Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE

Roman Ženka

Algoritmy pro nefotorealistické zobrazování

Kabinet software a výuky informatiky

Vedoucí diplomové práce: *RNDr. Josef Pelikán*

Studijní program: *Informatika*

Děkuji RNDr. Josefu Pelikánovi za cenné připomínky a rady při tvorbě této práce. Děkuji své rodině a přátelům, bez jejichž podpory a porozumění by tato práce nemohla v této podobě vzniknout.

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

V Praze dne 9. srpna 2001 Roman Ženka

Obsah

Úvod

Tato diplomová práce je rozdělena do tří základních částí. První část popisuje problematiku nefotorealistického zobrazování a uvádí stručnou kategorizaci používaných algoritmů. Vybrané algoritmy jsou podrobně popsány ve druhé části. Třetí část je věnována návrhu systému **NPRlab**pro experimentování s nefotorealistickými zobrazovači. V této části je také popsána implementace uvedených algoritmů spolu s příklady, jak si je v rámci systému vyzkoušet.

V přílohách je k dispozici implementace systému **NPRlab** spolu s podrobnou uživatelskou a programátorskou dokumentací.

1 Nefotorealistické zobrazování

1.1 Co je nefotorealistické zobrazování

V počítačové grafice se termínem *fotorealistické zobrazování* označují metody, které se snaží vytvořit fotorealistický obraz virtuální trojrozměrné scény. Fotorealismem je míněna míra podobnosti obrazu virtuální scény s fotografií odpovídající scény skutečné. *Nefotorealistické zobrazování* pak zahrnuje metody, které se o fotorealismus nesnaží. Těmto metodám nejde o to, přesvědčit pozorovatele, že se dívá na fotografii.
Poznámka. V anglické literatuře se pro tyto dva přístupy používají zkratky PR (*Photorealistic Rendering*) a NPR (*Non-Photorealistic Rendering*). Tyto zkratky budou nadále používány ve zbytku diplomové práce.

1.2 Proč zobrazovat nefotorealisticky

Metody PR[1] jsou již dnes schopné zobrazovat velice komplikované a detailní modely scén v kvalitě takřka nerozeznatelné od fotografie. Pokud budeme brát míru fotorealismu jako jediné kritérium kvality zobrazení, nezbude pro metody NPR místo. Často však není cílem získat co nejvěrnější zobrazení. Fotorealismus se pak stává nepotřebným, či dokonce nežádoucím. Uveďme si několik příkladů takových situací.

Vyšší přehlednost zobrazení. Realistická scéna může být nepřehledná. Při zobrazování je nutné učinit určité zjednodušení, aby byl výsledek snáze pochopitelný. Např. návod na údržbu složitého systému potrubí nemá smysl ilustrovat fotografiemi, protože skutečnou scénu může každý vidět. Mnohem více pomůže barevné odlišení jednotlivých funkčních celků, odstranění vržených stínů a zvýraznění siluet.

[1] Metody fotorealistického zobrazování, tzn. metody produkující fotorealistické obrázky.

Zdůraznění vlastností scény. Další důvod pro použití NPR může být potřeba zdůraznit určité vlastnosti scény. Např. pomocí šrafování lze zdůraznit zakřivení povrchů, které by při běžném stínování nemuselo být dostatečně patrné. Pro zobrazování lesklých částí v technických ilustracích byl navržen speciální osvětlovací model [8], který lépe ilustruje zakřivení ploch.

Vizualizace doplňujících údajů. Scéna může obsahovat kromě informací o vzhledu objektů i další údaje, které je třeba zobrazit. NPR umožňuje vizualizovat tyto údaje mnoha různými způsoby - změnou šrafování, druhu čar, osvětlovacího modelu a podobně.

Přizpůsobení se stylu. NPR je možné s výhodou použít pro tvorbu animovaných filmů. Na počítači lze vytvořit složité scény, které se pak kombinují s ručně kreslenými postavami a objekty. Protože by realisticky zobrazené prostředí vedle kreslených částí působilo velice rušivě, počítačová grafika se musí přizpůsobit co nejvíce stylu používanému animátory.

Nedostatečné zadání scény. Nemusí být dostupná dostatečně detailní data popisující scénu. V tomto případě by fotorealistické zobrazení částečného modelu scény mohlo působit nepřirozeně a rušivě. Fotorealistické zobrazení dává určitý dojem definitivnosti, který nemusí být žádoucí vyvolat. To je případ scén, které jsou teprve vytvářeny a navrhovány. Hrubou skicu návrhu nového automobilu je spíše možné prezentovat, než fotografii stejně hrubého modelu.

Jiným příkladem může být zobrazování materiálů, jako jsou travnaté povrchy nebo srst zvířat. Modelovat a následně vykreslovat každé stéblo trávy je neúnosné. Zobrazit trávník tak, aby bylo pozorovateli jasné, že se jedná o trávník, je mnohem snazší - např. stačí do plochy trávníku umístit několik různě zakřivených čar. Výsledek sice není fotorealistický, ale svůj účel plní.

Média nedovolující fotorealistické zobrazení. NPR přístup může být vynucen přímo médiem. Mnohé knihy jsou omezeny na používání černobílých ilustrací z ekonomických důvodů. Lépe než fotografie zobrazené hrubým polotónováním vypadají ručně tečkované nebo šrafované obrázky.

Oblíbený formát SWF pro ukládání interaktivní webové grafiky klade jiný druh omezení. Ačkoliv tento formát podporuje rastrovou grafiku, je mnohem výhodnější používat grafiku vektorovou, zejména kvůli rychlosti přenosu dat. NPR zobrazovač schopný generovat atraktivní vektorové scény může být velice užitečným pomocníkem.

Atraktivnost NPR. Někdy je nefotorealistický styl zobrazení považován za atraktivnější, než možné realistické zobrazení. Ve světě výtvarného umění

je realismus jen jedním z velkého množství stylů, ve kterých výtvarníci tvoří svá díla. NPR je schopné do jisté míry napodobit práci výtvarníka a zvýšit tak estetický dojem z generované scény. Dalším — poněkud všednějším — příkladem mohou být různé reklamní materiály, které mají přitáhnout pozornost. Dobře provedené NPR zobrazení často dokáže zaujmout více než fotografie, které jsou k vidění všude.

1.3 Použití NPR v praxi

Protože je NPR věcí relativně novou a tedy nepříliš známou a neověřenou, je možné pochybovat jeho použitelnosti v praxi. Proto bych rád zmínil článek [21], který snad pomůže tyto pochyby vyvrátit. Mezi německými architekty byl proveden zajímavý průzkum. Architektům byl předložen návrh domu zobrazený postupně jako a) drátěný model s odstraněním neviditelných hran, b) s vystínovanými plochami bez zvýraznění hran a c) jako počítačem generovaná hrubá skica. Architekti měli určit, jak na ně tato zobrazení působí. Různé druhy působení byly rozděleny podle klasifikačního schématu uvedeného v [24] do těchto skupin:

- *rozpoznávání* — snadnost porozumění obrázku, jeho přehlednost, prostorový dojem, ...
- *působivost* — emocionální působení obrázku, jeho zajímavost
- *motivace* — jak moc vybízí obrázek pozorovatele, aby se účastnil navrhování jeho úprav.

Autoři článku zjistili, že ačkoliv stínované a drátové obrázky dávají lepší prostorový dojem a jsou snáze srozumitelné, NPR zobrazení je preferováno pro prezentaci hrubých návrhů, protože vybízí pozorovatele pro zapojení se do tvůrčího procesu a dává mu pro něj větší prostor. Emocionální působení NPR obrázku bylo také lepší. Výsledek průzkumu ukázal, že většina architektů by dala přednost NPR v průběhu tvorby návrhu a také by NPR zobrazení použila při komunikaci se svými klienty.

1.4 Kategorizace metod NPR

Protože cíl NPR není jednoznačně dán předem jako u PR (vytvořit obraz nerozeznatelný od fotografie), vznikla celá řada metod, které se od sebe často diametrálně liší. NPR lze dělit do kategorií podle různých, vzájemně ortogonálních hledisek. Dělení uvedené v této kapitole je neformální, určené jen pro interní potřebu této diplomové práce. Jiné kategorizace jsou k dispozici v pracích [13, 20].

1.4.1 Míra interakce s uživatelem

Kromě plně automatických metod, které jsou schopné zobrazit vstupní data bez interakce s uživatelem, existuje mnoho technik, umožňujících uživateli zasahovat do procesu zobrazování. Míra interakce bývá různá - od měnění parametrů zobrazení až po přímé vytváření scény. Jako příklad uveďme program Teddy [10], kde uživatel kreslením čar na obrazovce vytváří jednoduché objekty, které se okamžitě zobrazují nefotorealisticky. Protože kreslení rukou je nepřesné, hrubé zobrazení, které program nabízí, působí velice přirozeně. Příkladem přímého vytváření 3D scén mohou být systémy [6, 23, 26]. Program [22] umožňuje uživateli interaktivně zvolit oblasti, u kterých určí, jak detailně se mají zobrazit.

1.4.2 Druh vstupních dat

Algortimy NPR mohou dostávat řadu různých druhů vstupních dat. Protože není potřeba zobrazovat scény fotorealisticky, mohou být tato data i velmi omezená a neúplná ve srovanání s popisem scén pro PR. Uveďme si typické druhy vstupních dat:

1. *3D data* — trojrozměrný popis geometrie scény, případně doplněný popisem osvětlení, materiálů objektů, ...
2. *2,5D data* — částečná informace o geometrii scény, prezentovaná v 2D podobě.
3. *2D data* — již existující dvourozměrný obraz scény
4. *Akce uživatele*

Algoritmy se vstupy prvního druhu jsou často implementovány ve formě pluginů do již existujícího modelovacího software. Řada algoritmů je navrhována s ohledem na knihovnu OpenGL. Například popis osvětlovacího modelu [8] obsahuje návod, jak tento model snadno v OpenGL implementovat. Přímým přepsáním jádra OpenGL knihovny vznikla NPR varianta hry Quake.

Algoritmy s 2,5D vstupem stojí na rozhraní mezi 3D a 2D. Dostávají sice určitou informaci o trojrozměrném utváření scény, ta je však jen částečná. Příkladem typického vstupu může být mapa hloubek, případně mapa normálových vektorů ploch. Algoritmus tato data dostává ve formě dvourozměrného pole hodnot.

Většina algoritmů s 2D vstupem jsou různé filtry, které jsou schopné simulovat malířské techniky jako je například olejomalba, pointilismus a podobně. Nevýhodou tohoto přístupu je, že algoritmus ztrácí informaci o trojrozměrném utváření scény. Ta však může být algoritmu poskytnuta dodatečně, například rozčleněním obrazu na vrstvy o různé hloubce [2, 7].

Za vstupní data je možné vzít i akce uživatele. Systém [11] snímá pohyb malířských nástrojů (štětce, vědro na vylévání barvy) přímo ve 3D a dovoluje tak uživateli kreslit prostorové obrazy. Systém [3] navíc simuluje hmatovou odezvu při kreslení virtuálním 3D štětcem. Již zmíněný Teddy [10] vystačí s myší. Jiné systémy přímo snímají obraz uživatele pomocí kamery a reagují na jeho mimiku [5], případně na jeho pohyby [9]. Obraz poskytovaný kamerou je pak v reálném čase měněn na nefotorealistický a zobrazován.

1.4.3 Bitmapy versus vektory

Kromě klasických bitmapových dat jsou mnohé NPR metody schopny generovat data vektorová. Vektorový výstup lze sice kdykoliv převést na bitmapu, metody generující vektorová data jsou však odlišné od metod klasických. Hlavní rozdíl spočívá v nemožnosti stanovit jemnost výpočtu s přesností na pixely, případně části pixelu — vektorové metody musí být schopné generovat obraz použitelný v libovolném rozlišení. Tento požadavek lze zeslabit, pokud uživatel zadá míru detailnosti, která ho zajímá. Příkladem tohoto přístupu může být algoritmus [17], který odstraňuje čáry, které jsou příliš blízko u sebe, takže by se při rasterizaci překrývaly.

1.4.4 Statické obrazy versus animace

Metody schopné generovat animace jsou oproti metodám generujícím statický obraz v nevýhodě — musí zachovávat časovou koherenci obrazu. To je obtížné zvláště v případě, že je při generování obrazu používána náhoda, což je typické pro metody snažící se napodobit práci člověka. Hrbolaté siluety, náhodně umístěné šrafy a tečky jsou hlavním zdrojem problémů. Některé statické metody lze použít pro generování animací, pokud se upraví generátory náhodných čísel tak, aby produkovaly výsledky zachovávající prostorovou a časovou koherenci se scénou. Jedním z řešení je vytváření speciálních textur obsahujících náhodná čísla. Tyto textury se před vlastním generováním animace aplikují na objekty scény. Algoritmy zobrazující čáry nebo případné šrafy pak používají náhodná čísla přímo ze zobrazované části scény.

1.5 Hlavní oblasti výzkumu NPR

Současné práce o NPR je možné rozdělit do několika oblastí, na které se výzkum soustřeďuje. Každá z těchto oblastí má svá specifika a řeší odlišný okruh problémů. Následující výčet zahrnuje základní, často diskutované oblasti.

1.5.1 Simulace tradičních malířských technik

Pro tuto oblast je typická snaha vytvořit fyzikální model činnosti určitého malířského nástroje a co nejvěrněji jej napodobit. Kromě modelování vlastní

činnosti nástroje je třeba vzít v úvahu i způsob jeho použití. Například při modelování malby vodovými barvami [7] byl napodoben postup malíře, který nejprve nanáší barvu na papír a poté dodatečně přidává vodu nebo další barvu na místa, která je třeba zesvětlit, případně ztmavit. Rozpíjením barvy je pak získán vzhled, kterého by nebylo možné dosáhnout přímým vytvořením malby bez korekcí. Tento algoritmus v sobě obsahuje zpětnou vazbu, která proces úprav simuluje.

Kromě modelování vlastních nástrojů je potřeba vzít do úvahy i médium, na které se maluje - existují práce podrobně popisující difúzi kapalin na papíře, případně strukturu malířského plátna.

1.5.2 Simulace kreseb a rytin

Při této simulaci je výstupem síť relativně tenkých čar, na rozdíl od vyplněných ploch získaných malířskými technikami, viz 1.5.1. Čáry je možné uložit do některého z vektorových formátů, případně je zobrazit do bitmapy pomocí simulovaných nástrojů.

Vhodné nástroje používané v této skupině zejména ty, které zanechávají úzkou stopu - jako je tužka, uhel, pero, případně rydlo. Protože tímto druhem nástrojů typicky nelze vyplňovat plochy odstíny barev, je řešen problém šrafování, případně tečkování ploch.

Dalším problémem je detekce a zobrazení siluet, které jsou v kresbách velice důležitým prvkem. Tento problém je frekventovaný a proto je v literatuře velice silně zastoupen, viz 1.5.5.

Často jsou používány metody pro zjednodušení scény v místech, kde se vyskytuje příliš mnoho čar. Zde se využívá koherence směru skupiny čar pro nahrazení této skupiny čárou jedinou. Pro scény obsahující složité objekty, jako je srst, tráva nebo koruny stromů, byla vyvinuta technika graftálních textur [12, 15], které jsou automaticky schopné se zjednodušovat v závislosti na měřítku.

Specifickým problémem je tvorba animací kreseb. Pokud jsou totiž šrafovací čáry umisťovány náhodně, při animování scény vzniká rušivý šum těchto čar. Je tedy nutné zaručit časovou koherenci umístění čar. Jednou možností je držet čáry na místě relativně k obrazu. Tak je sice zaručena určitá koherence, ale animace vytvořené tímto způsobem trpí tzv. *shower-door* efektem — animace působí dojmem, že se odehrává za nepohyblivým reliéfním sklem. Lepší dojem zanechávají techniky, které se snaží držet čáry na místě relativně k objektům scény, které zobrazují. Tak je možné vytvořit například animaci rotující koule, na jejímž povrchu se budou čáry skutečně pohybovat, jako by byly na této kouli nakreslené.

1.5.3 Názorné zobrazování

Metody pro názornější zobrazování scén se snaží využít NPR pro zdůraznění určitých vlastností scény, které by jinak nebyly dostatečně patrné. Své místo zde mají metody pro zobrazování technických výkresů [8] nebo metody pro vizualizaci objemových dat. Zajímavé je také zobrazení pohybu na statickém obrázku. Zde se používají tzv. *motion lines* — čáry naznačující směr a rychlost pohybu objektu.

1.5.4 Urychlení zobrazování scén

NPR techniky schopné zobrazit složité scény pomocí několika důležitých čar mohou výrazně urychlit zobrazovací proces. Proto je možné je použít tam, kde je požadována rychlá manipulace s hrubými náhledy na scénu, případně v zařízeních, které nemají dostatečný výkon pro zobrazování fotorealistických scén. Například algoritmus [14] je schopen díky probabilistickému přístupu zobrazit drátový model scény v čase $O(\sqrt{n})$, kde n je počet hran.

1.5.5 Hledání siluet

Jak již bylo zmíněno, hledání siluet objektů je potřeba ve velké řadě NPR metod. Siluety objektů dávají pozorovateli mnoho informací o scéně, mohou obraz zpřehlednit, je možné je použít pro zvýraznění určitých částí scény, pro dodání hloubky obrazu a podobně, proto jsou velice často zobrazovány. Někdy stačí samotné siluety dát dostatek informace, takže další čáry již nejsou nutné.

Algoritmy pro zobrazování siluet je možné dělit do dvou skupin podle přesnosti, se kterou jsou siluety určeny. Tzv. *object precision* algoritmy [4, 14, 17] hledají siluety přímo v rámci 3D popisu scény. Siluety, které jsou tímto způsobem získány, korespondují s hranami původní scény a jsou tedy určeny velice přesně. Zároveň je známa topologie těchto siluet, kterou je možné využít pro další zpracování.

Na druhé straně existují tzv. *image precision* algoritmy, kterým stačí získat obraz siluet s přesností závislou na rozlišení výstupního obrazu. Například algoritmus [19] rozdělí scénu na přivrácené a odvrácené plochy. Odvrácené plochy jsou pak poněkud zvětšeny, takže při zobrazení takovéto scény přečnívají a vytvářejí tak obraz siluet. Jinou možností je použít klasický z-buffer a pomocí filtrů pro detekci hran najít místa, kde dochází k prudké změně hloubky — tam se pravděpodobně nacházejí siluety. Podobnou informaci může dát pole předpočítaných normálových vektorů ploch pro jednotlivé pixely obrazu. Siluety jsou v místech, kde jsou tyto normálové vektory kolmé na vektor pohledu.

1.5.6 Netradiční projekce

Perspektivní projekce je pro fotorealistické zobrazovače víceméně jedinou použitelnou volbou, protože odpovídá funkci fotoaparátu, případně lidského oka. Výtvarníci se na druhé straně na perspektivní zobrazení neomezují — buď z neznalosti, nebo záměrně pro dosažení určitého uměleckého záměru. Staré dřevoryty, které byly vytvořeny před objevením zákonitostí perspektivy, používají většinou tzv. *multiperspektivní projekce* — tedy projekce, která různé části scény zobrazuje v různých perspektivách, případně v axonometrii (viz obr. 1). I v současné době je multiperspektivní zobrazení používáno, pokud chce umělec zdůraznit dojem hloubky vyvolávaný jednotlivými objekty scény, nebo pokud se snaží vyvolat v divákovi určitý pocit. Algoritmus [1] dokáže zobrazovat scény tímto specifickým způsobem i v animaci.

Obr. 1: Multiperspektivní dřevoryt

Jinou možností, jak vytvořit multiperspektivní obraz, je změna geometrie objektů scény v závislosti na úhlu pohledu. Tento přístup je ale možné dále zobecnit a dovolit libovolně složité změny geometrie, viz práce [16, 18]. Tím je umožněno automatické zobrazování například klasických animovaných postaviček, jejichž tvar se mění se změnou úhlu pohledu — důvodem je vyšší výraznost postaviček a jejich lepší působení na diváka.

Speciální druh multiperspektivní projekce je používán při generování panorámat [25], používaných zejména v animovaných filmech. Tato panoráma byla používána pro tvorbu záběrů, ve kterých se kamera pohybuje statickou trojrozměrnou scénou. Panoráma lokálně zobrazuje obraz pomocí

téměř korektní perspektivy, tato perspektiva se však v rámci panorámatu plynule mění. Při lineárním posouvání výřezu po panorámatu tak vzniká dojem pohybu trojrozměrnou scénou.

2 Algoritmy pro nefotorealistické zobrazování

3 Systém NPRlab

Aby bylo možné NPR algoritmy snadno implementovat, bylo potřeba zvolit nějaký nástroj, který by jejich tvorbu usnadnil. Protože jsou NPR metody zatím nepříliš rozšířené, nebylo možné najít nástroj specializovaný na tuto problematiku. Proto byl tento nástroj vytvořen a pojmenován **NPRlab**. Jak jeho název napovídá, jedná se o laboratoř pro experimentování s NPR metodami.

3.1 Požadavky na systém

Obecnost. Tento nástroj měl být dostatečně obecný, aby bylo možné implementovat i algoritmy, které příliš nezapadají do klasického schématu fotorealistických zobrazovačů. Protože jsou NPR přístupy velice různorodé, bylo obtížné najít netriviálního společného jmenovatele pro všechny metody. Zároveň nebylo možné vytvořit naprosto všezahrnující systém v rozumném čase. Proto byl požadavek na naprostou obecnost omezen. Z jednotlivých kategorií NPR byly vybrány některé, na které byl kladen důraz:

1. metody, mající na vstupu hotová 3D data. Důvodem byla obtížnost zobecnit interaktivní metody, které používají velice odlišné způsoby reprezentace scény. Metody s 2D vstupem jsou na tom obdobně.

2. metody, produkující statický obraz. Animace lze získat uložením informací zajišťujících časovou koherenci do scény a opakovaným spouštění algoritmu na data reprezentující jednotlivé rámečky animace.

3. metody pro generování černobílých kreseb (viz kapitola 1.5.2). Tato oblast byla vybrána při zadání diplomové práce, jako hlavní cíl výzkumu.

Modularita. Algoritmy NPR lze rozdělit na části, řešící jednotlivé dílčí problémy. Většina algoritmů není v literatuře řešena kompletně — od vlastního vstupu scény až po výstup — pomocné problémy při jejich implementaci jsou řešené v jiných článcích a typicky existuje možnost volby, jak tyto pomocné problémy řešit. Požadavkem tedy bylo přizpůsobit se tomuto způsobu

tvorby algoritmů a umožnit uživateli kombinovat různé, již hotové metody do funkčních celků pro testování.

Přehlednost a srozumitelnost. Protože se jedná o nástroj určený k experimentování, byla preferována přehlednost a srozumitelnost před vysoce výkonnou implementací. Přehlednost napomáhá rychlejší implementaci algoritmů, které po odladění a otestování mohou být jako návrh předány dále k dodatečné optimalizaci.

Otevřenost. Systém by měl být rozšiřitelný, aniž by bylo nutné do něj zasahovat podstatným způsobem. Ideální by byla rošiřitelnost za běhu, která by umožnila vyvíjet, testovat a opravovat jednotlivé komponenty bez nutnosti systém restartovat. Testování NPR algoritmů by tím bylo podstatně zrychleno.

Snadná manipulace se vstupními daty. Systém by měl mít možnost získávat vstupní data v přehledném, snadno upravitelném vstupním formátu. Ideálním by byl textový formát, který se dá snadno zpracovávat a vytvářet buď ručně, nebo pomocí jednoduchého software. Textový formát je také snadno přenositelný mezi jednotlivými platformami.

Přehledná databáze objektů. Protože každý algoritmus potřebuje ke své činnosti specifická data, systém by měl v sobě obsahovat databázi, obsahující mechanismy pro snadnou extrakci dat. Protože data pro NPR zobrazovače bývají často velice komplikovaná, tato databáze by je měla být schopna ukládat přímo ve formě objektů. Dobrá by byla možnost kromě jednoduchých tabulek ukládat i složitější, strukturovaná data.

Rozšiřitelné objekty. Protože algoritmy typicky pracují tak, že vstupní data rozšíří o další, vypočítané údaje, bylo potřeba navrhnout systém, který by objektům umožňoval růst. Příkladem může být například zobrazování trojúhelníku. Původní reprezentace trojúhelníku zná jenom souřadnice svých vrcholů. Algoritmus pro výpočet normálových vektorů tento trojúhelník obohatí o normálový vektor. Algoritmus pro stínování z normálového vektoru spočítá barvu a trojúhelníku ji přiřadí. Algoritmus pro projekci přepočítá souřadnice vrcholů trojúhelníku na dvourozměrné. Získáváme tak trojúhelník, který zná svou pozici ve 3D prostoru, zná místo, kam se bude vykreslovat a barvu, kterou se bude vykreslovat. Takto upravený trojúhelník je pak dalším algoritmem možné nakreslit na správné místo na obrazovce čarou správné barvy, jejíž tloušťka se může měnit podle informací o 3D umístění trojúhelníku.

3.2 Neřešené problémy

Kromě výčtu věcí, které by měl systém umět, byly od začátku stanoveny problémy o jejichž řešení se tento systém snažit nebude — buď z praktických důvodů, nebo proto, že již byly vyřešeny jinde a lépe.

Získávání vstupních dat. Nebylo možné ani rozumné vytvářet další nástroj pro tvorbu 3D scén. Těchto nástrojů již existuje velké množství a další stále vznikají. Vstupní data pro algoritmy by mělo být možné vytvořit v jednom z těchto nástrojů a pak je následně upravit a doplnit o případná další data pro NPR zobrazování.

S touto problematikou souvisí konvertory vstupních formátů. Systém by měl podporovat dostatečně obecný formát pro zadávání vstupních dat, aby bylo možné většinu současných 3D formátů do tohoto formátu konvertovat. Protože těchto formátů je velká řada, nebylo možné vyvinout pro ně jednotlivé konvertory. Konverze je tak ponechána na uživateli. Protože má systém umožňovat vstup dat v textovém formátu, ve kterém je na internetu k dispozici celá řada testovacích modelů, uživateli může stačit libovolný textový editor pro případnou ruční konverzi dat.

Výstupní data. Podobně jako u vstupních dat, nebyla řešena problematika ukládání dat do mnoha různých formátů. Systém by měl umět data uložit buď do jednoduchého vektorového formátu (SVG), nebo do bitmapy (GIF, JPEG). Další konverze jsou ponechány na uživateli.

Výkon systému. Systém by se neměl zaměřovat primárně na rychlost a paměťovou nenáročnost. Tyto optimalizace mohou systém výrazně znepřehlednit a ztížit jeho další rozšiřování. Cílem bylo vytvořit přehledný systém pro experimentování, nikoliv nástroj pro tvorbu komerčního software, u kterého je rychlost žádoucí.

Animace. Problematika tvorby animací je velice zajímavá, ale také velice náročná, co se týče objemu dat i výpočetního výkonu. Proto bylo rozhodnuto ji zatím do systému neimplementovat a ponechat ji budoucímu výzkumu.

3.3 Implementace systému

Po zvážení požadavků byl proveden návrh systému a následně byl systém implementován. Při navrhování systému bylo potřeba učinit několik významných rozhodnutí, které ovlivnily další vývoj.

3.3.1 Programovací jazyk

Pro implementaci byl zvolen jazyk Java[2]. Použití tohoto jazyka sice snížilo výkon celého systému, na druhé straně však mělo mnoho velkých výhod:

- *Přehlednost.* Jazyk Java je relativně jednoduchý a přehledný, proto je stále více používán při vytváření jednoduchých a přehledných aplikací pro výuku. Kromě své jednoduchosti tento jazyk umožňuje díky podpoře rozhraní[3] vytvoření velice přehledného objektového návrhu systému. Rozhraní umožnila obejít se bez násobné dědičnosti, která bývá problematická, jak ví většina programátorů C++.

- *Přenositelnost.* Programy vytvořené v Javě mohou být snadno přeneseny na řadu platforem, což rozšiřuje množinu potenciálních dalších vývojářů tohoto systému. Kromě toho je jazyk Java jako jeden z mála k dispozici velkému množství uživatelů díky jeho integraci do webových prohlížečů.

- *Rozšiřitelnost za běhu.* Java díky svému návrhu umožňuje nahrávání nových tříd přímo za běhu. Třídy je dokonce možné nahrávat po síti. To dává uživateli systému **NPRlab** velké možnosti — může snadno spolupracovat s ostatními uživateli a vytvářet své vlastní algoritmy, aniž by musel běžící systém restartovat. Díky podpoře reflexe[4] může systém dynamicky zjišťovat informace o vkládaných algoritmech. Díky tomu je možné tyto vkládané algoritmy korektně zobrazovat a spouštět.

3.3.2 Formát vstupních dat

Jako formát vstupních dat byl zvolen jazyk XML. Tento jazyk byl zvolen z několika důvodů:

- *Univerzálnost.* Formát XML umožňuje ukládat libovolná data, která jsou organizována ve stromové struktuře. To se ukázalo být pro potřeby systému **NPRlab** dostačující. Pokud by se stromová struktura ukázala nedostatečnou, je možné použít rozšíření jazyka XML, jako jsou specifikace XLink a XPointer.

- *Snadná editace.* Formát XML je velice snadno editovatelný pomocí libovolného textového editoru. Kromě toho existuje velké množství specializovaných editorů, které poskytují kromě přehledné editace i nástroje pro automatickou kontrolu XML dokumentů.

[2]Konkrétně Java™ 2 Platform, verze 1.3.1. V době započetí práce na systému to byla nejnovější veřejně dostupná verze tohoto jazyka.

[3]V práci je použito české slovo *rozhraní* místo anglického *interface*.

[4]`java.lang.reflect` — balík objektů, umožňující javovým programům za běhu zjišťovat podrobné informace o jednotlivých třídách.

- *Přenositelnost.* Podobně jako jazyk Java i formát XML umožňuje snadnou přenositelnost dat mezi platformami. V kombinaci s Javou tak XML tvoří mocnou dvojici nástrojů, které jsou schopné bezproblémově přenést kód i data na velké množství platforem.

- *Snadná implementace.* Podpora pro načítání a zápis XML souborů je v jazyce Java snadno k dispozici. Novější distribuce Javy dokonce mají tuto podporu v sobě již předem zabudovánu. Podobně jsou na tom i jiné jazyky, což umožňuje kombinovat systém **NPRlab** s nástroji vytvořenými v jiných prostředích.

- *Možnost simulace XML při načítání jiných formátů.* Knihovny pro zpracování XML bývají navrženy tak, že je místo standardních objektů pro načítání XML použit objekt vlastní. Ten může například provádět přímou konverzi z jiných formátů, aniž by zbytek knihovny (a programy ji používající) zaznamenal změnu. Díky tomu je možné do budoucna celý systém snadno rozšířit o načítání dalších formátů.

Kromě uvedených výhod má tento jazyk i své nevýhody. Oproti binárním formátům je paměťově náročnější a jeho zpracování je pomalejší. Při ručním psaní bez pomocných nástrojů může být méně přehledný, než existující textové formáty. Kromě toho ke svému zápisu potřebuje mnoho znaků, které se mohou zdát uživatelům jiných formátů redundantní.

Byla zvažována možnost použití některých existujících textových formátů pro ukládání 3D scén. Tyto formáty však většinou nepočítají s požadavky, které kladou algoritmy NPR jejichž vstupní data mohou být zadaná velice netradičně. Navíc by při převzetí již existujícího formátu bylo nutné implementovat veškeré jeho konstrukce, což bylo při vývoji systému **NPRlab** zbytečnou zátěží.

3.3.3 Vnitřní reprezentace dat

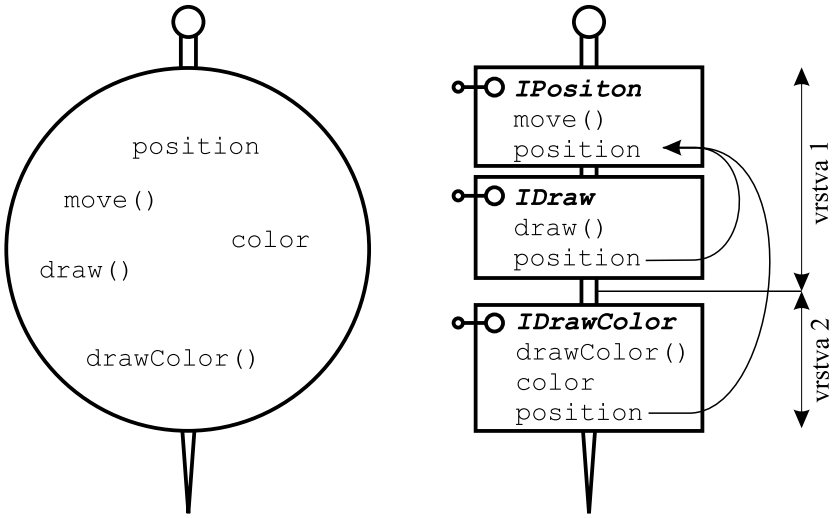
Data jsou v systému reprezentována objekty jazyka Java. Protože bylo potřeba vytvořit rozšiřitelné objekty, které by algoritmy mohly obohacovat o další data a metody, byla vytvořena knihovna pro práci s tzv. meta-objekty[5].

Meta-objekt je objekt, který je možné přímo za běhu aplikace obohacovat o další datové položky a metody (bez újmy na obecnosti budeme dále mluvit jen o metodách). Protože takovéto objekty jazyk Java sám nepodporuje, byl vytvořen systém, který tyto objekty simuluje pomocí seznamu obyčejných objektů. Přidání dalších metod je realizováno přidáním nové třídy do tohoto seznamu. Pokud program potřebuje zavolat určitou metodu meta-objektu, zadá rozhraní, ve kterém je tato metoda definována. Toto rozhraní je vráceno a algoritmus metodu volá již přímo na něm. Ačkoliv by bylo možné

[5]Pro podrobnější informace viz programátorská dokumentace, kapitola B.1.8.

volat metody přímo, pomocí specifikování jména metody a parametrů, výše uvedený přístup je rychlejší a při volání několika souvisejících metod na jednom objektu i praktičtější.

Protože jeden meta-objekt může obsahovat několik objektů stejného typu, systém umožňuje přiřadit každému takovému objektu vrstvu. Dotazy je pak možné upřesnit specifikací této vrstvy. Obrázek 2 ukazuje představu meta-objektu a její skutečnou implementaci.



Obr. 2: Meta-objekt a jeho implementace v systému **NPRlab**.

3.3.4 Struktura databáze

Struktura databáze je na začátku určena vstupním XML souborem. Při načtení XML dat je v paměti je vytvořen tzv. DOM[6] dokument, což je datová struktura typu strom, která popisuje obsah vstupního XML souboru. Jednotlivé uzly tohoto stromu jsou meta-objekty, které na začátku obsahují jedinou podtřídu[7] — reprezentaci XML uzlu. Algoritmy pak touto databází procházejí a přidávají uzlům své podtřídy, případně již existující podtřídy upravují, nebo je extrahují do jiných datových struktur.

3.3.5 Práce s databází

Databáze slouží jako pracovní plocha, na kterou algoritmy pokládají výsledky své činnosti. Databáze je tak neustále obohacována, a to až do okamžiku, kdy

[6]Document Object Model. Byl použit DOM Level 2 poskytovaný balíkem JAXP1.1.

[7]Ve skutečnosti obsahují podtřídy dvě — tou druhou je jednoduchá třída obsahující informaci o pozici uzlu v rámci XML souboru. Tato pomocná třída je používána pro chybová hlášení.

je dat dostatek na vyprodukování výsledku. Ačkoliv je možné části databáze mazat, není to typicky potřebné.

Algoritmy mají několik nástrojů pro práci s databází. Jedním z nich jsou tzv. *akce*[8] - algoritmy, které jsou aplikovány na jednotlivé uzly databáze prohledáváním do šířky, případně do hloubky. Tyto akce je možné vzájemně kombinovat. V systému **NPRlab** jsou akce používány na získávání dat z databáze, přidávání podtříd a provádění modifikací dat.

Speciální druh akcí[9] je použit pro zpracování vstupních XML dat. Tyto akce vždy zpracovávají jeden uzel XML. Akce dostává informace o potomcích tohoto uzlu. Pro každého potomka akce specifikuje další akci, která je schopná pokračovat ve zpracování o úroveň níž. Po zpracování potomka akce dostane zpracovaná data, která může dále použít.

3.3.6 Reprezentace trojrozměrných dat

Současná implementace systému používá homogenní souřadnice pro ukládání trojrozměrných dat. Každý vektor $\vec{a} \in \mathbb{R}^3$ je tak popsán pomocí čtyř čísel v dvojité přesnosti (a_x, a_y, a_z, a_w).

Reprezentace dat pomocí homogenních souřadnic je náročnější, ale má mnoho výhod. Navíc mnoho algoritmů přímo tuto reprezentaci vyžaduje. Protože jsou někdy potřeba čistě trojrozměrná data, objekty reprezentující body podporují metodu `project`, která transformuje souřadnice následujícím způsobem:

$$\texttt{project}(\vec{a}) = \left(\frac{a_x}{a_w}, \frac{a_y}{a_w}, \frac{a_z}{a_w}, 1\right)$$

Pokud je čtvrtá souřadnice nulová, projekci nelze pochopitelně provést. Přesto má takovéto zadání souřadnic smysl - udává vektor bez zadání velikosti, který pouze signalizuje směr v trojrozměrném prostoru.

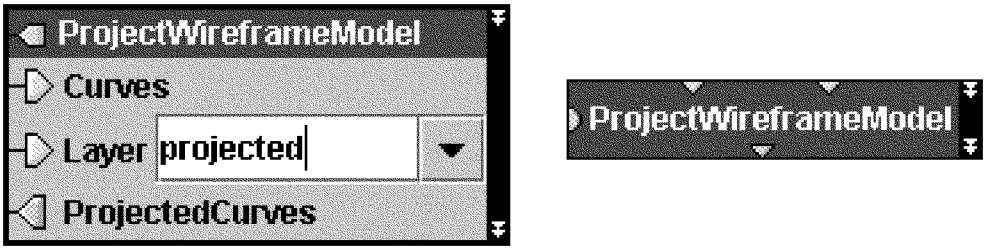
3.4 Způsob implementace algoritmů

Algoritmem v systému **NPRlab** je libovolný objekt, mající konstruktor bez parametrů a implementující rozhraní `npr.alg.IAlgorithm`. Tyto algoritmy umí systém analyzovat a graficky zobrazit, viz obr. 3 Algoritmy mají vstupy a výstupy různých typů. Výstupem, který mají všechny algoritmy, je ukazatel `this`, což umožňuje vytvářet algoritmy, jejichž vstupem je opět algoritmus.

Kromě standardních vstupů, které algoritmus dostává, má algoritmus možnost získat přístup k databázi objektů a pracovat s ní. Takovéto algoritmy (implementující rozhraní `npr.alg.ISceneModifier`) mají dvě speciální položky pro vstup databáze a výstup databáze pozměněné. Propojením těchto

[8] Viz popis balíku `npr.alg.action` v kapitole 3.8.

[9] Viz popis balíku `npr.data.parse` v kapitole B.1.8. Seznam implementovaných akcí tohoto druhu je uveden v kapitole B.3.



Obr. 3: Dvě různé zobrazení algoritmu.

položek je možné určit pořadí vykonávání algoritmů, pokud na něm záleží. Algoritmus následující dostává databázi pozměněnou algoritmem předchozím.

Protože existují algoritmy, které nejsou schopné vyprodukovat výstupní data okamžitě po získání dat vstupních, byla do systému přidána podpora pro dlouhotrvající výpočty. Algoritmy, jejichž výpočet může trvat dlouho, implementují rozhraní `npr.alg.ICalculation`, které umožňuje výpočet spustit ve zvláštním vlákně, což dává uživateli možnost výpočet přerušit.

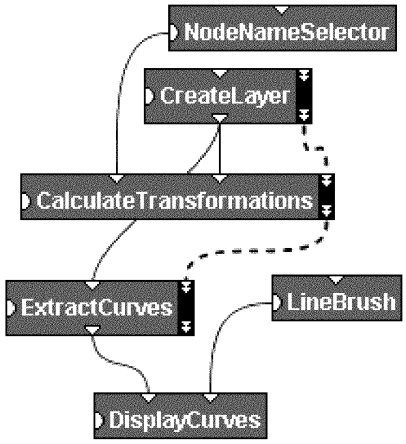
Poslední třídou algoritmů jsou algoritmy interaktivní, které při běhu komunikují pomocí okna s uživatelem. Tyto algoritmy jsou vhodné pro dynamické zadávání parametrů, případně pro zobrazení výsledků činnosti. Pro tyto algoritmy je určen rozhraní `npr.alg.IInteractiveAlgorithm`.

3.5 Kombinování algoritmů

Výstupy jednoho algoritmu je možné napojit na vstupy algoritmu druhého, viz obr. 4. Tímto způsobem je možné algoritmy snadno kombinovat. Systém zaručuje, že algoritmy poběží ve správném pořadí tak, aby vždy měly k dispozici vstupní data z algoritmů předchozích. Při napojování výstupů na vstupy je prováděna typová kontrola. Ta zamezí základním chybám při skládání algoritmů, nemůže však předejít chybám logickým. Algoritmy by proto měly být psány tak, aby dokázaly přijmout i nesprávná, byť typově korektní data.

3.6 Implementované algoritmy

Zde jsou popsány algoritmy, které jsou součástí systému **NPRlab** a jsou tak uživatelům přímo dispozici. Není zde uveden podrobný popis funkce složitých algoritmů, tím se zabývá kapitola 2. Pro popis algoritmů bylo použito následující značení:



Obr. 4: Systém provázaných algoritmů.

 interaktivní algoritmus

 algoritmus pracující s databází

 algoritmus s přerušitelným výpočtem

⇨ vstup algoritmu

⇦ výstup algoritmu

Algoritmy jsou uvedeny v abecedním pořadí podle svého jména. Jednotlivé balíky algoritmů jsou popsány ve zvláštních kapitolách.

3.7 Balík `npr.alg`

Tento balík obsahuje základní obecné algoritmy, které pracují s databází.

3.7.1 `CalculateTransformations`

⇨	`Selector`	selektor objektů požadujících transformaci
⇨	`TransformationsLayer`	vrstva pro ukládání transformací

Projde databázi a všem uzlům, které projeví zájem o znalost své transformace — tedy těm, které vybere zadaný selektor — přidá informaci o jejich transformaci do zadané vrstvy. Uzel obdrží informaci o transformaci, která odpovídá postupnému složení všech transformací od kořene databáze až k tomuto uzlu. Uzel není transformován, pouze je mu přidáno rozhraní `ITransformationProvider`.

3.7.2 `ClipWireframeModel`

⇨	`Curves`	pole křivek pro ořezání
⇨	`Layer`	vrstva, kam uložit ořezané křivky
⇨	`Projection`	perspektivní projekce definující ořez
⇦	`ClippedCurves`	pole ořezaných křivek

Ořeže zadané křivky do části prostoru zobrazované zadanou perspektivní projekcí. Ořezávání je prováděno v homogenních souřadnicích. Ořezané křivky jsou vráceny v poli a zároveň jsou přidány zdrojovým meta-objektům křivek do specifikované vrstvy.

3.7.3 `CreateLayer`

⇨	`LayerName`	jméno vytvářené vrstvy
⇦	`Layer`	vytvořená vrstva

Vytvoří a vrátí novou vrstvu zadaného jména, pokud taková vrstva ještě neexistuje. Existuje-li, je vrácena vrstva existující. Vrstvy pak lze používat při výběru dat z meta-objektů.

3.7.4 `CreateWireframeModel`

⇨	`WireframeLayer`	vrstva, kam přidat vzniklá data

Tento algoritmus je zde zařazen jen pro pohodlí uživatelů. Ve skutečnosti aplikuje několik akcí na uzly databáze, které odpovídají tělesům scény, čímž vytvoří objekty reprezentující drátové modely. Pokud uživatel zavede nová tělesa, může buď tento algoritmus upravit, nebo si jej může ručně poskládat přímo v GUI systému **NPRlab**.

3.7.5 `DisplayCurves`

⇨	`Brush`	štětec, kterým se má malovat
⇨	`Curves`	pole křivek, které se má zobrazit

Zobrazí okno, do kterého pomocí zadaného štětce namaluje zadané křivky. Křivky v tomto poli by již měly být ořezány a projektovány do roviny. Algoritmus je vhodný pro vytváření rychlých náhledů na zpracovaná vektorová data pomocí jednodušších štětců, protože obraz dynamicky překresluje při změně velikosti okna.

3.7.6 `ExtractCurves`

⇨	`CurvesLayer`	vrstva, ze které jsou data extrahována
⇦	`Curves`	pole extrahovaných křivek

Projde databázi a vrátí pole všech nalezených křivek, které tvoří hrany těles ve scéně obsažených. Parametr `layer`, pokud je zadán, omezuje výběr křivek jen ze specifikované vrstvy.

3.7.7 `ProjectWireframeModel`

⇨	`Curves`	pole křivek pro projekci
⇨	`Layer`	vrstva, kam uložit projektované křivky
⇦	`ProjectedCurves`	pole projektovaných křivek

Projektuje zadané křivky do roviny. Tato projekce je provedena převedením vektoru homogenních souřadnic $\vec{a} = (a_x, a_y, a_z, a_w)$ na vektor $\vec{b} = (b_x, b_y)$ pomocí následující funkce:

$$f(\vec{a}) = \left(\frac{a_x}{a_w}, \frac{a_y}{a_w}\right)$$

Algoritmus předpokládá, že dělení bude vždy proveditelné. To by mělo zaručit ořezání křivek před samotnou projekcí, viz algoritmus 3.7.2.

3.7.8 `WireframePreview`

⇨	`Curves`	pole křivek pro zobrazení
⇦	`Projection`	perspektivní projekce zvolená uživatelem

Zobrazí drátový model scény a umožní uživateli zvolit pohled na scénu. Jakmile uživatel okno s modelem uzavře, zvolená projekce je vrácena a je možné ji použít například pro kvalitní zobrazení scény z uživatelem zadaného pohledu.

3.8 Balík `npr.alg.action`

Tento balík obsahuje tzv. *akce*, které je možné aplikovat na jednotlivé položky složených datových struktur, jako je celá databáze systému **NPRlab** pole objektů nebo vektor objektů.

3.8.1 `ActionAddSubclass`

⇨	`SubclassLayer`	vrstva, do které je podtřída přidána
⇨	`SubclassConstructor`	objekt, schopný vytvořit podtřídu

Akce přidá každému objektu na kterém je vyvolána podtřídu do specifikované vrstvy. Podtřídy jsou vytvářeny speciálními objekty, viz 3.9.

3.8.2 `ActionCalculateTopology`

⇨	`GatherTopology`	sbírat vytvořené objekty do vektoru?
⇨	`TopologyLayer`	vrstva, kam se mají vytvořené objekty přidat
⇦	`GatheredTopology`	vektor posbíraných objektů

Akce se pokusí objektu vypočítat topologii a vzniklý objekt přidá do specifikované vrstvy. Je-li příznak `GatherTopology` nastaven, je veškerá vytvářená informace o topologii ukládána kromě databáze i do vektoru, který pak akce vrací. Tento vektor je pochopitelně platný až poté, co je akce aplikována na nějakou datovou strukturu, proto je nutné zajistit čtení dat až po řádném aplikování akce.

3.8.3 `ActionExtractCurves`

⇨	`CurvesLayer`	vrstva, ze které se mají křivky extrahovat
⇦	`ExtractedCurves`	extrahované křivky

Akce extrahuje z objektu křivky, na které se objekt odkazuje a které zároveň leží ve specifikované vrstvě. Křivky jsou ukládány do vektoru, ze kterého mohou být načteny poté, co je akce úspěšně aplikována na některou datovou strukturu. Proto je nutné zajistit čtení dat až po řádném aplikování akce.

3.8.4 `ActionGroup`

⇨	`Action1`	první akce
⇨	`Action2`	druhá akce
⇨	`Action3`	třetí akce

Akce postupně vykoná jednotlivé specifikované akce na každém objektu, na které je akce aplikována. Není nutné specifikovat všechny tři akce. Akce jsou vykonávány postupně ve správném pořadí.

3.8.5 `ActionSelect`

⇨	`Action`	akce, která se má vykonat
⇨	`Selector`	selektor objektů, na kterých se má akce vykonat

Akce vykoná zadanou akci jenom na objektech, které zadaný selektor vybere. Selektory jsou popsány v kapitole B.1.6, seznam implementovaných selektorů je v kapitole 3.12.

3.8.6 `ApplyArray`

- ⇨ `Action` akce pro aplikování
- ⇨ `Array` pole, na které se akce aplikuje
- ⇨ `Ascending` pořadí aplikování

Zadanou akci postupně aplikuje na všechny prvky zadaného pole. Pokud je příznak `Ascending` nastaven, je pole procházeno vzestupně, jinak sestupně.

3.8.7 `ApplyTree`

- ⇨ `Action` akce pro aplikování
- ⇨ `Type` druh průchodu stromem

Aplikuje zadanou akci na uzly stromu databáze. Parametr `Type` určuje, zda se použije procházení do hloubky (DFS), nebo do šířky (BFS).

3.8.8 `ApplyVector`

- ⇨ `Action` akce pro aplikování
- ⇨ `Ascending` pořadí aplikování
- ⇨ `Vector` vektor, na které se akce aplikuje

Zadanou akci postupně aplikuje na všechny prvky zadaného vektoru. Pokud je příznak `Ascending` nastaven, je pole procházeno vzestupně, jinak sestupně.

3.9 Balík `npr.alg.action.subclass`

Tento balík obsahuje tzv. konstruktory podtříd. To jsou objekty schopné zadanému meta-objektu přidat určitou podtřídu, v závislostí na datech tohoto meta-objektu. Konstruktory podtříd jsou typicky volané akcí 3.8.1.

3.9.1 `CubeLinesConstructor`

- ⇨ `OutputLayer` vrstva pro uložení pole čar

Vytvoří pole čar, odpovídajících drátovému modelu krychle. Toto pole čar je uloženo do dané vrstvy jako část meta-objektu s popisem krychle.

3.9.2 `SimpleSubclassConstructor`

- ⇨ `Subclass` typ podtřídy pro vytvoření

Vytvoří podtřídu zadaného typu. Tento typ musí odpovídat třídě jazyka Java, která se může stát podtřídou meta-objektu, viz kapitola B.1.15.

3.9.3 `TransformedObjectConstructor`

⇨ `OutputLayer` vrstva pro uložení transformovaného objektu

V zadaném meta-objektu najde rozhraní transformovatelného objektu[10] a rozhraní poskytující informace o tom, jak tento objekt transformovat[11]. Objekt je následně pomocí transformace získané z druhého rozhraní transformován a vzniklý objekt je vrácen. Tím se například z krychle v základní poloze vytvoří další krychle, umístěná na správné místo ve scéně.

3.10 Balík `npr.alg.image`

Tento balík obsahuje datové struktury a algoritmy pro zpracování dvourozměrných dat.

3.11 Balík `npr.alg.image.brush`

Tento balík sdružuje specializované algoritmy pro simulaci různých druhů štětců.

3.11.1 `LineBrush`

⇨ `Width` tloušťka štětce

Tento štětec kreslí jednoduché čáry s antialiasingem o zvolené tloušťce.

3.11.2 `PixelBrush`

Základní štětec kreslící pixel tlusté čáry bez antialiasingu. Díky své jednoduchosti je to nejrychlejší štětec, proto je ho výhodné použít pro zobrazování rychlých náhledů.

3.12 Balík `npr.alg.select`

Tento balík obsahuje tzv. selektory, což jsou algoritmy, které jsou schopné vybrat podle určitých kritérií některé objekty a jiné zamítnout. Selektor má možnost držet si svůj vnitřní stav a podle něj měnit strategii výběru objektů.

3.12.1 `AndSelector`

⇨ `Selector1` první selektor
⇨ `Selector2` druhý selektor

[10] `npr.data.scene.trans.ITransformable`, viz kapitola B.1.12
[11] `npr.data.scene.trans.ITransformationProvider, popsán tamtéž.`

Vybere jen ty objekty, které vybere zároveň první i druhý selektor. Zamítne-li první selektor výběr, druhý selektor se již nevyhodnocuje.

3.12.2 `NodeNameSelector`

⇨ `NodeNames` jména vybíraných objektů

Vybere jen ty objekty, které jsou meta-objektem vzniklým z XML uzlu zadaného jména. Pokud má být jmen zadáno několik, je nutné je v zápise oddělit znakem středník, například takto: `cube;sphere;mesh`.

3.12.3 `NotSelector`

⇨ `Selector` negovaný selektor

Vybere jen ty objekty, které zadaný selektor nevybere.

3.12.4 `OrSelector`

⇨ `Selector1` první selektor
⇨ `Selector2` druhý selektor

Vybere jen ty objekty, které vybere alespoň jeden ze dvou zadaných selektorů. Potvrdí-li první selektor výběr, druhý selektor se již nevyhodnocuje.

3.12.5 `SubclassSelector`

⇨ `Layer` vrstva objektů
⇨ `Type` třída objektů

Vybere jen ty objekty, které jsou meta-objektem a mají v zadané vrstvě třídu, která je převeditelná na zadanou třídu. Pokud není vrstva zadána, prochází se všechny vrstvy.

3.13 Příklady použití algoritmů

Protože implementovaných algoritmů (byť mnohdy velice jednoduchých) je velké množství, je zde uvedeno několik návodů, jak algoritmy v systému **NPRlab** skládat dohromady do smysluplných celků. Uživatel pak může tyto ukázkové algoritmy měnit a rozšiřovat podle svých potřeb. Všechny uvedené ukázky jsou přímo k dispozici na přiloženém CD.

Závěr

Podařilo se implementovat systém **NPRlab** pro snadný vývoj algoritmů pro nefotorealistické zobrazování. Ačkoliv tento systém není dokonalý a neumožňuje plně pokrýt škálu všech možných přístupů k problematice NPR (což ostatně není ani prakticky uskutečnitelné, kvůli velké různorodosti různých přístupů), ukázal se být užitečným pro implementaci algoritmů tvořících relativně velkou podmnožinu NPR.

Byly implementovány a v praxi vyzkoušeny následující algoritmy:

- *Appelův algoritmus*

Implementace a odladění algoritmů bylo díky systému **NPRlab** relativně jednoduché. Algoritmy byly implementovány s ohledem na modularitu a přehlednost implementace. To umožnilo kombinovat vzniklé algoritmy a otestovat tak možnosti jejich vzájemné kooperace.

Samotný systém **NPRlab** byl psán s ohledem na budoucí možnost dalších rozšíření. Rozšiřování přidáváním nových algoritmů je možné provést přímo za běhu celého systému. Zásahy do samotného systému restart vyžadují. Tyto zásahy jsou pro potenciální další vývojáře zjednodušeny díky přehlednému návrhu systému a bohaté dokumentaci, která je automaticky generována přímo ze zdrojových kódů pomocí programu `javadoc`. Systém je také snadno a přehledně konfigurovatelný, díky uložení veškeré konfigurace ve formátu XML.

Ačkoliv zvolený jazyk Java neumožňuje vytvářet aplikace, které se zaměřují na výkon, pro testovací účely se ukázal být vhodným kvůli své přehlednosti a jednoduchosti. Vytvořené algoritmy jsou snadno přenositelné mezi jednotlivými platformami. Celou aplikaci je možné je nechat pracovat v rámci WWW stránek v podobě appletu, což tuto aplikaci činí velice snadno dostupnou.

Další vývoj

Do systému nebyla pro velkou náročnost implementována podpora pro tvorbu animací. Na animace a techniky produkující časově koherentní obrazy by se měl zaměřit další vývoj a rozšiřování systému **NPRlab**.

Některé jednoduché, ale přesto často používané algoritmy (zejména pro práci s bitmapami) by bylo možná vhodné přepsat do jiného programovacího jazyka kvůli efektivitě. Tím se ovšem ztratí výhoda snadné přenositelnosti celého systému.

Bylo by vhodné přidat podporu více vstupních a výstupních formátů, pro větší komfort uživatele. Systém se nicméně ukázal být použitelným i bez nich.

Zatím neimplementovanou, i když plánovanou možností je vytvoření zjednodušeného NPR appletu, který by byl schopen přímo nahrát vstupní algoritmus a demonstrovat tak NPR techniky. Tento applet by byl velice vhodný pro tvorbu stránek popisujících NPR, kde by umožnil uživatelům vyzkoušet si NPR techniky v praxi. Díky velké modularitě celého systému by tento applet typicky potřeboval jen podmnožinu všech dostupných funkcí, takže by nebylo příliš náročné jej používat i s pomalejším připojením na internet.

A Uživatelská dokumentace

A.1 Požadavky systému

Systém **NPRlab** byl vytvořen v jazyce Java verze 1.3.1. Pro svou činnost potřebuje počítač s instalovanou podporou tohoto jazyka, kterou je možné získat například na stránkách firmy Sun Microsystems. Systém byl testován bez problémů na 466MHz Celeronu se 128MB RAM, je možné jej ale provozovat i na o něco pomalejších strojích. Vlastní program i s daty a pomocnými knihovnami zabere 5MB místa na disku, podpora jazyka Java zabírá typicky kolem 70MB.

A.2 Instalace

A.2.1 Uživatelská instalace

Uživatelská instalace je velice jednoduchá — stačí zkopírovat obsah adresáře `/bin` na pevný disk. Tento adresář obsahuje veškeré potřebné konfigurační a datové soubory a kromě toho obsahuje archív `npr.jar` s veškerým potřebným kódem. Tento archív je možné spustit pomocí příkazu `java -jar npr.jar`, případně při zabudované podpoře spouštění `.jar` souborů přímo příkazem `npr.jar`.

A.2.2 Instalace pro vývoj systému

Pro vývoj celého systému je potřeba zkopírovat zdrojové soubory z adresáře `/src` na disk. Adresář `/src` obsahuje soubor `npr.jpx`, což je soubor s projektem pro uživatele JBuilderu verze 4. Po jeho otevření by se měl celý projekt nahrát.

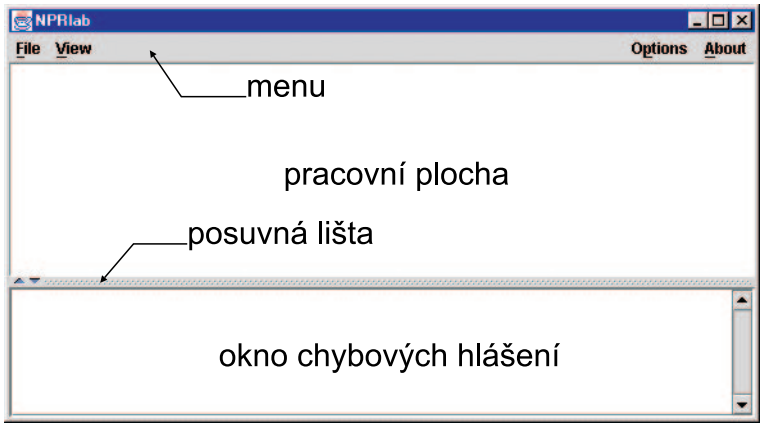
Protože projekt používá některé externí knihovny, je potřeba tyto knihovny nainstalovat a jejich umístění přidat do systémové proměnné `CLASSPATH`. Jedná se o knihovny z adresáře `lib`. Úprava proměnné `CLASSPATH` se na různých systémech provádí různě, doporučujeme nahlédnout do dokumentace jazyka Java na Vašem systému.

A.3 Zprovoznění

Uživatelská instalace by měla být okamžitě k použití. Instalace pro vývoj potřebuje zkompilovat programem `javac`, případně přímo v některém vývojovém prostředí. Pokud by kompilace selhávala, je možné použít předkompilovanou verzi uloženou v adresáři `/classes`. Tento adresář by měl být přidán do systémové proměnné `CLASSPATH`, pak je možné program spustit přímo zavoláním příkazu `java npr.ui.NPR`.

A.4 Práce se systémem

Po spuštění se objeví okno aplikace, které ve standardní konfiguraci vypadá jako na obrázku 5.



Obr. 5: Okno aplikace

Chyby při spouštění se zobrazují v dolní části okna. Obsah tohoto okna je možné vyčistit klepnutím pravým tlačítkem myši do jeho plochy a zvolením položky `Clear` ze zobrazené nabídky.

Vlastní plocha pro práci je umístěna nad lištou, pomocí které je možné měnit její velikost. V této ploše se otevírají jednotlivá okna a zobrazují se zde dialogy.

Celá aplikace je ovládána z menu umístěného v horní části obrazovky. Menu se skládá z několika položek, které sdružují logicky související příkazy. Podoba menu není daná programem, kapitola B.4.4 popisuje způsob, jak změnou konfiguračního souboru změnit vzhled menu.

A.5 Menu `File`

Toto menu sdružuje položky pro otevírání a ukládání souborů. Soubory v systému **NPRlab** jsou dvojího druhu — soubory s definicí scény a soubory popisující algoritmy. Scénu je možné pouze otevřít (`Open Scene`). Po otevření scény jsou zpřístupněny položky pro vytvoření nového algoritmu (`New Algorithm`) nebo otevření již existujícího algoritmu (`Open Algorithm`). Algoritmy jsou svázány se scénou, jejíž okno bylo aktivní v době otevírání algoritmu. Po případné modifikaci algoritmu je algoritmus možné uložit pod jeho starým jménem (`Save Algorithm`), případně se zadáním jména nového (`Save Algorithm As`).

Důležitou položkou tohoto menu je položka `Exit`, jejíž vyvolání ukončí činnost systému.

A.6 Menu `View`

Toto menu ovládá pohled na databázi objektů. Vždy je manipulováno jen s aktuálně zvoleným pohledem na databázi, pokud žádný takový pohled zvolen není, je většina položek nepřístupná.

A.6.1 Položka `New Data View`

Vytvoří nový pohled na databázi. Tuto položku je možné zvolit, je-li vybráno libovolné okno otevřeného dokumentu.

A.6.2 Položka `Refresh Data View`

Během práce s databází se její data mohou změnit. Tato položka menu způsobí obnovení okna databáze tak, aby ukazovalo aktuální stav.

A.6.3 Položka `Reload Data`

Tato položka uvede databázi do stavu, ve kterém byla těsně po nahrání dat. Data vytvořená algoritmy při práci s databází jsou zrušena. Protože se při obnovení dat může zásadním způsobem změnit struktura stromu databáze, je strom uveden do stavu s uzavřenými položkami.

A.6.4 Položka `View Layers`

Meta-objekty mají své jednotlivé třídy ukládány do vrstev různých jmen. Tato položka otevře paletu, která zobrazuje seznam aktuálně používaných vrstev.

A.6.5 Položky `Grouping`

Toto podmenu obsahuje přepínače zapínající seskupování uzlů stejného druhu. Pokud má uzel např. velké množství atributů, může být výhodné tyto atributy seskupit do jednoho uzlu, jehož uzavřením nebo otevřením lze jejich zobrazení nastavovat. Seskupovat lze atributy, třídy meta-objektu i potomky. Po změně seskupování jsou všechny uzly uzavřeny, protože dochází k velké změně uspořádání stromu.

A.6.6 Položka `Show Source Nodes`

Každý meta-objekt XML uzlu má automaticky přidělenu jednu základní třídu, která si pamatuje číslo řádku ve zdrojovém souboru, ze kterého byl uzel načten. Tato položka umožňuje zapnout či vypnout zobrazování těchto tříd.

A.6.7 Položka `Show Text Nodes`

XML uzly mohou mít jako potomky přímo kusy textu. To je případ atributů, u nichž je textová hodnota atributu chápána jako potomek uzlu atributu. Tyto textové hodnoty je možné zvolením této položky nechat zobrazovat. Protože by texty mohly být příliš dlouhé, systém **NPRlab** je automaticky zkracuje.

A.7 Menu `Options`

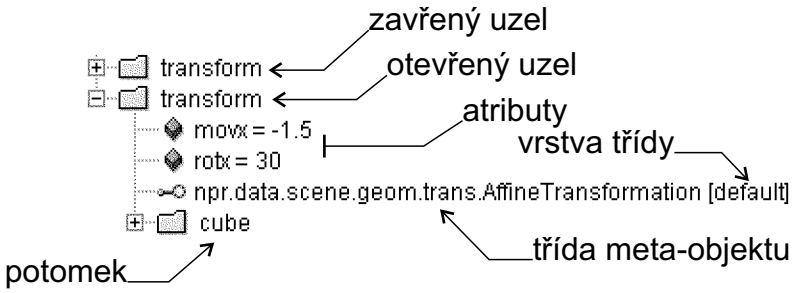
Zde jsou sdruženy nastavení prostředí systému **NPRlab**. Momentálně je z tohoto menu možné nastavit jenom vzhled celého systému, tzv. *look&feel*. Ne všechny zobrazené vzhledy z tohoto menu musí být na konkrétní platformě dostupné.

A.8 Menu `About`

Toto menu obsahuje jedinou položku, po jejímž zvolení je zobrazeno okno se základními informacemi o systému **NPRlab**. Okno je možno zavřít klepnutím na tlačítko `Ok`.

A.9 Okno XML databáze

Okno zobrazuje databázi jako strom jednotlivých XML uzlů. Každý uzel zná kromě svého jména i své atributy, třídy meta-objektu, jehož je částí a své potomky. Tato data jsou zobrazena jako potomci daného uzlu, viz obrázek 6.



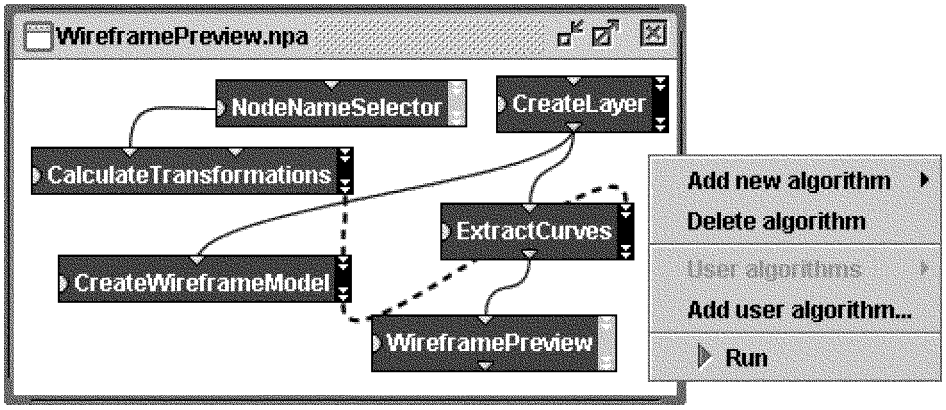
Obr. 6: Zobrazení XML uzlu

Jednotlivé uzly je možné otevírat nebo zavírat levým tlačítkem myši, čímž je možné sbalit celý podstrom do jediného uzlu.

A.10 Okno editoru algoritmů

Okno editoru algoritmů je vytvořeno pomocí položky menu `New Algorithm` nebo `Open Algorithm`. Okno vizuálně zobrazuje aktuální algoritmus. Veškeré

manipulace s oknem se provádí pomocí vyskakovacího menu, které je aktivováno klepnutím pravého tlačítka myši do plochy okna. Obrázek 7 ukazuje okno s otevřeným menu a rozpracovaným algoritmem.



Obr. 7: Okno editoru algoritmů

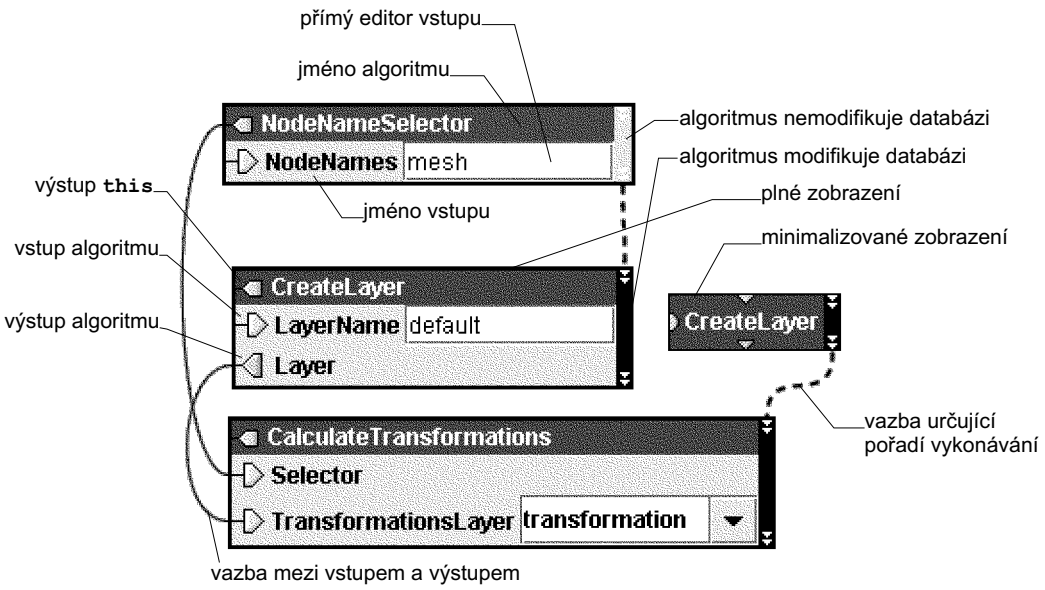
Jednotlivé algoritmy jsou zobrazeny jako obdélníky se jménem. Toto zobrazení je možné přepínat mezi zjednodušenou a podrobnější podobou klepnutím pravého tlačítka myši na titulek algoritmu.

Každý algoritmus může mít řadu vstupů a výstupů. Ty je možné pomocí metody drag & drop vzájemně propojovat. Pokud propojení není možné, systém **NPRlab** jej nepovolí vytvořit. Propojení jsou znázorňována jako křivky. Speciální propojení, které určuje pořadí vykonávání algoritmů, je znázorněno křivkou čárkovanou, ostatní propojení jsou znázorněny hladce. Každý algoritmus má automaticky jeden výstup `this`, který vrací ukazatel na samotný algoritmus. Obrázek 8 ukazuje způsob zobrazení algoritmů a jejich propojení.

A.10.1 Přidání nového algoritmu

Nový algoritmus je možné přidat klepnutím pravým tlačítkem myši do prázdného prostoru okna a zvolením algoritmu z nabídky `Add algorithm`. Algoritmus je zobrazen v okně a je automaticky zvolen, což se projeví světle modrou barvou jeho titulku.

Pokud je potřeba přidat algoritmus, který v menu není obsažen (například uživatel jej právě naprogramoval), je potřeba zvolit položku menu `Add user algorithm`. Uživatel v nabídnutém dialogu zadá jméno nového algoritmu (například `npr.alg.UserAlg`). Nový algoritmus je pak zařazen pod položku menu `User algorithms`, odkud je možné ho zvolit stejně jako při přidávání normálního algoritmu.



Obr. 8: Zobrazení algoritmů

A.10.2 Zrušení algoritmu

Algoritmus, který je aktuálně vybraný (má světle modrý titulek) je možné smazat zvolením položky vyskakovacího menu `Delete algorithm`. Algoritmus je možné vybrat klepnutím na jeho titulek levým tlačítkem myši.

A.10.3 Vytvoření vazby mezi algoritmy

Vazba mezi algoritmy se vytvoří klepnutím a tažením položky vstupu a puštěním této položky nad položkou výstupu (nebo obráceně). Systém zobrazuje nově vytvářenou vazbu tlustší čarou a červeně. V okamžiku, kdy systém zjistí, že navázání je možné uskutečnit, čára zčerná. Po puštění tlačítka myši se pak vazba trvale vytvoří.

Vazbu nemusí být možné vytvořit z důvodu typové nekompatibility vstupu a výstupu. Není možné spojovat vstup se vstupem nebo výstup s výstupem. Jeden vstup může být spojen jen s jediným výstupem, výstup však může být napojen na několik vstupů. Není dovoleno vytvářet cyklické závislosti vstupů na výstupech.

A.10.4 Zrušení vazby mezi algoritmy

Vazba se zruší klepnutím a tažením za položku vstupu, kam vazba ústí. Vazba se od položky odpoutá a uživatel ji pak může napojit někam jinam. Neučiní-li tak, je vazba při puštění tlačítka myši zrušena.

A.10.5 Spuštění algoritmu

Algoritmus je možné spustit klepnutím na položku menu `Run`. Při běhu algoritmu se mohou objevovat okna vybízející uživatele k interaktivnímu zadání různých parametrů. Po zadání parametrů je ve výpočtu pokračováno po uzavření okna. Algoritmus je možné přerušit klepnutím na tlačítko `Cancel` v okně ukazujícím průběh algoritmu.

Poté, co algoritmus proběhne je databáze typicky modifikována. Opětovné spuštění téhož algoritmu může mít za následek vytvoření duplikátů dat v databázi. Proto může být (v závislosti na charakteru algoritmu) nutné databázi uvést do původního stavu, viz položka menu A.6.3.

B Programová dokumentace

Tato dokumentace obsahuje přehled základních balíků a důležitých tříd systému **NPRlab**, umožňující rychlou orientaci v systému. Dále jsou zde popsány základní způsoby rozšíření systému. Podrobná technická dokumentace vytvořená programem `javadoc` je k dispozici na přiloženém CD.

B.1 Důležité třídy a rozhraní

Tato kapitola popisuje důležité třídy a rozhraní definované v jednotlivých balících systému **NPRlab**. Není zde uveden kompletní přehled všech tříd, ten je k dispozici na přiloženém CD ve formátu dokumentace generované programem `javadoc`.

B.1.1 Balík `npr.alg`

Tento balík obsahuje základní rozhraní, které se používají při vytváření algoritmů zobrazitelných v GUI systému **NPRlab**. Aby byla tvorba algoritmů jednodušší, balík poskytuje implementace těchto rozhraní pro přímé dědění. Implementované algoritmy se nicméně okamžitě v systému nezobrazí, uživatel je musí začlenit do konfiguračního souboru, viz kapitola B.4.3. V níže uvedeném seznamu tříd nejsou obsaženy konkrétní implementace algoritmů, ty jsou popsány v kapitole 3.7.

Rozhraní `IAlgorithm`

Toto je základní rozhraní, které musí všechny algoritmy implementovat, aby byly zobrazitelné. Rozhraní neklade na algoritmus téměř žádné požadavky, kromě nutnosti pamatovat si `tag`, což je objekt používaný pro ukládání pomocných dat u algoritmu. Algoritmus také musí implementovat metodu `reset`, která po nastavení dat a použití tohoto algoritmu uvede algoritmus do původního stavu a připraví ho pro další spuštění, aby nebylo nutné celý objekt konstruovat znovu.

Algoritmy musí definovat prázdný konstruktor, aby je bylo možné vytvořit před tím, než jsou známy hodnoty vstupů. Vstupní a výstupní položky algoritmů jsou definovány pomocí metod deklarovaných ve speciálním formátu. Podporované formáty jsou následující:

`public void set`<*jméno*>`(`<*typ*>`)`	vstup algoritmu
`public` <*typ*> `get`<*jméno*>`()`	výstup algoritmu
`public boolean is`<*jméno*>`()`	výstup algoritmu
`public IFieldEditor edt`<*jméno*>`()`	editor pro vstup

Důležité je, že vstupy a výstupy algoritmů musí být objekty. Nelze tudíž použít primitivní datové typy stylu `int`. Ty je nutné balit do odpovídajících objektových obálek, jako je třída `Integer`.

Následující příklad deklaruje algoritmus `Names`, který má vstupní položku `Username` pro zadání jména uživatele a výstupní položky `Length` a `Empty`, vracející délku jména ve znacích a příznak, že zadané jméno je prázdné.

```
public class Names implements IAlgorithm
{
  String m_name; // Uložené jméno

  public Names() { m_name=""; }
  public void setUsername(String n) { m_name=n; }
  public IFieldEditor edtUsername() {
      return new StringFieldEditor("Josef");
  }
  public Integer getLength() {
    return new Integer(m_name.length());
  }
  public Boolean isEmpty() {
    return new Boolean(getLength()==0);
  }

  // ... implementace rozhraní IAlgorithm ...
}
```

V příkladu je pro snadné zadání jména použit editor `StringFieldEditor`, který je na začátku naplněn implicitní hodnotou `Josef`. Podrobnější popis editorů vstupů je obsažen v kapitole B.1.20.

Abstraktní třída `Algorithm`

Základní implementace rozhraní `IAlgorithm` pro snadné vytváření algoritmů. Tato implementace řeší nastavováni položky `tag`. Původní deklaraci algoritmu uvedeného v předchozí kapitole je tak možné napsat jako

```
public class Names extends Algorithm {
  ...
}
```

V těle algoritmu pak není nutné implementaci rozhraní `IAlgorithm` uvádět. Jedinou výjimku tvoří metoda `reset`, kterou je typicky potřeba upravit.

Rozhraní `ICalculation`

Rozšíření rozhraní `IAlgorithm` o podporu dlouhotrvajících přerušitelných výpočtů. Tyto výpočty běží ve zvláštním vlákně. Postup činnosti je následující — nejprve jsou algoritmu nastaveny všechny dostupné vstupy a algoritmus je spuštěn. Poté, co jeho výpočet doběhne, jsou teprve čteny jeho výstupy. Protože funkce tohoto rozhraní je celkem obsáhlá, je doporučeno použít přímo její implementaci `Calculation`.

Abstraktní třída `Calculation`

Základní implementace rozhraní `ICalculation` pro snadné vytváření dlouhotrvajících výpočtů. Jedinou metodou, kterou je nutné implementovat, je `runCalculation`. Tato metoda provádí vlastní výpočet. Pokud dojde během výpočtu k jakékoliv chybě, metoda může vyhodit výjimku `CalculationException`.

Rozhraní `IInteractiveAlgorithm`

Rozšíření rozhraní `IAlgorithm` o podporu interaktivních algoritmů. Tyto algoritmy podporují metodu `getView`, která vrací objekt typu `IView`, reprezentující okno pro interakci s uživatelem. Po ukončení interakce musí algoritmus dát vědět pomocí rozhraní `ICalculationFinishedSource`, že výpočet skončil. K implementaci tohoto rozhraní je doporučeno použít třídu `InteractiveAlgorithm`.

Abstraktní třída `InteractiveAlgorithm`

Základní implementace rozhraní `IInteractiveAlgorithm` pro snadné vytváření interaktivních algoritmů. Jedinou metodou, kterou je třeba implementovat, je metoda `getView` pro získání okna pro interakci s uživatelem.

Rozhraní `ISceneModifier`

Rozhraní, kterým se označují algoritmy potřebující pro svou práci databázi systému. Rozhraní definuje jedinou metodu `setScene`, která před započetím činnosti algoritmu nastavuje informaci o zpracovávané scéně, která mimo jiné obsahuje i databázi. Více informací o objektu scény je uvedeno v kapitole B.1.10.

B.1.2 Balík `npr.alg.action`

V tomto balíku jsou soustředěny třídy tzv. *akcí*. Akce jsou jednoduché algoritmy, které mohou být aplikovány na určitý objekt.

Rozhraní `IAction`

Toto rozhraní musí implementovat každá akce. Rozhraní definuje jedinou metodu `perform`, která zadanou akci vykoná. Tato metoda dostává jako parametr objekt, na který se má akce aplikovat. Kromě samotného objektu lze také předat tzv. *kontext*, což je objekt specifikující dodatečné parametry akce. Protože je však možné vytvářet akce s parametry předem nastavenými, není kontext většinou použit.

Rozhraní `IApply`

Kromě akcí samotných se v balíku objevují i algoritmy, které akce dokáží aplikovat na jednotlivé prvky datových struktur, jako jsou pole, vektory, nebo celá databáze systému **NPRlab**. Tyto algoritmy implementují rozhraní `IApply`. Toto rozhraní definuje jedinou metodu `perform`, která má o parametr víc, než stejnojmenná metoda pro akce - novým parametrem je struktura, na kterou se má metoda aplikovat.

Konkrétní implementace akcí jsou popsány v kapitole 3.8.

B.1.3 Balík `npr.alg.action.subclass`

Tento balík obsahuje specializované algoritmy, které jsou schopné vytvořit pro daný meta-objekt podtřídu. Tato podtřída je pak vrácena a uživatel ji může meta-objektu ručně přidat do správné vrstvy.

Rozhraní `ISubclassConstructor`

Základní rozhraní implementované všemi třidami tohoto balíku. Rozhraní definuje metodu `newInstance`, která vytvoří novou podtřídu pro zadaný meta-objekt. Kromě této metody rozhraní definuje metodu `wantsCreate`, která je schopná ještě před vlastním vytvořením podřídy zjistit její budoucí typ. To je výhodné, pokud je potřeba najít třídu, která dokáže meta-objektu vytvořit podtřídu zadaného typu.

B.1.4 Balík `npr.alg.image`

Tento balík sdružuje algoritmy a datové struktury pro práci s dvourozměrným obrazem.

Rozhraní `IImage`

Toto rozhraní definuje metody pro práci obrázkem určitých rozměrů. Tento obrázek podporuje řadu metod, pomocí nichž je na něj možné kreslit. Obrázek si pamatuje, co na něj bylo nakresleno. Při kreslení je v závislosti na konkrétní implementaci rozhraní buď vytvářena bitmapa určité barevné

hloubky, nebo jsou příkazy ukládány přímo vektorově pro budoucí zobrazení. Jako implementaci rozhraní `IImage` je také možné vytvořit filtr pro ukládání výstupních dat přímo do souboru.

B.1.5 Balík `npr.alg.image.brush`

Tento balík sdružuje specializované algoritmy pro simulaci různých druhů štětců. Štětce používají rozhraní `IImage` pro vstup obrázku, do kterého kreslí. Jako vstup dostává štětec meta-objekt popisující křivku, kterou je třeba vykreslit. Na štětci pak záleží, kolik informace z meta-objektu je schopen pro vykreslování použít.

Rozhraní `IBrush`

Obecné rozhraní implementované všemi štětci. Definuje metodu `render` pro vykreslení zadané čáry do zadaného obrázku. Složitější algoritmy pro jednotlivé štětce jsou popsány v kapitole 2, seznam implementovaných štětců je v kapitole 3.11.

B.1.6 Balík `npr.alg.select`

Tento balík obsahuje třídy tzv. *selektorů* — algoritmů, které definují určitou množinu objektů. Selektor umí pro zadaný objekt určit, zdali do této množiny patří.

Rozhraní `ISelector`

Základní rozhraní, které všechny selektory musí implementovat. Rozhraní definuje jedinou metodu `isSelected`, která jako parametr dostává objekt, o kterém selektor určí, jestli bude nebo nebude vybrán. Selektory bývají typicky použity jako parametr akce `ActionSelect`.

Seznam implementovaných selektorů je v kapitole 3.12.

B.1.7 Balík `npr.data`

Tento balík sdružuje třídy implementující datové struktury systému **NPR-lab**. Kromě vlastních dat obsahuje i prostředky úzce související s manipulací s nimi. Protože jednotlivých tříd bylo velké množství, byly dále rozděleny do několika menších balíků, které jsou uvedeny níže.

B.1.8 Balík `npr.data.parse`

Tento balík obsahuje třídy pro zpracování XML dokumentů.

Rozhraní `IParseNode`

Toto rozhraní je použito při definici tříd, které jsou schopné zpracovat vždy jeden uzel XML dokumentu — tzv. *parsery*. Zpracování uzlu probíhá následujícím způsobem:

1. Je zavolána metoda `startParse`, která dostává jako parametr XML uzel, který je potřeba zpracovat. V této metodě je možné provést potřebnou inicializaci.

2. Pokud má XML uzel nějaké atributy, jsou zpracovány jako by se jednalo o potomky tohoto uzlu, viz krok 3.

3. Pokud má XML uzel nějaké potomky, pro každý z nich je zavolána metoda `parseChild`. Tato metoda vrátí pro každého potomka objekt parseru, který je schopen tohoto potomka zpracovat. Pokud metoda potomka nerozpozná, vrátí `null` a větev odpovídající tomuto potomkovi nebude dále zpracovávána.

4. Rekuzivně je zavoláno zpracování uzlu potomka.

5. Zpracovaný potomek je předán zpět parseru metodou `childParsed`, který může použít data jím zpracovaná k další činnosti.

6. Pokud XML uzel obsahuje text, který již není dále členěn, je tento text předán objektu pomocí metody `parseText`.

7. Kroky 3 až 6 jsou opakovány, dokud uzel obsahuje nezpracovaná data.

8. Jako poslední je zavolána metoda `endParse`, která signalizuje konec zpracování. V této metodě je možné provést úklid pomocných dat a uložit výsledky.

Rozhraní `IParseContext`

Toto rozhraní definuje objekt, který poskytuje parserům (viz výše) dodatečná data potřebná pro jejich činnost. Objekty implementující toto rozhraní musí umožňovat překlad relativních cest a ukládání pomocných proměnných, pomocí kterých spolu mohou jednotlivé parsery komunikovat. Kromě těchto funkcí rozhraní také definuje metody pro hlášení chyb, které se mohou při zpracování XML dokumentu vyskytnout.

B.1.9 Balík `npr.data.io`

Tento balík obsahuje nízkoúrovňové třídy pro vstup a výstup dat. Běžný uživatel s nimi do styku nepřijde, protože jsou typicky používány objekty na vyšší úrovni, které jejich služby dále rozšiřují.

Rozhraní `IXMLStoreable`

Toto je jediné rozhraní tohoto balíku, které uživatel typicky bude implementovat. Označuje třídy, které jsou schopné uložit informace o sobě do XML dokumentu. Rozhraní definuje jedinou metodu `storeToXML`, která jako parametry dostává XML dokument pro uložení dat a uzel, do kterého se mají data uložit. Metoda má možnost v tomto uzlu vytvořit libovolně složitou strukturu.

B.1.10 Balík `npr.data.scene`

Tento balík sdružuje třídy pro popis trojrozměrné vstupní scény a její základní zpracování. Protože je těchto tříd velké množství, jsou děleny dále do menších balíku. Tento balík proto obsahuje jenom obecné třídy.

Třída `Scene`

Tato třída popisuje scénu pro zpracování algoritmy. Scéna se skládá z databáze objektů a z databáze vrstev. Přístup k databázi objektů je realizován pomocí metody `getRoot`, která vrací kořen databázového stromu. Pro práci s vrstvami slouží metody z rozhraní `ISubclassLayerFactory`, které je blíže popsáno v kapitole B.1.15. Důležitou vlastností objektu scény je jeho schopnost načíst svá data z XML dokumentu. Při vytváření scény je jako jeden parametr konstruktoru předáván objekty typu `InputSource`, který reprezentuje dokument. Pomocí tohoto objektu může být dokument nahrán buď přímo z disku, nebo ze sítě.

Třída `SceneParseContext`

Tato třída obsahuje základní data a metody, které jsou k dispozici třídě `Scene` při zpracování XML dokumentu. Třída poskytuje pomocí metody `getLayer` výchozí vrstvu, do které se umístí případné vytvářené podtřídy metaobjektů.

Metoda `getChildParseNode` poskytuje pro zadané jméno uzlu parser (viz kapitola B.1.8), který je schopen zpracovat data tohoto XML uzlu. Tento objekt je po zpracování dat přidán meta-objektu XML uzlu. Tímto způsobem je tedy možné automaticky zpracovat databázi okamžitě po načtení vstupního souboru.

B.1.11 Balík `npr.data.scene.geom`

Tento balík shrnuje třídy pro definici geometrie scény a práci s ní.

Rozhraní `IGeometryElement`

Geometrie scény je popsána jednotlivými geometrickými elementy, jako jsou úsečky, křivky, plochy, nebo jednotlivé vrcholy. Tyto elementy musí implementovat rozhraní `IGeometryElement`, aby bylo možné je zařazovat do meta-objektů.

Tyto elementy většinou patří tzv. *geometrickému kontextu* — což je určitá množina elementů, které jsou na sobě vzájemně závislé. Elementy patřící kontextu je možné transformovat do jiného kontextu. Pro transformaci definuje rozhraní metodu `transform`. Tato metoda dostává jako parametr kontext, do kterého se transformuje a dále příznak typu `boolean`. Pokud je příznak nastaven, při transformaci složených geometrických elementů se zároveň přetransformují i všechny elementy, na kterých složený element závisí. Pokud parametr nastaven není, složený element se nejprve do kontextu podívá, zdali již v něm transformované elementy nejsou obsaženy pomocí metody kontextu `getTransformed`.

Pokud element není schopen transformovat se do zadaného kontextu, vrátí metoda `transform` hodnotu `null`, jinak je vrácen nový, transformovaný element.

Pokud se transformace popisovaná kontextem změní (například uživatel změní úhel pohledu na scénu), bylo by nutné vytvořit novou sadu transformovaných objektů, což by bylo časově i paměťově náročné. Proto rozhraní definuje metodu `updateTransform`, která je volána na transformovaném objektu. Jako parametr transformovaný objekt dostává zdrojový objekt, ze kterého byl transformován. Klasická sekvence operací je následující:

```
IGeometryElement a, b;
IGeometryContext context;

// ...
b = a.transform(context, true);

// ... Změna parametrů kontextu ...

b.updateTransform(a, context);
```

Rozhraní `IGeometryContext`

Toto rozhraní popisuje geometrický kontext, ve kterém jsou definovány jednotlivé geometrické elementy. Každý geometrický kontext má u sebe uloženu geometrickou transformaci, kterou provádí. Tato transformace je zjistitelná pomocí metody `getTransformation`. Kontext si dále drží databázi

svých elementů. Do databáze je možné přídávat nové elementy pomocí metody `register`. Tato metoda dostává jako parametr zdrojový objekt a jeho transformovanou verzi, která je transformovaná do tohoto kontextu. Metodou `transform` je pak všem zaregistrovaným objektům transformace přepočítána v pořadí, v jakém byly tyto objekty zaregistrovány. Díky tomu je možné například vytvořit úsečku, která si do kontextu zaregistruje jako první své koncové body a pak teprve sebe. Při transformaci kontextu jsou body transformovány jako první a úsečka tak dostává už předpočítaná data.

Kontexty mohou provádět kromě jednoduchých geometrických transformací i složitější operace, jako je ořezávání.

Rozhraní `IPoint`

Jeden bod definovaný svými homogenními souřadnicemi.

Rozhraní `ICurve`

Obecná prostorová křivka. Každá křivka má svůj počáteční a koncový bod. Kromě toho musí mít definovanou parametrizaci, aby bylo možné pomocí metody `getPoint` získat pro daný parametr pozici bodu na křivce. Metoda `isDegenerate` vrací příznak, že křivka degeneruje celá do jednoho bodu (nestačí, že má počáteční a koncový bod identický). Kromě těchto metod musí každá křivka být schopná spočítat svůj hashovací kód, který by měl být pro dvě identické křivky identický. Křivky také předefinovávají metodu `equals` tak, aby například dvě různé parametrizace též křivky byly považovány za identické.

Rozhraní `ISurface`

Obecná prostorová plocha. Plocha je definována svými hraničními křivkami.

Rozhraní `IParametricSurface`

Plocha, která má definovánu parametrizaci. Pro libovolnou dvojici parametrů $s, t \in \langle 0, 1 \rangle$ plocha definuje bod na svém povrchu. Tento bod je možné získat pomocí funkce `getPoint`.

Rozhraní `IMultiSurface`

Prostorová plocha s dírami. Tento objekt se tváří jako jediná plocha s uměle přidanými hranami spojujícími původní obrys a díry v něm. Algoritmy, které umí pracovat přímo s děravými plochami, mají možnost získat obrys celého tělesa a jednotlivé jeho díry zvlášť, pomocí metod `getNumHoles` a `getHole`.

Rozhraní `IClippable`

Toto rozhraní popisuje geometrický element, který je ořezávatelný pomocí `IClipContextu`, viz níže. Rozhraní definuje metodu `clip`, která objekt ořeže, a metodu `visibility`, která vrátí, zda byl objekt ořezán celý, je celý viditelný, nebo byly ořezány jen části.

Rozhraní `IClipContext`

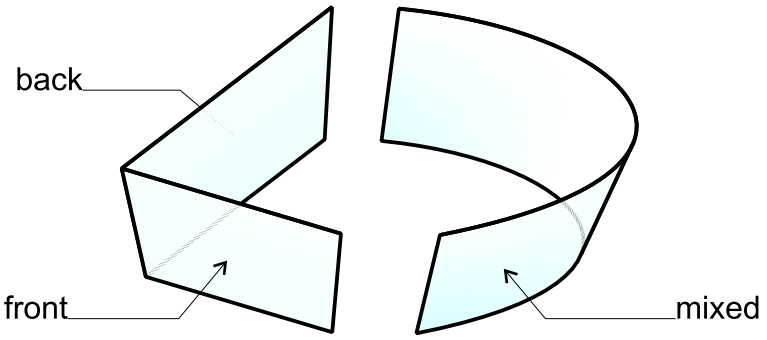
Kontext pro ořezávání. Součástí kontextu je pole poloprostorů, podle kterých se ořezává. Jednotlivé elementy jsou ořezávány podle těchto jednotlívých poloprostorů.

Rozhraní `IDynClippedCurve`

Rozhraní pro složité křivky, které není možné ořezat přímo. Tyto křivky jsou pouze schopné zjistit, zda jejich jednotlivé body jsou nebo nejsou viditelné. Při vykreslování se pak zobrazí jen ty segmenty křivky, které mají viditelné oba své konce.

Rozhraní `ISurfaceOrientation`

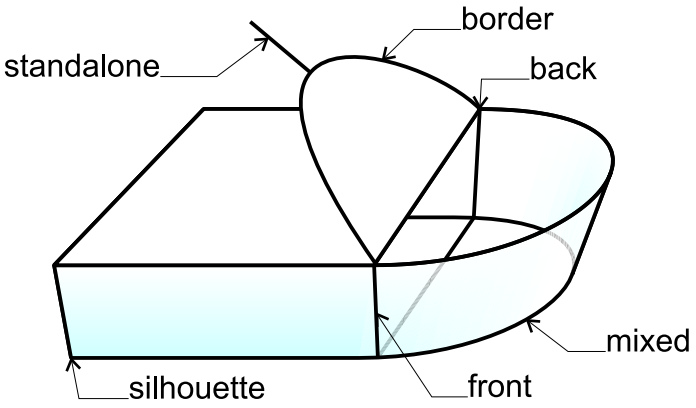
Rozhraní pro plochy schopné určovat svou orientaci vzhledem ke kameře. Plocha může být ke kameře přivrácená, nebo může být od kamery odvrácená. Složité plochy mohou být zároveň přivrácené i odvrácené, viz obrázek 9.



Obr. 9: Orientace ploch

Rozhraní `ICurveOrientation`

Rozhraní pro křivky schopné určovat svou orientaci vzhledem ke kameře. Orientace křivky je určna orientací k ní přilehlých ploch. Různé možnosti orientace křivky ukazuje obrázek 10.



Obr. 10: Orientace křivek

B.1.12 Balík `npr.data.scene.geom.trans`

Tento balík obsahuje třídy pro transformaci souřadnic.

Rozhraní `I3DTransformation`

Toto rozhraní popisuje libovolnou transformaci homogenních souřadnic. Každá transformace musí být schopná zobrazit zadaný bod pomocí funkce `transform` a dále musí umět vrátit transformaci vzniklou složením sebe s transformací zadanou.

Rozhraní `I3DLinearTransformation`

Toto rozhraní popisuje lineární transformace homogenních souřadnic, tedy veškeré transformace, které lze vyjádřit maticí 4×4. Protože se jedná o transformace homogenních souřadnic, je tímto způsobem popsatelná i perspektivní projekce.

Rozhraní `I3DAffineTransformation`

Toto rozhraní popisuje afinní transformace. Tyto transformace jsou speciálním případem transformace lineární, protože nemění čtvrtou složku homogenních souřadnic. Maticový zápis afinní transformace

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & t_x \\ a_{21} & a_{22} & a_{23} & t_y \\ a_{31} & a_{32} & a_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

je možné rozdělit do dvou částí — je to matice A rozměrů 3×3 definující změnu směru souřadných os a vektor $\vec{t}$ definující posunutí. Tyto dvě části matice umožňuje rozhraní nastavovat samostatně.

Rozhraní `ILinePreserving`

Toto rozhraní nedefinuje žádné metody, používá se jenom jako značka pro transformace, které zobrazí úsečky opět jako úsečky. Transformace označené tímto rozhraním umožňují urychlit zpracování úseček, u kterých pak stačí transformovat jen jejich koncové body.

B.1.13 Balík `npr.data.scene.solids`

Zde jsou umístěny třídy popisující jednotlivá tělesa, která systém **NPRlab** podporuje. Některá tělesa je díky jejich charakteru možné reprezentovat odděleně jako seznam vrcholů, hran nebo ploch. Popis těchto těles je rozdělen do několika tříd, které se dají složit do jednoho meta-objektu obsahujícího plnou informaci o tělese. Na druhé straně složitá tělesa, jako je třeba `mesh` takto dělit nelze. Aby bylo možné jednotlivé třídy do meta-objektu správně složit, jsou v tomto balíku obsaženy i speciální třídy pro synchronizaci.

Rozhraní `IReferenced...`

Tato rozhraní implementují objekty, které obsahují jako své datové položky odkazy na jednotlivé elementy geometrického popisu tělesa. Rozhraní definuje metody pro zjištění počtu těchto elementů a přístup k nim. Jedná se o následující tři rozhraní:

- `IReferencedPoints` — vrcholy tělesa
- `IReferencedCurves` — hrany tělesa
- `IReferencedSurfaces` — plochy tělesa

Rozhraní `IReferences`

Toto rozhraní shrnuje všechny tři výše uvedené rozhraní. Kromě toho doplňuje jejich funkci o možnost přidávání podtříd meta-objektům jednotlivých vrcholů, hran a ploch.

Třídy `Cube...`

Tato sada tříd popisuje jediné těleso — krychli, několika různými způsoby:

- `CubePoints` — vrcholy krychle
- `CubeLines` — hrany krychle
- `CubeSurfaces` — stěny krychle
- `CubeTopology` — informace o topologii krychle

Aby bylo možné používat tyto objekty v kombinaci, byla vytvořena třída `CubeReferences`, která poskytuje odkazy na jednotlivé části krychle. Tak je možné vytvořit objekt popisující topologii, jehož datové položky se správně navážou na meta-objekty již existujících bodů nebo čar. Tyto jednotlivé reprezentace krychle byly od sebe odděleny, protože ne vždy je potřeba mít u krychle zadané všechny druhy dat najednou. Například pro zobrazení drátového modelu stačí samotný objekt `CubeLines`.

Třída `Mesh`

Tato třída implementuje nejobecnější těleso, definované sítí vrcholů, křivek a ploch, které tyto vrcholy spojují. Plochy mohou být libovolného druhu, včetně ploch děravých. Třída umožňuje nahrávat svá data z XML databáze, transformovat těleso a konvertovat případné zakřivené plochy na sítě trojúhelníků.

B.1.14 Balík `npr.data.scene.topo`

V tomto balíku jsou obsaženy třídy definující topologii scény. Topologická informace je typicky svázána s informací geometrickou. Například třída popisující topologii vrcholu je většinou součástí meta-objektu vrcholu, který zároveň obsahuje i informaci geometrickou.

Rozhraní `ITopologyProvider`

Toto rozhraní je určeno pro tělesa scény, která jsou schopná vygenerovat informaci o své topologii.

Rozhraní `ITopologyElement`

Toto rozhraní je společné všem třídám popisujícím topologii. Zajišťuje, že tyto třídy budou začlenitelné do meta-objektů.

Rozhraní `IVertex`

Rozhraní popisující vrchol. Topologická informace vrcholu obsahuje seznam hran, které z vrcholu vychází a seznam cyklů hran (viz rozhraní `ILoop`), které vrchol obsahují.

Rozhraní `IEdge`

Rozhraní popisující hranu. Každá hrana zná svůj počáteční a koncový vrchol a cykly (viz rozhraní `ILoop`), kterých je součástí. Pro tzv. *2-manifoldy* tyto cykly musí být právě dva.

Rozhraní `ILoop`

Rozhraní popisující uzavřený cyklus hran náležící jedné ploše. Tento cyklus je většinou možné s plochou ztotožnit, kromě děravých ploch, které těchto cyklů mohou obsahovat několik. Každý cyklus má seznam svých hran a odkaz na plochu, které náleží. Pro každou hranu je cyklus schopen určit její orientaci tak, aby hrany na sebe postupně navazovaly.

Rozhraní `IFace`

Rozhraní popisující plochu. Plocha se skládá z minimálně jednoho cyklu, pokud je cyklů víc, definují díry v ploše. Plocha zná všechny své cykly a tzv. *povrch* tělesa (viz rozhraní `IShell`), kterému patří.

Rozhraní `IShell`

Rozhraní popisující tzv. *povrch* tělesa. Povrchem je míněna množina na sebe navazujících ploch. Většina těles má jediný povrch, tělesa s bublinami mohou mít povrchů více. Teoreticky je možné pomocí povrchů popisovat i tělesa složená z několika nesouvisejících komponent, ačkoliv tato tělesa nejsou příliš užitečná. Každý povrch zná všechny své plochy a těleso, kterému patří.

Rozhraní `IBody`

Toto rozhraní popisuje těleso. Těleso je tvořeno jedním a více povrchy.

Implementace rozhraní

Výše uvedená rozhraní jsou implementována dvěmi sadami tříd. Jedna z nich je statická, takže po vytvoření už neumožňuje topologickou informaci měnit. Tyto třídy jsou pojmenovány stejně jako rozhraní, bez počátečního znaku `I`... tedy rozhraní `IBody` je implementováno třídou `Body`.

Druhá sada je dynamická, určená pro ukládání informace o topologii, která se může měnit, nebo je teprve vytvářena. Třídy z druhé sady mají předponu `Dyn`.

B.1.15 Balík `npr.data.subclass`

Tento balík obsahuje objekty potřebné pro definici a zpracování meta-objektů[12] Jsou to třídy definující jednotlivé části meta-objektu, třídy definující jednotlivé vrstvy, na které se meta-objekt člení a třídy pro manipulaci s meta-objekty.

[12]Hrubá představa meta-objektu je spolu s obrázkem uvedena v kapitole 3.3.3.

B.1.16 Balík `npr.error`

Zde jsou obsaženy třídy a rozhraní pro zpracování chyb. Kromě objektů reprezentujících chyby to jsou i objekty, které umí chyby zobrazovat.

Rozhraní `IReport`

Rozhraní popisující hlášení o výskytu chyby. Toto hlášení má přiřazenu prioritu, textový popis chyby, případné jméno souboru, ve kterém byla chyba detekována a číslo řádky. Libovolné z těchto údajů lze vynechat.

Rozhraní `IErrorReporter`

Toto rozhraní definuje metodu `report`, které umí zadané hlášení o chybě zpracovat. Objekty implementující toto rozhraní hlášení většinou zobrazují uživateli přímo v GUI systému **NPRlab**, případně ho tisknou na systémovou konzoli.

B.1.17 Balík `npr.ui`

Tento balík obsahuje veškeré třídy, které tvoří uživatelské prostředí systému **NPRlab**. Protože byla pro vytvoření uživatelského prostředí použita knihovna Swing[13], mnoho těchto tříd je potomky objektů této knihovny, nebo je ke své činnosti využívá.

B.1.18 Balík `npr.ui.actions`

Objekty v tomto balíku definují akce vyvolatelné uživatelem, například z menu. Tyto akce nemají nic společného s akcemi z balíku `npr.alg.actions`.

B.1.19 Balík `npr.ui.alg`

Tento balík obsahuje třídy pro grafické zobrazení algoritmů z balíku `npr.alg` a pro práci s nimi v rámci uživatelského prostředí systému **NPRlab**. Kromě zobrazení algoritmů jsou zde definovány i třídy pro kreslení spojnic mezi jejich vstupy a výstupy. Protože uživatel systému nebude typicky do tohoto balíku potřebovat jakýmkoliv způsobem zasahovat, nejsou zde podrobně rozepsány jednotlivé třídy balíku.

B.1.20 Balík `npr.ui.alg.fields`

Tento balík obsahuje třídy implementující editory vstupních položek algoritmů. Tyto editory usnadňují zadávání jednoduchých vstupů algoritmů tím, že uživateli umožňují tyto vstupy přímo editovat. Algoritmy mohou určit,

[13] Byla použita knihovna Swing verze 1.1.1.

který editor bude pro editaci jejich vstupů použit pomocí metod s prefixem `edt`, viz popis rozhraní `IAlgorithm` v kapitole B.1.1.

Rozhraní `IFieldEditor`

Základní rozhraní pro editor vstupů. Tento editor musí umět pomocí metody `getComponent` poskytnout komponentu knihovny Swing, která zajišťuje vlastní editaci. Kromě toho editory vstupů musí být schopné uložit svá data do XML dokumentu a opět je z tohoto dokumentu načíst, aby bylo možné ukládat algoritmy vytvořené v uživatelském prostředí systému **NPRlab**.

B.1.21 Balík `npr.ui.dv`

Tento balík obsahuje rozhraní pro definici tzv. *document-view modelu* pro zobrazování dokumentů v uživatelském prostředí. Tento model popisuje *document*, neboli objekt obsahující data o dokumentu, a *view*, což je pohled na data dokumentu zobrazovaný v samostatném okně. Uživatel pomocí tohoto modelu může otevřít několik oken, které zobrazují týž dokument různými způsoby.

Rozhraní `IDocument`

Rozhraní popisující nahraný dokument. Tento dokument zná všechny pohledy na svá data a je schopen v případě své změny nechat tyto pohledy překreslit.

Rozhraní `IView`

Rozhraní popisující pohled na data dokumentu. Pohled zná svůj titulek, umí se skrýt a zase objevit a je zdrojem událostí o změnách svého stavu. Tak se mohou další části systému dozvědět o tom, co uživatel s určitým pohledem momentálně provádí.

B.1.22 Balík `npr.ui.parse`

Tento balík obsahuje třídy pro zpracování konfiguračního souboru s popisem uživatelského prostředí systému **NPRlab**. Jednotlivé sekce tohoto souboru jsou zpracovávány jednotlivými třídami pro větší přehlednost. Popis formátu konfiguračního souboru je uveden v kapitole B.4.

B.1.23 Balík `npr.ui.tree`

Tento balík obsahuje třídy pro zobrazení jednotlivých položek stromu databáze scény. Je zde obsažen model tohoto stromu, speciální filtry pro zobrazení jen některých položek a další pomocné třídy.

B.1.24 Balík `npr.ui.tree.render`

Tento balík obsahuje třídy schopné zobrazit jeden uzel databáze. Třídy jsou schopné rozpoznat uzly, jejichž zobrazení umožňují. Způsob, kterým se přidávají nové třídy tohoto druhu je popsán v kapitole B.2.4.

Abstraktní třída `AbstractSceneNode`

Tato třída obsahuje základní implementaci objektu pro zobrazování uzlů systémové databáze. Ostatní třídy pro zobrazování objektů tuto třídu rozšiřují.

B.1.25 Balík `npr.util`

Tento balík obsahuje třídy obsahující pouze statické metody a datové položky. Jedná se o soubor určitých pomocných funkcí, které jsou užitečné natolik rozdílným třídám, že je nebylo možné zařadit do jiných balíků.

Třída `Assert`

Tato třída implementuje metodu `Assert`, která odpovídá makru `ASSERT` z jazyka C++. Tato metoda umožňuje kontrolu určité podmínky, při jejímž nesplnění je vykonávání programu přerušeno. Metoda je velice užitečná při ladění celého systému a je možné její funkci snadno zrušit nastavením datové položky `enabled` této třídy na `false`. Metoda při nesplnění daných podmínek vyhazuje výjimku `AssertionException`, která be neměla být nikde v systému odchytávána.

Třída `MathWorks`

Třída se základními matematickými operacemi, které se často opakují. Umí počítat vektorové a skalární součiny, průsečíky různých geometrických objektů a podobně.

Třída `PathWorks`

Tato třída usnadňuje práci s cestami. Umožňuje ořezávat koncová lomítka, navazovat dvě cesty na sebe a podobně.

Třída `StringWorks`

Tato třída obsahuje základní funkce pro zpracování řetězců. Umí například načíst z řetězce souřadnice vrcholů, pole čísel a poskytuje další užitečné metody pro pohodlnější programování.

B.2 Rozšiřování systému

V této kapitole je popsáno několik frekventovaných postupů, jak systém **NPRlab** rozšířit.

B.2.1 Vytváření vlastních algoritmů

Vytvoření dalšího algoritmu je jednoduché, pokud jsou použity připravené třídy. Ačkoliv je možné algoritmus vytvořit přímou implementací rozhraní `IAlgorithm`, tento postup je zbytečně pracný a může vést k chybám.

Nejprve je třeba určit, zdali se bude jednat o obyčejný, přerušitelný, nebo interaktivní algoritmus. Podle toho je potřeba vytvořit podtřídu tříd `Algorithm`, `Calculation` nebo `InteractiveAlgorithm` z balíku `npr.alg`.

Vytvořené podtřídě je třeba definovat metody pro vstupy a výstupy. To se realizuje pomocí metod se jmény ve speciálním formátu, viz kapitola B.1.1. Dále je dobré implementovat metodu `reset`, která by měla obnovit stav těsně po skončení konstruktoru třídy — pomocí této metody je možné algoritmus spouštět několikrát bez nutnosti vytváření další instance.

Pro interaktivní algoritmy je potřeba definovat metodu `getView`, která vrátí objekt typu `IView` pro interakci s uživatelem. Tento objekt je možné vytvořit rozšířením objektu `ViewImpl`, případně přímou implementací rozhraní `IView` z balíku `npr.ui.dv`. V okamžiku ukončení interakce uživatele s algoritmem je důležité odstranit vytvořený objekt typu `IView` pomocí jeho metody `putOffDesktop`. Kromě toho je nutné dát vědět zbytku systému, že interakce skončila. To je realizováno zavoláním metody `fireCalculationFinished`.

Algoritmy s přerušitelným výpočtem musí mít definovanou metodu `runCalculation`, která provádí vlastní výpočet. O tom, že výpočet skončil, bude systém informován automaticky po ukončení této metody.

Posledním krokem je začlenění nového algoritmu do menu systému **NPRlab**. To je realizováno přidáním položky do konfiguračního souboru systému. Přesné informace o formátu tohoto souboru a zvláště jeho sekce pro definice menu algoritmů jsou uvedeny v kapitole B.4.3. Další možností je dočasné přidání třídy algoritmu přímo za běhu systému z uživatelského prostředí, viz uživatelská dokumentace, kapitola A.10.

B.2.2 Rozšiřování vstupního formátu

Vstupní formát pro definici scény je možné rozšířit naprosto libovolně. Databáze bude načtena a připravena pro zpracování algoritmy. Rozšíření vstupního formátu tedy spočívá pouze ve vytvoření dalších algoritmů, které budou nové vstupy rozpoznávat.

Existuje také možnost předzpracovat vstupní data hned po načtení. Je potřeba vytvořit nové objekty parserů (viz kapitola B.1.8) pro nově zavedené uzly XML dokumentu. Pak je potřeba pozměnit metodu `getChildParseNode`

třídy `SceneParseContext` z balíku `npr.data.scene` tak, aby vracela pro zadané jméno uzlu nově implementovaný parser.

B.2.3 Přidávání nových uživatelských akcí

Pokud je potřeba přidat novou akci, kterou bude uživatel vyvolávat z menu, je potřeba vytvořit podtřídu jednoho z následujících objektů z balíku `npr.ui.actions`:

- `UserAction` — akce, kterou je možné provést kdykoliv
- `UserCheckBoxAction` — akce proveditelná kdykoliv, zobrazující navíc zaškrtávací políčko
- `ViewDependentAction` — akce, která závisí na aktuálně zobrazeném pohledu na scénu. Tato akce může být povolená, nebo zakázaná
- `ViewDependentCheckBoxAction` — dtto, navíc zobrazuje zaškrtávací políčko

Vlastní tělo akce je umístěno v metodě `actionPerformed`. Stav zaškrtávacího políčka je možné zjistit metodou `getSelected` a nastavit metodou `setSelected`. U akce závislé na pohledu na scénu je třeba definovat metodu `viewActivated`, která je zavolána, pokud se změní aktivní pohled na scénu. Kromě této metody je pro pohodlné programování volána i metoda `allViewsClosed` pokud uživatel zavře všechny pohledy na scénu. Protože jedna akce může vykonávat několik různých činností, je možné definovat metodu `setParam`, která umožní uživateli této akce specifikovat, o kterou činnost mu konkrétně jde.

Akce je po vytvoření uvést ve speciální sekci konfiguračního souboru systému **NPRlab**, viz kapitola B.4.2. To samo o sobě nestačí k tomu, aby akce byla z menu vyvolatelná. Akci je potřeba přiřadit patřičné položce menu v další sekci konfiguračního souboru, viz kapitola B.4.4.

B.2.4 Úprava zobrazení databázového stromu

Uzly databázového stromu jsou zobrazovány pomocí sady objektů rozšiřujících abstraktní třídu `AbstractSceneNode` z balíku `npr.ui.tree.render`. Tyto třídy musí definovat metodu `getTreeCellRendererComponent`, která je schopna vrátit komponentu knihovny Swing, která daný uzel zobrazí. Kromě toho je důležité definovat metodu `isRecognized`, která určí, zda je daný třída schopna zobrazit zadaný objekt.

Po vytvoření vlastního zobrazovače je potřeba jej uvést v konfiguračním souboru systému **NPRlab**, viz kapitola B.4.5.

B.3 Podporovaný vstupní formát dat

Tato kapitola popisuje formát vstupních dat systému **NPRlab**, který je momentálně rozeznáván implementovanými algoritmy. Přehled je organizován podle jednotlivých algoritmů.

B.3.1 Uspořádání souboru

Soubor s daty pro systém **NPRlab** je libovolný XML dokument. Standardní přípona jména vstupních souborů je `.npr`. Pro práci se systémem **NPRlab** je však doporučována standardní struktura dokumentu s kořenovým uzlem pojmenovaným `scene`. Typická struktura dokumentu je následující:

```
<?xml version="1.0"?>
<!DOCTYPE scene [
  <!ENTITY teddy SYSTEM "teddy.mesh">
]>
<scene>
  <!-- Vlastní data -->
</scene>
```

Speciální sekce `DOCTYPE` umožňuje vkládat do XML souboru obsah externích souborů. V našem případě je v této sekci definována jediná entita `teddy`, která bude nahrazena obsahem souboru `teddy.mesh`. Entitu je možné použít zadáním řetězce `&teddy;` v rámci vlastních dat dokumentu.

B.3.2 Zadávání transformací

Transformace je v dokumentu definována uzlem `transform`. Tento uzel definuje afinní transformaci, kterou umí zpracovat třída `AffineTransformation`, viz kapitola B.1.12. Uzel může mít následující rozeznávané atributy:

- `rotx` — rotace kolem osy x, parametr je udáván ve stupních.
- `roty` — rotace kolem osy y, ...
- `rotz` — rotace kolem osy z, ...
- `rot` — rotace postupně kolem os x, y a z. Parametrem jsou tři čísla oddělená středníky.
- `movx` — posun ve směru osy x.
- `movy` — posun ve směru osy y.
- `movz` — posun ve směru osy z.

- `mov` — posun ve směru os x, y a z. Parametrem jsou tři čísla oddělená středníky.
- `scalex` — zvětšení ve směru osy x.
- `scaley` — zvětšení ve směru osy y.
- `scalez` — zvětšení ve směru osy z.
- `scale` — zvětšení ve směru os x, y a z. Parametrem jsou tři čísla oddělená středníky.

Transformace transformuje uzly, které jsou uvedeny jako její potomci. Pokud je jako potomek transformace uvedena další transformace, tyto transformace jsou složeny v pořadí od uzlu k listům. Skládání transformací a přiřazování transformací uzlům provádí algoritmus `CalculateTransformations`, viz kapitola B.1.1.

B.3.3 Zadávání těles

Jednotlivá tělesa jsou vytvořena ve standardní poloze - jejich středem je počátek souřadné soustavy $(0, 0, 0)$. Tělesa jsou konstruována tak, aby vyplnila jednotkovou krychli. Příkazy pro jednotlivá tělesa jsou následující:

- `cube` — krychle o straně délky 1.
- `sphere` — koule o průměru 1.
- `cylinder` — válec orientovaný ve směru osy z, průměr podstavy 1, výška 1.
- `cone` — kužel orientovaný ve směru osy z, pr;m2r podstavy 1, výška 1.
- `mesh` — obecné těleso popsané sítí bodů a jejich spojnic. středníky.

Kromě obecného tělesa `mesh` není potřeba zadat žádné další atributy. Změny umístění, tvaru či velikosti těles jsou realizovány pomocí uzlů pro transformaci.

B.3.4 Obecné těleso `mesh`

Obecné těleso je zadáno množinou bodů, křivek a ploch, které tyto body spojují. Uzly s daty jsou uváděny jako potomci uzlu `mesh`. Body a křivky jsou automaticky indexovány od nuly v pořadí jejich výskytu. První uvedený bod i první uvedená křivka tak získají index 0. Tyto indexy je možné použít při odkazování se na již definované body či křivky.

Pro zadávání dat jsou možné dva alternativní přístupy, které je možné vzájemně kombinovat. Protože zadávat velké množství dat pomocí XML může být zdlouhavé a nepříjemné, existuje možnost zadat data přímo textem bez formátování. Například vrchol o souřadnicích $(1, 2, 3)$ je tak možné zapsat jako `<v> 1 2 3 </v>`, nebo mnohem jednodušeji jako `V 1 2 3`. Výhodou prvního přístupu je možnost přiřadit vrcholu další atributy, výhodou druhého přístupu je jednodušší zápis. Následující tabulka uvádí podporovaná vstupní data a způsob jejich zápisu. Vstupy x, y a z jsou reálná čísla, $i_?$ značí index vrcholu, $j_?$ značí index křivky, P a $D_?$ jsou libovolně zadané plochy.

druh vstupu	zkrácený zápis	XML zápis
vrchol	`V` $x\ y\ z$	`<v>` $x\ y\ z$ `</v>`
úsečka	`L` $i_1\ i_2$	`<l>` $i_1\ i_2$ `</l>`
trojúhelník	`T` $i_1\ i_2\ i_3$	`<t>` $i_1\ i_2\ i_3$ `</t>`
mnohoúhelník	`P` $n\ i_1\ \ldots\ i_n$	`<p>` $i_1\ \ldots\ i_n$ `</p>`
děravá plocha	`MS` $n\ P\ D_1\ \ldots\ D_n$	`<ms>` $P\ D_1\ \ldots\ D_n$ `</ms>`
Bèzierův plát 4×4	`BS` $i_{11}\ i_{12}\ \ldots\ i_{44}$	`<bs>` $i_{11}\ i_{12}\ \ldots\ i_{44}$ `</bs>`

B.4 Formát konfiguračního souboru

Soubor `config.xml`[14] obsahuje konfiguraci uživatelského prostředí systému **NPRlab**. Tento soubor se dělí na několik sekcí, které jsou obsaženy v jednom kořenovém uzlu `config`. Celý soubor tedy vypadá takto:

```
<?xml version="1.0"?>
<config>
...sekce...
</config>
```

Pořadí sekcí je důležité, protože některé sekce využívají informací definovaných v sekcích předcházejících. Při tvorbě konfiguračního souboru je dobré držet se níže popsaného pořadí.

B.4.1 Sekce `<environment>`

Tato sekce obsahuje seznam proměnných prostředí. Aplikace je používá pro zjišťování svých parametrů. Jako proměnná prostředí je uložen například rozměr okna při startu aplikace, titulek aplikace a podobně. Význam jednotlivých proměnných je dokumentován přímo v konfiguračním souboru, proto zde není znovu uveden.

Proměnné prostředí se ukládají do sekce . Momentálně jsou podporovány tři druhy proměnných prostředí:

- `<var name="`*jméno*`" text="`*textová hodnota*`"\>`
- `<var name="`*jméno*`" icon="`*cesta k obrázku*`"\>`
- `<var name="`*jméno*`" path="`*relativní cesta*`"\>`

Textové hodnoty jsou uloženy jako objekt typu `String`. Obrázky jsou nahrány a zpřístupněny jako objekty typu `javax.swing.JIcon`. Cesty jsou uloženy jako řetězce, pouze v případě uvedení relativní cesty jsou doplněny o pozici aktuálního adresáře.

B.4.2 Sekce `<actionlist>`

Zde je seznam objektů typu `npr.ui.actions.UserAction`, které vykonávají uživatelské akce. Tyto akce je pak možné vyvolávat z menu. Formát položky je následující:

`<action name="`*jméno*`" class="`*jméno třídy* `[param="`*text*`"]"\>`

Hodnota nepovinného parametru `param` je předána objektu akce beze změny a umožňuje tak použít jednu akci pro několik různých činností.

[14] Soubor je uložen v kořenovém adresáři systému **NPRlab**. Díky formátu XML však mohou být jednotlivé části tohoto souboru vkládány odjinud.

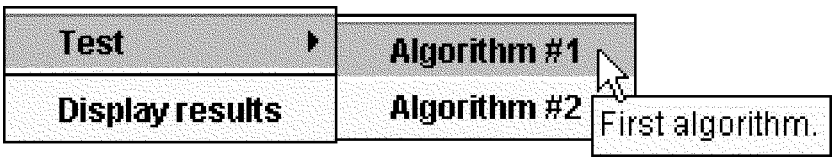
B.4.3 Sekce `<algorithmlist>`

Tyto sekce obsahují seznamy tříd, které je pak možné prezentovat uživateli. Jednotlivé sekce jsou rozlišeny parametrem `name`. Například editor algoritmů[15] používá seznam `alg.list` pro předdefinované algoritmy. Seznam je členěn do skupin pro větší přehlednost. Jsou podporovány následující položky:

- `<group name="`*jméno*`" description="`*popis*`">`
- `<algorithm name="`*jméno*`" class="`*jméno třídy*`"`
  `description="`*popis*`"\>`
- `<separator\>`

Hodnoty `description` jsou použity jako tooltipy. Jako příklad uveďme následující kód spolu s menu, které tento kód vytváří:

```
<group name="Test" description="Test of grouping.">
  <algorithm name="Algorithm #1" class="npr.alg.Alg1"
      description="First algorithm."/>
  <algorithm name="Algorithm #2" class="npr.alg.Alg2"
      description="Second algorithm."/>
</group>
<separator/>
<algorithm name="Display results" class="npr.alg.display"
  description="..."/>
```



B.4.4 Sekce `<menubar>`

Tato sekce popisuje vlastní menu aplikace. Jednotlivé položky menu se vytvářejí a sdružují do skupin podobným způsobem jako algoritmy. Seznam podporovaných položek spolu s parametry, které je jim možné přiřadit ukazuje tabulka 1.

Položka `<menu>` vytváří skupinu, do které je možné umisťovat jednotlivé položky menu, jako jsou obyčejné položky `<item/>`, které přímo vyvolávají specifikovanou akci, nebo položky `<checkbox/>`, které kromě vyvolání

[15] Používání editoru algoritmů je popsáno v kapitole A.10.

	name	key	acc	icon	tooltip	action	param
<menu>	√	√	√	√	√	×	×
<item/>	√	√	√	√	√	√	√
<checkbox/>	√	√	√	√	√	√	√
<separator/>	×	×	×	×	×	×	×
<glue/>	×	×	×	×	×	×	×

Tab. 1: Povolené parametry položek menu.

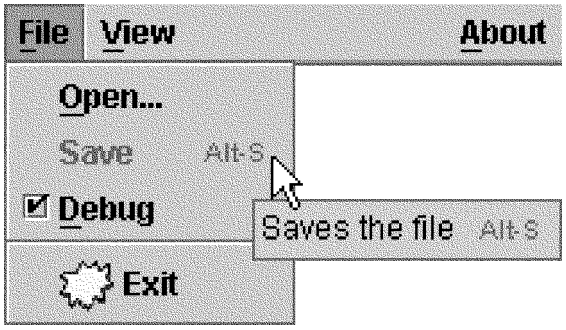
akce mohou zobrazovat příznak zapnutí/vypnutí této volby. Speciální položka `<separator/>` vkládá horizontální oddělovač, opticky členící menu na skupiny. Položka `<glue/>` je použitelná jen v nejhořejší úrovni menu. Používá se pro optické oddělení položek hlavního menu vložením mezery, která se roztahuje tak, aby menu vždy zabralo plnou šířku aplikačního okna. Význam parametrů těchto položek je následující:

- `name` — text zobrazovaný v položce.
- `key` — podtržené písmeno položky pro snadné ovládání menu pomocí klávesnice
- `acc` — horká klávesa pro rychlé vyvolání této položky menu. Může být zadána například jako `alt X` nebo `control shift C`.
- `icon` — cesta k ikoně zobrazované u menu. Ikony mohou být formátu GIF, JPEG nebo PNG.
- `tooltip` — nápověda, která se zobrazí, když uživatel nechá kurzor myši nad položkou menu.
- `action` — jméno akce, kterou položka menu vyvolává, viz kapitola B.4.2. Pokud toto jméno není uvedeno, položka menu sice je zobrazena, ale je neaktivní.
- `param` — nepovinný parametr akce, pokud nebyl zadán přímo při definici akce, viz kapitola B.4.2.

Opět uveďme příklad jednoduchého menu, které ozřejmí funkci těchto příkazů:

```
<menubar>
  <menu name="File" key="F">
    <item name="Open..." key="O" action="OPEN"/>
    <item name="Save" acc="alt S" tooltip="Saves the file"/>
    <checkbox name="Debug" key="D" action="DEBUG">
    <separator/>
    <item name="Exit" icon="data/star.gif" action="EXIT"/>
```

```
    </menu>
    <menu name="View" key="V"/>
    <glue/>
    <menu name="About" key="A">
       <item name="About NPRlab" key="A"/>
    </menu>
  </menubar>
```



B.4.5 Sekce `<scenenodes>`

Tato sekce popisuje způsob zobrazení jednotlivých položek XML dokumentu v okně databáze [16]. Datový model a vzhled tohoto zobrazení je plně určován objekty implementujícími rozhraní `npr.ui.tree.ISceneNode`. Každý takový objekt je pro zadanou položku databáze schopen určit, jestli je schopen ji zobrazit. Objekt, který je nejvhodnější pro zobrazení položky je vybrán a použit.

Vhodnost objektu je posuzována podle jeho umístění v seznamu. Z potenciálně vhodných objektů pro zobrazení dané položky je vybrán první objekt (v pořadí uvedení v seznamu), jehož všichni rodiče jsou schopni tuto položku také zobrazit, zatímco žádné z jeho dětí toho už schopné není. Tento objekt tak reprezentuje nejvíce specializovanou položku, u které se předpokládá, že zobrazí data nejlépe.

Formát popisu jednotlivých objektů je následující:

`<node class="`*jméno třídy*`"` *[další parametry...]* `>`

Kromě povinného jména třídy je možné zadat mnoho dalších parametrů, které jsou přímo předány jednotlivým objektům při jejich vytváření. Většina objektů podporuje následující parametry:

- `leaficon` — ikona zobrazovaná, je-li tato položka listem
- `openicon` — ikona zobrazovaná u neprázdných otevřených adresářů

[16] Používání okna zobrazení databáze je popsáno v kapitole A.9.

- `closeicon` — ikona zobrazovaná u neprázdných zavřených adresářů
- `icon` — neměnná ikona zobrazovaná ve všech třech výše uvedených případech
- `tooltip` — nápověda pro uživatele

Literatura

[1] Maneesh Agrawala, Denis Zorin a Tamara Munzner, *Artistic multiprojection rendering*, Rendering Techniques 2000: 11th Eurographics Workshop on Rendering (2000), 125–136.

[2] Ken Anjyo, *"Tour into the picture" as a non-photorealistic animation*, Computer Graphics **33** (1999), no. 1, 54–55.

[3] William Baxter, Vincent Scheib, Ming C. Lin a Dinesh Manocha, *DAB: Interactive haptic painting with 3d virtual brushes*, připravováno pro SIGGRAPH 2001.

[4] John Buchanan a Mario Sousa, *The edge buffer: A data structure for easy silhouette rendering*, Proceedings of the First International Symposium on Non Photorealistic Animation and Rendering (NPAR) for Art and Entertainment, 2000.

[5] Ian Buck, Adam Finkelstein, Charles Jacobs, Allison Klein, David Salesin, Joshua Seims, Rick Szeliski a Kentaro Toyama, *Performance-driven hand-drawn animation*, ? (2000), ?

[6] Jonathan M. Cohen, John F. Hughes a Robert C. Zeleznik, *Harold: A world made of drawings*, Proceedings of the First International Symposium on Non Photorealistic Animation and Rendering (NPAR) for Art and Entertainment, 2000, Annecy, Francie.

[7] Cassidy J. Curtis, Sean E. Anderson, Joshua E. Seims, Kurt W. Fleischer a David H. Salesin, *Computer-generated watercolor*, SIGGRAPH 97 Conference Proceedings (1997), 421–430.

[8] Amy Gooch, Bruce Gooch, Peter Shirley a Elaine Cohen, *A non-photorealistic lighting model for automatic technical illustration*, Computer Graphics **32** (1998), no. Annual Conference Series, 447–452.

[9] Aaron Hertzmann a Ken Perlin, *Painterly rendering for video and interaction*, ? (2000), ??–??

[10] Takeo Igarashi, Satoshi Matsuoka a Hidehiko Tanaka, *Teddy: A sketching interface for 3D freeform design*, SIGGRAPH 99 Conference Proceedings (1999), 409–416.

[11] Daniel F. Keefe, Daniel Acevedo Feliz, Tomer Moscovich, David H. Laidlaw a Joseph J. LaViola Jr., *CavePainting: A fully immersive 3D artistic medium and interactive experience*, ACM Symposium on Interactive 3D Graphics, 2001.

[12] Michael A. Kowalski, Lee Markosian, J. D. Northrup, Lubomir Bourdev, Ronen Barzel, Loring S. Holden a John Hughes, *Art-based rendering of fur, grass, and trees*, SIGGRAPH 99 Conference Proceedings (1999), 433–438, ISBN 0-20148-560-5.

[13] John Lansdown a Simon Schofield, *Expressive rendering: a review of nonphotorealistic techniques*, IEEE Computer Graphics and Applications **15** (1995), no. 3, 29–37.

[14] Lee Markosian, Michael A. Kowalski, Samuel J. Trychin, Lubomir D. Bourdev, Daniel Goldstein a John F. Hughes, *Real-time nonphotorealistic rendering*, SIGGRAPH 97 Conference Proceedings (1997), 415–420, ISBN 0-89791-896-7.

[15] Lee Markosian, Barbara J. Meier, Michael A. Kowalski, Loring S. Holden, J. D. Northrup a John F. Hughes, *Art-based rendering with continuous levels of detail*, Proceedings of the First International Symposium on Non Photorealistic Animation and Rendering (NPAR) for Art and Entertainment, červen 2000, Annecy, Francie.

[16] D. Martín, S. García a J. C. Torres, *Observer dependent deformations in illustration*, Proceedings of the First International Symposium on Non Photorealistic Animation and Rendering (NPAR) for Art and Entertainment, 2000, pp. 75–82.

[17] J. D. Northrup a Lee Markosian, *Artistic silhouettes: A hybrid approach*, Proceedings of the First International Symposium on Non Photorealistic Animation and Rendering (NPAR) for Art and Entertainment, červen 2000, Annecy, Francie.

[18] Paul Rademacher, *View-dependent geometry*, Siggraph 1999, Computer Graphics Proceedings, 1999, pp. 439–446.

[19] Ramesh Raskar a Michael Cohen, *Image precision silhouette edges*, Symposium on Interactive 3D Graphics, duben 1999., 1999.

[20] Craig Reynolds, *Stylized depiction in computer graphics: Non-photorealistic, painterly and 'toon rendering — an annotated survey of online resources*, `http://www.red3d.com/cwr/npr`.

[21] Jutta Schumann, Thomas Strothotte a Stefan Laser, *Assessing the effect of non-photorealistic rendered images in computer-aided design.*, ACM Human Factors in Computing Systems, SIGCHI '96 (1996), 35–41.

[22] Scott Sona Snibbe a Golan Levin, *Interactive dynamic abstraction*, Proceedings of the First International Symposium on Non Photorealistic Animation and Rendering (NPAR) for Art and Entertainment, 2000.

[23] Osama Tolba, Julie Dorsey a Leonard McMillan, *A projective drawing system*, ACM Symposium on Interactive 3D Graphics, 2001.

[24] D. M. Willows a H. A. Houghton, *The psychology of illustration*, Basic Research, volume 1. New York, Berlin, Heidelberg, London, Paris, Tokyo: Springer, 1987., 1987.

[25] Daniel N. Wood, Adam Finkelstein, John F. Hughes, Craig E. Thayer a David H. Salesin, *Multiperspective panoramas for cel animation*, Computer Graphics **31** (1997), no. Annual Conference Series, 243–250.

[26] Robert C. Zeleznik, Kenneth P. Herndon a John F. Hughes, *Sketch: An interface for sketching 3D scenes*, SIGGRAPH 96 Conference Proceedings (1996), 163–170.