Geometric figures 1.3.3 – dokumentace

Lukáš Ondráček

Uživatelská příručka

Viz samostatný soubor.

Základní informace

Program je psaný v jazyce C, některé části využívají i skriptování v (G)Awk nebo v Perlu před kompilací; zásuvné moduly jsou psány v Pythonu. Postup k vytvoření spustitelné aplikace je popsaný v souboru `Makefile`.

Z knihoven se využívá OpenGL, GLU (OpenGL Utility Library), GLUT (OpenGL Utility Toolkit), Python 2.7 a standardní knihovny jazyka.

Zdrojový kód aplikace je členěn do modulů, jejichž exportované symboly mají obvykle název modulu jako prefix. Dokumentace se věnuje převážně kódu aplikace, rozhraní zásuvných modulů je popsáno v Uživatelské příručce a jejich podrobnější popis naleznete v poslední kapitole dokumentace. (Slovo *modul* se tedy odkazuje na moduly jazyka C, není-li uvedeno jinak.)

`main`
Inicializace programu, reakce na události.

`safe`
Ošetřování chyb, funkce pro kontrolu rozsahů hodnot,
zapouzdření funkcí pro alokaci paměti (není-li přidělena, aplikace se ukončí).

`util`
Různé užitečné funkce – expanze cest, exponenciální alokace paměti stringů, spojový seznam nad stringy, přenosné verze systémově závislých funkcí.

`matrix`
Maticové operace nad poli typu `GLfloat*` –
od skalárního součinu až po ortogonalizaci vektoru vzhledem k dané ortonormální bázi.

`figure`
Otevírání, ukládání, rotace, modifikace (vrcholů), ... tělesa, porovnávání reálných čísel s tolerancí.
Obsahuje datovou strukturu uchovávající dané těleso v jednoduchém formátu pro vykreslování.

`drawer`
Překreslování obrazovky, vykreslování těles.

`anim`
Časované události, plynulá rotace.

`hid`
Obsluha klávesnice a myši.

`console`
Obsluha příkazového řádku a vypisovaného textu.

`consoleCmds`
Překlad příkazů do Pythonu.

`consoleCmd`, `consoleCmdSet`, ...
Některé příkazy konzole.
(Také přístupné z Pythonu.)

`convex`
Modifikace hranice tělesa včetně udržování konvexního obalu a lámání stěn.
Soubor s tímto názvem obsahuje funkce pro synchronizaci s modulem `figure`.

`convexSpace`
Správa afinních prostorů.
K jednotlivým tělesům (všech rozměrů) přiřazuje ortonormální báze,
umožňuje počítání normálového vektoru jednoho prostoru v jiném
a následně orientovanou vzdálenost ve směru tohoto vektoru.
Pomocí binárního vyhledávacího stromu najde všechna existující tělesa generující daný afinní prostor.

`convexFig`, `convexFigMark`, `convexFigList` a `convexFigBst`
Správa topologie těles pro snadné modifikace (rodič má za děti svou hranici),
získání vrstvy dané dimenze, ...
Přiřazení značek jednotlivým tělesům.
Spojové seznamy nad tělesy a operace s nimi.
AVL strom řadící tělesa podle jejich afinního prostoru.

`convexHull` a `convexHullUtil`
Správa konvexního obalu celého načteného tělesa –
vytvoření, ověření a oprava, aktualizace po přidání, odebrání, nebo změně polohy vrcholu.
Funkce pro správu konvexního obalu libovolného tělesa.

`convexInteract`
Zajišťuje interakci s uživatelem během generování konvexního obalu –
zobrazuje průběh a umožňuje přerušení.

`convexLoopDetect`
Detekuje zacyklení vznikající při úpravě konvexního obalu
a pomocí `convexInteract` informuje uživatele o chybě.
(Všechny známé případy zacyklení ale již byly opraveny.)

`script`
Zajišťuje inicializaci a základní rozhraní pro práci s Pythonem z C –
provádění kódu, obsluhu výjimek, ...

`scriptVertex` a `scriptFigure`
Poskytuje rozhraní pro práci s otevřeným tělesem z Pythonu.

`scriptEvents`
Poskytuje možnost registrace callbacků v Pythonu.

`scriptWrappers` a `scriptWrappers.pl`
Automaticky zapouzdřuje funkce C pro volání z Pythonu.

`debug`
Koncentruje veškeré funkce potřebné pouze při ladění
a poskytuje makra pro výpis ladících informací přímo z kódu.
Je-li v hlavičkovém souboru vše deaktivováno, modul není potřeba linkovat.

`strings` a `stringsData.awk`
Spravuje texty nápovědy a jejich začlenění do spustitelného souboru aplikace.

Vykreslování a uživatelské rozhraní

Formát dat (`figure`)

Ve struktuře `figureData` je pro celá čísla použit datový typ `GLint` a pro reálná `GLfloat`, což by mělo zajistit jednotný formát datových souborů na všech platformách.

`figureData` obsahuje

(`GLint dim`) dimenzi prostoru,
(`GLint *count`) počty objektů pro každou dimenzi (vrcholů, hran, ...),
(`GLfloat **vertices`) pole souřadnic vrcholů,
(`GLint ***boundary`) pole všech hran, stěn, ...

První index `boundary` udává dimenzi (nultá položka nevyužita), druhý indexuje stěny dané dimenze, tím dostaneme pole indexů stěn o 1 menší dimenze indexované od 1, které stěnu ohraničují, nultá položka udává počet prvků.

Formát souboru:

```
GLint dimenze
GLint počet vrcholů
pro každý vrchol
    pro každou souřadnici
        GLfloat souřadnice
pro každou dimenzi od 1
    GLint počet stěn dané dimenze
    pro každou stěnu
        GLint počet stěn nižší dimenze, které ji ohraničují
        pro každou z nich
            GLint její index
```

`figureRotMatrix` je matice aktuálního natočení (původně jednotková), pro otočení tělesa se tato matice vždy pronásobí jinou maticí rotace. `figureScale` udává aktuální měřítko. Před vykreslením se souřadnice každého vrcholu pronásobí těmito dvěma proměnnými, čímž se získá těleso v aktuálním natočení v jednotkové kouli. (V `drawer` probíhají ještě další změny měřítka.)

`figureVerticesOfFaces` vrací pole indexů vrcholů 2D stěn pro vykreslení. Stěny, jejichž hranice není korektně uzavřena, se přeskočí. Přepočítávání probíhá pouze po změně tělesa.

Při úpravách vrcholů tělesa (`figureVertexAdd`, `figureVertexMove`, `figureVertexRm`) se pro aktualizaci jeho hranice volají funkce `convex`.

Vykreslování (`drawer`, `anim`)

K vykreslování vrcholů (`drawVert3D`) se používá funkce z knihovny GLU, hrany se vykreslují skládáním z trojúhelníků (`drawEdge3D`) pomocí OpenGL, pro vykreslení stěn (`drawFace3D`) se používá teselace z knihovny GLU, která podporuje i nekonvexní mnohoúhelníky.

Stěny se záměrně vykreslují pouze jako jednoduché filtry stejné barvy bez odlesků, protože jinak by bylo potřeba je předem seřadit, a protože se v projekci do 3D můžou i protínat, tak také najít jejich průsečíky.

Při perspektivní projekci (`drawFigure`, `matrixPerspective`) se upravuje měřítko tak, aby vždy byla vidět celá koule dané dimenze o poloměru 1.1, přičemž těleso se předem zmenší, aby se (včetně vykreslených vrcholů) vešlo do jednotkové koule.

Rotace je i u vícerozměrných těles řešena maticovým násobením.

Vykreslování textu probíhá pomocí knihovny GLUT, která umí vypisovat pouze znaky ASCII. (Háčky v uvítací zprávě složené z levého a pravého apostrofu proto nevypadají nejlépe.)

Při plynulém otáčení (`animFrame`) se měří skutečná doba vykreslení snímku pro zajištění konstantní rychlosti otáčení.

Modul `anim` obsahuje vlastní smyčku událostí (`animSleep`) volanou z Pythonu příkazem `gf.sleep`. Primární smyčku z knihovny GLUT totiž není možné volat rekurzivně. Při vykonávání příkazu `gf.sleep` tak může být mírně snížená odezva a zvýšené využití procesoru.

Obsluha klávesnice a myši (`hid`)

Obsluha klávesových zkratek je v GLUT trochu problematická. K dispozici je většinou znak ASCII, stejný jako v konzoli, a seznam modifikátorů; Ctrl+H tedy má jiný ASCII kód než H, ale stejný kód i modifikátory jako Ctrl+Backspace; díky modifikátorům navíc může dojít ke stisknutí jiné klávesy, než je později uvolněna. Mapování kláves se tak někdy může chovat podivně.

Všechny namapované události klávesnice i myši se uchovávají v poli (`mapped`) a všechna stisknutá tlačítka v seznamu (`mappedPressed`), který se smaže vždy, když se počet stisknutých kláves dostane na nulu (při uvolnění jiné klávesy, než byla stisknuta, se pouze sníží čítač).

Pokud je vyvolána událost při provádění `animSleep`, toto se přeruší (`animSleepInterrupt`) a čekající událost je zpracována později.

Je-li to potřeba, předávají je událostí klávesnice modulům `console` a `convexInteract`. Pro rotaci těles se využívá `anim`.

Příkazový řádek (`console`, `consoleCmds`, `consoleCmd`, ...)

Pokud je otevřen příkazový řádek, předávají se stisky kláves pomocí funkcí `consoleKeyPress`, `consoleBackspace`, `consoleEnter`, ...

Výstup je realizován přes `consolePrint`, `consolePrintErr`, `consolePrintBlock`, `consoleClear` a dalších. Standardně `consolePrint` připisuje další řádky pod sebe, ty jsou uloženy ve spojovém seznamu (`struct utilStrList`), stejně jako historie příkazů.

Veškerá práce s textem využívá alokace paměti pomocí `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. Funkce si ve statické proměnné uchovává seznam všech alokovaných stringů včetně jejich velikostí. Pokud se funkce zavolá s požadovanou velikostí 0, dojde k uvolnění paměti. Ke zmenšení nikdy nedochází. Funkce přijímá dva ukazatele, kde první je začátek bloku a druhý může být kdekoliv uvnitř, požadovaná velikost je uvažována vzhledem k druhému ukazateli (je-li zadán); v případě přesunu bloku jsou změněny oba ukazatele.

V modulu `consoleCmds` jsou uloženy všechny příkazy (vč. uživatelských) v trii, což umožňuje kromě snadného překladu do Pythonu (`consoleCmdsToScriptExpr`) také funkci automatického doplňování (`consoleCmdsComplete`). V trii je vždy uložen konstantní prefix příkazu, u něj pak informace k parametrům a výraz Pythonu s placeholdery. Podobně modul uchovává i jména barev a doplňuje také cesty.

Moduly prefixované `consoleCmd` (kromě `consoleCmds`) obsahují některé příkazy dostupné z příkazového řádku, k nimž je možné automaticky vygenerovat obalovací funkci Pythonu. (Funkce se složitějšími argumenty jsou v některých z modulů prefixovaných `script`.)

Texty nápovědy (`strings`, `stringsData.awk`)

Texty nápovědy jsou uloženy ve složce `src/stringsData/` a pomocí skriptu `stringsData.awk` se s kompresí posloupností mezer zkopírují jako konstanty do souboru `stringsData.c.tmp`. Všechny jsou tak uloženy přímo ve spustitelném souboru aplikace.

První řádek každého souboru obsahuje přípustné dvojice názvu sekce a textu ve formátu:

název_sekce:*název_textu*[;*název_sekce*:*název_textu*[;...]],

kde název sekce je zatím vždy `help` a názvem textu jsou všechny přípustné tvary příkazu. Podporovány jsou pouze znaky ASCII, kde většina řídících znaků má také přidělen nějaký symbol, BS vrací o znak zpět a LF láme řádek. Název souboru smí obsahovat pouze znaky, které je možné použít v názvu proměnné v C.

Nalezení správného textu, rozbalení mezer a vytvoření seznamu (`struct utilStrList`) řádků pro vypsání se provádí pomocí `stringsGet`.

Integrace Pythonu

Základní rozhraní přístupné aplikaci (`script`)

Modul `script` obsahuje vše potřebné pro většinu funkcí interagujících s Pythonem, včetně funkcí s automaticky generovanou obalovací funkcí Pythonu. Hlavičkový soubor `Python.h` tak ve většině modulů aplikace není potřeba.

Při inicializaci (`scriptInit`) importuje všechny zásuvné moduly ze složky modules i modul `gf` umožňující přístup k aplikaci.

Umožňuje jednoduché vyvolání (`scriptThrowException(char *str)`) i odchycení (`char *scriptCatchException()`) výjimek, volitelně i s vypsáním stack trace na chybový výstup (`scriptCatchExceptionAndPrint`). (Nevýhodou je, že všechny vyvolané výjimky jsou typu `RuntimeError`.)

Je možné vyhodnotit výraz (`scriptEvalExpr`) i vykonat skript (`scriptExecFile`).

Pro práci s Pythonem je potřeba získat jeho hlavní zámek (`scriptAcquireGIL`), pro umožnění běhu vedlejších vláken skriptů je potřeba jej naopak uvolnit (`scriptReleaseGIL`). Zámek není potřeba zamykat při používání funkcí tohoto modulu a ve funkcích volaných z Pythonu (ale například v `animSleep` je potřeba jej odemknout pro povolení vláken).

Automatické generování obalujících funkcí (`scriptWrappers`, `scriptWrappers.pl`)

O vytvoření seznamu všech funkcí přístupných z Pythonu (přes `gf`) a automatické generování potřebných obalujících funkcí se stará skript `scriptWrappers.pl` volaný před kompilací.

V hlavičkovém souboru na řádku s deklarací funkce, která má být přístupná z Pythonu, stačí uvést komentář `[SCRIPT_NAME:` *název_funkce*`]` a funkce bude pod tímto názvem ve jmenném prostoru `gf` přístupná. Povolené typy parametrů a návratové hodnoty jsou `char *`, `int`, `float` a `bool`. Byla přidána i podpora pro variabilní počet argumentů (popsáno v hlavičkovém souboru `script.h`), která ale nakonec nebyla využita. Pokud více deklarací uvádí stejný název funkce, zavolá se vždy ta první, jejíž signatura vyhovuje.

Vygenerovaná funkce Pythonu se postupně pokouší zkonvertovat argumenty pro dané funkce C a po zavolání zkonvertuje zpět návratovou hodnotu; příp. vyvolá výjimku, nebyla-li nalezena odpovídající signatura. Funkce C může vyvolat výjimku například pomocí `scriptThrowException`.

Pokud je funkce deklarovaná `extern PyObject *`*název*`(PyObject *self, PyObject *args)`, volá se z Pythonu přímo. Nefunguje u ní tedy ani přetěžování.

Přístup k tělesu (`scriptVertex`, `scriptFigure`)

Funkce modulů `scriptVertex` a `scriptFigure` jsou volány z Pythonu přímo (`scriptWrappers.pl` je pouze přidává do seznamu) a samy provádí konverzi parametrů a návratových hodnot, které tak můžou být strukturované.

Správa událostí a registrace callbacků (`scriptEvents`)

Jedná se o události vyvolané aplikací při vytvoření prostoru, otevření nebo úpravě tělesa a při nečinnosti. (O mapování kláves a os myši se stará modul `hid`.)

Tyto události jsou, hlavně kvůli častému volání `idle`, více provázány s Pythonem a namísto zpracování textových příkazů se přímo volají funkce Pythonu.

Funkce `scriptEventsRegisterCallback` a `scriptEventsUnregisterCallback` se přímo volají z Pythonu a u každé události udržují spojový seznam funkcí, které mají být volány.

Naproti tomu z aplikace se volají funkce pro okamžité vykonání callbacků (`scriptEventsPerform`) a pro jejich naplánování před příští `idle` (`scriptEventsInvoke`); první parametr těchto funkcí určuje událost, za ním následují parametry události (jde o funkce s variabilním počtem parametrů).

Pro vykonání naplánovaných událostí (následovaných `idle`) se používá funkce `scriptEventsSchedulePending`, která jejich vykonání naplánuje jako událost hlavní smyčky programu knihovny GLUT. Ta je volána mj. v každém snímku z modulu `anim`.

Úpravy topologie tělesa a generování konvexního obalu

(Převzato z předchozí dokumentace téměř beze změn.)

Formát dat v modulu convex (convexFig)

Struktura `convexFig` může představovat těleso libovolné dimenze (včetně vrcholů).

Každé takové těleso obsahuje seznam těles (`boundary`), které jej ohraničují, seznam těles (`parents`), do jejichž hranice patří, afinní prostor (`space`), který generuje, značky (`mark`), kterými je označeno, `hash` a index tělesa ve `figureData`, pokud tam existuje. Pokud jde o vrchol, hranice je prázdná a polohu je možné zjistit z bezrozměrného afinního prostoru.

Hash vrcholu je vždy náhodné číslo, hash ostatních těles je xor hashů vrcholů. Lze pomocí něj rychle vyloučit existenci tělesa daného seznamem vrcholů v seznamu těles (rodičů stěny, kterou by muselo obsahovat v hranici). Nepoužívá se žádná hashovací tabulka.

Pro vytváření a rušení těles se používají funkce `convexFigNew` a `convexFigDelete`, které nepotřebné proměnné uchovávají v seznamu, aby nebylo potřeba alokovat paměť příliš často. `convexFigDestroy` odpojí těleso od všech rodičů, stěn, i od afinního prostoru a uvolní jej; rekurzivně se zavolá na stěny, které již nemají rodiče, a naopak u svých rodičů nahlásí změnu.

`convexFigTouch` se používá k nahlášení změn. Nastaví index tělesa na -1, díky čemuž se těleso vezme v úvahu při synchronizaci s `figureData`, a rekurzivně se zavolá na rodiče.

Pro připojování hranice k tělesu slouží funkce `convexFigBoundaryAttach` a `convexFigBoundaryDetach`.

Důležitou funkcí je `convexFigGetLayer`, která rekurzivně prochází všechny rodiče nebo hranici tělesa a vrací seznam všech těles dané dimenze, na která narazí (předpokládá se konzistence – rodič má vždy o 1 vyšší dimenzi). Dimenzi je možné zjistit z příslušného afinního prostoru. Funkce používá značky (`convexFigMark`), aby se nezpracovávalo dvakrát stejné těleso, a umožňuje pomocí nich také filtrovat výsledky. Její složitost je lineární vzhledem k počtu všech hran v části grafu topologie tělesa, která se prochází.

Další nástroje pro správu topologie těles (convexFigList, convexFigMark)

`convexFigList` umožňuje vytváření a správu jednosměrného seznamu nad tělesy. K dispozici jsou funkce pro přidání (`convexFigListAdd`) a odebrání (`convexFigListRm`) prvku, pro odebrání (`convexFigListRmFig`) nebo spočítání (`convexFigListContains`) všech výskytů daného prvku, zničení (`convexFigListDestroy`), zkopírování (`convexFigListCopy`), přesunutí (`convexFigListMove`) nebo označení (`convexFigListMarkSet`, `convexFigListMarkClear`) celého seznamu, ...

Připadalo mi logické nejprve uvádět odkud se kopíruje a pak kam, takže u všech mých funkcí pro kopírování nebo přesouvání jsou argumenty v tomto pořadí; až později jsem si uvědomil, že je to naopak oproti funkcím standardní knihovny.

`convexFigMark` umožňuje přiřazovat tělesům různé značky a (obvykle v konstantním čase) je u všech těles vymazat. Pro každou značku je u každého tělesa číslo bez znaménka; pokud se rovná aktuálnímu pořadovému číslu, je těleso označeno. Vymazání všech značek (`convexFigMarkReset`) tedy znamená inkrementaci příslušného čítače, příp. vynulování značek u všech těles (`hardReset`), kdyby došlo k přetečení bezznaménkového `int`u. (Složitost `convexFigMarkReset` budu dále v dokumentaci považovat za konstantní.)

Afinní prostory (convexSpace, convexFigBst)

Každý afinní prostor (`convexSpace`) obsahuje počet rozměrů (`dim`), pozici libovolného bodu prostoru (`pos`), pokud je prostor přiřazen tělesu, jedná se o vnitřní vrchol, dále obsahuje ortonormální bázi (`ortBasis`), `hash`, příp. normálový vektor (`normal`) a posun v jeho směru vůči počátku (`normalPos`).

Je možné vytvořit bázi vrcholu (`convexSpaceCreateVert`) nebo jiného tělesa (`convexSpaceCreate`) a rozšiřovat ji dalšími vrcholy (`convexSpaceExpand`, `convexSpaceReexpand`).

Hotová báze se poté může přiřadit (zkopírovat) k tělesu (`convexSpaceAssign`, `convexSpaceUnassign`), čímž se také těleso přidá do binárního vyhledávacího stromu (`convexSpaces`) řazeného dle hashe afinního prostoru.

Pro výpočet ortonormální báze se využívá Gram-Schmidtovy ortogonalizace.

Hash afinního prostoru nesplňuje úplně definici hashe, protože podobné afinní prostory mají také podobný hash. Jedná se o (kolmou) vzdálenost afinního prostoru od předem náhodně zvoleného bodu jednotkové koule.

(Například celý prostor pak má hash 0.) K vyhledávání těles generujících daný afinní prostor se používá AVL strom (`convexFigBst`), který hashe porovnává s určitou tolerancí (`figureDistCmp`). U prostorů se správným hashem se pak zjišťuje příslušnost pozice a každého vektoru báze jednoho prostoru do druhého (`convexSpaceContains`, `convexSpaceEq`). Pokud by se ve zvolené toleranci nacházel pouze konstantní počet prostorů, byla by složitost vyhledání $\mathcal{O}(\log n + d^3)$, kde n je celkový počet prostorů ve stromě a d dimenze celého prostoru (počet souřadnic), s touto složitostí budu počítat v dalších odhadech.

Zvolení náhodného bodu jednotkové koule není úplně uniformní – nejprve se zvolí bod v krychli příslušné dimenze, jeho vzdálenost od počátku se poté normalizuje na náhodnou velikost od 0 do 1. Také poloměr koule není ideální, protože nejvzdálenější bod určuje měřítko a tím i toleranci porovnávání (při špatném měřítku se náhodný bod blíží počátku). Vrcholy je ale možné přidávat průběžně a proto náhodný bod musí být zvolen bez ohledu na jejich polohu.

Synchronizace s `figure` (`convex`)

Tělesa nejvyšší dimenze jsou uložena v seznamu `convexFigure`, volné vrcholy v `convexFreeVertices`.

Pro účely synchronizace jsou adresy těles uloženy také v poli `convexShadow`, kde indexy mají stejný význam jako u `figureData.boundary`, zde i včetně vrcholů. Dojde-li ke změně tělesa, nastaví se jeho index na -1 a příslušná položka v poli `convexShadow` se nastaví na 0 (`convexFigTouch`). Při exportu do `figureData` (`exportHull`) se aktualizuje jen to, co je potřeba.

Import se provádí jednou pomocí `convexAttach`, když se zjistí, že to bude potřeba (při změně tělesa nebo pro přepočítání obalu). Dále se veškeré změny vrcholů provádí synchronně (`convexVertexAdd`, `convexVertexRm`, `convexVertexMove`).

Správa konvexního obalu (`convexHull`, `convexHullUtil`)

Značení: *Těleso* může mít libovolný počet rozměrů d. *Vrchol*, *hrana* a *stěna* budou mít bez upřesnění v této části pevný počet rozměrů (0, 1 a 2). Protože je ale často potřeba pojmenovat útvary na hranici tělesa relativně k jeho dimenzi, označím $(d-1)$-stěnu jako *-stěnu-*, $(d-2)$-stěnu jako *-hranu-* a $(d-3)$-stěnu jako *-vrchol-*. Popis by tak měl být srozumitelný pro 3D tělesa s jednoznačným zobecněním pro jiné počty rozměrů. *Prostorem* je myšlen afinní prostor, může mít libovolný počet rozměrů.

`convexHullUtil` obsahuje funkce pracující s libovolným tělesem (prefix budu vynechávat), hlavní z nich jsou vzájemně rekurzivní funkce `-ExpandDim`, `-Expand`, `-Repair`, `-WrongFacesRm` a `-Complete`.

`convexHull` pak obsahuje funkce pro práci s celým tělesem `convexFigure` využívající `convexHullUtil`.

V odhadech časové složitosti budu používat d pro dimenzi prostoru (počet souřadnic) a n pro celkový počet všech těles všech dimenzí (počet prvků stromu convexSpaces).

`convexHullUtilExpandDim`

Vytvoří těleso z jedné jeho -stěny- a vrcholu ležícího mimo její prostor. Ostatní -stěny- vytvoří rekurzivním voláním na -hrany- a daný vrchol. (Jednorozměrné těleso vznikne triviálně spojením vrcholů.)

Pokud již těleso existuje (porovná se jeho hash a poté seznam vrcholů), nevytváří se znovu.

Pokud již existuje těleso generující požadovaný prostor, dojde k jeho rozšíření o nové vrcholy pomocí `-Expand`. (Kromě rozšířeného tělesa je v takovém případě vrácen i seznam těles vyšší dimenze s porušenou hranicí, která těleso obsahovala.)

Vrátí-li se z některého z rekurzivních volání větší -stěna-, než byla požadována, dojde k opravě tělesa i všech ostatních porušených těles pomocí `-Repair`.

V případě, že v žádné úrovni rekurzivního volání nedošlo k rozšíření ani opravě tělesa, je hrubý odhad složitosti $\mathcal{O}(m \cdot (m^2 + d^3 + \log n + r))$, kde m je počet těles v hranici vstupní -stěny- a r je počet jejich rodičů.

`convexHullUtilExpand`

Rozšíří těleso o nové vrcholy.

Nejprve se pomocí `-WrongFacesRm` odstraní -stěny-, které do nové hranice nepatří, přičemž se uvažují pouze nové vrcholy (těleso mělo před voláním `-Expand` konzistentní hranici).

Poté se pomocí `-Complete` přidají chybějící -stěny-.

Nakonec se pomocí `-Repair` opraví porušená tělesa stejné dimenze (sdílela některé -stěny-, které musely být rozšířeny) a kromě rozšířeného tělesa se vrátí také seznam porušených těles vyšší dimenze.

`convexHullUtilRepair`

Opraví porušenou hranici všech těles stejné dimenze, která dostane ve vstupním seznamu i která v průběhu poruší. Ve výstupním seznamu vrátí porušená tělesa vyšší dimenze.

U každého tělesa `-WrongFacesRm` odstraní nadbytečné -stěny- (zde se berou v úvahu všechny vrcholy tělesa) a `-Complete` přidá chybějící. To může vést k rozšíření vstupního seznamu porušených těles.

Porušená tělesa jsou označena značkou `convexFigMarkIdHullInconsistent`, aby se v seznamu neobjevila vícekrát.

`convexHullUtilWrongFacesRm`

Odstraní všechny -stěny- tělesa, k nimž existuje alespoň jeden vrchol (ze vstupního seznamu) s kladnou orientovanou vzdáleností od (prostoru) -stěny-. Vrací všechna porušená tělesa **stejného** počtu rozměrů.

Pro výpočet orientované vzdálenosti (`convexSpaceOrientedDist`) je potřeba, aby všechny -stěny- měly před voláním funkce vypočítány normálové vektory v prostoru tělesa (`convexSpaceNormalCalc`). Orientovaná vzdálenost je pak kladná, pokud se bod nachází v opačném poloprostoru určeném -stěnou- než zvolený vnitřní bod tělesa (z pohledu této -stěny- je tedy vně tělesa).

Nedojde-li k odebrání žádné -stěny-, je složitost $\mathcal{O}(svd)$, kde s je počet -stěn- a v délka vstupního seznamu vrcholů. Při odebrání -stěny- možná bude třeba odebrat i prvky její hranice, to je ale možné započítat u jejich vytvoření.

`convexHullUtilComplete`

Doplní všechny chybějící -stěny- vstupního tělesa, které má být konvexním obalem vstupního seznamu vrcholů. Vrací všechna porušená tělesa **stejného** počtu rozměrů.

Pokud těleso neobsahuje žádnou -stěnu- (všechny byly odstraněny předchozím voláním `-WrongFacesRm`), vytvoří se první -stěna- pomocí `-InitialFace`.

Vybere se libovolná -hrana-, která tvoří hranici právě jedné -stěny- tělesa. Na tuto -hranu- musí u uzavřené hranice navazovat ještě jedna -stěna-. (Pokud žádná taková -hrana- neexistuje, má těleso uzavřenou hranici.)

Najdou se všechny vrcholy, nacházející se v prostoru zvolené -stěny-, a pomocí `-Expand` se ověří, že -stěnu- není třeba rozšiřovat. Pokud byla rozšířena, pokračuje se znova výběrem -hrany-.

Prostor -hrany- se rozšíří libovolným vrcholem, který neleží ve -stěně-, a vypočítá se mu normálový vektor. (V nově vzniklém prostoru možná bude ležet nová -stěna-, jinak se bude prostor postupně kolem -hrany- otáčet.) Dále se prochází všechny vrcholy (neležící v prostoru existující -stěny-) a pokud má některý kladnou orientovanou vzdálenost, rozšíří se prostor -hrany- tímto. (Pro určení orientace se zde použije vnitřní bod -stěny-, která má -hranu- v hranici. Při použití libovolného vnitřního bodu tělesa by se nový prostor mohl kolem -hrany- otáčet špatným směrem.)

Nová -stěna- vznikne rozšířením -hrany- vybraným vrcholem pomocí `-ExpandDim`. Pokud v daném prostoru již nějaká -stěna- existovala a `-ExpandDim` vrátila větší -stěnu-, než byla potřeba, přidají se nově nalezené vrcholy do seznamu a znovu se provede odstranění nevhodných -stěn- pomocí `-WrongFacesRm`.

Pokračuje se výběrem další -hrany-.

Hledání -stěn- (vrcholů) hrany je řešeno zvlášť.

Složitost nalezení vhodné dvojice -hrany- a vrcholu, pokud nedojde k rozšiřování existující stěny, je $\mathcal{O}(m^2+vd^3)$, kde m je celkový počet těles v hranici vstupního tělesa a počet nejbližších rodičů a v počet vrcholů. Hrubý odhad celkové složitosti v případě, že nedojde k rozšiřování žádného tělesa, je $\mathcal{O}(m\cdot(m^2+vd^3+\log n))$.

`convexHullUtilInitialFace`

Vytvoří těleso s maximálně daným počtem rozměrů, které se bude nacházet v hranici tělesa libovolné vyšší dimenze obsahujícího pouze vrcholy ze vstupního seznamu.

Rekurzivně vytvoří takovéto těleso nižší dimenze, a pokud existuje vrchol, který neleží v jeho prostoru, rozšíří jej tímto vrcholem pomocí `-ExpandDim`. Pokud takovýchto vrcholů existuje více, vybere lexikograficky nejmenší.

Složitost je $\mathcal{O}(d\cdot v)$, kde d je požadovaná dimenze a v počet vrcholů vstupního seznamu.

`convexHullUtilCreate`

Vytvoří konvexní obal vstupního seznamu vrcholů.

Nejprve se pomocí `-InitialFace` vytvoří minimální (co do počtu vrcholů) těleso dané dimenze. To se následně rozšíří pomocí `-Expand`.

V případě, že žádné tři vrcholy nebudou v jedné přímce, žádné čtyři v jedné rovině, ..., akorát v prostoru generovaném všemi vrcholy nebude jejich počet omezen, tedy nedojde k žádným opravám a funkce `-Expand` bude

zavolána pouze jednou, bude hrubý odhad složitosti $\mathcal{O}(n^3 + vnd^3)$, kde v je celkový počet vrcholů a n celkový počet těles tvořících obal.

V ostatních případech by složitost měla být také polynomiální vzhledem k výstupu – na každé těleso se expanze (která provede změny) zavolá maximálně jednou a opravy nebudou potřeba (využijí se až při aktualizacích obalu).

`convexHullVertAdd`, `convexHullVertRm`, `convexHullCreate`, `convexHullUpdate`
Vytváří a udržují obal `convexFigure` pomocí funkcí `convexHullUtil`.

`convexHullBreakNearVert`
Rozlamuje tělesa incidentní s daným vrcholem.
Používá se pouze v případě, že je vypnuto hledání konvexního obalu (`noconvexhull`).
Seznam `convexFigure` může po volání funkce obsahovat více než jedno těleso. Při vypnutém hledání konvexního obalu také nedochází k aktualizacím afinních prostorů. Pro opětovné zapnutí hledání obalu je proto potřeba těleso znovu nahrát do modulu `convex` pomocí `convexAttach`.

Interakce s uživatelem během výpočtů (`convexInteract`)

Aktualizace hranice tělesa je ohraničena funkcemi `convexInteractStart` a `convexInteractStop`. Na některých místech v kódu aktualizace se volá funkce `convexInteractUpdate`, která aktualizuje vypisované informace a nechá zpracovat události, maximálně však jednou za 100ms. Aplikace má tak během výpočtu nižší odezvu, ale „nezasekne se“ a výpočet je možné kdykoliv přerušit stiskem libovolné klávesy.

Po přerušení výpočtu dojde k exportu aktuální části konvexního obalu.

Zacyklení (`convexLoopDetect`)

Odladění některých chyb se zdálo být nereálné a občas kvůli nim docházelo k zacyklení; nyní nevím o žádném případě, kdy by se tak stalo, ale pro jistotu zde tento modul nechávám.

Vždy při spojení nebo rozpojení tělesa a části jeho obalu se zavolá funkce `convexLoopDetectAction`, která porovná hash otce a syna s uloženou dvojicí. Pokud souhlasí, inkrementuje se u dvojice počet akcí (spojení, rozpojení), při přesáhnutí zvolené hranice se předpokládá, že došlo k zacyklení. Pokud se jedná o akci, jejíž pořadové číslo je mocnina dvojky, nahradí se uložená dvojice aktuální.

Takto se detekuje zacyklení v lineárním čase vzhledem k celkové době výpočtu (a velikosti periody).

Při detekování zacyklení dojde k přerušení výpočtu. Opakovaným nastavováním `convexhull` se může výpočet dokončit.

Zásuvné moduly Pythonu

Rozhraní zásuvných modulů je popsáno v uživatelské příručce a jejich implementace je většinou přímočará, takže zde uvedu jen několik výjimek.

Vytváření duálních těles (`duals`)

Vytváření duálních těles je z pohledu topologie jednoduché, stačí obrátit relace *„je v hranici“*/*„je ohraničováno“*.

Dále je potřeba určit souřadnice nově vzniklých vrcholů z faset původního tělesa, k tomu jsem použil postup popsaný na Wikipedii jako Polar reciprocation: Vrchol bude ležet na polopřímce směřující z počátku kolmo k fasetě a jeho vzdálenost od počátku bude převrácená hodnota vzdálenosti fasety od počátku.

Nevýhodou je, že takto lze vytvářet duální tělesa pouze ke konvexním tělesům.

Řezání těles (`cuts`)

Těleso se řeže vždy nadrovinou, u níž je známa normála a pozice nadroviny v jejím směru.

Vzdálenost bodu (vrcholu) od nadroviny se určí jako jeho skalární součin s normálou, přičemž znaménko výsledku udává, ve kterém poloprostoru se bod nachází. Takto se vrcholy rozdělí do dvou množin a podobně se určí poloha nových vrcholů na hranách mezi množinami. Dále se už pokračuje jen pomocí topologie.

Funkce `cutFigure` je popsaná rekurzivně. Zavolá se na prvky hranice vstupního tělesa čímž se získají (vynecháme-li triviální případy) hranice těles na obou stranách nadroviny kromě oddělující stěny a prvky hranice oddělující stěny. Prvky hranice oddělující stěny vznikly ze stěn zpracovávaného tělesa a tak by měly tvořit hranici tělesa na řezu. Obvykle by stačilo z nich vytvořit těleso (oddělující stěnu), tu přidat k hranicím těles na obou stranách a i z nich vytvořit tělesa; a nakonec všechna tři vrátit.

Problém je v tom, že těleso se může rozpadnout i na více než dvě části a v rovině řezu se tak může nacházet i více stěn, které je potřeba odlišit. Proto se ještě před vytvářením těles hledají na obou stranách nadroviny komponenty souvislosti (spojení v nadrovině řezu se ignorují) a ze společného podrozdělení vzniklých dělení těles na řezu se nakonec vytvoří stěny, které se přidají k příslušným komponentám na obou stranách.

Bez hledání komponent by složitost řezání byla $O(m + (v + e) \cdot d)$, kde m je velikost tělesa, tj. počet všech hran spojujících těleso s prvkem své hranice (o jednu dimenzi menším), v počet vrcholů, e počet hran a d dimenze prostoru. U zpracování hran a vrcholů se pracuje se souřadnicemi, u vyšších těles už pouze s jejich topologií.

Složitost hledání komponent bude $O(s_b \cdot (k + \log s))$, kde s je počet stěn v obou komponentách, k je počet komponent na jedné straně nadroviny a s_b je součet velikostí hranic (o jednu dimenzi menší) stěn v obou komponentách. Celková složitost pak bude $O(ms \cdot (k + \log s) + d \cdot (v + e))$, kde s je maximální počet stěn přes všechna tělesa (v hranici i hlavní těleso) a k je počet komponent, na něž se těleso po řezu rozpadne. Má-li řezané těleso stejnou dimenzi jako prostor, počet stěn musí být větší než dimenze a druhý člen je tedy možné zanedbat: $O(ms(k + \log s))$

V obou odhadech předpokládám, že přidání prvku do množiny, odebrání i rozhodnutí náležení je v Pythonu konstantní a procházení prvků v množině lineární.

Odřezávání vrcholů/hran/stěn (`cutOffFaces`) se provádí po jednom, pomocí předchozí funkce.

Jeho složitost tedy bude $O(N \cdot ms(k + \log s))$, kde N je celkový počet řezů, ostatní proměnné jsou popsány výše, a protože se pokaždé řeže jiné těleso, je potřeba uvážit maximum přes všechna.

Podpora přídavného polohovacího zařízení (`spaceNavigator`)

Modul `spaceNavigator` používá standardní rozhraní zařízení umístěných v `/dev/input/`, pro použití jiného polohovacího zařízení je tedy akorát potřeba upravit počet a indexy os a tlačítek.

Ze zařízení musí být možné číst, což vyžaduje vytvoření pravidla `udev`, jak je popsáno v souboru `spaceNavigator.py`. Uvedené pravidlo zároveň vytvoří symlink s pevným názvem.

Pro čtení informací ze zařízení se používá nové vlákno, které se spustí při prvním zavolání funkce `idle`; to poté nové hodnoty přičítá k proměnným, které `idle` čte a nuluje.