins/` konzolové aplikace a složku `AvaloniaApp.Desktop/bin/Release/net8.0/Plugins/` v případě aplikace s grafickým uživatelským rozhraním.

V přiložené aplikaci jsou jako zásuvné moduly implementovány enkodéry formátů `stl` a `obj`. Při sestavování přiložené aplikace jsou zásuvné moduly přeloženy a jejich `dll` soubory jsou poté nakopírovány do složky `Plugins`. Není tak potřeba je kopírovat ručně.

5.5.1 Implementace zásuvného modulu

Při implementaci vlastního zásuvného modulu vytvořte novou knihovnu tříd. Pro kompatibilitu se zbytkem aplikace musí tato knihovna cílit na .NET 7.0, nebo .NET 8.0.

Do projektu přidejte implementaci interfacu `IEncoder`, který je definován v projektu `OrthoTo3D`. Samotné ukládání do nového formátu pak implementujte v metodě `Encode`. Enkodér je navržen za použití návrhového vzoru *Singleton*, k předání jediné instance třídy slouží vlastnost `Instance`. Dále jsou na tomto interfacu definovány vlastnosti, které poskytují základní informace o formátu souboru jako je jeho jméno a přípona. Vzorovou implementaci pluginu naleznete v souboru `ObjEncoder/ObjEncoder.cs`.

V souboru projektu je důležité povolit dynamické načtení projektu, aby jej bylo možné použít jako zásuvný modul. Také správně nastavte elementy `Private` a `ExcludeAssets`, které zajistí, že projekt `OrthoTo3D` a jeho závislosti nejsou při překladu zahrnuty ve výstupu.

```xml
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net7.0</TargetFramework>
    <EnableDynamicLoading>true</EnableDynamicLoading>
  </PropertyGroup>

  <ItemGroup>
    <ProjectReference Include="..\OrthoTo3D\OrthoTo3D.csproj">
        <Private>false</Private>
        <ExcludeAssets>runtime</ExcludeAssets>
    </ProjectReference>
  </ItemGroup>
</Project>
```

Závěr

V rámci této práce jsme vytvořili aplikaci, která úspěšně rekonstruuje 3D model z jeho kolmých průmětů. Výsledný model můžeme uložit do několika různých formátů 3D modelů. Pomocí zásuvných modulů však můžeme snadno rozšířit podporu pro ukládání i do dalších formátů.

Během vývoje aplikace jsme také navrhli algoritmus pro generování trojúhelníkové sítě z binárních voxelových dat. Tento algoritmus se adaptuje na rekonstruovaný povrch a na rovných plochách generuje malý počet trojúhelníků. Nedostatkem algoritmu je generování velkého množství vrcholů na hranách modelu. Tyto hrany pak nejsou ostré a na některých z nich je pro lepší kvalitu výsledného modelu potřeba provézt operaci prohození hrany.

Výsledné modely nemusí být, již vzhledem k nejednoznačnosti rekonstrukce, dostatečně přesné pro každé určení. Přesto může aplikace usnadnit rekonstrukci modelu a sloužit jako výchozí bod pro následné úpravy vygenerovaného modelu. Použitý algoritmus se může zařadit po bok stávajících algoritmů pro rekonstrukci povrchu z voxelové reprezentace modelu. Jeho výhodou bezesporu je, že mu stačí pouze binární data a pro úspěšnou rekonstrukci nevyžaduje žádné informace o normálách na povrch objektu.

Literatura

DOI, Akio; KOIDE, Akio, 1991. An efficient method of triangulating equi-valued surfaces by using tetrahedral cells. *IEICE TRANSACTIONS on Information and Systems*. Roč. 74, č. 1, s. 214–224.

GIBSON, Sarah F. F., 1998. Constrained elastic surface nets: Generating smooth surfaces from binary segmented data. In: WELLS, William M.; COLCHESTER, Alan; DELP, Scott (ed.). *Medical Image Computing and Computer-Assisted Intervention — MICCAI'98*. Berlin, Heidelberg: Springer Berlin Heidelberg, s. 888–898. MICCAI 1998. ISBN 978-3-540-49563-5. Dostupné z DOI: 10.1007/BFb0056277.

HECKBERT, Paul S., 1990. A seed fill algorithm. In: GLASSNER, Andrew S. (ed.). *Graphics Gems*. Boston, MA, USA: Academic Press Professional, Inc., s. 275–277. ISBN 0122861663. Dostupné z DOI: 10.1016/B978-0-08-050753-8.50058-9.

JU, Tao; LOSASSO, Frank; SCHAEFER, Scott; WARREN, Joe, 2002. Dual contouring of hermite data. In: *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*. San Antonio, Texas: Association for Computing Machinery, s. 339–346. SIGGRAPH '02. ISBN 1581135211. Dostupné z DOI: 10.1145/566570.566586.

KOBBELT, Leif P.; BOTSCH, Mario; SCHWANECKE, Ulrich; SEIDEL, Hans-Peter, 2001. Feature sensitive surface extraction from volume data. In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: Association for Computing Machinery, s. 57–66. SIGGRAPH '01. ISBN 158113374X. Dostupné z DOI: 10.1145/383259.383265.

LORENSEN, William E.; CLINE, Harvey E., 1987. Marching cubes: A high resolution 3D surface construction algorithm. In: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: Association for Computing Machinery, s. 163–169. SIGGRAPH '87. ISBN 0897912276. Dostupné z DOI: 10.1145/37401.37422.

SHU, Renben; ZHOU, Chen; KANKANHALLI, Mohan, 1995. Adaptive Marching Cubes. *The Visual Computer*. Roč. 11, s. 202–217. Dostupné z DOI: 10.1007/BF01901516.

Seznam obrázků

2.1 Základní konfigurace buněk v algoritmu *Marching Cubes*. Tyto konfigurace určují podobu části generované trojúhelníkové sítě. . . 9
2.2 Při triangulaci sousedních buněk např. dle konfigurací 3 a inverzní 6 na sebe části povrchu nenavazují. V generovaném povrchu tak mohou vzniknout trhliny. 10
2.3 Fáze algoritmu *Extended Marching Cubes*: Algoritmus *Marching Cubes* generuje trojúhelníkovou síť bez ostrých prvků. Nejdříve identifikujeme buňky obsahující ostré hrany nebo rohy (vlevo). Do těchto buněk přidáme nové vrcholy (uprostřed). Po překlopení hran získáme výslednou trojúhelníkovou síť s ostrými hranami (vpravo). (Převzato z *Feature sensitive surface extraction from volume data* Kobbelt et al. (2001)) 12
2.4 Povrch generovaný algoritmem *Marching Cubes* nad binárními daty s patrnými terasovitými artefakty (červeně). Síť vrcholů algoritmu *Surface Nets* před vyhlazováním (šedě) a po rozvolnění s minimální energií (zeleně). 13
2.5 Triangulace sítě algoritmu *Surface Nets*. Každý vrchol může být propojen s až 6 sousedy, s nimiž pak tvoří 12 trojúhelníků. Při triangulaci je však používána pouze polovina z nich. 13
2.6 Červené trojúhelníky by při závěrečné triangulaci neměli být vygenerovány. V postupu popsaném v *Constrained elastic surface nets* (Gibson, 1998) však generovány jsou. 14
2.7 Algoritmus *Marching Cubes* umisťuje vrcholy na hrany mřížky a nemůže tak reprezentovat ostrou hranu (vlevo). Algoritmus *Extended Marching Cubes* rozpozná buňky, které obsahují ostré prvky, a vygeneruje v nich další vrcholy (uprostřed). Algoritmus *Dual Contouring* pak umisťuje všechny vrcholy dovnitř buněk (vpravo). Algoritmy *Extended Marching Cubes* a *Dual Contouring* vyžadují informace o normálách na povrch v místech, kde rekonstruovaný povrch prochází mřížkou. 15
2.8 Při použití adaptivní mřížky u primárních metod mohou ve výsledném povrchu vzniknout trhliny na rozhraní buněk rozdílných velikostí. 16

4.1 Krychle zrekonstruovaná z kolmých průmětů je tvořena minimem trojúhelníků. 22
4.2 Hranol zrekonstruovaný z kolmých průmětů. Stěny jsou hladké nezávisle na orientaci vůči mřížce. Algoritmus však generuje úzké trojúhelníky. 22
4.3 Algoritmus zvládá rekonstruovat také oblé plochy. Nedostatkem jsou neostré hrany, na nichž leží velké množství vrcholů. Povšimněte si také záhybu povrchu v pravé části. 23
4.4 Model plně neodpovídá kouli, jejíž kolmé průniky byly pro rekonstrukci použity. Hrana směřující do pravého dolního rohu není trojúhelníky respektována. 23

5.1 Obrazovka aplikace pro výběr kolmých průmětů. 27
5.2 Prohlížeč zrekonstruovaného modelu. 28
5.3 Výběr formátu pro uložení modelu. 28