Děkuji RNDr. Josefu Pelikánovi za hodnotné rady a velmi motivující a odborné vedení během mé práce na tomto projektu.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 27. května 2009 Martin Podloucký

Obsah

Seznam obrázků

Název práce:	Editor implicitních ploch
Autor:	Martin Podloucký
Katedra (ústav):	Kabinet software a výuky informatiky
Vedoucí bakalářské práce:	RNDr. Josef Pelikán
e-mail vedoucího:	Josef.Pelikan@mff.cuni.cz

Abstrakt: Tato práce se zabývá využitím implicitně definovaných ploch k modelování trojrozměrných objektů v počítačové grafice. Nejprve na teoretické úrovni rozebírá běžné postupy a nástroje, které se k takovému modelování používají a soustředí se na jejich matematické pozadí. Poté následuje dokumentace k interaktivnímu editoru BlobsEd, jenž byl vyvinut, aby demonstroval základní možnosti těchto nástrojů pro vytváření různých trojrozměrných tvarů a ploch.

Klíčová slova: implicitní plochy, bloby, metaballs.

Title:	Implicit surfaces editor
Author:	Martin Podloucký
Department:	Department of Software and Computer Science Education
Supervisor:	RNDr. Josef Pelikán
Supervisor‘s e-mail address:	Josef.Pelikan@mff.cuni.cz

Abstrakt: This thesis dwells on usage of implicitely defined surfaces for modeling three-dimensional objects in computer graphics. At the beginnig common approach to this kind of modeling tools is presented and its mathematical background is revealed. Then a documentation for BlobsEd editor follows which was developed to demonstrate basic capabilities of those tools to create various kinds of three-dimensional shapes and surfaces.

Keywords: implicit surfaces, blobs, metaballs.

Kapitola 1

Úvod

Cílem této práce je popsat a demonstrovat základní techniky modelování pomocí takzvaných implicitních ploch. V první kapitole se těmito technikami a nástroji zaobíráme na teoretické úrovní. Budeme se snažit vysvětlit, co to implicitní plochy jsou, jak jsou matematicky definovány a jak je možné pomocí nich vytvářet různé trojroměrné tvary způsobem, který je intuitivní a nespoléhá na to, že uživatel rozumí matematickému pozadí celé věci.

V následujících kapitolách je poté popsáno fungování jednoduchého editoru implicitních ploch, který byl vyvinut právě proto, aby demonstroval použití technik a postupů, které jsou vyloženy v teoretické části a umožnil tak čtenáři si práci s těmito plochami vyzkoušet.

1.1 Typografické konvence

Následují typografické konvence použité v celém dokumentu.

Bezpatkové písmo budeme používat pro názvy počítačových programů a jejich komponent, ovládacích prvků, ale také tříd, rozhraní a pro všechny pojmy z technické javadoc dokumentace k editoru BlobsEd. Vzhledem k tomu, že nejen grafické rozhraní tohoto programu ale i komentáře v kódu a celá javadoc dokumentace jsou psány anglicky, budeme v tomto textu používat dvojjazyčné označení pro některé pojmy, abychom čtenáři usnadnili orientaci. Vždy, když poprvé použijeme takový pojem, bude v závorce uveden jeho anglický ekvivalent používaný v editoru, například *scéna* (scene).

Kurzívou bude vždy vyznačen první výskyt pojmu, který je nějakým způsobem důležitý a který bude nadále používán bez další změny písma, například *dotykový vektor*. Kurzívou dále píšeme znění matematických vět a také jiné pasáže, které chceme nějak výrazněji odlišit od ostatního textu.

Tučným písmem budeme vyznačovat části textu, které by čtenář neměl přehlédnout nebo by jim měl z nějakého důvodu věnovat zvýšenou pozornost.

Kapitola 2

Implicitní plochy

Pokud se v počítačové grafice využívají k modelování geometrických těles křivky či plochy, většinou se volí takové, které jsou definovány parametrickým předpisem, viz. například Beziérovy křivky či pláty, které používá velké množství kreslících a modelovacích programů. Kdo někdy používal nějaký vektorový grafický editor (například CorelDRAW nebo Inkscape) ví, že s Beziérovými křivkami se velice pohodlně pracuje, především díky tomu, že uživatel má dobrou kontrolu nad přesným tvarem křivky. Některé praktické aplikace však i přesto používají modelování, založené na implicitním vyjádření objektů.

Myšlenku vyjádřit těleso pomocí implicitních funkcí používá již v roce 1982 Blinn, který se v [1] zabýval zobrazováním izoploch elektrického potenciálu kolem elementárních částic. Už zde vlastně nastínil většinu technik použitých v této práci. Jeho myšlenky byly postupně dále zobecněny a objekty pomocí nich vzniklé dostaly rozličná jména jako například *blobby objects* či *metaballs*.

Může se zdát, že modelování pomocí ploch zadaných implicitně pomocí rovnice musí být nepohodlné a vhodné spíše pro specifické problémy, jako je například zobrazování tvaru atomových orbitalů a podobně. Lze však vymyslet postupy a techniky, jejichž základ je nastíněn právě v práci [1], které zpřístupní tento způsob vytváření objektů i běžnému uživateli grafických nástrojů a dají mu velkou kontrolu nad tvarem výsledného objektu, aniž by musel rozumět matematickému pozadí celé věci či pracovat přímo s rovnicemi, které definují tvar ploch.

2.1 Základní pojmy a definice

Za implicitně zadanou plochu v prostoru budeme považovat každou množinu bodů vyhovujících rovnici

$$f(x, y, z) = 0, \tag{2.1}$$

kde f je nějaká reálná funkce tří reálných proměnných. Takové plochy budeme také někdy nazývat *izoplochy*, neboť spojují ty body v prostoru, ve kterých funkce f nabývá všude stejné hodnoty 0. Je zřejmé, že tato funkce nemusí být ani příliš divoká, abychom obdrželi velmi komplikovanou izoplochu. Pracovat s objekty, jejichž hranice je tvořena takto obecně zadaným povrchem, by ale bylo pro uživatele velice neintuitivní. Představíme tedy přístup, jak pomocí implicitních ploch modelovat objekty způsobem, který nevyžaduje pracovat přímo s předpisem funkce f.

Modelování objektů bude sestávat z práce s takzvanými *generátory* (generator). Generátorem nazveme každou neprázdnou kompaktní konvexní množinu $G \subset \mathbb{R}^3$. Definujme dále pojem *dotykový vektor* (touch vector) z bodu P do generátoru G jako takový vektor $\vec{v}_p$, pro který platí $(P + \vec{v}_p) \in G$ a zároveň

$$|\vec{v}_p| = \min\{|\vec{v}| \mid (P + \vec{v}) \in G\}. \tag{2.2}$$

Konvexnost množiny bodů generátoru požadujeme proto, aby dotykový vektor existoval pro každý bod v prostoru nejvýše jeden. To nám později usnadní výpočet normály k izoploše a s tím spojené výpočty osvětlení. Požadavek na kompaktnost (tedy omezenost a uzavřenost) množiny je spíše teoretického charakteru. Je zřejmé, že při modelování chceme pracovat pouze s omezenými objekty. Pokud by množina bodů generátoru nebyla navíc uzavřená, nemuselo by existovat minimum ve vztahu (2.2). Mohli bychom jej sice nahradit infimem, ale to je zbytečná komplikace, neboť v praktickém světě počítačové grafiky není mezi otevřenou a uzavřenou **konvexní** množinou žádný podstatný rozdíl.

Věta 2.1.1. *Nechť $G \subset \mathbb{R}^3$ je kompaktní množina. Potom pro každý bod $P \in \mathbb{R}^3$ existuje alespoň jeden vektor $\vec{v}_p$, pro který platí zároveň*

1. $(P + \vec{v}_p) \in G$
2. $|\vec{v}_p| = \min\{|\vec{v}| \mid (P + \vec{v}) \in G\}$

Důkaz. Mějme zadán bod $P \in \mathbb{R}^3$. Definujme zobrazení $f_P : G \to \mathbb{R}$ předpisem

$$f_P(X) = |X - P|$$

Tato funkce je zřejmě spojitá. Potom ovšem na kompaktní množině nabývá svého minima. □

Věta 2.1.2. *Nechť $G \subset \mathbb{R}^3$ je konvexní množina. Potom pro každý bod $P \in \mathbb{R}^3$ existuje nejvýše jeden vektor $\vec{v}_p$, pro který platí zároveň*

1. $(P + \vec{v}_p) \in G$
2. $|\vec{v}_p| = \min\{|\vec{v}| \mid (P + \vec{v}) \in G\}$

Důkaz. Pro spor předpokládejme, že pro nějaký bod $Q \in \mathbb{R}^3$ existují dva vektory $\vec{u}_1 \neq \vec{u}_2$ vyhovující oběma podmínkám. Zřejmě platí $|\vec{u}_1| = |\vec{u}_2|$ a z toho hned $\vec{u}_1, \vec{u}_2 \neq \vec{0}$.

Označíme-li $S = Q + \vec{u}_1$ a $T = Q + \vec{u}_2$, mohou nastat dvě možnosti:

1. Body S, Q, T leží na přímce. Z konvexnosti G potom plyne $Q \in G$ a tedy dotykový vektor z Q do G je nulový, tedy kratší než $\vec{u}_1$ i $\vec{u}_2$, což je spor.
2. Body S, Q, T tvoří rovnostranný trojúhelník. Nechť bod P je pata výšky vedené z bodu Q. Z konvexnosti G potom plyne $P \in G$ a jelikož výška na základnu v rovnoramenném trojúhelníku je vždy kratší než jeho ramena, dotykový vektor $\vec{v}_p = P - Q$ je kratší než $\vec{u}_1$ i $\vec{u}_2$, což je spor.

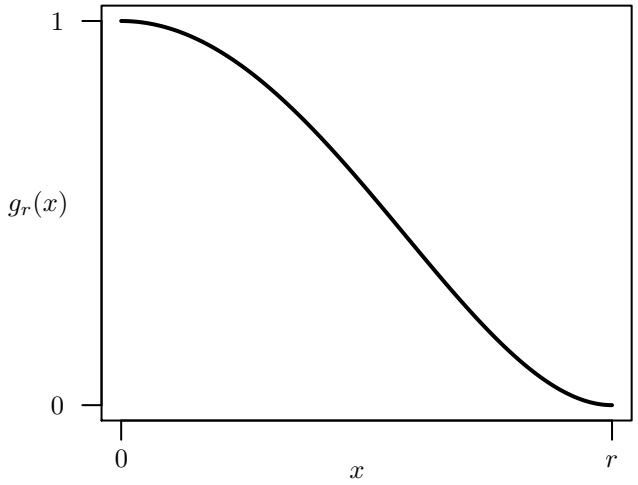
□

Z předchozích dvou tvrzení plyne, že dotykový vektor je dobře definován. Máme-li zadán generátor G, pro každý bod v prostoru existuje **právě jeden** dotykový vektor do G. Nyní již můžeme definovat *vzdálenost* (distance) z bodu P do generátoru G jako velikost dotykového vektoru z P do G.

Každé reálné funkci jedné reálné proměnné, do které budeme dosazovat vzdálenosti od nějakého generátoru, budeme říkat *potenciálová funkce* (potential function). Příkladem takové funkce může být například

$$g_r(d) = \begin{cases} \left(1 - \frac{d^2}{r^2}\right)^2 & d \in \langle 0, r) \\ 0 & d \geq r, \end{cases}$$

převzatá z [3, str. 225]. Konstanta r udává takzvaný *poloměr* (radius) generátoru. Ke každému generátoru tedy přiřadíme nějakou potenciálovou funkci. BlobsEd editor se omezuje pouze na ten speciální případ, kdy se potenciálové funkce jednotlivých generátorů liší pouze konstantou r.



Obrázek 2.1: Průběh funkce $g_r(x)$

Mějme tedy v rovině jeden generátor G o poloměru r. Jestliže $d(x, y, z)$ je funkce udávající vzdálenost z bodu $P = (x, y, z)$ do generátoru G, potom funkce $g_r \circ d$ vytvoří kolem tohoto generátoru skalární pole. Pro naše účely si rovnici (2.1) napíšeme ve tvaru

$$f(x, y, z) = h, \tag{2.3}$$

kde

$$f(x, y, z) = g_r(d(x, y, z)).$$

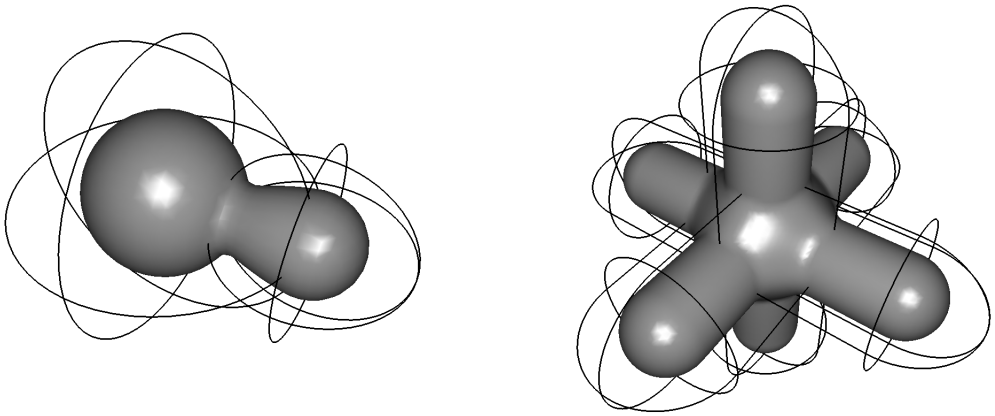
Implicitní rovnice (2.3) potom definuje izoplochu spojující všechny body, ve kterých funkce f nabývá hodnoty h. Tuto hodnotu budeme nazývat *hladina* (level).

Podobně můžeme vytvořit skalární pole kolem množiny generátorů $G_1, \ldots, G_n$ o poloměrech $r_1, \ldots, r_n$, položíme-li

$$f(x, y, z) = \sum_{k=1}^{n} g_{r_k}(d_k(x, y, z)),$$

kde $d_1, \ldots, d_n$ jsou funkce udávající vzdálenosti od jednotlivých generátorů. Objekt, jehož objem je omezen plochou definovanou výše popsaným způsobem přes množinu nějakých generátorů, budeme nazývat *blob* (blob).

Všimněme si průběhu funkce g_r (obrázek 2.1). Pro vzdálenost větší, než je poloměr generátoru, už jsou její hodnoty nulové. Vliv jednoho generátoru na celkové skalární pole blobu je tedy omezen jeho poloměrem. Část prostoru, na kterou je omezen vliv určitého generátoru, budeme nazývat *obálka* (envelope). Podobně můžeme také definovat obálku blobu jako sjednocení obálek všech jeho generátorů.



Obrázek 2.2: Ukázka dvou blobů vytvořených pomocí dvou bodových generátorů o různých poloměrech (vlevo) nebo tří úsečkových generátorů se stejným poloměrem (vpravo). Čáry kolem ploch znázorňují obálky.

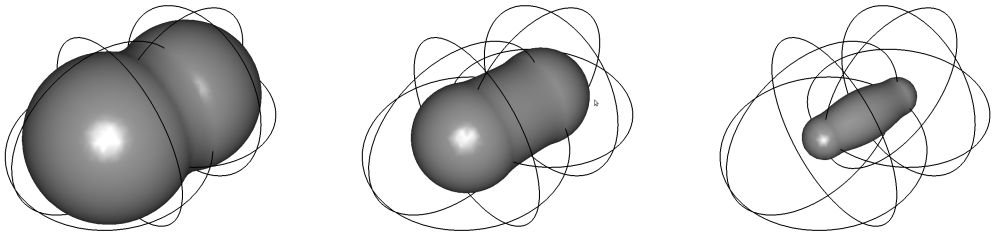
Škálu tvarů, které lze pomocí těchto nástrojů vymodelovat, můžeme dále rozšířit tak, že každému generátoru přiřadíme reálné číslo w, kterému budeme říkat *váha* (weight). Jestliže generátorům $G_1, \ldots, G_n$ přiřadíme váhy $w_1, \ldots, w_n$, bude funkce f vypadat takto

$$f(x, y, z) = \sum_{k=1}^{n} w_k g_{r_k}(d_k(x, y, z)).$$

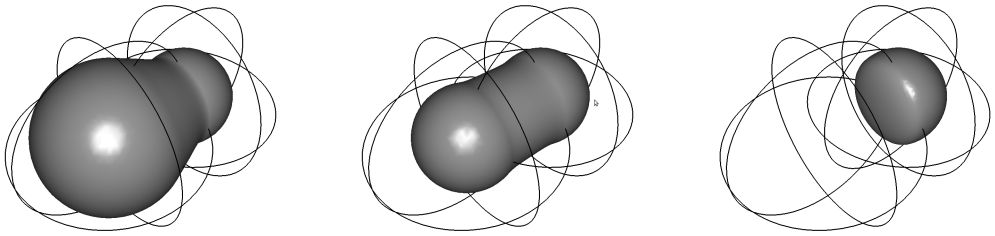
2.2 Modelování pomocí implicitních ploch

V předchozí kapitole jsme předvedli způsob, jak vytvářet izoplochy tak, že skládáme dohromady jakési komponenty – generátory, namísto abychom pracovali přímo s matematickým zápisem funkcí, které tyto plochy definují. Za generátory můžeme vzít jakékoliv konvexní objekty v prostoru, například body, úsečky, konvexní polygony, dokonce konvexní sítě trojúhelníků a tak dále. Představu, jak bude vypadat izoplocha kolem určitého generátoru s potenciálovou funkcí podobnou funkci g_r získáme tak, že si generátor představíme jakoby nafouknutý. Z bodu tedy vznikne koule, z úsečky jakási kapsle apod. (viz. obrázek 2.2). Když měníme hladinu blobu, jeho povrch se zmenšuje nebo naopak zvětšuje. Když měníme váhu generátoru, ovlivňujeme tím, zhruba řečeno, jak moc přispívá k celkovému povrchu blobu (viz. obrázky 2.3 a 2.4).

Způsobem, jakým jsou bloby a generátory v editoru BlobsEd implementovány, se budeme podrobně zabývat v kapitole o datové vrstvě.



Obrázek 2.3: Blob tvořený dvěma bodovými generátory se stejným poloměrem. Demonstrace vlivu různých hodnot hladiny na tvar plochy. Zleva doprava $h = 0{,}1$, $h = 0{,}5$, $h = 0{,}9$.



Obrázek 2.4: Blob tvořený dvěma bodovými generátory se stejným poloměrem. Demonstrace vlivu různých hodnot váhy generátoru blíže ke kameře na tvar plochy. Zleva doprava $w = 3$, $w = 1$, $w = -1$.

2.3 Zobrazování blobů

Aby uživatel vůbec mohl s generátory a bloby pracovat, je třeba je nějakým způsobem zobrazit. K tomuto úkolu lze přistoupit ze dvou směrů. Jedna možnost je zvolit nějakou přímou zobrazovací metodu, jako je vrhání či sledování paprsku. *Vrhání paprsku* (ray-casting) funguje jednoduše tak, že se každým pixelem, který chceme zobrazit, vyšle paprsek směrem od kamery. Když tento protne nějakou izoplochu, daný pixel vyplníme této ploše přiřazenou barvou. Jak později uvidíme v dokumentaci k zobrazovací vrstvě editoru, ze znalosti dotykového vektoru ke generátoru dokážeme snadno vypočítat normálu k izoploše v daném bodě, což nám umožní aplikovat také nějaký osvětlovací model.

Pokud známe normály k plochám, nic nám nebrání použít namísto vrhání paprsku pokročilejší metodu, totiž velmi známé *sledován paprsku* (ray-tracing), kdy se paprsek sleduje i po vícenásobném odražení od plochy. Tímto bychom získali další zobrazovací schopnosti jako jsou odlesky, zrcadlení, stíny či dokonce průhlednost. Tyto přímé zobrazovací metody ovšem vyžadují, abychom byli schopni vypočítat s dostatečnou přesností průsečík s izoplochou. To je záležitost vyžadující použití některé numerické metody na hledání nulových bodů funkce. Tyto přímé metody se vyznačují vysokou kvalitou zobrazení, jsou však velmi náročné na čas. My se proto jejich implementaci nebudeme věnovat, neboť naším cílem je vytvořit interaktivní editor, který bude schopen rychlé odezvy na změny provedené uživatelem.

Další možností, jak zobrazit tvar izoplochy, je její převedení na nějakou jinou reprezentaci, například na ploškovou. Existují i pokusy o převádění na parametrické plochy, tyto ovšem také pomineme. Pro náš editor je nejvhodnější převést plochu na síť trojúhelníků, kterou potom snadno zobrazíme pomocí klasických metod. Převod povrchu na trojúhelníky (takzvaná *polygonizace*) má výhodu v tom, že polygonizační algoritmus stačí spustit pouze ve chvíli, kdy je třeba změnit tvar blobu – přidáním nového generátoru, nastavením parametrů již existujícího generátoru apod. Pokud se s blobem dějí pouze transformace, kdy se tvar izoplochy nemění (například posun, rotace atd.), není potřeba síť znovu přepočítávat. Takový přístup se v editoru velmi vyplatí, neboť polygonizace může trvat delší dobu a proto je dobré ji spouštět, jen když je to opravdu nezbytně nutné. Navíc to uživateli umožní prohlížet si scénu v reálném čase i se vším stínováním a osvětlením, neboť normály k povrchu blobu se také mění, jen když se mění jeho tvar. Za rychlost však zaplatíme cenu v podobě méně přesného zobrazení plochy. Později však ukážeme, že tato nepřesnost je ve většině případů únosná.

K převodu izoplochy na síť trojúhelníků použijme algoritmus známý pod jménem *pochodující kostky* (marching cubes), který publikovali Lorensen a Cline roku 1987 [2]. Jde o v principu velmi jednoduchý postup, jak získat dobrou aproximaci povrchu blobu. Tento algoritmus byl do nedávna zatížen patentem, avšak dnes už je možné jej volně používat, neboť patent vypršel roku 2005.

2.4 Algoritmus marching cubes

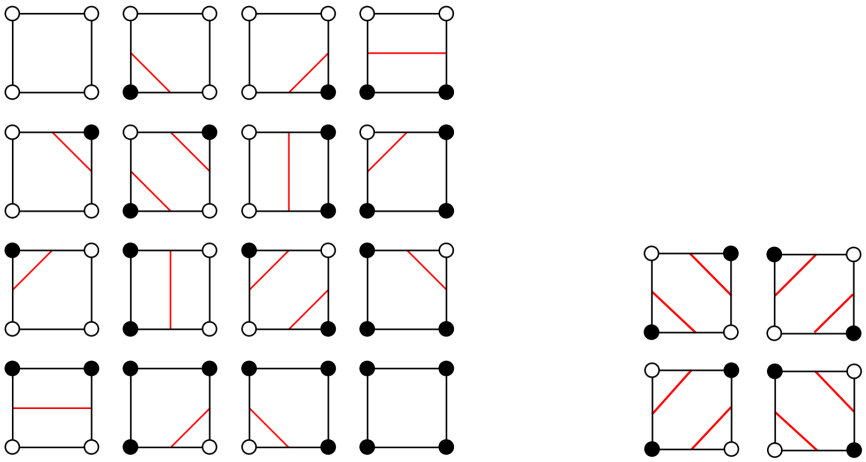
Jak jsme již výše vyložili, plocha určitého blobu se může nacházet pouze v prostoru omezeném jeho obálkou. Pokud kolem této obálky vytvoříme kvádr, který ji celou obsahuje, máme jistotu, že izoplocha se nachází celá uvnitř tohoto kvádru. Tento potom můžeme rozdělit na trojrozměrnou síť tvořenou krychlemi s konstantní délkou hrany. Čím kratší hrana krychle, tím dostaneme hustší síť a tím také přesnější aproximaci povrchu. Pokud si vezmeme jednu konkrétní krychli této sítě a spočítáme hodnoty skalárního pole blobu ve všech jejích osmi vrcholech, můžeme podle těchto hodnot přibližně odhadnout, jak vypadá část izoplochy uvnitř této krychle.

Fungování algoritmu snadno nastíníme, přesuneme-li se na chvíli do dvoudimenzionálního prostoru a budeme místo izoploch uvažovat izočáry. Veškeré dosud uvedené definice zůstanou ve 2D prostoru naprosto analogické, jestliže v nich očividným způsobem vynecháme třetí proměnnou. Hraniční kvádr tedy bude hraničním obdélníkem a namísto krychlí budeme pracovat se čtverci. V každém vrcholu V čtverce mohou podle jeho pozice ve skalárním poli tvořeném funkcí $f(x, y)$ nastat dva případy. Buď $f(V) \leq h$ nebo $f(V) > h$. První případ budeme pro názornost značit plným puntíkem, druhý puntíkem prázdným. Máme tedy šestnáct různých případů, které mohou nastat (viz. obrázek 2.5).

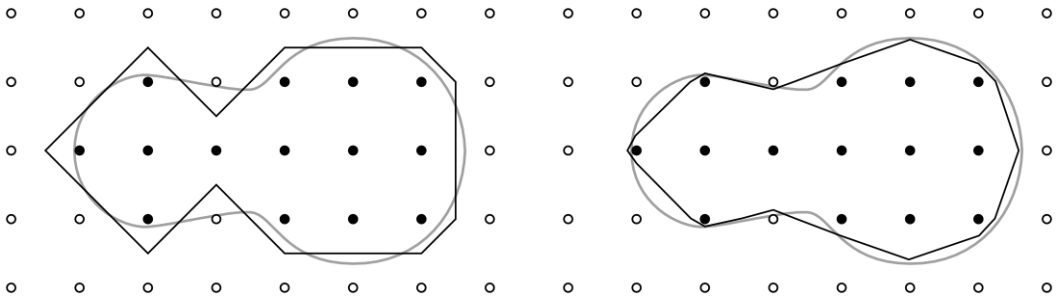
Úsečky uvnitř čtverce představují aproximaci izočáry tímto čtvercem procházející. Tyto úsečky protínají hranu čtverce vždy přesně v polovině. Kvůli tomu bychom na dobrou aproximaci izočáry potřebovali velmi hustou síť. Můžeme si však vypomoct tak, že nebudeme koncové body úseček dávat přesně do poloviny hran, nýbrž jejich pozici vypočteme lineární interpolací hodnot skalárního pole v koncových bodech příslušné hrany čtverce. Označme P_1, P_2 tyto koncové body a h_1, h_2 jim příslušné hodnoty. Nechť se body P_1, P_2 liší například v x-ové souřadnici. Průsečík s hranou na ose x pak vypočteme podle klasického vzorečku pro lineární interpolaci

$$x = x_1 + (h - h_1)\frac{x_2 - x_1}{h_2 - h_1}.$$

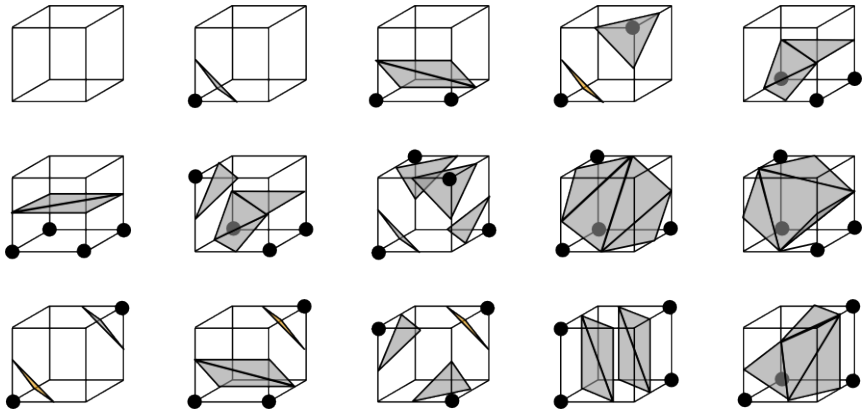
Rozdíl mezi interpolovanou a neinterpolovanou aproximací ukazuje obrázek 2.6.



Obrázek 2.5: Šestnáct různých případů, které mohou nastat při různém umístění čtverce ve skalárním poli (vlevo). Nejednoznačné případy algoritmu pochodujících čtverců (vpravo).



Obrázek 2.6: Rozdíl mezi interpolovanou (vpravo) a neinterpolovanou (vlevo) aproximací izočáry.



Obrázek 2.7: Patnáct unikátních konfigurací krychlí pro trojrozměrnou verzi marching cubes algoritmu.

Algoritmus pochodujících kostek v trojrozměrném prostoru je dvojrozměrnému případu velice podobný. Namísto čtverců budeme používat krychle a místo úseček budeme izoplochu uvnitř těchto krychlí aproximovat skupinami trojúhelníků. Jediné, co tedy zbývá udělat, je nadefinovat všech 256 možností rozestavení trojúhelníků uvnitř krychle v závislosti na hodnotách skalárního pole v jejích vrcholech. Opravdu unikátních konfigurací, pomineme-li symetrii a otáčení, je pouze 15, viz. obrázek 2.7.

Máme-li tedy k dispozici tabulku se všemi 256 konfiguracemi, není implementace algoritmu obecně nic těžkého. Jediný lehce netriviální problém může spočívat v tom, jak zorganizovat kód, abychom nepočítali funkci definující povrch blobu vícekrát, než je nezbytně třeba. Krychle v síti může mít totiž až dvacet šest sousedů, se kterými sdílí vrcholy a hrany. Detailněji se implementací budeme zabývat v části popisující zobrazovací vrstvu editoru.

Algoritmus marching cubes má však jeden problém v tom, že může nastat nejednoznačná situace. Na obrázku 2.5 jsou znázorněny nejednoznačné konfigurace v dvojrozměrném případě. Kvůli této skutečnosti mohou ve výsledné aproximaci plochy být chyby v podobě děr, které v izoploše ve skutečnosti nejsou. Přestože existuje způsob, jak algoritmus upravit tak, aby takové situace nenastávaly, BlobsEd tento problém nijak neřeší, neboť se objevuje spíše výjimečně a při velmi nízkém rozlišení sítě. Naším cílem není dokonale přesné zobrazování. Chceme spíše co nejrychleji zprostředkovat uživateli hrubou představu o tvaru blobu.

Nejednoznačnosti v aproximaci plochy lze řešit také použitím jiného polygonizačního algoritmu jménem *pochodující čtyřstěny* (marching tetrahedra). Ten prostorovou mřížku ještě dále dělí tak, že každou krychli rozdělí na pět čtyřstěnů. Existují dvě možnosti jak toto rozdělení provést, a ty se musí v zájmu zachování návaznosti

v mřížce pravidelně střídat, nebo se musí krychle dělit na 6 či 24 částí. To ovšem neúměrně zvyšuje počet trojúhelníků potřebných k reprezentaci povrchu, což má za následek, že tento algoritmus se v praxi příliš nepoužívá a ani my se mu zde nebudeme dále věnovat.

Kapitola 3

Dokumentace k editoru BlobsEd

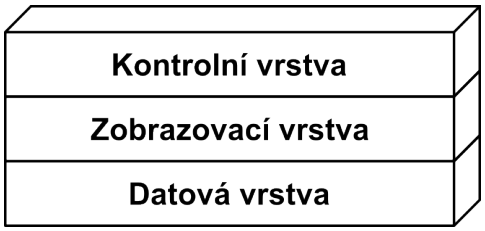
BlobsEd je programem, jenž poskytuje nástroje k modelování trojrozměrných objektů definovaných pomocí implicitně zadaných ploch v prostoru. Jde o multiplatformní open source software naprogramovaný v jazyce Java, šířený pod licencí GNU GPL a vyvinutý především jako takzvaný ročníkový projekt na Matematicko-fyzikální fakultě Univerzity Karlovy v Praze. Jeho smyslem není zkoumání nových možností a pohledů na téma, nýbrž především demonstrace jedné konkrétní metody modelování.

Text této kapitoly je programátorskou dokumentací k editoru. Jejím cílem je popsat a vysvětlit strukturu kódu tohoto programu a vyložit všechny netriviální algoritmy, principy a myšlenky, které byly při jeho návrhu a vývoji použity. Naopak zde nejsou popsány vyloženě technické detaily implementace, konkrétní způsoby jak spolu jednotlivé komponenty komunikují apod. Toto je vysvětleno v javadoc dokumentaci a v komentářích v kódu. Tento dokument a javadoc na sebe navzájem odkazují a pro detailní pochopení fungování programu jsou oba důležité.

Následující odstavce a kapitoly předpokládají, že čtenář si práci s editorem BlobsEd alespoň povrchně vyzkoušel. Návod, jak tento program spustit a používat, je součástí přílohy B.

3.1 Struktura aplikace

Nyní vyložíme, podle jakých principů je kód editoru BlobsEd strukturován. Základní myšlenkou je rozdělení logiky aplikace do vrstev, které fungují tak, že každá vrstva vidí jen ty, které jsou pod ní. Všechny závislosti mezi třídami aplikace tedy směřují



Obrázek 3.1: Rozložení vrstev editoru.

buď dolů nebo vodorovně v rámci vrstev, nikdy ne směrem nahoru (obrázek 3.1). To vede k omezení závislosti mezi těmito třídami a tedy k menší provázanosti jednotlivých komponent, což dělá editor modulárnější a snadněji rozšiřitelný. Vrstvy aplikace směrem zdola nahoru:

Datová vrstva představuje data, jako jsou pozice generátorů a blobů v prostoru, jejich parametry apod. Je reprezentována třídami odvozenými od tříd Document a Entity.

Zobrazovací vrstva se stará o interpretaci datové vrstvy. Patří sem jednak renderování scény pomocí OpenGL, ale také implementace algoritmu marching cubes.

Kontrolní vrstva umožňuje uživateli pracovat s daty a měnit způsob jejich zobrazení. Patří sem všechny prvky grafického rozhraní, jako jsou panely nástrojů, nabídky, různé panely pro nastavení parametrů, ale i třídy – takzvaní *návrháři* (designer), kteří umožňují interaktivní návrh generátorů a blobů přímo v oknech zobrazujících scénu.

V tomto výčtu vrstev jsme se odkazovali na pojem *scéna* (scene). Scénou je zde myšleno v podstatě to, co vidí uživatel při pohledu na data, který mu zprostředkuje zobrazovací vrstva. Souboru nastavení, který ovlivňuje způsob tohoto zobrazení, budeme říkat *prostředí* (environment) scény. Scéna je tedy spojením dat a prostředí a zasahuje proto jak do datové, tak do zobrazovací vrstvy. V kódu editoru je scéna reprezentována instancí třídy Scene. Takováto třída, která vlastně vybočuje z našeho třívrstevnatého paradigmatu, je potřeba proto, že právě scéna je to, co se ukládá na disk, pokud uživatel ukončí práci s editorem. Nestačí uložit pouze data, neboť by tak byla ztracena informace například o natočení kamery v 3D pohledu nebo polygonová síť blobů, která, jak už jsme uvedli, také nepatří do datové, nýbrž do zobrazovací vrstvy.

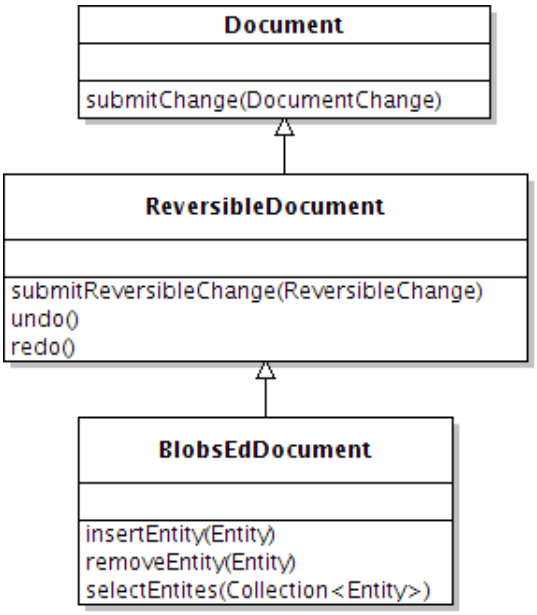
3.2 Datová vrstva

Datová vrstva obsahuje veškerá data, která plně definují všechny entity a jejich rozestavení ve scéně. Třídy odvozené od Document, kterými jsou ReversibleDocument a BlobsEdDocument, představují kontejner, jenž obsahuje objekty ve scéně. Tyto objekty jsou potom instancemi tříd implementujících rozhraní Entity, jako jsou Blob nebo PointGenerator.

Třída Document

Třída Document je kontejnerem obsahujícím data scény, což jsou například pozice a různé parametry generátorů a blobů, informace o tom, které z těchto objektů jsou právě zahrnuty do výběru a podobně. Tato třída je stěžejním prvkem datové vrstvy. Je jakýmsi správcem dat editoru, a proto také zavádí jednotný způsob provádění změn na těchto datech. Kdykoliv chce někdo provést změnu (například posunout generátor na jiné místo), nelze to provést jednoduše tak, že se přímo zavolá nějaká metoda na objektu představujícím tento generátor. Nejdřív se musí vytvořit instance třídy implementující rozhraní DocumentChange. Toto rozhraní obsahuje metodu perform, jejíž implementace provede kýženou změnu dat. Objekt, který představuje tuto změnu, se pak předá metodě Document.submitChange, která na něm zavolá metodu DocumentChange.perform.

Tento přístup má hned několik výhod. Jedna spočívá v tom, že dokument může své okolí vyčerpávajícím způsobem informovat nejen o tom, že nastala změna dat, ale také jaký druh změny to byl, jakých objektů se týkal atd. Kdokoliv implementuje rozhraní DocumentChangeListener, může o těchto změnách být informován a reagovat na ně. Přesně takto funguje zobrazovací vrstva, která při každé relevantní změně dat v dokumentu překreslí pohled. S tím souvisí druhá výhoda tohoto přístupu. Pokud by dokument informoval o každé elementární změně, jako je například posun objektu na jinou pozici v prostoru, při každé této změně by se pohled překreslil, neboť je to relevantní změna dat. Jestliže bychom ale chtěli například přesunout deset objektů najednou, pohled by se při tomto přesunu překreslil desetkrát, pro každý objekt zvlášť. To, co si samozřejmě přejeme je, aby se pohled překreslil až poté, co všech deset objektů dokončí svůj přesun. Použitím výše popsaného mechanizmu toho dosáhneme snadno, neboť celý tento přesun provedeme najednou v třídě změny, která implementuje posouvání skupiny objektů. Tím se tato změna bude chovat jako elementární a pohled se překreslí jen jedinkrát.



Obrázek 3.2: Hierarchie tříd odvozených od Document.

Od třídy Document jsou odvozeny další třídy, které dále rozšiřují její funkcionalitu. Celou hierarchii ukazuje obrázek 3.2.

Třída ReversibleDocument

Fakt, že každá změna v dokumentu je objektem, přímo vybízí k implementaci třídy, která umožní vracet změny v dokumentu zpět a dát tak uživateli přístup k známému undo/redo mechanizmu. To je přesně to, co dělá třída ReversibleDocument. Vedle nevratných změn implementujících DocumentChange existují změny vratné, které implementují od tohoto rozhraní poděděné ReversibleChange. Toto navíc přidává dvě metody, a to undo a redo, které slouží k navrácení změny zpět a k jejímu opětovném provedení. Metody stejného jména má také třída ReversibleDocument. Ty slouží k navrácení naposledy provedené změny a znovuprovedení naposledy navrácené změny. Každá třída implementující rozhraní ReversibleChangeListener může být informována o každé provedené, navrácené nebo znovu provedené změně v dokumentu.

Tíha problému, jak vrátit dokument do přesně takového stavu, v jakém se nacházel před provedením určité změny, leží na programátorovi, jenž implementuje metodu ReversibleDocument.undo. Určité problémy mohou plynout z toho, že ReversibleDocument umožňuje provádět vedle vratných změn ReversibleChange také nevratné změny DocumentChange. Princip fungování nevratných změn je následující. Dejme tomu,

že uživatel provede vratnou změnu, kterou označíme A, poté několik nevratných změn a následně vratnou změnu B. Když chce poté tyto změny vrátit zpět, vrátí nejprve B. Následně pak může vrátit až změnu A. Nevratné změny tedy vracení těch provedených před nimi neblokují. Jsou jednoduše ignorovány. Stejný princip funguje, jestliže chce uživatel navrácené změny znovu provést (operace redo). Opět se provede nejdříve A a potom B bez ohledu na nevratné změny mezi nimi.

Tato vlastnost dokumentu je důležitá z následujících důvodů. Jednak mohou alespoň teoreticky existovat změny, které z principu vrátit vůbec nelze. Takové změny zatím editor neimplementuje. Jsou ovšem také změny, jejichž vracení je z uživatelského hlediska nepohodlné. Jsou to

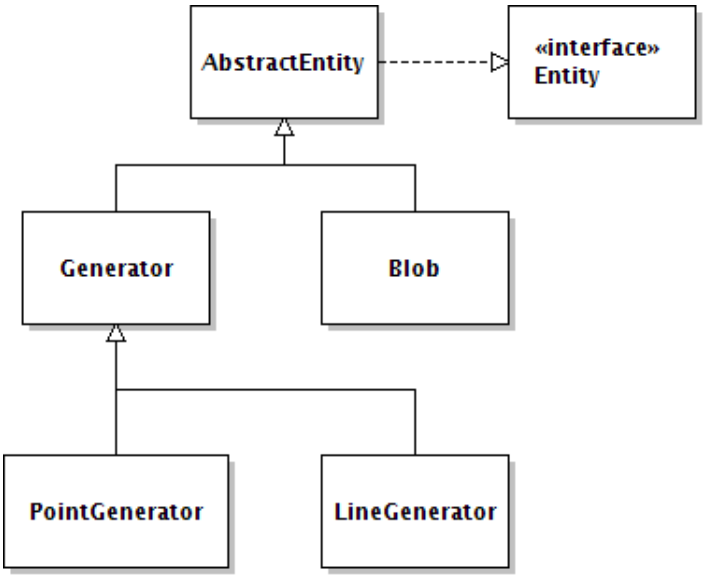
Výběr. Přestože informace o tom, které objekty jsou zrovna vybrány, je součástí dat dokumentu, BlobsEd neumožňuje vracet zpět změnu výběru, neboť v některých situacích to přináší značné implementační komplikace. Například když editor přechází do různých módů pro kontrolu scény. Naštěstí výběry v tomto editoru nejsou nikdy natolik komplikované, že by uživatel mohl příliš litovat, že se nelze vrátit zpět, když se mu povedlo omylem zrušit označení nějaké větší skupiny objektů.

Krokové změny. Jestliže uživatel například posunuje objektem tažením myší, nechce vracet zpět elementární posunutí, které nastane při každé změně polohy kurzoru, nýbrž chce vrátit objekt zpět na místo, odkud tažení začalo. Proto jsou tyto elementární změny polohy implementovány jako nevratné. Ke krokovým změnám se ještě vrátíme při popisu konkrétních změn, které editor BlobsEd implementuje.

Vzhledem k existenci nevratných změn je na programátorovi, aby manipuloval s vratnými či nevratnými změnami v dokumentu natolik obratně, aby nedošlo k žádnému nekonzistentnímu stavu. Co se týče nevratných změn výběru, tam se inkonzistenci předchází tak, že se před vracením změny zruší označení všech entit ve scéně. Mechanizmus vracení krokových a dalších nevratných změn rozvedeme, jakmile se budeme detailněji věnovat implementaci konkrétních změn dokumentu na konci tohoto oddílu.

Rozhraní Entity

Než vyložíme fungování třídy BlobsEdDocument, která je kontejnerem dat v editoru BlobsEd, nejprve popíšeme hierarchii tříd, které tato data představují.



Obrázek 3.3: Hierarchie tříd a rozhraní představujících data dokumentu

Všechny třídy tvořící data dokumentu implementují rozhraní Entity. Hierarchie tříd představujících data dokumentu je vyobrazena na obrázku 3.3. Každé třídě, jež implementuje toto rozhraní, budeme říkat *entita* (entity). Každá entita, kterou je možné vložit do dokumentu, tedy musí implementovat funkcionalitu, tímto rozhraním předepsanou:

Pozice v prostoru. Každý objekt má přiřazenu pozici v trojrozměrném prostoru scény. Pozice může být i null, což znamená, že objekt je již vytvořen, ale ještě není umístěn ve scéně.

Báze souřadnicového systému objektu. Báze jsou tři ortonormální vektory představované instancí třídy SpaceBasis. Více k této třídě viz. javadoc a zdrojový kód editoru.

Identifikátor entity představovaný třídou ElementID (viz. dále).

Jméno nebo-li řetězec, který identifikuje entitu v grafickém rozhraní editoru.

Výběr. Každá entita může být uživatelem označena a tak zahrnuta do výběru (viz. uživatelská dokumentace).

Aby uživatel měl možnost označovat (vybírat) entity, je třeba umět poznat, že se pod kurzorem myši zrovna nějaká entita nachází. Toto umožňuje třída ElementID, která je nástrojem, jak entitám nebo jejich částem přiřazovat unikátní identifikátory. Každý takový identifikátor je ve skutečnosti 24-bitové číslo. Právě tolik bitů

má proto, že jde o barvu v RGB prostoru v klasickém kódování osm bitů na barevnou složku. Díky tomu, že identifikátor entity je vlastně barva, může zobrazovací vrstva při každém pohybu myší nejprve vykreslit scénu ve speciálním režimu s černou barvou pozadí, kde se každá entita (nebo její část) vykreslí barvou svého identifikátoru. Pak už je jednoduché zjistit barvu pixelu, který se zrovna nachází pod kurzorem a tak určit, nad kterou entitou či její částí právě uživatel pohybuje myší.

Jak už jsme poznali, identifikátory nemusí mít jen entity ale i jejich části jako například táhla na koncích úsečky úsečkového generátoru. Ty od sebe také potřebujeme odlišit, abychom mohli s generátorem manipulovat. Částem scény, jenž mohou mít své vlastní ID, budeme říkat *elementy* (element).

Třída BlobsEdDocument

BlobsEdDocument, odvozená od ReversibleDocument, je třídou dokumentu pro BlobsEd. Obsahuje a spravuje všechny entity ve scéně. Seznam entit je implementován jako mapa, kde klíče jsou identifikátory a hodnoty jsou entity. Každá entita dostane při vkládání do dokumentu přidělen jedinečný identifikátor. Tento identifikátor má entita výhradně na dobu, kdy je součástí dokumentu. Jakmile je z dokumentu odstraněna, její identifikátor je opět uvolněn k použití dalším entitám a elementům. Za přidělování a unikátnost identifikátorů zodpovídá instance třídy IDManager. Dokument tuto třídu využívá ke generování identifikátorů. Může ji ale využít jakýkoliv element, který si potřebuje rezervovat unikátní ID.

Dále si také dokument udržuje seznam vybraných entit a pamatuje si, který blob je zrovna v sub-object módu. O sub-object módu napíšeme více později, až budeme rozebírat fungování blobů.

Implementace generátorů

BlobsEd nabízí dva druhy generátorů a to bodový (třída PointGenerator) a úsečkový (třída LineGenerator). Oba dva jsou potomky třídy Generator, která implementuje vlastnosti společné pro všechny generátory. Jak už jsme popsali v části o teorii implicitních ploch, každý generátor má dva hlavní parametry – poloměr a váhu. Těmto budeme říkat *obecné parametry* (general parameters). Dál ke každému generátoru chceme znát jeho osově zarovnaný ohraničující kvádr, který obsahuje celou jeho obálku. To nám umožní při polygonizaci spočítat ohraničující kvádr pro celý blob, který poté procházíme algoritmem marching cubes.

Každý potomek abstraktní třídy Generator musí implementovat metodu touchVector, která počítá dotykový vektor ze zadaného bodu do generátoru. Tato metoda slouží jak k výpočtu vzdálenosti bodu od generátoru, tak k určení normály k jeho ploše v daném bodě. Schopnost počítat normály k povrchu přímo z funkce definující izoplochu blobu nám umožní zobrazovat velmi dobré stínování povrchu při osvětlování scény. Další abstraktní metoda toBlobSpace vytváří **kopii** generátoru, jejíž parametry jsou relativní vzhledem k bázi blobu, ke kterému generátor přísluší. Obě metody jsou důležité pro implementaci polygonizace, k jejímuž výkladu se dostaneme později.

Implementace bodového generátoru je nejjednodušší. Ten totiž nemá žádné jiné parametry než obecné. Také výpočet dotykového vektoru není těžký. Hledáme-li dotykový vektor z bodu P do bodu G, jenž je středem bodového generátoru, dotykový vektor $\vec{v}_p$ spočteme snadno podle vztahu

$$\vec{v}_p = G - P.$$

Co se týče úsečkového generátoru, ten je definován pomocí dvou bodů v prostoru. Jeden je samotná pozice této entity poděděná od třídy AbstractEntity a druhý bod úsečky je jediným parametrem, který má úsečkový generátor navíc oproti třídě Generator. Tomuto parametru budeme říkat jednoduše *druhý bod* (second point). Chceme-li spočítat dotykový vektor z bodu P do úsečky definované body G_1, G_2, musíme rozlišit tři případy (viz. obrázek 3.4). Jestliže je bod P uvnitř pásu vymezeném dvěma kolmicemi k úsečce generátoru, které procházejí jejími krajními body, je dotykový vektor vektorem kolmým k úsečce G_1, G_2 (případ P_2 na obrázku 3.4). Označíme-li $\vec{q} = G_2 - G_1$, potom dotykový vektor spočteme podle vztahu (dle [3, str. 561])

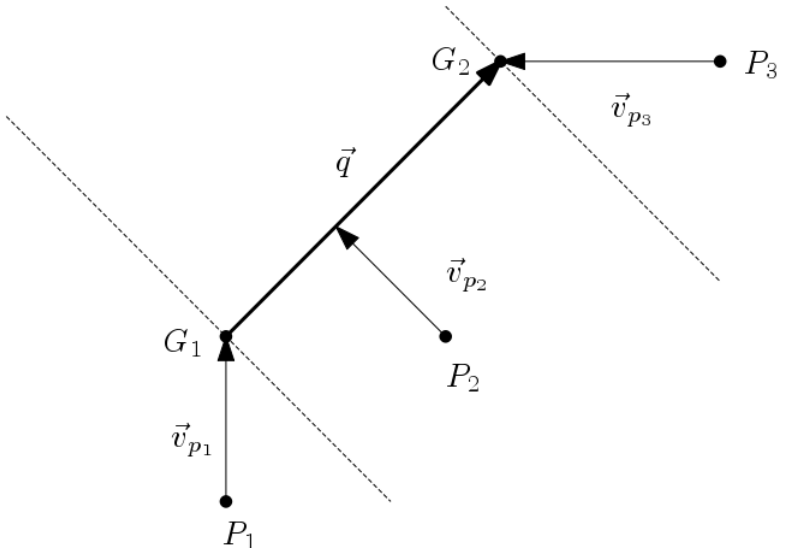
$$\vec{v}_p = -(P - G_1) + \frac{(P - G_1)\vec{q}}{\vec{q} \cdot \vec{q}}\vec{q}$$

Pokud bod P neleží uvnitř výše popsaného pásu, dotykový vektor ke generátoru je potom dotykovým vektorem k bližšímu z bodů G_1, G_2 (viz. případy P_1, P_3 na obrázku 3.4). Jestliže navíc sestrojíme dva vektory

$$\vec{u}_1 = G_1 - P \quad \vec{u}_2 = G_2 - P$$

a podíváme se na znaménka skalárních součinů $\vec{u}_1 \cdot \vec{q}$ a $\vec{u}_2 \cdot \vec{q}$, můžeme podle nich velice snadno rozhodnout, jakou pozici bod P zaujímá vzhledem k úsečce generátoru. Jsou-li totiž oba součiny kladné, jde o případ P_1, pokud je první záporný a druhý kladný, je to případ P_2 a jestliže jsou oba dva záporné, nastal případ P_3.

Všimněte si, že generátory samy nijak nepředepisují, jaká potenciálová funkce se má na vygenerování jejich skalárního pole použít. Je tedy možné plochy blobů



Obrázek 3.4: Tři možné případy polohy bodu vůči úsečkovému generátoru.

definovat pomocí různých potenciálových funkcí bez toho, abychom měnili přímo parametry konkrétních generátorů.

Implementace blobu

V úvodu k implicitním plochám jsme blobem rozuměli plochu generovanou množinou generátorů. V implementaci editoru entita zvaná blob obsahuje jen množinu generátorů a sadu parametrů ovlivňujících tvar plochy, kterou tyto generátory vytvoří. Třída Blob si tedy pamatuje hladinu skalárního pole, barvu blobu a *přesnost* (precision) zobrazení. Přesnost je přirozené číslo udávající, na kolik krychlí se má při polygonizaci rozdělit nejkratší hrana ohraničujícího kvádru. Samotná třída nemá žádný přístup k výsledné trojúhelníkové síti vzniklé polygonizací blobu, který definuje. Tato síť je totiž interpretací jejích parametrů a patří logicky do zobrazovací, nikoliv do datové vrstvy. Z toho důvodu také ani třída Blob nedefinuje žádnou konkrétní potenciálovou funkci pro své generátory.

Každý blob se může nacházet ve speciálním editačním módu zvaném *sub-object*. Tento mód umožňuje uživateli manipulovat s jednotlivými generátory blobu, posunovat je, měnit jejich parametry, odstranit je apod. To v normálním módu možné není, neboť blob se chová jako jeden celistvý objekt. V tomto módu může být vždy nejvýše jeden blob. S ostatními entitami ve scéně v takovém případě nelze manipulovat, dokud se tento blob zase nevrátí do normálního módu. Sub-object tedy není

výhradně stav blobu, ale spíše stav celé scény. Proto je také informace o tom, zda je právě scéna v tomto módu, součástí třídy dokumentu.

Změny v dokumentu

Jak už jsme výše zmínili, každá změna v datech dokumentu má svoji vlastní třídu. Nikdo jiný než instance těchto tříd tedy nevolá přímo metody dokumentu, které manipulují s jeho daty. Následuje velmi stručný popis všech tříd, které provádějí tyto změny. Všechny nevratné změny začínají písmenem C, vratné změny písmeny RC.

CGeneratorParams
: Nevratná změna obecných parametrů generátoru. Nevratná proto, že uživatel může provést velké množství malých změn, které chce poté vrátit zpět všechny najednou. To se děje například ve chvíli, kdy uživatel mění polohu objektu tažením myší. Tyto malé kroky se provádějí pomocí této třídy. Po jejich skončení se dokumentu předá změna RCGeneratorParams. Ta při svém provedení nedělá nic. Při braní zpět nebo opakovaném provedení však provede jednu velkou změnu, která je ekvivalentní posloupnosti malých nevratných změn. Tomuto mechanizmu budeme říkat *kroková změna* (stepping change).

  Tato změna se ve skutečnosti přímo nepoužívá, neboť samotné obecné parametry se v editoru krokovým způsobem měnit nedají. Slouží jako bázová třída pro generátory (jako je například úsečkový generátor, viz. třída CLineGenParams), které krokovou změnu některých svých dalších parametrů podporují. Jelikož například bodový generátor žádné jiné než obecné parametry nemá, krokové změny se u něj neuplatňují a neexistuje tedy žádná třída CPointGenParams.

CLineGenParams
: Nevratná změna pro krokovou změnu parametrů úsečkového generátoru. Krokově se mění pouze parametr udávající polohu druhého bodu úsečky, kterým může uživatel interaktivně posunovat v pohledech na scénu.

CMoveEntities
: Nevratná změna polohy pro krokové posunování entitami ve scéně.

CSelectEntities
: Změna výběru. Výběr je vždy nevratný.

CSubObject
: Přepnutí blobu do sub-object mód. Nevratná, neboť sub-object mód je pouze

jiným stavem dokumentu, který sám nemění parametry žádné entity ve scéně. Jen zpřístupňuje uživateli možnost měnit parametry konkrétního blobu, což už mohou být změny vratné. O sub-object módu napíšeme více v části o implementaci blobů.

RCAddNewEntity
: Přidání nové entity.

RCAttachToBlob
: Připojení množiny generátorů k blobu.

RCBlobColor
: Změna barvy blobu.

RCBlobParams
: Změna parametrů blobu.

RCDeleteEntities
: Odstranění entit.

RCEntityName
: Změna jména entity.

RCGeneratorParams
: Vratná změna pro krokovou změnu parametrů generátoru.

RCPointGenParams
: Změna parametrů bodového generátoru. Jelikož bodový generátor nemá jiné než obecné parametry, jde pouze o prázdné rozšíření třídy RCGeneratorParams.

RCLineGenParams
: Vratná změna pro krokovou změnu parametrů úsečkového generátoru.

RCMoveEntities
: Vratná změna pro krokové posunování entit ve scéně.

3.3 Zobrazovací vrstva

Zobrazovací vrstva slouží k interpretaci datové vrstvy a k její prezentaci uživateli. Jelikož BlobsEd je 3D editor, používá k zobrazování scény OpenGL, respektive jeho nadstavbu pro jazyk Java jménem Java OpenGL Library, krátce JOGL. Vyloženě technickými detaily souvisejícími se způsobem využití této knihovny v projektu se

zde nebudeme zabývat. Práce s JOGL zde příliš nevybočuje z klasického způsobu nakládání s touto knihovnou.

Prostředí scény

BlobsEd obsahuje čtyři pohledy na scénu, tři dvojrozměrné a jeden trojrozměrný. Každý může být nastaven na jiný způsob zobrazování – například drátěné modely versus stínované plochy apod. Souboru nastavení, který ovlivňuje způsob zobrazování pohledu, říkáme *prostředí* (environment). Nastavení prostředí určitého pohledu shrnuje třída ViewEnvironment a od ní odvozené třídy View2DEnvironment a View3DEnvironment.

Prostředí každého pohledu umožňuje nastavit typ *stínování* (shading), který ovlivňuje zobrazování izoploch blobů. Jsou to čtyři běžné způsoby zobrazování trojúhelníkových sítí – *drátěné* (wireframe), *ploché* (flat), *hladké* (smooth) a *hladké + odlesky* (smooth + specular). Kromě stínování má každý pohled tři *módy zobrazení* (rendering mode), což jsou *výběrový* (selection), *běžný* (regular) a *sprite* (sprite).

Ve výběrovém módu se scéna vykreslí s černým pozadím, na kterém se vyrenderují všechny entity barvou svého identifikátoru. To umožňuje rozpoznávat, nad jakou entitou se právě nachází kurzor myši. Běžný mód je určen, jak název napovídá, pro běžné zobrazení scény. Jakmile je scéna zobrazena v běžném módu, prostředí se ještě nastaví do sprite módu, ve kterém se vykreslují takzvané *táhla* (pull) generátorů. Jsou to malé lesklé kuličky, které v editoru označují například střed bodového generátoru nebo koncové body úsečky úsečkového generátoru. Tyto prvky slouží k manipulaci s tvarem a polohou generátoru a jsou zobrazeny jako dvojrozměrné prvky překrývající vždy všechny objekty ve scéně. Také si zachovávají konstantní velikost, ať už se pohled jakkoliv přiblíží či oddálí.

Sprite mód funguje tak, že se v prostředí pohledu nastaví speciální dvojrozměrný souřadnicový systém, kde každý pixel odpovídá přesně jedné jednotce souřadnicového systému OpenGL. Tak je možné vždy vykreslovat textury a jiné geometrické tvary v přesné velikosti v pixelech, bez ohledu na nastavení souřadnic v běžném módu. Třídy View2DEnvironment a View3DEnvironment poskytují funkce, které převádějí souřadnice bodů ze souřadnicového systému v běžném módu do systému ve sprite módu.

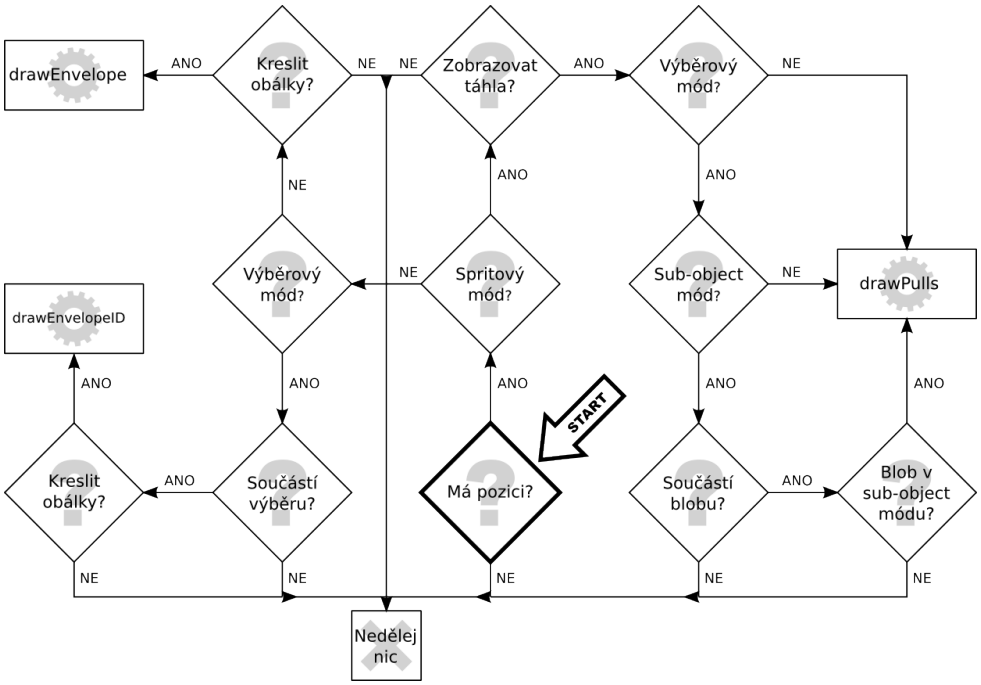
Zobrazovače

Všimněte si, že rozhraní Entity nepředepisuje žádnou metodu, která by sloužila k zobrazení dané entity. Je to proto, že architektura editoru BlobsEd se snaží striktně oddělit data od jejich interpretace. K zobrazování entit slouží speciální třídy zvané *zobrazovače* (renderer).

Pokud si na obrázku 3.3 doplníte za všechny názvy příponu Renderer, dostanete přesně hierarchii zobrazovačů pro všechny entity ve scéně. Zobrazovače fungují tak, že kdykoliv se vytvoří nová entita, zobrazovací vrstva vytvoří novou instanci příslušného zobrazovače, který se poté stará o renderování této entity. Každá instance nějaké entity tak má k sobě instanci zobrazovače. Seznam zobrazovačů lze se seznamem entit ve scéně snadno synchronizovat díky tomu, že dokument informuje o každém vložení či odstranění entity ze scény. Tento seznam všech zobrazovačů spravuje třída Scene. Kdykoliv tedy chceme překreslit scénu, stačí projít tento seznam a na všech zobrazovačích zavolat metodu draw. Později, až budeme hovořit o vkládání nových entit do scény uvidíme, že ve scéně jsou nejenom entity obsažené v dokumentu, ale také ty, které uživatel teprve do dokumentu vložit zamýšlí. Zobrazovače nám tak poskytují jednotný nástroj, jak zobrazit celou scénu, aniž bychom se museli na cokoliv ptát dokumentu.

Obecné schéma pro kreslení libovolného generátoru je již součástí základní třídy GeneratorRenderer, která implementuje metodu draw, jež volá abstraktní metody této třídy starající se o vykreslování generátoru v konkrétním stavu. Jelikož vzhled generátoru závisí z velké části na zvoleném nastavení prostředí scény a také na stavu, v jakém se zrovna generátor nachází – zda je zrovna vybrán, je-li součástí blobu apod., je průběh funkce draw poměrně komplikovaný. Obrázek 3.5 znázorňuje pomocí vývojového diagramu v jakých situacích jsou volány jaké abstraktní metody třídy GeneratorRenderer.

Později uvidíme, že BlobsEd se dá relativně jednoduchým způsobem rozšiřovat o nové generátory. Množina generátorů tedy může být různá při každém spuštění aplikace. Potřebujeme proto nástroj jak poznat, ke kterému generátoru patří který zobrazovač. K tomu složí takzvané *továrny* (factory) generátorů. Továrny generátorů jsou třídy implementující rozhraní GeneratorFactory. V základní nerozšířené verzi editoru jsou to podle očekávání PointGeneratorFactory a LineGeneratorFactory. Podíváte-li se na definici rozhraní GeneratorFactory uvidíte, že předepisuje metodu createRenderer, která ke zvolenému generátoru vytvoří správnou instanci jeho zobrazovače. Toto rozhraní má ještě několik dalších metod, které souvisí s kontrolní vrstvou, a rozebereme je tedy později.



Obrázek 3.5: Vývojový diagram znázorňující průběh funkce draw třídy GeneratorRenderer.

Problém jsme si však zatím příliš neulehčili, neboť nyní musíme zjistit, k jakému generátoru patří jaká továrna, abychom k němu mohli vyrobit zobrazovač. Tento problém řeší třída GeneratorExtension. Tato třída je jádrem architektury, která umožňuje přidávání nových generátorů bez nutnosti zkompilovat znovu celý projekt. Podrobně se jí budeme věnovat v kapitole 4. Nyní pouze řekneme, že má k dispozici seznam všech továren generátorů, a jelikož každá továrna ví, ke kterému generátoru patří, stačí při hledání továrny k danému generátoru tento seznam projít.

Zobrazovač blobu BlobRenderer navíc obsahuje odkaz na celou trojúhelníkovou síť blobu reprezentovanou třídou Mesh a také metodu updateSurface, která slouží ke spuštění algoritmu polygonizace.

Scéna

Scéna je reprezentována instancí třídy Scene. Obsahuje odkaz na dokument, seznam všech zobrazovačů a čtyři prostředí pro všechny čtyři pohledy. Celý tento balíček vlastností je potom možné ukládat do souboru a nahrávat zpět do editoru. Uloží se tedy nejen data dokumentu, ale i nastavení všech pohledů a ke všem entitám i jejich zobrazovače, takže se například uloží i trojúhelníková síť každého blobu.

Třída GLView

GLView je základní třídou pro takzvané *pohledy* (view). Pohledy jsou čtyři velká okna, která zabírají největší část grafického rozhraní editoru a zprostředkovávají uživateli obraz scény. BlobsEd zobrazuje scénu pomocí jednoho perspektivního 3D pohledu (komponenta View3D) a tří dvourozměrných pohledů (komponenta View2D), používajících pravoúhlé rovnoběžné promítání.

Třída GLView implementuje základní model vykreslování scény. K tomu používá komponentu GLCanvas a implementuje rozhraní GLEventListener (obě z knihovny JOGL). Každý pohled se standardně překresluje při těchto událostech myši – stisk a uvolnění tlačítka, pohyb kurzoru, vstup a opuštění komponenty a pohyb kolečka. Třída GLView předepisuje šest abstraktních metod:

- cursorReleased
- cursorPressed
- cursorMoved
- cursorEntered
- cursorExited
- cursorWheelMoved

Jejich implementace v poděděné třídě mají reagovat na příslušné události. Jakmile komponenta GLView zachytí událost myši, zapamatuje si ji a naplánuje překreslení okna. Jakmile se má okno překreslit, knihovna JOGL volá metodu display. Ta nejdříve zavolá příslušnou abstraktní obsluhu uložené události a poté volá abstraktní metodu viewDisplay, jejíž implementace se stará o vykreslování. Poděděná třída je tedy vždy nejdříve informována o události myši ještě před tím, než se pohled překreslí. Má tedy možnost nějakým způsobem zareagovat na tuto událost, popřípadě následné překreslení zakázat. Takto je překreslení zakázáno po událostech opuštění a vstupu kurzoru do komponenty, neboť je v tomto případě zbytečné.

Takto komplikovaný přístup, kdy se na každou událost myši reaguje jaksi nepřímo až uvnitř vykreslovací metody, a nikoliv rovnou v metodě příslušného posluchače, je dán tím, že pohledy mohou v reakci na nějakou událost chtít také něco vykreslit (například výběrový mód scény jak uvidíme dále). To vyžaduje, aby měli přístup k OpenGL kontextu, ke kterému mají přístup pouze metody rozhraní GLEventListener. Mezi nimi je právě metoda display, kterou implementuje třída GLView. Více k tomuto viz. dokumentace ke knihovně JOGL.

Třídy pohledů

Třídy View2D a View3D odvozené od GLView pracují velmi podobně. Využívají například mechanizmu zpracování událostí v třídě GLView k převodu událostí myši na události 3D kurzoru. Tento kurzor, jak název napovídá, se pohybuje v trojrozměrném prostředí scény. Informace o poloze a ostatních vlastnostech kurzoru ve scéně nese třída CursorEvent. Ta spolupracuje s rozhraním CursorEventListener, které slouží k zachytávání událostí kurzoru. Kdokoliv si tedy může u tříd View2D či View3D zaregistrovat tento posluchač a reagovat na pohyb kurzoru v 3D prostoru namísto reagování na události 2D ukazatele myši.

View2D a View3D vždy nejdříve dostanou od základní třídy GLView zprávu o události myši. V reakci na ni nejprve vykreslí scénu do zadního bufferu ve výběrovém módu. Tím zjistí identifikátor elementu, nad kterým se právě nachází ukazatel myši. Vytvořením nové instance třídy CursorEvent je pak převeden 2D ukazatel na 3D kurzor. Tato událost mimo jiné nese také informaci o identifikátoru elementu pod kurzorem. Jakmile jsou o této události spraveni všichni posluchači, je scéna vykreslena v normálním módu, čímž se překreslí výběrový mód v zadním bufferu, takže uživatel jeho vykreslení nevidí.

Samotný proces vykreslování už je velice jednoduchý. Jelikož všechny pohledy mají referenci na aktuální scénu, stačí, když po vykreslení mřížky a jiných pomocných objektů, projdou všechny zobrazovače ve scéně a zavolají jejich vykreslovací metodu.

Polygonizace blobů

Spravovat trojúhelníkové sítě tvořící izoplochy blobů je úkolem třídy BlobRenderer. Pro vytvoření sítě ze skalárního pole slouží třída MarchingCubesPolygoniser. Ta rovnou počítá s tím, že tato úloha může trvat dlouho, čili kromě metody createSurface, jež vytváří výslednou síť, je zde ještě metoda cancelTask, která slouží k přerušení započatého výpočtu. Lze navíc zaregistrovat PropertyChangeListener, který je průběžně informován o průběhu výpočtu. Toho využívá dialog, který v editoru zobrazuje průběh polygonizace.

Ještě než se pustíme do popisu implementace algoritmu marching cubes, musíme vyložit, jakým způsobem vůbec budeme počítat hodnoty skalárního pole okolo generátorů. Jak jsme viděli dříve, samotná třída Generator v tomto směru neumí nic jiného, než počítat dotykové vektory. My ovšem potřebujeme jednak vypočíst

hodnotu skalárního pole kolem množiny generátorů tvořících blob, ale také gradient tohoto pole, abychom dokázali určit normálové vektory k výsledné izoploše. K tomu ale nejdříve musíme zvolit konkrétní potenciálovou funkci. Dejme tomu, že bychom chtěl použít funkci g_r, kterou jsme definovali v úvodu do implicitních ploch takto

$$g_r(d) = \begin{cases} \left(1 - \frac{d^2}{r^2}\right)^2 & d \in \langle 0, r) \\ 0 & d \geq r. \end{cases}$$

Výsledné skalárné pole jsme potom definovali pomocí hodnot funkce

$$f(x, y, z) = \sum_{k=1}^{n} w_k g_{r_k}(d_k(x, y, z)),$$

kde d_k je délka dotykového vektoru z bodu (x, y, z) do generátoru G_k. Pro gradient této funkce platí

$$\nabla f(x, y, z) = \nabla \sum_{k=1}^{n} w_k g_{r_k}(d_k(x, y, z)) = \sum_{k=1}^{n} w_k \nabla g_{r_k}(d_k(x, y, z)),$$

kde

$$w_k \nabla g_{r_k}(d_k(x, y, z)) = w_k g'_{r_k}(d_k(x, y, z)) \nabla d_k(x, y, z).$$

Všimněme si, že jelikož funkce d_k udávají vzdálenost od generátoru G_k (definovanou pomocí dotykového vektoru), jejich gradient má vždy směr dotykového vektoru a velikost jedna (představte si graf funkce udávající vzdálenost od bodu). Pro bod $P = (x, y, z)$ tedy platí

$$\nabla d_k(x, y, z) = \frac{1}{d_k(x, y, z)} \vec{v}_p,$$

z čehož po dosazení ihned dostáváme

$$\nabla f(x, y, z) = \vec{v}_p \sum_{k=1}^{n} w_k \frac{g'_{r_k}(d_k(x, y, z))}{d_k(x, y, z)}. \tag{3.1}$$

Předpis funkce g_r je chytře zvolen tak, že při výpočtu hodnot skalárního pole nemusíme počítat časově náročnou druhou odmocninu. Místo abychom zbytečně počítali délku dotykového vektoru d_k, když ji stejně po dosazení do g_r budeme mocnit na druhou, spočítáme daleko jednodušeji rovnou d_k^2. Podobné zjednodušení si můžeme dovolit také při výpočtu gradientu. Upravíme-li totiž výraz vyskytující se ve vztahu (3.1)

$$\frac{g'_{r_k}(d)}{d} = \frac{4d(d^2 - r_k^2)}{d r_k^4} = \frac{4d^2}{r_k^4} - \frac{4}{r_k^2},$$

dostaneme opět výraz, který obsahuje pouze druhé mocniny vzdálenosti. Při implementaci musíme ovšem dát pozor na případ $d = 0$, neboť v takové situaci přirozeně chceme, aby nám gradient vyšel nulový.

Máme tedy vcelku efektivní způsob, jak počítat normály k izoploše. Stačí obrátit vektor gradientu na opačnou stranu a zkrátit ho na jednotkovou délku. To je také jediné místo ve výpočtu, kdy je třeba použít druhou odmocninu.

Různé potenciálové funkce pro generátory jsou v editoru reprezentovány třídami implementujícími rozhraní GeneratorEvaluator, které má jednu metodu pro výpočet hodnoty skalárního pole a jednu pro výpočet gradientu. Naši potenciálovou funkci g_r implementuje třída QuarticEvaluator. Projekt z testovacích důvodů obsahuje ještě třídu SexticEvaluator, která implementuje výpočetně náročnější potenciálovou funkci (opět z [3, str. 225])

$$q_r(d) = -\frac{4}{9}\frac{d^6}{r^6} + \frac{17}{9}\frac{d^4}{r^4} - \frac{22}{9}\frac{d^2}{r^2} + 1,$$

která má velmi podobný průběh jako funkce g_r, ale je například symetrická pro hodnotu $0{,}5r$. Jelikož se tvary výsledných blobů na pohled nijak neliší od těch vygenerovaných funkcí g_r, je editor zkompilovaný s polygonizací pomocí této funkce, neboť je výpočetně méně náročná než q_r.

Implementace algoritmu marching cubes

Algoritmus marching cubes implementuje třída MarchingCubesPolygoniser. Nejprve je třeba této třídě předat instanci třídy implementující rozhraní EvaluatorFactory, která pro každý generátor vrátí správný evaluátor. Poté se všechny generátory v blobu pomocí metody Generator.toBlobSpace okopírují a převedou do jeho souřadného systému. Kopírování se děje proto, abychom neměnili pozici generátorů přímo ve scéně, což bychom museli dělat pomocí změn v dokumentu. Následně se vypočte osově zarovnaný ohraničující kvádr obsahující obálku blobu a může začít polygonizace pochodujícími kostkami.

Nejtěžší částí je organizace procházení 3D mřížky takovým způsobem, že se hodnoty skalárního pole počítají přesně tolikrát, kolikrát je to nezbytně potřeba. Skalární pole se bezpodmínečně musí vyhodnotit ve všech vrcholech každé krychle mřížky. Kdybychom k implementaci přistupovali naivně, některé hodnoty bychom počítali až osmkrát, neboť každý vrchol může být společný až osmi krychlím. Také bychom museli trojúhelníky tvořící výsledný povrch považovat za disjunktní trojice vrcholů, namísto aby šlo jen o trojice indexů do nějakého seznamu všech vrcholů.

Taková reprezentace nejen plýtvá pamětí, ale zároveň způsobuje problémy, pokud bychom následně síť chtěli efektivně exportovat do nějakého jiného 3D formátu.

Hlavní pozorování je, že krychle se skládají z hran, které mezi sebou sdílejí. Hrany naopak mezi sebou sdílejí jednotlivé vrcholy. Jednu krychli mřížky představuje třída Cube. Ta obsahuje ukazatele na svých dvanáct hran (třída Edge). Naopak každá hrana obsahuje ukazatele na své dva vrcholy, což jsou v podstatě reálná čísla udávající hodnoty skalárního pole. Navíc každá hrana, pokud na ní leží vrchol nějakého trojúhelníka, zná jeho index v seznamu všech vrcholů izoplochy. Díky tomu, že jsou tyto objekty takto provázány pomocí odkazů, může každá hrana i každý vrchol krychle i trojúhelníka existovat v paměti pouze jednou. Hlavním problémem ovšem je, jak takovouto síť navzájem provázaných krychlí co nejelegantněji vytvořit.

Abychom popsali konstrukci této 3D mřížky, vypůjčíme si malinko z divadelní terminologie. Představme si, že krychle jsou diváci sedící v divadle na *sedadlech* (seat), která jsou uspořádána do *řad* (line) probíhajících ve směru osy x. Řady stojí za sebou ve směru osy z a tvoří takzvané *podlaží* (floor). Podlaží jsou na sebe naskládána ve směru zbývající osy y. Pomyslné jeviště se tedy nachází ve směru klesajících hodnot osy z a polygonizace začíná v prvním podlaží prvním sedadlem vlevo v první řadě a pokračuje směrem od jeviště ve směru osy z. Končí se v nejvyšším patře sedadlem v poslední řade vpravo.

Co se týče sdílení hran se svými sousedy, existuje dohromady osm různých druhů pozic, na kterých se krychle může nacházet.

První podlaží, první řada, první sedadlo neodkazuje na nikoho. Všechny hrany a vrcholy jsou alokovány nově.

První podlaží, první řada, nikoli první sedadlo odkazuje na svého souseda vlevo.

První podlaží, nikoli první řada, první sedadlo odkazuje na krychli sedící před ní.

První podlaží, nikoli první řada, nikoli první sedadlo odkazuje na krychli vpředu i vlevo.

Nikoli první podlaží, první řada, první sedadlo odkazuje na krychli pod ní.

Nikoli první podlaží, první řada, nikoliv první sedadlo odkazuje na svého souseda vlevo a pod sebou.

Nikoli první podlaží, nikoliv první řada, první sedadlo odkazuje na krychli sedící před ní a pod ní.

Nikoli první podlaží, nikoliv první řada, nikoliv první sedadlo odkazuje na krychli vpředu, vlevo i pod.

O vytvoření odkazů na správné hrany či alokování nových hran se stará konstruktor třídy Cube. Ten vždy obdrží odkaz na krychli vlevo, před i pod (null pokud některá z nich neexistuje) a nasdílí ty správné hrany jednoduchým rozborem podle výše popsaných osmi případů. Tento konstruktor navíc rovnou dopočítá hodnoty skalárního pole v nově alokovaných vrcholech. Vždy po vytvoření nové krychle tak můžeme rovnou přidat nové trojúhelníky. Díky tomuto postupu navíc nemusíme v paměti udržovat celou síť. V každém okamžiku nám stačí znát právě rozpracované patro a patro pod ním.

Samotné vytváření trojúhelníků už je jednoduché. Každá krychle dostane podle hodnot ve svých vrcholech index do tabulky, která obsahuje pro každý z 256 možných případů záznam, udávající trojice indexů hran, na kterých je třeba interpolací vytvořit vrcholy nových trojúhelníků. Taková hrana už buď má index vrcholu vytvořeného v některé z předchozích fází výpočtu, nebo se interpoluje nový vrchol a vloží se do seznamu vrcholů vytvářené izoplochy. Každý trojúhelník tak obsahuje opět pouze indexy do seznamu vrcholů izoplochy.

Třída MainView

MainView je třída, která již ze zobrazovací vrstvy přesahuje do vrstvy kontrolní. Sdružuje okna čtyř pohledů v editoru do jedné komponenty a tak vlastně vytváří celkový pohled na scénu a je jakýmsi uzávěrem zobrazovací vrstvy. Navíc umožňuje uživateli interaktivním způsobem měnit scénu pomocí takzvaných návrhářů, které podrobně rozebereme v následujícím oddíle.

3.4 Kontrolní vrstva

Kontrolní vrstva slouží, jak její název napovídá, k manipulování se scénou a jejím prostředím. To zahrnuje správu entit ve scéně a jejich parametrů, ale také změnu nastavení pohledů či ukládání scény do souboru nebo její export do nějakého externího datového formátu.

Kontrolní vrstva umožňuje manipulovat se scénou jednak prostřednictvím různých nabídek, panelů nástrojů a dialogů, ale také pomocí hlavního pohledu na scénu, kde může uživatel přímo manipulovat s entitami nebo jejich skupinami. Hlavní pohled editoru je, jak už jsme zmínili, reprezentován třídou MainView, která tedy musí kromě schopnosti zobrazovat scénu poskytovat také mechanizmus, jak s ní manipulovat. Změna nastavení pohledů (tedy například jejich natočení, posunutí, způsob vykreslování) se provádí přes vyskakovací menu. Ovšem věci jako vybírání a posouvání entit, seskupování generátorů do blobů aj. potřebují speciální mechanizmus, kterému budeme říkat *správa scény* (scene management).

Správa scény

Základ pro správu scény vytváří rozhraní SceneManagement. Každá instance třídy implementující toto rozhraní se takzvaně *zapojí* (connect) do instance třídy MainView, přičemž v každém okamžiku může být zapojena pouze jedna správa. Ta potom dostává události myši v pohledu a může na ně specifickým způsobem reagovat. Správy scény se v podstatě starají o dvě věci. Jednak o manipulaci s celými entitami a jednak o jejich vytváření a měnění jejich parametrů.

Editor BlobsEd umožňuje tři druhy manipulace s celými entitami. Jsou to *výběr* (selection), *posun* (movement) a *připojování k blobu* (attach to blob). Tyto mechanizmy jsou implementovány po řadě třídami SelectionManagement, MovementManagement a AttachToBlobManagement. Všechny tři implementují rozhraní CursorEventListener, takže mohou adekvátním způsobem reagovat na pohyb kurzoru ve scéně. Například správa připojování k blobu takto umožňuje vybrat skupinu generátorů a potom je připojit k vybranému blobu.

Návrháři

Takzvaní *návrháři* (designer) jsou dalším typem správy scény. Umožňují vytvářet a měnit jednotlivé entity ve scéně. Hierarchie tříd návrhářů začíná rozhraním EntityDesigner a opět kopíruje hierarchii entit nebo zobrazovačů. Každý návrhář má dva úkoly. Za prvé umožnit uživateli vytvořit entitu a za druhé umožnit již vytvořenou entitu upravit. Chce-li například uživatel vložit do scény úsečkový generátor, zapojí se do hlavního pohledu instance návrháře LineGeneratorDesigner ve stavu pro vytváření nové entity. Tento návrhář potom zachytává z hlavního pohledu všechny události kurzoru a umožní uživateli nejprve vložit první a poté druhý bod úsečkového generátoru. Pokud naopak uživatel chce například posunout jeden z koncových bodů

již existujícího generátoru, vytvoří se pro tento generátor nový návrhář ve stavu pro úpravy a opět stejným způsobem dovolí uživateli hýbat s vybraným bodem.

Kontrolní panely a příkazy

Kromě spravování scény přímo skrz hlavní pohled může uživatel měnit parametry entit i pomocí takzvaných *kontrolních panelů* (control panel), které se zobrazují napravo od hlavního pohledu. Podle toho, jaké entity jsou zrovna vybrány, se zobrazují ty správné panely umožňující přesně zadat jejich vlastnosti.

Posledními částmi grafického rozhraní editoru jsou panel nástrojů a hlavní nabídka. Jelikož tyto dvě komponenty nabízejí v podstatě tytéž ovládací prvky, existují takzvané *příkazy* (command), což jsou instance tříd odvozených od BlobsEdCommand. Příkaz je objekt, jenž spojuje tlačítko na panelu nástrojů s položkou v hlavním menu stejně, jako to dělá rozhraní Action z jazyka Java, které je touto třídou implementováno. Jednotlivé příkazy tak mohou měnit svoje stavy (dostupný/nedostupný a jiné) například podle toho, jaké entity jsou zrovna ve scéně vybrány apod. Toto propojení skrz příkazy potom zajišťuje synchronizaci tlačítek na panelu nástrojů a položek v menu.

Serializace a export scény

BlobsEd nabízí jednak ukládání rozpracované scény do vlastního formátu založeného na XML, jednak export blobů do formátu Wavefront OBJ, který mohou dále importovat jiné grafické editory.

O ukládání a načítání scény se stará třída SceneXMLSerializer, která využívá standardního mechanizmu z jazyka Java, totiž tříd XMLEncoder/XMLDecoder, k serializaci objektů scény do XML. Do XML souboru se ukládá celý obsah aktuální instance třídy Scene, tedy jak dokument, tak prostředí scény, tak seznam všech zobrazovačů.

Export do OBJ formátu zajišťuje třída SceneWavefrontEncoder, která přímočarým způsobem převede sítě trojúhelníků jednotlivých blobů do textové reprezentace specifikované tímto formátem. Projdou se všechny zobrazovače ve scéně, které zobrazují bloby, a jejich sítě trojúhelníků se exportují podle specifikace formátu OBJ.

Kapitola 4

Rozšiřitelnost editoru

BlobsEd si neklade za cíl být editorem s velkým množstvím funkcí a možností. Spíše má býti takovému editoru základním kamenem. Z tohoto důvodu byl napsán s důrazem na modularitu, a to především v oblasti rozšiřování o nové druhy generátorů. Kód je navržen a strukturován tak, aby bylo třeba co nejmenšího úsilí k tomu, aby programátor mohl přidat nový druh generátoru. Aby se tento nový generátor dal v editoru použít, není dokonce třeba ani celý projekt znovu kompilovat. Soubory s kódem generátoru se připojí ve formě pluginu.

4.1 Přidávání nových generátorů

Přidávání generátorů do editoru probíhá tak, že programátor zkopíruje zkompilované .class soubory do příslušné složky. Editor tedy potřebuje mechanizmus rozpoznávající, které generátory jsou při jeho spuštění k dispozici. Toto zajišťuje třída GeneratorExtension, která umí v příslušné složce vyhledat všechny třídy, které implementují rozhraní GeneratorFactory. Každý generátor má takovouto továrnu, která slouží k vytváření instancí všech tříd, které zajišťují fungování generátoru, což jsou generátor samotný, jeho zobrazovač, návrhář, kontrolní panel a také příkaz umožňující vkládání generátoru do scény, jenž se zobrazuje na panelu nástrojů a v hlavní nabídce editoru.

Nyní popíšeme postup, jakým se do editoru přidává nový generátor. Krok za krokem vysvětlíme, jak by se naprogramoval například obdélníkový generátor, tedy takový, jehož základem je libovolně orientovaný obdélník v prostoru.

1. **Třída pro samotný generátor**
   Nejprve odvodíme novou třídu RectangleGenerator od třídy Generator. Ta bude obsahovat souřadnice čtyř vrcholů obdélníka v prostoru. Toto je dostatečná informace, jež jednoznačně definuje instanci obdélníkového generátoru. Jedinou náročnější prací v této části bude implementace metody touchVector, neboť musíme vymyslet, jak vypočítat dotykový vektor k obdélníku v prostoru. Jelikož tento návod má popisovat API editoru, nebudeme se touto část návrhu detailněji zabývat.

2. **BeanInfo pro generátor**
   Nesplňuje-li náš generátor specifikaci Java Beans, musíme doplnit ještě třídu RectangleGeneratorBeanInfo, která definuje způsob, jak se náš generátor bude ukládat do XML. Tento přístup volí například třída LineGenerator. Je tedy možno se inspirovat její implementací.

3. **Zobrazovač**
   Vytvoříme třídu RectangleGeneratorRenderer odvozenou od GeneratorRenderer, jenž bude zobrazovačem pro náš nový generátor. Implementujeme všechny abstraktní metody, které nám předepisuje GeneratorRenderer takovým způsobem, že umožníme vykreslit generátor, jeho obálku i ovládací táhla pro všechny čtyři body obdélníka. Každé toto táhlo potřebuje svoje vlastní ID, abychom ho ve scéně poznali ve chvíli, kdy budeme chtít tvar již vytvořeného generátoru dále upravovat. Budeme tedy chtít metody, pomocí nichž se tato čtyři ID budou dát nastavit. Ty potom využije návrhář, který tato ID zaregistruje a předá našemu zobrazovači ve chvíli, kdy bude připojen do hlavního pohledu. Inspiraci jak toto provést najdeme v implementaci zobrazovače pro úsečkový generátor.

4. **BeanInfo pro zobrazovač**
   Ještě musíme specifikovat třídu RectangleGeneratorRendererBeanInfo, která definuje způsob, jakým se bude zobrazovač našeho generátoru serializovat do XML. Zobrazovače totiž nemají bezparametrický konstruktor, musíme tedy specifikovat vlastního persistence delegáta. Více k tomuto viz. dokumentace k Java API a implementace ostatních generátorů v editoru.

5. **Návrhář**
   Návrhářem bude nová třída RectangleGeneratorDesigner odvozená od GeneratorDesigner. Ta bude mít dva konstruktory. Jeden bude bezparametrický pro návrhář ve vytvářecím módu, kdy budeme vytvářet nový generátor. Druhý bude brát jako parametr již existující obdélníkový generátor a bude sloužit k jeho úpravě. V metodách reagujících na události kurzoru ve scéně poté implementujeme samotné vytváření či modifikaci generátoru. Inspiraci opět

najdeme v návrháři pro úsečkový generátor. API editoru samozřejmě nepředepisuje, jaké mají mít návrháři konstruktory. Jejich instance jsou vytvářeny pomocí továrny implementující GeneratorFactory (viz. dále). Jde tedy pouze o doporučený postup, jak tuto třídu implementovat.

6. **Kontrolní panel**
   Třída RectangleGeneratorPanel odvozená od GeneratorPanel bude kontrolním panelem pro náš nový generátor. Třída GeneratorPanel poskytuje základní ovládací prvky pro nastavování parametrů společných všem generátorům, jako je poloměr nebo váha. Do rozvržení tohoto panelu můžeme v odvozené třídě přidat další panel, který umožní nastavovat parametry specifické pro náš obdélníkový generátor, jako je poloha jednotlivých bodů obdélníka apod. K nastavování polohy bodu v prostoru obsahuje BlobsEd třídu PositionPanel. Tu můžeme využít podobně, jako to dělá například třída LineGeneratorPanel.

7. **Definice příkazu**
   K fungování obdélníkového generátoru budeme ještě potřebovat třídu odvozenou od CommandDefinition, která definuje, jak bude vypadat příkaz nabídky a tlačítko panelu nástrojů, které vkládají náš generátor do scény. Tuto třídu vytvoříme anonymní, neboť je krátká a její instanci bude vracet pouze metoda továrny pro tento generátor.

8. **Změna v dokumentu**
   Aby uživatel mohl provádět změny parametrů generátoru a tyto změny potom také vracet zpět, potřebujeme ještě třídy CRectangleGenChange a RCRectangleGenChange. Máme vratnou i nevratnou změnu, neboť chceme implementovat mechanizmus krokové změny ve chvíli, kdy uživatel posunuje jednotlivými krajními body obdélníka. Naprogramovat tyto změny znamená opět pouze prohlédnout si, jak jsou implementovány pro úsečkový generátor a lehce je upravit pro náš případ.

9. **Továrna**
   Nakonec budeme potřebovat továrnu RectangleGeneratorFactory implementující rozhraní GeneratorFactory. Ta umožní spojit všechny třídy, které jsme dosud vytvořili, do jednoho celku. Opět stačí, abychom mírně upravili implementaci továrny například pro úsečkový generátor.

Výše naprogramované třídy už nyní stačí umístit do správných balíčků. Třída RectangleGeneratorFactory patří do složky

cz/matfyz/lishaak/blobsed/scene,

kde ji bude hledat třída GeneratorExtension. Následují třídy změn CRectangleGenChange a RCRectangleGenChange, které umístíme do složky

cz/matfyz/lishaak/blobsed/doc/changes

a pro všecny ostatní výše uvedené třídy vytvoříme speciální složku

cz/matfyz/lishaak/blobsed/scene/rectanglegenerator.

Jakmile jsou všechny .class soubory na svých místech, stačí spustit BlobsEd editor a nový generátor by se měl objevit v hlavní nabídce i na panelu nástrojů. Ve skutečnosti není tak důležité, ve kterých složkách se nové soubory nacházejí. Stačí když se budou nové třídy navzájem importovat ze správných balíčků. Jediná třída, která musí být správně umístěna je RectangleGeneratorFactory, jinak ji totiž GeneratorExtension nenajde.

Výše uvedený postup rozšiřování editoru na první pohled nevypadá jednoduše ani přímočaře, neboť je třeba naprogramovat více tříd, které spolu netriviálním způsobem komunikují. To je dáno především tím, že je nutné implementovat vytváření a navrhování generátoru uživatelem. Na druhou stranu, kód obsažený v nových třídách není většinou nijak dlouhý či komplikovaný a použité mechanizmy lze snadno odpozorovat z implementace bodového či úsečkového generátoru.

Kapitola 5

Závěr

V této práci jsme z teorie a praxe implicitních ploch stačili vyložit a implementovat pouze naprosté základy. Soustředili jsme se hlavně na modelovací techniky a pominuli tak jiné oblasti a témata, například různé techniky přímého zobrazování jako ray-casting a ray-tracing nebo jiné méně obvyklé přibližné metody. Nezabývali jsme se také teorií a implementací složitějších generátorů, které by dále rozšířili množství tvarů, které lze modelovat. Jistě by stálo za to pokusit se odstranit požadavek na konvexnost generátoru, který by umožnil mít jako generátor například Beziérovu křivku, což by opět zásadním způsobem rozšířilo možnosti editoru. Nejspíše však za cenu komplikovanějšího výpočtu normál k povrchu blobů nebo průsečíků s paprskem v případě přesných zobrazovacích metod.

Stejně tak editor BlobsEd byl od začátku koncipován jako jednoduchý modelovací program, který si neklade za cíl porazit již existující nástroje pracující s implicitními plochami. Po celou dobu návrhu a programování byla důležitým ukazatelem především rozšiřitelnost. Do budoucna by proto mohlo být zajímavé a přínosné editor doplnit a vylepšit například o následující schopnosti

- Techniky přímého zobrazování, například podpora ray-tracingu. Uživatel by jistě ocenil možnost jednou za čas si prohlédnout vymodelovaný objekt v přesném zobrazení i za cenu toho, že si na jeho vyrenderování bude muset chvíli počkat. Jelikož výpočet normál k povrchu blobů není těžký, jediné, čemu by se musela věnovat větší pozornost především z teoretického hlediska, je co nejrychlejší numerické počítání průsečíku paprsku s povrchem blobu. Po zvládnutí tohoto problému bychom navíc rázem získali další zobrazovací možnosti jako jsou přesné odlesky, zrcadlení, vrhání stínů, průhlednost apod.

- Předchozí bod implikuje nutnost rozšíření editoru a možnosti jako přidávání světel do scény a aplikace různých materiálů na vytvořené plochy a to včetně texturování.
- Lepší spolupráce s jinými programy, které též umí pracovat s implicitními plochami. Mezi ně patří například POV-Ray nebo grafický editor Blender.
- Rozšíření nástrojů editoru jednak o jinde běžné věci, jako je například práce se schránkou, na druhou stranu o více specifické nástroje, například pro práci se skupinami blobů, definování vzájemného působení těchto skupin, tvorba různých hierarchií objektů a podobně.
- Možnost tvorby animací.

Literatura

[1] Blinn J. (1982): A Generalization of Algebraic Surface Drawing. *ACM Transaction on Graphics*, 1:232.

[2] Lorensen W., Cline H. (1987): Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics (SIGGRAPH '87 Proceedings)*, 21(4):163-169.

[3] Žára J., Beneš B., Sochor J., Felkel P. (2004): Moderní počítačová grafika. Computer Press, Brno.

Příloha A

Obsah přiloženého CD

Obsah adresářů na přiloženém CD:

- bcprace
  Digitální podoba této práce ve formátu PDF.
- bin
  Editor BlobsEd zkompilovaný pro všechny podporované platformy. Více viz. uživatelský manuál v příloze B.
- javadoc
  Hypertextová javadoc dokumentace k editoru.
- source
  Zdrojové kódy editoru.

Příloha B

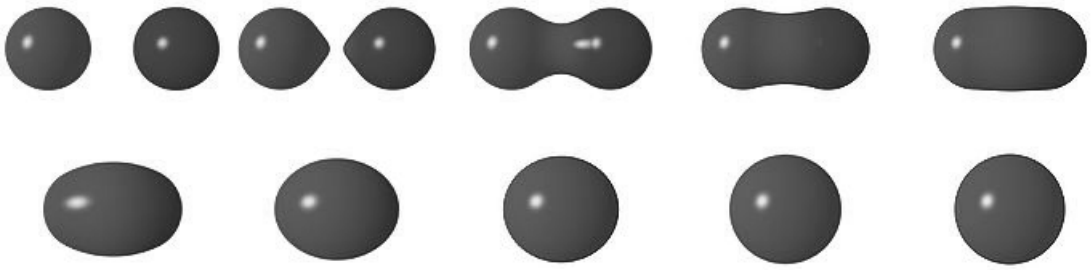
Uživatelský manuál

B.1 Modelování pomocí implicitních ploch

Modelování pomocí implicitních ploch je přístup značně odlišný od běžných postupů modelování pomocí polygonových sítí, parametrických ploch a dalších. Nedává uživateli tak přesnou kontrolu nad tvarem objektů, zato však pracuje s myšlenkou, že tyto objekty navzájem ovlivňují svůj tvar podle vzájemné polohy vůči sobě. Přibližujeme-li k sobě například dvě koule, čím je vzdálenost mezi nimi menší, tím se více deformují, až splynou v jeden celek (viz. obrázek B.1). Místo koulí můžeme samozřejmě použít i jiné tvary. Tyto tvary jsou definovány pomocí takzvaných *generátorů*. Každý generátor je v podstatě geometrický útvar jako například bod, úsečka, trojúhelník a podobně. Kolem každého takového generátoru se potom vytvoří plocha, jejíž tvar vznikne tak, jako bychom daný generátor nafoukli. Takže například bodový generátor vytvoří plochu tvaru koule a úsečkový jakousi kapsli. Ploše vygenerované nějakou skupinou generátorů budeme říkat *blob*. To, co tedy vidíme na obrázku B.1, je jeden blob, vytvořený pomocí dvojice bodových generátorů v různých fázích přiblížení. Editor BlobsEd poskytuje nástroje pro práci právě s těmito (a samozřejmě také daleko složitějšími) bloby.

B.2 Instalace a spuštění programu

BlobsEd editor lze používat v operačních systémech Windows i Linux, na obou buď v 32-bitové nebo 64-bitové variantě. Pro každou z platforem existuje samostatný archiv obsahující vše potřebné ke spuštění editoru:



Obrázek B.1: Ukázka vzájemné deformace dvou přibližujících se koulí.

- blobsed-linux32.zip
- blobsed-linux64.zip
- blobsed-win32.zip
- blobsed-win64.zip

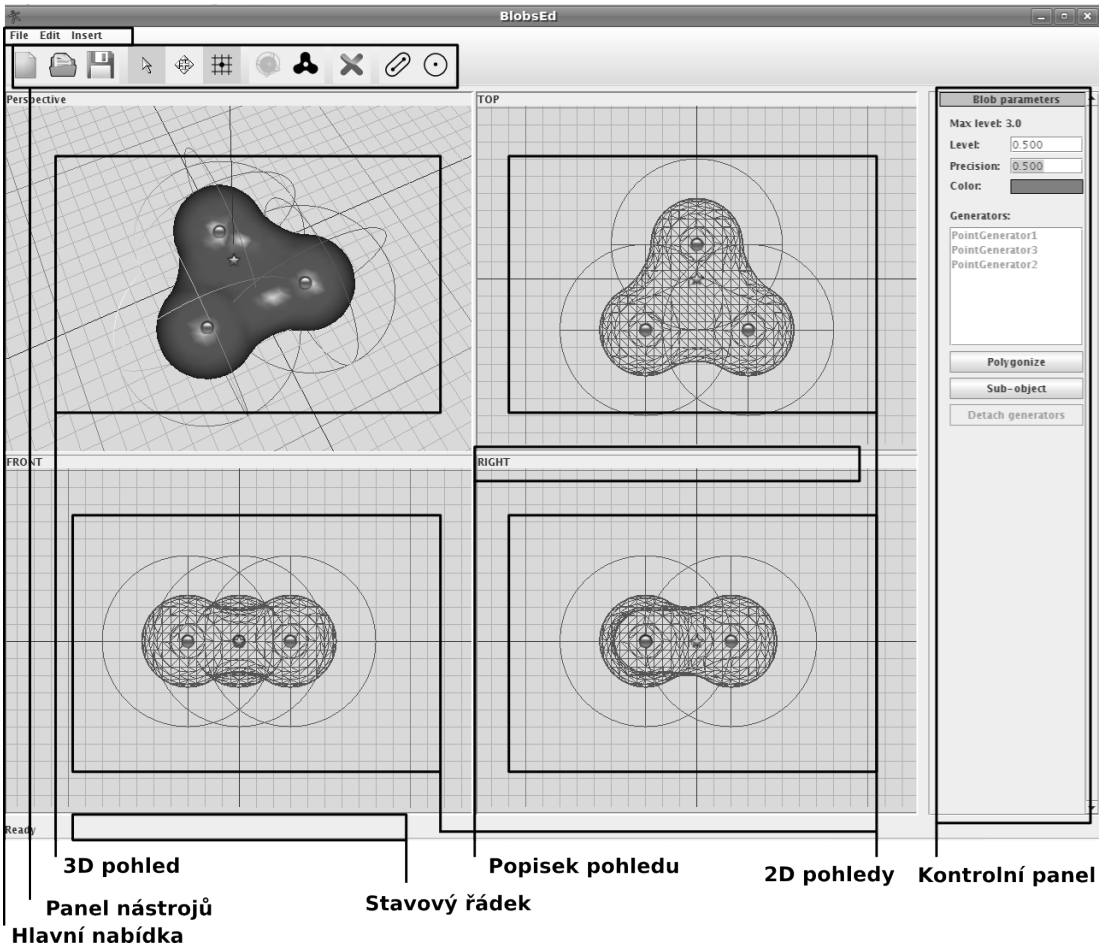
Abyste editor mohli spustit, je třeba mít nainstalován Java Runtime Environment (JRE) verze 1.6. Jakmile máte správný archiv, stačí jej rozbalit do libovolné složky na disku. Editor se potom spouští speciálním skriptem, který se ve Windows jmenuje blobsed.bat a v Linuxu blobsed.sh. Pro spuštění editoru pod Windows potom stačí jednoduše dvojkliknout na tento skript. V Linuxu se přesuňte do složky s rozbalenými soubory editoru a napište příkazy

```
chmod +x blobsed.sh
sh blobsed.sh
```

B.3 Grafické rozhraní editoru

Hlavní okno

Po spuštění programu se objeví hlavní okno editoru (viz. obrázek B.2). Největší část okna zabírají čtyři pohledy na modelované objekty. Pohled s nadpisem perspective zobrazuje scénu ve 3D perspektivě, ostatní tři používají kolmé rovnoběžné promítání. V horní části je klasická hlavní nabídka a panel nástrojů. V pravé části se pak nachází kontrolní panel, který slouží k úpravě parametrů právě modelovaných objektů. V dolní části okna se pak nachází stavový řádek, který zobrazuje například



Obrázek B.2: Grafické rozhraní editoru.

pozici kurzoru ve scéně. Pojem *scéna* označuje v podstatě to, co vidíte skrz pohledy v hlavním okně, tedy prostor obsahující všechny modelované objekty. Ihned po spuštění editoru je scéna prázdná, čili pohledy nezobrazují nic než mřížku usnadňující orientaci v 3D prostoru.

Ovládání pohledů

Každý pohled standardně zobrazuje mřížku. Slabšími čarami je vyznačena takzvaná hlavní mřížka, silnějšími takzvaná vedlejší, která má oproti hlavní mřížce desetinásobnou rozteč. Nejsilnějšími čarami jsou vyznačeny souřadnicové osy. Souřadný systém v editoru používá tři osy x, y a z tak, že pokud osa x míří doleva a osa y nahoru, potom osa z roste směrem k pozorovateli.

K přibližování a oddalování pohledů slouží kolečko myši. Chcete-li pohledem posunout, táhněte myší při stisknutém prostředním tlačítku. Ve 3D pohledu navíc

funguje rotace, která se spustí tak, že při stisknutém prostředním tlačítku myši zmáčknete klávesu Alt.

Každý pohled má v horní části štítek, který popisuje, jakým způsobem zrovna daný pohled zobrazuje scénu. Kliknete-li pravým tlačítkem myši na štítek pohledu, vyskočí rozbalovací nabídka, která vám umožní tento způsob zobrazení měnit. Nabídka pro 3D pohled je mírně odlišná od nabídky pro 2D pohledy, mají však tyto společné položky:

- **Show grid**
  Vypíná/zapíná zobrazení mřížky.
- **Show generator envelopes**
  Vypíná/zapíná zobrazení obálek generátorů.
- **Show generator pulls**
  Vypíná/zapíná zobrazení táhel generátorů.
- **Wireframe render**
  Zobrazuje pouze drátěné modely blobů ve scéně.
- **Flat render**
  Zobrazuje plochy blobů bez vyhlazování hran.
- **Smooth render**
  Zobrazuje plochy blobů s vyhlazenými hranami.
- **Smooth + specular render**
  Zobrazuje plochy blobů s vyhlazenými hranami a odlesky.
- **Reset**
  Nastaví výchozí polohu pohledu.

Rozbalovací nabídka pro 2D pohled ještě navíc umožňuje zvolit jeho takzvanou orientaci:

- **TOP**
  Pohled shora. Hodnoty na horizontální ose x rostou na obrazovce zleva doprava a na vertikální ose z shora dolů.
- **BOTTOM**
  Pohled zdola. Horizontální x roste zleva doprava, vertikální z zdola nahoru.

- **LEFT**
  Pohled zleva. Horizontální z roste zleva doprava, vertikální y zdola nahoru.
- **RIGHT**
  Pohled zprava. Horizontální z roste zprava doleva, vertikální y zdola nahoru.
- **FRONT**
  Pohled zepředu. Horizontální x roste zleva doprava, vertikální y zdola nahoru.
- **BACK**
  Pohled zezadu. Horizontální x roste zprava doleva, vertikální y zdola nahoru.

Tím, že měníte orientaci pohledu, můžete tedy scénu sledovat z různých stran. Po spuštění editoru scéna ještě neobsahuje žádné objekty, takže tento princip tolik nevynikne. To ovšem v následující části rychle napravíme.

B.4 Modelování blobů

Nyní ukážeme, jak se v editoru BlobsEd pracuje s bloby. Jak už jsme psali dříve, bloby se vytvářejí seskupováním generátorů. Editor nabízí ve své základní verzi dva generátory – bodový a úsečkový. Každý generátor má kolem sebe takzvanou *obálku* (envelope), která vyznačuje, kam až sahá jeho vliv, nebo-li, zjednodušeně řečeno, kam až se daný generátor může nafouknout, jestliže je součástí blobu. Velikost obálky udává takzvaný *poloměr* (radius) generátoru. Například obálka kolem bodového generátoru je povrch koule, jejíž poloměr je právě poloměr generátoru. Práci s generátory názorně předvedeme v následující části, kde vytvoříme pomocí nástrojů editoru jednoduchý blob.

Modelování činky

Tento návod vás krok za krokem provede vytvořením jednoduchého blobu ve tvaru činky.

1. Spusťte editor BlobsEd a kliknutím na tlačítko  na panelu nástrojů zapněte přichytávání k mřížce, což umožní přesné umísťování objektů.
2. Z hlavního menu vyberte Insert – Point generator, nebo klikněte na tlačítko  na panelu nástrojů. Když nyní začnete pohybovat myší v libovolném

pohledu, uvidíte obálku nového bodového generátoru. Zároveň se dole ve stavovém řádku zobrazuje aktuální pozice tohoto generátoru ve scéně.

3. Kliknutím levým tlačítkem myši v TOP pohledu umístěte jeden generátor na pozici X: -10, Y: 0, Z: 0 a druhý na pozici X: 10, Y: 0, Z: 0. Kliknutím pravým tlačítkem na libovolném místě v pohledu pak zrušte vytváření bodových generátorů.

4. Jakmile máte ve scéně nějaké objekty, můžete je kliknutím levého tlačítka myši označit. Když takto označíte jeden z našich dvou bodových generátorů, na pravé straně hlavního okna se zobrazí panely, které vám umožní měnit parametry vybraného generátoru. Panel nadepsaný Object parameters nastavuje parametry společné všem objektům. Takovými parametry jsou jméno a pozice v prostoru. Každý objekt má přiřazeno jméno, podle kterého ho můžeme ve scéně odlišit od ostatních objektů stejného typu. Panel nadepsaný Point generator params nastavuje parametry bodového generátoru. Zde můžeme nastavit jeho poloměr a *váhu* (weight).

   Co je poloměr generátoru už jsme popsali výše. Každý generátor má však navíc ještě přiřazeno číslo, kterému se říká váha. Ta ovlivňuje, zhruba řečeno, jak moc tento generátor přispívá k ploše výsledného blobu. Váha každého generátoru je při jeho vytvoření nastavena na 1. Vyšší čísla znamenají větší příspěvek k výsledné ploše, nižší naopak příspěvek menší. Jako váhu lze zadat i záporné číslo, což znamená, že generátor nebude k objemu blobu přispívat, ale bude jej naopak odebírat, což umožňuje vytvářet v blobech různé díry.

5. Generátoru nalevo nastavte jméno "Left", napravo "Right". Oběma generátorům pak nastavte poloměr 7. Všimněte si, jak se jejich obálky zvětšily. Váhu obou generátorů zatím necháme nastavenou na 1.

6. Z hlavního menu vyberte Insert – Line generator, nebo klikněte na panelu nástrojů na tlačítko  . Tím začnete návrh úsečkového generátoru. Umístěte jeho koncové body na souřadnice X: -8, Y: 0, Z: 0 a X: 8, Y: 0, Z: 0. Kuličkám, které vidíte v koncových bodech, se říká táhla. Kliknutím na libovolné z nich úsečkový generátor označte.

7. Nastavte jeho jméno na "Middle". Všimněte si, že stejně jako u bodového generátoru můžete nastavit poloměr a váhu. Tyto necháme nezměněny. Dále můžete nastavit polohy obou bodů generátoru. Jeden z nich je samotná pozice objektu. Pokud změníte pozici, generátor se celý přesune na tuto pozici. Polohu druhého bodu pak můžete nastavit v části kontrolního panelu nadepsané Second point. Chcete-li polohu bodů generátoru nastavit pomocí myši

ve scéně, stiskněte tlačítko Edit generator. Poté stiskněte a držte levé tlačítko myši nad příslušným táhlem. Tažením myší pak můžete měnit tvar generátoru. Jakmile jste s tvarem spokojeni, opětovným stisknutím tlačítka Edit generator se vrátíte do normálního módu, kde opět můžete pracovat se všemi objekty scény.

8. Ve scéně již máme všechny generátory, které potřebujeme. Nyní z nich vytvoříme blob. V hlavní nabídce vyberte Insert – Blob, nebo na panelu nástrojů stiskněte tlačítko  . Blob se ve scéně zobrazuje jako hvězdička. V tuto chvíli není důležité, kam tento blob umístíte. Jeho pozice ve scéně nehraje žádnou roli. Jakmile je blob umístěn, označte všechny generátory ve scéně. Vícenásobné označení se provádí tak, že při kliknutí levého tlačítka držíte zmáčknutou klávesu Ctrl. Nyní z hlavní nabídky vyberte Edit – Attach to blob, nebo stiskněte na panelu nástrojů tlačítko  . Nyní levým tlačítkem klikněte na blob ve scéně. Tím se generátory seskupí a vytvoří se kolem nich povrch blobu.

   Jakmile je blob vybrán, objeví se na pravé straně opět kontrolní panel, který nastavuje jeho parametry. Každý blob má dva číselné parametry a to takzvanou *hladinu* (level) a *přesnost* (precision).

   Hladina musí být číslo větší než nula a menší nebo rovno hodnotě označené na panelu jako Max level. Čím vyšší hodnota hladiny, tím menší má výsledný blob objem. Při vysokých hodnotách může povrch blobu dokonce úplně zmizet. Pokud zadáme hodnoty blízké nule, bude povrch blobu vyplňovat téměř celou obálku. Při zadávání hladiny je ovšem třeba mít na paměti, že síť trojúhelníků představující povrch blobu, je pouze aproximací plochy, která je definována matematickou formulí. Plocha blobu je tedy zobrazena jen přibližně, což způsobuje, že při velmi nízké hladině se na výsledném povrchu začnou objevovat nežádoucí artefakty. Řešením může být buď malinko zvýšit hodnotu hladiny nebo zvýšit přesnost zobrazení (viz. dále).

   Parametr přesnost udává, jak hustá bude síť trojúhelníků představující povrch blobu. Čím větší je číslo přesnosti, tím je povrch blobu zobrazen věrněji, bez ostrých hran a přechodů. O to více času je ovšem potřeba na kalkulaci tohoto povrchu, takže při vysokých hodnotách přesnosti může počítač pracovat velmi dlouho. V extrémním případě může dokonce dojít k vyčerpání dostupné paměti a pádu programu. Proto se při výpočtu povrchu vždy zobrazuje dialog, který ukazuje průběh celého výpočtu. Když výpočet trvá příliš dlouho, můžete jej přerušit tlačítkem Cancel. Namísto hvězdičky je potom takový blob ve scéně představován nápisem INVALID MESH, který indikuje, že výpočet povrchu

byl přerušen. Výpočet povrchu blobu můžete opět kdykoliv spustit tlačítkem Polygonise na kontrolním panelu.

Na kontrolním panelu můžete dále zvolit barvu povrchu blobu tím, že kliknete na barevné pole vedle nápisu Color. Níže pak vidíte seznam všech generátorů v tomto blobu uskupených.

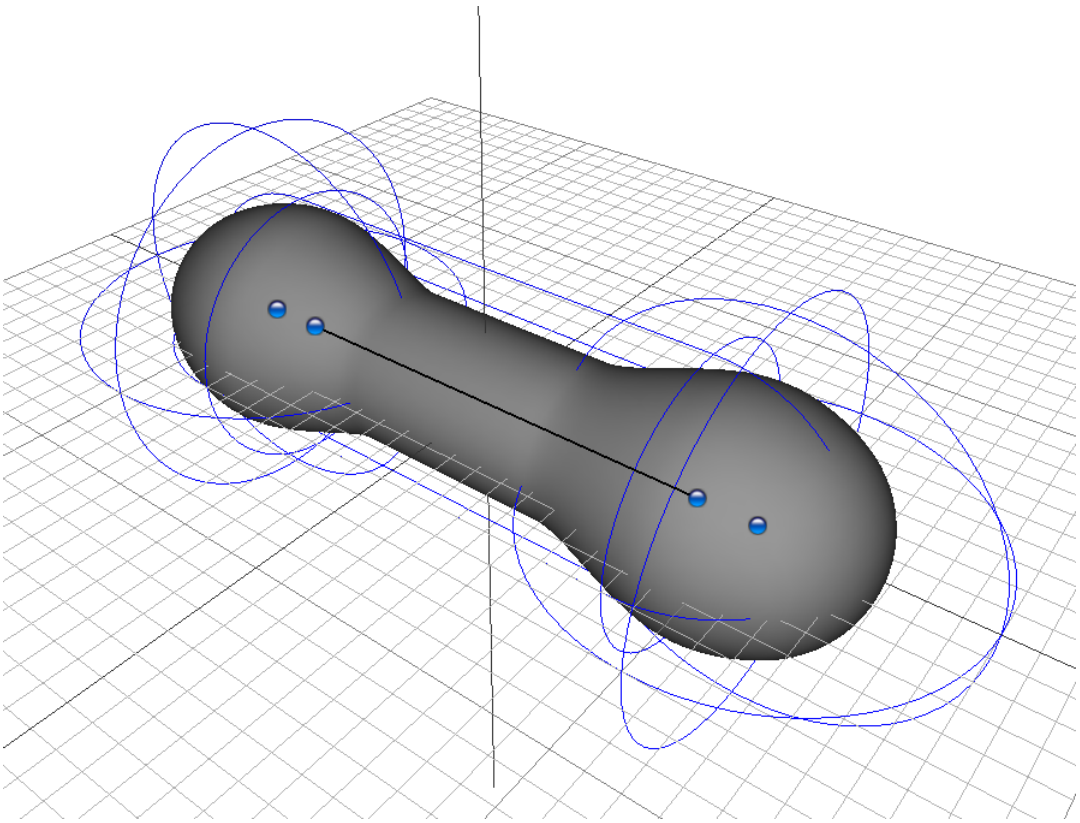
Nyní máme vytvořen blob ve tvaru jakési činky. Tato skupina generátorů se nyní chová jako jeden objekt. Pokud se nám tvar výsledného blobu nelíbí a chtěli bychom upravit parametry některého z jeho generátorů, je třeba stisknout tlačítko Sub-object na kontrolním panelu. Tím se zpřístupní jednotlivé generátory a je možné s nimi pracovat samostatně. Generátory je v tomto módu také možno vybírat skrze jejich seznam na kontrolním panelu. Klávesa Ctrl opět umožňuje vícenásobný výběr. Stisknutím tlačítka Detach generator můžeme vybranou skupinu generátorů z blobu vyjmout, takže se stanou opět samostatnými objekty. Jakmile jsme s úpravou generátorů skončili, opět stiskneme tlačítko Sub-object, které nás vrátí zpět do normálního módu. Automaticky se přitom spustí přepočet plochy blobu, takže rovnou vidíme výsledek naších úprav.

Vytvořený blob si nyní můžete prohlédnout ve 3D pohledu. Měl by vypadat jako na obrázku B.3. Případně můžete vyzkoušet různé způsoby zobrazení nebo zkusit experimentovat s nastavením parametrů. Vyzkoušejte například, jaký vliv má na tvar blobu váha jednotlivých generátorů.

Další editační možnosti

Na panelu nástrojů nejdete kromě jiných tlačítka  a  . První z nich zapíná takzvaný výběrový mód, ve kterém je možné objekty myší pouze označovat. Druhé tlačítko zapíná mód posunovací, ve kterém může uživatel kliknutím levým tlačítkem a následným tažením myší s objekty a jejich skupinami také pohybovat. Další tlačítko  slouží k odstranění označených objektů ze scény. Stejnou funkci má také položka nabídky Edit – Delete selected.

Každou změnu provedenou ve scéně lze navíc vrátit zpět pomocí položky hlavního menu Edit – Undo. Pokud chceme navrácenou změnu naopak znovu provést, vybereme Edit – Redo.



Obrázek B.3: Ukázka jednoduchého blobu ve tvaru činky.

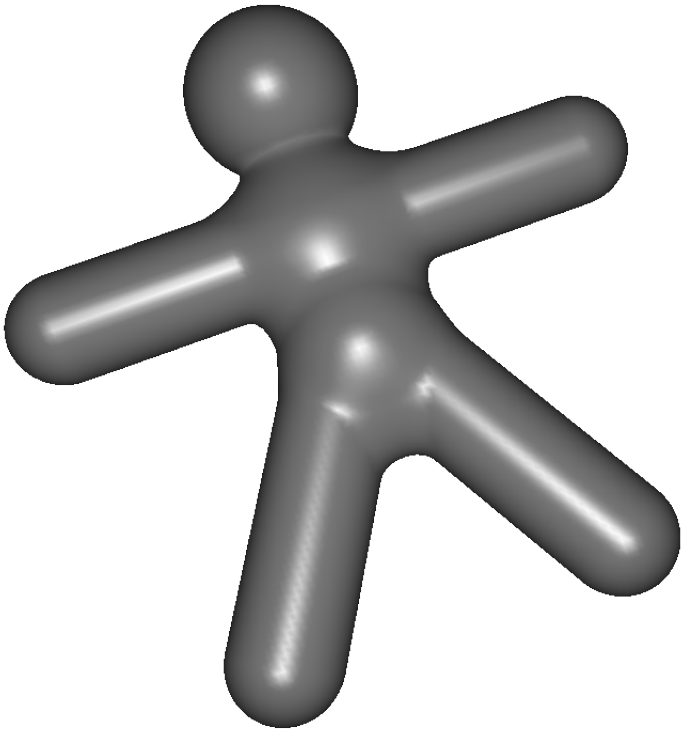
B.5 Ukládání a export scény

Scénu s naším činkovým blobem můžeme nyní uložit na disk pomocí File – Save z hlavní nabídky nebo tlačítka  z panelu nástrojů. Scénu můžeme uložit jako čisté XML, což je textová reprezentace scény, nebo XML zkomprimované metodou Gzip, které zabírá na disku méně místa. Pozor, jestliže jsou ve scéně bloby s hustou sítí trojúhelníků. Ukládání scény pak může trvat delší dobu. Takto uloženou scénu potom můžeme z disku načíst pomocí File – Open file nebo tlačítka  . Chceme-li začít pracovat s prázdnou scénou, vybereme File – New scene nebo stiskneme  .

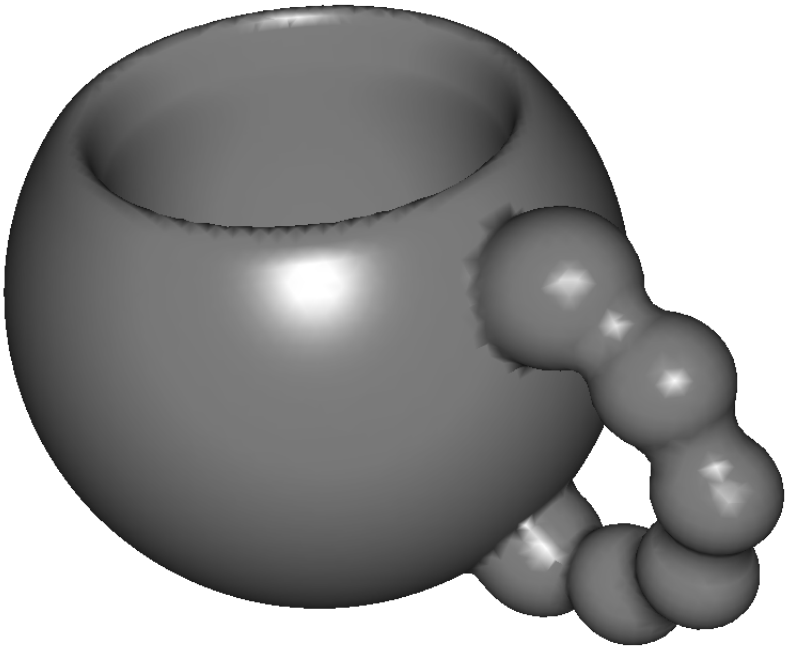
Bloby ve scéně je také možno exportovat pomocí File – Export do formátu Wavefront OBJ, který umí číst velké množství jiných 3D editorů, například známý modelovací program Blender. Ten však neumí z tohoto formátu načíst informace o normálách vrcholů, takže importované bloby nejsou bohužel tak hladké, jako v editoru BlobsEd.

B.6 Ukázka složitějších blobů

Pokud otevřete některý se souborů blobman.bes nebo cup.bes nacházející se ve složce samples, můžete si prohlédnout o něco komplikovanější bloby ve tvaru panáčka či hrníčku (obrázky B.4 a B.5).



Obrázek B.4: Ukázka blobu ve tvaru panáčka. Byly použity čtyři úsečkové a jeden bodový generátor.



Obrázek B.5: Ukázka blobu ve tvaru hrníčku. Bylo použito deset bodových generátorů.