PŘÍRODOVĚDECKÁ FAKULTA UNIVERZITY PALACKÉHO
KATEDRA INFORMATIKY

DIPLOMOVÁ PRÁCE

Syntaktická kontrola konstrukcí v geometrii

2006 Přemysl Šrubař, Lukáš Zapletal

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval samostatně.

22. srpen 2006 Přemysl Šrubař, Lukáš Zapletal

Anotace

Cílem diplomové práce bylo navrhnout a vytvořit systém pro syntaktickou kontrolu v geometrických disciplínách. Zaměřili jsme se na deskriptivní geometrii a řešení úloh v Mongeově projekci. Systém by měl být dostatečně abstraktní, aby se dal snadno rozšířit na obecnou syntaktickou kontrolu libovolných úloh.

Děkujeme svým rodičům.

Obsah

Seznam obrázků

Seznam tabulek

1. Úvod

Problémem současných konstrukčních geometrických systémů je absence kontroly vytvářených konstrukcí. Tato kontrola může být prováděna jak analyticky, tak syntakticky. Při analytické kontrole systém zkonstruovaný výsledek ověří analyticky, tj. například dosazením do rovnic. U syntaktické kontroly program ověřuje celý postup konstrukce a je schopen předem rozhodnout, zda daný postup vede (přímou cestou), či nevede ke správnému výsledku.

Pokud bude analytická i syntaktická část takového systému vytvořena dostatečně obecně a jestliže se ukáže, že tento přístup vede k přijatelným výsledkům, bude jistě zajímavé pokusit se jej aplikovat i v jiných oblastech, jako je například řízení projektů. Může se také ovšem stát, že syntaktická kontrola bude výpočetně náročná a algoritmy budou mít nepřijatelnou složitost.

Planimetrie nám poslouží jako dobré testovací téma. Tato část geometrie je dostatečně prozkoumána a k dispozici je nám nespočet konstrukčních úloh. K dispozici je též dostatek kvalitní literatury. Hlavní uživatelská část se tedy bude odehrávat v Mongeově projekci.

Otázkou, kterou jsme museli ještě před zahájením prvotní analýzy vyřešit, byla úroveň kontrol. V podstatě máme tři možnosti, jak kontrolu geometrických postupů řešit. Nejjednodušší situace nastane, pokud se rozhodneme kontrolovat na úrovni Mongeovy projekce. Základní útvary zde budou pouze body a přímky, syntaktická kontrola však bude muset pokrývat větší počet možností.

Druhým případem, který jsme pro náš projekt zvolili, je obecná kontrola v třírozměrném prostoru. Modelované útvary v Mongeově projekci jsou promítány do prostoru, kde je provedena analytická a syntaktická kontrola. Hlavní výhodou je dostatečná obecnost kontrol a nezávislost na promítací metodě, což nám otevírá dveře pro další bádání, které bylo uvedeno v odstavci výše.

Třetí možností je hybridní řešení, které by používalo oba přístupy. K tomuto bychom se byli uchýlili v případě, že by realizace kontrol v prostoru nevedla k uspokojivým výsledkům. Bohužel jsme nenašli žádný referenční projekt, který by se zabýval možnostmi syntaktické kontroly, takže jsme museli stavět doslova na zelené louce.

1.1. Syntaktická kontrola

Problém syntaktické kontroly systémů spočívá ve stanovení vlastností a vztahů objektů a definování pravidel, za kterých se budou vztahy jiné odvozovat. Nejlepší bude vysvětlit tuto poněkud krkolomnou definici na příkladu.

Představme si úkol vytvořit takovou přímku a v prostoru, která bude procházet volně ležícím bodem X a zároveň bude kolmá na volně ležící rovinu α. U takto jednoduchého úkolu může řešitel onu přímku bez problémů zkonstruovat a prezentovat své řešení zadavateli. Ten je však na této úrovni schopen pouze

posoudit, zda-li je přímka zkonstruována správně po analytické stránce. Jinými slovy může ověřit kolmost na rovinu α a také jestli $X \subset a$.

Pokud budou v systému nějakým způsobem definovány vlastnosti a vztahy objektů, musí řešitel v konstrukci postupovat tak, aby byly vlastnosti zachovány, případně korektně odvozeny. Musí tedy vytvořit přímku procházející bodem X, systém tedy přidá do vlastností přímky fakt, že prochází bodem. Analogicky bod získá vlastnost, že jím prochází přímka. Podobně by to bylo s rovinou. Zadavatel pak má možnost ověřit nejen analytickou správnost problému, ale také syntaktickou: Náleží bod X přímce a? Je přímka a kolmá na rovinu α? Přesněji řečeno se zadavatel může ptát, zda-li má přímka a tu vlastnost, že je kolmá na rovinu α, aniž by musel vzít do ruky pravoúhlé pravítko a kolmost analyticky změřit.

Ačkoli se jedná o skutečně primitivní příklad, je na něm vidět výrazné obohacení možností kontroly řešení. Modelovaný systém je pak schopen vztahy mezi objekty odvozovat, a tím umožnit řešení složitějších problémů (konstrukcí). V našem případě můžeme jako příklad uvést následující odvozovací pravidlo:

$$\alpha \perp \beta \wedge \beta \perp \gamma \Rightarrow \alpha \| \gamma$$

Je známo, že člověk je schopen řešit dobře snadné úlohy. Podobně jako u softwarového inženýrství, kde se problém rozkládá na mnoho malých podproblémů, které se řeší lidem lépe a rychleji, se i při vytváření stromu konstrukce vyplatí určitá dekompozice. V tomto případě zavedeme takzvané podkonstrukce, ze kterých se bude celý strom skládat. Bude tak umožněno opakované využití podobně jako v programování (opakované využití kódu).

Jednotlivé konstrukce a podkonstrukce budou v systému napevno dány, ale systém je navržen tak, aby jej bylo možné rozšířit o vytváření a ukládání konstrukcí. Uživatel pak nebude omezen a bude možné realizovat libovolně složité konstrukce.

Vůbec nejtěžším úkolem byla syntaktická část systému, tedy implementace popisů konstrukcí, instancí konstrukcí a zejména pak odvozovacího systému nad vlastnostmi všech objektů. Všechny tyto pojmy budou vysvětleny v teoretické části této diplomové práce. Ačkoli jsme se snažili vytvořit účinné algoritmy na řešení všech prezentovaných problémů, zjistili jsme, že řada z nich má kvadratickou nebo dokonce exponenciální složitost. Je zde tedy velký prostor pro další bádání a implementaci efektivnějších postupů pro syntaktickou kontrolu.

Systém pro kontrolu syntaktických konstrukcí jsme pojmenovali jménem *Josephine*.

Kontakt na autory diplomové práce:
Přemysl Šrubař <psrubar@pikeelectronics.com>,
Lukáš Zapletal <lukas.zapletal@qcm.cz>.

2. Popis systému

Systém Josephine sestává z tří základních částí: analytická část, konstrukční část a část realizující kontroly.

2.1. Úroveň analytické geometrie

Všechny základní geometrické objekty mají určité analytické vyjádření. Zvolili jsme parametrické vyjádření, pro jeho obecnost.

2.1.1. Maticový zápis

Protože rovina závisí na dvou parametrech, přímka na jednom a bod na žádném, dá se parametrické vyjádření geometrického objektu zapsat pomocí matice 3×3. Aby se však daly použít transformační matice podle zažitých konvencí, je třeba přidat ještě jeden sloupec a řádek. Pro maticové analytické vyjádření základního geometrického objektu jsme tedy zvolili tento tvar:

$$\begin{array}{c} \\ \text{x} \\ \text{y} \\ \text{z} \\ \\ \end{array} \begin{array}{c} \begin{array}{cccc} u & v & 1 & \end{array} \\ \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

Přičemž bod má vždy první dva sloupce nulové a přímka má vždy nulový první, nebo druhý sloupec.

Příklad analytického vyjádření bodu:

$$A_{bod} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{array}{ll} x = \mathbf{0} \cdot u + \mathbf{0} \cdot v + \mathbf{1} \cdot 1 & x = 1 \\ y = \mathbf{0} \cdot u + \mathbf{0} \cdot v + \mathbf{2} \cdot 1 & y = 2 \\ z = \mathbf{0} \cdot u + \mathbf{0} \cdot v + \mathbf{3} \cdot 1 & z = 3 \end{array}$$

Příklad analytického vyjádření přímky:

$$A_{primka} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{array}{ll} x = \mathbf{0} \cdot u + \mathbf{1} \cdot v + \mathbf{0} \cdot 1 & x = v \\ y = \mathbf{0} \cdot u + \mathbf{0} \cdot v + \mathbf{2} \cdot 1 & y = 2 \\ z = \mathbf{0} \cdot u + \mathbf{0} \cdot v + \mathbf{0} \cdot 1 & z = 0 \end{array}$$

Zřejmě se jedná o přímku rovnoběžnou s osou x ležící v rovině xy.
Příklad analytického vyjádření roviny:

$$A_{rovina} = \begin{bmatrix} 4 & 0 & -1 & 0 \\ 0 & 3 & -2 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{array}{ll} x = \mathbf{4} \cdot u + \mathbf{0} \cdot v + \mathbf{\text{-}1} \cdot 1 & x = u - 1 \\ y = \mathbf{0} \cdot u + \mathbf{3} \cdot v + \mathbf{\text{-}2} \cdot 1 & y = v - 2 \\ z = \mathbf{0} \cdot u + \mathbf{0} \cdot v + \mathbf{\text{-}1} \cdot 1 & z = -1 \end{array}$$

Jedná se o rovinu rovnoběžnou s rovinou xy.

Mějme tedy analytické vyjádření nějakého základního geometrického objektu vyjádřené maticí A. Konkrétní bod příslušící geometrickému objektu získáme v závislosti na parametrech u a v takto:

$$\overline{(x,y,z,1)} = A \cdot \begin{bmatrix} u \\ v \\ 1 \\ 1 \end{bmatrix} \text{ což odpovídá: } \overline{(x,y,z,1)} = \overline{(u,v,1,1)} \cdot A$$

Kde první tři složky vektoru $\overline{(x,y,z,1)}$ jsou hledané souřadnice bodu, který náleží objektu vyjádřeného maticí A.

2.1.2. Transformace

Transformace geometrických objektů jsou realizovány pomocí transformačních matic.

$$T_{translace} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix} \quad T_{meritko} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{rotx} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(r_x) & \sin(r_x) & 0 \\ 0 & -\sin(r_x) & \cos(r_x) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{roty} = \begin{bmatrix} \cos(r_y) & 0 & \sin(r_y) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(r_y) & 0 & \cos(r_y) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{rotz} = \begin{bmatrix} \cos(r_z) & \sin(r_z) & 0 & 0 \\ -\sin(r_z) & \cos(r_z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Kde $t_{xyz}, s_{xyz}, r_{xyz}$ jsou hodnoty posunutí, změny měřítka a otočení podle os x, y, z.

Tyto základní transformační matice se po doplnění parametrů skládají pomocí operace násobení matic následovně:

$$T_{srt} = T_{meritko} \cdot T_{rotx} \cdot T_{roty} \cdot T_{rotz} \cdot T_{translace}$$

2.1.3. Báze

Takovéto transformace jsou však vztaženy pouze k základní bázi $\overline{(1,0,0)}, \overline{(0,1,0)}, \overline{(0,0,1)}$, což většinou nestačí. Pro transformování vzhledem k obecné bázi $\overline{a}, \overline{b}, \overline{c}$ (dále též pivot) je třeba nejdříve udělat transformaci $T_{depivot}$ takovou, aby

$$\begin{aligned}\overline{(1,0,0)} &= \overline{a} \cdot T_{depivot} \\ \overline{(0,1,0)} &= \overline{b} \cdot T_{depivot} \\ \overline{(0,0,1)} &= \overline{c} \cdot T_{depivot}\end{aligned}$$

Tedy abychom aplikací transformace $T_{depivot}$ na bázové vektory $\overline{(1,0,0)}, \overline{(0,1,0)}, \overline{(0,0,1)}$ obecné báze získali bázi základní.

Pokud bychom znali parametry základních transformací (míra posunutí, otočení, změna měřítka), kterými se základní báze transformuje na obecnou, tak bychom matici $T_{depivot}$ získali snadno pomocí výše uvedeného skládání transformací, ale v opačném pořadí a s opačnými hodnotami parametrů (u změny měřítka převrácenou hodnotou). V případě, že by obecná báze vycházela ze základní a veškerá změna její polohy by se realizovala transformacemi, bylo by možné uchovávat si parametry těchto transformací a použít je při hledání matice $T_{depivot}$. Tento postup však nelze obecně uplatnit, protože často potřebujeme bázi, která je určitým způsobem vázaná na nějaký geometrický objekt. Například pokud požadujeme, aby bod A vždy ležel na přímce a, očekáváme také, že posouváním tohoto bodu A ve směru osy x se bude bod posouvat po přímce a, ne ve směru globální osy x. Proto by měla mít báze svoji osu x vždy rovnoběžnou s přímkou a. Pro takovouto vázanou bázi tedy nelze přímo uchovávat parametry transformací a ani je nelze v obecném případě získat zprostředkovaně z parametrů transformace přímky a. Přímka a totiž může být určena například průnikem dvou rovin.

Použitelným řešením v tomto případě je získat nějakým způsobem transformační matici T_{pivot}, která transformuje základní bázi do báze obecné, tedy

$$\begin{aligned}\overline{(1,0,0)} \cdot T_{pivot} &= \overline{a} \\ \overline{(0,1,0)} \cdot T_{pivot} &= \overline{b} \\ \overline{(0,0,1)} \cdot T_{pivot} &= \overline{c}\end{aligned}$$

Kde $\overline{a}, \overline{b}, \overline{c}$ jsou bázové vektory obecné báze. Matice $T_{depivot}$ je pak inverzní maticí T_{pivot}. Z lineární algebry víme, že takovou matici přechodu získáme vyjádřením obecné báze $\overline{a}, \overline{b}, \overline{c}$ jako souřadnic vzhledem k základní bázi. A protože základní báze je $\overline{(1,0,0)}, \overline{(0,1,0)}, \overline{(0,0,1)}$, jsou tyto souřadnice přímo $\overline{a}, \overline{b}, \overline{c}$ a můžeme je tedy po řádkách zapsat do matice.

Bod transformovaného geometrického objektu se tedy získá takto:

$$\overline{(x,y,z,1)} = T^T \cdot A \cdot \begin{bmatrix} u \\ v \\ 1 \\ 1 \end{bmatrix}$$

kde

$$T = T_{pivot}^{-1} \cdot T_{srt} \cdot T_{pivot}$$

Pro každý geometrický objekt se udržuje hned několik matic. První matice představuje jakousi základní polohu, která se během existence objektu mění jen výjimečně. Je to maticový zápis podle výše uvedeného schématu. Další matice jsou matice statické a dynamické transformace. Obě jsou složením nějakých transformací včetně matic T_{pivot} a $T_{depivot}$. Statická transformace určuje jakousi pevnou polohu objektu, dynamická se používá, když se zrovna objekt transformuje. Když se zahájí transformování nějakého geometrického objektu, vznikne objekt transformátor, který nastavuje parametry transformace (posunutí, rotace, měřítko a pivot) dynamické transformace. Až se transformování ukončí nebo se zahájí znovu, vypočítá se maticový součin původní matice statické transformace s dynamickou a výsledná matice se uloží jako nová statická transformace. Celkové analytické vyjádření geometrického objektu vypadá následovně:

$$Analytic = (T_{static} \cdot T_{dynamic})^T \cdot A$$

kde A je maticový zápis geometrického objektu v základní poloze.

2.1.4. Deklarátory

Každý základní geometrický objekt je po vytvoření svázán s tzv. deklarátorem. Deklarátor je třída, která má tuto zodpovědnost:

- Určuje, jestli je analytické vyjádření správné (je splněna vazební podmínka).
- Vypočítá základní analytické vyjádření tak, aby bylo správné (je-li to možné).
- Určuje, jestli má objekt nějakou volnost v závislosti na svých nadřízených objektech (jestli lze objekt transformovat).
- Vytváří bázi (pivot) pro transformátor tak, aby transformace probíhaly podle očekávání.

Deklarátory, které jsou implementovány v systému, jsou uvedeny v tabulkách 1., 2., 3..

2.1.5. Knihovna MTJ

Analytická část hojně používá matice a operace s nimi. U transformací je to hlavně násobení matic, ale také hledání inverzní matice. Pro řešení průniků objektů je třeba zase řešit soustavy lineárních rovnic. Proto jsme pro práci s maticemi použili hotovou knihovnu. Zvolili jsme knihovnu „Matrix Toolkits for Java“, která je šířena pod open-source licencí.

Tato knihovna mimo jiné obsahuje:

Bod	
Třída	Popis
Free	Volně v prostoru
LiesInPlane	Musí ležet v rovině
LiesOnLine	Musí ležet na přímce
IntersectsLineLine	Musí ležet na průniku přímek

Tabulka 1. Deklarátory bodu

Přímka	
Třída	Popis
Free	Volně v prostoru
ContainsPoint	Musí procházet bodem
LiesInPlane	Musí ležet v rovině
InPlaneNormalsLineContPoint	V rovině, být kolmá na přímku a procházet bodem
ContainsPointNormalsPlane	Musí být kolmá na rovinu a procházet bodem
ContainsTwoPoints	Musí procházet dvěma body
IntersectsPlanePlane	Musí ležet ve dvou rovinách (jejich průnik)
LiesInPlaneContainsPoint	Musí ležet v rovině a procházet bodem

Tabulka 2. Deklarátory přímky

Rovina	
Třída	Popis
Free	Volně v prostoru
ContainsLine	Musí obsahovat přímku
ContainsLineLine	Musí obsahovat dvě přímky
NormalsPlaneContainsLine	Musí být kolmá na rovinu a obsahovat přímku
NormalsPlane	Musí být kolmá na rovinu

Tabulka 3. Deklarátory roviny

- Obecné rozhraní pro matice a vektory a základní operace s nimi.
- Algoritmy pro přímé řešení soustavy lineárních rovnic (na bázi LU-rozkladu).
- Podporu pro speciální typy matic, jako symetrické, pozitivně-definitní apod.
- Algoritmy pro numerické řešení soustav, hledání vlastních čísel a další.

Výhoda této knihovny spočívá také v tom, že se v podstatě jedná o adaptaci knihoven BLAS a LAPACK do jazyku Java. V případě, že by výkon nebyl dostačující, je možné přilinkovat verze těchto knihoven napsané přímo v jazyku C. Nám se však čistá verze, napsaná v jazyku Java, zdá dostatečně rychlá.

2.2. Úroveň konstrukcí

Základním požadavkem bylo, aby systém podporoval nejen základní geometrické objekty, ale aby bylo také možné určitým způsobem zachytit posloupnost jejich vytváření a případně i tuto posloupnost zopakovat na jiném místě. Tento požadavek plní právě konstrukce.

2.2.1. Základní pojmy

Klíčový bod je množina konstrukcí a podmínek, které tyto konstrukce musí splňovat. Konstrukce z klíčového bodu jsou navíc pojmenovány, mají přidělené symboly. Toto pojmenování je v klíčovém bodě jednoznačné. Podmínky jsou v podstatě formule predikátové logiky bez kvantifikátorů, ve speciálním normovaném tvaru. Přesným popisem relací, podmínek a odvozovacích pravidel se zabývá kapitola 2.3.

Krok je jednotka konstrukce. Výsledkem každého kroku je další konstrukce. Říkáme, že **krok deleguje** tuto výslednou konstrukci nebo také, že krok je **obalovým krokem** konstrukce. Krok má svůj vstup – **hlavičku**. Ta je určena svým klíčovým bodem. Vykonáním kroku vznikne delegovaná konstrukce.

Konstrukce je složení kroků. Každá konstrukce vždy obsahuje jeden speciální krok: **hlavičkový krok** (též návratový krok, *return*, *headerStep*). Vstup a výstup (ve formě klíčového bodu) tohoto kroku je stejný a definuje výsledek celé konstrukce. Proto název *hlavičkový krok*, protože určuje hlavičku celé konstrukce. Konstrukce není dokončena, dokud nemá vykonaný hlavičkový krok. Vstup hlavičkového kroku budeme označovat jako **hlavička konstrukce**.

Systém pracuje ve dvou úrovních abstrakce. Symbolická (popisná, meta) úroveň a úroveň instancí. Symbolická úroveň představuje jenom určitý popis, zatímco úroveň instancí je již konkrétní provedení. Z (meta) objektů na symbolické úrovni lze vytvářet nové objekty na úrovni instancí. Jedná se o obdobný vztah, jako je vztah třídy a objektu ve smyslu objektově orientovaného programování. Instance

Třída na meta úrovni	Třída na instanční úrovni	Popis
ConstrInfo	ConstrInstance	Konstrukce
StepInfo	UserStepInstance	Krok konstrukce
KeyPointInfo	KeyPointMatch	Klíčový bod
Relation	Prop	Vztah mezi konstrukcemi

Tabulka 4. Popisná a instanční úroveň

si uchovávají vazbu na svůj popis (info); obdoba reflexe v jazycích vyšší úrovně. V předchozím úvodu nebyly tyto úrovně záměrně odlišeny a nebudeme je rozlišovat ani dále, pokud to nebude nutné. Na 4. tabulce popisujeme třídy na jednotlivých úrovních. Pro úplnost jsou uvedeny i třídy, které budou popsány dále.

Popis konstrukce se tedy skládá z popisů kroků. Tyto popisy kroků nemají určené konkrétní pořadí, v jakém se mají vykonat. Pořadí, ve kterém se tyto kroky mohou vykonat (z popisu vznikne instance) je omezeno pouze vstupem kroku. Aby se mohl krok konstrukce vykonat, musí se k popisu klíčového bodu (*KeyPointInfo*) jeho vstupu najít vhodná instance klíčového bodu *KeyPointMatch*. Tj. navázat na jednotlivé symboly popisu klíčového bodu konkrétní instance konstrukcí tak, aby byla splněna všechna omezení daná klíčovým bodem. To je hlavně správný typ konstrukce (instance musela vzniknou podle popisu konstrukce, která je požadována pro konkrétní symbol) a po přiřazení všech symbolů musí být splněny i podmínky klíčového bodu.

Cesta P (*Path*) v popisu klíčového bodu K je trojice

$$[K, s, P_s],$$

kde K je popis klíčového bodu, s je symbol z K a P_s je cesta v hlavičce konstrukce C_s, určené symbolem s. P_s může být také nedefinováno, pak K nazveme listová cesta. Říkáme, že cesta vede ke konstrukci C_s. Listovou cestu je možné vytvořit přímo, nelistová se vytvoří z listové tak, že se určí symbol v hlavičce konstrukce C_s, jejíž hlavička je pak klíčovým bodem pro novou podcestu. Tímto způsobem vytváření je zajištěno, že nelze vytvořit cyklickou cestu.

Cestu $P_1 = [K_2, s_2, P_{s_2}]$ lze připojit k cestě $P_1 = [K_1, s_1, P_{s_1}]$ tak, že se nahradí podcesta P_{s_1} cesty K_1 podcestou P_{s_2} cesty K_2. Vznikne tak cesta $K_{1+2} = [K_1, s_1, P_{s_2}]$. Aby spojením nevznikla cyklická cesta, nahrazuje se ve skutečnosti podcesta P_{s_1} hloubkovou kopií podcesty P_{s_2}.

Cestu v klíčovém bodu budeme většinou zapisovat zkráceně. Listovou cestu $[K, s, \text{NULL}]$ zapíšeme jako pouhý symbol s. Nelistovou cestu $[K, s, P_s]$ pak $s/|P_s|$, kde $|P_s|$ znamená zkrácený zápis podcesty P_s.

Mapování (*PathMapping*) popisu klíčového bodu K_1 na popis klíčového bodu K_2 je kolekce dvojic (P_i^1, P_i^2), kde P_i^1 je cesta v klíčovém bodu K_1, nazýváme ji zdrojová, a P_i^2 je cesta v klíčovém bodu K_2, nazýváme ji cílová. Přitom každá

cílová cesta musí být listová. Většinou také požadujeme, aby mapování bylo úplné, tj. aby ke každému symbolu z K_2 vedla nějaká cílová cesta.

2.2.2. Příklad klíčového bodu a cesty

Popis klíčového bodu budeme zapisovat tak, že do řádků rozepíšeme jednotlivé požadavky klíčového bodu. Každý požadavek pak zapíšeme ve tvaru:

$$s_k\text{:}\textit{název popisu konstrukce},$$

kde s_k je k-tý symbol v klíčovém bodu. Prozatím vynecháme seznamy podmínek jednotlivých požadavků, kterými se zabývá kapitola 2.3.
První příklad:
O:*3D.Point*
π:*3D.Plane*
ν:*3D.Plane*
x:*3D.Line*
Označme tento popis klíčového bodu jako K_{mong}. Tento klíčový bod je hlavičkou konstrukce $Mong.CORE.MongProjection$, což je konstrukce samotného Mongeova promítání. Tento klíčový bod se skládá ze čtyř požadavků. Požaduje:

- Bod, který se označí symbolem O (počátek souřadnicového systému).
- Rovinu, která se označí symbolem π (první průmětna).
- Rovinu, která se označí symbolem ν (druhá průmětna).
- Přímku, která se označí symbolem x (osa x).

Všimněme si, že tento klíčový bod požaduje pouze primitivní konstrukce.
Příklad klíčového bodu, složeného nejen z primitivních konstrukcí:
pr_1:*3D.Plane*
pr_2:*3D.Plane*
a_1:*3D.Line*
a_2:*3D.Line*
a:*3D.Line*
mong:*Mong.CORE.MongProjection*
Označme tento popis klíčového bodu jako K_{mLine} Skládá se z těchto požadavků. Požaduje:

- Rovinu, která se označí symbolem pr_1 (první promítací rovina).
- Rovinu, která se označí symbolem pr_2 (druhá promítací rovina).
- Přímku, která se označí symbolem a_1 (půdorys přímky).
- Přímku, která se označí symbolem a_2 (nárys přímky).

- Přímku, která se označí symbolem a (promítaná přímka v prostoru).
- Konstrukci podle popisu $Mong.CORE.MongProjection$ označenou $mong$.

Tento klíčový bod je hlavičkou konstrukce $Mong.CORE.MongLine$. Jedná se o konstrukci přímky v Mongeově promítání.
Příklad listové cesty v klíčovém bodě:

$$\begin{aligned} P^1_\pi &= [K_{mong}, \pi, \text{NULL}] \\ &= [(O{:}\textit{3D.Point } \pi{:}\textit{3D.Plane } \nu{:}\textit{3D.Plane } x{:}\textit{3D.Line}), \pi, \text{NULL}] \end{aligned}$$

Tato cesta je listová a vede k popisu roviny π Mong. projekce. Zkrácený zápis představuje pouhý symbol π.
Příklad nelistové cesty:

$$\begin{aligned} P^2_\pi &= [K_{mLine}, mLine, P^1_\pi] \\ &= [K_{mLine}, mLine, [K_{mong}, \pi, \text{NULL}]] \\ &= [K_{mLine}, mLine, [(O{:}\textit{3D.Point } \pi{:}\textit{3D.Plane } \nu{:}\textit{3D.Plane } x{:}\textit{3D.Line}), \pi, \text{NULL}]] \end{aligned}$$

Tato cesta vede opět k rovině π Mong. projekce, ale tentokrát z hlavičky konstrukce přímky v Mong. projekci. Zkrácený zápis: $mLine/\pi$.

2.2.3. Konstrukce a její kroky

Nyní můžeme upřesnit, z čeho se skládá krok konstrukce:

- Vstup, je určen popisem klíčového bodu.
- Výstup, kterým je delegovaná konstrukce.
- První krok delegované konstrukce (delegovaný krok).
- Úplné mapování ze vstupu na hlavičku delegovaného kroku.
- Kolekce doporučených pokračování (uvádíme dále).

Každý krok konstrukce má kolekci doporučených pokračování. Doporučené pokračování (*StepSuggestion*) se skládá z:

- Odkazu na popis doporučeného kroku stejné konstrukce (kterým krokem se má pokračovat).
- Kolekce vstupních kroků, které už musely být vykonány, aby šlo pokračovat podle tohoto doporučení.

- Úplné mapování z hlaviček vstupních kroků na hlavičku doporučeného kroku. Tímto mapováním se poskládá vstup pro doporučený krok.

Kromě kolekce doporučených pokračování u kroku má konstrukce také kolekci doporučených zahájení. Doporučené zahájení je pouze kolekce popisů kroků, kterými se doporučuje zahájit konstrukci. Nemá žádné mapování, protože před zahájením konstrukce ještě nebyl vykonán žádný její krok.

2.2.4. Primitivní konstrukce

Primitivní konstrukce představují jakýsi most mezi analytickou úrovní (základní geometrické objekty) a úrovní konstrukcí. Primitivní konstrukce jsou pevnou součástí systému a mají speciální podporu, hlavně při vytváření. Primitivní konstrukce tedy jsou:

- konstrukce bodu – *PointConstrInfo*
- konstrukce přímky – *LineConstrInfo*
- konstrukce roviny – *PlaneConstrInfo*

Každá instance primitivní konstrukce má vazbu na příslušný geometrický objekt a stejně i geometrický objekt má vazbu na svoji primitivní konstrukci. Popisy kroků primitivních konstrukcí souvisí s deklarátory jednotlivých geometrických objektů, jak je popsáno v kapitole 2.1. Tyto kroky mají jako svůj vstup jiné primitivní konstrukce a jako výstup vždy jednu příslušnou primitivní konstrukci. Některé kroky primitivních konstrukcí nepotřebují žádný vstup, jejich hlavička je prázdný klíčový bod. Jako každá konstrukce i primitivní konstrukce má speciální hlavičkový krok. Tento krok má stejný vstup (hlavičku) i výstup, a tím je konstrukce, které tento krok náleží.

Vykonáním nehlavičkového kroku primitivní konstrukce vznikne geometrický objekt. Nejdříve se vstup toho kroku (*KeyPointMatch*) převede z instancí konstrukcí na geometrické objekty. To je možné díky již zmíněné vzájemné vazbě mezi nimi. S pomocí těchto geometrických objektů se vytvoří nejdříve deklarátor a teprve z deklarátoru se vytvoří geometrický objekt. Za samotné vytváření geometrických objektů je zodpovědná továrna (*factory*) na vytváření geometrických objektů (*GeomObjectFactory*). Kroky primitivních konstrukcí nepoužívají mapování vstupu, protože nedelegují žádnou další konstrukci (přesněji, delegují svoji vlastní konstrukci).

Vykonáním nehlavičkového kroku tedy nevzniká instance konstrukce, ale pouze geometrický objekt. Aby vznikla primitivní konstrukce, je třeba ještě vykonat hlavičkový krok. Ten má však (u primitivních konstrukcí) v popisu jako vstup i výstup popis své vlastní konstrukce. Hlavičkový krok by tedy nebylo možné nikdy vykonat a nebylo by tedy ani možné dokončit primitivní konstrukci

(za předpokladu, že ještě neexistují žádné jiné instance primitivních konstrukcí). Proto mají primitivní konstrukce ještě další specifickou vlastnost: po vykonání jakéhokoli nehlavičkového kroku se automaticky vykoná i hlavičkový krok, a tím se konstrukce dokončí.

Na množině všech instancí primitivních konstrukcí jsou také definovány relace, které popisují syntaktické vztahy mezi primitivními konstrukcemi. Každá instance primitivní konstrukce si uchovává kolekci objektů *Prop*, které vyjadřují skutečnost, že konstrukce patří do příslušné relace. Podrobnosti rozebírá kapitola 2.3.

2.2.5. Složené konstrukce

Složené konstrukce umožňují zachytit určitý složitější postup vytváření, než umožňují primitivní konstrukce.

Každý krok složené konstrukce deleguje jinou konstrukci (primitivní i složenou). Tato delegovaná konstrukce je výstupem kroku. Krok nabízí také tento výstup ve formě klíčového bodu, který obsahuje pouze jeden symbol (návratový symbol), na nějž je navázána tato delegovaná konstrukce. Krok má navíc odkaz na krok delegované konstrukce, který se má vykonat jako první. Budeme ho označovat jako delegovaný krok. Delegovaná konstrukce tedy nemusí nutně začínat svým doporučeným zahájením, ale krok může zahájit svoji delegovanou konstrukci libovolným krokem, ke kterému existuje úplné mapování ze vstupu kroku na vstup delegovaného kroku.

Pro vykonání kroku složené konstrukce je třeba najít vstup pro tento krok (*KeyPointMatch*). Vstup se pomocí mapování převede na vstup pro delegovaný krok. Pak se delegovaný krok vykoná. Vykonáním tohoto delegovaného kroku se však ještě delegovaná konstrukce nedokončí, jedině že by delegoval přímo hlavičkový krok (nebo by šlo o primitivní konstrukci). Proto je třeba delegovanou konstrukci dokončit. Dokončení konstrukce spočívá v postupném vykonání jejích kroků podle doporučeného pokračování. Přitom se začne od naposledy vykonaného kroku (tím byl právě onen delegovaný). Pokud je konstrukce prázdná, tj. nebyl vykonán žádný krok, začne se krokem z doporučeného zahájení konstrukce.

Jak již bylo řečeno, krok složené konstrukce může delegovat libovolný krok jiné konstrukce. To například umožňuje udělat část konstrukce nějakým nepopsaným způsobem (nepopsanými vytvořenými popisy konstrukcí a kroků) a zbytek konstrukce dokončit podle známého popisu. To je samozřejmě možné, pouze pokud se podaří nalézt vstup pro některý popsaný krok. Odtud název *Klíčový bod*, protože je nutné scénu dostat do určitého klíčového bodu (dá se chápat také jako určitý kontext), od kterého je již známo, jak pokračovat.

V následující kapitole se budeme věnovat tomu, jakým způsobem systém odvozuje syntaktická pravidla. Navrhli jsme obecný způsob, který je použitelný i pro jiné disciplíny, než je geometrie.

2.3. Systém relací a odvozovacích pravidel

Pro syntaktickou kontrolu je klíčové znát u každé konstrukce její vlastnosti a vztahy k jiným konstrukcím, které jsou nezávislé na konkrétním umístění geometrických objektů v prostoru, ze kterých se tyto konstrukce skládají. Nad množinou všech popisů konstrukcí v systému si proto definujeme relace. Systém relací podporuje obecně n-ární relace, ale momentálně se prakticky využívají výhradně binární. Každá instance konstrukce je vytvořena podle popisu konstrukce, což určuje jakýsi typ, třídu konstrukce. Tento typ by se dal chápat jako unární relace nad konstrukcemi. Z praktických důvodu však relační system takto typ konstrukcí nerozlišuje, ale konstrukce jsou předem rozděleny (v hashovací tabulce) podle typu (popisu). Stejně tak i relace jsou typové, každá složka relace má předem určenou doménu (kolekce přípustných typů).

Některé relace jsou symetrické nebo tranzitivní. Například relace *rovina je kolmá na rovinu.* Nebo *přímka leží v rovině, rovina obsahuje přímku.* Takovéto relace nemají zvláštní podporu, ale jsou řešeny pomocí odvozovacích pravidel (viz. dále). Výjimkou je relace ekvivalence (*EQUALS*), která má speciální podporu. Je implementována jako identita nad konstrukcemi. Z toho také přímo vyplívají její vlastnosti (symetrie, traniztivita, reflexivnost) a nejsou pro ni zvlášť definována odvozovací pravidla. Tato relace je také jako jediná definována na celé množině konstrukcí. Ostatní relace jsou definovány pouze nad primitivními konstrukcemi.

2.3.1. Relace

Každá relace (přesněji popis relace), tak jak ji reprezentujeme v našem systému, má své jméno (symbol). Toto jméno je unikátní. Kromě jména obsahuje relace také uspořádanou n-tici popisů konstrukcí (domény). Tato n-tice obsahuje nejčastěji dva prvky, počet prvků určuje aritu relace. Každý prvek n-tice, což je popis konstrukce, určuje typy konstrukcí (*ConstrInfo*), mezi kterými je relace definována. Pokud je nějaký prvek nedefinován, berou se všechny typy.

Každá instance primitivní konstrukce si udržuje kolekci svých vlastností (vztahů) k ostatním konstrukcím. Pokud je primitivní konstrukce C_1 v relaci ρ s primitivní konstrukcí C_2, přidá se do seznamu vlastností konstrukce C_1 instance relace, objekt *Prop*. Tento objekt má odkaz na obě konstrukce C_1 i C_2 a na relaci ρ. Přitom se kontroluje, aby popis konstrukce C_1 odpovídal první doméně relace ρ a popis konstrukce C_2 druhé doméně relace ρ. Výjimku tvoří relace ekvivalence $=$, u které se fakt, že $C_1 = C_2$, neukládá pomocí objektu *Prop*.

Seznam všech (popisů) relací v relačním systému:

${}^0\in^1$ – Bod leží na přímce.

${}^1\ni^0$ – Přímka obsahuje bod.

${}^0\in^2$ – Bod leží v rovině..

${}^2\ni^0$ – Rovina obsahuje bod.

${}^1\in^2$ – Přímka leží v rovině.

${}^2\ni^1$ – Rovina obsahuje přímku.

${}^2\|^2$ – Rovina je rovnoběžná s rovinou.

${}^1\|^1$ – Přímka je rovnoběžná s přímkou.

${}^1\|^2$ – Přímka je rovnoběžná s rovinu.

${}^2\|^1$ – Rovina rovnoběžná s přímku.

${}^2\perp^2$ – Rovina je kolmá na rovinu.

${}^1\perp^1$ – Přímka je kolmá na přímku.

${}^1\perp^2$ – Přímka je kolmá na rovinu.

${}^2\perp^1$ – Rovina je kolmá na přímku.

$=$ – Konstrukce jsou referenčně ekvivalentní..

Všechny tyto relace jsou definovány mezi primitivními konstrukcemi, relace $=$ i mezi ostatními. Proto jsme zavedli konvenci v pojmenovávání relací: název relace začíná a končí číslem 0, 1 nebo 2, které souvisí s první a druhou doménou. Přitom čísla mají následující význam:

0 – Konstrukce bodu

1 – Konstrukce přímky

2 – Konstrukce roviny

2.3.2. Rozšíření klíčového bodu o podporu relací

Jak již bylo uvedeno dříve, klíčový bod se skládá z požadavků. Požadavek je pojmenovaný a má přiřazenou konstrukci. Navíc každý takový požadavek obsahuje dva seznamy podmínek, negativní a pozitivní. Každý požadavek pak zapíšeme ve tvaru:

s_k:*název popisu konstrukce* WHERE [*podmínky*$+_k$] AND NOT[*podmínky*-$_k$],

kde s_k je k-tý symbol v klíčovém bodu a *podmínky*$\pm_k$ je seznam pozitivních a negativních podmínek tvaru:

$$P_k\rho_{i_1}P_{i_1},\ P_k\rho_{i_2}P_{i_2},\ \ldots,P_k\rho_{i_n}P_{i_n},$$

kde ρ_{i_j} je binární relace nad množinou všech popisů konstrukcí, P je cesta v popisovaném klíčovém bodě. Přitom cesta P_k musí být listová. Po použití zkráceného zápisu cesty můžeme psát:

$$s_k\rho_{i_1}|P_{i_1}|,\ s_k\rho_{i_2}|P_{i_2}|,\ \ldots, s_k\rho_{i_n}|P_{i_n}|,$$

kde $|P|$ představuje zkrácený zápis. Dále požadujeme, aby souhlasily domény relací. Pokud relace ρ představuje relaci $=$, domény souhlasí vždy. Pokud ne, musí cesta P vést k popisu primitivní konstrukce, odpovídajícímu příslušné doméně relace ρ.

Dvě instance konstrukcí C_1 a C_2 jsou v relaci $=$ právě tehdy, když jsou C_1 a C_2 identické, jsou v relaci ρ, která není relace $=$, právě tehdy, když jsou C_1 a C_2 primitivní a obsahují objekt *Prop*, který má vazbu na C_1, ρ a C_2.

Prvek podmínky (dvojice $s_k\rho_i|P_i|$) je splněn, pokud jsou konstrukce C_1 a C_2 v relaci ρ_i. Přitom C_1 je konstrukce určena symbolem s_k a konstrukce C_2 je určena cestou P_i (cesta k ní vede).

Instancí klíčového bodu (*KeyPointMatch*) je pak navázání konkrétních instancí konstrukcí na symboly jednotlivých požadavků. Přitom musí být splněno:

- Každá instance konstrukce musí být vytvořena podle popisu odpovídajícího požadovanému popisu.
- Každý prvek z pozitivních podmínek *podmínky+*$_k$ musí být splněn.
- Žádný prvek z negativních podmínek *podmínky-*$_k$ nesmí být splněn.

Díky takovému rozšíření je možné vytvářet i „složité dotazy“. Například: „všechny dvojice různých bodů, které leží v jedné rovině, která je kolmá na nějakou přímku“. Klíčové body se kromě definování vstupu pro jednotlivé kroky konstrukcí používají i jinde. Například ta část grafického rozhraní, která vykresluje konstrukce v Mongeově promítání, pomocí klíčových bodů hledá objekty, které má vykreslit. Používá k tomu čtyři dotazy, pomocí kterých hledá všechny přímky a body v půdorysně(π) a nárysně(ν) konstrukce Mongeova promítání.

Hledání všech instancí klíčového bodu (*KeyPointMatch*) podle daného popisu klíčového bodu (*KeyPointInfo*) jsme realizovali asi nejjednodušším možným způsobem. Postupně se vyčíslují všechny možnosti. Hledání lze částečně ovlivnit určením množiny konstrukcí, které se musí použít pro konkrétní požadavek klíčového bodu, nebo množinou konstrukcí (fixování), které by se měly použít kdekoli (preferování). Přesto je při vetším počtu konstrukcí realizovatelné hledání klíčových bodů, které mají maximálně 4 až 5 požadavků. Hledání má zjevně nejhorší složitost x^n, kde x je počet prohledávaných konstrukcí a n je počet požadavků. Prakticky jsme ale zjistili, že to není zásadní překážkou. Je to hlavně z následujících příčin:

- Požadavky klíčových bodů mají průměrně 2-3 požadavky.

- Konstrukce jsou uchovávány rozděleny podle typu (popisu), které se zohledňuje při hledání.
- Příklady vytvořené v systému jsou jednoduché, skládají se řádově z desítek konstrukcí
- Hledání instancí klíčového bodu vynechává některé konstrukce, které nemohou nikdy vyhovět podmínkám a ukládá si je pro další průchod.
- Většinou není nutné najít úplně všechny instance klíčového bodu.

2.3.3. Odvozovací pravidla

Jak již bylo řečeno, primitivní konstrukce jsou přímo spjaty s geometrickými objekty, a kroky primitivních konstrukcí s jejími deklarátory. Podle toho lze po vytvoření primitivní konstrukce ihned zjistit, že má nějaký trvalý vztah k jiné konstrukci. Tento vztah zachytíme relací a objektem *Prop*. Takto získané vlastnosti budeme říkat *faktická*. Příklad: Pokud vytvoříme bod A v rovině α, znamená to samozřejmně, že bod A vždy leží v rovině α. Tuto vlastnost popisuje relace ${}^0\in^2$. Do seznamů objektů *Prop* primitivní konstrukce odpovídající bodu A tedy přidáme nový objekt *Prop*, který bude mít vazbu na bod A, relaci ${}^0\in^2$ a rovinu α.

Další věc je, že tento vztah bodu A k rovině α je symetrický. Ihned vyplývá, že i rovina α obsahuje bod A. Takovéto vlastnosti budeme říkat *odvozená*. Podobné (i složitější) odvození jako je toto, zajišťují právě odvozovací pravidla.

Princim odvozovacích pravidel se dá volně popsat tatko: Platí-li tvrzení K a konstrukce C nemá vlastnost v, přidej konstrukci C vlastnost v. K reprezentaci tvrzení K pravidla využívají již popsané klíčové body. Do tohoto klíčového bodu K se zahrne i negativní podmínka ověřující, že konstrukce ještě nemá vlastnost v. To, že „platí tvrzení K“ pak znamená, že k popisu klíčového bodu K existuje nějaká instance klíčového bodu. Samotná vlastnost v je reprezentována pomocí již popsaných relací. V systému jsou tato odvozovací pravidla:

Symetrická tvaru:

$$A\rho_1 B \Rightarrow B\rho_2 A$$

B:Line WHR $\neg$[B ${}^1\ni^0$ A], A:Point WHR [A ${}^0\in^1$ B] $\Rightarrow$ B+=B ${}^1\ni^0$ A
B:Point WHR $\neg$[B ${}^0\in^1$ A], A:Line WHR [A ${}^1\ni^0$ B] $\Rightarrow$ B+=B ${}^0\in^1$ A
B:Plane WHR $\neg$[B ${}^2\ni^0$ A], A:Point WHR [A ${}^0\in^2$ B] $\Rightarrow$ B+=B ${}^2\ni^0$ A
B:Point WHR $\neg$[B ${}^0\in^2$ A], A:Plane WHR [A ${}^2\ni^0$ B] $\Rightarrow$ B+=B ${}^0\in^2$ A
B:Plane WHR $\neg$[B ${}^2\ni^1$ A], A:Line WHR [A ${}^1\in^2$ B] $\Rightarrow$ B+=B ${}^2\ni^1$ A
B:Line WHR $\neg$[B ${}^1\in^2$ A], A:Plane WHR [A ${}^2\ni^1$ B] $\Rightarrow$ B+=B ${}^1\in^2$ A

B:Plane WHR $\neg$[B $^2\perp^2$ A], A:Plane WHR [A $^2\perp^2$ B] $\Rightarrow$ B+=B $^2\perp^2$ A
B:Line WHR N$\neg$[B $^1\perp^1$ A], A:Line WHR [A $^1\perp^1$ B] $\Rightarrow$ B+=B $^1\perp^1$ A
B:Plane WHR $\neg$[B $^2\perp^1$ A], A:Line WHR [A $^1\perp^2$ B] $\Rightarrow$ B+=B $^2\perp^1$ A
B:Line WHR $\neg$[B $^1\perp^2$ A], A:Plane WHR [A $^2\perp^1$ B] $\Rightarrow$ B+=B $^1\perp^2$ A
B:Plane WHR $\neg$[B $^2\|^2$ A], A:Plane WHR [A $^2\|^2$ B] $\Rightarrow$ B+=B $^2\|^2$ A
B:Line WHR $\neg$[B $^1\|^1$ A], A:Line WHR [A $^1\|^1$ B] $\Rightarrow$ B+=B $^1\|^1$ A
B:Line WHR $\neg$[B $^1\|^2$ A], A:Plane WHR [A $^2\|^1$ B] $\Rightarrow$ B+=B $^1\|^2$ A
B:Plane WHR $\neg$[B $^2\|^1$ A], A:Line WHR [A $^1\in^2$ B] $\Rightarrow$ B+=B $^2\|^1$ A

Tranzitivní tvaru:

$$A\rho_1 B \wedge B\rho_2 A \Rightarrow A\rho_3 C$$

C:Plane, B:Line WHR [B $^1\in^2$ C], A:Point WHR [A $^0\in^1$ B] $\wedge$ $\neg$[A $^0\in^2$ C] $\Rightarrow$ A+=A $^0\in^2$ C

C:Plane, B:Plane WHR [B $^2\perp^2$ C], A:Plane WHR [A $^2\perp^2$ B] $\wedge$ $\neg$[A $^2\|^2$ C] $\Rightarrow$ A+=A $^2\|^2$ C

C:Line, B:Line WHR [B $^1\perp^1$ C], A:Plane WHR [A $^2\perp^1$ B] $\wedge$ $\neg$[A $^2\|^1$ C] $\Rightarrow$ A+=A $^2\|^1$ C

C:Plane, B:Line WHR [B $^1\perp^2$ C], A:Line WHR [A $^1\perp^1$ B] $\wedge$ $\neg$[A $^1\|^2$ C] $\Rightarrow$ A+=A $^1\|^2$ C

C:Plane, B:Plane WHR [B $^2\|^2$ C], A:Plane WHR [A $^2\|^2$ B] $\wedge$ $\neg$[A $^2\|^2$ C] $\Rightarrow$ A+=A $^2\|^2$ C

Ostatní:

B:Point WHR [B $^0\in^2$ α, B $^0\in^1$ a] $\wedge$ $\neg$[B = A], A:Point WHR [A $^0\in^2$ α, A $^0\in^1$ a] $\wedge$ $\neg$[A = B], a:Line WHR [a $^1\ni^0$ A, a $^1\ni^0$ B] $\wedge$ $\neg$[a $^1\in^2$ α], α:Plane WHR [α $^2\ni^0$ A, α $^2\ni^0$ B] $\Rightarrow$ a+=a $^1\in^2$ α

WHERE bylo zkráceno na *WHR*.

Vyjadřovací schopnost takto reprezentovaných pravidel je obdobná jako u formulí výrokové logiky tvaru:

$$\bigvee (l_1 \wedge l_2 \wedge \cdots \wedge l_n) \Rightarrow C$$

Kde formule na levé straně implikace je v disjunktivní normální formě. Do tvaru pro klíčové body tuto formuli můžeme převést takto:

- Všechny pozitivní (negativní) literály z $l_1 ... l_n$ dáme k sobě, budou to pozitivní (negativní) podmínky klíčového bodu. Toto je možné díky vlastnostem konjunkce.

- Vytvoříme takto pravidla pro všechny elementární konjunkce literálů se
- stejnou formulí C na pravé straně.

3. Architektura systému

Celý systém je rozdělen do dvou hlavních částí:

- jádro systému;
- aplikační rozhraní.

Snažili jsme se jádro striktně od rozhraní oddělit, aby byl projekt rozšiřitelný a znovupoužitelný. Dalo by se tedy vytvořit další uživatelské rozhraní, například pro řádkového klienta, nebo použít jádro jako knihovnu v jiném programu.

Aplikační rozhraní bylo popsáno v kapitole o implementačních detailech ve 4. kapitole. V této kapitole se budeme věnovat detailnějšímu popisu jádra systému.

Při návrhu jsme se drželi standardních postupů využívaných při programování Java aplikací. Některé věci jsme zavedli po svém, jako například vlastní systém událostí, který jsme převzali z platformy .NET a využili jsme u toho nové vlastnosti jazyka Java 5.0. Naše obsluha událostí využívala generických typů a nabízela také automatickou odregistraci v případě garbage collectingu (pomocí slabých referencí), čímž se eliminuje riziko vzniku potencionálních memory leaků v důsledku neodregistrování se klienta od sledování události. Platforma .NET pro nás byla také inspirací při vytváření popisných objektů (BeanInfo), které využívá například okno vlastností. Tady jsme vytvořili celý system (balíček *Propgen*), který pomocí reflexe a anotací (obdoba *attribute* z .NET) tyto popisné objekty generuje.

3.1. Iterativní vývoj

Při vlastním vývoji jsme postupovali iterativně s tím, že jsme se snažili vždy řešit co nejmenší úkoly. Velmi pomalu jsme tedy přidávali novou a novou funkčnost, vše řádně dokumentovali a psali testy pomocí knihovny JUnit. Všechny testovací třídy končí slovem „Test“ a jejich spouštěni lze automatizovat nástrojem Apache Ant.

Dokumentace byla nutná i z toho důvodu, že jsme pracovali v týmu. Pokud jsme přidali funkčnost, která nebyla zdokumentována, později musela být dokumentace stejně dopsána, aby druhý člen týmu věděl, jak třídy používat. Včasná dokumentace nám šetřila čas, protože daleko hůře se dokumentuje kód, který jste psali před týdny či dokonce měsíci.

3.2. Návrhové vzory použité v jádře

V následující kapitole krátce popíšeme návrhové vzory, kterými se to v celé aplikaci doslova jen hemží. Za referenční literaturu v tomto případě považujeme zejména slavnou knihu „Gang Of Four“ [6] a hodně informací jsme čerpali ve vynikající knize o programování v Javě od spoluautora jazyka [3].

Abstraktní továrnu jsme využili při vytváření základních geometrických objektů (GeomObjectFactory). Tovární a šablonové metody jsou rozesety po celém jádře a dnes už se staly denním chlebem programátorů v Javě. Velmi podobně je na tom návrhový vzor Jedináček. Objekty World a Hinter zastupují návrhový vzor Fasáda.

Zajímavý přístup jsme použili při ukládání scény na disk. Jelikož byla jednou z požadavků potřeba zpětné kompatibility a nutnost načítání starších verzí souborů, použili jsme návrhový vzor Memento. Při potřebě vytvořit novou verzi, která ovlivní formát souboru, se původní Memento objekty zachovají v samostatném balíčku (jsou nezávislé na jádře), vytvoří se nová Mementa v novém balíčku, kde se také vytvoří konvertor, který převede případná stará Mementa na nová.

Za zmínku také stojí systém filtrů, který je naprogramován velmi flexibilně, a to za použití návrhového vzoru Dekorátor. Lze sestavovat libovolně složité podmínky pouhým řetězením a skládáním filtrovacích tříd.

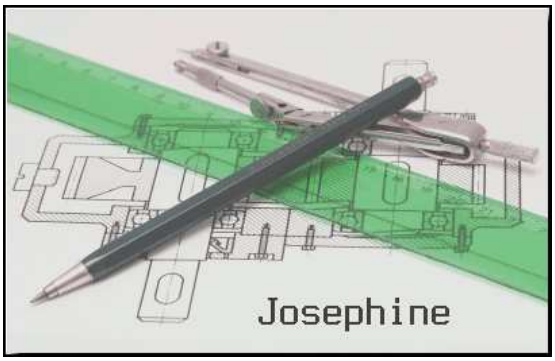
Celé jádro je pečlivě rozděleno do Java balíčků, které jsou detailně popsány v druhé příloze. Všechny třídy a drtivá většina všech veřejných metod je dokumentována, ale rozhodli jsme se dokumentaci aplikačního rozhraní jádra do diplomové práce nezařadit, protože by se počet stran zhruba zpětinásobil. V druhé příloze jsou tedy stručně popsány některé důležité třídy jádra.

4. Implementace uživatelského rozhraní

Pro náš systém jsme zvolili platformu Java, protože hlavním cílem našeho snažení bude využití ve výuce geometrie. Tato technologie dovoluje využití takzvaných appletů, programů vestavěných ve webových prohlížečích. Dalšími hledisky byla vyspělost jazyka Java a dobrá míra přenositelnosti. Vývoj totiž probíhal paralelně na systémech Linux a Windows.

Pro implementaci uživatelského rozhraní jsme nejprve využili knihovny AWT/Swing, které jsou součástí aplikačního programovacího rozhraní Javy. Postupem času jsme však narazili na několik problémů. Jak rostla velikost aplikace, zvyšovala se velikost kódu pro implementaci uživatelského rozhraní a věcí spojených (dialogy, menu, nastavení aplikace). Problémy také nastaly s použitím knihovny OpenGL, kterou jsme chtěli pro implementaci zobrazování použít. Proto jsme se rozhodli využít již hotový framework pro tvorbu uživatelských rozhraní Eclipse Rich Client Platform (dále jen RCP).

Tato platforma dovoluje vytvářet znovupoužitelné aplikace (resp. takzvané zásuvné moduly, plug-ins, pro společné běhové prostředí – Eclipse Platform Runtime). Díky Eclipse RCP je možné tvořit přehlednější uživatelská rozhraní rychle a efektivně. Programátoři mají k dispozici mnoho hotových komponent a prostřednictvím jasně definovaných rozhraní mohou využít mnoho věcí, které by jinak každý projekt musel realizovat sám (nastavení, dialogy, průvodci, manipulace s dokumenty, dokovatelná okna a podobně).



Obrázek 1. Úvodní obrázek při spuštění systému.

Další výhodou Eclipse Rich Client Platform byla výborná spolupráce s OpenGL. Eclipse RCP totiž využívá přirozených prostředků operačního systému (native GUI), takže vzhled aplikace v systému Windows je k nerozeznání od vestavěných programů či programů napsaných pro platformu .NET. Odezva uživatelského rozhraní je také daleko rychlejší. V systému Linux je využito knihovny GTK+, takže i na tomto systému je aplikace daleko přirozenější.

4.1. Eclipse Rich Client Platform

V prvé řadě bychom měli nastínit, co je vlastně Eclipse. V doslovném překladu toto slovo znamená zatmění, ve světě open source je to však komunita pohybující se kolem portálu www.eclipse.org a platformy Java. Hlavním produktem úsilí vývojářů z celého světa je pak Eclipse IDE, což je prvotřídní nástroj pro vývoj Java aplikací. Toto prostředí vzniklo v laboratořích firmy IBM, která uvolnila značnou část ze svých produktů WebSphere jako open source. Firma nadále své produkty zakládá na Eclipse (vývojová prostředí J2EE WebSphere, nové verze nástrojů firmy Rational a podobně). Společnost IBM zaměstnává celý vývojový tým, který udává hlavní směr vývoje.

Srdcem celého Eclipse IDE je Eclipse Rich Client Platform (zkráceně Eclipse RCP). Toto rozhraní poskytuje robustní komponentový model, kde každá komponenta může existovat zcela nezávisle (v různých verzích), aplikace mohou komponenty jednoduše instalovat, sdílet a aktualizovat on-line. V terminologii Eclipse se těmto komponentám říká pluginy.

Pro vlastní vývoj jádra jsme využili novinky verze jazyka Java 1.5 (5.0), zejména parametrizované datové typy. Kód, který tvoří vizuální stránku aplikace, je však psán za pomoci knihoven Eclipse verze 3.2. Tato verze ještě nevyužívá nové prvky jazyka, a tak bylo na mnoha místech nutné vypořádat se s převody do parametrizovaných typů. Celkově nám však využití nové verze jazyka Java zpříjemnilo a zjednodušilo vývoj.

4.1.1. Architektura Eclipse RCP

Pluginy v Eclipse RCP je možno vytvářet zcela nezávisle na sobě. Je to dáno tím, že každá komponenta, která je do prostředí připojena v bodu rozšíření (extension point), do něj určitým způsobem přispěje (přidá položky do nabídek menu, tlačítka mezi nástroje, novou schopnost otevřít soubor určitého typu a podobně). Tím se výrazně podporuje další rozšiřování a obohacování produktů, které jsou takto navrženy.

Eclipse RCP nabízí API pro efektivní tvorbu uživatelského rozhraní pomocí návrhového vzoru MVC (Model-View-Controller). Platforma nepoužívá standardní javovskou knihovnu Swing, která je dodávána společně s jazykem Java v rámci J2SE, ale vlastní implementaci UI nazvanou Standard Widget Toolkit (SWT). Nejnižší grafické prvky nejsou vykreslovány v Javě, ale pomocí základních knihoven operačního systému. To má řadu výhod i nevýhod, mezi hlavní výhody patří možnost používání některých komponent, které nejsou ve Swingu dostupné (například ikona v oznamovací oblasti), a také kratší odezva při událostech. Nevýhodou je pak nutnost investovat větší úsilí do přenositelnosti, protože SWT používá na systémech Windows knihovnu MVC, na unixech pak GTK+.

Knihovna SWT zajišťuje pouze „hloupé“ vytváření grafických prvků a komponent – není tedy schopna pracovat s daty. V chápání MVC se jedná o View. Zde

je patrný rozdíl s knihovnou Swing firmy Sun, kde je každá grafická komponenta odpovědná jak za vykreslování (View), tak za samotná data (Model).

V Eclipse RCP jsou data zapouzdřena ve zvláštních objektech, což dává programátorům možnost implementovat své datové objekty tak, aby byly přímo použitelné v grafických komponentách. Potřebná rozhraní a pomocné třídy jsou dostupné ve zvláštní knihovně nazvané JFace. Poskytuje také podpůrné nástroje pro vytváření prvků uživatelského rozhraní vyšší úrovně (například průvodce, okna nastavení či často používané dialogy). Tento přístup také umožňuje vytvořit si model-vrstvu zcela nově.

Samotné prostředí Eclipse je implementováno v komponentách, které řadíme do balíku UI. Tento balík komponent poskytuje rozhraní a nástroje pro vytváření dvou hlavních komponent, které jsou určeny k interakci s uživatelem – editory a nástrojová okna (v terminologii Eclipse: editors a views). Dalšími neméně důležitými prvky, které jsou v těchto komponentách implementovány, je vlastní pracovní prostředí (tzv. workbench – hlavní okno aplikace), uspořádání (tzv. perspective) a také systém akcí a příkazů (actions, commands).

Abychom probrali všechny základní stavební kameny Eclipse RCP, zbývá nám osvětlit balíky Runtime a OSGi. První jmenovaný definuje to, jakým způsobem se celé prostředí spouští, jak se inicializují a zavádějí pluginy, produkty (product – skupina vzájemně spolupracujících pluginů, které tvoří jeden celek) a aplikace (finální instalace aplikace založené na Eclipse Runtime Platform).

Promyšlený systém nasazování pluginů (deployment) je jednou z hlavních předností Eclipse. Celá platforma nabízí dlouhou řadu takzvaných přípojných bodů (extension points), navíc může programátor vytvářet vlastní. Právě v těchto místech (a ne jinde) pluginy do prostředí vstupují. Každý plugin může do prostředí přispět novými ovládacími prvky a implementovat akce, které budou provedeny po aktivování těchto ovládacích prvků.

My jsme celou aplikaci implementovali jako jeden velký plugin, protože nebyla potřeba program žádným způsobem tříštit. Tato architektura však dovoluje snadno program obohatit o nové funkce pouhým přidáním nových pluginů.

Celá komponentová architektura Eclipse je postavena na modelu OSGi (www.osgi.org), což je komponentová architektura navržená výhradně pro jazyk Java. Poskytuje efektivní nástroje zejména na řešení závislostí mezi komponentami, sestavování a nasazování komponent. Ačkoli se jedná o zcela obecný přístup, prozatím OSGi nenachází širšího využití mimo Eclipse.

Platforma Eclipse RCP poskytuje služby širokému spektru vývojářů a firem po celém světě. Kromě firmy IBM/Rational jej používá například americká NASA, firma Oracle a dokonce firma Borland ohlásila, že její nástroj JBuilder bude v příští verzi založen právě na Eclipse.

4.1.2. Uspořádání, editory a nástrojová okna

Pokud poprvé spustíte rozsáhlejší program založený na Eclipse RCP (napří-

klad právě Eclipse IDE), může být pro vás styl práce v něm menším překvapením. Možná právě proto, že Eclipse RCP v některých věcech poskytuje trošku neortodoxní přístup pro uživatele systému Windows, byla tato platforma zpočátku odmítána.

Hlavními dvěma prvky, se kterými se budete v naší aplikaci (a většinou programů postavených na Eclipse RCP) setkávat, jsou editory a nástrojová okna. Editor má své pevné místo uprostřed pracovní plochy a jedná se v podstatě o komponentu, která je zodpovědná za editaci dokumentu, obrázku či jiných dat.

Všechny editory se otvírají na jednom místě – v takzvaném editor site. Pokud je aplikace schopna otevírat a editovat více typů dat (např. již zmiňovaný obrázek a třeba textový dokument), editory jsou otevírány vedle sebe. Každý editor může mít několik pohledů (stránek). V našem případě jsou to pohledy do konstrukce – perspektivní a Mongeovo promítání.

Editory doplňují nástrojová okna – views. Tato okna mohou být celkem na pěti různých pozicích (vpravo, vlevo, nahoře, dole a plovoucí) a uživatel je může libovolně přesouvat, zavírat, znovu otevírat či měnit velikost rozdělení mezi editory a nástrojovými okny.

Nástrojová okna společně s editory tvoří takzvané uspořádání. Uživatel i pluginy mohou přepínat mezi jednotlivými uspořádáními (např. editace textového dokumentu, prohlížení struktury dokumentu). Uspořádání může uživatel také vytvářet, platforma se postará o perzistenci.

V naší aplikaci je použito hned několik nástrojových oken a jeden typ editoru pro vlastní pohled do scény. Možnost přepínat uspořádání jsme zakázali, protože program nemá tak složité uživatelské rozhraní, aby se v něm uživatel ztrácel a měl potřebu vytvářet nová uspořádávání.

4.2. OpenGL

V počátcích projektu jsme se rozhodovali, jak vyřešit grafický výstup. V podstatě jsme měli čtyři základní možnosti. První volbou, kterou jsme se také v raných alfa verzích projektu vydali, byla vlastní implementace zobrazování. Kreslili jsme pomocí základního Java2D API.

Tento přístup měl řadu výhod, zejména byl perfektně přenositelný. Rozhraní Java2D se výborně hodí pro Java applety, protože funguje v každém prohlížeči bez dalších instalací či potvrzování uživatelem. Celé řešení však mělo jednu velkou nevýhodu – veškerý výstup jsme si museli programovat sami. Také plynulost překreslování při pohybech scény nebyla ideální.

Zkoušeli jsme tedy rozhraní Java3D. Zjistili jsme, že je značně komplikované a na naše jednoduché úkony bylo potřeba mnoho kroků k tomu, abychom se dopracovali výsledku. Vše nakonec rozhodl fakt, že jsme místo knihovny Swing použili Eclipse RCP. Tam jsme mohli opět použít ruční vykreslování na kreslicí plátno, avšak třídy pro kreslení nebyly kompatibilní s knihovnou Java2D.

Další možností bylo OpenGL, protože Eclipse nabízí vlastní zásuvný modul a v podstatě bez instalačních potíží lze používat OpenGL. Toho jsme také využili. Knihovna OpenGL je ve světě standardem, navíc jsme měli za sebou dva semestry programování v OpenGL a jako třetí argument, který nám pomohl při rozhodování, byl vysoký výkon. Už první pokusy ukázaly, že vykreslování v OpenGL bude velmi rychlé, a tedy i natáčení perspektivní scény plynulé.

Projekt jsme původně psali pro Eclipse RCP verze 3.1, kde byla pouze jediná možnost, jak k OpenGL funkcím přistupovat. Během vývoje této diplomové práce však vyšla verze Eclipse RCP 3.2, kde byly možnosti ohledně OpenGL výrazně širší. Jednak byla podpora pro OpenGL již součástí knihovny SWT 3.2, na které je RCP stejné verze postavena, ale zejména přibyla možnost využít OpenGL knihovny pro jazyk Java od třetích stran.

My jsme toho však nevyužili. Základní podpora v knihovně SWT 3.2 nám plně vyhovovala a ačkoliv implementuje jen standard OpenGL 1.1, vystačili jsme si s ní. Jediná funkce, která nám chyběla, byl přímý přístup do OpenGL bufferu. Pro tento účel jsme naprogramovali plugin, který přístup umožňuje. Protože plugin obsahuje funkce závislé na platformě, vytvořili jsme verzi jak pro Windows, tak pro Linux.

Ve finální verzi projektu tedy není nutné instalovat žádné další OpenGL knihovny, protože základní podpora je přímo v Eclipse RCP. Je však nutné platformu obohatit o náš plugin org.eclipse.opengl.selectionbuffer.

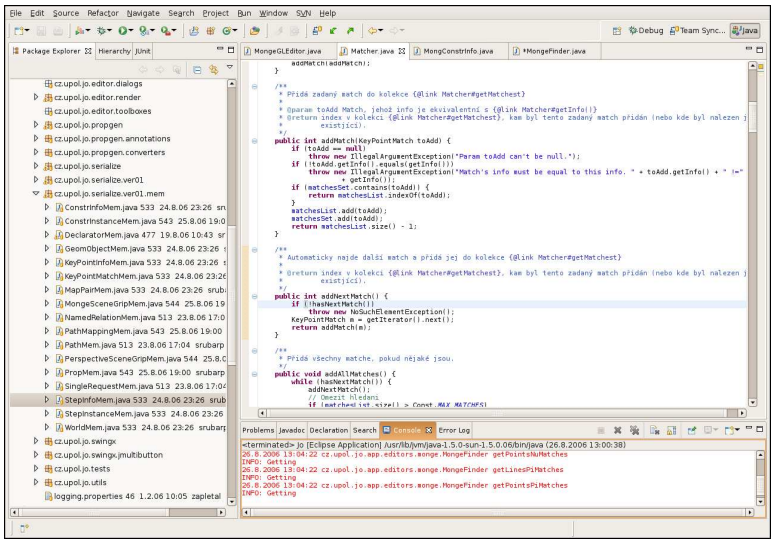
Pro běh celé aplikace je nutno mít nainstalováno Java 5.0 Runtime. Toto běhové prostředí je k dispozici na přiloženém CD, a to jak ve verzi pro Windows, tak pro Linux (i386).

4.3. Podpůrné nástroje

4.3.1. Eclipse IDE

Zcela zásadním vývojovým nástrojem bylo pochopitelně Eclipse IDE. Nejenže nabízí komfortní editaci kódu v Javě, ale hlavně také nástroje pro snadnou tvorbu pluginů a konfigurací. Zejména konfigurační XML soubory mají poměrně složitou strukturu, a proto Eclipse IDE poskytuje speciální editory. Jako integrované programovací prostředí obstál tento nástroj na jedničku, obsahuje velmi komfortní průvodce, refaktoring, intellisense a také lze stáhnout plugin, který zpřístupňuje podporu pro Subversion. Ten je, podobně jako celé Eclipse IDE, k dispozici zdarma a pod open-source licencí.

Bez Eclipse IDE by bylo velmi komplikované také sestavování výsledného produktu. To totiž bohužel neprobíhá přímo, ale je nutno provést značné množství kroků, vygenerovat různé manifesty a další konfigurační soubory a také finální spustitelný soubor. Zatím tuto činnost nelze provést bez samotného Eclipse IDE, ale již se objevují projekty, které se tento proces snaží alespoň trochu automatizovat (např. nástroje pro Ant).



Obrázek 2. Eclipse IDE v prostředí Linuxu.

4.3.2. Subversion

Jelikož jsme na tuto diplomovou práci byli dva, bylo nutné nějakým způsobem ošetřit správu verzí. Využili jsme open-source nástroj Subversion, který vychází z dnes již legendárního systému CVS. Oproti CVS má navíc Subversion zejména atomické odevzdávání (atomic commits – přerušení při komunikaci nenaruší repozitář), lepší přenositelnost mezi platformami (což bylo pro nás velmi důležité – například se objevily problémy s kódováním a zakončením řádků), více možnosti v odpojeném (offline) režimu, vylepšené řešení konfliktů a vývojáři udávají také vyšší výkon.

Subversion nám umožnil současnou práci na jednom souboru, což pro nás bylo stěžejní. Zejména v prvních fázích projektu se často stávalo, že jsme oba v rámci jednoho vývojového cyklu změnili týž soubor. Systémy jako CVS nebo Subversion si s těmito situacemi poradí. To se nám hodilo také při psaní dokumentace, kterou jsme zpracovávali pomocí LaTeXu.

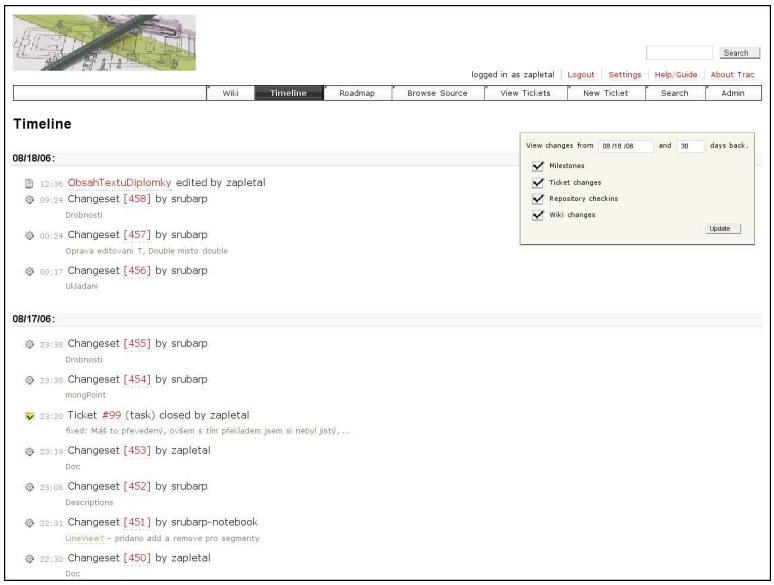
Centrální repozitář byl dostupný z internetu, takže vývoj mohl probíhat i na služebních cestách či z domova. Server byl pravidelně zálohován, což byla také jedna z výhod, která plynula z používání systému pro správu verzí Subversion. Pokud jsme zjistili, že jsme se vydali špatnou cestou, nebyl vůbec žádný problém vrátit se v čase do určitého stavu a zkusit vytvořit řešení jiné.

Každé odevzdání dílčího úkolu jsme řádně zdokumentovali v komentářích systému Subversion, aby bylo jasné, k čemu se dané řešení váže. Po celou dobu projektu jsme tedy měli přehled o tom, co dělá kolega. Ani zdaleka jsme nevyužili všechny vlastnosti, které Subversion nabízí – projekt nebylo třeba například vůbec větvit či vytvářet značky.

Repozitář je v současné době k dispozici na adrese svn://svn.zapletalovi.com/josephine a pro čtení není potřeba heslo. Je tedy možné spustit si program v některé z dřívějších revizí a podívat se, jak probíhal vývoj v čase. Pro detailní informace o postupech prací je zde však jiný nástroj – Trac.

4.3.3. Trac

Webová aplikace Trac firmy Edgewall Software je dalším open-source nástrojem, který jsme hojně využívali. Poskytoval nám jakousi základnu pro další komunikaci. Trac nabízí několik služeb: wiki, plánování projektu, přehled prací, prohlížeč repozitáře a systém hlášení chyb.



Obrázek 3. Trac – časová osa.

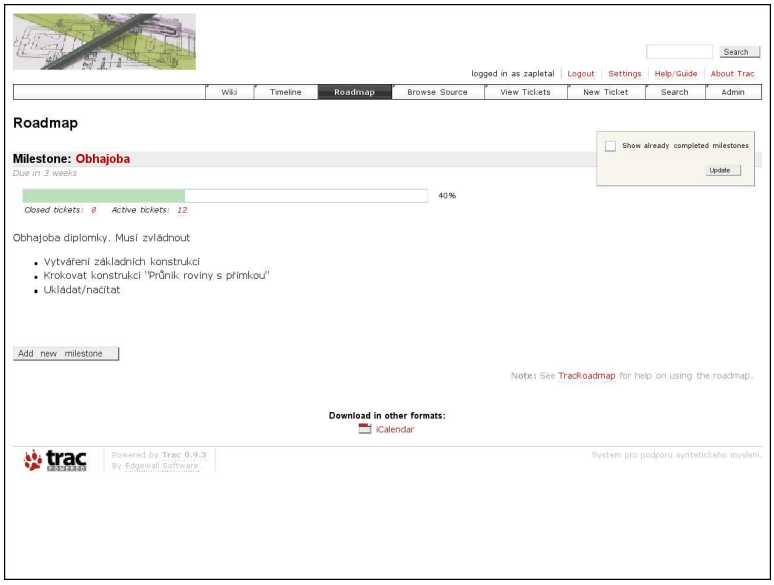
Myšlenka wiki je asi každému dobře známá díky projektu Wikipedia.org. Samotná myšlenka sdílení informací na internetu (wiki) je však daleko starší než samotná Wikipedie. V systému Trac je k dispozici jednoduchá wiki, která nám umožnila sdílet informace. Na wiki jsme shromažďovali své návrhy, návody či implementační detaily.

Asi nejdůležitější součástí Tracu je systém hlášení chyb (bugzilla, bug tracker nebo také bugtraq). Jelikož je na nich závislé také plánování projektu a do Tracu se nevkládají jen hlášení o chybách, ale také úkoly (tasks), nazývají se tyto záznamy lístky (tickets – vzhledem připomínají žluté nalepovací lístečky na ledničku).

Pomocí lístků se dají do systému vkládat jednak chybová hlášení, čehož jsme pochopitelně využili, když například někdo našel chybu v komponentě, kterou

měl na starosti kolega. Lístky jsme ale používali také k plánování.

Při plánování se využívá plánovacího modulu (roadmap), kde lze vytvářet takzvané projektové milníky. K těmto milníkům se poté mohou „přilepovat“ lístky, na které jsme zaznamenávali dílčí úkoly. Trac je schopen zobrazovat přehled stavu projektu a jednotlivých milníků tak, že sčítá vyřešené a nevyřešené lístky.



Obrázek 4. Trac – plán vývoje.

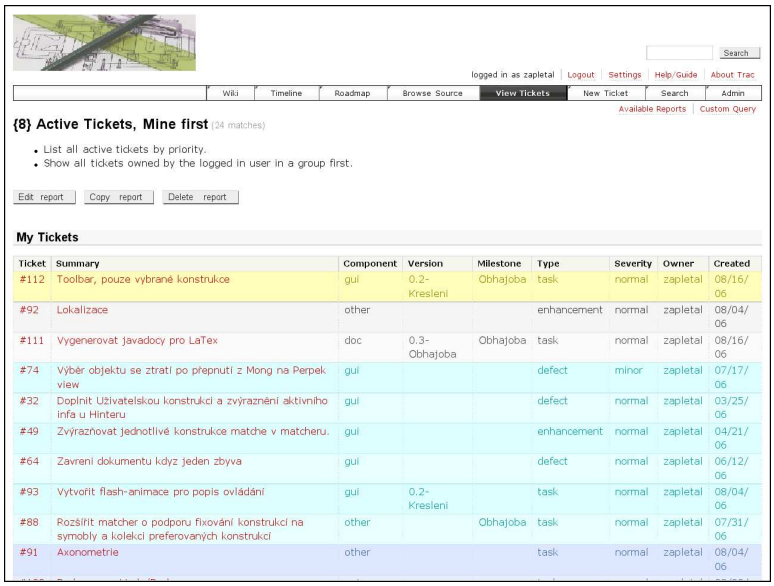
Každý lístek má určitý stav (otevřen, vyřešen, nevyřešen, chybný), což se projeví právě v plánovacím modulu. Ve všech fázích projektu jsme měli okamžitý přehled o tom, kolik úkolů nemáme hotových, kolik zbývá času a podobně. Trac vše zobrazuje graficky pomocí zelených a červených čar.

Prohlížeč repozitáře (code browser) nám a zejména vedoucímu diplomové práce poskytoval komfortní nástroj k prohlížení zdrojových kódů. Jeho hlavní předností je schopnost zobrazovat soubory i ve starších verzích, eventuálně ukázat rozdíly mezi verzemi – a to přehlednou formou.

Přehled prací (timeline) využíval vedoucí diplomové práce ke sledování konkrétních implementací. Tento modul poskytuje přehled všech modifikací v Tracu (lístky, wiki stránky) i Subversionu (včetně komentářů u jednotlivých odevzdání).

Formátování pomocí wiki hraje v Tracu důležitou roli, ve všech komponentách je možno text takto formátovat, navíc lze použít několika různých speciálních forem odkazování se na jiné komponenty. Například v záznamu o odevzdání stačí napsat ♯138, což Trac formátuje jako hypertextový odkaz směřující na lístek s číslem 138. Jedním kliknutím je tedy možné zobrazit si hlášení o chybě, kterou tento commit opravuje.

Webová aplikace Trac je k dispozici pro nahlédnutí na adrese



Obrázek 5. Trac – aktivní tickety.

http://trac.zapletalovi.com/trac/josephine. Přihlašovací jméno „guest“, heslo je stejné. Účet je nastaven pouze pro čtení.

4.3.4. Ant

Ačkoli jsme pro sestavování výsledné aplikace (release engineering) používali výhradně Eclipse IDE, nástroj Ant jsme využili pro sestavování dokumentace. Tento nástroj je popsán v článku [11]:

Ant je konfigurovatelný, flexibilní a plně přenositelný sestavovací systém napsaný v Javě. Jeho primárním cílem je kompilace a sestavování programů v Javě, ale existují určité nadstavby a speciální konfigurace umožňující kompilovat i jiné jazyky, které se zatím v oficiální distribuci bohužel nenacházejí. Ve světě Javy tu byla před Antem černá díra. Sestavování každý řešil po svém. Někdo používal některý standardní make program, někdo měl vlastní a jiný používal sadu shellovských skriptů. Jakmile přišel Ant, komunita ho okamžitě přijala a téměř každý větší GNU projekt v Javě využívá jeho schopností. Ant přijalo mnoho integrovaných prostředí buď formou pluginů (JBuilder, jEdit, Netbeans/SunONE), či dokonce jako pevnou součást prostředí (Eclipse, IDEA).

4.3.5. Jabber

Pro on-line komunikaci (instant messaging) jsme využili protokol Jabber, který je založen na XML a je multiplatformní. Vývoj probíhal paralelně na systémech Windows a Linux a jelikož jsme většinu vývojového času nebyli na dosah,

bylo nutné nějak komunikovat. Občas jsme také použili IP telefonii díky programu Skype.

Jednou za čtrnáct dní jsme se vždy sešli, abychom si vzájemně předvedli svou práci. Součástí schůzky bylo společné řešení dalších problémů, plánování vývoje a také brainstorming, kterým jsme obvykle řešili různé zádrhely nebo technické složitosti.

5. Popis uživatelského rozhraní

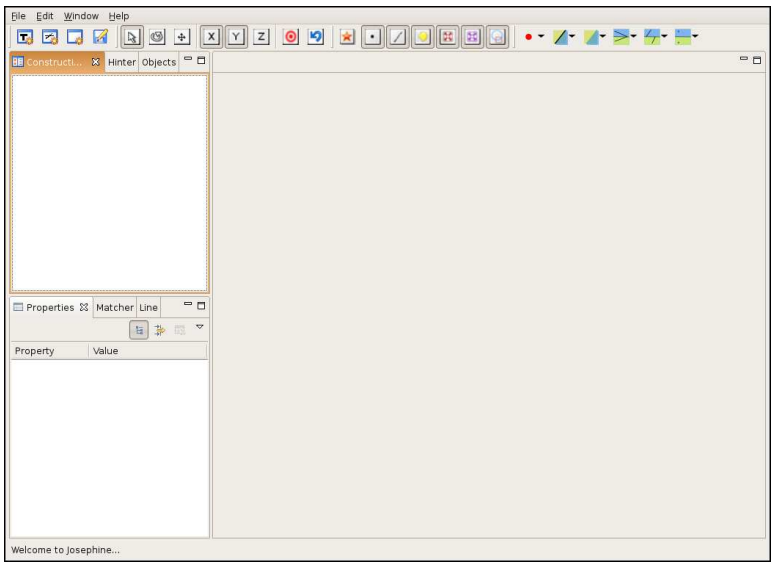
Rozhraní programu je vytvořeno jako standardní grafická aplikace pro platformu Windows.

5.1. Instalace a první spuštění

Instalace programu se provede pouhým nakopírováním adresáře josephine na disk. Spouštění z CDROMu nebo jiného média pouze pro čtení je sice za jistých okolností možné, ale program nebude pracovat správně.

Ještě před vlastním spuštěním je nutné nainstalovat Java Virtual Machine verze 5.0 nebo vyšší. Instalační soubory pro operační systémy Windows a Linux ve variantách pro platformu x86 jsou v adresáři bin/java na doprovodném CD. Lze použít také libovolnou verzi nainstalovanou z internetu (`http://www.java.com`), ale musí být verze 5.0 nebo novější.

Spuštění se provede pomocí souboru josephine.exe, což může být pro některé znalce platformy Java překvapením. Tento program však zavede virtuální stroj jazyka Java a načte potřebné knihovny. Jelikož má kód aplikace téměř 50 tisíc řádků, start aplikace trvá několik vteřin a na obrazovce se po tuto dobu objeví splash screen.



Obrázek 6. Po prvním spuštění.

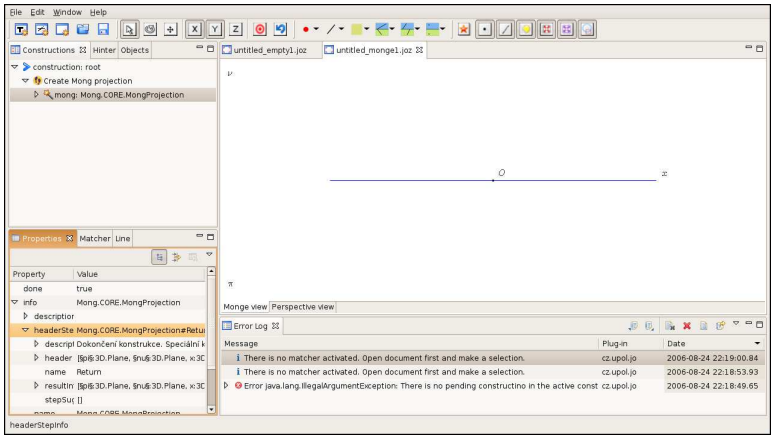
Rozložení ovládacích prvků při prvním spuštění ukazuje obrázek 6. Velká šedá plocha vpravo je místo pro editory, na levé straně je pak otevřeno několik oken. Všechna okna jsou přesouvací a lze je umístit do mnoha pozic.

Okna lze pochopitelně úplně uzavřít, zpětné vyvolání lze provádět z hlavní nabídky Window. Ta nejdůležitější okna jsou přímo v podnabídce a mají také zkratkové klávesy. Ostatní okna, jako je například Error Log, lze aktivovat přes nabídku Other.

Vytvoření nové scény se provede z nabídky nebo toolbaru. K dispozici jsou dvě možnosti, pokud nebudeme počítat testovací scénu: Mongeovo promítání a prázdná scéna. První varianta se liší od prázdné scény tím, že je už automaticky provedena konstrukce půdorysny, nárysny, osy a počátku.

Okno je rozděleno do dvou částí – Mongeovo a perspektivní promítání. Mongeovo promítání se aktivuje pouze, pokud byla provedena konstrukce vytvoření Mongeovy projekce (případně vytvořen dokument v tomto režimu). V levé části jsou umístěna všechna okna, jejichž funkce je následující:

Properties – okno vlastností. Zde se zobrazují veškeré vlastnosti objektů, které nějaké mají. Jsou to zejména primitivní geometrické objekty, konstrukce, respektive popisy konstrukcí – metadata. Přehledně jsou seskupeny a tlačítkem lze také zobrazit pokročilé (advanced) vlastnosti nebo obnovit výchozí hodnotu.



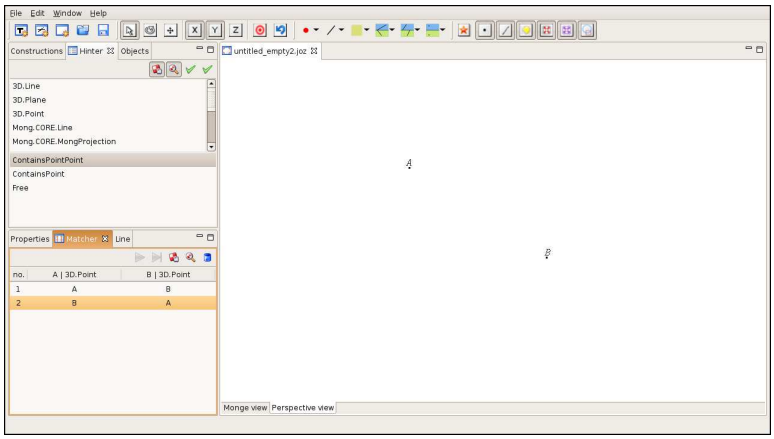
Obrázek 7. Strom konstrukcí, vlastnosti a okno záznamů.

Constructions – strom konstrukcí. Obsahuje strom všech konstrukcí (a podkonstrukcí). Vlastnosti jsou zobrazovány v předešlém okně. Při označení konstrukce v aktuálním dokumentu krátce zablikají všechny primitivní objekty, které jsou touto konstrukcí vytvořeny.

Error Log – zprávy programu. V tomto okně jsou zachyceny všechny zprávy a výstražná hlášení, které program generuje. Uchováván je také čas, záznamy se ukládají na disk. Pomocí několika tlačítek lze záznamy ukládat či zpětně načítat a také samozřejmě okno vyčistit. Všechna tři okna jsou vidět na obrázku 7.

Hinter – výběr konstrukcí. Pomocí těchto dvou seznamů lze vybrat konstrukci (horní seznam) a krok (spodní seznam), který se provede. Po označení kroku se v okně Matcher, které popíšeme v dalším odstavci, nabídnou možnosti provedení konstrukce.

Přepínacími tlačítky je možné nechat si zobrazit jen ty kroky, které jsou proveditelné (implicitně zapnuto). Kroky také umí systém heuristicky třídit od těch nejrelevantnějších (taktéž zapnuto). Zbývajícími tlačítky se provádí buď samostatný krok konstrukce, nebo kompletně celá konstrukce.



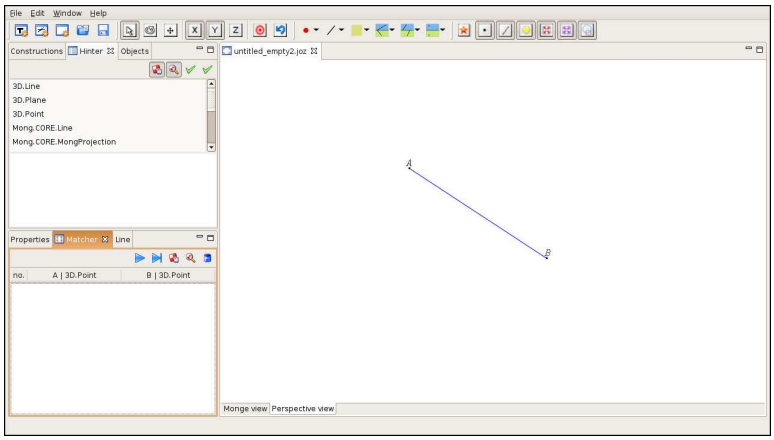
Obrázek 8. Výběr a provedení konstrukce.

Matcher – výběr provedení. Na tomto místě se provádí navázání souvisejících objektů konstrukce. Matcher nabízí všechny varianty možností, které mohou nastat. Jelikož může být seznam dlouhý, je možné si jej buď vygenerovat celý (buď dvojitým klikem na krok v hinteru nebo tlačítkem), nebo po jednom prvku (tlačítkem).

Matcher má další tři tlačítka. Prvním (zprava) se jeho obsah smaže a může být provedeno nové hledání, druhý setřídí nalezené klíčové body podle relevantnosti a třetí se pomocí heuristiky pokusí najít ten nejvhodnější.

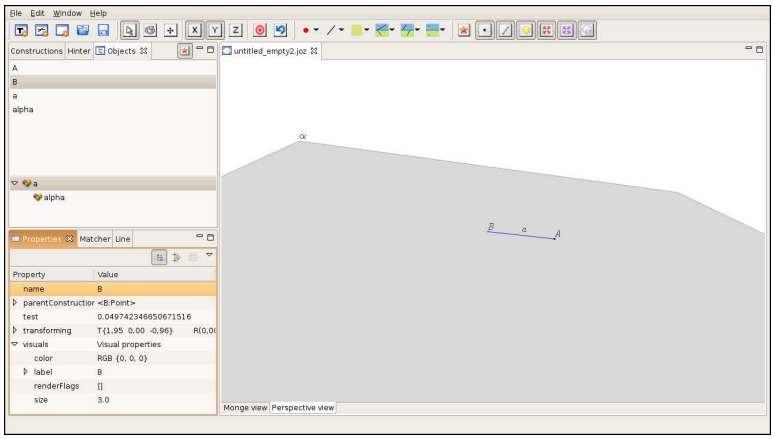
Provedení konstrukce s vybraným klíčovým bodem se pak provádí dvojklikem. V dalším textu blíže popíšeme princip práce s těmito okny, protože se toto ovládání může někomu jevit jako neintuitivní. Po vybrání prvního klíčového bodu z předešlého obrázku (bod A, bod B) bude situace vypadat podobně jako na obrázku 9.

Objects – seznam primitivních objektů. Ačkoli lze objekty ve scéně označovat přímo pomocí myši, výběr více objektů je pohodlnější a přehlednější právě v tomto okně. Opět lze aktivovat okno s vlastnostmi modifikovat zejména skupinu „visuals“, která je pro primitivní objekty zřejmě nejpodstatnější.



Obrázek 9. Po provedení kroku: přímka procházející dvěma body.

Scéna s dvěma body A a B, přímkou a procházející těmito body a volně ležící rovinou α procházející přímkou a lze vidět na obrázku 10.

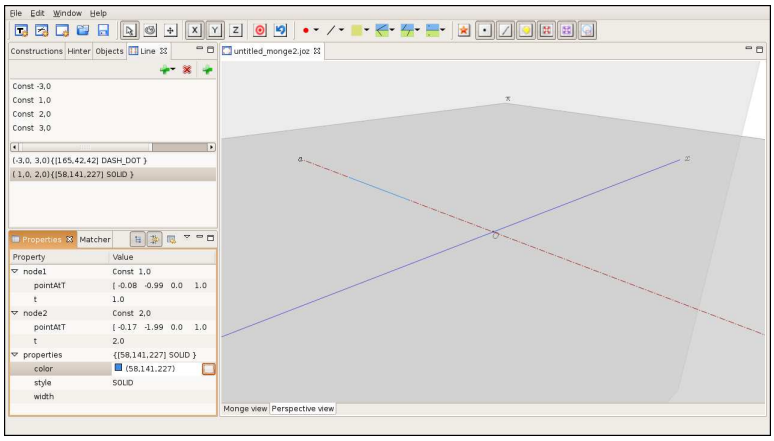


Obrázek 10. Seznam objektů ve scéně.

Line – parametrizace přímky. Na tomto místě lze přidávat přímce pomyslné body v určitých parametrech a dělit tak přímku na úseky. Těmto úsekům pak lze přidělovat různé vizuální vlastnosti (barvu, styl čáry nebo tloušťku).

Okno je rozděleno do dvou částí – seznamů. V horním jsou uzly, v dolním pak segmenty, které jsou nad těmito uzly vytvořeny. Uzly na přímce může uživatel vytvářet pomocí tohoto okna. Vytvořit lze několik typů uzlů. Buď s absolutní hodnotou parametru, relativní vůči jinému uzlu, a nebo průnikem s jiným objektem. Toho systém využívá například pro čárkování přímek v neviditelných kvadrantech v Mongeově promítání.

Nastavení vizuálních vlastností se poté provádí ve spodním seznamu segmentů pomocí okna vlastností.



Obrázek 11. Parametrizace přímky.

5.2. Vytváření konstrukcí

Jak již bylo nastíněno, pro vytváření konstrukcí se používá Matcher a Hinter. Nejdříve je třeba ve vytvářené konstrukci vybrat její počáteční krok. To je nejlepší provést dvojklikem, protože to rovnou aktivuje tento krok v okně Matcher.

Dalším úkolem je z mnoha kombinací klíčových bodů vybrat ten správný a poté podle tohoto klíčového bodu konstrukci vytvořit dvojitým poklepáním na vybraný řádek. Je-li těchto kombinací příliš mnoho, uživatel musí označit některé již vytvořené objekty a stisknout znovu tlačítko (nebo provést dvojklik na krok v Hinteru) pro nové vygenerování klíčových bodů. To omezí hledání pouze na označené objekty. Pokud chce uživatel provést jen jeden krok, musí použít tlačítka v Hinteru.

V tomto případě se vytvoří takzvaně neuzavřená (nedokončená) konstrukce. Zároveň je označena jako aktivní a v okně konstrukcí má ikonu modré šipky. Také v okně Hinter je hvězdičkou zvýrazněný popis konstrukce, která je nedokončená. Systém bude postupně nabízet doporučené kroky, kterými lze v konstrukci pokračovat.

Celá konstrukce se musí dokončit speciálním krokem Return. Pokud by uživatel chtěl během konstrukce nechat systém automaticky konstrukci dokončit, stačí použít tlačítko Finalize construction.

Toto ovládání je přes svou obecnost poněkud těžkopádné, a proto je k dispozici také pomocný toolbar se všemi hlavními konstrukcemi. Je to šest rozbalovacích tlačítek (tři pro volné objekty, tři pro Mongeovo promítání). Při stisku se vždy provede celá konstrukce (všechny kroky).

Jelikož je pro systém životně důležité, aby byly správně vloženy informace o vztazích mezi objekty, je ovládání zvoleno touto formou.

5.3. Transformace

Výběr objektů se provádí myší přímo ve scéně. Pokud je pod kurzorem více objektů, program obrazí kontextovou nabídku. Do aktuálního výběru lze přidávat pomocí klávesy Shift a odebírat pomocí Ctrl.

Na toolbaru jsou ikony pro dvě hlavní transformace – rotace a posunutí. Pro obě transformace je nutno nejdříve vybrat jednu resp. dvě osy, podle kterých se bude objekt transformovat.

Vlastní transformace s vybraným objektem se provádí opět pomocí myši. Rychlé pohyby transformují objekty na větší vzdálenost, pomalé pohyby myši jsou naopak přesnější.

Objekty, které leží v rovině (například bod), se automaticky transformují v dané rovině. Také všechny závislé objekty se při transformaci podřízeného objektu přesunují. Pokud však dojde k tomu, že se jeden z podřízených objektů dostane do neřešitelné pozice (např. bod jako průnik dvou přímek, které by ale byly rovnoběžné), pak se tento objekt označí červeně a eventuálně se přesune do výchozí pozice.

Dvě tlačítka na toolbaru pak mohou resetovat pozici do výchozí pozice (Reset), případně vrátit změny provedené v aktuálním výběru (Rollback).

5.4. Filtry

Protože u konstrukcí v Mongeově promítání může být situace značně nepřehledná, je zde možnost skrývání základních geometrických objektů pomocí filtrů. Filtr lze aktivovat pomocí tlačítka na toolbaru. Skrývat lze podle typu geometrického objektu (bod, přímka, rovina), ale také podle účelu (promítací objekty, promítané objekty a dočasné objekty).

Filtry pracují ještě na jemnější úrovni – pomocí okna vlastností lze vyhledat vlastnost v kategorii Visuals s názvem Render Flags. Lze nastavit takové konstanty, aby byl objekt vždy viditelný (resp. zcela neviditelný), a to bez ohledu na aktuální nastavení filtrů.

Okno geometrických objektů může zobrazit všechny objekty, i když je filtr zapnut. K tomuto účelu slouží ikona malého filtru a vybírat takto lze i skryté objekty.

5.5. Funkce Hold a Fetch

Inspirování programem 3D Studio Max jsme do programu integrovali funkci pozdržení stavu do paměti (Hold) a výběru z paměti (Fetch). V kterémkoliv okamžiku je možné udělat „snímek“ aktuálního stavu do paměti počítače a do

tohoto stavu se pak zase vrátit. Tento výběr z paměti nemusí být proveden ve stejném dokumentu, obsah tak může být zkopírován.

5.6. Ukládání, načítání, export

Program nabízí možnost uložení scény do souboru a opětovné načtení. Jako formát jsme zvolili komprimovaný XML, protože jsme chtěli zajistit maximální kompatibilitu, možnost modifikace vně programu. Scény s větším počtem objektů a závislostí jsou pak poměrně objemné, z tohoto důvodu jsme se tedy rozhodli pro komprimaci. Program je schopen rozpoznat, zda je soubor komprimován, takže lze nahrávat také čisté XML soubory.

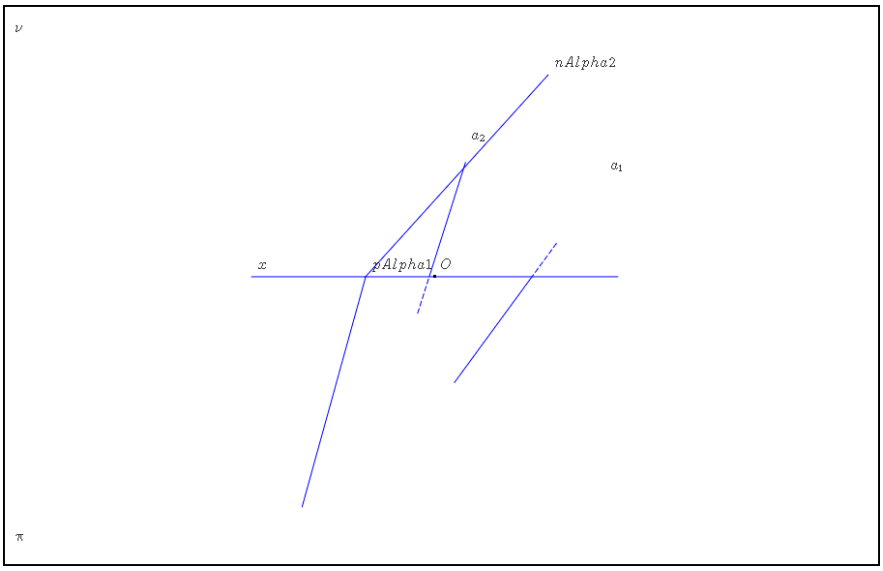
Export aktuálního pohledu do PNG usnadňuje prezentaci výsledku konstrukce. Obě položky jsou dostupné z hlavní nabídky a ukládání s načítáním také přímo z toolbaru.

5.7. Odinstalace

Adresář s programem stačí smazat z disku. V adresáři Documents and Settings (na Windows), nebo /home/uživatel (na Linuxu) stačí pak smazat adresář runtime-jo, do kterého se ukládá nastavení programu. Tím je odinstalace ukončena, program nepoužívá registry systému Windows.

6. Ukázka konstrukce

Naším úkolem bude provést konstrukci průniku přímky s rovinou. Vytvořme tedy novou Mogeovou projekci tlačítky *Mong.Plane: Volně* a *Mong.Line: Volně*.



Obrázek 12. Ukázka konstrukce – vytvoření přímky a bodu.

Nyní je třeba natočit rovinu do takové pozice, aby jejich průnik ležel v prvním kvadrantu v takové vzdálenosti, aby nám výsledek neunikl mimo obrazovku. Označíme tedy přímku $nAlpha_1$ a posuneme ji vlevo dolů, obdobně $nAlpha_2$ vlevo nahoru. Musíme dát pozor na to, aby se přímky protnuly v jednom bodě na ose x. Toho docílíme tím, že si scénu přiblížíme kolečkem myši a plynulými pohyby přesuneme přímku do požadované polohy.

Pokud bychom tak neučinili, výsledná rovina α by měla červenou barvu. Podobně označte přímku a_1 a posuňte ji mírně doprava. Výsledek je vidět na obrázku 12.

Nyní se můžeme přepnout do perspektivního promítání, aktivovat filtr a vypnout pomocné objekty (projected, projector). Levým tlačítkem myši lze scénu natočit do lepšího úhlu. Pro přehlednost obarvíme rovinu i přímku jinou barvou.

Po vybrání stačí aktivovat okno s vlastnostmi, u roviny stačí nastavit hodnotu Visuals.Color, u přímky pak Visuals.Segmentation.Segments.Proper.Color. Vybíráme světlejší a tmavší fialovou barvu.

Přímku také protáhneme – změníme jí parametrizaci. Vybereme okno Line, označíme ji a v horním seznamu změníme parametry z $-3, 3$ na $-10, 10$. Přímka se výrazně protáhne, situace bude jako na obrázku 13.

Nyní je třeba vytvořit stopníky zadané přímky, tedy body, ve kterých přímka protíná nárysnu (N_1, N_2) a půdorysnu (P_1, P_2). Toto provedeme přes Hinter

(Mong.CORE.Stoppers a klikem na ikonu Finish Construction). Obrázky jsou vytvořeny za použití filtrů, aby byly přehlednější.

V dalším kroku je potřeba vytvořit pomocnou rovinu β, která bude procházet přímkou a. Využijeme tedy stopníků a zkonstruujeme dvě přímky procházející body P_1, O a O, N_2.

K tomu, abychom zkonstruovali rovinu v prostoru, budeme potřebovat stopy roviny v Mongeově promítání. Z těchto stop teprve můžeme zkonstruovat rovinu v prostoru. Vytvoříme tedy dvě přímky v prostoru procházející dvěma body.

K tomu použijeme tlačítko na toolbaru, poklepáním na něj se objeví Matcher, ve kterém vybereme vyhovující řádek obsahující postupně body P_1, O, resp. O, N_2. Abychom snížili počet řádků v matcheru a pohodlněji našli potřebnou kombinaci, stačí objekty, na kterých bude konstrukce závislá, před stiskem tlačítka označit.

Označení více objektů lze provést přímo ve scéně pomocí klávesy Shift, nebo v seznamu objektů v okně Objects klávesou Ctrl. Tento postup funguje vždy, takže jej nadále nebudeme v této kapitole explicitně uvádět.

Nyní vytvoříme rovinu β pomocí tlačítka Rovina: ze stop. Opět lze výběr kombinace zrychlit tím, že si obě vytvořené přímky nejprve vybereme. Je vhodné tyto přímky po vytvoření přejmenovat, aby byla celá situace přehlednější. Barvu i název vytvořené roviny je vhodně také změnit.

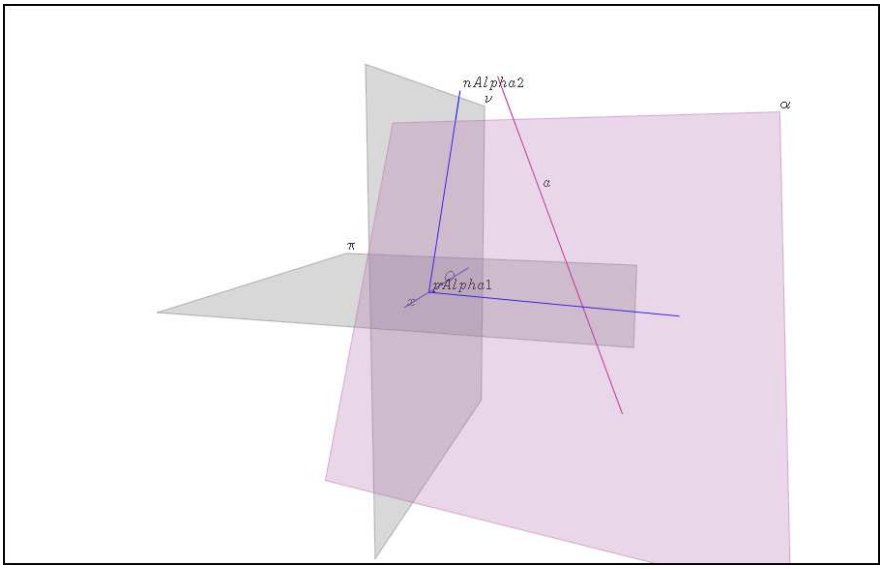
Jako průniky stop rovin α a β vytvoříme body B_1, B_2. Jedná se tedy o konstrukci bodů jako průnik dvou přímek. V tomto zadání se body vytvořily ve větší vzdálenosti od počátku, nicméně jelikož se jedná o body pomocné, tak nám to nevadí.

Zkonstruujeme ordinály bodů B_1, B_2, tedy kolmice na osu x procházející těmito body. V programu se tedy znovu jedná pouze o označení vstupních objektů (osa, bod, rovina) a stisknutí tlačítka (přímka procházející bodem, kolmá na přímku, ležící v rovině – tedy v půdorysně nebo nárysně).

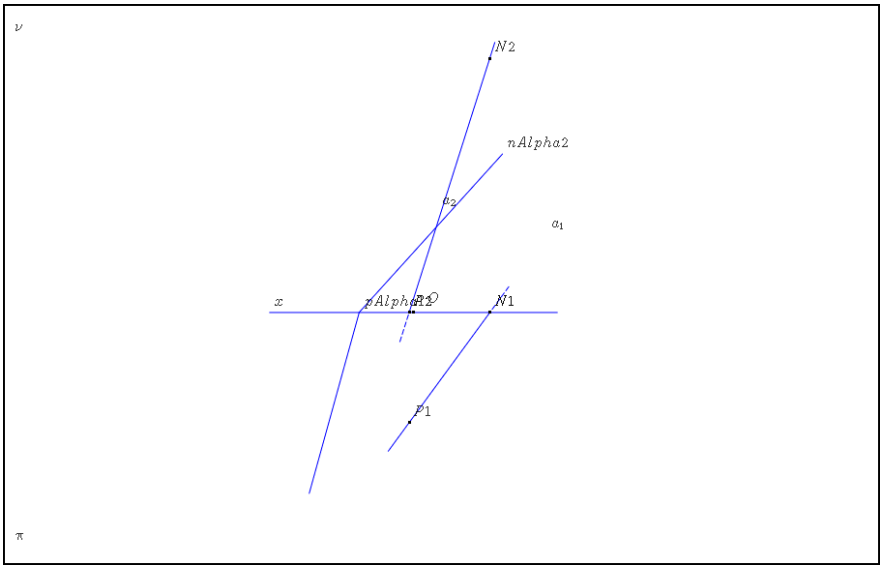
Pro přehlednost jsme vytvořili ordinály čárkovaně, jak je vidět na obrázku 17.

Jsme téměř hotovi, ordinály protínají osu x a na těchto průnicích vytvoříme body X_1, X_2. Pomocné body poté spojíme přímkami (B_1, X_1 a B_2, X_2), které nakonec protnou stopy zadané přímky. Tyto průniky, které existují v případě, že zadaná přímka není rovnoběžná se zadanou rovinou, nazveme S_1, S_2.

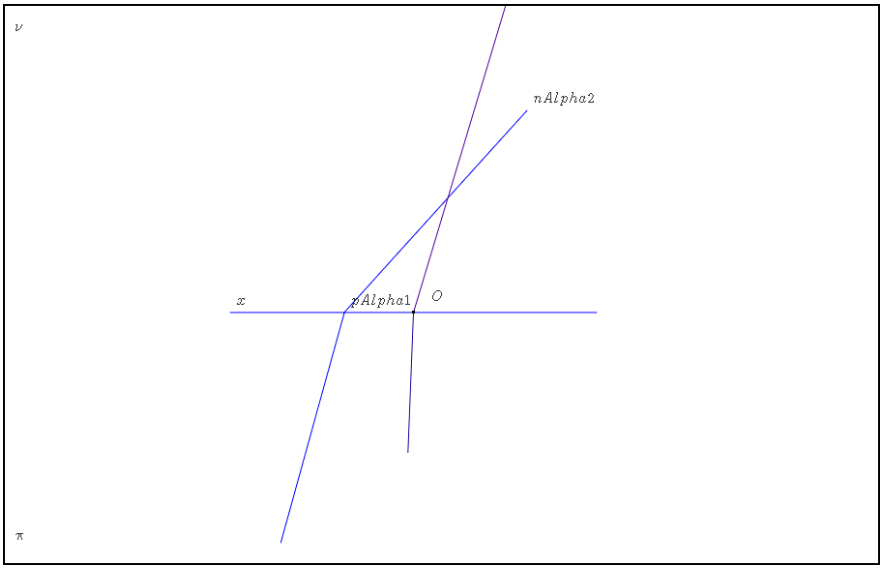
Od konečného výsledku nás dělí jen jeden stisk tlačítka myši. Vybereme body S_1, S_2 a pomocí tlačítka vybereme bod v Mongeově projekci.



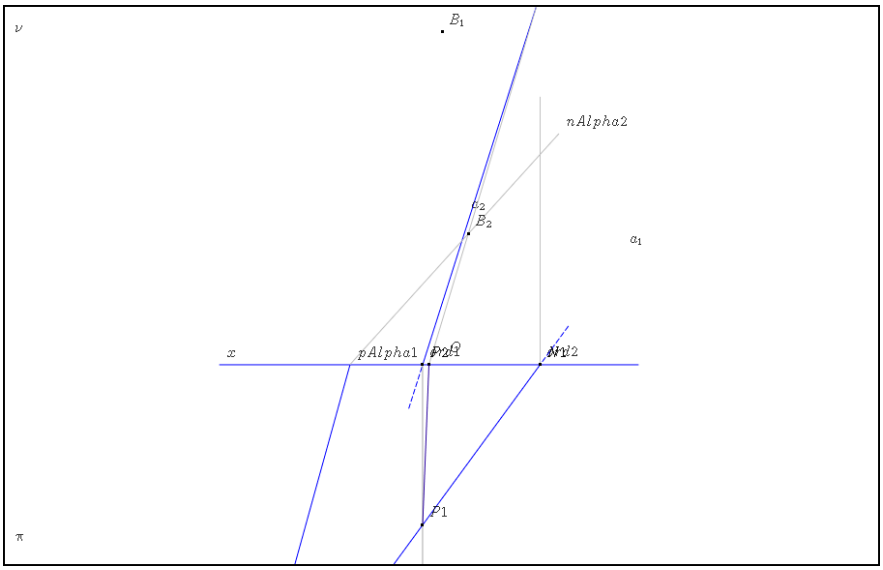
Obrázek 13. Ukázka konstrukce – pohled na zadání.



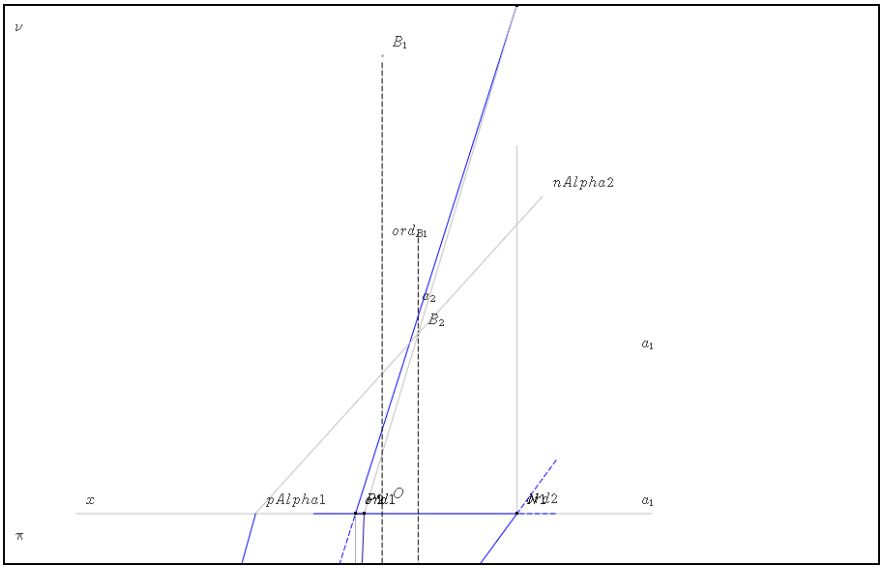
Obrázek 14. Ukázka konstrukce – stopníky přímky.



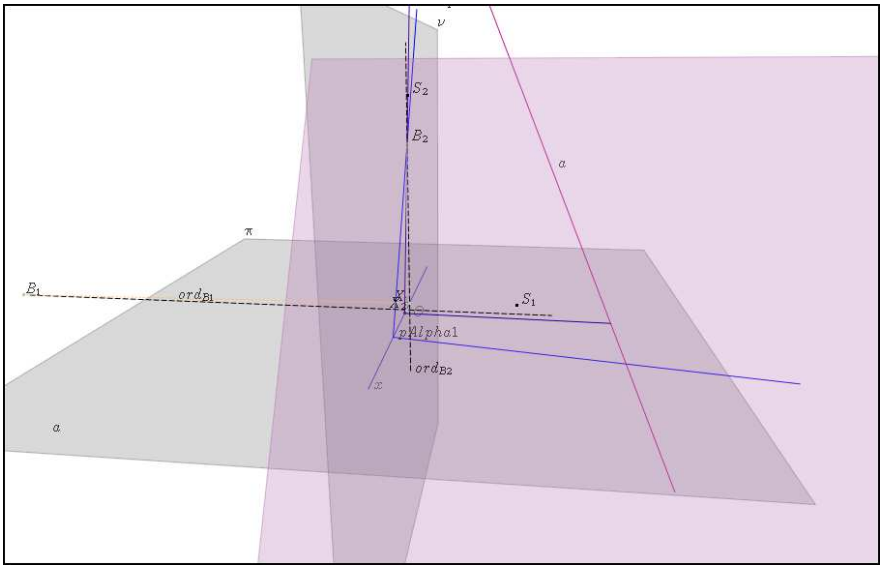
Obrázek 15. Ukázka konstrukce – vytvoření pomocné roviny.



Obrázek 16. Ukázka konstrukce – pomocné body B_1, B_2.



Obrázek 17. Ukázka konstrukce – ordinály bodů B_1, B_2.



Obrázek 18. Ukázka konstrukce – výsledný bod v Mongeově projekci.

Závěr

Tato diplomová práce nás naučila vedení projektu, týmové spolupráci, dělbě dílčích úkolů, komunikaci a práci se systémem správy verzí. Tedy věcem, které se při vypracování diplomové práce jako jednotlivec vůbec nenaučíte nebo si je nevyzkoušíte.

Cílem našeho snažení bylo vytvořit obecný systém pro syntaktické odvozování pravidel a ověření správnosti na geometrických úlohách z deskriptivní geometrie. Tento systém by měl umožnit kontrolu úloh po syntaktické stránce. V samotném závěru naší práce nezbývá než konstatovat, že jsme udělali vše proto, aby se nám podařilo vytýčeného cíle dosáhnout.

Conclusions

This MSc. thesis teached us project management, team cooperation, division of labour, communication and work with revision control systems. Things which you probably won‘t learn or even try by working as individual.

Our objective was to create an universal system for syntactic derivation rules and verification of legitimacy on descriptive geometry. Modelled system should enabled syntactic verification of tasks. At the very end of our work it left to state we did our best to bring designated objective off.

Reference

[1] Bartsch, Hans – Jochen. *Matematické vzorce* SNTL – nakladatelství technické literatury, Praha, 1983.

[2] Brackeen D., Barker B., Vanhelsuw L. *Vývoj her v jazyce Java* Grada Publishing, Praha, 2004.

[3] Bloch, Joshua. *Java efektivně – 57 zásad softwarového experta* Grada Publishing, Praha, 2002.

[4] Drdla, Josef. *Počítačová geometrie* Skripta UP, Olomouc, 1991.

[5] Drdla, Josef. *Geometrické modelování křivek a ploch* Učební texty UP, Olomouc, 2001.

[6] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software* Addison-Wesley, USA, 1995.

[7] McAffer J., Lemieux J. M. *Eclipse Rich Client Platform: Designing, Coding and Packaging Java Applications* Addison-Wesley Professional, Boston, 2005.

[8] Naider J., Davis T., Woo M. *The OpenGL Programming Guide, 4rd. ed.* Addison-Wesley Professional, Boston, 2004.

[9] Rogelberg, D. (ed.) *The OpenGL Reference Manual* Elektronická publikace, 1992.

[10] Urban, Alois. *Deskriptivní geometrie 1* Státní nakladatelství technické literatury, Praha, 1965.

[11] Zapletal, Lukáš. *Apache Ant 1.5* Elektronický článek, 2001.

A. Příloha 1: Popis jazyku pro generování popisků

Pro popisky objektů jsme museli vyřešit způsob zápisu matematických symbolů, písmen řecké abecedy a také horních a dolních indexů, kterých se v deskriptivní geometrii často využívá.

Využili jsme tedy sady matematických fontů, které jsou dostupné v bitmapách jako public domain (zdarma k volnému použití). Sadu vytvořil Norman Megill a je k dispozici na adrese http://us.metamath.org/downloads/symbols.zip. Sada písem sice neobsahuje diakritiku, na druhou však nabízí matematické symboly, znaky řecké abecedy, horní a dolní indexy a podobně.

Syntaxe je velice jednoduchá a je postavena nad několika párovými znaky nebo příkazy uvozenými znakem paragrafu. Jsou to tyto řídicí znaky:

^ - párový znak zapíná nebo vypíná horní index (např. 2^x = 2^x^);

_ - párový znak zapíná nebo vypíná dolní index (např. x_{i+1} = x_i+1_);

' - jednoduchým apostrofem lze dělat čárkování symbolů (α' = §alpha§');

@ - znakem „at“ (také známý jako zavináč) lze dělat tečkování symbolů ($b^{\cdot\cdot}$ = b@@);

- znak hash slouží k hvězdičkování (x^* = x#);

§ - párový znak paragrafu slouží pro speciální řídicí kombinace (vizte níže).

Znak paragrafu je zvolen záměrně, protože je snadno dostupný na české klávesnici. Pokud je však uživatel navyklý na jiné rozložení klávesnice, je možno místo znaku paragraf použít obrácené lomítko (se stejným významem).

Možné jsou změny řezů písma:

§Reg§ – základní písmo (matematická italika);

§Rom§ – svislé písmo (pro delší úseky textu);

§Bof§ – tučné písmo (svislé);

§Cal§ – kaligrafické psací;

§Scr§ – kaligrafické ozdobné (skloněné);

§Bkb§ – zdvojené velké (blackboard);

§Fra§ – ozdobné (fraktura).

U ostatních znaků jiných abeced, speciálních symbolů, matematických symbolů a šipek se používají speciální řídicí kombinace ve tvaru §symbol§. U řídicích znaků nezáleží na velikosti písmen, takže „Symbol“ je totéž, co „symbol“ nebo „SYMBOL“. Z nejdůležitějších zde uvádíme:

- malá písmena řecké abecedy píšeme přímo malými písmeny: §alpha§, §beta§, atd.;
- velká písmena řecké abecedy začínají znakem „c“: §cgamma§, §cdelta§, atd.;
- písmena hebrejské abecedy píšeme přímo: §aleph§, §daleth§ a podobně.

Poznámka: pro řeckou abecedu existují také počeštěné aliasy, takže budou fungovat i řídicí znaky jako §alfa§, §gama§, §jota§ či §ny§ (se stejným významem jako anglické §alpha§, §gamma§, §iota§ resp. §nu§). Podobně kapitálky (§calfa§ a podobně).

Úplně všechny znaky, které je program schopen zobrazovat, jsou dostupné přes tyto speciální příkazy. Ačkoli se užití například kombinace §cx§ místo přímého napsání velkého písmena X může zdát nevýhodná, lze této vlastnosti využít pro zkrácení zápisu.

Pokud je nastaven například obyčejný řez (matematická italika) a potřebujeme vypsat jen jeden jediný znak kaligrafickým řezem, museli bychom přepnout řez, poté vložit znak (např. x) a změnit řez zpět na normální: §Cal§x§Reg§. Můžeme však využít přímo symbol pro kaligrafický znak x: §calx§.

A.1. Seznam všech příkazů

V této části uvádíme kompletní seznam všech rozpoznávaných příkazů. Jaké znaky vzniknou použitím těchto příkazů, zjistíme v tabulce A.2. Příkazy jsou děleny do logických skupin, které jsou očíslovány a je jich celkem 50.

1 (Roman) :

```
§rma§ §rmb§ §rmc§ §rmd§ §rme§ §rmf§ §rmg§ §rmh§ §rmi§ §rmj§
§rmk§ §rml§ §rmm§ §rmn§ §rmo§ §rmp§ §rmq§ §rmr§ §rms§ §rmt§
§rmu§ §rmv§ §rmw§ §rmx§ §rmy§ §rmz§
```

2 (Roman caps) :

```
§rmca§ §rmcb§ §rmcc§ §rmcd§ §rmce§ §rmcf§ §rmcg§ §rmch§ §rmci§
§rmcj§ §rmck§ §rmcl§ §rmcm§ §rmcn§ §rmco§ §rmcp§ §rmcq§ §rmcr§
§rmcs§ §rmct§ §rmcu§ §rmcv§ §rmcw§ §rmcx§ §rmcy§ §rmcz§
```

3 (math italic) :

§a§ §b§ §c§ §d§ §e§ §f§ §g§ §h§ §i§ §j§ §k§ §l§ §m§ §n§ §o§ §p§ §q§ §r§ §s§ §t§ §u§ §v§ §w§ §x§ §y§ §z§

4 (math italic caps) :

§ca§ §cb§ §cc§ §cd§ §ce§ §cf§ §cg§ §ch§ §ci§ §cj§ §ck§ §cl§ §cm§ §cn§ §co§ §cp§ §cq§ §cr§ §cs§ §ct§ §cu§ §cv§ §cw§ §cx§ §cy§ §cz§

5 (bold) :

§bfa§ §bfb§ §bfc§ §bfd§ §bfe§ §bff§ §bfg§ §bfh§ §bfi§ §bfj§ §bfk§ §bfl§ §bfm§ §bfn§ §bfo§ §bfp§ §bfq§ §bfr§ §bfs§ §bft§ §bfu§ §bfv§ §bfw§ §bfx§ §bfy§ §bfz§

6 (bold caps) :

§bfca§ §bfcb§ §bfcc§ §bfcd§ §bfce§ §bfcf§ §bfcg§ §bfch§ §bfci§ §bfcj§ §bfck§ §bfcl§ §bfcm§ §bfcn§ §bfco§ §bfcp§ §bfcq§ §bfcr§ §bfcs§ §bfct§ §bfcu§ §bfcv§ §bfcw§ §bfcx§ §bfcy§ §bfcz§

7 (digits and miscellaneous) :

§0§ §1§ §2§ §3§ §4§ §5§ §6§ §7§ §8§ §9§ §emptyset§ §circleds§ §backp§ §doubles§ §yen§ §copyright§ §dollar§ §pounds§ §ss§ §flat§ §natural§ §sharp§ §clubsuit§ §diamondsuit§ §heartsuit§ §spadesuit§ §ddag§ §vareighthnote§ §pd§ §nc§

8 (Greek) :

§alpha§ §beta§ §gamma§ §delta§ §epsilon§ §varepsilon§ §zeta§ §eta§ §theta§ §vartheta§ §iota§ §kappa§ §lambda§ §mu§ §nu§ §xi§ §pi§ §varpi§ §rho§ §varrho§ §sigma§ §varsigma§ §tau§ §upsilon§ §varphi§ §phi§ §chi§ §psi§ §omega§

9 (Greek caps) :

§cgamma§ §cdelta§ §ctheta§ §clambda§ §cxi§ §cpi§ §csigma§ §cupsilon§ §cphi§ §cpsi§ §comega§

10 (Hebrew and letter-like symbols) :

§aleph§ §varaleph§ §beth§ §gimel§ §daleth§ §amalg§ §imath§ §jmath§ §ell§ §wp§ §re§ §im§ §partial§ §nabla§ §sum§ §prod§ §coprod§ §digamma§ §varkappa§ §hbar§ §hslash§ §mho§ §finv§

§game§ §bbbk§ §complement§ §eth§ §backepsilon§ §subscrh§
§subscrl§

11 (calligraphic) :
§cala§ §calb§ §calc§ §cald§ §cale§ §calf§ §calg§ §calh§ §cali§
§calj§ §calk§ §call§ §calm§ §caln§ §calo§ §calp§ §calq§ §calr§
§cals§ §calt§ §calu§ §calv§ §calw§ §calx§ §caly§ §calz§

12 (script) :
§scra§ §scrb§ §scrc§ §scrd§ §scre§ §scrf§ §scrg§ §scrh§ §scri§
§scrj§ §scrk§ §scrl§ §scrm§ §scrn§ §scro§ §scrp§ §scrq§ §scrr§
§scrs§ §scrt§ §scru§ §scrv§ §scrw§ §scrx§ §scry§ §scrz§

13 (blackboard bold) :
§bba§ §bbb§ §bbc§ §bbd§ §bbe§ §bbf§ §bbg§ §bbh§ §bbi§ §bbj§
§bbk§ §bbl§ §bbm§ §bbn§ §bbo§ §bbp§ §bbq§ §bbr§ §bbs§ §bbt§
§bbu§ §bbv§ §bbw§ §bbx§ §bby§ §bbz§

14 (fractur) :
§fraka§ §frakb§ §frakc§ §frakd§ §frake§ §frakf§ §frakg§ §frakh§
§fraki§ §frakj§ §frakk§ §frakl§ §frakm§ §frakn§ §frako§ §frakp§
§frakq§ §frakr§ §fraks§ §frakt§ §fraku§ §frakv§ §frakw§ §frakx§
§fraky§ §frakz§

15 (fractur caps) :
§frakca§ §frakcb§ §frakcc§ §frakcd§ §frakce§ §frakcf§ §frakcg§
§frakch§ §frakci§ §frakcj§ §frakck§ §frakcl§ §frakcm§ §frakcn§
§frakco§ §frakcp§ §frakcq§ §frakcr§ §frakcs§ §frakct§ §frakcu§
§frakcv§ §frakcw§ §frakcx§ §frakcy§ §frakcz§

16 (subscripts) :
§suba§ §subb§ §subc§ §subd§ §sube§ §subf§ §subg§ §subh§ §subi§
§subj§ §subk§ §subl§ §subm§ §subn§ §subo§ §subp§ §subq§ §subr§
§subs§ §subt§ §subu§ §subv§ §subw§ §subx§ §suby§ §subz§ §sub0§
§sub1§ §sub2§ §sub3§ §sub4§ §sub5§ §sub6§ §sub7§ §sub8§ §sub9§

17 (superscripts) :
§supa§ §supb§ §supc§ §supd§ §supe§ §supf§ §supg§ §suph§ §supi§

§supj§ §supk§ §supl§ §supm§ §supn§ §supo§ §supp§ §supq§ §supr§
§sups§ §supt§ §supu§ §supv§ §supw§ §supx§ §supy§ §supz§ §sup0§
§sup1§ §sup2§ §sup3§ §sup4§ §sup5§ §sup6§ §sup7§ §sup8§ §sup9§

18 (capital subscripts) :

§subca§ §subcb§ §subcc§ §subcd§ §subce§ §subcf§ §subcg§ §subch§
§subci§ §subcj§ §subck§ §subcl§ §subcm§ §subcn§ §subco§ §subcp§
§subcq§ §subcr§ §subcs§ §subct§ §subcu§ §subcv§ §subcw§ §subcx§
§subcy§ §subcz§ §subinfty§ §sublp§ §subrp§ §subplus§ §subminus§

19 (capital superscripts) :

§supca§ §supcb§ §supcc§ §supcd§ §supce§ §supcf§ §supcg§ §supch§
§supci§ §supcj§ §supck§ §supcl§ §supcm§ §supcn§ §supco§ §supcp§
§supcq§ §supcr§ §supcs§ §supct§ §supcu§ §supcv§ §supcw§ §supcx§
§supcy§ §supcz§ §supinfty§ §suplp§ §suprp§ §supplus§ §supminus§

20 (Greek subscripts) :

§subalpha§ §subbeta§ §subgamma§ §subdelta§ §subepsilon§
§subvarepsilon§ §subzeta§ §subeta§ §subtheta§ §subvartheta§
§subiota§ §subkappa§ §sublambda§ §submu§ §subnu§ §subxi§
§subpi§ §subvarpi§ §subrho§ §subvarrho§ §subsigma§
§subvarsigma§ §subtau§ §subupsilon§ §subvarphi§ §subphi§
§subchi§ §subpsi§ §subomega§ §subeq§ §sublt§ §suble§ §subgt§
§subge§

21 (Greek superscripts) :

§supalpha§ §supbeta§ §supgamma§ §supdelta§ §supepsilon§
§supvarepsilon§ §supzeta§ §supeta§ §suptheta§ §supvartheta§
§supiota§ §supkappa§ §suplambda§ §supmu§ §supnu§ §supxi§
§suppi§ §supvarpi§ §suprho§ §supvarrho§ §supsigma§
§supvarsigma§ §suptau§ §supupsilon§ §supvarphi§ §supphi§
§supchi§ §suppsi§ §supomega§ §supeq§ §suplt§ §suple§ §supgt§
§supge§

22 :

§llangle§ §llbrack§ §rrangle§ §rrbrack§ §vdvdash§ §lt§ §join§
§percent§ §minus§ §shortminus§ §period§ §smallprime§ §atsign§
§bigdiamond§ §semicolon§ §checkmark§ §bigbox§ §comma§ §plus§
§eq§ §colon§ §notsubset§ §backtick§ §apostrophe§ §backquote§

§quote§ §octothorpe§

23 :

§vdots§ §ldots§ §notapprox§ §maltese§ §notsucccurlyeq§ §questionmark§ §invquestion§ §ddots§ §gt§ §notpreccurlyeq§ §notsupset§ §cdots§ §largetimes§ §acute§ §bar§ §vec§ §dot§ §ddot§ §hat§ §supfrown§ §grave§ §tilde§ §check§ §breve§ §supast§ §subin§ §notsim§

24 :

§bigtriangleup§ §bigtriangledown§ §vee§ §wedge§ §oplus§ §ominus§ §otimes§ §oslash§ §odot§ §dagger§ §amp§ §le§ §prec§ §preceq§ §ll§ §subset§ §subseteq§ §sqsubseteq§ §in§

25 :

§vdash§ §smile§ §frown§ §ne§ §ge§ §succ§ §succeq§ §gg§ §supset§ §supseteq§ §sqsupseteq§ §owns§ §dashv§ §parallel§ §notin§ §equiv§ §sim§ §simeq§

26 :

§asymp§ §approx§ §cong§ §bowtie§ §propto§ §models§ §doteq§ §perp§ §supperp§ §infty§ §smallint§ §prime§ §surd§ §top§ §forall§ §exists§ §lnot§ §leftarrow§ §bigleftarrow§ §to§ §subto§ §bigto§

27 :

§leftrightarrow§ §bigleftrightarrow§ §uparrow§ §updownarrow§ §nearrow§ §swarrow§ §mapsto§ §hookleftarrow§ §leftharpoonup§ §rightharpoonup§ §rightleftharpoons§ §longleftarrow§ §biglongleftarrow§ §onetoone§ §onto§

28 :

§longrightarrow§ §biglongrightarrow§ §longleftrightarrow§ §biglongleftrightarrow§ §biguparrow§ §bigdownarrow§ §bigupdownarrow§ §searrow§ §nwarrow§ §longmapsto§ §hookrightarrow§ §leftharpoondown§ §rightharpoondown§ §onetooneonto§

29 :

§bigvee§ §bigcap§ §bigcup§ §bigsqcup§ §biguplus§ §bigodot§ §bigotimes§ §bigoplus§ §bigwedge§ §int§ §oint§ §lp§ §rp§ §lfloor§ §rfloor§ §vert§ §lbrack§ §rbrack§ §lceil§ §rceil§ §lbrace§ §rbrace§ §langle§ §rangle§ §solidus§

30 :

§bang§ §downdownarrows§ §lplp§ §rprp§ §downarrow§ §invbang§ §vartriangle§ §triangledown§ §square§ §lozenge§ §angle§ §measuredangle§ §nexists§ §backprime§ §varnothing§ §blacktriangle§ §blacktriangledown§ §blacksquare§ §blacklozenge§ §bigstar§ §sphericalangle§ §diagup§ §diagdown§ §dotplus§ §smallsetminus§

31 :

§doublecap§ §doublecup§ §barwedge§ §veebar§ §doublebarwedge§ §boxminus§ §boxtimes§ §boxdot§ §boxplus§ §divideontimes§ §ltimes§ §rtimes§ §leftthreetimes§ §rightthreetimes§ §curlywedge§ §curlyvee§

32 :

§circleddash§ §circledast§ §circledcirc§ §centerdot§ §intercal§ §leqq§ §leqslant§ §eqslantless§ §lesssim§ §lessapprox§ §approxeq§ §lessdot§ §llless§ §lessgtr§ §lesseqgtr§ §lesseqqgtr§

33 :

§doteqdot§ §risingdotseq§ §fallingdotseq§ §backsim§ §backsimeq§ §subseteqq§ §ssubset§ §sqsubset§ §preccurlyeq§ §curlyeqprec§ §precsim§ §precapprox§ §vartriangleleft§ §trianglelefteq§ §vddash§ §vvdash§

34 :

§smallsmile§ §smallfrown§ §bumpeq§ §bbumpeq§ §varpropto§ §blacktriangleleft§ §therefore§ §geqq§ §geqslant§ §eqslantgtr§ §gtrsim§ §gtrapprox§ §gtrdot§ §gggtr§

35 :

§gtrless§ §gtreqless§ §gtreqqless§ §eqcirc§ §circeq§

§triangleq§ §thicksim§ §thickapprox§ §supseteqq§ §ssupset§ §sqsupset§ §succcurlyeq§ §curlyeqsucc§ §succsim§ §succapprox§ §vartriangleright§ §trianglerighteq§

36 :

§v2dash§ §shortmid§ §shortparallel§ §between§ §pitchfork§ §blacktriangleright§ §because§ §nless§ §nleq§ §nleqslant§ §nleqq§ §lneqq§ §lneq§ §lvertneqq§ §lnsim§ §lnapprox§

37 :

§ngtr§ §ngeq§ §ngeqslant§ §ngeqq§ §gneq§ §gneqq§ §gvertneqq§ §gnsim§ §gnapprox§ §nprec§ §npreceq§ §precneqq§ §precnsim§ §precnapprox§ §nsim§ §nshortmid§ §nmid§

38 :

§nvdash§ §nv2dash§ §ntriangleleft§ §ntrianglelefteq§ §nsubseteq§ §nsubseteqq§ §subsetneq§ §varsubsetneq§ §subsetneqq§ §varsubsetneqq§ §nsucc§ §nsucceq§ §succneqq§ §succnsim§ §succnapprox§ §ncong§

39 :

§nshortparallel§ §nparallel§ §nvddash§ §nvvddash§ §ntriangleright§ §ntrianglerighteq§ §nsupseteq§ §nsupseteqq§ §supsetneq§ §varsupsetneqq§ §supsetneqq§ §varsupsetneq§ §leftleftarrows§ §leftrightarrows§ §lleftarrow§

40 :

§twoheadleftarrow§ §leftarrowtail§ §looparrowleft§ §leftrightharpoons§ §curvearrowleft§ §circlearrowleft§ §lsh§ §upuparrows§ §upharpoonleft§ §downharpoonleft§ §multimap§ §leftrightsquigarrow§ §rightrightarrows§ §rightleftarrows§ §rrightarrow§

41 :

§twoheadrightarrow§ §rightarrowtail§ §looparrowright§ §curvearrowright§ §circlearrowright§ §rsh§ §restriction§ §downharpoonright§ §rightsquigarrow§ §nleftarrow§ §nbigleftarrow§ §nleftrightarrow§ §nrightarrow§

§nbigrightarrow§ §nbigleftrightarrow§

42 :
§pm§ §mp§ §setminus§ §cdot§ §times§ §ast§ §star§ §diamond§ §circ§ §bullet§ §div§ §cap§ §cup§ §uplus§ §sqcap§ §sqcup§ §triangleleft§ §triangleright§ §wr§ §bigcirc§

43 :
§pfun§ §ffun§ §psurj§ §bij§ §pinj§ §finj§ §defs§ §ndres§ §nrres§ §smallcirc§ §spot§ §semi§ §inbag§ §uminus§ §limg§ §rimg§ §lblot§ §rblot§ §osmallplus§ §osmalltimes§ §filledsquarewithdots§ §squarewithdots§ §convolution§

44 :
§currency§ §cent§ §wlozenge§ §kreuz§ §smiley§ §blacksmiley§ §frownie§ §sun§ §brokenvert§ §diameter§ §invdiameter§ §phone§ §recorder§ §clock§ §permil§ §bell§ §ataribox§ §pointer§ §lightning§ §photon§ §gluon§ §eighthnote§ §quarternote§ §halfnote§ §fullnote§ §twonotes§

45 :
§vhf§ §aplbox§ §aplinv§ §aplleftarrowbox§ §aplrightarrowbox§ §apluparrowbox§ §apldownarrowbox§ §aplinput§ §aplminus§ §apllog§ §aplstar§ §aplvertdown§ §aplnotdown§ §aplnotland§ §aplnotlor§ §aplcirc§ §aplcircbot§ §notbackslash§ §notslash§ §aplcomment§ §desnode§ §astrosun§ §newmoon§ §fullmoon§ §leftmoon§ §rightmoon§

46 :
§mercury§ §venus§ §mars§ §jupiter§ §saturn§ §uranus§ §neptune§ §pluto§ §earth§ §conjunction§ §opposition§ §ascnode_or_leo§ §vernal_or_aries§ §libra§ §taurus§ §scorpio§ §gemini§ §sagittarius§ §cancer§ §capricornus§ §aquarius§ §virgo§ §pisces§ §hexstar§ §varhexstar§

47 :
§davidstar§ §leftcircle§ §leftcircleb§ §rightcircle§ §rightcircleb§ §leftbcircle§ §rightbcircle§ §wbox§ §xbox§ §wbowtie§ §wdiamond§ §octagon§ §hexagon§ §varhexagon§

§pentagon§ §varangle§ §invneg§ §leftturn§ §rightturn§
§wvarpropto§ §leadsto§ §varint§ §iint§ §iiint§ §varoint§
§oiint§ §thorn§ §cthorn§ §dh§ §cdh§ §openo§ §inve§

48 :

§moon§ §varuranus§ §varneptune§ §varpluto§ §skull§
§biohazard§ §radiation§ §laserbeam§ §textdbend§ §stopsign§
§bicycle§ §blitza§ §mayazero§ §jackstar§ §sixteenstarlight§
§snowflakechevron§ §scissorright§ §scissorleft§ §handright§
§handleft§ §bighandright§ §bighandleft§

49 (Roman subscripts) :

§subrma§ §subrmb§ §subrmc§ §subrmd§ §subrme§ §subrmf§ §subrmg§
§subrmh§ §subrmi§ §subrmj§ §subrmk§ §subrml§ §subrmm§ §subrmn§
§subrmo§ §subrmp§ §subrmq§ §subrmr§ §subrms§ §subrmt§ §subrmu§
§subrmv§ §subrmw§ §subrmx§ §subrmy§ §subrmz§

50 (Roman capital subscripts) :

§subrmca§ §subrmcb§ §subrmcc§ §subrmcd§ §subrmce§ §subrmcf§
§subrmcg§ §subrmch§ §subrmci§ §subrmcj§ §subrmck§ §subrmcl§
§subrmcm§ §subrmcn§ §subrmco§ §subrmcp§ §subrmcq§ §subrmcr§
§subrmcs§ §subrmct§ §subrmcu§ §subrmcv§ §subrmcw§ §subrmcx§
§subrmcy§ §subrmcz§

A.2. Tabulka symbolů

Pro úplnost uvádíme na samotný závěr tabulku všech symbolů. Jednotlivé řádky jsou očíslovány, pro všechny řádky jsou pak k dispozici seznamy příkazů, které jsme uvedli výše.

1 a b c d e f g h i j k l m n o p q r s t u v w x y z
2 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
3 *a b c d e f g h i j k l m n o p q r s t u v w x y z*
4 *A B C D E F G H I J K L M N O P Q R S T U V W X Y Z*
5 **a b c d e f g h i j k l m n o p q r s t u v w x y z**
6 **A B C D E F G H I J K L M N O P Q R S T U V W X Y Z**
7 0 1 2 3 4 5 6 7 8 9 ∅ Ⓢ ¶ § ¥ © $ £ ß ♭ ♮ ♯ ♣ ♢ ♡ ♠ ‡ ♪ ℗ ©
8 α β γ δ ϵ ε ζ η θ ϑ ι κ λ μ ν ξ π ϖ ρ ϱ σ ς τ υ φ ϕ χ ψ ω
9 Γ Δ Θ Λ Ξ Π Σ Υ Φ Ψ Ω
10 ℵ א ב ג ד ⨿ ı ȷ ℓ ℘ ℜ ℑ ∂ ∇ ∑ ∏ ∐ Ⅎ ϰ ℏ ℏ ℧ ⅎ ∂ 𝕜 ℭ ð ϶ ℋ ℒ
11 𝒜 ℬ 𝒞 𝒟 ℰ ℱ 𝒢 ℋ ℐ 𝒥 𝒦 ℒ ℳ 𝒩 𝒪 𝒫 𝒬 ℛ 𝒮 𝒯 𝒰 𝒱 𝒲 𝒳 𝒴 𝒵
12 𝒜 ℬ 𝒞 𝒟 ℰ ℱ 𝒢 ℋ ℐ 𝒥 𝒦 ℒ ℳ 𝒩 𝒪 𝒫 𝒬 ℛ 𝒮 𝒯 𝒰 𝒱 𝒲 𝒳 𝒴 𝒵
13 𝔸 𝔹 ℂ 𝔻 𝔼 𝔽 𝔾 ℍ 𝕀 𝕁 𝕂 𝕃 𝕄 ℕ 𝕆 ℙ ℚ ℝ 𝕊 𝕋 𝕌 𝕍 𝕎 𝕏 𝕐 ℤ
14 𝔞 𝔟 𝔠 𝔡 𝔢 𝔣 𝔤 𝔥 𝔦 𝔧 𝔨 𝔩 𝔪 𝔫 𝔬 𝔭 𝔮 𝔯 𝔰 𝔱 𝔲 𝔳 𝔴 𝔵 𝔶 𝔷
15 𝔄 𝔅 ℭ 𝔇 𝔈 𝔉 𝔊 ℌ ℑ 𝔍 𝔎 𝔏 𝔐 𝔑 𝔒 𝔓 𝔔 ℜ 𝔖 𝔗 𝔘 𝔙 𝔚 𝔛 𝔜 ℨ
16 *a b c d e f g h i j k l m n o p q r s t u v w x y z* 0 1 2 3 4 5 6 7 8 9
17 *a b c d e f g h i j k l m n o p q r s t u v w x y z* 0 1 2 3 4 5 6 7 8 9
18 *A B C D E F G H I J K L M N O P Q R S T U V W X Y Z* ∞ ( ) + −
19 *A B C D E F G H I J K L M N O P Q R S T U V W X Y Z* ∞ ( ) + −
20 α β γ δ ϵ ε ζ η θ ϑ ι κ λ μ ν ξ π ϖ ρ ϱ σ ς τ υ φ ϕ χ ψ ω = < ≤ > ≥
21 α β γ δ ϵ ε ζ η θ ϑ ι κ λ μ ν ξ π ϖ ρ ϱ σ ς τ υ φ ϕ χ ψ ω = < ≤ > ≥
22 《 ⟦ 》 ⟧ ⊬ < ⋈ % – — . ′ @ ◊ ; ✓ □ , + = : ⊄ ‘ ’ “ ” #
23 ⋮ … ≉ ✠ ≭ ? ¿ ⋱ > ≮ ⊅ ⋯ × ´ ˜ ¯ ˙ ¨ ˆ ⌒ ˋ ~ ˇ ˘ * ∊ ≁
24 △ ▽ ∨ ∧ ⊕ ⊖ ⊗ ⊘ ⊙ † & ≤ ≺ ⪯ ≪ ⊂ ⊆ ⊑ ∈
25 ⊢ ⌣ ⌢ ≠ ≥ ≻ ⪰ ≫ ⊃ ⊇ ⊒ ∋ ⊣ ∥ ∉ ≡ ∼ ≃
26 ≍ ≈ ≅ ⋈ ∝ ⊨ ≐ ⊥ ⊥ ∞ ∫ ′ √ ⊤ ∀ ∃ ¬ ← ⇐ → ⇒
27 ↔ ⇔ ↑ ↕ ↗ ↙ ↦ ↩ ↼ ⇀ ⇌ ⟵ ⟸ $\xrightarrow{1\text{-}1}$ $\xrightarrow[\text{onto}]{}$
28 ⟶ ⟹ ⟷ ⟺ ⇑ ⇓ ⇕ ↘ ↖ ⟼ ↪ ↽ ⇁ $\xrightarrow[\text{onto}]{1\text{-}1}$
29 ⋁ ⋂ ⋃ ⨆ ⨄ ⨀ ⨂ ⨁ ⋀ ∫ ∮ ( ) ⌊ ⌋ ⌈ ⌉ { } ⟨ ⟩ /
30 ! ⇊ ⦅ ⦆ ↓ ¡ △ ▽ □ ◊ ∠ ∡ ∄ ∖ ∅ ▲ ▼ ■ ◆ ★ ◁ ╱ ╲ ∔ ∖
31 ⋒ ⋓ ⊼ ⊻ ⩞ ⊟ ⊠ ⊡ ⊞ ⊛ ⋉ ⋊ ⋋ ⋌ ⋏ ⋎
32 ⊝ ⊛ ⊚ · ⊤ ≦ ⩽ ⪕ ≲ ⪅ ≊ ⋖ ⋘ ≶ ⋚ ⪋
33 ≑ ≒ ≓ ∽ ⋍ ⫅ ⋐ ⊏ ≼ ⋞ ≾ ⪷ ⊲ ⊴ ⊨ ⊪
34 ⌣ ⌢ ≏ ≎ ∝ ◀ ∴ ≧ ⩾ ⪖ ≳ ⪆ ⋗ ⋙
35 ≷ ⋛ ⪌ ≖ ≗ ≜ ∼ ≈ ⫆ ⋑ ⊐ ≽ ⋟ ≿ ⪸ ⊳ ⊵
36 ⊩ ∣ ∥ ≬ ⋔ ▶ ∵ ≮ ⪇ ⪉ ≨ ≰ ⪯̸ ⋦ ⋨ ⪹
37 ≯ ⪈ ⪊ ≩ ≱ ⪰̸ ⋧ ⋩ ⪺ ⊀ ⋠ ⪵ ⪶ ≁ ∤ ∤
38 ⊬ ⊮ ⋪ ⋬ ⊈ ⊊ ⊈ ⫋ ≆ ⊁ ⋡ ⪶ ⋩ ⪺ ≇
39 ∦ ⊭ ⊯ ⋫ ⋭ ⊉ ⊉ ⊋ ⫌ ⊋ ⇇ ⇆ ⇚
40 ↞ ↢ ↫ ⇋ ↶ ↺ ↰ ⇈ ↿ ⇃ ⊸ ↭ ⇉ ⇄ ⇛
41 ↠ ↣ ↬ ↷ ↻ ↱ ↾ ⇂ ⇝ ↚ ⇍ ↮ ↛ ⇏ ⇎
42 ± ∓ ∖ · × ∗ ⋆ ⋄ ∘ • ÷ ∩ ∪ ⊎ ⊓ ⊔ ◃ ▹ ≀ ◯
43 ⇸ ⇻ ⇼ ↣ ⤔ ⤕ ≙ ⊲ ⊳ ∘ • ⦂ ⋿ ⊌ ⦇ ⦈ ⦉ ⦊ ⊕ ⊗ ✸ ✧ ∗
44 ¤ ¢ ¤ ✠ ☺ ☻ ☺ ☼ ¦ ⌀ ✎ ☎ ⌕ ⌚ ‰ ✥ ⇨ ↯ 〰 ♪ ♩ ♩ ♫
45 ≋ ☐ ⊞ ⊟ ⊟ ⇧ ⇩ ⎕ ‾ ⊛ ⋆ ▿ ▿ ∧ ∨ ○ ⊥ ∤ ∤ ♌ ☋ ☉ ● ○ ☾ ☽
46 ☿ ♀ ♂ ♃ ♄ ♅ ♅ ♇ ♆ ☌ ☍ ☊ ♈ ♎ ♉ ♏ ♊ ♐ ♋ ♑ ♒ ♍ ♓ ∗ ∗
47 ✡ ◖ ◐ ◗ ◑ ◒ ◓ ⊡ ⊠ ⋈ ◇ ◯ ⬠ ⬡ ⌂ ⊀ ⌐ ↺ ↻ ∝ ⇝ ∫ ∬ ∭ ∲ ∯ þ Þ ð Ð ɔ ɘ
48 ☽ ♅ ♆ ♀ ☠ ☣ ☢ ✷ ♀ 🛑 🚲 ↯ ◍ ✢ ✺ ❆ ✂ ✄ ☞ ☜ ☞ ☜
49 a b c d e f g h i j k l m n o p q r s t u v w x y z
50 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z ☺

B. Příloha 2: Dokumentace programového rozhraní jádra

Následující příloha obsahuje zevrubný přehled důležitých tříd jádra. Pro bližší popis a kompletní dokumentaci veřejných metod můžete využít HTML dokumentaci, která je na doprovodném CD v adresáři doc/api.

B.1. Balíček cz.upol.jo.core

Jádro celého systému. Obsahuje základní objekty jako je geometrický svět, geometrický objekt a základní geometrické útvary.

B.1.1. Třídy

AbstractBindedPivot – Základní třída pro pivoty, které jsou vázány na jiný geometrický objekt.

AbstractDeclarator – Bázová třída pro deklarátor geomObjektu.

BindedPivotPoint – Privot, který je vázán k bodu.

GeomObject – Geometrický objekt.

Hinter – Třída nabízející seznam konstrukcí (kroků), které je možné vykonat.

Line – GeomObjekt přímka.

Plane – GeomObjekt rovina.

Point – GeomObjekt bod.

World – Objekt zapouzdřující geometrický „svět“ – tedy geometrické objekty a konstrukce.

B.2. Balíček cz.upol.jo.core.alg

Balíček s algoritmy pro algebru.

B.2.1. Rozhraní

Transformable – Objekty, které mají vyjádření pomocí matice a které je možno transformovat.

Transformator – Rozhraní, pro transformování transformovatelných (Abstract-Transformable) objektů.

B.2.2. Třídy

AbstractPivot – Bázová třída pro specifické pivoty.

AbstractTransformable – Bázová třída pro objekty, které je možné transformovat pomocí transformační matice a mají maticové analytické vyjádření.

AbstractTransformator – Bázová třída, obsahuje základní implementaci transformátoru.

BackTransformSolver – Vypočítá hodnoty transformací podle zadané roviny.

BasePivot – Pivot pro základní bázi (1,0,0) (0,1,0) (0,0,1).

Compare – Statická třída pro operace související s porovnáváním čísel, vektorů matic apod.

GroupTransformator – Transformátor spojující několik jiných dohormady.

MatrixTransformator – Třída zajišťující transformaci jednoho objektu.

Pivot – Báze (osy otáčení, střed zvětšování apod.) pro všechny transformace.

SelectionTransform – Výběr objektů, které se mají hromadně transformovat.

Tools – Statická třída obsahující různé podpůrné metody.

B.3. Balíček cz.upol.jo.core.anims

B.3.1. Rozhraní

Animated – Rozhraní pro animaci.

B.3.2. Třídy

AnimationInfo – Třída jako struktura pro informace o animačních vláknech (voláních).

AnimationManager – Správce animací.

B.3.3. Výjimky

AnimationStopException – Výjimka, která by měla být vyvolána, pakliže je nutno zastavit proces animování uprostřed animačního cyklu.

B.4. Balíček cz.upol.jo.core.axiom

Systém axiomatických vlastností.

B.4.1. Rozhraní

Matchable<E> – Rozhraní definující objekt, který může být výsledkem při hledání klíčového bodu KeyPointMatch.

B.4.2. Třídy

AbstractMatchable<E> – Abstraktní rozhraní klíčového bodu.

AbstractRelation – Popis relace mezi konstrukcemi (jejich popisy).

AxiomSystem – Systém (databáze) všech relací a odvozovacích pravidel.

Fixing – Omezuje generování matchů.

IdentityRelation – Popis relace mezi popisy konstrukcí.

KeyPointInfo – Popis klíčového bodu.

KeyPointInfoBuilder – Třída vytvářející imutabilní objekty KeyPointInfo.

KeyPointMatch – Klíčový bod (match) vyhovující některému popisu klíčového bodu KeyPointInfo.

KeyPointMatchBuilder – Builder pro třídu KeyPointMatch.

Matcher – Udržuje matche jednoho klíčového bodu.

MatchIterator – Iterátor nad objekty KeyPointMatch.

Path – Cesta ke konstrukci v popisu klíčového bodu.

PathMapping – Mapování mezi dvěma popisy klíčových bodů.

PathMapping.MapPair – Jeden prvek mapování.

PrefsMatchIterator – Prvek dekorátoru.

Prop – Třída reprezentující jednu syntaktickou vlastnost (vztah jedné primitivní konstrukce k jiným).

PropRelation – Popis relace mezi popisy konstrukcí.

RelationSystem – Systém (databáze) všech relací.

Rule – Odvozovací pravidlo.

RuleSystem – Systém všech odvozovacích pravidel Rule.

SingleRequest – Požadavky kladené na jednu konstrukci v popisu klíčového bodu, které musí konstrukce splnit, aby vyhověla.

SingleRequestBuilder – Builder, vytváří instance SingleRequest.

B.4.3. Výčtové typy

RelationNames – Jména zabudovaných relací.

B.5. Balíček cz.upol.jo.core.constr

Balíček realizující geometrické konstrukce.

B.5.1. Rozhraní

ConstrInstance – Instance jedné konstrukce podle popisu konstrukce AbstractConstrInfo.

StepInfo – Představuje popis jednoho kroku konstrukce.

B.5.2. Třídy

AbstractConstrInfo – Základní třída pro popis konstrukce.

AbstractConstrInstance – Bázová implementace ConstrInstance.

AbstractStepInfo – Bázová implementace StepInfo.

ConstrSystem – Systém (databáze) všech popisů konstrukcí.

HintConstr – Ke každé konstrukci v hinteru je jeden HintConstr, který udržuje pro všechny kroky jedné konstrukce objekty HintStep.

HintStep – Ke každému kroku konstrukce je udrožována kolekce objektů HintStepHintStep.

StepSuggestion – Doporučení dalšího kroku konstrukce.

UserConstrInfo – Třída pro popis uživatelské (složené) konstrukce.

UserConstrInstance – Třída pro instanci jedné konstrukce.

UserStepInfo – Popis kroku uživatelské konstrukce.

UserStepInfoBuilder – Builder pro třídu UserStepInfo.

UserStepInstance – Představuje provedení jednoho kroku konstrukce podle popisu StepInfo.

B.6. Balíček cz.upol.jo.core.constr.def

Základní (primitivní) konstrukce.

B.6.1. Třídy

AbstractDefaultConstrInfo – Bázová třída pro popisy konstrukcí, které jsou napevno vytvořeny a mají speciální podporu.

AbstractPrimitiveConstrInfo – Bázová třída pro primitivní konstrukce.

AbstractPrimitiveStepInfo – Bázová třída pro krok primitivní konstrukce.

LineConstrInfo – Konstrukce přímky.

MongConstrInfo – Popis konstrukce Mongeova promítání.

MongLineInfo – Popis konstrukce přímky v Mongeově promítání.

MongPlaneInfo – Popis konstrukce roviny v Mongeově promítání.

MongPointConstrInfo – Popis konstrukce bodu v Mongeově promítání.

NewMongLineInfo – Popis konstrukce přímky v Mongeově promítání.

NewMongPlaneInfo – Popis konstrukce roviny v Mongeově promítání, pro vytváření nových konstrukcí.

PlaneConstrInfo – Popis primitivní konstrukce „rovina v prostoru“.

PointConstrInfo – Popis primitivní konstrukce „bod v prostoru“.

PrimitiveConstrInstance – Bázová třída pro instanci primitivní konstrukce.

StoppersConstrInfo – Popis konstrukce nalezení stopníků přímky.

TestConstructions – Vytvoří popisy testovacích konstrukcí.

B.7. Balíček cz.upol.jo.core.filters

Podpora filtrů.

B.7.1. Rozhraní

FilterSupport – Rozhraní definující podporu filtrů.

B.7.2. Třídy

AbstractCompositeFilter – Základní třída pro složený filtr.

AbstractFilter – Základní třída pro filtr nad konstrukcemi.

AndFilter – Složený filtr reprezentující „a zároveň“.

CreationTimeFilter – Filtr, který propouští pouze konstrukce, které mají čas vytvoření v zadaném intervalu.

FilterManager – Podpůrné operace s filtry.

NameFilter – Filtr, který propouští pouze konstrukce, které mají jméno podle zadaného regulérního výrazu.

NotFilter – Filtr, který propouští pouze konstrukce, které jeho podfiltr nepropustí.

OrFilter – Složený filtr reprezentující „nebo“.

RenderFlagFilter – Filtr, který propouští pouze konstrukce, jejichž GeomObject obsahuje některý z určených příznaků.

TypeFilter – Filtr, který propouští pouze konstrukce podle určeného infa.

B.8. Balíček cz.upol.jo.core.interfaces

Balíček s různými rozhraními pro celé jádro.

B.8.1. Rozhraní

CollectionChangedListener<E> – Listener, slouží k notifikaci o zmněně obsahu NotifyCollection.

GeomObjectFactory – Abstraktní továrna pro geometrické objekty.

GeomObjectRenderer – Rozhraní pro vykreslování geom. objektů.

GraphicsDevice<D> – Grafické zařízení – návrhový vzor Most.

Nominable – Představuje rozhraní pro pojmenovatelný objekt.

Removable – Toto rozhraní definuje, že objekt je odstranitelný.

SenderListener<E> – Naslouchač, který neposílá žádné parametry než sender.

B.8.2. Třídy

AbstractSystem<O extends Nominable> – Představuje bázovou třídu pro singletone systémy obsahující hash tabulku nějakých objektů.

Const – Konstanty.

Description – Slovní popis (konstrukce, kroku konstrukce apod.)

Event<L> – Podpora událostí.

NotifyCollection<E> – Wrapper pro kolekci, který umožňuje registrovat události vyvolané při přidání/odebrání objektu.

NotifyCollectionRO<E> – ReadOnly adapter nad notifyCollection.

B.8.3. Výčtové typy

Const.ConstrNamespaces – Jmenné prostory (kategorie) názvů konstrukcí.

GraphicsDevice.FontType – Typ fontu, který lze měnit v popiskách.

RenderFlags – Zpřesňující vlastnosti geomObjektů.

B.8.4. Výjimky

DuplicateNameException – Výjimka: Požadované jméno není unikátní.

IllegalInfoException – Tato výjimka je vyvolána, pokud se neshoduje popis nějakého objektu s očekávaným.

B.9. Balíček cz.upol.jo.core.visual

Zapouzřuje vizuální stránku geometrických objektů, které vzniknou aplikováním konstrukcí.

B.9.1. Rozhraní

Icon – Vlastnosti ikony.

PlaneParametrization.ParametrizationChangedListener – Naslouchač změn parametrizace.

B.9.2. Třídy

AbstractNode – Uzel na přímce.

AbstractSegment – Popisuje typ segmentu přímky.

AbstractVisuals – Vizuální vlastnosti geomObjektů.

ColorVisual – Deprecated.

ConstantNode – Uzel na přímce, který ma hodnotu parametru pevně danou.

IconManager – Systém všech ikonek.

Label – Popisek geometrického objektu GeomObject.

LineNode – Uzel, jehož parametr je určen průnikem s přímkou.

LineSegmentation – Popisuje segmenty přímky.

LineVisuals – Vizuální vlastnosti specifické pro přímku.

PlaneNode – Uzel, jehož parametr je určen průnikem s rovinou.

PlaneParametrization – Parametrizace geomObjektu.

PlaneVisuals – Vizuální vlastnosti specifické pro rovinu.

PointNode – Uzel, jehož hodnota parametru t je určena bodem, který leží na přímce tohoto uzlu.

PointVisuals – Vizuální vlastnosti specifické pro bod.

RelativeNode – Uzel, jehož parametr je určen relativně vzhledem k jinému uzlu.

RemoveCommand – Příkaz pro odebrání uzlu.

SegmentProperties – Vlastnosti segmentu přímky.

TwoNodeSegment – Segment přímky, který je určen dvěma uzly.

UserIcon – Uživatelská ikona.

B.9.3. Výčtové typy

DefaultIcons – Výčet jmen ikon, které jsou implicitně využívány jádrem.

RelativeNode.DistanceType – Typ vzdálenosti u uzlového bodu.

SegmentProperties.LineStyle – Styl úsečky.