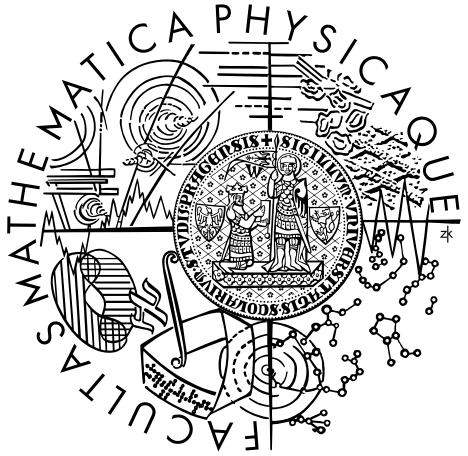
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Lukáš Ondráček

Zobrazování mnohostěnů v různých dimenzích

Informatický ústav Univerzity Karlovy

Vedoucí bakalářské práce:	doc. Mgr. Robert Šámal, Ph.D.
Studijní program:	Informatika
Studijní obor:	Obecná informatika

Praha 2016

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

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

V ..... dne ..... Podpis autora

Název práce: Zobrazování mnohostěnů v různých dimenzích

Autor: Lukáš Ondráček

Ústav: Informatický ústav Univerzity Karlovy

Vedoucí bakalářské práce: doc. Mgr. Robert Šámal, Ph.D., Informatický ústav Univerzity Karlovy

Abstrakt: Práce se zabývá implementací aplikace pro vykreslování tří- a vícerozměrných mnohostěnů s možností plynulé rotace. Aplikace Geometric Figures pro Linux a Windows je napsána v jazyce C s použitím OpenGL a podporuje zásuvné moduly psané v Pythonu. K vykreslování je použita iterovaná perspektiva s barvením hran podle jejich polohy. Aplikace dále umožňuje hledání konvexního obalu množiny bodů, řezy tělesa nadrovinou, stelaci, vytvoření geometricky duálního tělesa a odřezávání částí tělesa; vše nezávisle na počtu rozměrů. Výhodou této aplikace oproti již existujícím je její snadná rozšiřitelnost pomocí modulů a otevřenost zdrojového kódu.

Klíčová slova: kreslení, zobrazování, mnohostěny, dimenze

Title: Drawing polytopes in various dimensions

Author: Lukáš Ondráček

Institute: Computer Science Institute of Charles University

Supervisor: doc. Mgr. Robert Šámal, Ph.D., Computer Science Institute of Charles University

Abstract: The thesis describes implementation of an application for drawing three- and multidimensional polytopes allowing their fluent rotation. The application Geometric Figures for Linux and Windows is written in the C language using the OpenGL library and it supports plug-ins written in the Python language. Iterated perspective projection and edges coloring according to their location is used. In addition, the application is able to generate the convex hull of a set of points, cut figures with hyperplanes, stellate figures, create geometrically dual polytopes and cut off parts of figures; all functions are independent to the number of dimensions of the polytopes. The application profits from its easy extensibility using modules and its code being open-source.

Keywords: drawing, polytope, dimension

Děkuji doc. Mgr. Robertu Šámalovi, Ph.D., za vedení práce a vše, co s tím souvisí.

Obsah

Úvod

Je přirozené představovat si trojrozměrná tělesa nebo rovinné útvary, u vícerozměrných prostorů ale představivost často selhává. Už připustit čtyři na sebe kolmé přímky není úplně samozřejmé. Na Internetu je možné nalézt obrázky i animace trojrozměrných projekcí známých čtyřrozměrných těles, ty jsou ale obvykle neinteraktivní a neumožňují tak uživateli, aby tělesem sám otáčel.

Aplikace Geometric Figures, vyvíjená od roku 2014, dává uživateli právě tuto možnost. Vykresluje vícerozměrná tělesa iterováním perspektivní projekce, tedy stejným způsobem, na který jsme zvyklí například z fotografií, a pro lepší orientaci v prostoru obarvuje hrany těles podle jejich polohy.

Je možné si vybrat z předem připravených pravidelných mnohostěnů nebo vytvářet vlastní tělesa. Součástí aplikace je možnost generování konvexního obalu zadané množiny bodů a jeho přepočítání při posunu těchto bodů. Pro úpravy těles je k dispozici například odřezávání jejich částí (např. vrcholů, hran, apod.), stelace (zhvězdnatění) a vytváření geometricky duálních těles. Pro snížení počtu rozměrů je také možné zobrazit část tělesa na řezu nadrovinou. Veškeré funkce aplikace jsou navrženy nezávisle na počtu rozměrů, který je z implementačních důvodů omezen na 100.

Z existujících projektů je potřeba zmínit aplikaci Stella4D, jejímž autorem je Robert Webb (Webb, 9. 5. 2016). Její vývoj započal již v roce 2001 a nabízí proto obrovské množství funkcí. Jde ale o proprietární aplikaci, která je v plné verzi placená.

Geometric Figures má naproti tomu otevřený zdrojový kód (pod licencí GNU General Public License verze 3) a je dobře rozšiřitelná pomocí zásuvných modulů Pythonu. Aplikace byla od začátku určena spíše pokročilejším uživatelům, čemuž odpovídá i její ovládání inspirované UNIXovým editorem Vi.

Je k dispozici verze pro Linux i Windows. Hlavní část je napsána v jazyce C s použitím OpenGL a interpretu Pythonu, některé součásti jsou pak jako zásuvné moduly implementovány v Pythonu.

1. Analýza

1.1 Terminologie a vlastnosti mnohostěnů

Prostorem budeme mít v této práci na mysli afinní prostor s euklidovskou normou a *nadrovinou* jeho o jednu dimenzi menší podprostor.

Definice 1 (mnohostěn, stěna, faseta, hřeben, hrana, vrchol)**.** *Mnohostěn $t \subset \mathbb{R}^{d'}$ dimenze $d > 0$ je omezený uzavřený geometrický útvar generující prostor dimenze d. Hranice mnohostěnu v tomto prostoru je navíc souvislá a je tvořena konečným počtem $(d-1)$-rozměrných mnohostěnů – faset. Mnohostěn dimenze $d = 0$ je bod.*

Stěna mnohostěnu je jeho faseta, nebo stěna některé jeho fasety.

Hřeben mnohostěnu je faseta některé jeho fasety (tedy $(d-2)$-rozměrná stěna).

Hrana je jednorozměrná stěna, vrchol je nularozměrná stěna.

Výraz *těleso* zde budeme využívat ve stejném významu jako mnohostěn. Stěny tělesa v obvyklé definici zahrnují také celé těleso a prázdnou množinu, ty zde nebudeme uvažovat pro zjednodušení konstrukcí typu „pro všechny stěny tělesa. . . “ v pseudokódu. Souvislou hranicí tělesa máme na mysli, že nejsou povoleny „dutiny, které neústí na povrch“.

V anglické terminologii se někdy pro označení $(d-2)$-rozměrné stěny a $(d-3)$-rozměrné stěny používají pojmy *ridge* a *peak*, které nemají obecně užívaný český ekvivalent; *hřeben* tak vznikl překladem anglického *ridge*. Naproti tomu *faseta* (*facet*) je běžně užívaná v obou jazycích. Anglické názvy jsou použity v kódu aplikace.

Definice 2 (graf svazu stěn, rodič, syn)**.** *Množina všech stěn tělesa (tentokrát včetně prázdné množiny i celého tělesa) uspořádaná inkluzí tvoří svaz. Graf tohoto svazu má za vrcholy všechny stěny tělesa a hrany spojují vždy mnohostěn s jeho fasetou; mnohostěn v tomto vztahu nazveme rodičem a jeho fasetu synem.*

Hlavní částí datové struktury uchovávající těleso je seznam jeho faset (příp. pozice vrcholu). Graf svazu stěn tak tuto strukturu dobře znázorňuje a budeme jej proto používat k popisu procházení strukturou i následným odhadům časové složitosti.

V aplikaci a jejím kódu jsou pro jednoduchost za *vrcholy* označovány i ty body, které nakonec nejsou součástí hranice tělesa. (Např. vstupem pro hledání konvexního obalu je seznam vrcholů, z nichž některé jsou později označeny jako *volné*.) Často je tím také odlišována nularozměrná instance struktury tělesa od vektoru souřadnic. V analýze budeme využívat obvyklou terminologii popsanou výše.

1.1.1 Konvexní mnohostěny

Tvrzení 1. *Pro konvexní mnohostěn t platí, že ke každé jeho stěně s ($\neq t$) existuje nadrovina p splňující:*

1. *t leží v jednom z uzavřených poloprostorů určených nadrovinou p,*
2. *$p \cap t = s$.*

Naopak pro každou nadrovinu p splňující první podmínku platí, že $p \cap t$ je stěna tělesa, nebo prázdná množina.

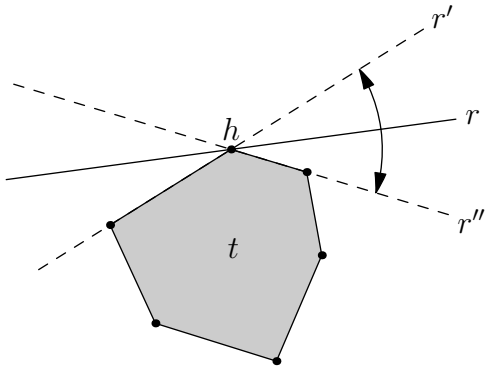
Ve skriptech prof. Matouška (Matoušek, 2002) je Tvrzení 1 uvedeno jako definice stěny konvexního mnohostěnu. Zde využíváme definici mnohostěnu v obecnějším kontextu a tvrzení ponecháme bez důkazu.

Lemma 2. *Každý hřeben konvexního tělesa je součástí právě dvou jeho faset.*

Důkaz. Mějme hřeben h tělesa t a k němu příslušnou nadrovinu r z Tvrzení 1 (viz Obrázek 1.1). Nadrovinu můžeme otáčet kolem h dokud se její průnik s tělesem (v inkluzi) nezvětší. Jakmile to nastane, průnik musí generovat větší prostor než h, tedy celou nadrovinu a podle Tvrzení 1 v ní leží faseta. Další rotací by se porušila první podmínka předchozího tvrzení. Otáčením na opačnou stranu získáme druhou fasetu; pokud bychom získali tu samou, muselo by to být celé t, což je spor s tím, že jde o jeho fasetu.

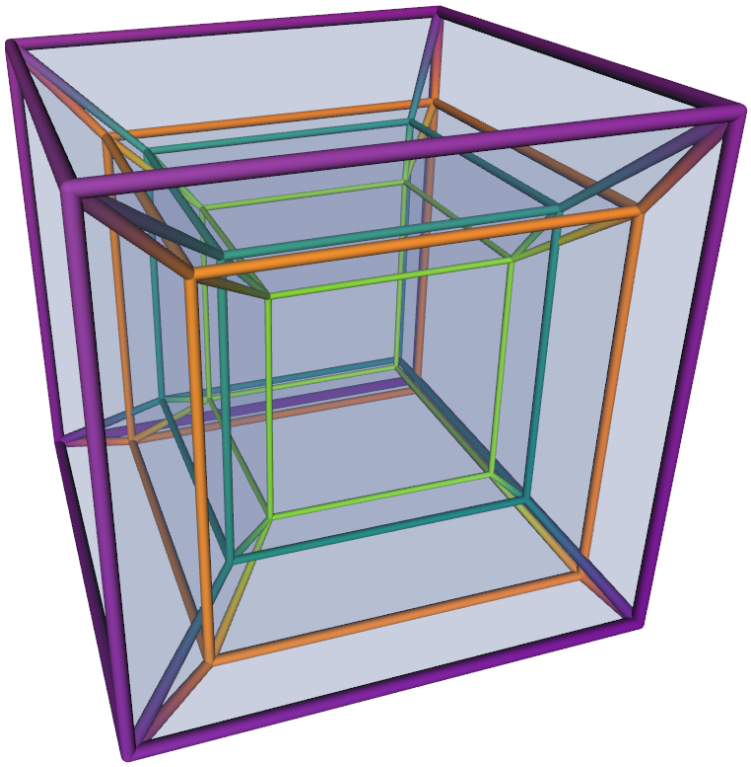
Bude-li těleso d-rozměrné, jeho hřeben bude $(d-2)$-rozměrný a ortogonálním doplňkem prostoru hřebenu v prostoru tělesa pak bude rovina. V této rovině bude probíhat rotace, která je tak dobře definovaná, a dostaneme pomocí ní všechny nadroviny obsahující h. Prostor r bude v průniku s rovinou tvořen přímkou a prostor hřebenu h bodem. (Obrázek 1.1 můžeme chápat jako dvourozměrný příklad nebo jako řez popsanou rovinou.)

□



Obrázek 1.1: K důkazu Lemmatu 2. K hřebenu h rotací příslušné nadroviny r nalezneme právě dvě fasety, které h obsahují.

1.2 Vykreslování



Obrázek 1.2: Pětirozměrná hyperkrychle jako ukázka projekce a barvení hran.

1.2.1 Projekce

Pro vykreslení vícerozměrného tělesa je potřeba použít zobrazení, které sníží počet jeho rozměrů. Zde je toho docíleno opakovaným použitím perspektivy.

Perspektivu si zjednodušeně můžeme představit tak, že vyjdeme ze souřadnic původního bodu bez poslední (tedy snížíme počet rozměrů) a zahozená souřadnice pak bude ovlivňovat vzdálenost bodu od středu souřadného systému nižší dimenze. Ona poslední souřadnice bude představovat vzdálenost od kamery: Bude-li zobrazovaný bod daleko, bude jeho obraz blízko středu souřadného systému; bude-li blízko, obraz bude dále od středu (při zachování ostatních souřadnic).

Použijeme-li toto zobrazení na trojrozměrný prostor, dostaneme známý efekt: Při posunu tělesa kolmo od projekční roviny (od kamery) se bude jeho obraz (na fotografii) přibližovat středu.

Stejné je to i v případě projekce ze 4D do 3D, jenom musíme připustit, že čtyřrozměrná kamera má trojrozměrný povrch, na nějž se bude promítat; ten leží v jedné nadrovině, na kterou je kolmá osa určující vzdálenost. Budeme-li tělesa posouvat dále od kamery, budou se jejich obrazy přibližovat počátku i sobě navzájem.

Takto si například můžeme představit kostru čtyřrozměrné krychle. Ta je zepředu a zezadu ohraničena krychlemi, které jsou spojené, což v perspektivní projekci vytváří (3D) obraz krychle v krychli. Vnitřní krychle představuje tu vzdálenější, vnější tu bližší, pojem vzdálenosti tak má stále jasný význam, vnímáme-li vzniklý (3D) obraz jako celek.

Zvláštní situace nastane, použijeme-li perspektivu vícekrát, abychom snížili počet rozměrů o více než jeden. Začneme-li se například na onu „krychli v krychli“ dívat z některé konkrétní strany, objeví se druhá vzdálenost kamery – jiná než ta předchozí –, což je naprosto nepřirozené.

Zajímavou vlastností perspektivy je rozdíl v tom, jestli se kamera posune blíž, nebo se obraz pouze zvětší. Jsou-li pozorované objekty daleko od kamery (což se kompenzuje zvětšením), nejsou až tak patrné rozdíly jejich vzdáleností od kamery. (Efekt velkého měsíce na fotografiích.)

Perspektiva v aplikaci je proto parametrizovaná nastavitelnou vzdáleností kamery, díky čemuž je možné dosáhnout i téměř rovnoběžného promítání. Tuto vzdálenost je možné nastavit nezávisle pro každou iteraci perspektivy, tedy pro všechny osy kromě prvních dvou.

Formální popis

Chceme zobrazit d-rozměrnou kouli umístěnou v počátku souřadného systému $\mathbb{R}^d$ na $(d-1)$-rozměrnou kouli se středem v počátku $\mathbb{R}^{d-1}$, přičemž d-tá osa $\mathbb{R}^d$ směřuje ke kameře. Zobrazení je navíc parametrizováno vzdáleností kamery.

Definice 3 (perspektiva)**.** *Nechť r je poloměr viditelné koule a $k_d > r$ vzdálenost kamery od počátku d-rozměrného souřadného systému. Pak*

$$P_d : \{(x_1, x_2, \ldots, x_d) \in \mathbb{R}^d \mid x_d < k_d\} \to \mathbb{R}^{d-1},$$
$$P_d((x_1, x_2, \ldots, x_d)) = \frac{f_d}{k_d - x_d} \cdot (x_1, x_2, \ldots, x_{d-1}),$$

kde ohnisková vzdálenost $f_d = \sqrt{k_d^2 - r^2}$.

Ohnisková vzdálenost f_d udává vzdálenost nadroviny, v níž se budou zachovávat vzájemné vzdálenosti mezi body, od kamery. Proč právě tato hodnota zajišťuje zobrazení d-rozměrné koule na $(d-1)$-rozměrnou kouli, je znázorněno na Obrázku 1.3:

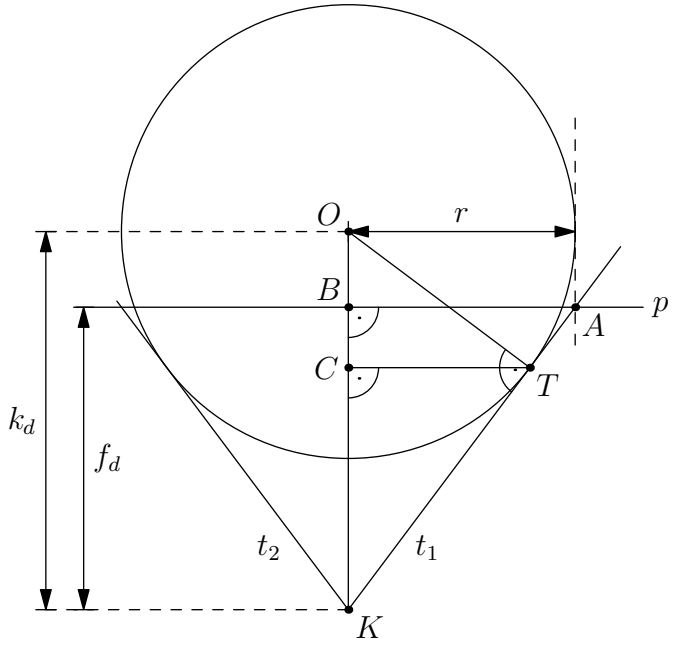
Kruh představuje řez d-rozměrnou koulí o poloměru r se středem v počátku souřadného systému O. Na svislé ose je d-tá souřadnice. Bod K představuje kameru vzdálenou k_d od O a polopřímky t_1 a t_2 vytyčují její zorné pole. Přímka p představuje nadrovinu, do níž budeme promítat.

Bod T bude mít v projekci maximální vzdálenost od počátku; tato vzdálenost bude právě r, bude-li zachováno měřítko v nadrovině p procházející bodem B. Projekce bodu A pak bude splývat s projekcí T. Zbývá dokázat, že $|KB| = \sqrt{k_d^2 - r^2}$.

Trojúhelník KOT je podobný trojúhelníku KTC (shodný úhel u K a pravý) a ten je podobný trojúhelníku KAB (shodné úhly). Vzdálenost $|AB| = |OT|$, tedy trojúhelníky KAB a KOT jsou shodné a hledaná vzdálenost $|KB| = |KT|$. Z Pythagorovy věty $|KT| = \sqrt{|KO|^2 - |OT|^2} = \sqrt{k_d^2 - r^2}$.

Definice 4 (iterovaná perspektiva)**.** *Iterovaná perspektiva z d-rozměrného prostoru do d'-rozměrného (pro $d > d'$) je zobrazení:*

$$P_{d,d'} : \{x \in \mathbb{R}^d \mid \|x\| \leq r\} \to \{x \in \mathbb{R}^{d'} \mid \|x\| \leq r\},$$
$$P_{d,d'}(x) = \begin{cases} P_d(x) & \text{pro } d' = d-1, \\ P_{d-1,d'}(P_d(x)) & \text{jinak.} \end{cases}$$



Obrázek 1.3: K perspektivní projekci. Kruh představuje řez d-rozměrnou koulí se středem v počátku O. Bod K představuje kameru, polopřímky t_1 a t_2 vytyčují její zorné pole a přímka p je nadrovina, do níž chceme promítat.

Při vykreslování se na každý vrchol tělesa aplikuje zobrazení $P_{d,2}$, které jej zobrazí do kruhu o poloměru r; ten se poté vykreslí na obrazovku. (O poslední iteraci se částečně stará knihovna OpenGL.)

Zobrazení se také aplikuje na velikosti koulí, představujících vrcholy, a konců komolých kuželů, které je spojují.

1.2.2 Rotace

Rotace se často definuje jako otočení kolem něčeho: v rovině kolem bodu, v prostoru kolem osy. Otáčet ve čtyřrozměrném prostoru kolem roviny už je ale trochu matoucí, proto zde budeme o rotaci mluvit jako o otočení o daný úhel kolem daného bodu v dané rovině, nezávisle na počtu rozměrů.

Otáčet budeme vždy v rovině dané dvěma poloosami souřadného systému kolem počátku a kladný směr otáčení pak bude určen pořadím poloos. Jedna rotace tak nikdy nezmění více než dvě souřadnice a lze ji zadefinovat obvyklým způsobem:

Definice 5 (rotace)**.** *Rotace v $\mathbb{R}^d$ ve směru poloos i, j je zobrazení:*

$$R_{i,j} : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d,$$

$$R_{i,j}(x,\theta) = \begin{array}{c} \\ \\ \\ i \\ \\ \\ \\ j \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccccccc} & & & & i & & & & j & & \end{array} \\ \begin{pmatrix} 1 & & 0 & & & & & & & & \\ & \ddots & & & & & & & & & \\ 0 & & 1 & & & & & & & & \\ & & & \cos\theta & & & & -\sin\theta & & & \\ & & & & 1 & & 0 & & & & \\ & & & & & \ddots & & & & & \\ & & & & 0 & & 1 & & & & \\ & & & \sin\theta & & & & \cos\theta & & & \\ & & & & & & & & 1 & & 0 \\ & & & & & & & & & \ddots & \\ & & & & & & & & 0 & & 1 \end{pmatrix} \end{array} \cdot x.$$

V aplikaci je s prohlíženým tělesem spojena jeho matice otočení (na začátku jednotková), kterou se při překreslování před projekcí vynásobí pozice všech vrcholů. Při rotaci se pouze tato matice vynásobí maticí dílčí rotace.

1.2.3 Barvení hran a vrcholů

Pro usnadnění orientace ve vícerozměrném prostoru je možné hrany a vrcholy obarvit podle jejich souřadnic.

Například u čtyřrozměrných těles jsou ve výchozím nastavení body s kladnou čtvrtou souřadnicí (blízko u kamery) zbarveny do červena a body se zápornou (daleko od kamery) do zelena, mezi nimi se pak tvoří lineární barevný přechod. S otáčením tělesa se barvy mění podle aktuálních souřadnic.

Definice 6 (barva bodu v prostoru)**.** *Nechť r je poloměr viditelné koule a b_0 barva (třísložkový vektor RGB) přiřazená počátku souřadného systému. Nechť dále pro každé $i \in \{1, 2, \ldots, d\}$: b_i^+ a b_i^- jsou barvy přiřazené k i-té kladné a záporné poloose a $a_i^+, a_i^- \in [0,1]$ určují krytí (neprůhlednost) těchto barev. Pak je barva $B(x)$ bodu $x \in \mathbb{R}^d$ určena následovně:*

$$B(x) = \begin{cases} s/t & \text{pro } t \geq 1, \\ s + (1-t) \cdot b_0 & \text{jinak,} \end{cases}$$

kde

$$s = \sum_{i=1}^{d} \frac{|x_i|}{r} \cdot a_i^{\operatorname{sgn}(x_i)} \cdot b_i^{\operatorname{sgn}(x_i)}$$

$$t = \sum_{i=1}^{d} \frac{|x_i|}{r} \cdot a_i^{\operatorname{sgn}(x_i)}$$

Při plných krytích $a_i^{\pm}$ tedy vektory b_0, b_1^+, b_1^-, b_2^+, b_2^-, … určují barvy v bodech $(0,0,\ldots)$, $(r,0,\ldots)$, $(-r,0,\ldots)$, $(0,r,\ldots)$, $(0,-r,\ldots)$, …, mezi nimiž se vytváří barevný přechod.

Při vykreslování se určí barvy vrcholů, barvy hran pak tvoří lineární přechod mezi nimi.

1.3 Správa afinních prostorů

Afinní prostor je určen bodem a ortonormální bází.

Tato kapitola slouží spíše jako seznam časových složitostí. Obsahuje mírně upravenou Gramovu-Schmidtovu ortogonalizaci a další nástroje lineární algebry popsané ve skriptech Milana Hladíka (Hladík, 15. 2. 2016).

Ortogonalizace vektoru vzhledem k ortonormální bázi

Pro $B \subset \mathbb{R}^d$ ortonormální bázi podprostoru $\mathbb{R}^d$ a $v \in \mathbb{R}^d$ vektor má ortogonalizovaný vektor tvar:

$$v' = v - \sum_{b \in B} \langle v, b \rangle b.$$

Složitost je $\mathcal{O}(d^2)$. Pro každý vektor báze se počítá skalární součin.

Rozšíření prostoru novým bodem

Rozdíl nového bodu a bodu afinního prostoru ortogonalizujeme vzhledem k bázi prostoru, výsledný vektor normalizujeme na velikost 1 a přidáme k bázi.

Složitost je $\mathcal{O}(d^2)$.

Zjištění příslušnosti bodu k prostoru

Opět ortogonalizujeme rozdíl nového bodu a bodu afinního prostoru. Je-li výsledný vektor nulový, bod patří do prostoru.

Složitost je $\mathcal{O}(d^2)$.

Vytvoření afinního prostoru ze seznamu bodů

První bod označíme za bod afinního prostoru dimenze 0. Dále bázi postupně rozšiřujeme ostatními body (současně ověřujeme, zda již do afinního prostoru nepatří).

Složitost je $\mathcal{O}(d^3)$, pokud jsou body afinně nezávislé; jinak je $\mathcal{O}(nd^2)$, kde n je počet bodů.

Výpočet normály

Výpočet normály prostoru dimenze $d-1$ s bází B v prostoru dimenze d s bází B_r: Postupně zkoušíme ortogonalizovat vektory báze B_r k bázi B, vyjde-li vektor nenulový, použijeme jej po normalizaci jako normálový.

Složitost je $\mathcal{O}(d^3)$.

Výpočet orientované vzdálenosti od nadroviny

Orientovaná vzdálenost od nadroviny je vzdálenost bodu ve směru normálového vektoru nadroviny. Body se stejným znaménkem leží ve stejném poloprostoru určeném touto nadrovinou.

Skalární součin bodu a (normalizované) normály nadroviny udává vzdálenost bodu od počátku ve směru normály. Od této vzdálenosti odečteme stejným způsobem předpočítanou orientovanou vzdálenost bodu nadroviny od počátku.

Složitost je $\mathcal{O}(d)$.

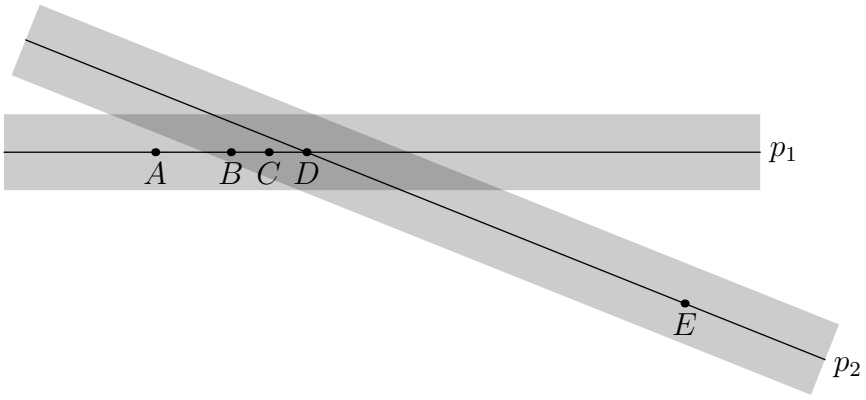
1.3.1 Reprezentace reálných čísel

Reálná čísla jsou v aplikaci reprezentována obvyklým způsobem, tedy jako čísla s pohyblivou řádovou čárkou zaokrouhlená na určitý počet platných číslic. To může v některých případech způsobovat problémy.

Typickým případem je rozhodování, zda bod leží v nadrovině, nebo na některé její straně, k čemuž je potřeba porovnávat vzdálenost bodu od nadroviny s nulou. Pro tyto případy je v aplikaci stanovená mez 0,0001 a všechna čísla z intervalu $(-0{,}0001; 0{,}0001)$ se považují za nulová. Tím ale dostává nadrovina určitou tloušťku, což může vypadat například jako na Obrázku 1.4:

Nadrovina p_1 obsahuje body A, B, C a D a nadrovina p_2 body B, C, D, E. Pro zjednodušení předpokládejme, že jde o rovinný případ, dostali jsme tak dvě různé přímky sdílející tři různé body.

Tento problém nenastává v aplikaci příliš často, ale není vyřešen a může způsobit vypsání chyby například při generování konvexního obalu.



Obrázek 1.4: Příklad dvou nadrovin v nepřesné aritmetice. Nadrovina p_1 prochází body A, B, C a D, nadrovina p_2 prochází body D a E. Kvůli nepřesné aritmetice určujeme příslušnost k nadrovině s určitou tolerancí (zvýrazněnou šedě) a do nadroviny p_2 se proto dostanou také body B a C.

1.4 Hledání konvexního obalu

Algoritmus je založen na geometrické představě a implementován pomocí několika vzájemně rekurzivních funkcí:

Funkce `CreateHull` dostane seznam bodů a vytvoří těleso představující jejich konvexní obal.

Funkce `ExpandDim` dostane těleso t a seznam bodů, které spolu s t generují prostor s o jednu dimenzi vyšší než prostor t a které všechny leží na stejné straně nadroviny t v s. Výstupem je těleso, které je konvexním obalem vstupu a t je jedna z jeho faset. Pokud již toto těleso existovalo, vrátí se přímo. (Kvůli implementačním detailům funkce přijímá jako další parametr také prostor s.)

Funkce `Complete` dostane strukturu tělesa s chybějícími fasetami a seznam bodů, jejichž konvexním obalem má být těleso, a doplní je. Struktura musí obsahovat alespoň jednu fasetu.

Všechny funkce pracují v libovolném afinním podprostoru.

1.4.1 Datová struktura tělesa

těleso t:
  seznam synů S_t
  seznam rodičů R_t
  seznam vrcholů V_t
  heš tělesa h_t
  afinní prostor p_t:
    dimenze prostoru $d_{p_t} = d_t$
    souřadnice libovolného obsaženého bodu (obvykle vnitřního bodu t)
    ortonormální báze
    normálový vektor $n_{p_t} = n_t$ v nadprostoru některého rodiče
      včetně vzdálenosti od počátku v jeho směru
  verzované značky pro různé účely

Při úpravách hranice tělesa se vždy aktualizují současně příslušné seznamy synů a rodičů. Seznam vrcholů nemusí být během úpravy tělesa aktuální.

Normálový vektor se vztahuje vždy k právě zpracovávanému rodiči a jeho prostoru, není platný stále. Stejně tak orientace normálového vektoru má pouze lokální význam, v uvedených popisech algoritmů na její volbě nezáleží.

Heš vrcholu je náhodné číslo, heš jiného tělesa je xor jeho vrcholů.

Verzované značky fungují takto: Každá značka má vždy přiřazené číslo (příp. objekt) určující její aktuální verzi. Každé těleso má pro každou značku přiřazeno číslo určující poslední verzi značky, kterou bylo označeno. Těleso je označeno právě tehdy, shoduje-li se verze jeho značky s aktuální verzí. Inkrementací aktuální verze (příp. vytvořením nového objektu) tak zrušíme značky všech těles v konstantním čase. Například při rekurzivním průchodu tělesem se označují navštívené stěny a každá se tak navštíví pouze jednou, celý průchod pak má lineární časovou složitost vzhledem k počtu hran grafu svazu stěn tělesa.

1.4.2 Vytvoření tělesa (`CreateHull`)

V prostoru p generovaném vstupním seznamem bodů najdeme tečnou nadrovinu r k výslednému tělesu (prochází aspoň jedním bodem a všechny ostatní leží na stejné straně). Z vrcholů ležících v této nadrovině rekurzivně sestrojíme těleso, které bude tvořit stěnu výsledného tělesa.

Dokud je dimenze sestrojené stěny menší, než by měla faseta, rozšiřujeme ji: Najdeme o jednu dimenzi vyšší prostor q (podprostor jiné tečné nadroviny),

v němž bude ležet stěna, která bude mít předchozí za fasetu. Vytvoříme novou stěnu rozšířením předchozí pomocí `ExpandDim`.

Jakmile máme sestrojenou první fasetu hledaného tělesa, rozšíříme ji všemi zadanými body pomocí `ExpandDim` do prostoru p.

Podrobnější popis poskytne následující pseudokód:

```
vstup: seznam bodů V
1   p ← prostor generovaný body V
2   pokud d_p = 0: t ← první prvek V, konec
3   r ← prostor vzniklý z p odebráním jednoho vektoru báze (nadrovina v p)
4     n_r ← odebraný vektor (normálový vektor nadroviny r v prostoru p)
5     umístit prostor do prvního bodu V ve směru n_r
6   V' ← V ∩ r
7   t ← CreateHull(V')
8   dokud d_t < d_r:
9     pro každý bod v ∈ V \ p_t:
10      q ← rozšíření prostoru p_t bodem v
11      n_t ← normálový vektor p_t v q
12    (v, q, n_t) ← trojice z předchozího cyklu s minimálním součinem |⟨n_r, n_t⟩|
13    V' ← V ∩ q
14    t ← ExpandDim(t, V', q), viz sekce 1.4.3
15  t ← ExpandDim(t, V, p), viz sekce 1.4.3
výstup: těleso t
```

Asi hlavním bodem, který potřebuje komentář, je nalezení prostoru q, v němž bude ležet stěna vyšší dimenze než t, na řádku 12. Tento prostor musí obsahovat stěnu t a některý další vrchol ze vstupního seznamu, který způsobí zvýšení dimenze prostoru. To splňují právě všechny prostory q, které se v cyklu vyzkouší. Další podmínkou je, že q musí být součástí některé tečné nadroviny tělesa, což by měla zajistit minimalizace skalárního součinu $|\langle n_r, n_t\rangle|$; dvourozměrný příklad je znázorněn na Obrázku 1.5, v následujících odstavcích se to pokusíme osvětlit obecně.

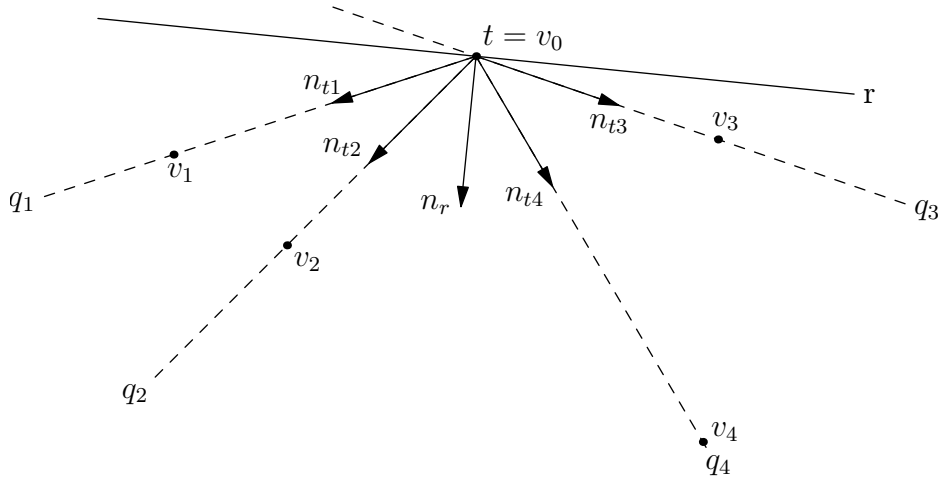
Označme t výsledné těleso (jako na konci pseudokódu) a definujme posloupnost tečných nadrovin $r = r_0, r_1, \ldots, r_k$ k t určujících stěny $t_i = t \cap r_i$ tělesa t tak, že $t_0 \subsetneq t_1 \subsetneq \cdots \subsetneq t_k$, t_k je faseta tělesa t a těleso t_{i+1} je vždy o jednu dimenzi větší než t_i. Nechť dále každá nadrovina r_{i+1} svírá s předchozí r_i nejmenší možný úhel. Označme $q_i \subset r_i$ prostor generovaný tělesem t_i (posloupnost q_i bude odpovídat hodnotám q v průběhu algoritmu a t_i příslušným stěnám).

Proces hledání tečné nadroviny obsahující fasetu tělesa si můžeme představit takto: Začneme s libovolnou tečnou nadrovinou $r = r_0$ obsahující stěnu t_0. Nadrovinu budeme otáčet kolem nějakého nadprostoru q_i tak, aby se dotkla dalšího vrcholu tělesa a v průniku vytvořila stěnu vyšší dimenze. To splňuje například nadrovina r_1. Stejným postupem pokračujeme.

Vektorový prostor o_i, který je ortogonálním doplňkem prostoru q_i v nadrovině r_i, je vždy součástí vektorového prostoru nadroviny r_{i-1} (jinak by otočení nebylo minimální), a tedy i všech předchozích včetně r. Skalární součin libovolného vektoru z tohoto ortogonálního doplňku s normálovým vektorem původní nadroviny r je tak nulový.

Normálový vektor n_i prostoru q_i v prostoru q_{i+1} naproti tomu do žádné z předchozích nadrovin $r_0, \ldots, r_i$ nepatřil a jeho skalární součin s normálovým vektorem nadroviny r je tak nenulový. Vzor vektoru n_i v nadrovině r_i před rotací je součástí o_i. V průběhu rotace se tedy skalární součin spojitě zvětšoval a jeho minimalizací bychom měli dosáhnout minimální rotace nadroviny.

Vytvoření prostoru p na 1. řádku má časovou složitost $\mathcal{O}(nd^2)$, skalární součin n_r se všemi vrcholy na 5. řádku $\mathcal{O}(n \cdot d)$ a filtrování vrcholů na 6. řádku $\mathcal{O}(n \cdot d^2)$. V následném cyklu je po `ExpandDim` nejdražší operací opět filtrování vrcholů a rozšiřování báze v čase $\mathcal{O}(n \cdot d^2)$. Cyklus se přes všechna rekurzivní volání opakuje maximálně d-krát, takže časová složitost všech volání `CreateHull` (bez `ExpandDim`) je $\mathcal{O}(nd^3)$, kde n je počet vrcholů a d počet souřadnic.



Obrázek 1.5: Rovinný příklad hledání prostoru q pro rozšíření tělesa t ve funkci `ConvexHull`. Nularozměrné těleso t je tvořeno bodem v_0, který byl první ve směru normálového vektoru n_r zvolené nadroviny r. Hledáme prostor q_i, který obsahuje t a některý vrchol v_i a je součástí některé tečné nadroviny výsledného tělesa (zde celou nadrovinou). Minimalizací skalárního součinu $\langle n_r, n_{ti} \rangle$ dostáváme prostor q_3.

1.4.3 Rozšíření tělesa (`ExpandDim`)

Pokud již těleso existuje v globální hešovací tabulce, vrátíme jej; jinak vytvoříme novou strukturu s jednou známou fasetou a ostatní doplníme pomocí `Complete`.

vstup: těleso t, seznam bodů V, afinní prostor p nového tělesa
1 vypočítáme možný heš h výsledného tělesa:
2 $h \leftarrow 0$
3 pro každý vrchol $v \in V_t \cup V$:
4 pokud $R_v \neq \emptyset$:
5 $h \leftarrow h \oplus h_v$
6 hledáme v hešovací tabulce d_p-rozměrné těleso t' s hešem h a vrcholy $V_t \cup V$
7 pokud těleso existuje: konec
8 vytvoříme nové těleso t':
9 $p_{t'} \leftarrow p$
10 jako vnitřní bod $p_{t'}$ nastavíme průměr bodů V
11 přidáme t do hranice t'
12 `Complete`(t', V), viz sekce 1.4.4
výstup: těleso t'

Heš tělesa je vždy xor všech vrcholů, které má obsaženy v hranici, což nemusí být všechny zadané body. Všechny vrcholy ale musí mít nějakého rodiče a naopak vnitřní bod libovolné stěny se nikdy nestane vrcholem. Pro výpočet heše proto na řádku 4 vybíráme pouze ty body, které mají neprázdný seznam rodičů; existuje-li těleso, bude mít v hranici právě tyto body. Naproti tomu, pokud těleso ještě neexistuje, mohli jsme některý vrchol opomenout, což nám ale v takovém případě nevadí.

Pro porovnávání seznamů vrcholů v případě nalezení heše v tabulce (řádek 6) používáme verzované značky. Všechny vrcholy jednoho seznamu spočítáme a označíme, druhý seznam pak musí být stejně dlouhý a všechny vrcholy musí být označené.

Změna bodu afinního prostoru na řádku 10 prostor neposune, bod může být ale později potřeba ve funkci `Complete` při zpracování nadřazeného tělesa.

Výpočet heše je lineární v počtu vrcholů, hledání v hešovací tabulce průměrně konstantní s lineárním porovnáním seznamů vrcholů. Průměrnou složitostí zde myslíme horní odhad složitosti nejhoršího vstupu při zprůměrovaném chování heše. Výpočet průměru bodů na řádku 10 je $\mathcal{O}(n \cdot d)$.

Pokud těleso již existovalo, je časová složitost $\mathcal{O}(n)$. Bylo-li těleso potřeba vytvořit, časová složitost bez započítání volání `Complete` je $\mathcal{O}(nd)$. Parametr n označuje počet bodů na vstupu (vč. vrcholů v hranici tělesa t) a d počet souřadnic.

1.4.4 Doplnění chybějících stěn tělesa (`Complete`)

Dokud existuje hřeben, který je součástí pouze jedné fasety, podle Lemmatu 2 některé fasety musí chybět. Druhou fasetu k nalezenému hřebenu vytvoříme tak, že prostor hřebenu rozšíříme libovolným vrcholem a se vzniklou nadrovinou budeme otáčet kolem hřebenu (vyměňováním vrcholu), dokud nebudou všechny vrcholy na jedné její straně; pak z hřebenu a vrcholů v nadrovině vytvoříme fasetu pomocí `ExpandDim`.

Nadrovinou vždy otáčíme směrem k vrcholu, který je na její opačné straně než existující faseta obsahující hřeben.

vstup: nedokončené těleso t s alespoň jednou fasetou, seznam bodů V

1 pokud $d_t = 1$:
2 přidáme do hranice tělesa nejvzdálenější bod, konec
3 označíme hřebeny, které mají jen jednoho rodiče mezi fasetami tělesa t
4 pro každý hřeben h ležící v právě jedné fasetě f tělesa t:
5 vytvoříme nový prostor p rozšířením p_h libovolným bodem $v \in V \setminus p_f$
6 $n_p \leftarrow$ normálový vektor p v p_t (nastavíme i posun)
7 pro každý vrchol $w \in V \setminus p_f$:
8 pokud w je na opačné straně nadroviny p_f než vnitřní bod f:
9 $v \leftarrow w$
10 $p \leftarrow$ rozšíření p_h vrcholem v
11 $n_p \leftarrow$ normálový vektor p v p_t (nastavíme i posun)
12 $V' = V \cap p$
13 $f' \leftarrow$ `ExpandDim`(h, V', p), viz sekce 1.4.3
14 přidáme f' do hranice t a aktualizujeme značky hřebenů
15 aktualizujeme seznam vrcholů V_t a heš h_t

Hřebeny označujeme dvěma vylučujícími se značkami (původně nemají žádnou), které určují, zda se nachází v hranici jedné nebo dvou faset. Na řádku 3 procházíme všechny fasety a jejich syny a u každého značkou zvyšujeme zjištěný počet jeho rodičů v tělese t. Při aktualizaci značek hřebenů na řádku 14 stačí upravit pouze značky v nové fasetě. Použití stejných značek v rekurzivním volání je možné, protože neupravuje hřebeny tělesa t. Z důvodu nepřesné aritmetiky (viz sekce 1.3.1) může být jeden hřeben navštíven i více než dvakrát, v takovém případě výpočet s chybou ukončíme.

V cyklu na řádku 4 ve skutečnosti odebíráme fasety z fronty, do níž jsme je předem přidali, a procházíme jejich syny (podle značky rozhodneme, které přeskočit). Na řádku 14 pak novou fasetu přidáme také do fronty. Není důležité pořadí zpracování faset, ale aby byla každá zpracována právě jednou.

Označení hřebenů na řádku 3 a jejich aktualizace na řádku 14 mají složitost $\mathcal{O}(f + h)$, kde f je počet faset (vč. nově vytvořených) a h počet hřebenů (každý hřeben se zpracuje maximálně dvakrát). Cyklus na řádku 4 se provede pro každou nově přidanou fasetu jednou. Procházení vrcholů, rozšiřování prostorů a filtrování vrcholů má za všechny fasety složitost $\mathcal{O}(f \cdot nd^2)$, kde n je počet vstupních bodů. Aktualizace seznamu vrcholů již složitost neovlivní. Celková časová složitost bez volání `ExpandDim` je $\mathcal{O}(h + fnd^2)$.

1.4.5 Celková časová složitost

Složitosti samostatných volání každé z funkcí:

`CreateHull`, viz sekce 1.4.2
  $\mathcal{O}(nd^3)$ za všechna volání
`ExpandDim`, viz sekce 1.4.3
  $\mathcal{O}(n)$, pokud těleso již existovalo
  $\mathcal{O}(nd)$ při vytváření tělesa
`Complete`, viz sekce 1.4.4
  $\mathcal{O}(h + fnd^2)$

Funkce `Complete` a `ExpandDim` (při vytváření tělesa) se zavolají jednou za každou stěnu tělesa. Funkce `ExpandDim` se celkem (při započítání vracení existujících těles) zavolá jednou za každou hranu grafu svazu stěn tělesa, jejich počet označme m.

Složitost všech volání `ExpandDim` je $\mathcal{O}(m \cdot n + s \cdot nd)$, kde s je celkový počet stěn. U složitosti `Complete` se počet hřebenů všech stěn i počet faset všech stěn sečte na $\mathcal{O}(m)$, celkem je tedy složitost všech volání `Complete` $\mathcal{O}(mnd^2)$.

Časová složitost hledání konvexního obalu je

$$\mathcal{O}(nd^3 + (mn + snd) + mnd^2) = \mathcal{O}(mnd^2 + nd^3), \tag{1.1}$$

kde m je počet hran grafu svazu stěn výsledného tělesa, n počet vstupních bodů a d počet souřadnic (dimenze prostoru). Jde o složitost při nejhorším vstupu s uvážením průměrného chování hešování v `ExpandDim`.

Složitost závislá pouze na velikosti vstupu

V učebnici prof. Matouška (Matoušek, 2002, Tvrzení 5.5.2) jsou dokázány následující odhady na počet faset a všech stěn tělesa:

$$f = \mathcal{O}(n^{\lfloor d/2 \rfloor}) \tag{1.2}$$

$$s = \mathcal{O}(n^{\lfloor d/2 \rfloor} \cdot 2^{d+1}) \tag{1.3}$$

Velikost grafu svazu stěn odhadneme napřed pro případ simpliciálních těles, tedy takových, jejichž každá stěna je simplex. Pro takováto tělesa je při konstantním počtu rozměrů počet hran zmíněného grafu asymptoticky stejný jako počet faset, protože graf simplexu je počtem rozměrů jednoznačně určen.

Pro obecné mnohostěny použijeme tzv. perturbační argument (podobně jako při dokazování počtu stěn ve výše zmíněné učebnici).

Věta 3. *Ke každému mnohostěnu existuje simpliciální mnohostěn, který má stejný počet vrcholů a stejný nebo větší počet hran v grafu svazu stěn.*

Důkaz. Vycházíme z obecného mnohostěnu, postupně budeme posouvat jeho vrcholy maximálně o ε, abychom je dostali do obecné polohy a výsledné simpliciální těleso nemělo více hran v grafu, než původní.

Každý vrchol posuneme maximálně o $\varepsilon \in \mathbb{R}^+$ tak, aby neležel v žádné nadrovině určené ostatními vrcholy tělesa. Tím dostaneme vrcholy do obecné polohy a jejich konvexní obal tak bude simpliciální těleso.

Konstanta ε může být libovolně malá, nastavíme ji tak, aby se žádný vrchol nemohl stát vnitřním bodem tělesa. (Musí být menší, než polovina vzdálenosti libovolného vrcholu od konvexního obalu zbývajících vrcholů původního tělesa.)

Při takovémto posunu vrcholu se některé stěny můžou rozdělit na dvě části oddělené novou stěnou, obě části ale budou mít stejné rodiče, jako měl jejich vzor. Původní fasety rozdělené stěny budou mít za rodiče jednu z nových stěn (pokud se také nerozdělily), ale počet zůstane stejný. Sledovaný počet hran grafu se tedy může pouze zvětšit.

□

Pro libovolný mnohostěn tedy platí při konstantním d následující odhad:

$$m = \mathcal{O}(f) = \mathcal{O}(n^{\lfloor d/2 \rfloor}). \tag{1.4}$$

Časová složitost algoritmu při konstantním počtu rozměrů (z (1.1)) pak bude

$$\mathcal{O}(mnd^2 + nd^3) = \mathcal{O}(n^{\lfloor d/2 \rfloor + 1}).$$

Porovnání složitosti s existujícími algoritmy

Optimální deterministický algoritmus pro hledání konvexního obalu v libovolném pevném počtu rozměrů je popsán v článku Chazelle (1993) s uváděnou časovou složitostí

$$\mathcal{O}(n \log n + n^{\lfloor d/2 \rfloor}).$$

Zde popsaný názorný algoritmus je tedy při nejhorším vstupu (a průměrném chování hešování) asymptoticky maximálně n-krát pomalejší než optimální.

1.5 Duální mnohostěny (`CreateDual`)

Geometrická dualita je popsána ve skriptech Matoušek (2002, kapitola 5.1).

Definice 7 (geometrická dualita)**.** *K nadrovině* $\{x \in \mathbb{R}^d \mid \langle a, x \rangle = 1\}$ *je duální bod* $a \in \mathbb{R}^d \setminus 0$*, duální nadrovinou k bodu* a *je původní nadrovina.*

Konvexní mnohostěn obsahující počátek zadaný množinou vrcholů V *je duální k mnohostěnu zadanému jako průnik poloprostorů*

$$\bigcap_{a \in V} \{x \in \mathbb{R}^d \mid \langle a, x \rangle \leq 1\}$$

určených duálními nadrovinami k vrcholům.

Ve skriptech je také ukázáno, že svazy stěn původního a duálního tělesa jsou až na směr uspořádání isomorfní.

Pro vytvoření duálního tělesa tedy stačí najít duální bod (vrchol) k nadrovině každé fasety a zbytek tělesa vystavět převrácením relace rodič–syn. Těleso t je zde tvořeno pouze svou dimenzí d_t a seznamem synů S_t, v průběhu algoritmu ke stěnám s vytvoříme také seznamy jejich vrcholů V_s a odpovídající stěny duálu t_s.

vstup: konvexní těleso t a jeho vnitřní bod o

1 pro každou stěnu s tělesa t, vždy nejdříve potomky:
2 pokud $d_s = 0$: $V_s \leftarrow \{s\}$, jinak: $V_s \leftarrow \bigcup_{s' \in S_s} V_{s'}$
3 vytvoříme nové těleso t_s dimenze $d_t - d_s - 1$
4 pro každou fasetu $f \in S_t$:
5 $p \leftarrow$ nadrovina vypočítaná z vrcholů V_f
6 $n \leftarrow$ normalizovaný normálový vektor nadroviny p
7 $a \leftarrow$ vzdálenost nadroviny od o ve směru n
8 umístíme duální vrchol t_f do bodu $n/a + o$
9 pro každou stěnu s tělesa t:
10 pro syny $f \in S_s$:
11 přidáme do hranice t_f fasetu t_s
12 pokud je s vrchol:
13 přidáme t_s do hranice tělesa t'

V algoritmu je explicitně určen vnitřní bod o tělesa, což je ekvivalentní s posunutím tělesa tak, aby o byl v počátku a poté posunutím duálního tělesa zpět.

Průchod na řádku 1 můžeme realizovat procházením do hloubky se zpracováním vrcholů při jejich opouštění. Sjednocení seznamů vrcholů můžeme provést jejich průchodem s označováním.

Propagování seznamů vrcholů do faset na řádku 2 bude stát celkem $\mathcal{O}(mn)$, kde m je velikost stromu svazu stěn a n počet vrcholů. Výpočet nadroviny fasety má složitost $\mathcal{O}(nd^2)$.

Časová složitost při konstantním počtu rozměrů je

$$\mathcal{O}(fnd^2 + mn) = \mathcal{O}(mn), \tag{1.5}$$

kde f je počet faset, d počet souřadnic, m počet hran grafu svazu stěn a n počet vrcholů.

Alternativně jsme mohli seznamy vrcholů faset zjišťovat průchodem tělesa, až když byly potřeba, což by vedlo k celkové složitosti $\mathcal{O}(mf)$. Ta se nám ale pro použití v dalších algoritmech nehodí.

1.6 Průniky poloprostorů (`CreateBounded`)

Duál jakéhokoliv tělesa je vždy konvexní a obsahuje-li konvexní těleso t počátek, je duálem jeho duálu opět původní těleso t (Matoušek, 2002, str. 80).

Těleso zadané jako průnik poloprostorů sestrojíme tak, že najdeme vrcholy duálního tělesa, sestrojíme jeho konvexní obal a k němu vytvoříme duální těleso.

vstup: seznam nadrovin P ohraničujících těleso, vnitřní bod o tělesa
1 $V \leftarrow$ duální vrcholy k nadrovinám v P (s počátkem o), viz sekce 1.5
2 $t \leftarrow$ `CreateHull`(V), viz sekce 1.4.2
3 $t' \leftarrow$ `CreateDual`(t, o), viz sekce 1.5
výstup: těleso t'

Nadroviny mohou být tentokrát tvořeny pouze svým normálovým vektorem a posunutím, vytvoření duálních bodů pak bude mít složitost $\mathcal{O}(nd)$, pro n počet

nadrovin. Vytvoření konvexního obalu bude mít složitost $\mathcal{O}(n^{\lfloor d/2 \rfloor+1})$ a vytvoření duálu $\mathcal{O}(mn)$, kde m můžeme opět odhadnout pomocí $n^{\lfloor d/2 \rfloor}$ (viz odhad (1.4)).

Časová složitost pro konstantní počet rozměrů tedy bude

$$\mathcal{O}(n^{\lfloor d/2 \rfloor+1}), \tag{1.6}$$

kde n je počet poloprostorů. Opět uvažujeme průměrné chování hešování.

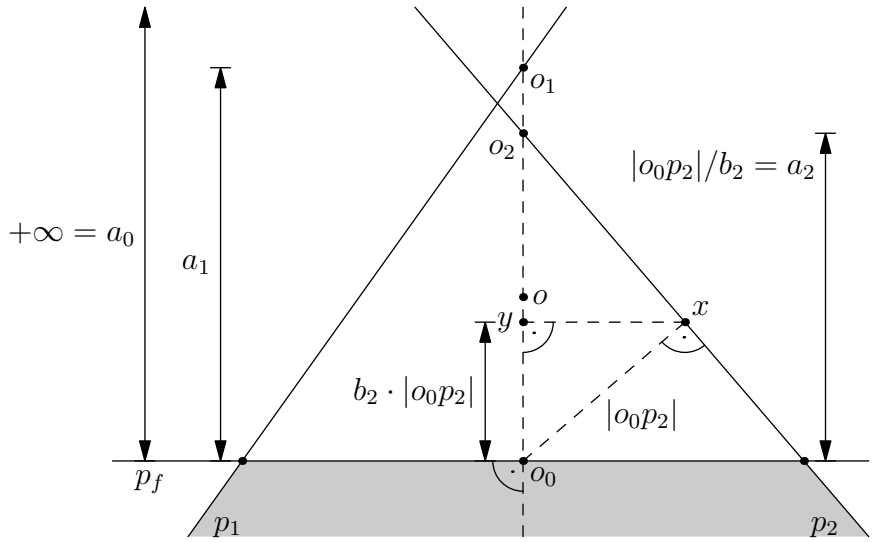
1.7 Stelace (`Stellate`)

Pro jednoznačnost zde stelaci definujme následovně:

Definice 8 (stelace)**.** *Stelace (zhvězdnatění) je operace, která vytvoří z konvexního tělesa t těleso t' takto: Pro každou fasetu f_i vytvoříme těleso t_i (hrot hvězdy) jako průnik poloprostoru neobsahujícího t určeného její nadrovinou s poloprostory obsahujícími t určenými nadrovinami sousedních stěn. Jsou-li všechna t_i omezená, výsledné těleso je sjednocením všech t_i a t, jinak stelace nemá pro t smysl.*

Nejdříve ke každé fasetě přiřadíme odpovídající nadrovinu, určenou normalizovaným normálovým vektorem a posunem a vytvoříme zpětné reference na otce hřebenů.

Pro každou fasetu najdeme vnitřní bod o nově vytvářeného tělesa t_f (hrotu) a těleso z průniku poloprostorů vytvoříme. Dále je potřeba unifikovat společné stěny nového a vstupního tělesa (vytvořením nového tělesa jsme získali duplicitní). Výsledné těleso t' pak budou tvořit právě všechny nové fasety (tedy žádná z původního tělesa v něm nebude obsažena).



Obrázek 1.6: Hledání vnitřního bodu hrotu při stelaci. Šedě je vyznačeno těleso t s fasetou f nahoře. Vzdálenosti a_i odpovídají hodnotám a v pseudokódu, přímky p_i nadrovinám p, body o_i průnikům s nimi. Bod o_0 je průměrem vrcholů fasety f. Výsledným vnitřním bodem o je střed úsečky o_0o_2.

```
vstup: konvexní těleso t
1  pro každý hřeben h tělesa t určíme jeho rodiče R_h
2  pro každou stěnu s tělesa t určíme její vrcholy V_s
3  pro každou fasetu f tělesa t:
4    P_f ← nadrovina vypočítaná z vrcholů
5      vč. normalizovaných normálových vektorů n_f orientovaných dovnitř t
6  t' ← nové těleso dimenze d_t
7  pro každou fasetu f tělesa t:
8    F ← ⋃_{h∈S_f} R_h \ f (sousední fasety)
9    P ← {P_{f'} | f' ∈ F}
10   najdeme vnitřní bod o budoucího tělesa t_f:
11     o_0 ← průměr vrcholů V_f
12     a ← +∞
13     pro každou nadrovinu p ∈ P:
14       b ← ⟨n_p, n_f⟩
15       pokud je b > 0:
16         a' ← (vzdálenost o od p)/b
17         pokud je a' < a: a ← a'
18     pokud a = +∞: končíme s chybou (t_f by bylo neomezené)
19     o ← o_0 − a/2 · n_f
20   t_f ← CreateBounded(P ∪ {p_f}, o), viz sekce 1.6
21   f' ← faseta tělesa t_f odpovídající nadrovině p_f
22   unifikujeme stěny t_f s existujícími stěnami t:
23     pro každou fasetu s tělesa t_f:
24       s_s ← faseta původního tělesa t odpovídající nadrovině p_s
25     pro každou stěnu s fasety f' v pořadí od nejvyšší dimenze:
26       (r_1, r_2) ← libovolní dva rodiče z R_s
27       s_s ← S_{s_{r_1}} ∩ S_{s_{r_2}}
28     procházíme stěny s tělesa t_f v pořadí od nejvyšší dimenze:
29       pro každou fasetu s' ∈ S_s:
30         pokud existuje s_{s'}:
31           nahradíme s' za s_{s'} v S_s
32           v cyklu na řádku 28 vynecháme s' a jeho stěny
33   přidáme fasety S_{t_f} \ f' do tělesa t'
výstup: těleso t'
```

Vnitřním bodem o (řádek 19) je vždy střed úsečky spojující dva body na hranici hrotu t_f. Prvním z bodů je průměr o_0 vrcholů fasety f (řádek 11), z něj vztyčíme kolmici k p_f a její průnik s nejbližší nadrovinou p (směrem od t) bude druhým bodem. Proměnná a představuje aktuální nejmenší vzdálenost průniku kolmice s některou nadrovinou od bodu o_0. Postup je znázorněný na Obrázku 1.6:

Vzdálenosti a_0, a_1, a_2 odpovídají postupně přiřazovaným hodnotám a, číslo b_2 příslušnému skalárnímu součinu b, přímky p_1, p_2 nadrovinám p a body o_1, o_2 průnikům s nadrovinami. Nadrovina p_f stěny f má průměr vrcholů v o_0. Řádek 16, tedy že vzdálenost $|o_0o_2| = |o_0p_2|/b_2 = a_2$ plyne z podobnosti trojúhelníků o_0xo_2 a o_0yx. Podmínka na řádku 15 je splněna právě tehdy, když se průnik s nadrovinou nachází na správně straně p_f.

K unifikaci hřebenů využíváme faktu, že k nově vytvořeným fasetám známe odpovídající původní struktury nadrovin a k nim fasety původního tělesa. To použité algoritmy z předchozích stran v uvedené podobě neumožňují, ale přidání této vlastnosti je jen drobná úprava, která přinese konstantní zpomalení (pro zjednodušení implementace bylo použito hešování, uvedené zpomalení tedy uvažujeme při průměrném chování heše).

Procházení stěn v pořadí od nejvyšší dimenze na řádcích 25 a 28 lze implementovat jako průchod (grafu svazu stěn) do šířky, třídění není potřeba.

Předpočítání vrcholů stěn a nadrovin faset na řádcích 2–5 trvá $\mathcal{O}(mn+fnd^2)$, kde m je počet hran grafu tělesa, n počet vrcholů a f počet faset (stejně jako u složitosti (1.5) hledání duálu). Řádky 8 a 9 zaberou celkem $\mathcal{O}(h)$, protože každý hřeben má jen dva rodiče (viz Lemma 2).

Při hledání vnitřního bodu o průměrování vrcholů (řádek 11) trvá $\mathcal{O}(nd)$ a cyklus na řádku 13 se provede dvakrát za každý hřeben v průběhu celého běhu algoritmu (stejné lemma). Hledání všech vnitřních bodů má tedy složitost $\mathcal{O}(fnd + hd)$.

Vytvoření tělesa ohraničeného nadrovinami (řádek 20) zabere $\mathcal{O}(k^{\lfloor d/2 \rfloor+1})$, kde k je maximální počet synů/sousedů fasety (viz složitost (1.6)). Celkem tedy $\mathcal{O}(fk^{\lfloor d/2 \rfloor+1})$.

Na řádcích 25–27 pro každou stěnu tělesa t_f hrotu hvězdy procházíme seznamy faset dvou rodičů. Označíme-li s' maximální počet stěn hrotu, bude složitost

$$\mathcal{O}(s'^2) = \mathcal{O}(k^d),$$

s využitím odhadu (1.3). Na řádcích 23 a 28 jsou jen lineární průchody dříve vytvořeného tělesa hrotu, které složitost neovlivní. Celková složitost unifikace tedy bude $\mathcal{O}(fk^d)$.

Složitost stelace pro pevný počet rozměrů a s uvážením průměrného chování hešování je

$$\mathcal{O}(fnd + mn + fk^d) = \mathcal{O}(mn + fk^d),$$

kde f je počet faset vstupního tělesa, n počet vrcholů, m velikost grafu svazu stěn a k maximální počet synů/sousedů fasety.

Stelaci bychom mohli zrychlit úpravou předchozích algoritmů tak, aby se použily všechny již existující části hrotu (všechny stěny příslušné fasety původního tělesa). Namísto pouhého převodu nadrovin na duální vrcholy bychom duální těleso doplnily i o všechny již známé stěny, tyto stěny bychom beze změny využili při hledání konvexního obalu a poté převedli zpět na stěny duálního tělesa (hrotu). Díky tomu by nebyla potřeba unifikace a časová složitost by se snížila na $\mathcal{O}(mn + fk^{\lfloor d/2 \rfloor+1})$. Reálná rychlost stelace v aplikaci je ale pro běžná využití dostačující.

1.8 Řez tělesa nadrovinou (`CutFigure`)

Zde popíšeme funkci pro řezání tělesa (i nekonvexního) nadrovinou. Jejím vstupem je těleso $\hat{t}$ a nadrovina p, kterou budeme řezat. Těleso zde opět bude tvořeno jen jeho dimenzí $d_{\hat{t}}$ a seznamem synů $S_{\hat{t}}$, nadrovina pak normálovým vektorem a posunem.

Výstup tvoří tři seznamy těles $L_{\hat{t}}$, $O_{\hat{t}}$ a $P_{\hat{t}}$. Seznamy $L_{\hat{t}}$ (levý) a $P_{\hat{t}}$ (pravý) obsahují tělesa vzniklá na průniku $t_{\hat{t}}$ s poloprostory určenými nadrovinou p a seznam $O_{\hat{t}}$ obsahuje tělesa na průniku $\hat{t} \cap p$. V případě konvexního tělesa bude každý seznam obsahovat maximálně jeden prvek. V seznamech $L_{\hat{t}}$ a $P_{\hat{t}}$ budou pouze ta tělesa, která mají neprázdný průnik s příslušnými otevřenými poloprostory; všechna tedy budou stejné dimenze jako vstupní těleso, což zjednoduší následný rozbor případů. V seznamu $O_{\hat{t}}$ se naproti tomu mohou objevit tělesa libovolné nižší dimenze.

Seznamy těles L_t, O_t, P_t budeme vytvářet postupně pro všechny stěny t tělesa $\hat{t}$ od nejnižších dimenzí. U bodu pouze určíme, do kterého seznamu patří. Hranu v případě různých poloprostorů vrcholů rozdělíme na dvě s novým vrcholem na řezu. U stěn vyšších dimenzí vytvoříme seznamy L_t, P_t těles z již sestrojených seznamů jejich faset; poté podle příslušnosti hřebenů v p k těmto fasetám sestrojíme fasety v p (jako společné podrozdělení příslušných faset těles v obou poloprostorech).

Podrobnější rozbor případů je zachycen v pseudokódu na následující straně.

vstup: těleso $\hat{t}$, nadrovina p

pro každou stěnu t tělesa $\hat{t}$ včetně $\hat{t}$ od nejnižší dimenze:
  $L_t, O_t, P_t, L, O, P \leftarrow \emptyset$
  pokud $d_t = 0$:
    $a \leftarrow$ orientovaná vzdálenost vrcholu t od p
    pokud $a = 0$: $O_t \leftarrow \{t\}$, další t (konec cyklu)
    pokud $a > 0$: $P_t \leftarrow \{t\}$, další t (konec cyklu)
    pokud $a < 0$: $L_t \leftarrow \{t\}$, další t (konec cyklu)
  $L \leftarrow \bigcup_{f \in S_t} L_f$, $O \leftarrow \bigcup_{f \in S_t} O_f$, $P \leftarrow \bigcup_{f \in S_t} P_f$
  pokud $L = P = \emptyset$: $O_t \leftarrow \{t\}$, další t (konec cyklu)
  $O_t \leftarrow$ celá tělesa na řezu z O, $O \leftarrow$ možné budoucí fasety nových z O:
    pro každé těleso $s \in O$:
      pokud $d_s \neq d_t - 2$:
        odebereme s z O
        pokud s neleží v hranici žádného tělesa z lib. existujícího O_*:
          přidáme s do O_t
  pokud $P = \emptyset$: $L_t \leftarrow \{t\}$, $O_t \leftarrow O_t \cup O$, další t (konec cyklu)
  pokud $L = \emptyset$: $P_t \leftarrow \{t\}$, $O_t \leftarrow O_t \cup O$, další t (konec cyklu)
  odebereme z O_t tělesa dimenze $d_t - 1$ (později by vznikla duplicitně)
  pokud $d_t = 1$:
    $v \leftarrow$ bod na spojnici dvou vrcholů v S_t ležící v p
    $t_L \leftarrow$ nová hrana s vrcholy $S_{t_L} \leftarrow L \cup \{v\}$
    $t_P \leftarrow$ nová hrana s vrcholy $S_{t_P} \leftarrow P \cup \{v\}$
    $L_t \leftarrow \{t_L\}$, $P_t \leftarrow \{t_P\}$, $O_t \leftarrow \{v\}$, další t (konec cyklu)
  pomocí Union-Find rozdělíme L, P na množiny faset samostatných těles:
    pro množiny $M \in \{L, P\}$ faset:
      prvky Union-Find tvoří fasety a hřebeny otevřeného poloprostoru
      pro všechny fasety $f \in M$:
        pro všechny hřebeny $h \in S_f \setminus O$:
          `Union`(f, h)
      pro všechny fasety $f \in M$:
        přidáme f do množiny jejího reprezentanta `Find`(f)
      množiny reprezentantů obsahují fasety samostatných těles
  $L_t, P_t \leftarrow$ vytvoříme tělesa (většinou) s chybějícími fasetami na řezu
  k tělesům v L_t a P_t vytvoříme fasety na řezu a přidáme je k nim a do O_t:
    pro každé těleso $t' \in L_t \cup P_t$:
      $H_{t'} \leftarrow \bigcup_{f \in S_{t'}} S_f \cap O$ (hřebeny nových těles na řezu)
    pro každé těleso $t_L \in L_t$:
      pro každé těleso $t_P \in P_t$:
        $H \leftarrow H_{t_L} \cap H_{t_P}$
        pokud $H \neq \emptyset$:
          vytvoříme novou fasetu f s hranicí $S_f \leftarrow H$
          přidáme f do O, S_{t_L} a do S_{t_P}
  nepoužitá tělesa z O přidáme do O_t

výstup: seznamy stěn $L_{\hat{t}}, O_{\hat{t}}, P_{\hat{t}}$

Jako struktura Union-Find je použita verze union-by-rank s kompresí cest a amortizovanou časovou složitostí $\mathcal{O}(\alpha(n))$ na operaci, kde α je inverzní Ackermannova funkce a n počet operací `Union`/`Find`. Analýzu složitosti je možné nalézt v prezentaci prof. Seidela (viz Seidel, 8. 5. 2016), příp. ve článku Tarjan a van Leeuwen (1984).

Podmínku na řádku 14 můžeme vyhodnotit v konstantním čase pomocí značky u tělesa s, kterou lze udržovat bez asymptotického zpomalení (stejnou značku pak využijeme i na řádku 43). Pokud je p tečná nadrovina, na řádku 10 pouze odebereme z řezu tělesa, která jsou již součástí jiných, a na řádku 16 nebo 17 skončíme. Pokud je p sečná nadrovina, použijeme rozdělení na hotová tělesa a budoucí fasety nových těles řezu; fasety původního tělesa t navíc z řezu na řádku 18 odebereme, abychom je později nevytvářeli duplicitní.

U každé stěny nových těles můžeme udržovat seznam jejích faset, které leží v p. Na řádku 36 pak stačí projít pouze tyto seznamy namísto seznamů všech synů.

Pro analýzu složitosti zaveďme nový nenulový parametr k, kterým budeme asymptoticky shora omezovat součin i součet počtů vzniklých těles v obou poloprostorech, tedy $k = |L_{\hat{t}}| \cdot |P_{\hat{t}}| + 1$. Tímto parametrem budeme také odhadovat počet těles v každé z množin $L_{\hat{t}}, O_{\hat{t}}, P_{\hat{t}}$. Z každé stěny původního tělesa pak může vzniknout maximálně k nových stěn těles v poloprostorech a k stěn na řezu. Celkový počet vytvořených stěn pak bude $\mathcal{O}(ks)$, kde s je počet stěn původního tělesa, a celkový počet všech hran grafů svazů stěn nových těles je $\mathcal{O}(km)$, kde m je počet hran původního grafu.

Řádek 8 má za všechny průchody cyklem složitost $\mathcal{O}(km)$, na řádku 10 se jenom prochází právě vytvořený seznam.

`Union` na řádku 29 se provede maximálně jednou pro každou dvojici nové stěny a jejího hřebenu, tedy $\mathcal{O}(km)$-krát; `Find` se provádí jen pro stěny. Maximálně bude ve struktuře Union-Find $\mathcal{O}(ks)$ prvků (dle Lemmatu 2). Celková složitost Union-Find na řádku 24 tedy bude $\mathcal{O}(km \cdot \alpha(ks))$.

Cyklus na řádku 35 se provede maximálně k-krát. Řádek 36 má celkovou časovou složitost $\mathcal{O}(k^2 m)$.

Cyklus na řádku 38 se provede také maximálně k-krát pro každé těleso t. Na řádcích 41 a 42 se lineárně vytváří některá výstupní tělesa, mají tedy složitost $\mathcal{O}(km)$. Řádek 39 má složitost $\mathcal{O}(k^2 s)$.

Celková časová složitost řezu tělesa $\hat{t}$ v nejhorším případě je

$$\mathcal{O}(km \cdot (k + \alpha(ks))),$$

kde m je počet hran ve svazu stěn původního tělesa $\hat{t}$, s je počet stěn tělesa, k je obvykle malý (pro konvexní těleso $\hat{t}$ konstantní) parametr popsaný výše a α je inverzní Ackermannova funkce.

Mnohostěn a všechny jeho stěny musí mít souvislou hranici (jak je uvedeno v Definici 1). Na řezu ale může vzniknout také těleso, jehož hranice sestává ze dvou oddělených částí, kdy jedna je uvnitř druhé (v tělese je dutina). Tato situace v algoritmu zatím není detekována a je implicitně vyřešena vytvořením tělesa pro každou část hranice. Výstupem je tedy v takovémto případě z pohledu aplikace korektní množina mnohostěnů, která ale neodpovídá zadání.

2. Vývojová dokumentace

Dokumentace se vztahuje k aplikaci Geometric Figures verze 1.5, tuto i aktuální verzi zdrojových kódů naleznete na adrese

`https://github.com/ondracek-lukas/geometric-figures`.

Zde popisované zdrojové kódy jsou také přílohou elektronicky odevzdávané verze bakalářské práce jako archiv `geometric-figures.tar.gz`, jehož obsah je popsán v sekci 2.1.

Aplikace je napsána v jazyce C s využitím následujících knihoven:

OpenGL (Open Graphics Library) zajišťuje vykreslování 3D grafiky prostřednictvím základních nástrojů, jako např. vykreslování trojúhelníků v prostoru. Je-li to možné, využívá k tomu grafickou kartu počítače.

GLU (OpenGL Utility Library) je nadstavba nad OpenGL, která jej využívá k vykreslování složitějších útvarů. V aplikaci se využívá k vykreslování nekonvexních mnohoúhelníků a koulí.

FreeGLUT 3.0 komunikuje s grafickým rozhraním systému a poskytuje tak jednotné rozhraní pro vytvoření okna a obsluhu událostí na všech podporovaných systémech. Částečně knihovna také rozšiřuje funkce OpenGL, např. o možnost vykreslování znaků ASCII. Jde o alternativu s otevřeným zdrojovým kódem ke knihovně GLUT (OpenGL Utility Toolkit).

Python 2.7 umožňuje spouštění kódu jazyka Python, pomocí něhož jsou implementovány zásuvné moduly. Dále tento jazyk využíváme například v konfiguračním souboru nebo k ukládání vytvořených těles z rozhraní aplikace.

Všechny zmíněné knihovny jsou k dispozici pro Windows i Linux a samotný kód aplikace je díky nim až na několik funkcí v modulu `util` systémově nezávislý.

Kromě zmíněných jazyků mezi zdrojovými kódy nalezneme také skript psaný v Perlu (`scriptWrappers.pl`), který zpřístupňuje některé funkce jazyka C z Pythonu (viz sekce 2.4.2), a skript v jazyce Awk (`stringsData.awk`), který integruje texty nápovědy do výsledného spustitelného souboru (viz sekce 2.3.5). Ty jsou ale potřeba jen při sestavování aplikace.

K sestavení aplikace na Linuxu (vč. verze pro Windows) se využívá **GNU make** a přiložený soubor `Makefile`, více v sekci 3.1.

Aplikace je rozdělena do několika modulů, jejichž exportované symboly jsou prefixovány názvy těchto modulů. Úplný seznam modulů (a ostatních souborů) uvedeme v následující podkapitole (2.1), zde si je stručně představíme. Hlavní závislosti mezi moduly jsou znázorněny na Obrázku 2.1.

Jména modulů jsou někdy tvořená hierarchicky, např. modul `convex` využívá modul `convexHull`, který existuje právě za tímto účelem. Pro společný popis skupiny modulů se stejným prefixem budeme využívat značení tvořené příslušným prefixem a třemi tečkami (např. `convex`...).

Po spuštění modul `main` nastaví obslužné funkce událostí a zavolá hlavní smyčku programu, která je již součástí knihovny FreeGLUT.

Aktuálně otevřené těleso a jeho rotaci spravuje modul `figure`.

O vykreslování se stará modul `drawer`, který získává informace o tělese z modulu `figure` a informace o všech textech k vykreslení z modulu `console`.

Obsluhu událostí klávesnice a myši zajišťuje modul `hid`, který také spravuje aktuální přiřazení příkazů k jednotlivým událostem. K vykonávání těchto příkazů využívá modul `console`, příp. `anim` pro plynulou rotaci. V závislosti na stavu aplikace může také události předávat dále do modulů `console`, `convexInteract` nebo `anim`.

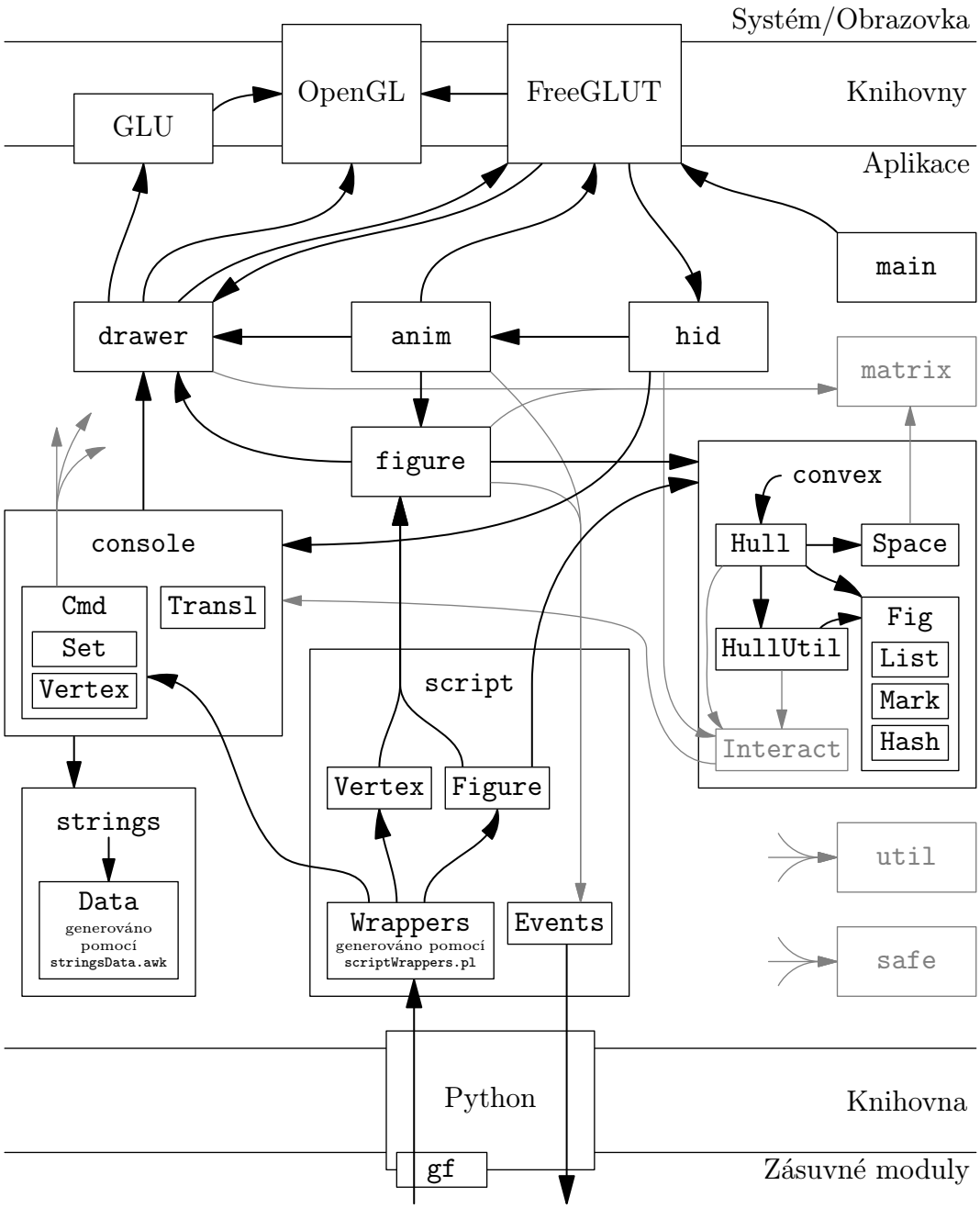
Příkazový řádek včetně vypisování stavových informací a bloků textu na střed obrazovky spravuje modul `console`. Zadané příkazy se pomocí `consoleTransl` přeloží na výrazy Pythonu, které se poté vykonají pomocí modulu `script`. Modul `consoleTransl` také zajišťuje automatické doplňování. Implementace některých příkazů konzole jsou (jako běžné funkce jazyka C) v modulech `consoleCmd`...

O veškerou interakci s Pythonem se starají moduly `script`... Funkce, které mají být přístupné z Pythonu, jsou buď implementovány v těchto modulech s použitím příslušných datových struktur Pythonu, nebo je k nim automaticky vygenerována obalující funkce skriptem `scriptWrappers.pl`. V Pythonu jsou poté přístupné ve jmenném prostoru modulu `gf`. Součást `scriptEvents` spravuje obsluhu některých událostí aplikace z modulů Pythonu.

Generování konvexního obalu probíhá v modulech `convex`... Tyto jsou využívány buď z Pythonu prostřednictvím `scriptFigure`, nebo z rozhraní aplikace pomocí modulu `figure`.

Modul `anim` zajišťuje veškeré časované akce: plynulou rotaci tělesa pomocí `figure`, uspání skriptu Pythonu s překreslováním obrazovky (pomocí funkce `gf.sleep`) a volání události `Idle` modulu `scriptEvents`.

Texty nápovědy jsou přístupné přes modul `strings`. Modul `matrix` zastřešuje základní prostředky lineární algebry, `safe` zapouzdřuje alokaci paměti a `util` obsahuje např. funkce pro exponenciální alokaci paměti řetězců a systémově závislé funkce.



Obrázek 2.1: Schéma hlavních závislostí mezi moduly aplikace a knihovnami. Šipka z A do B znamená, že modul A závisí na modulu B. Černé části jsou důležitější než šedé, ale jejich význam je stejný. U vnořených modulů je vynechávám jejich prefix tvořený názvem nadřazeného modulu.

2.1 Seznam souborů

V následujícím seznamu souborů nejsou uvedeny hlavičkové soubory (`*.h`), jejichž názvy jsou většinou shodné s názvy příslušných zdrojových souborů. V hlavičkových souborech je u každé funkce uveden její stručný popis.

`COPYING`	celý text GNU General Public License v3
`README`	základní informace ke kompilaci a licenci
`VERSION`	aktuální verze aplikace
`Makefile`	popis sestavení programu aplikací GNU make
`src/`	zdrojové soubory
`main.c`	inicializace programu

Vykreslování a časování:

`drawer.c`	vykreslování
`anim.c`	časované události, plynulá rotace

Obsluha vstupu, konzole a skriptování:

`hid.c`	obsluha událostí klávesnice a myši
`console.c`	obsluha příkazového řádku a vypisovaného textu
`consoleTransl.c`	překlad příkazů na výrazy Pythonu
`consoleCmd...`	implementace některých příkazů konzole
`consoleCmd.c`	
`consoleCmdSet.c`	příkaz `:set`
`consoleCmdVertex.c`	některé příkazy `:vertex...`
`script...`	implementace některých funkcí rozhraní Pythonu
`script.c`	rozhraní pro přístup k Pythonu z jiných modulů
`scriptEvents.c`	obsluha událostí
`scriptFigure.c`	funkce `gf.figure...`
`scriptVertex.c`	některé funkce `gf.vertex...`
`scriptWrappers...`	zpřístupnění funkcí C z Pythonu
`scriptWrappers.c`	
`scriptWrappers.pl`	generování obalovacích funkcí

Správa otevřeného tělesa a konvexního obalu:

`figure.c`	správa vykreslovaného tělesa
`convex...`	správa konvexního obalu a lámání stěn
`convex.c`	komunikace s ostatními moduly
`convexSpace.c`	afinní prostory
`convexFig...`	datová struktura uchovávající těleso
`convexFig.c`	hlavní část struktury
`convexFigMark.c`	verzované značky
`convexFigList.c`	spojový seznam nad tělesy
`convexFigHash.c`	globální hešovací tabulka
`convexHull...`	generování konvexního obalu
`convexHull.c`	správa připojeného tělesa
`convexHullUtil.c`	generování obalu libovolného tělesa
`convexInteract.c`	interakce s uživatelem během výpočtů

Různé:

`safe.c`	ošetřování chyb, ověřování vstupu, ...
`util.c`	práce s textovými řetězci, ...
`matrix.c`	vektorové a maticové operace
`strings...`	správa textů nápovědy
`strings.c`	
`stringsData.awk`	integrování textů do spustitelného souboru
`stringsData/`	texty nápovědy
`binFiles/`	soubory k přiložení bez kompilace
`COPYING`	celý text GNU General Public License v3
`README`	základní informace k předkompilované verzi

Součásti psané v Pythonu:

`config.py`	výchozí konfigurační soubor
`figures/`	přiložená tělesa
`modules/`	přiložené moduly
`algebra.py`	vektorové a maticové operace
`cuts.py`	řezy tělesa nadrovinou a odřezávání
`duals.py`	vytváření duálních těles
`figureInfo.py`	zobrazování informací o tělesech
`gfUtils.py`	jiná rozhraní ke gf
`helpMod.py`	nápověda k modulům a konfiguraci
`check.py`	kontrola konvexity a korektnosti těles, Union-Find
`objFigure.py`	objektová struktura tělesa
`randomRot.py`	náhodné rotace těles
`snapshots.py`	funkce zpět/vpřed
`spaceCuts.py`	těleso z průniků poloprostorů
`spaceNavigator.py`	obsluha 3D myši
`stellation.py`	stelace
`utils.py`	převody formátů, změna měřítka tělesa

Soubory generované programem make (nepřiložené):

`bin/`	soubory sestavené aplikace pro aktuální systém
`COPYING`	celý text GNU General Public License v3
`README`	informace o aplikaci, včetně čísla verze
`config.py`	výchozí konfigurační soubor
`geometric_figures`	spustitelný soubor aplikace
`figures/`	přiložená tělesa
`modules/`	přiložené moduly
`bin32/`	32bitová verze pro Linux
`bin64/`	64bitová verze pro Linux
`bin-win32/`	32bitová verze pro Windows
`pkg/`	zkomprimované archivy předchozích tří adresářů

2.2 Vykreslování a správa tělesa

2.2.1 Vykreslování (`drawer`)

K překreslení celého obsahu okna slouží funkce `drawerDisplay` volaná zpravidla knihovnou FreeGLUT, žádosti jiných částí aplikace o vyvolání překreslení realizuje funkce `drawerInvokeRedisplay`. O vykreslování jednotlivých objektů se pak starají privátní funkce modulu `drawer`, obvykle začínající `draw`.

Před projekcí všechny souřadnice vrcholů vynásobíme číslem `figureScale` zajišťujícím jednotkovou maximální vzdálenost od počátku a maticí aktuálního natočení `figureRotMatrix`. Obrazy bodů počítáme funkcí `matrixPerspective`, která kromě polohy bodů ve 3D také vrací relativní zvětšení vrcholů a konců hran v daném místě (viz sekce 1.2.1).

K vykreslování vrcholů (`drawVert3D`) využíváme knihovnu GLU, hrany vykreslujeme jako komolé jehlany (`drawEdge3D`) skládáním z trojúhelníků pomocí OpenGL. V obou případech počet trojúhelníků, z nichž se těleso skládá, určujeme dynamicky podle aktuální doby vykreslování snímku a požadované snímkové frekvence.

Dvourozměrné stěny s částečnou průhledností (`drawFace3D`) vykreslujeme jako jednoduché filtry (bez odlesků a stínování), aby výsledný obraz nebyl závislý na pořadí vykreslovaných stěn. Pro realističtější zobrazení by bylo potřeba stěny napřed seřadit podle vzdálenosti od kamery (ve 3D), a protože se ve zvolené projekci mohou i protínat, museli bychom také najít jejich průsečíky a stěny podle nich rozdělit. Seřazené stěny bychom poté vykreslovali od nejvzdálenějších. [1] To by mělo výrazný vliv na rychlost vykreslování. Pro rozložení stěn na trojúhelníky využíváme teselaci z knihovny GLU.

2.2.2 Správa vykreslovaného tělesa (`figure`)

Těleso v modulu `figure` uchováváme v následující jednodušší datové struktuře, než jsme používali v analyzovaných algoritmech (v sekci 1):

```
1 struct figureData {
2   int dim;
3   int *count;           // [i]:       i=0..dim
4   GLdouble **vertices; // [i][j]:    i=0..count[0]-1, j=0..dim-1
5   int ***boundary;      // [i][j][k]: i=1..dim,  j=0..count[i]-1,
6 };                       //             k=1..boundary[i][j][0]
```

Položka `dim` udává počet souřadnic prostoru; `count` určuje počty stěn daného počtu rozměrů (`count[0]` – počet vrcholů, ...); `vertices` je pole vrcholů, kde každý vrchol je pole jeho souřadnic (např. `vertices[0][dim-1]` je poslední souřadnice prvního vrcholu). Pole `boundary[`i`]` obsahuje i-rozměrné stěny tělesa (nultá položka je nevyužita). Položka `boundary[`i`][`j`][`k`]` udává index k-té fasety j-té i-rozměrné stěny v poli `boundary[`$i-1$`]`, příp. v poli `vertices` pro $i - 1 = 0$. Proměnná k je zde číslovaná od jedné a nultá položka obsahuje počet faset.

Původně měla struktura sloužit jen k popisu jednoho tělesa; obsahuje-li ale aktuální zobrazení více těles, jsou v ní obsažena všechna.

[1] Při vykreslování překrývajících se poloprůhledných objektů není možné použít z-buffer grafické karty, protože by se pak objekty skryté za již vykreslenými nevykreslily vůbec.

Pro správně fungující teselaci musíme seřadit hrany na hranici každé dvourozměrné stěny, což provádíme ve funkci `figureVerticesOfFaces`. Tu voláme před každým překreslením tělesa, ale její výstup přepočítáváme jen při jeho změně.

Při rotaci tělesa matici `figureRotMatrix` pomocí funkcí modulu `matrix` vynásobíme příslušnou maticí rotace získanou funkcí `matrixRotation`.

Při úpravách tělesa ve funkcích `figureVertexAdd`, `figureVertexRm` a `figureVertexMove` voláme příslušné funkce modulu `convex` pro úpravu hranice tělesa. Je-li aktivní aktualizace konvexního obalu (určeno proměnnou `convexHull`), vytvoří se celá hranice tělesa znova; jinak se pouze rozdělí stěny, jejichž vrcholy nově generují afinní prostor vyšší dimenze než měla původní stěna.

2.3 Ovládání a příkazový řádek

2.3.1 Obsluha klávesnice a myši (`hid`)

Knihovna FreeGLUT rozlišuje dva druhy kláves: *normální* a *speciální*. Speciální zahrnují klávesy F1, ..., F12, Home, End, PageUp, PageDown a kurzorové klávesy a jsou jednoznačně určeny svými kódy definovanými v knihovně. Ostatní (normální) klávesy jsou určeny svým kódem ASCII, který již jednoznačný není.

Ke každé události je možné zjistit aktuálně stisknuté modifikátory Ctrl, Alt a Shift. U normálních kláves jsou některé z těchto modifikátorů obsaženy i v kódu ASCII, což může způsobovat problémy. Například Ctrl+H má jiný kód ASCII než H, ale stejný kód i modifikátory jako Ctrl+Backspace. Díky modifikátorům může navíc dojít k uvolnění jiné klávesové zkratky, než byla stisknuta.

K událostem myši patří pohyb kurzoru a stisknutí/uvolnění tlačítka, přičemž mezi tlačítka se počítá i pootočení kolečka (jedním nebo druhým směrem), které generuje krátce po sobě stisk i uvolnění. Opět jsou k dispozici stisknuté modifikátory.

Obsluha 3D-myši je plně implementována příslušným modulem Pythonu (viz dále sekce 2.7 a 3.3.13) a v této kapitole se jí nebudeme zabývat.

Každou událost zakódujeme do jednoho čísla, to bude obsahovat její typ, modifikátory a informaci, které klávesy, tlačítka nebo osy myši se událost týká. Naopak nebude obsahovat vzdálenost, o níž se kurzor myši posunul. Tento kód budeme využívat k uchovávání uživatelem definovaného přiřazení událostí k akcím.

Událost do 32-bitového čísla binárně zakódujeme následovně:

```
  33222222 22221111 11111100 00000000
  10987654 32109876 54321098 76543210
1 00000000 00000ACS P0000000 ...ascii2 pro stisk/uvolnění normální klávesy,
2 00000000 00000ACS P0000001 .special2 pro stisk/uvolnění speciální klávesy,
3 00000000 00000ACS P10000.. ..button2 pro stisk/uvolnění tlačítka myši,
4 00000000 00000ACS 01XY00.. .buttons2 pro pohyb kurzoru myši v jedné ose.
```

Modifikátory Alt(`A`), Ctrl(`C`), Shift(`S`) jsou nastaveny na 1, pokud byly stisknuty; na 0 jinak. Stisk klávesy nebo tlačítka myši označuje `P` = 1, uvolnění `P` = 0. Bity `X`, `Y` označují, že jde o posun kurzoru myši v jedničkou označené ose; kombinace `11` není povolena a `00` je použito na třetím řádku.

Osmibitové kódy kláves ASCII (`...ascii`) a speciálních kláves (`.special`) určuje knihovna FreeGLUT. Desetibitový kód tlačítka myši (`.. ..button`) obsahuje právě jednu jedničku na pozici pořadového čísla tlačítka (určené knihovnou). Kód kombinace tlačítek myši (`.. .buttons`) je součtem kódů těchto tlačítek, stisknutá tlačítka mají u událostí posunu kurzoru stejnou funkci jako modifikátory.

Funkcí `hidCodeFromString` získáváme kódy událostí z jejich textových popisů (viz sekce 3.2.2) a funkcemi `hidCodeFromEvent` a `hidCodeFromMouseEvent` z událostí knihovny FreeGLUT.

Událostem přiřazujeme výraz Pythonu nebo rotaci. Výraz Pythonu se provede pokaždé, když nastane událost. Pokud je událostí pohyb kurzoru myši, může výraz obsahovat zástupný znak `%`, který nahradíme rozdílem nové a původní polohy kurzoru. Rotace je dána strukturou `animRotation` určenou dvěma poloosami souřadného systému. Rotaci můžeme přiřadit stisku klávesy nebo tlačítka myši, pak probíhá do jejich uvolnění, nebo osám kurzoru myši (při daných modifikátorech a stisknutých tlačítkách), kdy rotace závisí na pohybu kurzoru. Více k uživatelskému rozhraní uvedeme v sekci 3.2.

Nastavené akce událostí uchováváme v poli `mapped` setříděné podle jejich kódů. Přidání prvku do pole tak trvá až lineární čas, vyhledání je poté binární v logaritmickém čase. Typicky pole naplníme při provádění konfiguračního souboru a poté se již nemění, nastavených událostí navíc nebývá mnoho a kvadratický čas na inicializaci je proto dostačující.

Prvky pole `mapped` jsou instance následující struktury:

```
struct mappedItem {
  int code;
  char *expr[2];
  struct animRotation *rot;
  bool pressed;
  struct mappedItem *prev, *next;
};
```

Kód události (`code`) má vždy nulu na pozici příznaku `P` a součástí struktury je zároveň výraz pro stisk (`expr[1]`) i uvolnění (`expr[0]`) klávesy nebo tlačítka myši a příslušná rotace (`rot`). Je-li událostí posun kurzoru myši, využíváme pro uložení výrazu `expr[0]`. Všechny právě aktivní události (stisknuté klávesy a tlačítka myši) mají nastavený příznak `pressed` a uchováváme je v seznamu `mappedPressed` pomocí položek `prev` a `next`.

Binární vyhledávání zajišťuje funkce `mappedBS`, kterou potřebujeme pouze při stisku klávesy. Při uvolnění klávesy událost hledáme v seznamu `mappedPressed`, a pokud se tam nenachází (kvůli odlišným modifikátorům při stisku), žádná akce nenastane. Pro tyto případy udržujeme také skutečný počet stisknutých kláves (`pressedCnt`) a při jeho vynulování seznam `mappedPressed` vyprázdníme a zastavíme probíhající rotace.

Při zpracovávání události ve funkci `keyEvent` (volané `hidKeyEvent`) rozlišujeme následující stavy aplikace: Je-li aktivní modul `convexInteract` předáme událost jemu. Probíhá-li funkce `animSleep` (volaná z Pythonu jako `gf.sleep`), přerušíme jí a událost zpracujeme až po ukončení volajícího skriptu. Je-li otevřená konzole předáme potřebné události pomocí funkcí `consoleEnter`, `console-`

`Backspace`, ..., `consoleKeyPress` (pro znaky). Byla-li stisknuta dvojtečka, otevřeme konzoli. Jinak provedeme uživatelem definovanou akci.

2.3.2 Příkazový řádek (`console`)

V modulu `console` uchováváme všechny aktuálně zobrazené texty i historii zadaných příkazů. Modul obsahuje funkce pro vypisování textu (`consolePrint...`), vykonávání příkazů (`consoleExecuteCmd`) a pro obsluhu klávesnice při otevřeném příkazovém řádku. Texty k vypsání získává modul `drawer` pomocí vyhrazeného rozhraní `consolePrivDraw.h`.

Veškeré víceřádkové texty (a historie) jsou reprezentovány spojovými seznamy řádků (`struct utilStrList`). K alokaci paměti jednořádkových textů využíváme funkci `utilStrRealloc`. První volání této funkce vytvoří pole znaků přesné velikosti, každé další volání pole zvětšuje (je-li to potřeba) a to na nejbližší vyšší mocninu dvojky. (Výjimkou je pouze volání s požadovanou velikostí 0, kdy se pole uvolní.) Tento přístup umožňuje dynamické zvětšování řetězců s amortizovaně konstantní složitostí na znak.

Pro vykonání příkazu ve funkci `consoleExecuteCmd` jej nejprve přeložíme na výraz Pythonu pomocí `consoleTranslToScriptExpr` a ten poté vykonáme funkcí `consoleEvalExpr`. Tato funkce kromě volání `scriptEvalExpr` také zařídí odchycení případných výjimek (pomocí `scriptCatchException`) a jejich vypsání. Podobně je funkce pro vykonávání skriptů `scriptExecFile` zapouzdřena funkcí `consoleExecFile`.

Pro automatické dokončování voláme během zadávání znaků funkci `consoleTranslComplete`, která vrací seznam (`struct utilStrList`) možných pokračování příkazu. Všechny vestavěné příkazy konzole po spuštění aplikace zavádíme ve funkci `initCmds`.

2.3.3 Překlad příkazů na výrazy Pythonu (`consoleTransl`)

Modul `consoleTransl` spravuje všechny aktuálně zavedené příkazy, které překládá na výrazy Pythonu, a názvy barev, které překládá do číselné podoby. Také umožňuje automatické dokončování příkazů, barev a cest v souborovém systému.

Hlavní datovou strukturou je zde trie (prefixový strom):

```
1 struct trie {
2   char c;
3   struct trie *sibling;
4   struct trie *child;
5   char *scriptExpr;
6   char *paramsFlags;
7   int params;
8   enum importance importance;
9 };
```

Každý uzel obsahuje poslední znak (`c`) prefixu, který zastupuje, a může odkazovat na jednoho svého následníka (`child`), ten pak odkazuje na jiného následníka jako na svého sourozence (`sibling`); následníci uzlu tak tvoří spojový seznam. Ostatní položky struktury jsou využity pouze u koncových uzlů.

Řetězec `scriptExpr` obsahuje výraz Pythonu, na který se má daný příkaz (tvořený posloupností následníků) přeložit. Tento výraz může obsahovat zástupné znaky `%`, které budou při překladu nahrazeny parametry.

Typy parametrů určuje pole `paramsFlags`, které obsahuje jeden znak pro parametr, poslední znak udává typ všech zbývajících parametrů. Typy parametrů jsou literál (`'-'`), textový řetězec (`'s'`), cesta v souborovém systému (`'p'`), barva (`'c'`) a barva s průhledností (`'C'`).

Absolutní hodnota čísla `params` udává maximální počet parametrů, při kladné hodnotě se každý znak `%` nahradí jedním parametrem, při záporné se za první `%` dosadí čárkami oddělený seznam všech.

Hodnoty `paramsFlags` a `params` jsou stejné jako hodnoty parametrů funkce Pythonu `gf.addCommand`, kterou popíšeme v sekci 3.3.1.

Důležitost (`importance`) ovlivňuje pořadí nabízených pokračování příkazu. Nejprve se nabízí nezkrácené verze vestavěných příkazů (`builtinCmd`), poté uživatelem definované příkazy (`userCmd`) a nakonec zkratky vestavěných (`builtinAliasCmd`).

Trie příkazů je uložena v proměnné `commands`.

Při použití trie na názvy barev (proměnné `colors` a `colorsWithAlpha`) je výsledná barva ve formátu `'#RRGGBB'` nebo `'#AARRGGBB'` uložena v položce `scriptExpr` a `paramsFlags` i `params` jsou nevyužité.

2.3.4 Implementace příkazů konzole (`consoleCmd...`)

Moduly `consoleCmd`, `consoleCmdSet` a `consoleCmdVertex` obsahují implementace většiny příkazů dostupných z konzole. Tyto moduly jsou proto závislé na mnoha jiných. Například `consoleCmdSet` spravuje nastavení celé aplikace, které je často přímo součástí příslušných modulů.

Implementace některých příkazů nalezneme přímo v modulech, s nimiž souvisí. Všechny příkazy konzole jsou dostupné jako funkce Pythonu, na jejichž volání se překládají.

2.3.5 Texty nápovědy (`strings`)

Každá obrazovka příkazu `:help` je uložena v jednom textovém souboru ve složce `src/stringsData/`. Jména souborů jsou tvořena některým z příkazů, pod nimiž je daná obrazovka dostupná, s nahrazenými mezerami za podtržítka (např. `help_welcome`). Samotný název není podstatný, ale musí obsahovat znaky použitelné v názvu proměnné v C.

První řádek souboru obsahuje seznam dvojic oddělených středníky, kde každou dvojici tvoří název sekce (aktuálně vždy `help`) a název textu (část za příkazem `help`) oddělené dvojtečkou. Například uvítací zpráva, dostupná pod příkazy `:help welcome`, `:help license` a `:help about` má první řádek:

```
1 help:welcome;help:license;help:about
```

Další řádky tvoří obsah obrazovky nápovědy. Povolené jsou pouze znaky ASCII, které je možné používat v řetězcích jazyka C, včetně uvozovek a speciálních sekvencí uvozených zpětným lomítkem. Znak `\t` je zakázán a znak `\b` po-

souvá kurzor o znak doleva, což umožňuje napsat například znak podobný š jako `\b'\bs. Řetězec @VERSION@ se nahradí aktuální verzi aplikace.

Omezení na znaky ASCII je vynuceno knihovnou FreeGLUT, zajišťující vypisování textu, která jiné znaky nepodporuje.

Skript stringsData.awk vygeneruje soubor stringsData.c.tmp, který obsahuje všechny texty jako řetězcové konstanty a navíc obsahuje funkci pro jejich vyhledávání. Posloupnosti mezer (používané pro zarovnání) se přitom zkomprimují na znak \t a jednobajtové číslo udávající jejich počet.

V ostatních modulech aplikace poté můžeme využívat funkci stringsGet, vracející seznam řádků (s expandovanými mezerami), a stringsGetContent, vracející pole názvů textů dané sekce (pro použití při automatickém dokončování).

Zde je příklad souboru help_close a z něj vygenerované konstanty:

```
help:close
        close

Destroys opened space.
```

```
const char *stringsData_help_close[]={"help:close",
  "\t\010close\n\nDestroys opened space.\n"};
```

2.4 Integrace Pythonu (script...)

2.4.1 Rozhraní pro ostatní moduly aplikace (script)

Po spuštění aplikace ve funkci scriptInit importujeme všechny moduly Pythonu dostupné ve složce modules.

Modul script obsahuje funkce pro vyhodnocení výrazu (scriptEvalExpr) i vykonání souboru skriptu (scriptExecFile). Dále obsahuje funkce pro vyvolání výjimky (scriptEvalExpr) a její odchycení (scriptCatchException...) a!funkce pro práci s hlavním zámkem GIL Pythonu.

Vstupem i výstupem všech těchto funkcí jsou pouze proměnné základních typů jazyka C, díky čemuž nemusíme v ostatních částech aplikace využívat hlavičkový soubor Python.h. Na druhou stranu výjimka je zde definovaná pouze textovým řetězcem a veškeré vyvolané výjimky jsou typu RuntimeError.

Vyvolávání výjimek je potřeba ve funkcích, které jsou přístupné z Pythonu a naopak jejich odchytávání při volání funkcí Pythonu z C. Volání funkce se vždy normálně vrátí; pro zjištění, jestli nastala výjimka, je potřeba použít některou z funkcí scriptCatchException... Všechny výjimky se v aplikaci interpretují jako chyby, které jsou vypsány na obrazovku, a aplikací vyvolané výjimky obvykle nejsou určeny k odchytávání, což byl jeden z důvodů pro zavedení tohoto jednoduchého rozhraní bez rozlišování typů výjimek.

Při každém přístupu k interním strukturám Pythonu musí dané vlákno vlastnit hlavní zámek. To je vždy splněno při volání funkcí C z Pythonu; při vykonávání kódu pomocí výše uvedených funkcí se o zámek stará modul script. Jediné místo mimo modul script, kde se pracuje se zámkem, je při volání animSleep z Pythonu. Vykonávání této funkce záměrně trvá delší dobu a je proto potřeba

na začátku uvolnit zámek pomocí `scriptReleaseGIL` a poté jej opět získat pomocí `scriptAcquireGIL`, jinak by byla blokována všechna ostatní vlákna modulů Pythonu.

2.4.2 Obalovací funkce (`scriptWrappers`)

Pro snadné zpřístupnění funkcí C z Pythonu vytváříme při sestavování aplikace skriptem `scriptWrappers.pl` obalující funkce. Tyto funkce pracují s datovými typy interpretu, překládají je na datové typy jazyka C a zpět a volají původní funkce jazyka C.

Pro zpřístupnění funkce z Pythonu stačí v hlavičkovém souboru uvést na řádku s deklarací funkce komentář ve tvaru `[SCRIPT_NAME:` *název_funkce*`]`, kde *název_funkce* určuje název, pod nímž bude funkce přístupná ve jmenném prostoru modulu `gf` Pythonu.

Podporované datové typy parametrů a návratových hodnot jsou `char*`, `int`, `bool` (interpretuje se jako `int`), `float`, `double`, a jako návratová hodnota také `void`.

Funkce Pythonu také můžou mít variabilní počet argumentů. V tom případě musí být jeden z posledních dvou parametrů typu ukazatel na některý z výše uvedených typů (pole zbývajících argumentů) a druhý typu `int` se stejným názvem doplněným o suffix `Cnt` (počet zbývajících argumentů). Funkce pak bude přijímat parametry uvedené před posledními dvěma následované libovolným počtem parametrů daného typu. Například funkci s deklarací

```
extern
char *naseF(float f, char **s, int sCnt); //[SCRIPT_NAME: jejichF]
```

můžeme z Pythonu zavolat například takto:

```
vystupni_text=gf.jejichF(5.6, "text1", "text2", "posledni")
```

a v těle funkce potom budou proměnným přiřazeny hodnoty:

```
float f=5.6;
char **s;
int sCnt=3;
s[0]="text1";
s[1]="text2";
s[2]="posledni";
```

Variabilní počet argumentů nakonec nebyl v aplikaci využit.

Je také povoleno přetěžování funkcí. Má-li více funkcí stejný *název_funkce*, vybere se vždy první s vyhovující signaturou.

Funkce, které potřebují pracovat se složitějšími datovými strukturami Pythonu, jsou volány přímo (bez obalovací funkce) a není u nich proto povoleno přetěžování výše popsaným způsobem. Takovéto funkce musí mít následující signaturu:

```
extern
PyObject *f(PyObject *self, PyObject *args); //[SCRIPT_NAME: f]
```

2.4.3 Rozhraní gf Pythonu (scriptVertex, scriptFigure)

Implementace funkcí přístupných z Pythonu prostřednictvím jeho modulu `gf` se nacházejí v různých modulech aplikace. Velká část těch, které jsou zároveň přístupné z konzole, je v modulech `consoleCmd`... a například `gf.sleep` pochází z modulu `anim` jako funkce `animSleep`.

V modulech `scriptVertex` a `scriptFigure` se nacházejí funkce, které přímo pracují s datovými strukturami Pythonu. V modulu `scriptFigure` je například implementován obousměrný převod mezi datovou strukturou `figureData` (popsanou v sekci 2.2.2) a odpovídající datovou strukturou Pythonu, kterou popíšeme v sekci 3.3.1 uživatelské dokumentace.

2.4.4 Události (scriptEvents)

Modul `scriptEvents` umožňuje modulům Pythonu reagovat na některé události, které v aplikaci nastanou. Podporovanými událostmi jsou `Idle`, `New`, `Open`, `Write` a `Modified`. Poslední čtyři nastávají při změně otevřeného tělesa a vyvolává je modul `figure`, `Idle` je vyvoláváno modulem `anim` v každém snímku, neprobíhají-li jiné akce, které by to vyloučily. Podrobněji události popíšeme v uživatelské dokumentaci (v sekci 3.3.1).

Ke každému typu události existuje instance struktury `scriptEvent`, která obsahuje název události (`name`), řetězec popisující typy argumentů (`argsFormat`), a seznam registrovaných funkcí Pythonu (`callbacks`):

```
1 struct scriptEvent {
2   char *name;
3   char *argsFormat;
4   struct callbacks *callbacks;
5 };
6 struct callbacks {
7   PyObject *callback;
8   struct callbacks *next;
9 };
```

Řetězec `argsFormat` je přímo předáván funkci `Py_VaBuildValue` knihovny Pythonu, která zajišťuje převod (variabilního počtu) argumentů funkce `scriptEventsPerform` na argumenty registrované funkce Pythonu. Formát řetězce je tak definován knihovnou. Funkce `scriptEventsPerform` má jako parametry ukazatel na událost (`struct scriptEvent`) následovaný případnými argumenty události.

Rozhraní pro registraci obslužných funkcí Pythonu popíšeme v uživatelské příručce.

2.5 Časování (anim)

Modul `anim` zajišťuje pravidelné překreslování obrazovky (pomocí modulu `drawer`), volání události `Idle` Pythonu a čekání ve funkci `animSleep` (přístupné jako `gf.sleep`).

Proměnná `animFrameDelay` určuje očekávanou prodlevu mezi snímky (přepočítáváno z volby aplikace `maxfps`), která se používá k nastavení časování pomocí knihovny FreeGLUT. Ta v daných intervalech periodicky volá funkci `frame`,

která zajišťuje rotaci tělesa (při stisknuté klávese) a volání události `Idle`. K překreslení obrazovky dochází pouze při změně.

Událost `Idle` nenastává, je-li aktivní `animSleep`, `convexInteract` nebo je otevřená konzole.

Funkce `animSleep` je určena k pozastavení vykonávání skriptu na dobu určenou parametrem a dočasné obsluze událostí. Funkce obsahuje vlastní smyčku událostí, protože smyčku knihovny FreeGLUT není možné volat rekurzivně. Vždy se obslouží všechny události (pomocí `glutMainLoopEvent`) a poté se vlákno uspí funkcí `utilSleep` a to maximálně na dobu určenou konstantou `animResponseDelay`. Při vykonávání této funkce tak může být mírně zvýšeno využití procesoru a snížena odezva aplikace.

2.6 Správa konvexního obalu (`convex...`)

2.6.1 Správa prostorů (`convexSpace`)

Afinní prostory jsou uloženy v následující struktuře:

```
1 struct convexSpace {
2   int dim;
3   int coordsCnt;
4   GLdouble *pos;
5   GLdouble *ortBasis;
6   GLdouble *normal, normalPos;
7 };
```

Položka `dim` udává dimenzi daného prostoru a `coordsCnt` počet souřadnic. Vektor `pos` obsahuje souřadnice libovolného bodu afinního prostoru; `ortBasis` obsahuje za sebou vektory ortonormální báze (celkem $\texttt{dim} \cdot \texttt{coordsCnt}$ číselných položek). Normála (`normal`) a posun v jejím směru (`normalPos`), se vztahují k nějakému nadprostoru.

Struktura prostoru může existovat samostatně nebo jako součást struktury tělesa (`convexFig`). K tělesu je hned po vytvoření alokována paměť afinního prostoru. Pro přiřazení (zkopírování) existujícího prostoru k tělesu využíváme funkci `convexSpaceAssign`, která před zkopírováním alokuje potřebnou paměť pro bázi (`dim` vektorů).

Naproti tomu funkcemi `convexSpaceCopy`, `convexSpaceCreate...`, `convexSpaceDestroy` a dalšími, které mění bázi prostoru, upravujeme pouze samostatné prostory. Těm vždy alokujeme paměť pro maximální počet vektorů báze (`coordsCnt`). Přepočítání normály a obsaženého bodu je povoleno i u prostorů těles.

Pro maticové výpočty využíváme modul `matrix`. Operace s prostory byly popsány v sekci 1.3.

2.6.2 Správa tělesa (convexFig...)

V modulech convex... je těleso uloženo v následující datové struktuře, která již byla popsána v sekci 1.4.1 analýzy:

```
struct convexFig {
  int index;
  unsigned int hash;
  unsigned int mark[convexFigMarkCount];
  struct convexFigList *parents;
  struct convexFigList *boundary; // children
  struct convexFigList *vertices;
  struct convexSpace *space;
};
```

Seznamy rodičů (parents), synů (boundary) a vrcholů (vertices), heš tělesa (hash), prostor (space) i verzované značky (mark) byly popsány v analýze. Položka (index) se vztahuje k datové struktuře figureData, jejíž hranice se upravuje; při synchronizaci se aktualizují jen upravené položky.

V modulu convexFigMark jsou definovány následující značky:

```
enum convexFigMarkId {
  convexFigMarkIdTrue=-1,
  convexFigMarkIdLayer=0,
  convexFigMarkIdHash,
  convexFigMarkIdHull,
  convexFigMarkIdHullProcessed,
  convexFigMarkIdHullOneParent,
  convexFigMarkCount
};
```

Položka convexFigMarkCount pouze udává celkový počet značek (svou hodnotou). Značka convexFigMarkIdTrue má speciální význam (je vždy přiřazena všem tělesům) a může být použita jen některými funkcemi. Hodnoty ostatních značek odpovídají indexům v poli mark.

Příkladem použití značek je funkce convexFigGetLayer, která pro dané těleso vrátí seznam všech jeho stěn dané dimenze, příp. seznam všech jeho předků dané dimenze. Není-li parametrem určeno jinak, využívá přitom značku convexFigMarkIdLayer. Vrácený seznam je možné zároveň nechat profiltrovat, aby obsahoval jen prvky se značkou určenou dalším parametrem.

Modul convexFigList umožňuje vytváření jednoduchých spojových seznamů nad tělesy:

```
struct convexFigList {
  struct convexFig *fig;
  struct convexFigList *next;
};
```

Modul convexFigHash spravuje globální hešovací tabulku těles. Tělesa do ní můžeme přidávat funkcí convexFigHashAdd, mají-li platný heš (získaný funkcí convexFigHashCalc).

Pro vyhledávání používáme funkci `convexFigHashFind`, která má jako parametry heš, počet rozměrů tělesa a počet vrcholů a požaduje, aby právě všechny vrcholy tělesa měly značku `convexFigMarkIdHash`. Při nalezení tělesa v tabulce zkontrolujeme jeho heš (jako index používáme pouze jeho část) a dimenzi a poté spočítáme jeho vrcholy a zkontrolujeme značky. Nalezení tělesa tak trvá lineární čas vzhledem k počtu vrcholů při průměrném chování hešování.

2.6.3 Rozhraní (`convex`, `convexInteract`)

Moduly `convex`... je možné používat dvěma způsoby:

Pomocí funkce `convexUpdateHullAtOnce` se pouze aktualizuje konvexní obal tělesa (`figureData`) předaného parametrem, bez interakce s uživatelem. Tento způsob využíváme při volání z Pythonu.

Druhou možností je použití funkce `convexAttach` pro připojení tělesa a dalších funkcí volaných při jeho následných změnách (`convexVertexAdd`, ...). Původní těleso i těleso v modulech `convex`... se tak udržují synchronní. V tomto režimu moduly `convex`... umožňují také lámání stěn v případě, že není aktualizace konvexního obalu aktivní (při posunu vrcholu se incidentní stěny rozpadnou na simplex a zbylou část).

Druhá varianta zahrnuje také interakci s uživatelem v případě, že by hledání konvexního obalu trvalo dlouho, což se ale při současné implementaci nestává často. V průběhu se volají pravidelně funkce modulu `convexInteract`, které jednou za 200 ms překreslí obrazovku a zkontrolují, zda nebylo hledání přerušeno uživatelem.

2.6.4 Generování konvexního obalu (`convexHull...`)

Modul `convexHullUtil` obsahuje funkce pro správu konvexního obalu popsané v analýze s příslušnými prefixy: `convexHullUtilCreate` (viz sekce 1.4.2), `convexHullUtilExpandDim` (viz 1.4.3) a `convexHullUtilComplete` (viz 1.4.4).

Modul `convexHull` těchto funkcí využívá k hledání konvexního obalu aktuálně připojeného tělesa.

2.7 Moduly Pythonu

Implementace modulů je většinou přímočará, proto zde popíšeme jen některé její části. Popis rozhraní modulů uvedeme v uživatelské příručce v sekci 3.3, závislosti mezi nimi jsou znázorněny na Obrázku 2.2.

Základní operace lineární algebry zde poskytuje modul `algebra` (obdoba modulu `matrix` aplikace).

Pro práci s tělesem máme k dispozici opět dvě datové struktury: Jednodušší z nich slouží k výměně těles s aplikací pomocí modulu `gf` (obdoba struktury `figureData`) a obvykle bývá označována `gfFigure`. Druhá, poskytovaná modulem `objFigure`, umožňuje pohodlnější práci s tělesem (obdoba struktury `convexFig`). Modul `objFigure` zprostředkovává také vzájemné převody mezi strukturami.

V modulu `objFigure` je u třídy tělesa implementován iterátor, který prochází všechny stěny tělesa vč. jeho samého. K tomu jsou použity verzované značky

(popsané v sekci 1.4.1), které namísto číslování používají objekty. S novou verzí se vytvoří instance třídy `object`, která představuje aktuální verzi. Podobný princip využíváme i v některých dalších modulech.

V analýze již bylo popsáno vytváření duálních těles modulem `duals` (1.5), stelace pomocí `stellation` (1.7), řezy tělesa modulu `cuts` (1.8) i reprezentace těles průnikem poloprostorů v modulu `spaceCuts` (1.6).

Modul `cuts` dále umožňuje odřezávání částí konvexních těles (vrcholů, hran, apod.). To provádíme převodem tělesa na průnik poloprostorů, přidáním poloprostorů příslušných nadrovin řezů a převodem zpět. Původně byla tato funkce implementována prostřednictvím řezů modulu `cuts`, to ale bylo výrazně pomalejší, protože bylo potřeba řezy aplikovat po jednom.

Modul `check` ověřuje, zda tělesa mají dokončenou hranici, jsou konvexní a jejich hranice je souvislá. Hranice je dokončená právě tehdy, když splňuje Lemma 2. Těleso je konvexní, jsou-li pro každou fasetu všechny vrcholy na jedné její straně. Pro ověření souvislosti hranice se používá struktura Union-Find popsaná jako součást řezů těles (v sekci 1.8), která je také implementovaná v tomto modulu.

Modul `gfUtils` poskytuje pohodlnější rozhraní k některým funkcím modulu `gf`, které nemohlo být implementováno přímo v něm kvůli závislostem na jiných modulech. Další užitečné funkce (nezávislé na modulu `gf`) obsahuje modul `utils`.

Modul `helpMod` zprostředkovává nápovědu k ostatním modulům a ke konfiguračnímu souboru. Textové popisy těles spravuje modul `figureInfo`, který je využíván připravenými soubory s tělesy. Soubory těles i ostatních modulů jsou psány tak, aby je bylo možné používat i bez zmíněných dvou modulů.

Modul `snapshots` umožňuje zaznamenávání a obnovování historie vnitřního stavu aplikace i modulů v rámci jednoho běhu aplikace, mj. pak poskytuje funkce `undo` a `redo`.

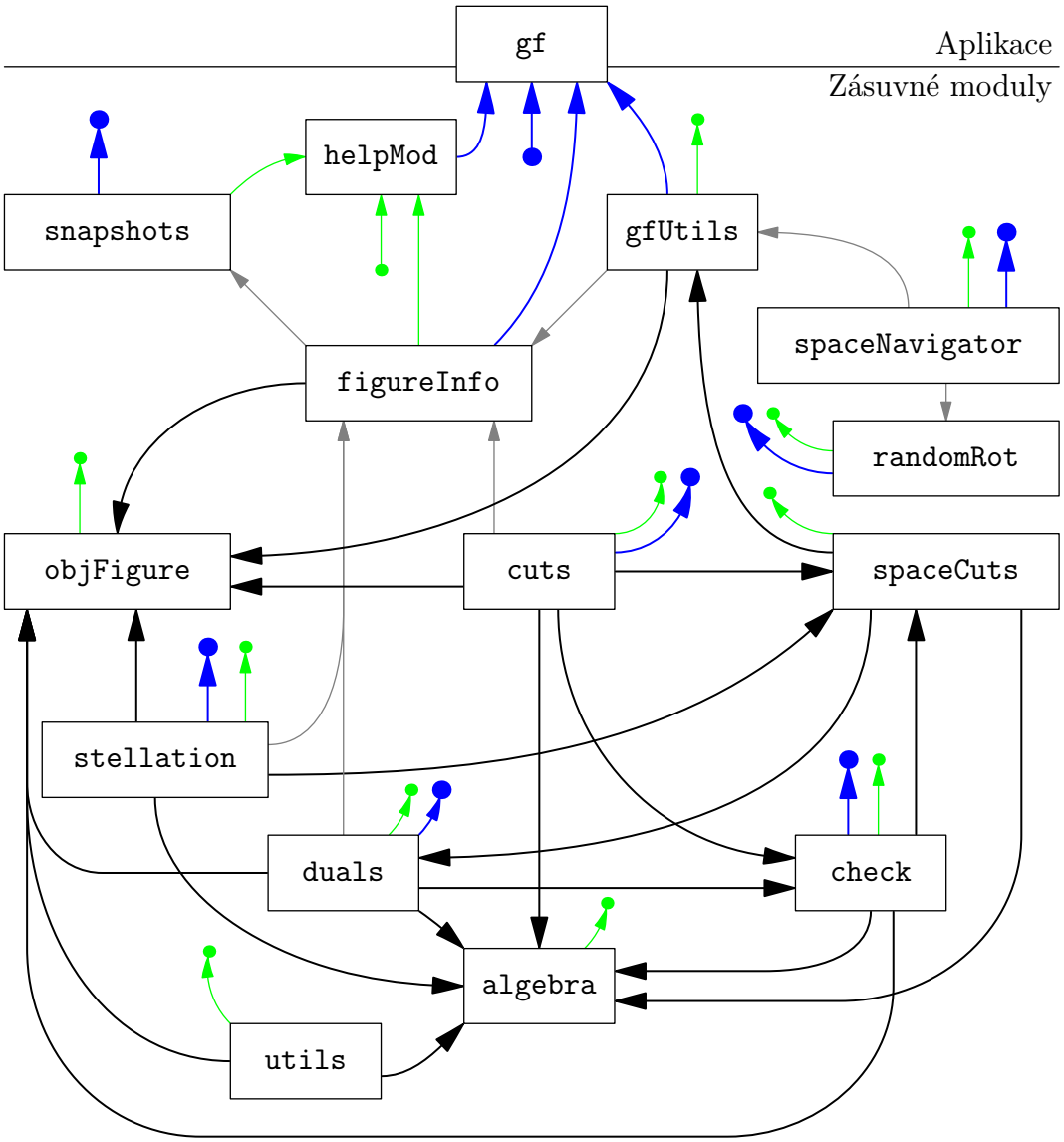
Vnitřním stavem aplikace je zde myšleno aktuálně otevřené těleso; ke sledování jeho změn využíváme události modulu `gf` (`new`, `open` a `modified`). Z modulů můžeme volat funkce pro hlášení změn a pro registraci funkcí obnovujících předchozí stav. Toho využíváme například v modulu `figureInfo` pro udržování relevantních textových popisů těles.

Při nahlášení změny stavu libovolné součásti aplikace se vytvoří nová položka historie, pokud již uběhlo více než 20 ms od nahlášení předchozí změny. To umožňuje seskupování změn, které spolu souvisí bez ohledu na jejich pořadí (např. otevření tělesa a nastavení jeho popisu). Položkou historie je slovník obsahující vnitřní stavy součástí (jimi definované struktury) jako hodnoty a názvy součástí jako klíče.

Modul `randomRot` umožňuje náhodné rotace těles. V každém snímku vždy těleso otočíme o malé náhodné úhly ve směrech všech kombinací dvojic poloos, a je-li to požadováno, upravíme podobným způsobem také vlastnosti perspektivy. K čekání na další snímek využíváme funkci `gf.sleep`, během náhodných rotací je tak blokována událost `Idle`, kterou využívá například modul `spaceNavigator` pro obsluhu 3D myši.

Ovládání aplikace 3D myší je podporováno pouze na Linuxu, protože využíváme systémově závislý nízkoúrovňový přístup k zařízení: Binárně čteme příslušný soubor vstupního zařízení v adresáři `/dev/input/`. Způsob zpracování těchto

binárních dat byl převzat z diskuze na serveru Stack Overflow (18. 5. 2016). Blokujícímu čtení je vyhrazeno samostatné vlákno. Díky jednotnému rozhraní je možné modul snadno upravit pro použití s jiným vstupním zařízením.



Obrázek 2.2: Schéma závislostí mezi zásuvnými moduly. Černá nebo modrá šipka z A do B znamená, že modul A závisí na modulu B; šedá nebo zelená šipka z A do B znamená, že modul A využívá služby modulu B, ale není na něm závislý. Všechny modré šipky vedou do modulu `gf`, všechny zelené do modulu `helpMod`.

3. Uživatelská dokumentace

Aplikace Geometric Figures je svobodný software s otevřeným zdrojovým kódem zveřejněný pod licencí GNU General Public License verze 3, kterou vydala nadace Free Software Foundation. Licence uživatelům aplikace umožňuje přístup ke zdrojovému kódu i jeho následné úpravy a šíření aplikace za podmínky zachování těchto svobod i uživatelům upravených verzí. Celý text licence je k dispozici v souboru COPYING přiloženém k aplikaci nebo na webu GNU (FSF, 2007).

Uživatelská příručka se vztahuje k verzi aplikace 1.5. Předkompilovaná aktuální verze aplikace pro Linux i Windows je k dispozici na stránkách projektu

`http://ondracek-lukas.hys.cz/geometric-figures`,

zdrojové kódy aktuální i starších verzí jsou dostupné na adrese

`http://github.com/ondracek-lukas/geometric-figures`.

Zde popisované zdrojové kódy jsou také přílohou elektronicky odevzdávané verze bakalářské práce jako archiv `geometric-figures.tar.gz`, jehož obsah je popsán v sekci 2.1.

3.1 Instalace

Aplikace vyžaduje knihovny **OpenGL**, **FreeGLUT 3.0** a **Python 2.7**.

Instalace na Linuxu

Knihovny nainstalujeme nejčastěji z repozitáře používané distribuce, na Debianu to jsou balíčky `freeglut3` a `python2.7`.

Aplikaci sestavíme podle postupu v následující podkapitole 3.1.1, nebo do libovolné složky rozbalíme balíček `tar.gz` s předkompilovanými soubory aplikace stažený z výše uvedených stránek projektu.

Zápis do souborového systému je aplikací vyžadován pouze při explicitním ukládání vytvořených těles a není vázán na adresář s aplikací, můžeme ji tedy umístit i do adresáře `/opt/` bez práv zápisu. V konfiguračním souboru `config.py` poté můžeme nastavit načítání uživatelského konfiguračního souboru přidáním:

```
1 import os
2 if os.access(
3     gf.expandPath("~/.geometric_figures-config.py"), os.R_OK):
4   gf.source("~/.geometric_figures-config.py");
```

Poté stačí spustit soubor `geometric_figures`.

Instalace na Windows

Aplikaci stáhneme z výše uvedených stránek projektu jako archiv formátu `zip` a rozbalíme ji do libovolné složky. Knihovny FreeGLUT a Python můžeme také stáhnout ze stejných stránek a bez instalace je rozbalit do složky s aplikací: Archiv

s aplikací obsahuje složku `geometric_figures`, do níž je potřeba přesunout obsah složky `python-freeglut-win32` z druhého archivu.

Poté stačí spustit soubor `geometric_figures.exe`.

Alternativně můžeme knihovny získat z jejich původních zdrojů následovně: Nainstalujeme Python verze 2.7 stažený z oficiální stránek

`https://www.python.org/`

a ze stránek

`http://www.transmissionzero.co.uk/software/freeglut-devel/`

stáhneme archiv *freeglut 3.0.0 for MinGW*, z něhož po rozbalení zkopírujeme soubor `freeglut.dll` do složky s aplikací.

Pro sestavení aplikace ze zdrojových souborů (i pro Windows) je v aktuální verzi potřeba Linux.

3.1.1 Sestavení aplikace ze zdrojových kódů

K sestavení aplikace pro všechny podporované systémy využíváme na Linuxu program `GNU make`. Zdrojové kódy stáhneme z adresy uvedené na začátku této kapitoly například takto:

```
git clone 'http://github.com/ondracek-lukas/geometric-figures.git'
cd geometric-figures
```

Pro vynucení zde popisované verze namísto aktuální můžeme použít příkaz:

```
git checkout '1.5'
```

Pro sestavení nativní verze jsou na Debianu potřeba balíčky `gawk`, `perl`, `freeglut3-dev`, `python2.7-dev` a další obvykle přítomné v systému.

Pro sestavení 64bitové verze na 32bitovém systému nebo naopak jsou potřeba některé balíčky příslušné architektury.

Pro sestavení verze pro Windows jsou navíc potřeba balíčky `gcc-mingw-w64` pro kompilaci a `dos2unix` pro převod konců řádků textových souborů na CRLF. Dále jsou potřeba knihovny `freeglut.dll` a `python27.dll`, jejichž získání je popsáno u instalace pro Windows (druhá z nich je součástí instalace Pythonu), a hlavičkové soubory obvykle dostupné v adresáři `C:\Python27\include` po instalaci Pythonu na Windows.

Po splnění závislostí můžeme použít následující příkazy:

```
make [compile]
```

Sestaví aplikaci pro aktuální systém ve složce `bin/`.

```
make package
```

Sestaví aplikaci a zkomprimuje ji do archivu `pkg/geometric-figures.tar.gz`.

```
make clean-tmp
```

Vymaže všechny dočasné soubory (`obj/`, `src/*.tmp`, ...).

`make clean`

Odstraní všechny soubory vytvořené předchozími příkazy.

`make ([compile]|package|clean-tmp|clean) arch=32`

Pracuje s 32bitovou verzí pro Linux. K názvům souborů přidává sufix `32`.

`make ([compile]|package|clean-tmp|clean) arch=64`

Pracuje s 64bitovou verzí pro Linux. K názvům souborů přidává sufix `64`.

`make ([compile]|package|clean-tmp|clean) arch=win32 [console=1]`

Pracuje s 32-bitovou verzí pro Windows. Přidává sufix `-win32`. Vytvořený archiv je formátu `zip`, textové soubory mají řádky ukončené CRLF a příponu `.txt`, soubor aplikace má příponu `.exe`. Nepovinný parametr `console=1` způsobí, že se kromě hlavního okna aplikace bude otevírat také textová konzole obsahující standardní výstup.

`make (compile|package|clean-tmp|clean)-all`

Provádí danou operaci se všemi třemi architekturami.

`make ... debug=1`

Při kompilaci přidává do souborů ladící symboly a ve verzi pro Linux vytváří hotová aplikace coredump při svém pádu.

3.2 Spuštění a ovládání aplikace

Aplikace sestává ze spustitelného souboru `geometric_figures`, konfiguračního souboru `config.py`, zásuvných modulů v adresáři `modules/` a přiložených těles v adresáři `figures/`. Základní informace k aplikaci a její licenci jsou k dispozici v textovém souboru `README`, plný text licence poté v souboru `COPYING`.

Konfigurační soubor, soubory těles i zásuvné moduly jsou psány v jazyce Python. Po spuštění aplikace dojde k importu modulů, vykonání konfiguračního souboru a případně vykonání dalších souborů předaných aplikaci jako argumenty. Argumentem aplikace tak může být například soubor s tělesem, který chceme otevřít. Více ke skriptům a modulům Pythonu uvedeme v sekci 3.3.

Názvy přiložených souborů těles jsou tvořeny počtem rozměrů a počtem faset. (Například soubor `figures/3d-06.py` obsahuje krychli.)

Uživatelské rozhraní aplikace je inspirováno UNIXovým editorem Vi, který umožňuje efektivní, ale ne úplně intuitivní, ovládání.

Základní způsob ovládání poskytuje příkazový řádek umístěny v levém dolním rohu, který se otevírá klávesou `:`. Například integrovanou nápovědu k aplikaci tak otevřeme napsáním příkazu `:help`. Samotná aplikace poskytuje ovládání pouze tímto způsobem. Seznam příkazů uvedeme v sekci 3.2.2.

Veškeré přiřazení funkcí ostatním klávesám a myši je určeno konfiguračním souborem (příp. moduly). Nápovědu ke konfiguraci a modulům zobrazíme stiskem klávesy F1 (alespoň ve výchozím nastavení). Výchozí přiřazení kláves k jejich akcím popíšeme v následující podkapitole.

3.2.1 Výchozí konfigurace

Klávesám je většinou přiřazena funkce podle polohy na anglické klávesnici, namísto jejich popisu. Výchozí přiřazení funkcí klávesám a myši je následující:

Zobrazení nápovědy:

`?/F1`	nápověda ke konfiguraci a modulům	...poskytuje modul `helpMod`

Otevírání přiložených těles: (řada kláves s čísly)

`~`	bod (`0d.py`)
`‘`	úsečka (`1d-2.py`)
`1`	trojúhelník (`2d-3.py`)
`2`	čtyřstěn (`3d-04.py`)
`3`	krychle (`3d-06.py`)
`4`	osmistěn (`3d-08.py`)
`5`	dvanáctistěn (`3d-12.py`)
`6`	dvacetistěn (`3d-20.py`)
`7`	5-nadstěn (`4d-005.py`)
`8`	teserakt (`4d-008.py`)
`9`	16-nadstěn (`4d-016.py`)
`0`	24-nadstěn (`4d-024.py`)
`-`	120-nastěn (`4d-120.py`)
`=`	600-nadstěn (`4d-600.py`)
`_`	penterakt (`5d-010.py`)
`+`	5-ortoplex (`5d-032.py`)

Otáčením kolečka myši lze procházet tělesa ve složce s posledně otevřeným.

Otáčení tělesa: (levá část klávesnice)

`q/w`	rovina 12 – rotace ve směru / proti směru poloos 1(x) a 2(y)	
`a/s`	rovina 13 – poloosy 1(x) a 3(z)	
`z/x`	rovina 23 – poloosy 2(y) a 3(z)	
`e/r`	rovina 34	
`d/f`	rovina 14	
`c/v`	rovina 24	(s modifikátorem `Shift` skokově po 15°)

Pohybem myši se stisknutým levým tlačítkem otáčíme v rovinách 13 a 23.
Pohybem myši se stisknutým pravým tlačítkem otáčíme v rovinách 14 a 24.

`Tab`	náhodná rotace	...poskytuje modul `randomRot`

Změna parametrů perspektivy – vzdáleností kamery od počátku:
Pohybem myši se stisknutým kolečkem měníme vzdálenost v osách 3(z) a 4.

Posuny vrcholů: (pravá část klávesnice)

`n/p`	výběr předchozího/následujícího vrcholu
`N/P`	vytvoření nového / smazání aktivního vrcholu
`Esc`	zrušení výběru (obě klávesy)
`h/l`	posun v 1. ose (x)
`j/k`	posun ve 2. ose (y)
`u/i`	posun ve 3. ose (z)
`m/,`	posun ve 4. ose

Ostatní:

`y/t`	zapnutí/vypnutí udržování konvexního obalu
`g`	obnovení původního natočení tělesa
`b`	zrušení hranice tělesa
`F2`	konfiguračním souborem definované výchozí nastavení barev
`F3`	barvení podle třetí a čtvrté souřadnice
`F4`	barvení podle čtvrté souřadnice
`F9`–`F12`	velikost hran a vrcholů

Konfigurační soubor dále zavádí názvy barev `black`, `white`, `red`, `green`, `blue`, `yellow`, `cyan`, `purple`, `gray`, předchozí barvy doplněné čísly 9–1 pro různé stupně průhlednosti a absolutně průhlednou barvu `transparent`.

3.2.2 Příkazový řádek a seznam příkazů

Příkazy se překládají na výrazy Pythonu a ty se poté vykonávají. Mohou být složeny z několika částí, příp. po nich mohou následovat parametry; jako oddělovač používáme vždy jednu mezeru. Je-li mezera součástí parametru, po němž by mohly následovat ještě další, je možné parametr uzavřít do uvozovek. (Uvozovky v posledním parametru jsou jeho součástí.)

Číselné literály mají stejný formát jako v Pythonu (spec. používáme desetinnou tečku namísto čárky). Textové řetězce zapisujeme bez uvozovek (kromě případu v předchozím odstavci).

Barvy zapisujeme ve formátu `#RRGGBB`, příp. `#AARRGGBB` vč. průhlednosti, kde `AA`, `RR`, `GG`, `BB` jsou čísla šestnáctkové soustavy v rozsahu `00`–`FF` (na velikosti písmen nezáleží) a značí po řadě krytí (neprůhlednost), červenou, zelenou a modrou barevnou složku. Barvy můžeme také zapisovat jejich názvy definovanými v konfiguračním souboru.

Cesty v souborovém systému jsou systémem definované textové řetězce. Ty navíc můžou začínat `~/` pro domovský adresář nebo `%/` pro adresář se spustitelným souborem aplikace. Ve verzi pro Windows je možné používat normální i zpětná lomítka, ve verzi pro Linux pouze normální.

Po otevření příkazového řádku dvojtečkou můžeme používat následující klávesy: Doleva, doprava pro pohyb kurzoru na řádku. Nahoru, dolů pro procházení historie zadaných příkazů. Tabulátor pro přijetí navrhovaného dokončení příkazu (první stisk – přesune kurzor na konec) a pro procházení ostatních (další stisky). Backspace pro mazání znaků (příp. celého navrhovaného dokončení) před kurzorem, delete za kurzorem.

Automatické doplňování je k dispozici pro názvy příkazů, názvy barev a cesty v souborovém systému. Během psaní se nejprve objeví navrhované dokončení příkazu šedou barvou, v této chvíli ještě není součástí příkazu. Pro přijetí stiskneme jednou tabulátor, kurzor se přesune na konec a dokončení příkazu zmodrá; při vykonání příkazu v tomto okamžiku je doplněná část již jeho součástí. Dalšími stisky tabulátoru přepínáme mezi navrhovanými dokončeními, která jsou stále modrá. Jedním stiskem klávesy backspace smažeme celou doplněnou část; připsáním libovolného znaku se ztratí informace o tom, která část byla zadána a která doplněna.

Máme k dispozici následující příkazy aplikace a zásuvných modulů (moduly se budeme podrobněji zabývat v sekci 3.3):

Základní příkazy

`:help` [*příkaz*]
`:help module` *modul* ... poskytuje modul `helpMod`
`:help config` ... poskytuje modul `helpMod`

Zobrazuje nápovědu k příkazu, k modulu nebo k aktuální konfiguraci. Pro zobrazení nápovědy musí mít okno dostatečné rozměry. Název modulu musí být zapsán malými písmeny.

`:q[uit]`
`:exit`

Ukončí aplikaci.

`:n[ew]` *počet_rozměrů*

Vytvoří nový prostor daného počtu rozměrů (otevřené těleso se zruší).

`:o[pen]` *cesta*
`:so[urce]` *cesta*

Vykoná skript uložený v daném souboru (např. otevře těleso). Obě varianty mají aktuálně stejný význam, soubory těles jsou také skripty.

`:w[rite]` *cesta*

Uloží aktuálně otevřené těleso do souboru. Je-li dostupný modul `figureInfo`, uloží se také název a popis tělesa.

`:close`

Zruší aktuálně otevřené těleso i prostor.

`:info` ... poskytuje modul `figureInfo`

Zobrazí informace o otevřeném tělese. Kromě jeho názvu a popisu (jsou-li k dispozici) vypisuje také statistické údaje o počtu stěn.

`:is convex` ... poskytuje modul `check`

Zjistí, zda je otevřené těleso konvexní.

`:undo` [*počet_kroků*] ... poskytuje modul `snapshots`
`:redo` [*počet_kroků*] ... poskytuje modul `snapshots`

Spravuje historii otevřených těles a jejich změn a obnovuje předchozí (resp. následující) stav.

`:history`

Vypíše historii zadaných příkazů.

Rotace

`:reset rot[ation]`
Obnoví výchozí natočení tělesa (nastaví matici rotace na jednotkovou).

`:rot[ate]` *osa1* *osa2* *úhel*
Otočí těleso ve směru zadaných poloos o daný úhel ve stupních. Osy jsou značeny čísly od jedné. Osa 1 směřuje doprava, osa 2 nahoru, osa 3 a každá další ke kameře.

`:randomrot` ... poskytuje modul `randomRot`
Zapíná náhodné otáčení tělesa.

Práce s vrcholy

`:vert[ex] sel[ect]` [*index_vrcholu*]
Vypíše index aktuálně vybraného vrcholu, nebo vybere vrchol.

`:vert[ex] next`
Vybere následující vrchol.

`:vert[ex] prev[ious]`
Vybere předchozí vrchol.

`:vert[ex] desel[ect]`
Zruší výběr vrcholu.

`:vert[ex] add` [*souřadnice1* [*souřadnice2* [...]]]
Přidá vrchol na dané souřadnice, chybějící se doplní nulami. Souřadnice jsou relativní k aktuálnímu natočení tělesa.

`:vert[ex] move` *souřadnice1* [*souřadnice2* [...]]
Posune vrchol o dané souřadnice. (K aktuálním souřadnicím relativním k natočení přičte zadané.)

`:vert[ex]` (`rm`|`remove`)
Odstraní vybraný vrchol.

Úpravy tělesa

`:reset boundary`
Zruší všechna tělesa a ponechá jen jejich vrcholy. Je-li aktivní aktualizace konvexního obalu, dojde k jeho přepočítání.

`:create dual` ... poskytuje modul `duals`
Vytvoří geometricky duální těleso k otevřenému, to musí být konvexní.

`:cut vertices` [*poměr*] ... poskytuje modul `cuts`
`:cut edges` [*poměr*] ... poskytuje modul `cuts`
`:cut faces` *dimenze* [*poměr*] ... poskytuje modul `cuts`
Odřeže od aktuálního (konvexního) tělesa vrcholy, hrany, nebo stěny vyšší dimenze. Nadrovina řezu je vždy kolmá ke spojnici středu tělesa a dané stěny

(spojnice je kolmá ke stěně, nejde-li o vrchol) a její poloha je určená poměrem mezi vzdálenostmi stěny od nadroviny a stěny od středu tělesa. Střed tělesa zde znamená průměr jeho vrcholů. Výchozí poměr je 0,1.

`:stellate` ... poskytuje modul `stellation`

Provede na aktuálním konvexním tělese stelaci – zhvězdnatění (viz sekce 1.7).

`:cut off` [*poslední_souřadnice*] ... poskytuje modul `cuts`

Odřeže část tělesa. Nadrovina řezu je kolmá k poslední ose (v aktuálním natočení) a její souřadnice je dána parametrem. Odřezává se část blíže ke kameře. Není-li zadána pozice, nadrovina prochází počátkem (hodnota 0).

`:cut figure` [*poslední_souřadnice*] ... poskytuje modul `cuts`

Zobrazí průnik aktuálního tělesa s nadrovinou řezu (sníží dimenzi prostoru). Nadrovina je určená stejně jako u předchozího příkazu.

Nastavení

`:set`
`:set` *proměnná*[`?`]
`:set` [`no`]*proměnná*
`:set` *proměnná*=*hodnota*

Mění nastavení aplikace. První varianta vypíše dostupné volby a jejich hodnoty. Druhá varianta vypíše hodnotu zadané proměnné (pro logické hodnoty je vyžadován otazník). Třetí varianta nastavuje logické hodnoty a čtvrtá ostatní. Seznam voleb uvedeme v následující podkapitole (3.2.3).

`:map` *událost* [*příkaz*]

Přiřadí příkaz k události klávesnice nebo myši a zruší tím předchozí přiřazený příkaz.

Událostí může být stisk nebo uvolnění klávesové zkratky nebo tlačítka myši, tj. znak ASCII nebo `<`[`c`][`a`][`s`][`r`]`-`*klávesa*`>` , kde pomlčku předchází volitelné modifikátory Ctrl (`c`), Alt (`a`), Shift (`s`) a příznak uvolnění klávesy (`r`) – jinak je událostí stisk –; *klávesa* pak může být znak ASCII nebo `enter`, `esc`, `bs`, `delete`, `up`, `down`, `left`, `right`, `home`, `end`, `pageup`, `pagedown`, `insert`, `f1`...`f12`, `mouse0`...`mouse9`. Pořadí tlačítek myši (`mouse0`...`mouse9`) je určen knihovnou FreeGLUT, první tři jsou levé tlačítko, kolečko, pravé tlačítko. (Pootočení kolečka také patří k těmto událostem a vyvolává stisk a uvolnění krátce po sobě.)

Událost také může mít tvar `<`[`c`][`a`][`s`]`-mouse`[`0`][`1`]...[`9`](`x`|`y`)`>` , kde číslice představují stisknutá tlačítka myši (jako další modifikátory) a `x`, `y` osy myši. Musí být uvedena alespoň jedna číslice. Tato událost nastane při posunu myši v dané ose, jsou-li stisknuté dané modifikátory a tlačítka myši.

Jako *příkaz* můžeme uvést příkaz konzole nebo přímo výraz Pythonu. V případě, že je událostí pohyb myši, může příkaz obsahovat také zástupný znak `%`, za nějž se dosadí rozdíl nové a původní polohy kurzoru myši v dané ose.

`:rmap` *klávesa_nebo_osa_myši osa1 osa2*

Přiřadí klávese nebo ose myši rovinu (a směr) rotace. První parametr má stejný tvar, jako v předchozím příkazu, jenom je zakázán modifikátor `r`. Rotace probíhá po dobu stisku klávesy nebo při pohybu myši v dané ose.

`:randomrot map` [*událost*] … poskytuje modul `randomRot`

Určí událost klávesnice nebo myši pro spuštění náhodné rotace a následné přepínání režimů.

`:randomrot` [`no`]`auto` … poskytuje modul `randomRot`

Nastavuje, zda se má po otevření tělesa spustit náhodná rotace.

`:reset colors`

Obnoví výchozí nastavení barev.

`:snapshots maxcount`[`=`*hodnota*] … poskytuje modul `snapshots`

Nastavuje (vypisuje) maximální počet stavů aplikace v historii.

`:spacenavigator` … poskytuje modul `spaceNavigator`

Zobrazí aktuální stav modulu pro obsluhu 3D myši.

`:spacenavigator sensitivity`[`=`*hodnota*] … poskytuje modul `spaceNavigator`

Nastavuje (vypisuje) citlivost 3D myši.

`:spacenavigator device`[`=`*cesta*] … poskytuje modul `spaceNavigator`

Určuje cestu k souboru zařízení 3D myši.

`:spacenavigator` (`on`|`off`) … poskytuje modul `spaceNavigator`

Zapíná/vypíná 3D myš.

3.2.3 Seznam proměnných nastavení (příkaz `set`)

`background` – barva

Barva pozadí.

`campos`*osa* – číslo

Vzdálenost kamery od počátku souřadného systému pro každou iteraci perspektivy (viz sekce 1.2.1). Před aplikováním perspektivy se upraví velikost tělesa, aby jemu opsaná koule měla poloměr 1; vzdálenost kamery pak pracuje s těmito přeškálovanými souřadnicemi.

`camposl`*osa* – číslo

Předchozí proměnná v logaritmické stupnici. Lépe vystihuje přechod mezi perspektivou a rovnoběžným promítáním.

`convexhull` – logická hodnota

Automatická aktualizace konvexního obalu při změně tělesa.

Nastane-li chyba „`Error: more than two facets sharing ridge`“, nejspíš jde o problém s nepřesnou aritmetikou zmíněný v sekci 1.3.1 a bude potřeba mírně posunout některé vrcholy.

`dimen` – celé číslo, jen pro čtení

Počet rozměrů aktuálního prostoru. (Hodnota -1 znamená neexistenci prostoru.)

`edgesize` – číslo

Šířka hrany v pixelech. Vztahuje se k hraně ležící v rovině xy procházející počátkem. Nesmí být větší než velikost vrcholu. Velikosti vybraného (`selvertsize`) i nevybraného (`vertsize`) vrcholu můžou být změnou `edgesize` také změněny.

`facecolor` – barva s průhledností

Barva a průhlednost dvourozměrných stěn tělesa.

`grabmouse` – logická hodnota

Odchytávání myši při stisku tlačítka. Je-li zapnuto, kurzor se po stisku tlačítka myši skryje a při uvolnění se objeví na původním místě. Při pohybu myši pak nedochází k zastavení o okraj obrazovky.

`history` – celé nezáporné číslo

Počet zadaných příkazů, které se uchovávají v historii (jen do ukončení aplikace).

`maxfps` – celé kladné číslo

Maximální a požadovaný počet snímků za sekundu. Pro jeho dosažení může být snížen počet plošek, z nichž se skládají vykreslované vrcholy a hrany.

`mousesens` – číslo

Citlivost myši při otáčení tělesa ve stupních na pixel.

`pyexpr` – logická hodnota

Je-li zapnuto, do příkazového řádku je možné zadávat kromě příkazů také výrazy Pythonu.

`selvertcolor` – barva s průhledností

Barva vybraného vrcholu. Je-li průhledná, má vrchol stejnou barvu jako nevybraný.

`selvertsize` – nezáporné číslo

Velikost vybraného vrcholu v pixelech. Vztahuje se k vrcholu, který je umístěn v počátku souřadného systému. Nesmí být menší než velikost nevybraného vrcholu (`vertsize`) ani velikost hrany (`edgesize`), hodnoty těchto vlastností se proto můžou zmenšit při úpravě `selvertsize`.

Velikost má vliv na měřítko zobrazení (vše se musí vejít do jednotkové koule) a to i v případě, že žádný vrchol není vybraný; může tak dojít k nežádoucímu zmenšení tělesa například při zvětšení `versize` (zvětší i tuto proměnnou) a následném obnovení původní hodnoty (na `selvertsize` již nemá vliv).

`spacecolor` – barva
`spacecolor(+|-)`*osa* – barva s průhledností

Nastavuje barvy vrcholů a hran v závislosti na jejich poloze v prostoru, tímto jsme se zabývali v sekci 1.2.3 analýzy.

`speed` – nezáporné číslo

Úhlová rychlost otáčení při držení klávesy namapované pomocí `rmap` ve stupních za sekundu.

`stdoutpyexpr` – logická hodnota

Vypisování výrazů Pythonu vykonávaných příkazů na standardní výstup. Lze použít k vygenerování části konfiguračního souboru. V předkompilované verzi pro Windows není standardní výstup k dispozici.

`vertsize` – nezáporné číslo

Velikost vrcholu v pixelech. Vztahuje se k vrcholu, který je umístěn v počátku souřadného systému. Nesmí být menší než velikost hrany (`edgesize`) ani větší než velikost vybraného vrcholu (`selvertsize`), tyto hodnoty můžou být změnou `vertsize` ovlivněny.

3.3 Rozhraní Pythonu a zásuvné moduly

Po spuštění aplikace se nejprve importují všechny soubory s příponou `.py` ze složky `modules` do hlavního jmenného prostoru `__main__`, do něj se také importuje modul `gf`, představující rozhraní pro přístup k aplikaci. Poté se v hlavním jmenném prostoru provede skript `config.py` a případně i další skripty uvedené v argumentech aplikace. Stejně tak skripty spouštěné příkazy `:open` a `:source` i přeložené příkazy aplikace se provádí v hlavním jmenném prostoru.

Můžeme tedy rozlišovat dva typy souborů Pythonu: zásuvné *moduly*, umístěné ve složce `modules`, a ostatní *skripty*, zahrnující například konfigurační soubor a soubory těles. Ve skriptech můžeme využívat rozhraní modulů bez jejich importu a veškeré změněné proměnné budou přístupné i následně spouštěným skriptům a příkazům aplikace. V modulech je potřeba importovat všechny ostatní moduly, které budeme využívat.

Aplikací vytvářené soubory těles, i většina přiložených, jsou nezávislé na modulech; zpravidla využívají pouze modul `figureInfo` pro zobrazení názvu a popisu tělesa, ale soubory je možné otevřít i bez tohoto modulu. Podobně i mezi moduly existují závislosti a možnosti využití jiných modulů, jsou-li k dispozici; graf těchto vazeb je znázorněn na obrázku 2.2 ve vývojové dokumentaci. Tento princip naproti tomu není uplatněn v konfiguračním souboru, který na všech využívaných modulech závisí, a předpokládá se jeho případná úprava uživatelem. Aplikace samotná není závislá na žádných modulech.

Vyvolá-li funkce Pythonu volaná z aplikace výjimku, zpravidla dojde k vypsání jejího textu červeně na příkazovém řádku aplikace, což je preferovaný způsob oznamování chyb uživateli.

3.3.1 Přístup k aplikaci (modul `gf`)

Přistupovat k aplikaci je možné pouze z hlavního vlákna Pythonu.

Zde popisované funkce můžou vyvolávat výjimky. Ty jsou téměř vždy typu `RuntimeError` blíže specifikovaného pouze textem výjimky a nepředpokládá se jejich odchytávání.

Pokud funkce přijímá nebo vrací barvu, vždy jde o textový řetězec, který má formát `"#[AA]RRGGBB"` nebo je to zavedené jméno barvy. Pro převod barvy z/do použitelnějšího formátu lze použít funkce modulu `utils`.

Poloha bodu je n-tice souřadnic. Těleso je v této podkapitole struktura:

```
[
  [ (vrchol1_x, vrchol1_y, ...), ...],
  [ [hrana1_vrchol1, hrana1_vrchol2], ...],
  [ [stěna1_hrana1, stěna1_hrana2, ...], ...],
  ...,
  [ [těleso1_faseta1, těleso1_faseta2, ...], ...]
]
```

Struktura je seznamem s $(d + 1)$ prvky (pro d počet souřadnic), kde i-tý prvek (počítáno od nuly) obsahuje seznam těles dimenze i. Vrchol je d-tice souřadnic, těleso vyšší dimenze je seznam indexů jeho faset (těles nižší dimenze) v příslušném seznamu. Jde o obdobu struktury `figureData` používané v aplikaci (viz sekce 2.2.2).

Objektově orientovaný přístup k tělesu zajišťuje modul `objFigure`, který budeme využívat ve většině ostatních podkapitol této kapitoly. Zde definované těleso budeme proto pro odlišení označovat jako *těleso_gf*.

Přístup ke konzoli

`echo(`*text*`)`

Vypíše text na příkazový řádek. Při opakování se připisují nové řádky.

`echoErr(`*text*`)`

Vypíše text červeně.

`clear()`

Vymaže vypsaný text.

`clearAfterCmd(`[*text*]`)`

Vypíše jeden nový řádek s daným textem. Začne-li uživatel poté psát příkaz, je smazán pouze tento řádek (předchozí řádky jsou smazány až po potvrzení příkazu). Výchozím textem je „`Press any key or type command to continue`“.

`clearBeforePrinting()`

Zakáže připisování nových řádků. Před další vypisováním textu se obsah příkazového řádku smaže.

`printCentered(`*text*`)`

Vypíše blok textu na střed obrazovky (jako texty nápovědy).

Zavádění barev, příkazů a obsluhy událostí

`addColorAlias(`*jméno*`,` *barva*`)`

Zavádí nový název barvy. Barva může být jakákoliv aktuálně přípustná včetně dříve zavedených.

`resetColorAliases()`

Ruší veškeré dříve zavedené názvy barev.

`addCommand(`*příkaz*`,` *výraz_pythonu*`[,` *počet_parametrů*`[,` *typy_parametrů*`]])`

Zavádí nový příkaz konzole. Není-li uveden počet parametrů (nebo je nulový), přeloží se příkaz přímo na daný výraz Pythonu bez dalších parametrů.

Je-li počet parametrů nenulový, očekávají se ve výrazu zástupné znaky `%`, které budou při překladu nahrazeny parametry. Absolutní hodnota omezuje maximální počet parametrů příkazu, poslední parametr bude obsahovat celý zbytek uživatelem zadaného řetězce. Je-li hodnota kladná, bude za každý znak `%` dosazen jeden parametr; je-li záporná, bude za první znak `%` dosazen čárkami oddělený seznam všech parametrů.

Typy parametrů jsou určeny textovým řetězcem, kde každý znak udává typ jednoho parametru a poslední znak zároveň určuje typ všech zbývajících. Typy mají vliv na překlad i na funkci automatického dokončování. Jsou povoleny následující typy:

- `-` Literál Pythonu (výchozí). Ponechá se beze změny.
- `s` Textový řetězec. Obalí se uvozovkami.
- `p` Textový řetězec představující cestu v souborovém systému.
- `c` Textový řetězec představující barvu.
- `C` Textový řetězec představující barvu včetně průhlednosti.

`removeCommand(`[*prefix_příkazu*]`)`

Ruší všechny uživatelsky definované příkazy sdílející daný prefix; není-li uveden, ruší se všechny příkazy.

`registerCallback(`*název_události*`,` *funkce*`)`
`unregisterCallback(`*název_události*`,` *funkce*`)`

Registruje (resp. ruší) obslužnou funkci některé z následujících událostí. Pro zrušení je potřeba použít stejné parametry jako při registraci.

`"idle"()`

Nastává jednou v každém snímku, není-li otevřený příkazový řádek, aktivní funkce `gf.sleep` a neprobíhá-li aktualizace konvexního obalu.

`"new"()`

Nastává při změně počtu rozměrů prostoru, přesněji při každém volání `gf.new` a `gf.figureOpen` bez zachování natočení.

`"open"()`

Nastává při otevření nového tělesa, tedy při volání `gf.figureOpen` bez zachování natočení.

`"modified"()`

Nastává při změně otevřeného tělesa, tj. při volání `figureOpen` se zachováním natočení nebo při práci s vrcholy.

`"write"(`*cesta*`)`

Nastává při uložení aktuálního tělesa voláním `gf.write`. Umožňuje připsání dalších informací k tělesu při jeho ukládání.

`map(`*událost_hid*`[ , `*příkaz*`])`
`rmap(`*událost_hid*`[ , `*osa1*` , `*osa2*`])`

Zavádí/ruší přiřazení akce k události klávesnice nebo myši. Událost je textový řetězec popsaný u stejnojmenných příkazů.

Nastavení a další příkazy aplikace

`get_`*proměnná*`([`*index*`]) → `*hodnota*
`set_`*proměnná*`([`*index*`,] `*hodnota*`)`

Získá/nastaví hodnotu proměnné aplikace (jejich seznam byl popsán dříve). Index je vyžadován u proměnných, které mají na konci názvu číslo; to zde není součástí názvu, ale prvním parametrem.

`source(`*cesta*`)`
`open(`*cesta*`)`

Vykoná skript Pythonu.

`rotate(`*osa1*`, `*osa2*`, `*úhel*`)`

Otočí těleso o daný úhel ve stupních ve směru daných poloos.

`resetRotation()`

Obnoví výchozí natočení tělesa.

`resetColors()`

Obnoví výchozí nastavení barev.

`help(`*název*`)`

Vypíše stránku nápovědy.

`history()`

Vypíše historii zadaných příkazů.

`quit()`

Ukončí aplikaci.

Práce s tělesem a prostorem

`new(`*počet_rozměrů*`)`
`close()`

Vytváří prostor daného počtu rozměrů nebo ruší aktuální. Při volání `new` nastává stejnojmenná událost. Není potřeba volat při otevírání tělesa.

`figureGet() → `*těleso_gf*

Vrací aktuálně otevřené těleso, nebo `None`, neexistuje-li ani prostor.

`figureOpen(`*těleso_gf*`[ , `*zachovat_natočení*`])`

Otevře dané těleso. Je-li druhý parametr nastaven na `False` (výchozí hodnota), vytvoří se nový prostor a v něm se těleso otevře (nastávají události `new` a `open`).

Je-li druhý parametr nastaven na `True`, prostor včetně natočení tělesa se zachovává (nastává událost `modified`); nové těleso musí mít stejný počet souřadnic jako aktuálně otevřený prostor.

`figureConvexHullUpdate(`*těleso_gf*`)` → *těleso_gf*

Aktualizuje konvexní obal zadaného tělesa. Funkce může vyvolat výjimku, nastane-li při výpočtu chyba; to může být způsobeno například problémem zmíněným v sekci 1.3.1.

`write(`*cesta*`)`

Uloží aktuálně otevřené těleso do souboru. Po uložení je vyvolána událost `write`, která umožňuje ostatním modulům připsání dalších informací do souboru.

`resetBoundary()`

Ruší všechny stěny kromě vrcholů.

Práce s vrcholy

`vertexAdd(`[*pozice*]`)`

Přidává nový vrchol do počátku souřadného systému, nebo na zadané souřadnice.

`vertexRemove(index)`

Odebírá vrchol daným indexem.

`vertexDeselect()`
`vertexNext()`
`vertexPrevious()`
`vertexSelect(`*index*`)`
`vertexSelected()` → *index*

Mění (poslední funkce zjišťuje) vybraný vrchol. Není-li žádný vrchol vybraný, poslední funkce vrací hodnotu -1.

`vertexSetPos(`*index*`,` *nová_pozice*`)`
`vertexGetPos(`*index*`)` → *pozice*

Nastavuje/zjišťuje polohu vrcholu.

Různé

`expandPath(`*cesta*`)` → *expandovaná_cesta*

Expanduje na začátku cesty `~/` na domovský adresář a `%/` na adresář se spustitelným souborem aplikace a vrátí výsledek. Ve verzi pro Windows můžou být lomítka zpětná. Při automatickém dokončování cest jsou nabízeny i tyto prefixy.

`normalizeColor(`*barva*`)` → *normalizovaná_barva*
`normalizeColorAlpha(`*barva*`)` → *normalizovaná_barva_s_průhledností*

Převádí barvu na textový řetězec tvaru `#RRGGBB`, resp. `#AARRGGBB` u druhé varianty. Není-li barva rozpoznána, je vyvolána výjimka.

`posRotate(`*pozice*`)` → *transformovaná_pozice*
`posRotateBack(`*pozice*`)` → *transformovaná_pozice*

Transformuje souřadnice bodu do souřadného systému aktuálního natočení nebo zpět.

`time()` → *číslo*

Vrací čas běhu aplikace v milisekundách.

`sleep(`*ms*`)` → *logická_hodnota*

Přeruší vykonávání skriptu minimálně na zadanou dobu v milisekundách (může být i 0). Během vykonávání příkazu jsou obsluhovány události okna a minimálně jednou dojde k jeho překreslení. Stisk klávesy nebo tlačítka myši přeruší vykonávání funkce a ta poté vrátí `False`, skript by měl v tomto případě skončit co nejdříve; po ukončení skriptu je příslušná událost klávesnice nebo myši obsloužena. Nedojde-li k přerušení, vrací funkce `True`.

Tuto funkci je možné spolu s předchozí použít k vytváření animací. Pomocí funkce `time` je možné zajistit konstantní rychlost provádění animace, přestože funkce `sleep` negarantuje dobu, po kterou bude trvat její vykonávání. Kostra animace může vypadat například takto:

```
minDelay=1000/gf.get_maxfps()
time=gf.time()
while gf.sleep(minDelay):
  newTime=gf.time()
  elapsed=newTime-time
  time=newTime
  # Vykonání pohybu za posledních elapsed milisekund
```

Naproti tomu pro pravidelné činnosti na pozadí je doporučeno využívat událost `idle`, aby se střídavě mohl vykonávat i jiný kód. V průběhu funkce `sleep` (i vykonávání skriptů obecně) událost `idle` nenastává.

3.3.2 Modul `algebra` – prostředky lineární algebry

Vektor i bod je n-tice souřadnic. Báze je seznam bázových vektorů.

`dotProduct(`*vektor1*`,` *vektor2*`)` → *číslo*

Skalární součin dvou vstupních vektorů.

`vectLen(`*vektor*`)` → *číslo*

Euklidovská norma vektoru.

`vectDiff(`*vektor1*`,` *vektor2*`)` → *vektor*

Rozdíl dvou vektorů.

`vectSum(`*vektor1*`,` *vektor2*`,` ...`)` → *vektor*

Součet všech vstupních vektorů.

`vectMult(`*číslo*`,` *vektor*`)` → *vektor*

Vynásobení vektoru skalárem.

`vectAvg(`*vektor1*`, `*vektor2*`, ...)` → *vektor*

Průměr všech vstupních vektorů.

`pointsDist(`*bod1*`, `*bod2*`)` → *číslo*

Vzdálenost dvou bodů v euklidovské normě.

`orthogonalizeVect(`*vektor*`, `*ortonormální_báze*`)` → *vektor*

Ortogonalizace vektoru vzhledem k dané ortonormální bázi.

`orthonormalizeBasis(`*báze*`)` → *ortonormální_báze*

Ortogonalizace báze. Výstupem je seznam normalizovaných bázových vektorů, kterých může být i méně než vstupních (v případě, že byly lineárně závislé).

`orthonormalBasisFromPoints(`*seznam_bodů*`)` → *ortonormální_báze*

Vytvoří ortonormální bázi ze seznamu bodů v prostoru. Poloha afinního prostoru není ve výstupu zohledněna.

Modul dále definuje třídu `Hyperplane` reprezentující nadrovinu:

`Hyperplane(`*normálový_vektor*`, `*posun_nadroviny*`)`
`.normal`
`.normalPos`
`.orientedDistance(`*bod*`)` → *číslo*
`.inverse()` → *nadrovina*

Nadrovina je určena normálovým vektorem a posunem v jeho násobcích (součin vektoru a posunu udává polohu některého bodu nadroviny). Ve struktuře se normálový vektor (`normal`) ukládá normalizovaný na jednotkovou velikost a posun (`normalPos`) se také příslušným způsobem upraví. Orientovaná vzdálenost bodu od nadroviny (`orientedDistance`) je jeho vzdálenost ve směru normálového vektoru. Funkce `inverse` vrací stejnou nadrovinu s opačným směrem normálového vektoru (struktura se často využívá pro určení konkrétního poloprostoru).

`hyperplaneFromPoints(`*seznam_bodů*`[, `*kladný_bod*`])` → *nadrovina*

Vytváří nadrovinu obsahující daný seznam bodů. Nepovinným parametrem je bod mimo nadrovinu, jehož orientovaná vzdálenost má být kladná.

3.3.3 Modul `cuts` – řezy těles a odřezávání částí

Těleso je reprezentováno objektem `Figure` modulu `objFigure`. Nadrovina je objekt `Hyperplane` modulu `algebra`.

`cutFigure(`*těleso*`, `*nadrovina*`)` → `(`*záporné_části*`, `*řez*`, `*kladné_části*`)`

Provede řez tělesa danou nadrovinou. Vrací trojici seznamů těles vzniklých na řezu a na obou stranách nadroviny (kladné části jsou ve směru normálového vektoru, záporné proti). Pro konvexní těleso obsahuje každý seznam maximálně jeden prvek.

`cutOff(`*těleso*`, `*seznam_nadrovin*`[, `*zobrazit_průběh*`])` → *seznam_těles*

Odřízne části tělesa v poloprostorech určených danými nadrovinami ve směru jejich normály. Opakovaně využívá `cutFigure`, což může trvat dlouho; je-li posledním parametrem hodnota `True`, vypisuje se průběžně stav uživateli a ten má možnost vykonávání přerušit.

`cutOffConvex(`*těleso*`, `*seznam_nadrovin*`, `*vnitřní_bod*`)` → *seznam_těles*

Po rozřezání konvexního tělesa nadrovinami vrací tu jeho část, která obsahuje daný vnitřní bod; není-li zadaný bod uvnitř tělesa, může být vyvolána výjimka `spaceCuts.WrongAreaError`. Vrácený seznam obsahuje vždy právě jedno těleso. Tato funkce je často zaměnitelná s předchozí a díky využití převodu na reprezentaci tělesa průnikem poloprostorů (pomocí modulu `spaceCuts`) bývá výrazně rychlejší.

`cutOffFaces(`*těleso*`, `*poměr*`, `*seznam_stěn*`)` → *seznam_těles*

Odřeže dané stěny konvexního tělesa nadrovinami. Nadrovina řezu je vždy kolmá ke spojnici středu tělesa a dané stěny (spojnice je kolmá ke stěně, nejde-li o vrchol) a její poloha je určená poměrem mezi vzdálenostmi stěny od nadroviny a stěny od středu tělesa. Střed tělesa zde znamená průměr jeho vrcholů. Výstupní seznam je jednoprvkový.

`cutOffFacesDim(`*těleso*`, `*poměr*`, `*dimenze_stěn*`)` → *seznam_těles*

Odřeže všechny stěny dané dimenze.

3.3.4 Modul `duals` – geometricky duální tělesa

Těleso je reprezentováno objektem `Figure` modulu `objFigure`. Nadrovina je objekt `Hyperplane` modulu `algebra`.

`createDual(`*těleso*`[, `*střed*`])` → *těleso*

Vytvoří geometricky duální těleso k zadanému konvexnímu. Není-li uveden střed, uvažuje se počátek souřadnic. Definice duality byla uvedena v sekci 1.5 jako Definice 7.

`dualPointFromHyperplane(`*nadrovina*`[, `*střed*`])` → *bod*

Nalezne bod duální k dané nadrovině.

3.3.5 Modul `figureInfo` – správa metadat těles

Modul `figureInfo` spravuje název a popis aktuálně otevřeného tělesa. Při ukládání tělesa do souboru zajišťuje současné uložení aktuálních hodnot. Využívá také modul `snapshots` k obnovování předchozích názvů při návratu v historii otevřených těles. Dále modul umožňuje počítání stěn po dimenzích.

Těleso je zde opět reprezentováno objektem `Figure` modulu `objFigure`.

`setNameDescPath(`*název*`, `*popis*`[, `*cesta_k_souboru*`])`

Přiřazuje k otevřenému tělesu jeho název a popis; je-li tento příkaz součástí souboru s tělesem, lze uvést i cestu k tomuto souboru.

`getNameDesc()` → (*název*, *popis*)

Vrací aktuální název a popis tělesa. Popis může být `None`.

`getPath()` → *cesta*

Vrací název souboru otevřeného tělesa, je-li k dispozici; `None` jinak.

`counts(`*těleso*`)` → *seznam_počtů_stěn*

Vrací seznam obsahující počty stěn dané (indexem v seznamu) dimenze tělesa. Celé těleso je také započítáno.

`countsToStr(`*seznam_počtů_stěn*`)` → *textová_reprezentace*

Vrací text popisující počty stěn v tělese.

`printAll()`

Vypisuje všechny dostupné informace o tělese (stejně jako příkaz `:info`).

3.3.6 Modul `gfUtils` – další funkce pro přístup k aplikaci

Tento modul poskytuje pohodlnější rozhraní k některým funkcím modulu `gf`.

Těleso je zde, na rozdíl od modulu `gf`, reprezentováno objektem `Figure` modulu `objFigure`.

`openFileRelative(`*číslo*`)`

Otevírá těleso ve složce s aktuálně otevřeným (příp. ve složce `Figures`). Parametr udává pozici v seznamu souborů (s příponou `.py`) relativně k otevřenému; hodnota 1 tak znamená následující, hodnota -1 předchozí těleso.

`openConvexFromVertList(`*seznam_bodů*[, *název*[, *popis*[, *cesta*]]])

Vytváří a otevírá těleso, které je konvexním obalem daného seznamu bodů. Další parametry jsou předány modulu `figureInfo`.

`createConvexObjFigure(`*seznam_bodů*`)` → `těleso`

Vytváří těleso jako konvexní obal seznamu bodů.

3.3.7 Modul `helpMod` – nápověda k modulům a konfiguraci

`addPage(`*název*, *obsah*`)`

Zavádí novou obrazovku nápovědy dostupnou pod příkazem `:help` *název*. Obsah je (obvykle víceřádkový) textový řetězec.

`addModule(`*název_modulu*, *obsah*`)`

Zavádí novou obrazovku nápovědy `:help module` *název_modulu* a přidává název modulu do seznamu modulů na obrazovce nápovědy ke konfiguraci. Název modulu se zapisuje malými písmeny.

`printPage(`*název*`)`

Zobrazí uživatelsky definovanou obrazovku nápovědy s daným názvem.

`configAdd(`*text*`)`

Přidává řádky k obrazovce nápovědy ke konfiguraci `:help config`, dostupné pod klávesou F1.

`configPrint()`

Zobrazí obrazovku nápovědy k aktuální konfiguraci.

3.3.8 Modul `check` – ověřování vlastností těles

Těleso je reprezentováno objektem `Figure` modulu `objFigure`.

`isBoundaryConnected(`*těleso*`[, `*včetně_stěn*`]) →` *logická_hodnota*

Zjišťuje, zda je hranice tělesa souvislá. Druhý parametr určuje, zda se má ověřovat také souvislost stěn tělesa; výchozí hodnotou je `True`. Není-li hranice tělesa nebo některé jeho stěny souvislá, není jeho vykreslení aplikací podporováno (viz Definice 1).

`isBoundaryComplete(`*těleso*`[, `*včetně_stěn*`]) →` *logická_hodnota*

Ověřuje, zda je hranice tělesa dokončená, tedy platnost Lemmatu 2. Výchozí hodnotou druhého parametru je `True`.

`isFigureConvex(`*těleso*`) →` *logická_hodnota*

Zjišťuje, zda je těleso konvexní.

`findComponents(`*seznam_těles*`,` *fasety_k_vynechání*`) →` *seznam_seznamů_těles*

Rozdělí vstupní seznam těles na komponenty souvislosti. Všechna tělesa vstupního seznamu musí mít stejnou dimenzi. Aby byla tělesa považována na sousedící, musí sdílet celou fasetu, která se navíc nenachází v seznamu faset k vynechání. Sjednocením vrácených seznamů je původní vstupní seznam.

3.3.9 Modul `objFigure` – objektový přístup k tělesům

Tento modul poskytuje ostatním modulům objektovou reprezentaci tělesa (obdobu struktury `convexFig` v aplikaci). Strukturu tělesa modulu `gf` zde budeme explicitně označovat příslušným sufixem, ostatní tělesa jsou objekty zde definované třídy.

```
Figure([seznam_faset[, index_v_tělese_gf]])
  .addToBoundary(těleso)
  .rmFromBoundary(těleso)
  .boundary
  .dim
  .spaceDim
  .gfIndex
```

Objekt tělesa je tvořen především množinou jeho faset (`boundary`); fasety do ní přidáváme metodou `addToBoundary` a odebíráme je pomocí `rmFromBoundary`. Dimenze tělesa (`dim`) i počet souřadnic prostoru (`spaceDim`) se nastaví při přidání první fasety, která již má tyto hodnoty nastaveny. Položka `gfFigure` udává index

v příslušné struktuře tělesa modulu `gf` a bývá vyplněna pouze po konverzi z tohoto tělesa. Ostatní moduly můžou objektům těles přidávat libovolné další položky. Na tělese je definovaný iterátor, který prochází všechny jeho stěny včetně tělesa samého.

`Vertex(`*pozice*`[ , `*index_v_tělese_gf*`])`
`  .position`

Potomek předchozí třídy, představuje vrchol; ten je určen svou pozicí (`position`). Položky `dim` a `spaceDim` jsou vyplněny při vzniku.

`fromGfFigure(`*těleso_gf*`)` → *seznam_těles*

Převádí těleso modulu `gf` na objektovou reprezentaci. Těleso modulu `gf` mohlo ve skutečnosti obsahovat více disjunktních těles, jeho obdobou je proto seznam těchto těles. Vstupní hodnotě `None` odpovídá prázdný seznam.

`toGfFigure(`*seznam_těles*`)` → *těleso_gf*

Převádí seznam objektově reprezentovaných těles na strukturu tělesa modulu `gf`. Prázdnému vstupnímu seznamu odpovídá výstupní hodnota `None`.

`figuresIterator(`*seznam_těles*`)` → *iterátor*

Vrací iterátor přes všechna tělesa a jejich stěny, každé těleso právě jednou.

`updateParentsLists(`*těleso*`)`

Vytvoří u všech stěn tělesa (vč. jeho samého) položku `parents` tvořenou seznamem jejich rodičů (tj. stěn, které mají danou stěnu za svou fasetu).

`updateVerticesLists(`*těleso*`)`

Vytvoří u všech stěn tělesa (vč. jeho samého) položku `vertices` tvořenou seznamem všech vrcholů dané stěny.

3.3.10 Modul `randomRot` – náhodné rotace těles

`map(`*událost_hid*`)`

Určí událost klávesnice nebo myši pro spuštění náhodné rotace a následné přepínání režimů. Modul podle svého stavu mění přiřazení této události různým akcím a zobrazuje aktuálně zvolenou událost v jednořádkové nápovědě.

`set_auto(`*logická_hodnota*`)`

Nastavuje, zda se má po otevření tělesa spustit náhodná rotace. Tím je po otevření tělesa zablokována událost `idle` modulu `gf`.

`randomRot(`[*povolit_pohyb_kamery*]`)`

Zapíná náhodnou rotaci, výchozí hodnotou parametru je `False`. Funkce vrátí řízení až po ukončení rotace uživatelem.

3.3.11 Modul `snapshots` – historie stavů aplikace

Modul `snapshots` umožňuje ukládání vnitřních stavů ostatních modulů a jejich následné obnovení při volání příkazů `:undo` a `:redo`. Stav konkrétní součásti (modulu) je identifikován textovým řetězcem (např. názvem modulu). Stavem může být libovolný objekt Pythonu.

`undo(`[*počet_kroků*]`)`
`redo(`[*počet_kroků*]`)`

Obnovuje předchozí nebo následující stav aplikace a modulů.

`setState(`*název*`,` *stav*`)`

Ukládá aktuální stav součásti. Došlo-li během krátké doby (20 ms) k uložení stavu jiné součásti, dojde k seskupení těchto změn do jedné položky historie; jinak se vytvoří nová položka.

`getState(`*název*[`,` *index_položky_historie*]`)` → *stav*

Vrací aktuální stav součásti, příp. stav požadované položky historie. Neexistuje-li požadovaný stav, je vrácena hodnota `None`. Položky historie můžou být kdykoliv přečíslovány, použití druhého parametru dává smysl pouze v obslužné funkci události obnovení stavu.

`restoringInProgress()` → *logická_hodnota*

Zjišťuje, zda právě probíhá obnovování stavu. To je užitečné k zabránění uložení stavu během jeho obnovování v případě, kdy jsou tím vyvolány stejné postupy jako při přirozené změně stavu.

`registerCallbackCreate(`*funkce*`)`
`unregisterCallbackCreate(`*funkce*`)`

Registruje (resp. ruší registraci) obslužné funkce události vytvoření položky historie; této funkci nejsou předávány žádné parametry. Událost je užitečná pro uložení stavu součástí, které by samy neměly vytvářet položky historie, ale jejich stav je potřeba obnovovat. Z obslužné funkce se očekává volání `setState`, které tak nastane bezprostředně po uložení stavu jiné součásti a nezpůsobí vytvoření nové položky historie.

`registerCallbackRestore(`*funkce*`)`
`unregisterCallbackRestore(`*funkce*`)`

Registruje (resp. ruší registraci) obslužné funkce události obnovení stavu; funkci je parametrem předán index položky historie, která právě přestává být aktuální. Z obslužné funkce se očekává volání `getState`; to bez udání indexu vrátí stav, který chceme obnovit, a při předání právě získaného indexu stav, který přestává platit. Získaný index stavu je platný pouze během volání obslužné funkce.

`setClearableProperty(`*název*[`,` *logická_hodnota*]`)`

Nastavuje u dané součásti příznak, že má být její stav vymazán vždy při vytvoření nové položky historie (výchozí hodnota `True`). Bez tohoto příznaku se stav součásti přenáší i do dalších položek historie.

`get_maxcount()` → *hodnota*
`set_maxcount(`*hodnota*`)`

Vrací nebo nastavuje maximální uložený počet položek historie, výchozí hodnotou je 1000.

3.3.12 Modul `spaceCuts` – těleso jako průnik poloprostorů

Těleso je objekt `Figure` modulu `objFigure`. Nadrovina je objekt `Hyperplane` modulu `algebra`.

`figureFromArea(`*seznam_nadrovin*`,` *vnitřní_bod*`)` → *těleso*

Vytváří těleso obsahující daný vnitřní_bod, které je ohraničené danými nadrovinami; těleso je tedy průnikem příslušných poloprostorů. Na vzniklé těleso se také můžeme dívat jako na tu část prostoru po jeho rozřezání nadrovinami, která obsahuje daný bod. Není-li výsledná část prostoru omezená, je vyvolána výjimka `WrongAreaError` definovaná v tomto modulu.

`hyperplaneOfFacet(`*těleso*[`,` *kladný_bod*]`)` → *nadrovina*

Vrací nadrovinu generovanou daným tělesem; těleso tak musí mít o jednu dimenzi menší počet rozměrů než počet souřadnic. Druhým parametrem je libovolný bod na té straně nadroviny, kam má směřovat normálový vektor.

`hyperplanesOfFigure(`*těleso*`)` → *seznam_nadrovin*

Vrací seznam nadrovin všech faset tělesa. Těleso musí mít stejný počet rozměrů jako je počet souřadnic.

3.3.13 Modul `spaceNavigator` – obsluha 3D myši

Tento modul zajišťuje obsluhu 3D myši 3Dconnexion SpaceNavigator. Z implementačních důvodů modul není součástí verze aplikace pro Windows.

Pro zpřístupnění zařízení aplikaci je potřeba nastavit příslušná oprávnění například vytvořením souboru `/etc/udev/rules.d/90-spacenavigator.rules` s následujícím obsahem (vše na jednom řádku):

```
1 KERNEL=="event[0-9]*", ATTRS{idVendor}=="046d",
  ATTRS{idProduct}=="c62[68]", MODE="0664", GROUP="plugdev",
  SYMLINK+="input/spacenavigator"
```

Při následujícím připojení bude zařízení přístupné uživatelům skupiny `plugdev` prostřednictvím souboru `/dev/input/spacenavigator`, jehož existenci aplikace automaticky detekuje.

Je-li zařízení nalezeno ve výchozím umístění, dojde k deaktivaci automatických rotací modulu `randomRot` a k aktivaci 3D myši.

Osy zařízení pro otáčení obsluhují rotaci trojrozměrného tělesa, osy určené pro pohyb zde slouží k rotaci tělesa v osách 14, 24 a 34.

Rozhraní Pythonu:

`get_device()` → *cesta*
`set_device(`*cesta*`)`

Získá/nastaví cestu k zařízení.

`get_sens()` → *hodnota*
`set_sens(`*hodnota*`)`

Získá/nastaví citlivost zařízení.

`on()`
`off()`
`isActive()` → *logická_hodnota*

Aktivuje/deaktivuje zařízení, příp. zjistí stav. Po první aktivaci zařízení se vytvoří nové vlákno čekající na příchozí události; toto vlákno čte soubor i po deaktivaci.

`info()`

Zobrazí aktuální stav modulu.

`leftBtnFunc`
`rightBtnFunc`

Proměnné obsahující obslužné funkce, které jsou volány při stisku levého nebo pravého tlačítka zařízení, nebo hodnotu `None`. Funkcím nejsou předávány žádné parametry. Proměnné je možné kdykoliv měnit.

3.3.14 Modul `stellation` – stelace (zhvězdnatění) těles

Těleso je objekt `Figure` modulu `objFigure`.

`stellateFigure(`*těleso*`)` → *těleso*

Provádí stelaci konvexního tělesa (viz sekce 1.7).

3.3.15 Modul `utils` – pomocné funkce

Těleso je objekt `Figure` modulu `objFigure`. Barva je aplikací definovaný textový řetězec (název nebo `"#[AA]RRGGBB"`).

`colorToTuple(`*barva*`)` → `(`*krytí*`, `*červená*`, `*zelená*`, `*modrá*`)`

Převádí barvu na čtveřici celých čísel 0–255.

`tupleToColor((`*krytí*`, `*červená*`, `*zelená*`, `*modrá*`))` → *barva*

Převádí čtveřici celých čísel 0–255 na barvu.

`figuresMaxRadius(`*seznam_těles*`)` → *číslo*

Zjistí maximální vzdálenost vrcholu od počátku souřadného systému.

`figuresScale(`*seznam_těles*`, `*poměr*`)`

Upraví na místě velikost těles (s pevným bodem v počátku) v daném poměru.

Závěr

Aplikace Geometric Figures poskytuje s malými omezeními všechny funkce uvedené v zadání a úvodu této práce. Umožňuje tedy vykreslování vícerozměrných mnohostěnů, hledání konvexního obalu množiny bodů, stelaci těles, vytváření geometricky duálních těles, řezy tělesy a odřezávání jejich částí a podporuje zásuvné moduly Pythonu, umožňující dobrou rozšiřitelnost aplikace.

K ovládání je možné využít klávesnici a myš s nastavitelnými akcemi jejich událostí. Akcemi v tomto případě můžou být kromě funkcí aplikace také libovolné funkce Pythonu – v případě události pohybu myši i parametrizované změnou souřadnic. Integrovaný příkazový řádek je snadno rozšiřitelný o nové příkazy s podporou automatického dokončování příkazů i jejich parametrů představujících barvy nebo cesty v souborovém systému.

Díky možnosti periodického volání kódu Pythonu je možné pomocí skriptů nebo modulů vytvářet také animace bez předem daného průběhu. Příkladem těchto animací můžou být náhodné rotace modulu `randomRot` nebo obsluha 3D myši prostřednictvím modulu `spaceNavigator`. Ten navíc využívá možností Pythonu a operačního systému ke komunikaci se zařízením, což ilustruje možnost komunikace s jinými částmi systému.

Aplikace také umožňuje procházení historie těles a jejich změn během jednoho jejího spuštění (funkce Zpět a Vpřed). Tato funkcionalita je plně implementována v zásuvném modulu `snapshots`, k čemuž je využita možnost obsluhy některých událostí aplikace z Pythonu (např. otevření a uložení tělesa). Tyto události také umožňují modulu (`figureInfo`) při ukládání tělesa připsat do výsledného souboru další metadata, která se po otevření tělesa opět načtou do daného modulu.

K implementaci aplikace můžeme zmínit využití skriptů při jejím sestavování. Ty umožňují například automatické zpřístupnění označených funkcí jazyka C ze zásuvných modulů Pythonu (včetně příslušné typové konverze parametrů a návratových hodnot, viz 2.4.2) nebo začlenění textových souborů nápovědy do spustitelného souboru aplikace (viz 2.3.5).

Z omezení zmíněných v prvním odstavci je asi hlavním implementace podpory 3D myši pouze pro Linux. Ani bez tohoto omezení by ale tuto součást nejspíše nevyužívalo příliš mnoho uživatelů a jejím hlavním významem (jako i jiných modulů) je spíše demonstrace rozšiřitelnosti aplikace zásuvnými moduly. Modul je možné na Linuxu snadno upravit pro libovolné jiné vstupní zařízení a na Windows je možné využít část kódu komunikující s aplikací.

Další omezení spočívá v nepřesné aritmetice s reálnými čísly, jak je popsáno v sekci 1.3.1. Toto omezení může například způsobovat odmítnutí některých vstupů při generování konvexního obalu. Takovéto případy ale nastávají vzácně, nejsou-li cíleně vyhledávány.

Snadná rozšiřitelnost aplikace zásuvnými moduly poskytuje široké možnosti pro navázání na tuto práci. Pro konkrétní rozšíření je možné se nechat inspirovat funkcemi proprietární aplikace Stella4D (Webb, 9. 5. 2016) zmíněné v úvodu.

Co se týče rozšiřování samotného kódu aplikace, je možné například vylepšit vykreslování přidáním odlesků dvourozměrných stěn; to by vyžadovalo vyřešení problému s řazením stěn, který byl zmíněn v sekci 2.2.1.

Seznam použité literatury

Chazelle, B. (1993). An optimal convex hull algorithm in any fixed dimension. *Discrete & Computional Geometry*, **10**, 377–409.

FSF (2007). GNU General Public License. URL `http://www.gnu.org/licenses/gpl-3.0.html`.

Hladík, M. (15. 2. 2016). Lineární algebra (nejen) pro informatiky. URL `http://iuuk.mff.cuni.cz/~hladik/LA/text_la.pdf`.

Matoušek, J. (2002). *Lectures on Discrete Geometry*. Springer-Verlag, New York. ISBN 0-387-95374-4.

Seidel, R. (8. 5. 2016). Prezentace The Union-Find problem. URL `http://www-tcs.cs.uni-sb.de/media/lectures/W1516/L-AlgorithmsandDataStructures(BlockCourse)/Union-FindClassPresentationNew1.pdf`.

Stack Overflow (18. 5. 2016). format of /dev/input/event*? URL `http://stackoverflow.com/a/16682549`.

Tarjan, R. E. a van Leeuwen, J. (1984). Worst-case analysis of set union algorithms. *Journal of the ACM*, **31**(2), 245–281.

Webb, R. (9. 5. 2016). Webová stránka aplikace Stella4D. URL `http://www.software3d.com/Stella.php`.

Seznam obrázků