ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Vizualizace implicitně zadaných ploch v SageMath
Student:	Jan Koza
Vedoucí:	Ing. Tomáš Kalvoda, Ph.D.
Studijní program:	Informatika
Studijní obor:	Teoretická informatika
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce letního semestru 2016/17

Pokyny pro vypracování

1. Seznamte se s počítačovým algebraickým systémem SageMath, zejména s jeho funkcemi a rozhraními pro zobrazování 3D grafiky.
2. Nastudujte algoritmy pro numerické vykreslování implicitně zadaných ploch v 3D (SageMath k tomuto účelu používá základní Marching cubes algoritmus).
3. Rozšiřte SageMath o funkci umožňující vykreslovat více ploch konstantní hodnoty funkce tří proměnných v jednom grafu (tzv. 3D contour plot).
4. Implementujte Vámi zvolený algoritmus pro vykreslování plochy konstantní hodnoty, který v SageMath není dostupný, a porovnejte jeho vlastnosti s "Marching cubes" algoritmem.
5. Všechny Vámi implementované funkce otestujte (pomocí doc testů v dokumentaci) a vyzkoušejte na zajímavých příkladech (vytvořte vizualizace elektronových orbitálů v atomu vodíku).

Seznam odborné literatury

1. Dokumentace SageMath, doc.sagemath.org
2. L. Velho, J. Gomes, L. H. de Figueiredo, Implicit Objects in Computer Graphics, Springer, 2002

L.S.

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 26. listopadu 2015

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra teoretické informatiky

Bakalářská práce

Vizualizace implicitně zadaných ploch v SageMath

Jan Koza

Vedoucí práce: Ing. Tomáš Kalvoda, Ph.D.

9. ledna 2017

Poděkování

Děkuji svému vedoucímu bakalářské práce Ing. Tomáši Kalvodovi, Ph.D. za cenné rady, připomínky a čas, který věnoval pravidelným konzultacím.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 9. ledna 2017 .....

Odkaz na tuto práci

Koza, Jan. *Vizualizace implicitně zadaných ploch v SageMath.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Tato bakalářská práce se zabývá algoritmy pro numerické vykreslování implicitně zadaných ploch. Porovnává vlastnosti vybraných algoritmů a uvádí jejich výhody a nevýhody. Podrobněji zkoumá algoritmus *Dual Marching Cubes*. Součástí práce je implementace tohoto algoritmu v open-source algebraickém systému SageMath a porovnání s algoritmem *Marching Cubes*, který je v systému dostupný. Implementované funkce jsou použity k vytvoření vizualizace elektronových orbitalů vodíku a prezentovány na dalších zajímavých příkladech implicitních ploch.

Klíčová slova Implicitní plocha, Polygonizace, Dual Marching Cubes, Marching Cubes, SageMath

Abstract

This bachelor's thesis is focused on algorithms for numerical polygonization of implicit surfaces. It compares properties of selected algorithms and specifies their advantages and drawbacks. In greater detail, it examines the algorithm *Dual Marching Cubes*. This work includes implementation of this algorithm in open-source algebraic system SageMath and compares it with the *Marching Cubes* algorithm, which is already present in the system. Implemented functions are used to visualize electron orbitals of hydrogen atom and presented on other interesting examples of implicit surfaces.

Keywords Implicit surface, Polygonization, Dual Marching Cubes, Marching Cubes, SageMath

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Plochu v trojrozměrném prostoru lze matematicky definovat třemi základními způsoby. Explicitně, parametricky a implicitně [1]. Explicitní vyjádření plochy pomocí grafu funkce má tvar

$$z = f(x, y),$$

kde x, y probíhají jistou podmnožinou $\mathbb{R}^2$ a z je třetí kartézská souřadnice v $\mathbb{R}^3$. Parametrickou plochu určuje soustava rovnic pro jednotlivé souřadnice s dvojicí parametrů $(u, v) \in \mathbb{R}^2$:

$$\begin{aligned} x &= f_x(u, v), \\ y &= f_y(u, v), \\ z &= f_z(u, v). \end{aligned}$$

Aby parametrická plocha byla spojitá, musí být spojité i funkce f_x, f_y a f_z. Implicitní plocha je definována rovnicí

$$f(x, y, z) = 0,$$

kde na pravé straně může být obecně libovolná konstanta. Bez újmy na obecnosti lze předpokládat, že se jedná o 0. Plocha konstantní hodnoty funkce se také nazývá izoplocha. Každý z těchto zápisů má své výhody i nevýhody a pro zobrazení ploch zadaných různými způsoby se používají jiné přístupy. Tyto vyjádření ploch obecně nemusí být navzájem převoditelné. Například sféru nelze celou popsat pomocí grafu funkce. Tato bakalářská práce se bude zabývat implicitně zadanými plochami a různými algoritmy pro jejich vykreslování.

Na začátku této práce popíšeme různé algoritmy sloužící k polygonizaci implicitně zadaných ploch. Větší pozornost budeme věnovat algoritmu *Marching Cubes*, který byl publikován skoro před 30 lety. Zmíníme i další algoritmy a porovnáme jejich vlastnosti, výhody a nevýhody. Uvedeme také, jak lze tyto plochy vykreslit v některých počítačových algebraických systémech.

V další kapitole se budeme podrobně zabývat algoritmem *Dual Marching Cubes* a postupně probereme jeho hlavní části. Ve třetí kapitole se zaměříme na implementaci tohoto algoritmu v SageMath, zejména na objektový návrh, optimalizace a testování. Poslední kapitola bude prezentovat použití algoritmu na zajímavých příkladech a porovná algoritmy *Dual Marching Cubes* a *Marching Cubes*. Na konci použijeme naši implementaci k vytvoření vizualizace elektronových orbitalů.

Kapitola 1

Algoritmy pro polygonizaci implicitně zadaných ploch

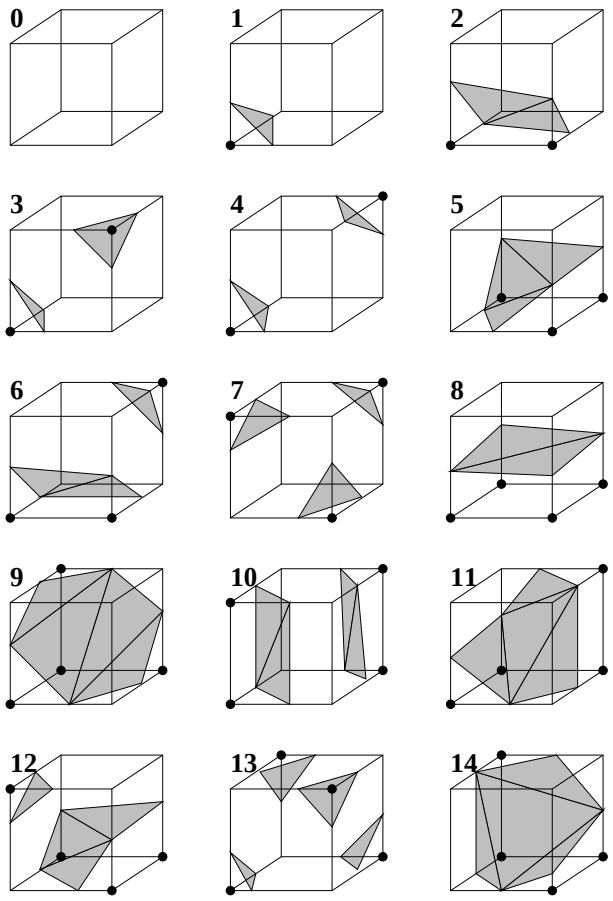
1.1 *Marching Cubes*

Marching Cubes, který je popsaný v článku [2], je známý a často používaný algoritmus sloužící ke konstrukci polygonální sítě z izoplochy trojrozměrného skalárního pole, tj. funkce $\mathbb{R}^3 \to \mathbb{R}$. Byl publikován v roce 1987 a jeho autory jsou William E. Lorensen a Harvey E. Cline. Svého času byl patentován a platnost tohoto patentu [3] vypršela v roce 2005. To umožnilo rozšíření různých implementací, včetně té v systému SageMath. Algoritmus byl vyvinut za účelem zobrazovaní snímků z výpočetní tomografie a magnetické rezonance.

Název *Marching Cubes* lze přeložit jako pochodující krychle, což naznačuje princip tohoto algoritmu. *Marching Cubes* využívá přístupu rozděl a panuj. Na začátku se rovnoměrně rozdělí prostor, pro který chceme vykreslit plochu, na určitý počet stejně velkých krychlí. Počet krychlí je parametr tohoto algoritmu a přímo určuje podrobnost výsledku. Algoritmus poté zkoumá každou krychli zvlášť a určuje, jakým způsobem ji zobrazovaná plocha protíná. V každém vrcholu krychle se spočítá hodnota zkoumané funkce tří proměnných a přiřadí se mu hodnota 0 nebo 1. Pokud je hodnota funkce větší nebo rovna zadané konstantě odpovídající vykreslované izoploše, uloží se pro vrchol hodnota 1, v opačném případě 0.

Každý vrchol krychle může nabývat dvou různých hodnot. Celkově tedy dostáváme $2^8 = 256$ možných ohodnocení jedné krychle. Podle tohoto ohodnocení je možné rozhodnout, jak a zda vůbec plocha krychlí prochází a které její hrany protíná. Pokud očíslujeme vrcholy a hrany jedné krychle, můžeme vytvořit tabulku, která přiřadí každé variantě množinu trojic protnutých hran. Všechny tyto konfigurace nejsou topologicky unikátní. Lorensen a Cline ukázali, že je možné každou z nich převést na jeden z 15 případů vyobrazených na obrázku 1.1. Převod lze uskutečnit prohozením 1 a 0 nebo rotací krychle. V případech, kdy mají všechny vrcholy stejnou hodnotu, se v dané krychli žádná

plocha nevykreslí. To nastane, pokud celá krychle leží nad nebo pod úrovní plochy. Může k tomu také dojít, pokud plocha prochází vnitřní částí krychle, ale vyhne se všem vrcholům. V takovém případě algoritmus plochu neodhalí. Pro ostatní konfigurace platí, že plocha se vyskytuje uvnitř krychle, a proto generují 1 až 4 výsledné trojúhelníky. Při implementaci algoritmu se používá tabulka se všemi 256 případy. V tabulce jsou uloženy množiny trojic hran, které plocha protíná. K získání souřadnic vrcholů výsledného trojúhelníku se použije lineární interpolace podle hodnot funkce v krajních bodech příslušné hrany. Vrcholy tak mohou ležet pouze na hranách jednotlivých krychlí.



Obrázek 1.1: Výsledné polygony pro 15 různých ohodnocení vrcholů krychlí. Vrchol polygonu nemusí ležet uprostřed hrany. Konkrétní poloha se určí lineární interpolací. Převzato z článků Evgenie Chernyaeva [4]

Tento algoritmus má několik nevýhod. V prvé řadě pro celou zobrazovanou oblast pracuje se stejnou přesností, která je určena rozměrem krychlí. To znamená, že rozděluje část povrchu, která je téměř rovná, stejně podrobně jako část, která je výrazně členitá. Rozlišovací schopnost algoritmu je přímo

daná velikostí krychlí. Pokud bychom například vykreslovali nesouvislou plochu, která se skládá z více částí, tak algoritmus neodhalí části menší než je velikost krychlí. Další špatnou vlastností je, že v některých případech je určení polygonu nejednoznačné. Ve spojitých plochách se tak mohou tvořit mezery, anebo naopak může být generován povrch tam, kde není.

Jeho výhodou je snadná paralelní implementace. Článek [5] jej uvádí jako téměř „embarrassingly parallelizable“, což doslova znamená trapně paralelizovatelný. Takto se označuje skupina algoritmů, u kterých lze výpočet rozdělit na mnoho vzájemně nezávislých částí, které nesdílejí data a nepotřebují synchronizaci. Jedinou společnou částí je zde ukládání výsledných trojúhelníků do sdíleného seznamu. Tento problém lze poměrně jednoduše vyřešit rozdělením seznamu do bloků pro každé výpočetní vlákno.

1.2 *Surface Nets*

Algoritmus *Surface Nets* volí jiný přístup k polygonizaci dvourozměrné plochy v trojrozměrném prostoru. Byl popsán v roce 1998 Sarah F. F. Gibsononvou v článku [6] a je prvním algoritmem z kategorie duálních metod. Ty stejně jako *Marching Cubes* a další navazující algoritmy rozdělují prostor na menší buňky, které se dále zkoumají nezávisle. Liší se však způsobem umístění jednotlivých vrcholů výsledných mnohoúhelníků představujících plochu. Neomezují se pouze na hrany příslušných buněk, ale mohou umístit vrchol kdekoliv uvnitř.

V případě *Surface Nets* jsou dílčí buňky také krychle, ale obecně to mohou být i jiné mnohostěny. Tento algoritmus také rovnoměrně rozdělí prostor do mřížky a ohodnotí vrcholy 1 a 0 stejně jako u *Marching Cubes*. Dále zkoumá jen ty krychle, kde se alespoň jedna hodnota ve vrcholu liší od ostatních. Tedy ty buňky, kterými prochází vykreslovaná plocha. Poté se do středu každé povrchové buňky umístí jeden vrchol a spojením vrcholů sousedních buněk se vytvoří síť, kde každé dvě sousední hrany svírají úhel 90° nebo 180°. Dále algoritmus síť iterativně vyhlazuje. Každému vrcholu se přiřadí hodnota, kterou autorka nazývá energií. Tato hodnota se vypočítá jako součet druhých mocnin vzdáleností od všech sousedních uzlů. Lze však použít i jiné metriky a dosáhnout tak mírně odlišných výsledků. V každé iteraci se vrcholy posunou tak, aby se jejich energie minimalizovala. Při použití uvedené metriky toho lze jednoduše dosáhnout posunutím vrcholu tak, aby si všechny vzdálenosti k sousedním uzlům byly rovny. Pokud bychom minimalizaci nijak neomezili, stala by se ze sítě postupně kulová plocha a nakonec by konvergovala do jediného bodu. Proto je nutné omezit polohu vrcholu tak, že musí zůstat ve své buňce. Počet kroků iterace se nastaví podle požadované přesnosti.

1.3 *Dual Contouring*

Dalším zástupcem duálních metod je algoritmus *Dual Contouring* [7]. Liší se od předchozích algoritmů tím, že dokáže zachytit i ostré hrany ploch. Výše uvedené metody povrch vyhlazují, a tak všechny ostré části zanedbají. Tento algoritmus však klade omezení na vstupní data. Vykreslovaná funkce musí mít v každém bodě definovaný gradient. Výše zmíněné algoritmy je možné použít i na funkce zadané tabulkou hodnot, například na data naměřená tomografem. Abychom mohli použít *Dual Contouring* musíme znát kromě hodnot funkce také derivace podle každé proměnné nebo lze počítat alespoň jejich aproximace.

Aby algoritmus podrobně zkoumal pouze ta místa, kde je kontura dané funkce, využívá k rozdělení buněk oktalový strom. Do každé buňky se umístí jeden vrchol podobně jako u *Surface Nets*. Jeho poloha se však zjistí minimalizací kvadratické chybové funkce, která zajistí zachování ostrých částí plochy. Myšlenku tohoto algoritmu dále rozvádí algoritmus *Dual Marching Cubes*, který jsme nakonec zvolili pro implementaci a budeme se mu důkladně věnovat v následujících dvou kapitolách.

1.4 Algebraické systémy

Vizualizace implicitně zadaných ploch je možné vytvořit pomocí několika specializovaných programů i obecných algebraických systémů. Zaměříme se zde na nejrozšířenější algebraické systémy a stručně popíšeme, které funkce k vykreslování implicitních ploch nabízejí.

1.4.1 Mathematica

Mathematica je software od společnosti Wolfram Research, který využívá vlastní programovací jazyk Wolfram Language. Umožňuje zobrazování implicitně zadané plochy pomocí funkce `ContourPlot3D`. Dle oficiální dokumentace [8] je nejdříve spočítána hodnota zadané funkce pro několik rovnoměrně rozprostřených bodů. Pro zjemnění vykreslené plochy je použit adaptivní algoritmus, který počítá hodnoty funkce v okolí těchto bodů. Jedná se tedy pravděpodobně o algoritmus založený na *Marching Cubes*.

1.4.2 Maple

Program Maple, který je vyvíjen firmou Maplesoft, je také schopný vizualizace implicitně zadané plochy pomocí funkce `implicitplot3d`. Dokumentace [9] však neuvádí použitý algoritmus.

1.4.3 Matlab

Systém Matlab sám o sobě nedokáže vykreslit plochu zadanou implicitně. Na oficiálních stránkách [10] je však možné najít skript, který tuto funkci zajišťuje.

1.4.4 SageMath

SageMath obsahuje funkci `implicit_plot3d` [11], která slouží právě k vykreslování implicitních ploch. K výpočtu hodnot funkce je použit algoritmus *Marching Cubes*, u kterého je možné určit počet krychlí a tím ovlivnit přesnost zobrazení výsledné plochy.

Kapitola 2

Algoritmus *Dual Marching Cubes*

Algoritmus *Dual Marching Cubes* publikovaný roku 2004 Scottem Schaeferem a Joem Warrenem [12] kombinuje přístup *Marching Cubes* a duálních metod. Postup algoritmu je možné rozdělit do tří kroků. Nejprve se vytvoří oktalový strom, který rozdělí prostor na různě velké krychle. Poté se sestaví mřížka, která je topologicky duální k oktalovému stromu. Do této duální mřížky se vygenerují trojúhelníky pomocí zobecněného algoritmu *Marching Cubes*.

2.1 *Quadratic Error Function*

Nejprve je potřeba zadefinovat kvadratickou chybovou funkci, která se v tomto algoritmu vyhodnocuje pro každou krychli oktalového stromu. *Quadratic Error Function* (dále jen QEF) je funkce definovaná předpisem

$$\mathrm{QEF}(w,x,y,z) = \sum_i \frac{(w - T_i(x,y,z))^2}{1 + |\nabla f(x_i,y_i,z_i)|^2}, \tag{2.1}$$

kde se sčítá přes několik vzorků uvnitř konkrétní krychle a

$$T_i(x,y,z) = \nabla f(x_i,y_i,z_i) \cdot ((x,y,z) - (x_i,y_i,z_i)) + f(x_i,y_i,z_i). \tag{2.2}$$

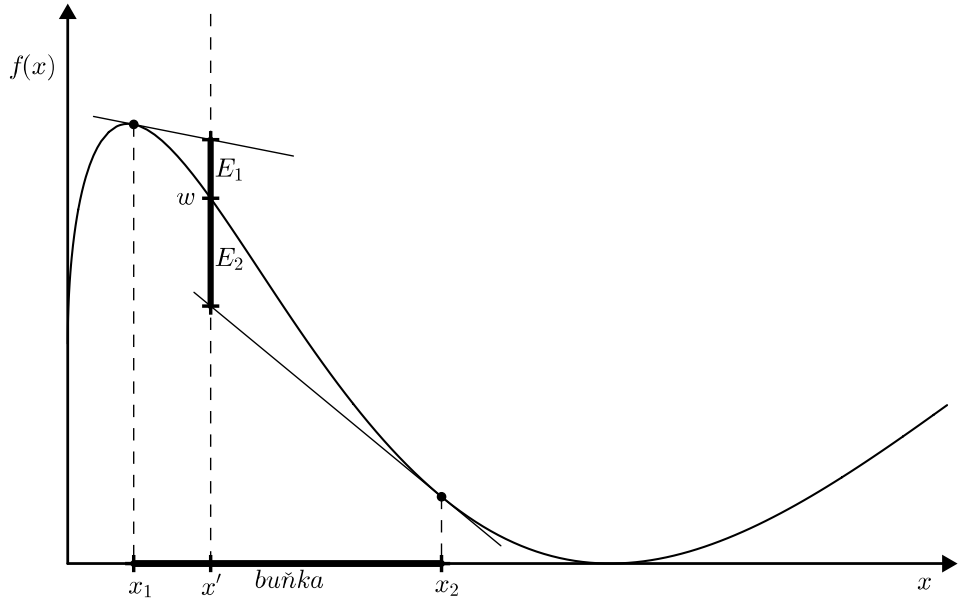
Index i odpovídá vzorkovacímu bodu (x_i, y_i, z_i) a $|\boldsymbol{v}|$ značí normu vektoru. *Dual Marching Cubes* využívá QEF ke zjištění polohy vrcholu duální mřížky uvnitř krychle. Abychom mohli sestavit tuto funkci, musíme nejprve spočítat tečné roviny grafu funkce $f(x,y,z)$ v několika bodech uvnitř buňky. Je možné volit různý počet a různé rozložení vzorkovacích bodů. V této práci jsme použili rovnoměrné vzorkování. Důležitým faktem je, že netvoříme tečnou rovinu k zobrazované implicitní ploše, ale přímo ke grafu funkce. Graf funkce tří proměnných je čtyřrozměrný, proto přesnější označení pro tuto tečnou rovinu je nadrovina dimenze 3 v prostoru dimenze 4.

Funkce určující tečnou rovinu T_i v bodě (x_i, y_i, z_i) je uvedena v předpisu (2.2). ∇f značí gradient funkce f. Dosazením bodu (x_i, y_i, z_i) do gradientu získáme vektor, který se skládá z hodnot parciálních derivací funkce v daném bodě:

$$\left(\frac{\partial f}{\partial x}(x_i, y_i, z_i), \frac{\partial f}{\partial y}(x_i, y_i, z_i), \frac{\partial f}{\partial z}(x_i, y_i, z_i)\right)$$

Tento vektor odpovídá směru největšího růstu funkce a určuje sklon tečné roviny. Skalární součin s vektorem (x, y, z) vytvoří nadrovinu s daným normálovým vektorem a odečtení bodu se souřadnicemi (x_i, y_i, z_i) posune nadrovinu po osách x, y a z do tohoto bodu. Přičtení hodnoty $f(x_i, y_i, z_i)$ nakonec posune nadrovinu ve směru 4. souřadnice tak, aby byla tečnou rovinou grafu. V původním článku [12] tento poslední absolutní člen chybí. Domníváme se, že je to chyba, a proto v této práci používáme zde uvedenou funkci, která skutečně popisuje tečnou rovinu.

Výsledná hodnota QEF se získá sečtením druhých mocnin rozdílů hodnot aproximovaných tečnou rovinou a hodnoty w, která je odhadem hodnoty funkce f. Jmenovatel výrazu pak normalizuje váhy jednotlivých odchylek podle sklonu tečné roviny.



Obrázek 2.1: Zjednodušená jednorozměrná ukázka QEF. E_1 a E_2 jsou rozdíly mezi hodnotou w a hodnotami odhadnutými pomocí tečen grafu $f(x)$ v bodech $f(x_1)$ a $f(x_2)$.

Použití QEF si ukážeme na zjednodušeném příkladu funkce jedné proměnné na obrázku 2.1. Buňka ve které počítáme hodnotu QEF je v tomto

případě interval na ose x. Pro názornost jsme jako místa vzorkování zvolili krajní body tohoto intervalu x_1 a x_2. Hodnotu x' jsme zvolili libovolně uvnitř buňky a w umístili do $f(x')$. Vzdálenosti mezi w a hodnotami aproximovanými tečnami jsme označili E_1 a E_2. Výpočet QEF pro tento příklad je

$$\mathrm{QEF}(w, x') = \frac{E_1^2}{1 + f'(x_1)^2} + \frac{E_2^2}{1 + f'(x_2)^2} = \frac{(w - t_1(x'))^2}{1 + f'(x_1)^2} + \frac{(w - t_2(x'))^2}{1 + f'(x_2)^2},$$

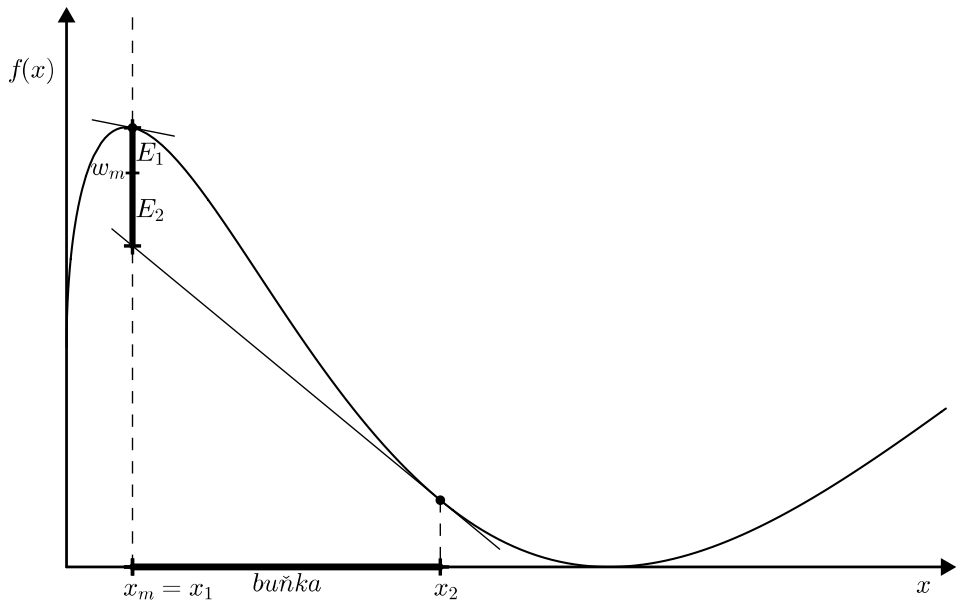
kde

$$t_i(x) = f'(x_i)(x - x_i) + f(x_i).$$

Geometrický význam této chybové funkce spočívá v přiřazení vyšší hodnoty buňce, ve které je funkce zakřivená. Naopak částem funkce, které jsou skoro lineární, přiřadí QEF nízké hodnoty.

2.2 Minimalizace QEF

Algoritmu *Dual Marching Cubes* nestačí pouze zjistit hodnotu QEF pro určitý bod, ale potřebuje najít minimum této funkce v dané buňce. Pro znázornění minimalizace na obrázku 2.2 použijeme stejnou funkci jako v předchozím příkladu.



Obrázek 2.2: Minimalizace zjednodušené $\mathrm{QEF}(w, x)$. Minima tato funkce nabývá pro dvojici (w_m, x_m).

Dvojici, ve které se nachází minimum QEF, označíme jako (w_m, x_m). Aby byla hodnota QEF minimální, musí být rozdíl w_m a hodnot aproximovaných

tečnami co nejmenší. V tomto příkladu bychom mohli nalézt dvojici, pro kterou je QEF nulová. Ta by odpovídala bodu, kde se tečny protínají a nacházela by se vně buňky. Aby algoritmus dosahoval smysluplných výsledků, omezuje minimalizaci na vnitřek buňky stejně jako jiné duální metody. Proto x_m posuneme na okraj buňky do x_1, kde je vzdálenost tečen nejmenší. Minimum získáme tak, že hodnotu w_m umístíme mezi hodnoty odhadnuté tečnami v poměru jmenovatelů příslušných sčítanců v QEF.

$$w_m = t_2(x_m) + \frac{1 + f'(x_1)^2}{1 + f'(x_2)^2}(t_1(x_m) - t_2(x_m))$$

Na tomto jednoduchém příkladu jsme si ukázali, jaký má QEF význam a jak zde nalézt minimum. Při reálném použití se chybová funkce sestaví z mnoha tečných rovin a je tak mnohem složitější. *Dual Marching Cubes* proto musí umět minimalizovat tuto funkci v obecném případě. Mohli bychom funkci iterativně optimalizovat a najít tak její přibližné minimum numericky. Tento přístup je ale velice neefektivní. Použijeme proto analytické řešení, které vychází z článků [13] a [14], na které odkazují autoři *Dual Marching Cubes*. Využijeme faktu, že chybová funkce je vždy polynom stupně nejvýše 2 a lze ji tak vyjádřit ve tvaru, ve kterém se zapisuje tzv. kvadrika:

$$\mathrm{QEF}(x_1, x_2, x_3, x_4) = \sum_{i,j=1}^{4} x_i Q_{ij} x_j + \sum_{i=1}^{4} P_i x_i + R$$

nebo zkráceně

$$\mathrm{QEF}(x) = xQx^{\mathrm{T}} + Px^{\mathrm{T}} + R,$$

kde $x = (x_1, x_2, x_3, x_4)$ je řádkový vektor, Q je matice 4×4 odpovídající koeficientům kvadratických členů, P je řádkový vektor koeficientů lineárních členů se 4 složkami a R je absolutní člen (konstanta).

V případě, že matice Q není nulová, je chybová funkce kvadratická. Protože jednotlivé sčítance tvořící QEF jsou větší nebo rovny 0 (viz (2.1)), platí $\mathrm{QEF}(x) \geq 0$ pro každý vektor x. Potom můžeme říct, že minimum se nachází tam, kde je gradient funkce nulový vektor. Jelikož matice Q je vždy symetrická, můžeme gradient chybové funkce vyjádřit jako

$$\nabla \mathrm{QEF}(x) = 2xQ + P.$$

Vektor x, který minimalizuje QEF je řešením lineární soustavy rovnic

$$xQ = -\frac{1}{2}P.$$

Může se však stát, že rovnice má nekonečně mnoho řešení a minimum chybové funkce tak není jednoznačně určeno. To nastane, pokud je matice Q singulární. V takovém případě postupujeme podle článku [14] a použijeme pseudoinverzní matici Q^+. Konkrétní vektor minimalizující QEF získáme vzorcem

$$x_m = x_c + Q^+(P - x_c Q),$$

který vybere takové minimum, které je nejblíže středu x_c zkoumané buňky.

2.3 Konstrukce oktalového stromu a výpočet vrcholů duální mřížky

Oktalový strom (anglicky *octree*) je stromová datová struktura, kde každý vnitřní uzel má právě 8 potomků. Je to tedy plný 8-ární strom, který se používá pro hierarchickou reprezentaci 3D objektů. Jeho konstrukce probíhá rekurzivním dělením prostoru do 8 stejně velkých částí – oktantů.

Algoritmus začne tím, že vytvoří oktalový strom s jediným uzlem, který zůstane kořenem stromu. Tento uzel odpovídá buňce tvaru kvádru nebo krychle, která obsahuje celý vykreslovaný prostor. Buňka se dále rekurzivně dělí na základě následujícího pravidla. Každá buňka se navzorkuje a vytvoří se pro ni QEF(w, x, y, z) tak, jak jsme ji popsali v předchozí části, a nalezne se její minimum. Dále se použije jak čtveřice (w_m, x_m, y_m, z_m), která danou funkci minimalizuje, tak minimum funkce v tomto bodě, které určuje velikost chyby při aproximaci pomocí tečných rovin. Pokud je chyba větší než zadané ϵ, buňka se rozdělí a rekurze pokračuje. Hodnota ϵ je vstupní parametr algoritmu a určuje jemnost dělení. Pokud je chyba menší, buňka se stane listem oktalového stromu. Pro každý list se uloží bod (x_m, y_m, z_m) a hodnota w_m. Bod (x_m, y_m, z_m) se stane vrcholem duální mřížky a hodnota w_m určuje odhad hodnoty funkce f v tomto bodě.

Algoritmus 1 Pseudokód konstrukce oktalové stromu

```
Require: ε > 0
1: procedure EXAMINECELL(CELL)
2:     cell.octants ← split(cell)            ▷ split rozdělí buňku na 8 oktantů
3:     for each octant in cell.octants do:
4:         error, aproxValue, vertex ← minimizeQEF(octant)
5:         if error < ε then:
6:             octant.vertex ← vertex
7:             octant.aproxValue ← aproxValue
8:         else:
9:             examineCell(octant)
```

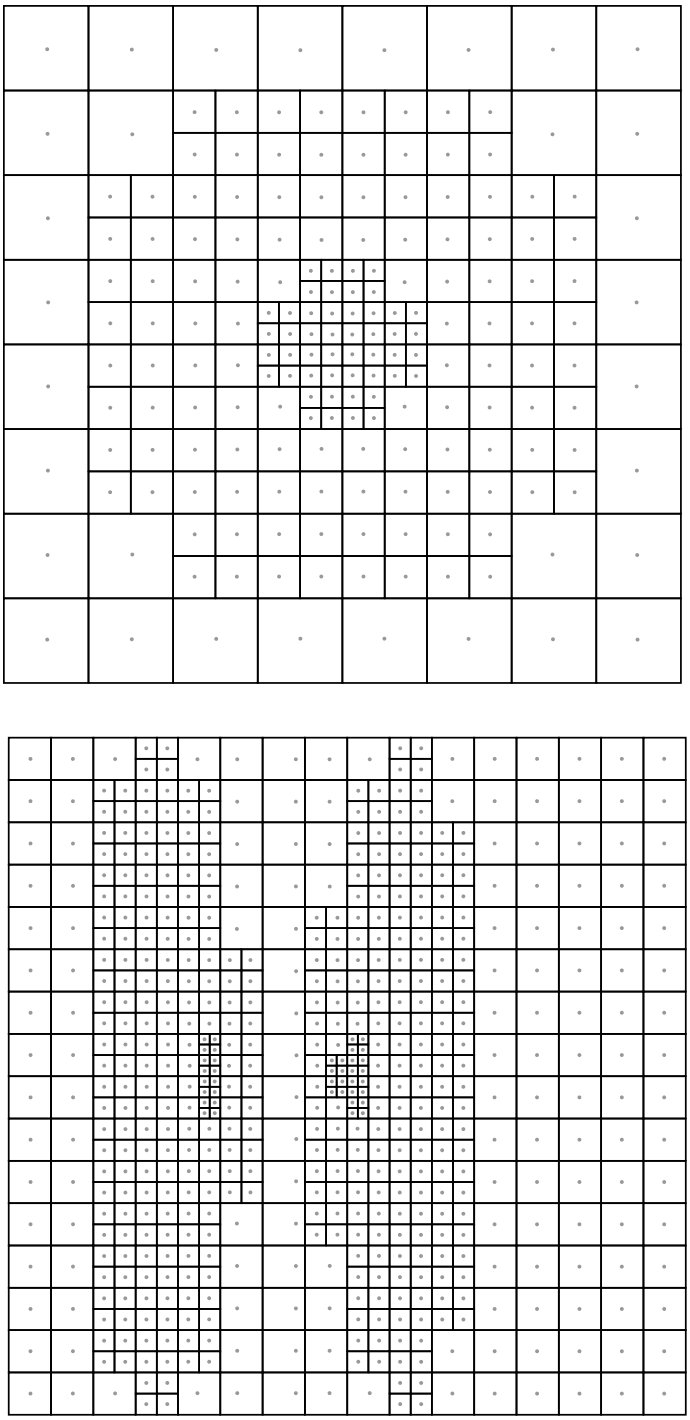
Uvedený algoritmus funguje dobře, pokud vhodně zvolíme hodnotu ϵ. To je však v některých případech velice obtížné. Výsledek Chybové funkce totiž závisí na vstupní funkci, na rozsahu vykreslovaného prostoru i na počtu vzorkovacích bodů. Při testování algoritmu na nevšedních funkcích jsme narazili na případy, kdy bylo třeba nastavit ϵ na 10^4 a jindy na 10^{-16}. Proto není možně zvolit výchozí hodnotu ϵ, která by vyhovovala všem vstupům. Pokud zvolíme příliš velké ϵ, bude výsledek až příliš zjednodušený a nebude odpovídat skutečnému tvaru implicitní plochy. Pokud naopak tomuto parametru nastavíme příliš nízkou hodnotu, zastaví se dělení stromu, až když dosáhne maximální hloubky. Tím přijdeme o výhodu *Dual Marching Cubes*, protože

plochu budeme zkoumat ve všech místech stejně podrobně a výsledek bude tvořen velkým množstvím polygonů.

Abychom předešli obtížné volbě parametru ϵ, navrhli jsme alternativní postup konstrukce oktalového stromu. K rozhodování, jestli buňku dále dělit, využíváme minimalizaci QEF stejně jako u původního algoritmu. Buňky postupně vkládáme do prioritní fronty podle hodnoty minimalizované chybové funkce a dělíme tak nejdříve ty, které mají největší odchylku. Parametrem, který určuje podrobnost výsledné plochy, je zde maximální počet listů oktalového stromu. Toto číslo už je snazší odhadnout, protože přímo ovlivňuje počet polygonů. Pokud například zdvojnásobíme maximum listů, počet výsledných polygonů bude také přibližně dvojnásobný.

Algoritmus 2 Pseudokód alternativní konstrukce oktalové stromu

```
Require: PriorityQueue pq, maxLeaves ∈ n, maxDepth ∈ n
 1: procedure enqueueOctants(cell)
 2:     for each octant in cell.octants do:
 3:         error, aproxValue, vertex ← minimizeQEF(octant)
 4:         octant.error ← error
 5:         octant.aproxValue ← aproxValue
 6:         octant.vertex ← vertex
 7:         pq.enqueue(octant, error)
 8: procedure examineCellPQ(cell)
 9:     cell.octants ← split(cell)
10:     enqueueOctants(cell)
11:     leaves ← 0
12:     while pq is not empty and (leaves + length of pq) < maxLeaves do:
13:         top ← pq.dequeue()
14:         for each next in pq do:
15:             if top.error ≈ next.value.error then:
16:                 next.priority ← ∞
17:             else:
18:                 break
19:         if top.depth < maxDepth then:
20:             cell.octants ← split(cell)
21:             enqueueOctants(cell)
22:         else:
23:             leaves ← leaves + 1
```

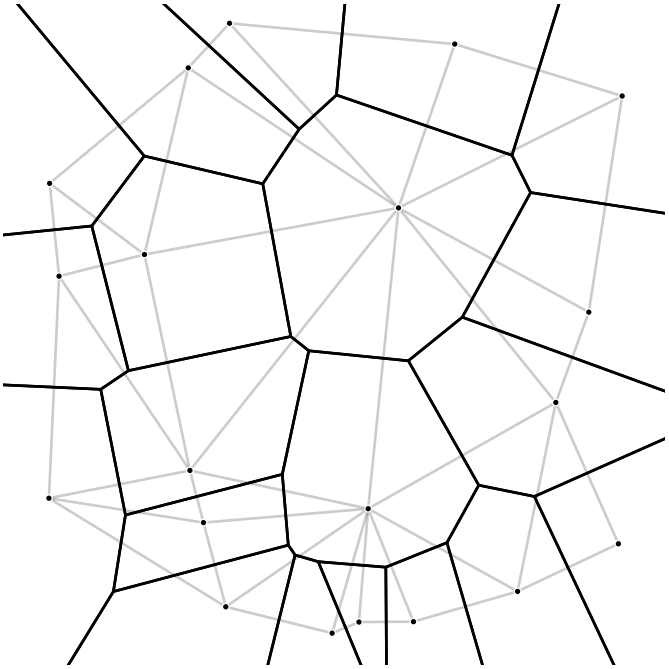


Obrázek 2.3: Ukázky kvadrantových stromů pro dvě různé funkce. Nahoře $f_1(x, y) = x^2+y^2$ a dole $f_2(x, y) = y^2-x^3-3x^2$. Implicitní křivka $f_1(x, y) = r^2$ definuje kružnici o poloměru r a $f_2(x, y) = b$ eliptickou křivku. Výsledné křivky jsou na obrázku 2.7.

Jednotlivé kroky algoritmu si postupně ukážeme na přehlednějších dvourozměrných příkladech. Použitý algoritmus je obdobný jako *Dual Marching Cubes*, pouze pracuje ve 2D a vykresluje tak implicitní křivky místo implicitních ploch. Na obrázku 2.3 jsou znázorněny kvadrantové stromy pro rovnice kružnice a eliptické křivky. Na obrázcích je vidět různě podrobné rozdělení roviny.

2.4 Duální mřížka

Síť, která vznikne propojením vrcholů sousedních buněk oktalového stromu, je topologicky duální k tomuto stromu. Dualita je symetrická vlastnost rovinných a prostorových struktur. Navzájem duální mohou být například planární grafy, teselace nebo tělesa. Teselace neboli mozaikování je rozdělení roviny na geometrické útvary, které se nepřekrývají a nejsou mezi nimi mezery. K dané teselaci vytvoříme duální tak, že do vnitřní části každého útvaru umístíme vrchol. Vrcholy uvnitř sousedících útvarů potom spojíme hranou. Zajímavým příkladem navzájem duálních teselací je Delaunayho triangulace a Voroného diagram na obrázku 2.4.

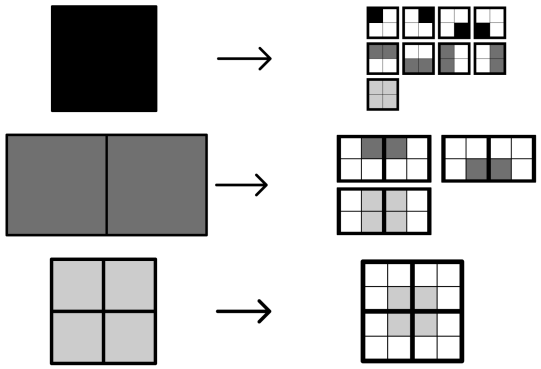


Obrázek 2.4: Příklad vzájemně duální teselace. Delaunayho triangulace je vyznačena šedivě a Voroného diagram černě.

Delaunayho triangulace množiny bodů P v rovině rozdělí body do trojúhelníků tak, aby žádný bod z P nebyl uvnitř opsané kružnice žádného z trojúhelníků. Tato podmínka maximalizuje nejmenší velikost vnitřního úhlu ze všech

trojúhelníku a předchází tak konstrukci velice tenkých trojúhelníků. Voroného diagram pro množinu bodů P rozdělí rovinu na množinu mnohoúhelníků R tak, že každý mnohoúhelník obsahuje právě jeden bod z P. Pro každý mnohoúhelník R_i s vnitřním bodem P_i platí, že každý bod uvnitř R_i je blíže k P_i než k jakémukoliv jinému bodu z P.

Duální mřížka, do které se budou generovat výsledné polygony, se vytvoří rekurzivním průchodem oktalového stromu. Pro jednoduchost si následující část algoritmu znovu popíšeme na dvourozměrných kvadrantových stromech. V daném stromu musíme nalézt všechny dvojice a čtveřice buněk, které mají společný vrchol. K tomu využijeme výhodné uspořádání buněk ve stromu. Průchod kvadrantovým stromem je uskutečněn pomocí tří rekurzivních funkcí. Použijeme pro ně stejné názvy jako v původním článku [12]: `faceProc`, `edgeProc` a `vertProc`. Funkce `faceProc` zpracovává jednu buňku stromu a volá sama sebe pro všechny čtyři její potomky. Obsahuje také čtyři volání `edgeProc` a jedno `vertProc`. Funkce `edgeProc` zpracovává dvojice buněk se společnou hranou. Dvakrát volá sama sebe a jednou funkci `vertProc`. Nakonec funkce `vertProc` bere jako parametr čtveřici buněk kolem jednoho vrcholu a volá sama sebe s potomky uzlů stromu, které dostala jako parametr. Volání těchto funkcí je znázorněno na obrázku 2.5



Obrázek 2.5: Znázornění volání rekurzivních funkcí pro konstrukci duální mřížky. Funkce `faceProc` je označena černou barvou, `edgeProc` tmavě šedivou a `vertProc` světle šedivou. Převzato z článku Scotta Schaefera a Joea Warrena [12].

Pokud jedním z parametrů těchto funkcí je list stromu, použije se pro další volání přímo tento list. Pokud jsou všechny parametry listy, žádná další funkce se už nevolá. Každé zavolání některé z těchto funkcí se tak dostává do hlubších pater kvadrantového stromu. Výsledné buňky duální mřížky se generují ve funkci `vertProc` a tvoří je čtveřice bodů, které odpovídají vypočítaným vrcholům uvnitř buněk kvadrantového stromu. Přestože jsou buňky duální mřížky tvořeny čtveřicí bodů, nemusí to být vždy čtyřúhelníky. Pokud se zavolá funkce `vertProc` pro listy v různé hloubce stromu, tak v některých

případech dostane funkce na vstupu dvakrát tu samou buňku. Vzniklá buňka potom degeneruje do tvaru trojúhelníku, protože dva ze čtyř bodů budou totožné.

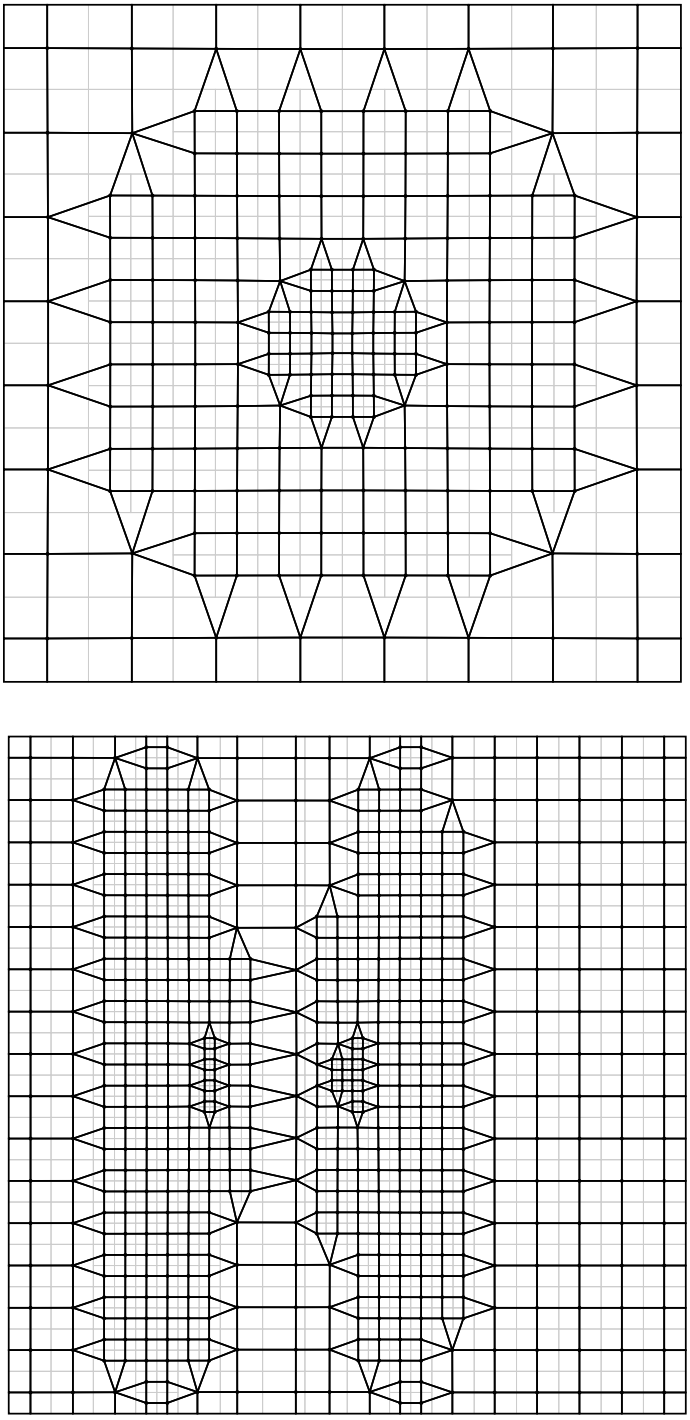
Tímto způsobem se vygeneruje duální mřížka pouze ve vnitřní části kořenové buňky kvadrantového stromu. Abychom pokryli celý vykreslovaný rozsah, musíme napojit mřížku na okraje základní buňky. Autoři algoritmu toto řeší dalším rekurzivním voláním funkcí `edgeProc` a `vertProc`. V této práci jsme však použili jiný přístup. Již při generování stromu si ukládáme informaci o tom, jestli se buňka nachází na okraji. Potom při rekurzivním průchodu pro každou takovou buňku vytvoříme další duální buňku spojující okraj. Na obrázku 2.6 jsou zobrazeny duální mřížky pro kvadrantové stromy z předchozí ukázky.

Při konstrukci trojrozměrné duální mřížky k oktalovým stromům je princip stejný. Je však potřeba přidat další funkci, která bude zpracovávat celou buňku. V naší implementaci jsme ji pojmenovali `cellProc`. Liší se také počet volání jednotlivých funkcí. Kolikrát se volají rekurzivní funkce znázorňuje následující tabulka.

Funkce	`cellProc`	`faceProc`	`edgeProc`	`vertProc`
`cellProc`	8×	12×	6×	1×
`faceProc`		4×	4×	1×
`edgeProc`			2×	1×
`vertProc`				1×

Tabulka 2.1: Tabulka počtu volání rekurzivních funkcí při konstrukci trojrozměrné duální mřížky. Počty udávají kolikrát se funkce uvedená v prvním řádku zavolá uvnitř funkce uvedené v prvním sloupci. Například v těle funkce `cellProc` se šestkrát zavolá funkce `edgeProc`.

V případě trojrozměrné duální mřížky jsou buňky tvořeny osmicí bodů. Ty mohou také degenerovat jako v dvourozměrném případě. Splynout v jeden mohou dva, nebo čtyři vrcholy duální buňky. Výsledné buňky duální mřížky tak mohou být pětistěny, šestistěny i jiné mnohostěny.

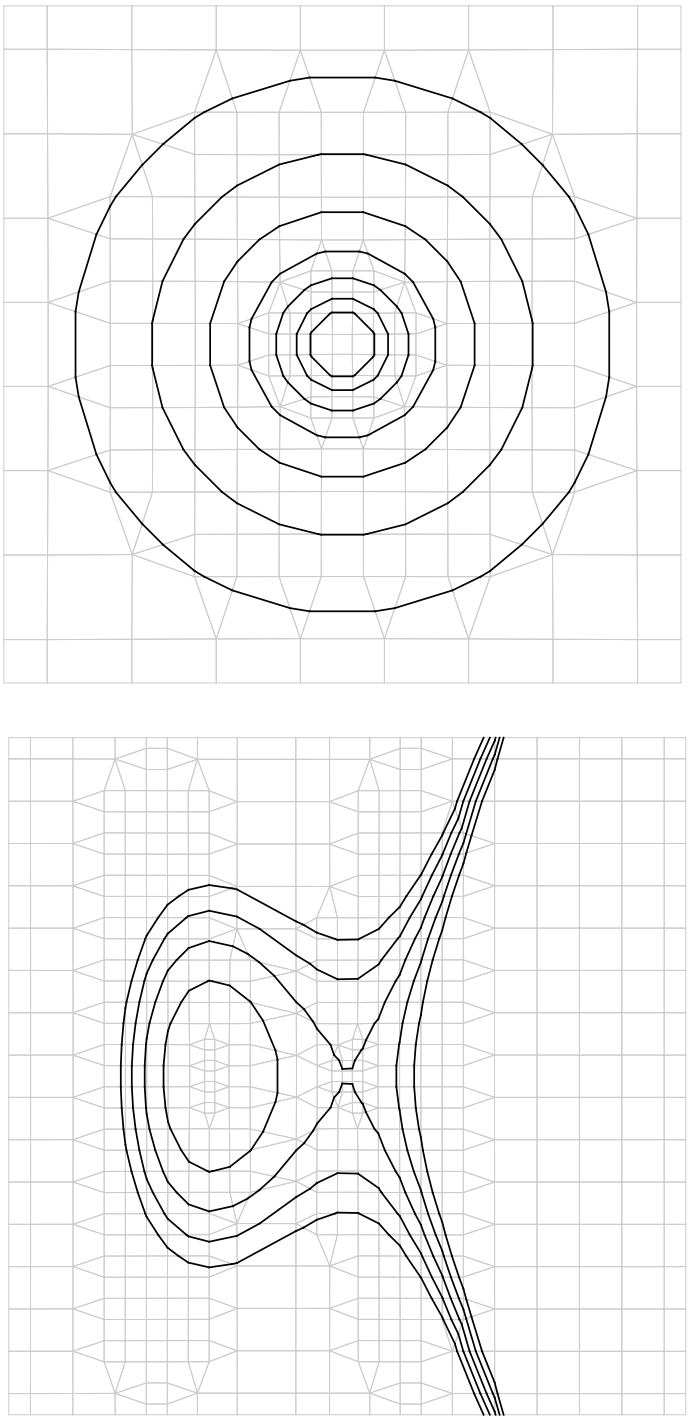


Obrázek 2.6: Ukázky mřížek, které jsou topologicky duální ke kvadrantovým stromům z předchozí ukázky 2.3.

2.5 Konturování duální mřížky

K získání výsledných polygonů implicitní plochy použijeme zobecněnou variantu algoritmu *Marching Cubes*. Ten pracuje s rovnoměrnou krychlovou mřížkou. *Dual Marching Cubes* využívá toho, že buňky duální mřížky jsou tvořeny osmi vrcholy stejně jako krychle a jsou také navzájem topologicky ekvivalentní. Zároveň mezi buňkami duální mrížky nejsou mezery a mřížka tak vyplnuje celý zkoumaný prostor. Tyto vlastnosti nám umožňují použít stejnou tabulku různých ohodnocení vrcholů jako používá *Marching Cubes*. Vrcholy buněk ohodnotíme 1 nebo 0 podle toho, jestli je hodnota funkce v tomto bodě větší nebo menší než zadaná konstanta, pro kterou konturu vykreslujeme. Algoritmus *Dual Marching Cubes* pro toto porovnání používá aproximovanou hodnotu w, kterou získá při konstrukci oktalového stromu. Tento přístup je obecnější a je možné ho použít i pro polygonizaci jiných objektů než implicitních ploch. Naše řešení se však zabývá pouze vykreslováním funkcí, pro které můžeme zjistit hodnotu v každém bodě definičního oboru funkce. Pro porovnání proto používáme přímo hodnotu funkce v tomto bodě. Dále postupujeme stejně jako algoritmus *Marching Cubes*. Vyhledávací tabulka nám určí, na kterých hranách duální mřížky budou ležet vrcholy trojúhelníku. Jejich konečnou polohu zjistíme lineární interpolací podle hodnot funkce v krajních bodech této hrany.

U implicitních křivek je tato část ještě jednodušší. Duální buňky jsou dány čtyřmi body, proto existuje pouze 16 různých ohodnocení a výsledná křivka je tvořena pouze úsečkami. Finální podoba příkladů implicitních křivek je znázorněna na obrázku 2.7.



Obrázek 2.7: Výsledné implicitní křivky. Nahoře jsou kružnice s rovnicí $x^2 + y^2 = a$, pro $a \in \{1, 2, 4, 8, 16, 32, 64\}$, dole potom eliptické křivky $y^2 - x^3 - 3x^2 = b$, kde $b \in \{-4, -2, 0, 2, 4\}$.

Kapitola 3

Implementace v SageMath

SageMath je open-source algebraický systém, který je volně šiřitelný pod licencí GPL. Skládá se z téměř stovky existujících balíčků jako například NumPy, SciPy, R nebo matplotlib. SageMath je kombinuje pomocí společného rozhraní, které je založeno na jazyku Python. Tento systém má široké využití, které zahrnuje základní algebru, matematickou analýzu, teorii čísel, kryptografii, numerické výpočty, komutativní algebru, teorii grup, kombinatoriku, teorie grafů, lineární algebru a další. Program je možné používat přes rozhraní webového prohlížeče nebo příkazovou řádku. Dostupná je také internetová varianta na adrese `cloud.sagemath.org`. Název SAGE byla původně zkratka anglického názvu „A Computer System for Algebra and Geometry Experimentation“.[1] Systém nabízí srovnatelné možnosti jako proprietární produkty Mathematica, Matlab nebo Maple, jeho výhodou je však právě dostupnost a otevřený zdrojový kód.

Pro zobrazení 3D grafiky používá SageMath program Jmol. Jmol je volně dostupný open-source software napsaný v Javě, který slouží k zobrazování molekul a dalších chemických struktur. SageMath tento program využívá k interaktivnímu zobrazení libovolného trojrozměrného objektu. Ve webovém prohlížeči používá SageMath JavaScriptovou verzi Jsmol. V projektu SageMathCloud vykresluje 3D grafiku JavaScriptová knihovna Three.js s podporou WebGL. Pro generování statických trojrozměrných scén v SageMath slouží knihovna Tachyon. Ta je napsaná v jazyce C a k renderování využívá techniku sledování paprsku (anglicky *ray tracing*). Obrázky implicitních ploch v této práci jsou generovány právě pomocí knihovny Tachyon.

Implementace algoritmu *Dual Marching Cubes* se skládá ze tříd: DualMarchingCubes, DualGrid, Octree a Cell. Třída DualMarchingCubes při inicializaci zpracuje parametry, případně je nastaví na výchozí hodnoty. Vstupní funkci `f`, pro kterou vykreslujeme implicitní plochu, chceme typicky zadat symbolicky. Náš program však vykresluje plochu numericky a počítá její hod-

[1] William Stein: SAGE: System for Algebra and Geometry Experimentation. Dostupné z: `http://wstein.org/sage.html`

notu mnohokrát v různých bodech. Použití symbolického vyhodnocení by tak bylo velice neefektivní. Proto funkci `f` transformujeme pomocí funkce `fast_callable`, která ji optimalizuje pro numerické vyhodnocení. Dále spočítáme parciální derivace podle každé ze tří proměnných a uložíme jejich optimalizovanou variantu. Poté vytvoříme instanci třídy Octree, která je kořenem oktalového stromu. Každý uzel stromu obsahuje seznam potomků, které jsou také typu Octree. Tímto způsobem je zajištěna hierarchická struktura. Každý uzel obsahuje instanci třídy Cell. Ta představuje buňku oktalového stromu tvaru krychle. Třída Cell také zajišťuje sestavení a minimalizaci funkce QEF a uložení vrcholu duální mřížky. Vypočítaný oktalový strom potom předáme třídě DualGrid, která slouží ke konstrukci duální mřížky. Výsledná duální mřížka je reprezentována jako seznam osmic bodů, které tvoří vrcholy duálních buněk. Nakonec duální mřížku použijeme v zobecněném algoritmu *Marching Cubes*. Při jeho implementaci jsme vycházeli z třídy ImplicitSurface, která tento algoritmus v SageMath realizuje. Z této třídy jsme přímo použili vyhledávací tabulku pro umístění polygonů. Naše implementace je uložena v souboru `dmc.pyx` a je možné ji načíst příkazem `load`, případně lze importovat funkce z přeložené knihovny `dmc.so`.

3.1 Funkce `contour_plot3d`

Program realizující algoritmus *Dual Marching Cubes* lze spustit zavoláním funkce `contour_plot3d`. Při návrhu rozhraní jsme se inspirovali již existujícími funkcemi `contour_plot` a `implicit_plot3d`. Funkce `contour_plot3d` je defnována následovně:

```
def contour_plot3d(f, contour, x_range, y_range, z_range,
↪  **kwds):
```

Funkce přijímá tyto povinné parametry:

- `f` – symbolický výraz definující implicitní plochu
- `contour` – hodnota, pro kterou chceme vykreslit konturu. Funkci lze předat i seznam hodnot a vykreslit tak více ploch najednou.
- `x_range` – trojice `(x, x_min, x_max)`, kde `x` je jedna z proměnných a hodnoty `x_min` a `x_max` určují interval na ose `x`.
- `y_range` – trojice `(y, y_min, y_max)`, kde `y` je jedna z proměnných a hodnoty `y_min` a `y_max` určují interval na ose `y`.
- `z_range` – trojice `(z, z_min, z_max)`, kde `z` je jedna z proměnných a hodnoty `z_min` a `z_max` určují interval na ose `z`.

Dále je možné funkci předat nepovinné parametry pomocí `**kwds`:

- `max_depth` – hodnota limitující maximální hloubku oktalového stromu.

- `qef_samples` – počet vzorkovacích bodů QEF v jednom směru. Pokud hodnotu nastavíme na n, potom celkový počet vzorků v každé buňce oktalového stromu bude n^3.
- `tree_algorithm` – parametr určující, který z algoritmů uvedených v sekci 2.3 se použije pro konstrukci stromu. Lze zadat buď `'epsilon'`, nebo `'queue'`.
- `epsilon` – hodnota, která určí podrobnost oktalového stromu. V případě použití algoritmu s prioritní frontou, tento parametr nemá vliv na výsledek.
- `max_leaves` – maximální počet listů oktalového stromu. Tento parametr určuje podrobnost stromu, pokud `tree_algorithm = 'queue'`
- `colors` – barva nebo seznam barev, které se použijí pro výsledné implicitní plochy. V případě, že tento parametr není zadán, se použije funkce `rainbow`, která plochám přiřadí co nejodlišnější odstíny.
- `verbose` – hodnota `True` nebo `False`, která zapíná nebo vypíná výpis informací o běhu algoritmu.

Použití funkce `contour_plot3d` si ukážeme na jednoduchém příkladu. Použijeme předpis $x^2+y^2+z^2 = r^2$, který definuje kulovou plochu o poloměru r. Vykreslíme tři různé kulové plochy pro poloměry 1, 2 a 3. Rozsah zobrazení ploch omezíme na krychli $\langle 0, 3\rangle^3$. Výsledek následujícího příkazu je znázorněn na obrázku 3.1.

```
contour_plot3d(x^2 + y^2 + z^2, (1, 4, 9), (x, 0, 3), (y, 0, 3),
↪  (z, 0, 3))
```

Obrázek 3.1: Ukázka použití funkce `contour_plot3d`. Tento příkaz vykreslí tři části soustředných kulových ploch s poloměry 1, 2 a 3.

3.2 *Quadratic Error Function*

V minulé kapitole jsme uvedli, že QEF vyjádříme ve tvaru kvadriky. Ten z předpisu funkce sestavíme následovně:

Začneme s čitatelem jednoho ze sčítancu pro vzrokovací bod (x_i, y_i, z_i).

$$(w - T_i(x, y, z))^2 =$$
$$= (w - (\nabla f(x_i, y_i, z_i) \cdot ((x, y, z) - (x_i, y_i, z_i)) + f(x_i, y_i, z_i)))^2.$$

Výsledek gradientu $\nabla f(x_i, y_i, z_i)$ označíme jako vektor (a, b, c), kde

$$a = \frac{\partial f}{\partial x}(x_i, y_i, z_i), \qquad b = \frac{\partial f}{\partial y}(x_i, y_i, z_i), \qquad c = \frac{\partial f}{\partial z}(x_i, y_i, z_i).$$
$$(w - (a, b, c) \cdot ((x, y, z) - (x_i, y_i, z_i)) - f(x_i, y_i, z_i))^2 =$$
$$= (w - (a, b, c) \cdot (x - x_i, y - y_i, z - z_i) - f(x_i, y_i, z_i))^2 =$$
$$= (w - (a(x - x_i) + b(y - y_i) + c(z - z_i)) - f(x_i, y_i, z_i))^2 =$$
$$= (w - ax - by - cz - (f(x_i, y_i, z_i) - ax_i - by_i - cz_i))^2$$

Absolutní člen $f(x_i, y_i, z_i) - ax_i - by_i - cz_i$ označíme d. Výraz má nyní tvar

$$(w - ax - by - cz - d)^2.$$

Po umocnění dostaneme výraz

$$\begin{aligned}
&w^2 - awx - bwy - cwz - dw + \\
-\ &awx + a^2x^2 + abxy + acxz + adx + \\
-\ &bwy + abxy + b^2y^2 + bcyz + bdy + \\
-\ &cwz + acxz + bcyz + c^2z^2 + cdz + \\
-\ &dw + adx + bdy + cdz + d^2.
\end{aligned}$$

Z něj už lze snadno vyčíst maticový zápis kvadriky $xQ_ix^{\mathrm{T}} + P_ix^{\mathrm{T}} + R_i$

$$Q_i = \begin{pmatrix} 1 & -a & -b & -c \\ -a & a^2 & ab & ac \\ -b & ab & b^2 & bc \\ -c & ac & bc & c^2 \end{pmatrix}, \qquad P_i = \begin{pmatrix} -2d & 2ad & 2bd & 2cd \end{pmatrix}, \qquad R_i = d^2.$$

Jmenovatel výrazu $1 + |\nabla f(x_i, y_i, z_i)|^2$ neobsahuje žádnou proměnnou, proto můžeme Q_i, P_i a R_i vydělit touto hodnotou. Abychom získali výsledný předpis QEF stačí už jen sečíst příslušné Q_i, P_i a R_i pro všechny vzorkovací body. Tímto způsobem sestavíme QEF v každé buňce oktalového stromu.

Vektor x_m, ve kterém kvadratická chybová funkce nabývá minima, získáme vyřešením maticové rovnice

$$xQ = -\frac{1}{2}P,$$

případně vzorcem

$$x_m = x_c + Q^+(P - x_cQ),$$

pokud matice Q je singulární. Realizovat tuto část v SageMath je jednoduché. Využijeme k tomu existující funkce `solve_left` a `pseudoinverse`.

```
try:
    X_m = - Q.solve_left(P / 2)
except:
    X_m = X_c + Q.pseudoinverse() * (P - X_c * Q)
```

V případě, že matice Q je singulární, vyvolá funkce `solve_left` výjimku. Tu zachytíme a použijeme alternativní nalezení minima. Hodnotu minima, která představuje výslednou velikost odchylky v konkrétní buňce, získáme dosazením x_m do QEF.

```
error = X_m * Q * X_m + P * X_m + R
```

Vytvoření kvadratické chybové funkce je kritická část implementace z hlediska rychlosti běhu programu. Kvadriku je potřeba sestavit několikrát v každém uzlu oktalového stromu podle hodnoty parametru `qef_samples`. Experimentovali jsme s různým množstvím vzorkovacích bodů. Hodnota 3 byla pro některé plochy příliš nízká a oktalový strom špatně vystihoval tvar plochy. Hodnota 4 už byla pro většinu vstupů dostatečná. Toto nastavení znamená, že se kvadrika vytvoří pro 64 bodů uvnitř buňky. Počet uzlů oktalového stromu závisí na nastavení vstupních parametrů. Zhruba tisíc uzlů stačí k přibližnému vyobrazení méně členitých ploch. Pro detailní vykreslení se muže vygenerovat i několik tisíc uzlů. V jednom běhu programu se tak kvadrika spočítá řádově stotisíckrát.

Proto jsme se snažili výpočet QEF co nejlépe optimalizovat. K tomu jsme využili jazyk Cython. Cython je programovací jazyk podobný Pythonu a je v něm napsaná většina zdrojového kódu SageMath. Cython je, až na drobné výjimky, nadmnožina jazyka Python. Na rozdíl od Pythonu je to jazyk kompilovaný, což se projevuje na jeho výkonosti, která se blíží jazyku C. Program napsaný v Cythonu se přeloží přímo do jazyka C. Umožňuje tak relativně snadné vytvoření modulu v jazyce C a následné použití v Pythonu. Proto jsme funkci sestavující QEF přepsali pomocí Cythonu a dosáhli tak výrazného zrychlení programu. Zrychlení programu jsme otestovali podrobnou polygonizací kulové plochy $x^2+y^2+z^2 = 1$. Rozdíl v naměřených časech s použitím Cythonu a bez něho je zachycen v tabulkách 3.1 a 3.2. V obou případech jsme pro uložení kvadrik použili efektivně implementované matice z knihovny NumPy.

Tabulka 3.1: Tabulka časů běhu jednotlivých funkcích před použitím Cythonu

Počet volání	čas [s]	čas %	Název funkce
1	29,768	100,00	`implicit_plot3d`
1	28,810	96,78	`compute_ot`
577	28,810	96,78	`examine_eps`
4616	27,404	92,06	`minimize_qef`
4616	**23,869**	**80,18**	**`qef`**
4616	0,868	2,92	`move_to_border`
577	0,636	2,14	`split`
1	0,484	1,63	`compute_dg`
4617	0,484	1,63	`cell_proc`
1	0,469	1,58	`triangulate`
4857	0,461	1,55	`marching_dualcells`
12924	0,343	1,15	`face_proc`
7476	0,207	0,70	`point_on_dualcell_edge`
7476	0,197	0,66	`interpolate_middle`
12042	0,194	0,65	`edge_proc`
3367	0,087	0,29	`vertex_proc`

Tabulka 3.2: Tabulka časů běhu jednotlivých funkcích s použitím Cythonu

Počet volání	čas [s]	čas %	Název funkce
1	8,546	100,00	`implicit_plot3d`
1	7,677	89,83	`compute_ot`
577	7,677	89,83	`examine_eps`
4616	6,577	76,96	`minimize_qef`
4616	**3,154**	**36,91**	**`qef_cython`**
4616	0,819	9,58	`move_to_border`
1	0,448	5,24	`triangulate`
4857	0,440	5,15	`marching_dualcells`
1	0,417	4,88	`compute_dg`
4617	0,417	4,88	`cell_proc`
577	0,336	3,93	`split`
12924	0,291	3,41	`face_proc`
7476	0,204	2,39	`point_on_dualcell_edge`
7476	0,194	2,27	`interpolate_middle`
12042	0,169	1,98	`edge_proc`
3367	0,073	0,85	`vertex_proc`

Tabulky udávají celkový kumulativní čas běhu jednotlivých funkcí. To zahrnuje čas strávený přímo v těle funkce i uvnitř všech zavolaných funkcích. Všechny časy v této práci jsou naměřeny na počítači Dell Latitude e6220 s procesorem Intel i5-2520M s maximální frekvencí 3,2 GHz.

K profilování našeho programu jsme použili modul `prun`, který je obsažen v interaktivním shellu IPython. Pro obě měření jsme použili kód, který je uvedený níže. Příklady se liší pouze načtením různých implementací třídy DualMarchingCubes.

```
settings = {
    'x_range': (-1, 1),
    'y_range': (-1, 1),
    'z_range': (-1, 1),
    'max_depth': 5,
    'qef_samples': 4,
    'epsilon': 0.001,
    'tree_algorithm': 'epsilon'
}
f = x^2 + y^2 + z^2
dmc = DualMarchingCubes(f, (x,y,z), **settings)

%prun -s cumtime dmc.implicit_plot3d(1)
```

Nakonec jsme pomocí Cythonu přeložili celý zdrojový kód a program tak ještě mírně zrychlili. Pro měření optimalizované implementace jsme použili funkci `timeit` a stejné nastavení jako při minulém měření:

```
timeit('dmc_cython.implicit_plot3d(1)', number=3, repeat=3)
3 loops, best of 3: 6.72 s~per loop
```

Tato funkce opakuje příkaz 3 krát a spočítá průměrnou dobu běhu. Celé měření se spustí také 3 krát (dle parametru `number`) a vypíše se nejkratší naměřená doba. Stejným způsobem jsme změřili variantu, kde byla Cythonem přeložena pouze funkce `qef`, a získali čas 8,42 s. Pro neoptimalizovanou implementaci jsem naměřili čas 28,32 s.

3.3 *Doctesty*

SageMath využívá k testování jednotlivých funkcí *doctesty*, které jsou součástí standardní knihovny jazyka Python. Doctest je jednotkový test, který je napsaný v komentáři zdrojového kódu pod definicí funkce nebo třídy programu. Tyto komentáře se nazývají *docstringy* a na rozdíl od běžných komentářů zdrojového kódu se neodstraňují při parsování. Z *Docstringů* se automaticky generuje oficiální dokumentace SageMath pomocí nástroje Sphinx. *Docstringy* v SageMath mají přesně stanovená pravidla formátování [15]. Každý *docstring* by měl začínat jednou větou, která popisuje funkci. Následuje pak několik povinných nebo nepovinných bloků, které dále specifikují chování funkce.

K testování slouží bloky `EXAMPLES` a `TESTS`. Blok `EXAMPLES` ukazuje použití funkce a zároveň kontroluje, jestli funkce vrací očekávaný výstup. Blok `TESTS`

pouze testuje správné chování funkce. Zápis *docstringu* si ukážeme na funkci `ContourPlot3D`:

```
def contour_plot3d(f, contour, x_range, y_range, z_range,
↪  **kwds):
    """
    Creates a~3D plot of an isosurface of given function.

    INPUT:
    -  ``f`` - function
    -  ``contour`` - isosurface f(x,y,z)==contour.
           Can be a~list, in which case
           multiple contours are plotted.
    -  ``x_range`` - a~3-tuple (x, x_min, x_max)
    -  ``y_range`` - a~3-tuple (y, y_min, y_may)
    -  ``z_range`` - a~3-tuple (z, z_min, z_maz)

    OUTPUT:
    A~Graphics3d Object

    EXAMPLES:
        sage: var('x,y,z')
        (x, y, z)

    A~sphere with a~radius 2::
        sage: contour_plot3d(x^2+y^2+z^2, 4,
            (x, -3, 3), (y, -3, 3), (z, -3, 3))
        Graphics3d Object

    A~torus::
        sage: contour_plot3d((sqrt(x^2+y^2)-3)^2 + z^2, 1,
            (x, -4, 4), (y, -4, 4), (z, -1, 1))
        Graphics3d Object

    TESTS:
        sage: nothing = contour_plot3d(x^2+y^2+z^2, 0,
            (x, -1, 1), (y, -1, 1), (z, -1, 1))
        sage: len(nothing.flatten().face_list())
        0
    """
```

Spuštění a výstup testů vypadá následovně:

```
$ sage -t dmc.pyx

Doctesting 1 file.
Compiling ./dmc.pyx...
sage -t dmc.pyx
    [51 tests, 6.96 s]
----------------------------------------------
All tests passed!
----------------------------------------------
Total time for all tests: 21.1 seconds
    cpu time: 2.6 seconds
```

Kapitola 4

Ukázky použití

Naši implementaci algoritmu *Dual Marching Cubes* porovnáme s algoritmem *Marching Cubes*, který je v SageMath použit ve funkci `implicit_plot3d`. Implementace porovnáme jak z hlediska časové náročnosti, tak z hlediska kvality výsledné polygonizace. Porovnání těchto dvou algoritmů není triviální, protože každý z nich používá jiné vstupní parametry pro volbu přesnosti polygonizace. Pokusíme se proto najít takové nastavení parametrů, které bude generovat plochy složené z podobného počtu trojúhelníků.

4.1 Porovnání s *Marching Cubes*

Nejprve obě implementace porovnáme z hlediska doby výpočtu. Z analýzy algoritmu lze předpokládat, že *Dual Marching Cubes* bude pomalejší než *Marching Cubes*. Celý zobecněný algoritmus *Marching Cubes* je totiž jeho součástí. Také autoři algoritmu [12] uvádějí, že konturování duální mřížky není pomalejší než konturování té rovnoměrné. Pomalá je však její konstrukce. Z našeho měření v předchozí kapitole dokonce vyplývá, že výpočet oktalového stromu a následné sestavení duální mřížky zabírá téměř 95 % času. Porovnání doby běhu algoritmů pro různé plochy je uvedeno v tabulce 4.1. Z tabulky je jasné, že algoritmus *Dual Marching Cubes* je výrazně pomalejší. Zatímco algoritmus *Marching Cubes* počítá polygonizaci několik milisekund, běh *Dual Marching Cubes* trvá několik sekund. Lze tedy říct, že naše implementace je řádově stokrát až tisíckrát pomalejší. Kód algoritmu *Marching Cubes* v SageMath je důkladně optimalizovaný. V naší implementaci je prostor pro další optimalizace. Šlo by například využít jazyka Cython i v dalších funkcích. Ani to by však nejspíše nepomohlo k tomu, aby se rychlost *Dual Marching Cubes* přiblížila algoritmu *Marching Cubes*.

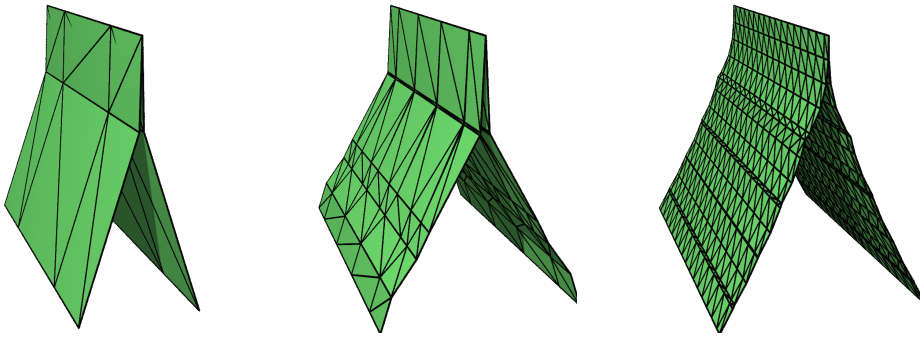
Tabulka 4.1: Porovnání času výpočtu *Marching Cubes* a *Dual Marching Cubes* pro různé implicitní plochy. Sloupce $|P|$ obsahují počet polygonů vykreslených ploch. Poslední sloupec udává, kolikrát je algoritmus *Marching Cubes* rychlejší než *Dual Marching Cubes*. Měření bylo provedeno na stejném počítači, který je popsaný v předchozí kapitole, pomocí funkce `timeit`.

implicitní plocha	MC		DMC		$\frac{t_{DMC}}{t_{MC}}$
	$\|P\|$	čas [ms]	$\|P\|$	čas [s]	
$x^2+y^2+z^2=1$	824	4,99	812	1,91	383
$x^2+y^2+z^2=1$	2588	11,65	2492	6,53	561
$x^4+y^4+z^4-x^2-y^2-z^2=0$	1340	9,56	1340	5,83	610
$x^2+y^2-\ln^2(z+3{,}2)=0{,}02$	1952	10,93	2000	14,05	1285

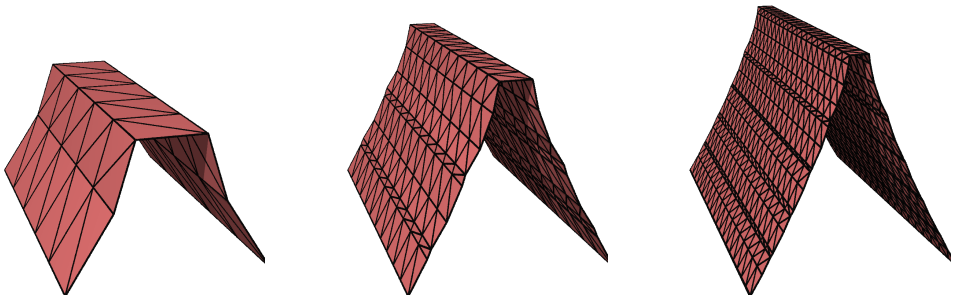
Hlavní motivací autorů *Dual Marching Cubes* bylo vytvořit algoritmus, který dobře reprodukuje ostré hrany a zároveň generuje výrazně méně polygonů než ostatní algoritmy. Článek [12] uvádí, že algoritmus lépe vystihuje vlastnosti plochy, a proto lze ke konturování použít méně podrobné mřížky. Porovnáme proto, jak přesně vystihují oba algoritmy různé implicitní plochy.

Na obrázku 4.1 je příklad implicitní plochy $\frac{1}{10}x^3 - y^2 = 0$, která obsahuje ostrou hranu v bodech, kde $x = 0$ a $y = 0$. Plocha je zobrazena v intervalech $x \in \langle -\frac{1}{2}, \frac{3}{2}\rangle$, $y \in \langle -1, 1\rangle$ a $z \in \langle -1, 1\rangle$. Nahoře jsou zelenou barvou vykresleny plochy algoritmem *Dual Marching Cubes*, dole potom červeně s použitím algoritmu *Marching Cubes*. Každým algoritmem je plocha zobrazena třikrát s různým nastavením podrobnosti. Pro algoritmus *Dual Marching Cubes* jsou hodnoty ϵ popořadě 9, 7 a 0. Nulová hodnota ϵ v posledním případě způsobí, že se oktalový strom vygeneruje rovnoměrně až do maximální hloubky (zde 4). Toto nastavení jsme zvolili proto, že se algoritmus zaměřil pouze na jednu část plochy, kde generoval mnoho malých trojúhelníků. Výsledek potom vypadal téměř stejně, jako příklad 4.1b. Pro *Marching Cubes* je nastaven počet krychlí na hodnoty 5^3, 11^3 a 19^3. Tento příklad potvrdil, že algoritmus *Dual Marching Cubes* dokáže správně vystihnou ostré hrany i při malém počtu výsledných trojúhelníků. Na druhou stranu o něco hůře zachycuje celkový tvar plochy. To je možné vidět na zalomení horní části plochy nebo na obrázku 4.1a, kde spodní část je příliš úzká. Ukázalo se také, že algoritmus *Marching Cubes* skutečně není při nízkém rozlišení schopný odhalit ostré čísti plochy.

Obrázek 4.2 ukazuje polygonizaci plochy $x^4 + y^4 + z^4 - (x^2 + y^2 + z^2) = 0$ uvnitř krychle $\langle -2, 2\rangle^3$. Vlevo pomocí algoritmu *Dual Marching Cubes*, vpravo pomocí *Marching Cubes*. Výsledné plochy jsou zobrazeny velice podobně. Je však možné si všimnout, že algoritmus *Dual Marching Cubes* produkuje různě velké trojúhelníky. Konkrétně v prohlubních ve prostředku objektu jsou trojúhelníky zhruba poloviční než v rozích. Trojúhelníky generované *Marching Cubes* jsou všechny přibližně stejně velké.

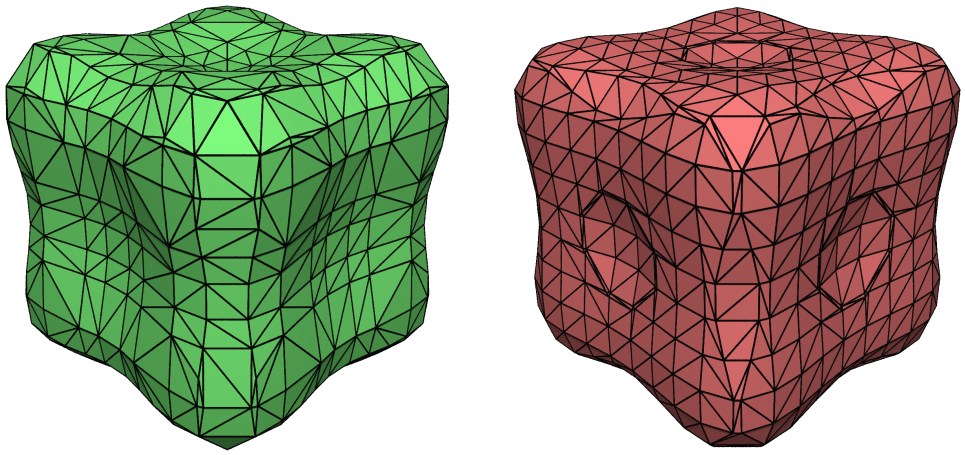


(a) 30 polygonů (b) 354 polygonů (c) 1258 polygonů



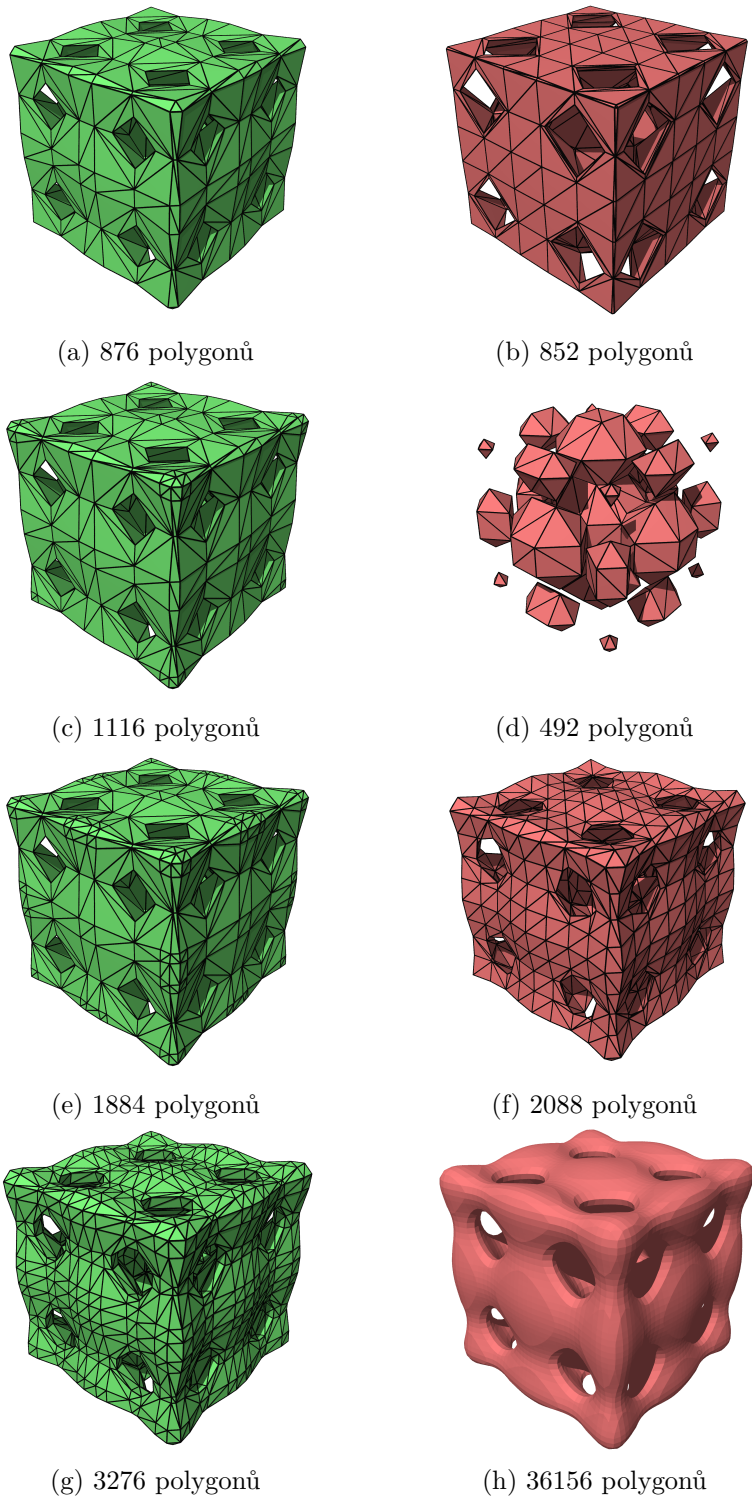
(d) 50 polygonů (e) 374 polygonů (f) 1330 polygonů

Obrázek 4.1: Porovnání vykreslení implicitní plochy $\frac{1}{10}x^3 - y^2 = 0$ s ostrou hranou. Nahoře jsou plochy polygonizované algoritmem *Dual Marching Cubes*, dole *Marching Cubes*.



(a) 1340 polygonů (b) 1340 polygonů

Obrázek 4.2: Porovnání *Dual Marching Cubes* (vlevo) a *Marching Cubes* (vpravo). Implicitní plocha je dána předpisem $x^4 + y^4 + z^4 - (x^2 + y^2 + z^2) = 0$



(a) 876 polygonů (b) 852 polygonů

(c) 1116 polygonů (d) 492 polygonů

(e) 1884 polygonů (f) 2088 polygonů

(g) 3276 polygonů (h) 36156 polygonů

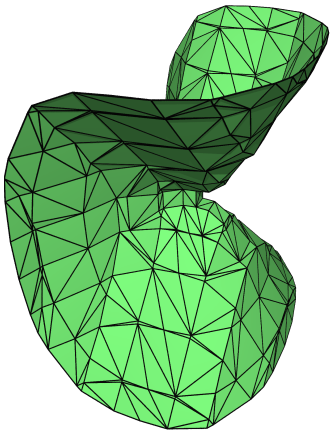
Obrázek 4.3: Porovnání *Dual Marching Cubes (vlevo)* a *Marching Cubes* (vpravo).

Na obrázku 4.3 je zobrazena složitá implicitní plocha daná rovnicí

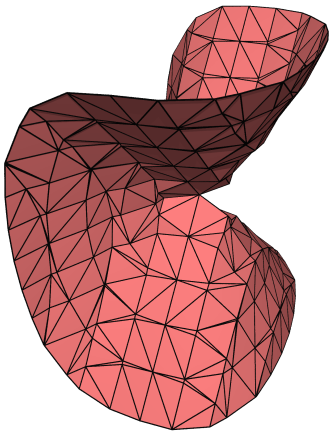
$$2\Big(x^2(3-4x^2)^2 + y^2(3-4y^2)^2 + z^2(3-4z^2)^2\Big) = 3,$$

kde x, y a z jsou z intervalu $\langle -1{,}3, 1{,}3\rangle$. Vlevo jsou plochy vykresleny algoritmem *Dual Marching Cubes*, vpravo algoritmem *Marching Cubes*. Parametr ϵ je pro algoritmus *Dual Marching Cubes* nastaven popořadě na hodnoty 0,8, 0,2, 0,1 a 0,01. Počty krychlí, které určují přesnost algoritmu *Marching Cubes*, jsou odshora dolů 7^3, 10^3, 12^3 a 49^3. Na obrázcích produkovaných algoritmem *Dual Marching Cubes* je vidět, že dobře vystihuje tvary plochy už pří nízkém počtu polygonů. Se snižující se hodnotou ϵ zůstává tvar plochy přibližně stejný, pouze se zpřesňuje v zakřivených místech. To je způsobeno tím, že se duální mřížka přizpůsobí vstupní funkci. Naopak u algoritmu *Marching Cubes* významně záleží na volbě počtu krychlí. Na obrázku 4.3b algoritmus správně určil základní tvar plochy. Naopak na obrázku 4.3d, který je vykreslený s větší přesností, algoritmus selhal a vyprodukoval naprosto odlišný tvar. Na obrázku 4.3h je použita velice detailní polygonizace, která odpovídá skutečnému tvaru implicitní plochy.

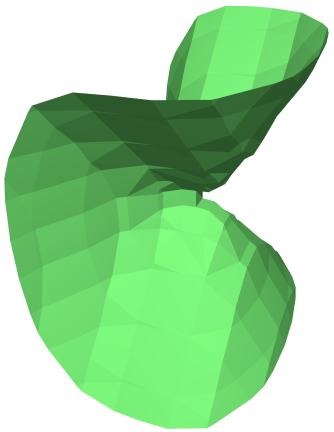
Na obrázku 4.4 je vykreslena implicitní plocha $x^3 + y^2 - z^2 = 0$ uvnitř krychle $\langle -2, 2\rangle^3$. Na levé straně pomocí *Dual Marching Cubes*, na pravo pomocí *Marching Cubes*. První dva obrázky od každého algoritmu jsou vykresleny se shodným nastavením. Liší se pouze v tom, že horní obrázek obsahuje zvýrazněné trojúhelníky. Méně detailní varianta byla polygonizována s hodnotou $\epsilon = 0{,}08$ pro algoritmus *Dual Marching Cubes* a pomocí 10^3 pochodujících krychlí pro algoritmus *Marching Cubes*. U podrobné varianty má ϵ hodnotu 0,0008 a počet krychlí je 33^3. I z této ukázky je patrné, že oba algoritmy vykreslují plochy velmi podobně, drobné rozdíly mezi nimi však jsou. Na obrázku 4.4a je vidět, že uprostřed, kde je plocha nejvíce zakřivená, jsou generovány menší polygony. Naopak na obrázku 4.4b jsou všechny trojúhelníky prakticky stejně velké. Podrobně vykreslené obrázky 4.4e a 4.4f jsou téměř totožné a oba velice dobře vystihují tvar plochy. Ale i na tomto příkladu je vidět, že *Dual Marching Cubes* o trochu přesněji vykresluje střed plochy.



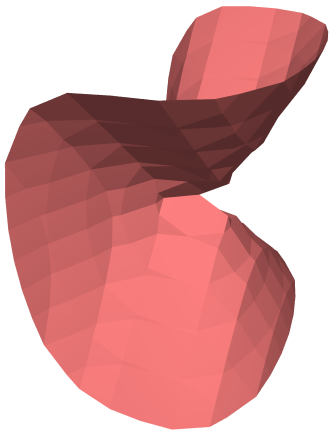
(a) 562 polygonů



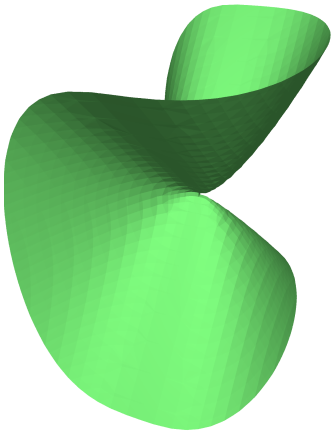
(b) 576 polygonů



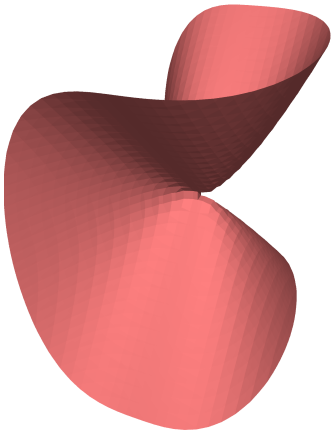
(c) 562 polygonů



(d) 576 polygonů



(e) 6362 polygonů



(f) 6458 polygonů

Obrázek 4.4: Implicitní plocha $x^3 + y^2 - z^2 = 0$, zobrazená pomocí *Dual Marching Cubes* (vlevo) a *Marching Cubes* (vpravo).

4.2 Zajímavé implicitní plochy

Použití algoritmu *Dual Marching Cubes* demonstrujeme na několika příkladech vizuálně zajímavých ploch. Ukázka 4.5 obsahuje čtyři detailně vykreslené implicitní plochy.

Na prvním obrázku 4.5a je zobrazena středově souměrná plocha, která je určena rovnicí

$$\cos(x) + \cos(y) + \cos(z) = 0.$$

Rozsah vykreslení je omezen krychlí $\langle -4, 4\rangle^3$.

Na obrázku 4.5b je příklad rotační implicitní plochy s předpisem

$$x^2 + y^2 - \ln^2(z + 3.2) = 0.02,$$

která je zobrazena uvnitř krychle $\langle -3, 3\rangle^3$.

Příklad 4.5c zobrazuje tzv. Clebschovu plochu uvnitř krychle $\langle -1, 1\rangle^3$. Vlastnostmi této plochy se zabývali němečtí matematici Alfred Clebsch a Felix Klein na konci 19. století. Tuto plochu je možné vyjádřit implicitně pomocí rovnice

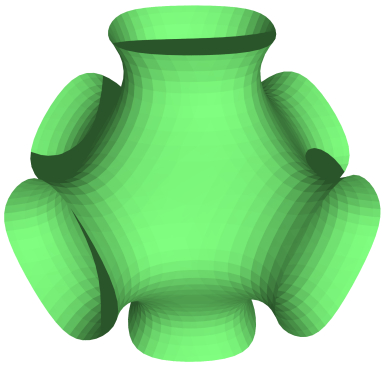
$$\begin{aligned} 81(x^3 + y^3 + z^3) \\ -189(x^2y + x^2z + y^2x + y^2z + z^2x + z^2y) \\ +54xyz + 126(xy + xz + yz) \\ -9(x^2 + y^2 + z^2) - 9(x + y + z) = -1. \end{aligned}$$

Na posledním obrázku 4.5d je zobrazen příklad plochy ze skupiny Banchoff-Chmutovových[2] ploch. Zde je zobrazena plocha konstantní hodnoty funkce

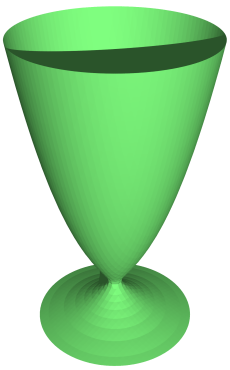
$$\begin{aligned} f(x, y, z) = &\Big((x^2 + y^2 - \alpha^2)^2 + (z^2 - 1)^2\Big) \\ &\Big((y^2 + z^2 - \alpha^2)^2 + (x^2 - 1)^2\Big) \\ &\Big((z^2 + x^2 - \alpha^2)^2 + (y^2 - 1)^2\Big). \end{aligned}$$

Parametr $\alpha = 0,75$ a plocha je vykreslena pro hodnotu $f(x, y, z) = 0,06$. Rozsah je omezen krychlí $\langle -\frac{6}{5}, \frac{6}{5}\rangle^3$.

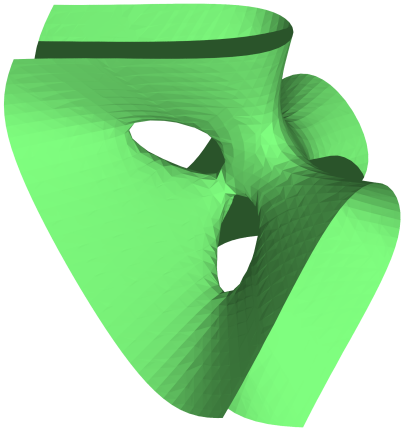
[2]Do skupiny Banchoff-Chmutovových ploch patří i plocha na obrázku 4.3.



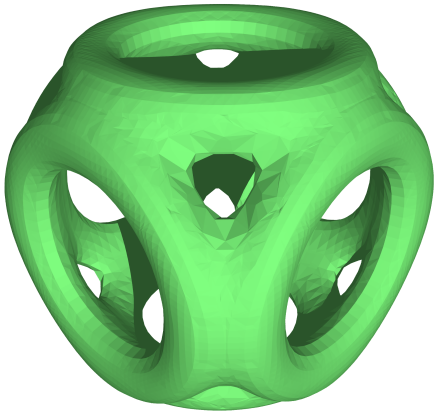
(a) $\cos(x) + \cos(y) + \cos(z) = 0$



(b) rotační plocha



(c) Clebschova plocha



(d) Banchoff-Chmutovova plocha

Obrázek 4.5: Ukázky zajímavých implicitních ploch, které jsou vykresleny pomocí algoritmu *Dual Marching Cubes*.

4.3 Vizualizace elektronových orbitalů v atomu vodíku

Na závěr této práce využijeme implementaci algoritmu *Dual Marching Cubes* k vytvoření vizualizací elektronových orbitalů v atomu vodíku. Atomový orbital je geometrický útvar, který vymezuje prostor v elektronovém obalu s určitou pravděpodobností výskytu elektronu. Elektron má duální charakter stejně jako každá elementární částice. To znamená, že vykazuje vlastnosti, které odpovídají jak částicím, tak vlnění. Pro částice je charakteristická definovaná hmotnost, poloha a hybnost. V důsledku vlnového charakteru elektronu však podle Heisenbergova principu neurčitosti nelze stanovit jeho přesnou polohu v atomu v daném časovém okamžiku. Tu můžeme vyjádřit pouze jako vlnovou funkci, která je řešením Schrödingerovy rovnice. Tato funkce, která je amplitudou pravděpodobnosti, že se v daném místě elektron nachází, popisuje mimo jiné i velikost a tvar orbitalu. Druhá mocnina absolutní hodnoty vlnové funkce vyjadřuje hustotu pravděpodobnosti výskytu částice v daném bodě.

Stav elektronu v atomu je specifikován kvantovými čísly. Ty popisují energii elektronu, typ a velikost orbitalů. Hlavní kvantové číslo n určuje energii elektronu a velikost orbitalu, zatímco typ a tvar orbitalu je určen vedlejším kvantovým číslem ℓ. Typ orbitalu se pro $\ell = 0$ značí **s**, pro $\ell = 1$ **p**, pro $\ell = 2$ **d** a pro $\ell = 3$ **f**. Magnetické kvantové číslo m popisuje prostorovou orientaci orbitalu. Posledním kvantovým číslem je číslo spinové s popisující magnetický moment elektronu (tzv. spin). Kvantová čísla mohou nabývat následujících hodnot:

$$\begin{aligned} n &= 1, 2, \ldots, \\ \ell &= 0, 1, \ldots, n-1, \\ m &= -\ell, -\ell+1, \ldots, \ell-1, \ell. \end{aligned}$$

Vlnová funkce ψ pro elektron v atomu vodíku je pro trojici sférických souřadnic r, ϑ a φ a kvantová čísla n, ℓ a m definována předpisem

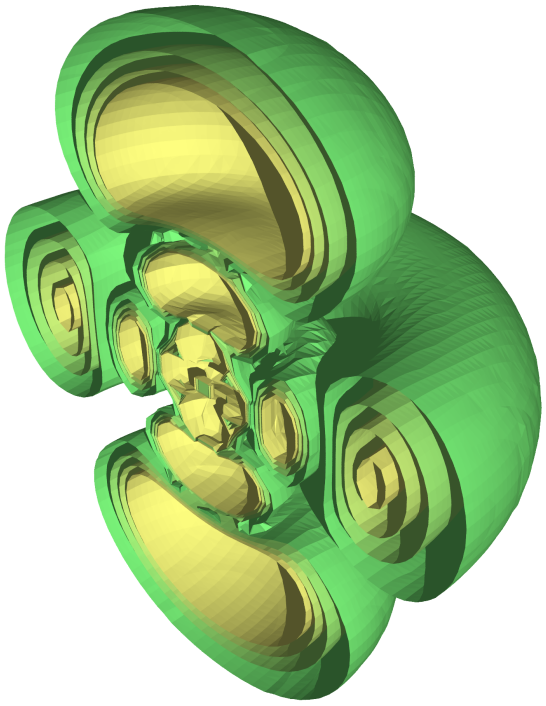
$$\psi_{n\ell m}(r, \vartheta, \varphi) = \sqrt{\left(\frac{2}{na_0}\right)^3 \frac{(n-\ell-1)!}{2n\Big((n+\ell)!\Big)}} e^{-\rho/2} \rho^\ell L_{n-\ell-1}^{2\ell+1}(\rho) Y_\ell^m(\vartheta, \varphi),$$

kde $\rho = \frac{2r}{na_0}$, konstanta a_0 je Bohrův poloměr, $L_{n-\ell-1}^{2\ell+1}(\rho)$ je zobecněný Laguerrův polynom stupně $n-\ell-1$ a $Y_\ell^m(\vartheta, \varphi)$ je sférická harmonická funkce stupně ℓ a řádu m.

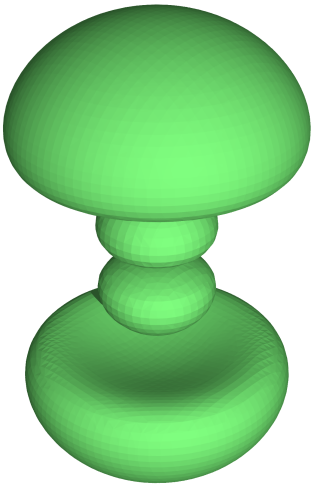
Abychom mohli pro vykreslení tvaru orbitalu použít naši implementaci, musíme sférické souřadnice převést na kartézské. K tomu použijeme následující vzorce:

$$r = \sqrt{x^2 + y^2 + z^2},$$
$$\vartheta = \arccos \frac{z}{r},$$
$$\varphi = \operatorname{atan2} \frac{y}{x}.$$

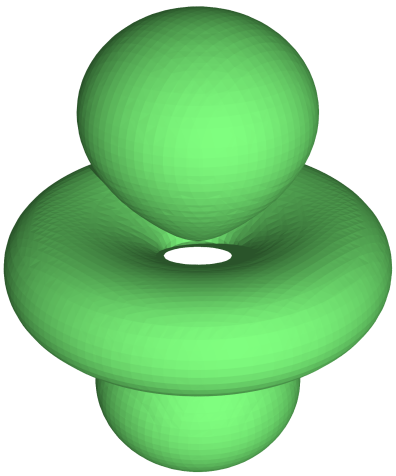
Vizualizace elektronových orbitalů v atomu vodíku vytvoříme vykreslením plochy konstantní hodnoty funkce $|\psi|^2$ – hustoty pravděpodobnosti výskytu elektronu. Na obrázku 4.6 je zobrazen řez orbitalem **d** s kvantovými čísly $n = 5, \ell = 2, m = 0$. Obrázek se skládá ze čtyř ploch pro různé hodnoty hustoty pravděpodobnosti. Na obrázku 4.7 jsou vykresleny čtyři různé tvary orbitalů. Typy orbitalů jsou popořadě **p**, **d**, **f** a **f**.



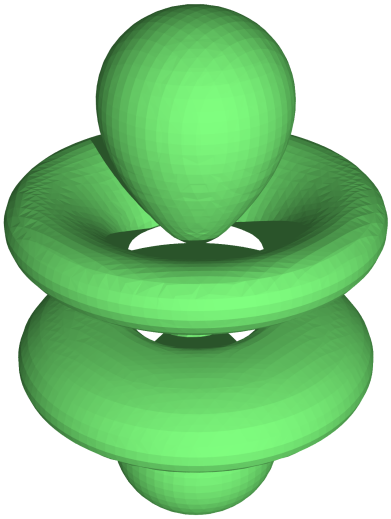
Obrázek 4.6: Řez elektronovým orbitalem typu **d** s kvantovými čísly $n = 5, \ell = 2, m = 0$. Orbital je vykreslen pro čtyři různé hodnoty hustoty pravděpodobnosti.



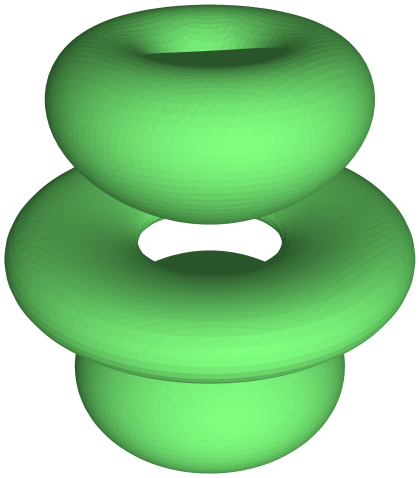
(a) $n = 3, \ell = 1, m = 0$



(b) $n = 3, \ell = 2, m = 0$



(c) $n = 4, \ell = 3, m = 0$



(d) $n = 4, \ell = 3, m = 1$

Obrázek 4.7: Vizualizace elektronových orbitalů v atomu vodíku pro různá kvantová čísla. K polygonizaci byl použit algoritmus *Dual Marching Cubes*.

Závěr

Na začátku této bakalářské práce jsme uvedli různé možnosti vyjádření plochy v trojrozměrném prostoru. Dále jsme se zabývali pouze implicitně zadanými plochami a především algoritmy pro jejich numerické vykreslování. Popsali jsme principy vybraných algoritmů a porovnali jejich hlavní vlastnosti, výhody a nevýhody. Podrobněji jsme se zaměřili na algoritmus *Marching Cubes*. Představili jsme také některé algoritmy z kategorie tzv. duálních metod a vysvětlili jsme, čím se od *Marching Cubes* liší.

V následující kapitole jsme detailně rozebrali algoritmus *Dual Marching Cubes*, který kombinuje přístupy duálních metod a algoritmu *Marching Cubes*. Uvedli jsme definici a popsali význam kvadratické chybové funkce, která je stěžejním bodem tohoto algoritmu. Popsali jsme datovou strukturu oktalový strom a to, jakým způsobem ho algoritmus sestavuje. V závěru kapitoly jsme se věnovali konstrukci mřížky, která je topologicky duální k oktalovému stromu, a generování polygonů výsledné plochy. Jednotlivé kroky algoritmu jsme postupně znázornili na zjednodušených dvourozměrných příkladech.

Ve třetí kapitole jsme zaměřili pozornost na implementaci algoritmu *Dual Marching Cubes* v open-source algebraickém systému SageMath. Popsali jsme rozhraní funkce `contour_plot3d` a způsob, jakým lze implementaci použít. Poté jsme se věnovali vytvoření kvadratické chybové funkce. Zjistili jsme, že je to kritická část algoritmu z hlediska časové náročnosti. Proto jsme funkci efektivně implementovali pomocí kompilovaného jazyka Cython a dosáhli tak podstatného zrychlení. Nakonec jsme program otestovali pomocí tzv. *doctestů*.

Poslední kapitola obsahuje ukázky použití. Porovnali jsme naši implementaci algoritmu *Dual Marching Cubes* s implementací *Marching Cubes*, kterou obsahuje systém SageMath. Nejdříve jsme srovnali dobu polygonizace při použití obou algoritmů. Ukázalo se, že kvůli náročnému výpočtu chybové funkce je algoritmus *Dual Marching Cubes* mnohonásobně pomalejší než *Marching Cubes*. Poté jsme porovnali kvalitu zobrazení výsledných ploch. Potvrdilo se, že algoritmus *Dual Marching Cubes* je schopný lepé odhalit ostré hrany plochy a v některých případech dokáže lépe zachytit celkový tvar plochy při použití

nižšího počtu výsledných polygonů. Zjistili jsme také, že je obtížné nalézt vhodnou kombinaci parametrů algoritmu *Dual Marching Cubes* tak, aby vykreslil plochu v požadované přesnosti. Nakonec jsme využili naši implementaci k vizualizaci elektronových orbitalů atomu vodíku.

Literatura

[1] Jain, R.; Kasturi, R.; Schunck, B.: *Machine Vision*. Computer Science Series, 1995, ISBN 9780070320185.

[2] Lorensen, W. E.; Cline, H. E.: Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *SIGGRAPH Comput. Graph.*, ročník 21, č. 4, Srpen 1987: s. 163–169, ISSN 0097-8930, doi:10.1145/37402.37422. Dostupné z: `http://academy.cba.mit.edu/classes/scanning_printing/MarchingCubes.pdf`

[3] Cline, H.; Lorensen, W.: System and method for the display of surface structures contained within the interior region of a solid body. Prosinec 1987, US Patent 4,710,876. Dostupné z: `https://www.google.com/patents/US4710876`

[4] Chernyaev, E. V.: Marching Cubes 33: Construction of Topologically Correct Isosurfaces. Dostupné z: `http://hyperfun.org/FHF_Log/Chernyaev_MC33.pdf`

[5] Gourlayn, M. J.: *Fluid Simulation for Video Games* [online]. Září 2014 [cit. 2016-12-24]. Dostupné z: `https://software.intel.com/en-us/articles/fluid-simulation-for-video-games-part-18`

[6] Gibson, S. F. F.: Constrained Elastic Surface Nets: Generating Smooth Surfaces from Binary Segmented Data. 1999. Dostupné z: `http://www.merl.com/publications/docs/TR99-24.pdf`

[7] Ju, T.; Losasso, F.; Schaefer, S.; aj.: Dual Contouring of Hermite Data. ročník 21, č. 3, Červenec 2002. Dostupné z: `http://www.frankpetterson.com/publications/dualcontour/dualcontour.pdf`

[8] Wolfram Research: *ContourPlot3D - Wolfram Language Documentation* [online] [cit. 2016-11-20]. Dostupné z: http://reference.wolfram.com/language/ref/ContourPlot3D.html

[9] Maplesoft: *Three-dimensional implicit plotting - Maple Help* [online] [cit. 2016-11-20]. Dostupné z: http://www.maplesoft.com/support/help/maple/view.aspx?path=plots2Fimplicitplot3d

[10] Morales, G.: *Ezimplot3: implicit 3D functions plotter* [online] [cit. 2016-11-20]. Dostupné z: http://www.mathworks.com/matlabcentral/fileexchange/23623-ezimplot3--implicit-3d-functions-plotter

[11] The Sage Development Team: *Implicit Plots - Sage Reference Manual v7.4: 3D Graphics* [online] [cit. 2016-11-20]. Dostupné z: http://doc.sagemath.org/html/en/reference/plot3d/sage/plot/plot3d/implicit_plot3d.html

[12] Schaefer, S.; Warren, J.: Dual Marching Cubes: Primal Contouring of Dual Grids. 2004. Dostupné z: https://www.cs.rice.edu/~jwarren/papers/dmc.pdf

[13] Garland, M.; Heckbert, P.: Surface Simplification Using Quadric Error Metrics. 1998. Dostupné z: http://mgarland.org/files/papers/quadrics.pdf

[14] Lindstrom, P.: Out-of-core Simplification of Large Polygonal Models. 2000. Dostupné z: http://www-evasion.imag.fr/people/Franck.Hetroy/Teaching/Geo3D/Articles/lindstrom2000.pdf

[15] The Sage Development Team: *General Conventions — Sage Developer's Guide v7.4* [online] [cit. 2016-12-24]. Dostupné z: http://doc.sagemath.org/html/en/developer/coding_basics.html

[16] Velho, L.; Gomes, J.; de Figueiredo, L.: *Implicit Objects in Computer Graphics*. Springer New York, 2007, ISBN 9780387216201. Dostupné z: https://books.google.cz/books?id=CGkKBwAAQBAJ

Příloha **A**

Seznam použitých zkratek

QEF Quadratic Error Function

MC Marching Cubes

DMC Dual Marching Cubes

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `notebooks`
  - `examples.ipynb` ..... příklady použití implementace
  - `orbitals.ipynb` ..... ukázka elektronových orbitalů
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `BP_Koza_Jan_2017.pdf` ..... text práce ve formátu PDF