3D vizualizace Analytického programování

3D visualization of Analytic programming

Michal Hmirák

Bakalářská práce
2008

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav aplikované informatiky
akademický rok: 2007/2008

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Michal HMIRÁK**
Studijní program:	**B 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**3D vizualizace Analytického Programování**

Zásady pro vypracování:

1. **Nastudujte základy teorie Analytického Programování.**
2. **Zhodnoťte současnou názornost prezentace daného tématu.**
3. **Zvolte vhodné příklady pro 3D vizualizaci.**
4. **Vyberte odpovídající softwarové prostředí pro tvorbu.**
5. **Vytvořte krátké 3D animace a obrázky ilustrující vybrané příklady.**
6. **Prezentujte výsledné práce.**

Rozsah práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **ZELINKA I. 2004, SOMA – Self Organizing Migrating Algoritm, kap. 7, str. 33, in B.V. Batu, G. Onwubolu (eds), New Optimization Techniques in Engineering, Springer-Verlag.**
2. **ZELINKA I., OPLATKOVÁ Z. 2003, Analytic programming – Comparative Stu-dy. CIRAS'03, The second International Conference on Computational Intelligen-ce, Robotics, and Autonomous Systems, Singapore, 2003, ISSN 0219-6131.**
3. **ZELINKA, I. 2002. Umělá inteligence v problémech globální optimalizace. Praha : BEN, 2002, 189 s. ISBN 80-7300-069-5.**
4. **VAŘACHA, Pavel. Syntéza neuronových sítí metodou symbolické regrese. Zlín, 2006. 83 s. UTB. Vedoucí diplomové práce doc. Ing. Ivan Zelinka, Ph.D.**
5. **POKORNÝ, Pavel. Blender – Naučte se 3D grafiku. Praha : BEN, 2006. 248 s. ISBN 80-7300-203-5.**
6. **3D scéna [online]. 2003 [cit. 2008-01-28]. Dostupný z WWW: http://www.3dscena.cz/3dshowks.php?xuid=207 .**
7. **Blender [online]. 2001 [cit. 2008-01-28]. Dostupný z WWW: http://www.blender.org/education-help/tutorials/c847 .**

Vedoucí bakalářské práce: **Ing. Pavel Vařacha**
Ústav aplikované informatiky

Datum zadání bakalářské práce: **20. února 2008**

Termín odevzdání bakalářské práce: **5. května 2008**

Ve Zlíně dne 20. února 2008

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

prof. Ing. Vladimír Vašek, CSc.
děkan

doc. Ing. Ivan Zelinka, Ph.D.
ředitel ústavu

ABSTRAKT

Táto práca sa zaoberá otázkou vizualizácie algoritmu analytického programovania s využitím evolučných algoritmov. Základnou úlohou práce bolo vhodné znázornenie priebehu algoritmu a jeho možností pri syntéze riešenia problému symbolickej regresie. Hlavnou motiváciou tejto práce je názorné a jednoduché znázornenie prebiehajúceho algoritmu pomocou dvojdimenzionálnych a trojdimenzionálnych animácii zobrazujúcich základné vlastnosti algoritmu potrebné pre jeho pochopenie. Práca obsahuje teoretické informácie k problému symbolickej regresie a vizualizácie súvisiace s riešením problému pomocou analytického programovania na priloženom DVD.

Kľúčové slová: vizualizácia, analytické programovanie, symbolická regresia, evolučné algoritmy

ABSTRACT

This thesis deals with question of visualization of analytic programming algorithm using evolutionary algorithms. The very task was to create a suitable visualization of running of this algorithm and its possibilities in case of synthesis of solution of symbolic regression. Motivation is simple and transparent visualization of algorithm using 2D and 3D animations showing basic principles necessary for its understanding. Thesis contains theoretical background of solving problem of symbolic regression by means of analytic programming and its visualization on attached DVD.

Keywords: visualization, analytic programming, symbolic regression, evolutionary algorithms

Na tomto mieste by som chcel poďakovať vedúcemu mojej bakalárske práce, Ing. Pavlovi Vařachovi, za jeho pomoc, metodické vedenie a uvedenie ma do problematiky analytického programovania a symbolickej regresie.

Taktiež by som chcel poďakovať mojím rodičom, ktorí mi umožnili štúdium na vysokej škole a počas štúdia ma hmotne a morálne podporovali.

Prohlašuji, že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků, je-li to uvolněno na základě licenční smlouvy, budu uveden jako spoluautor.

Ve Zlíně …….

Podpis diplomanta

OBSAH

ÚVOD

V dnešnej dobe je v inžinierskych výpočtoch čoraz viac potrebné aproximovať namerané dáta nejakou vhodnou funkciou. Proces, ktorým to dosiahneme, sa nazýva symbolická regresia. Donedávna bola táto schopnosť výhradne doménou ľudí, ale v posledných desaťročiach sa podarilo preniesť riešenie tohto problému pomocou viac či menej efektívnych algoritmov na počítače. Vedecké metódy a algoritmy schopné riešiť problém symbolickej regresie boli publikované v mnohých vedeckých časopisoch, knihách a štúdiách. Autori týchto algoritmov sa snažili v týchto svojich publikovaných prácach vysvetliť princípy algoritmov čo najnázornejšie, ale nie vždy sa im to podarilo. Mnohokrát bolo ich vysvetlenie príliš komplikované alebo vinou ich subjektívneho pohľadu nebolo podané dostatočne podrobné objasnenie daného problému, poprípade zložitosť problému neumožnila dokonalé vysvetlenie princípov riešenia tohto problému za pomoci konvenčných prostriedkov.

V tomto prípade sú potrebné rôzne, invenčné metódy vizualizácie, ktoré uľahčia čitateľom ich vedeckých prác jednoduchšie pochopiť podstatu problému a jeho riešenie. V súčasnosti je množstvo grafického softwaru, pomocou ktorého je možné vytvoriť animácie alebo obrázky, ktoré sú oveľa názornejšie a tým aj uľahčiť pochopenie daného problému.

Konkrétny problém predstavuje metóda symbolickej regresie, v našom prípade riešená algoritmom analytického programovania (Analytic Programming, AP), ktorý je dlhodobo rozvíjaný našou fakultou. Jeho princíp a fungovanie bolo opísané v knihách a vedeckých štúdiách, avšak niekomu môže pripadať vysvetlenie tvorcov tejto literatúry málo názorné.

Úlohou tejto bakalárskej práce je pomocou 2D a 3D animácii osvetliť fungovanie algoritmu, jeho hlavné princípy pri syntéze vhodného programu. Boli vybrané základné časti, z ktorých sa skladá celý priebeh algoritmu a tie boli animované pre lepšie pochopenie.

V práci sú najprv popísané jednotlivé metódy symbolickej regresie – genetické programovanie, gramatická evolúcia a analytické programovanie. V ďalšej časti sú popísané evolučné algoritmy, s ktorými pracuje algoritmus analytického programovania. Nasledujúca časť uvádza popis postupu vytvárania animácii vo vybraných grafických prostrediach.

I. TEORETICKÁ ČASŤ

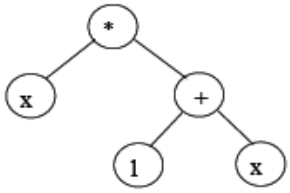
1 SYMBOLICKÁ REGRESIA

Termín symbolická regresia reprezentuje proces, v ktorom sú namerané dáta preložené vhodnou matematickou funkciou. Tento postup je použiteľný v prípade, že máme namerané dáta neznámej funkcie. Symbolická regresia ako metóda aproximácie dát bola dlhý čas doménou ľudí, ale v posledných desaťročiach sa stala doménou počítačov. Dnes sú známe viaceré metódy, v ktorých je možné použiť počítače k výpočtu symbolickej regresie. Prvou metódou je genetické programovanie (Genetic Programming, GP) a druhou je gramatická evolúcia (Grammatical Evolution, GE). Genetické programovanie je v podstate symbolická regresia používajúca evolučné algoritmy nahradzujúce dané schopnosti človeka. Gramatická evolúcia môže byť považovaná za vývoj genetického programovania pre ich niektoré spoločné princípy [5]. Okrem týchto dvoch metód existuje ešte algoritmus analytického programovania presadzovaný našou fakultou pod vedením doc. Ing. Ivana Zelinku Ph.D. [6].

1.1 Genetické programovanie

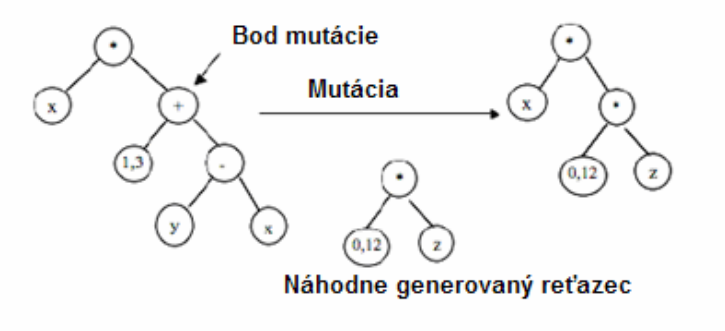
Genetické programovanie bolo predstavené koncom 80-tych rokov 20. storočia. Jeho princípom je, že nová generácia sa netvorí v numerickom, ale v analytickom tvare. To znamená, že po vytvorení nie sú výsledkom hodnoty parametrov, ale funkcie.

Podľa genetických algoritmov sa každá hodnota nazýva gén. V genetickom programovaní nie sú gény reprezentované celočíselnými alebo reálnymi hodnotami, ale parametrami v chromozómovom reťazci sú samotné funkcie. V najjednoduchšom prípade sú to premenné, konštanty, základné aritmetické funkcie a elementárne funkcie. Z tejto skupiny môže byť potom vytvorená funkcia. Pre ľahšie pochopenie je na obrázku znázornená funkcie zobrazená pomocou stromu. Vrchol stromu sa nazýva koreň. Počas vyhodnocovania funkcie sa postupuje smerom zospodu navrch.

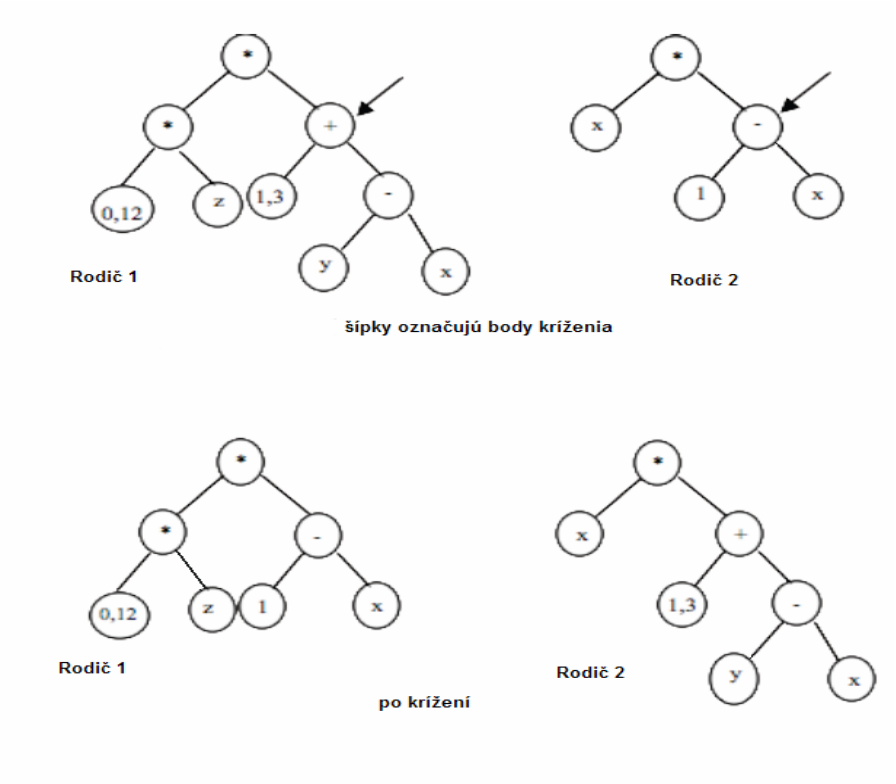


Obr. 1. Strom funkcie x(1+x) [4]

V genetickom programovaní sa používajú operátory kríženia a mutácie. V priebehu mutácie alebo kríženia sa menia celé časti stromu [4].



Obr. 2. Mutácia stromu v GP [4]



Obr. 3. Kríženie stromu v GP [4]

1.2 Gramatická evolúcia

Inou metódou symbolickej regresie je gramatická evolúcia. Jej výhodou v porovnaní s genetickým programovaním je, že gramatická evolúcia môže vyvíjať kompletné programy v ľubovoľnom programovacom jazyku pomocou binárneho reťazca variabilnej dĺžky.

Na mapovací proces využíva Backus Naur Form gramatickú definíciu. Mapovací proces je použitý k vytvoreniu výsledného programu v ktoromkoľvek jazyku použitím binárnych reťazcov k výberu pravidiel pre vytváranie v Backus Naur Form gramatickej definícii. Výsledkom je syntakticky správne vytvorený program z binárnych reťazcov, ktorý môže byť vyhodnotený pomocou účelovej funkcie.

Genómy – reťazce variabilnej dĺžky sú použité s kodónmi reprezentovanými celočíselnou hodnotou o dĺžke 8 bitov. Celočíselné hodnoty sú použité mapovacou funkciou k výberu vhodného pravidla pre vytvorenie z Backus Naur Form definície. Čísla vygenerované vždy reprezentujú jedno z pravidiel, ktoré môže byť použité. Backus Naur Form definícia sa skladá z terminálov a neterminálov, na základe ktorých sa vytvoria pravidlá. Mapovanie potom prebieha podľa rovnice:

Pravidlo = (celočíselná hodnota kodónu) Mod (počet možností pre daný neterminál)

Ak nie sú všetky neterminály nahradené mapovavacou funkciou za terminály a už nie je viac kodónov, potom sa cyklicky použijú kodóny zo začiatku [4].

1.3 Analytické programovanie

Algoritmus analytického programovania sa osvedčil pri riešení celej rady problémov symbolickej regresie ako sú: syntéza trigonometrických funkcii, polynomiálnych funkcii, funkcie boolovskej parity, funkcie boolovskej symetrie, riešenia diferenciálnych rovníc a optimalizácie cesty umelého mravca [6]. Základné vlastnosti analytického programovania, ktoré z neho činia algoritmus vhodný k tomuto účelu, sú popísané v ďalších podkapitolách.

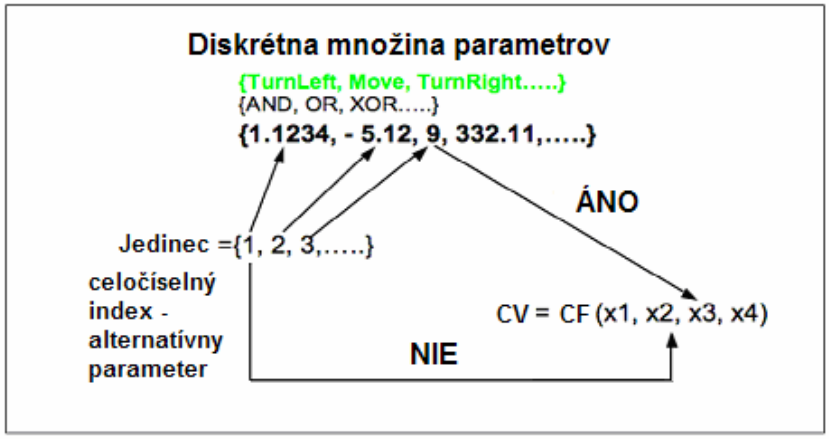
1.3.1 Základná myšlienka analytického programovania

Analytické programovanie bolo inšpirované numerickými metódami v Hilbertovom funkcionálnom priestore a tiež genetickým programovaním. Princíp analytického programovania je niekde medzi týmito dvoma filozofiami: Z genetického programovania preberá myšlienku evolučnej kreácie symbolických riešení, zatiaľ čo myšlienka funkcionálneho priestoru a budovania výslednej funkcie pomocou procesu prehľadávania je prevzatá

z Hilbertových priestorov. Rovnako ako aj genetické programovanie a gramatická evolúcia je aj analytické programovanie postavené na množine funkcii, operátorov a tzv. terminálov, ktoré sú väčšinou konštantami alebo nezávislými premennými, napríklad:

- Funkcie: Sin, Cos, Tan, Log, And, Or ...
- Operátory: +, -, *, /, dt ...
- Terminály: 3.14, 2.73, t ...

Tieto matematické objekty vytvárajú množinu, z ktorej sa analytické programovanie snaží syntetizovať vhodné riešenie. Základný princíp analytického programovania je založený na manipulácii s diskrétnymi množinami (Discrete Set Handling, DSH). Diskrétna množina vytvára rozhranie medzi evolučným algoritmom (Evolutionary Algorithm) a problémom. Vďaka tomuto môže byť v analytickom programovaní použitý takmer ľubovoľný evolučný algoritmus. Jednotlivec pozostáva z nenumerického vyjadrenia, ktoré je v evolučnom procese vyjadrené celočíselným indexom. Index slúži ako ukazateľ do množiny a analytické programovanie ho využíva k syntéze výslednej funkcie, programu [5].



Obr. 4. Manipulácia s diskrétnymi množinami [4]

1.3.2 Základná množina

Množinu všetkých dostupných matematických objektov – funkcii, operátorov a terminálov, nazývame základnou množinou (General Functional Set, GFS). Základná množina sa skladá z funkcii s rôznym počtom argumentov. Štruktúra základnej množiny je tvorená podmnožinami funkcii s rovnakým počtom argumentov. Obsah základnej množiny závisí na užívateľovi. Rôzne funkcie, operátory a terminály môžu byť spojené spolu do jednej množiny. Napríklad GFS_{all} je množinou všetkých funkcii, operátorov a terminálov,

GFS_{3arg} je množinou obsahujúcou funkcie s troma argumentmi a napríklad GFS_{0arg} je množina obsahujúca terminály [5], [6].

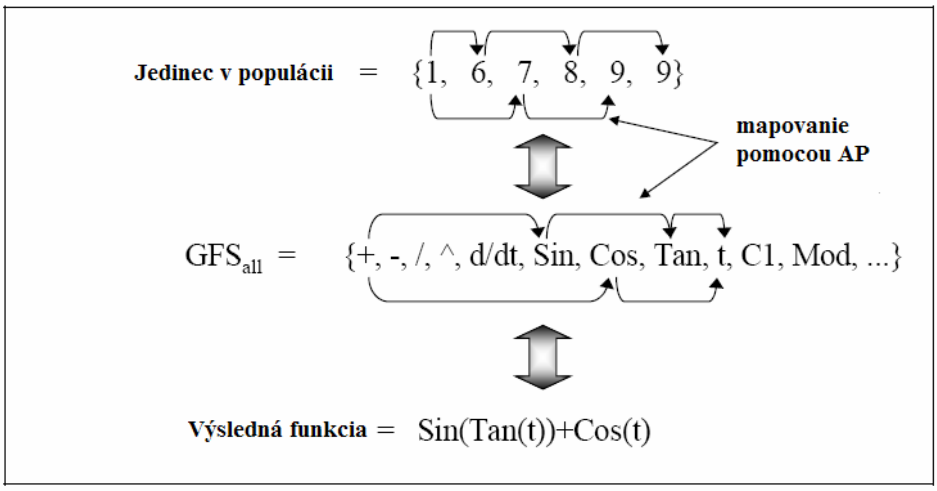
- GFS_{all} = { +, -, *, /, Log, Sin, Cos, Tan, t, C1, BetaRegulrized, … }
- GFS_{3arg} ={ BetaRegularized, ... }
- GFS_{2arg} ={ +, -, *, /, Log, ... }
- GFS_{1arg} ={ Sin, Cos, Tan, ... }
- GFS_{0arg} ={ t, C1, ... }

1.3.3 Operácie mapovania

Dôležitou časťou analytického programovania je sekvencia matematických operácii, ktoré sú použité pre programovú syntézu. Tieto operácie transformujú jedinca na použiteľný program (funkciu). Matematicky povedané, jedná sa o mapovanie z priestoru jedincov do priestoru programov. Takéto mapovanie sa skladá z dvoch častí, z Discrete Set Handling a bezpečnostných procedúr, aby sa predišlo generovaniu patologických jedincov [5].

Discrete Set Handling vytvorí celočíselný indexer, ktorý namapuje v podstate na ľubovoľné matematické objekty obsiahnuté v GFS na jedinca použitého v evolučnom algoritme. Tento indexer je v podstate vektorom ukazateľov na jednotlivé prvky v GFS. Evolučný algoritmus tak vníma jedinca ako vektor a iba pre potreby jeho ohodnotenia sa vykoná premapovanie na konkrétnu funkciu. Ktorá je použiteľná k výpočtu vhodnosti jedinca [6].

Analytické programovanie je sériou funkčných mapovaní, pritom je ale potrebné dodržať pravidlá, ktoré zaistia, že každý jedinec bude reprezentovať jednoznačnú a nedefektnú funkciu. Fungovanie takéhoto mapovania v praxi sa najlepšie ukáže na príklade:



Obr. 5. Mapovacia operácia v AP [5]

Jedinec z príkladu má na pozícii prvého parametru číslo 1, čo znamená, že je použitý operátor „+“ patriaci do GFS_{all}. Pretože operátor „+“ musí mať práve dva argumenty, indexy 6 a 7 sú určené ako jeho argumenty.

$$6+7 \tag{1}$$

Index 6 je nahradený „Sin“ a index 7 je nahradený „Cos“ patriacimi do GFS_{all}.

$$Sin+Cos \tag{2}$$

„Sin“ a „Cos“ sú funkcie s jedným argumentom. Nasledujúcimi indexami sú 8 a 9. Sú nahradené „Tan“ a „t“, ktoré sú použité ako argumenty funkcii „Sin“ a „Cos“.

$$Sin(Tan)+Cos(t) \tag{3}$$

Funkcia „Tan “ má jeden argument, teda ďalší index – 9 je argumentom funkcie „Tan“. „t“ nemá žiaden argument, čoho vyplýva, že vetva sa uzatvára. Argumentom funkcie „Tan“ je tiež „t“ a teda aj tu je vetva uzavretá [4]. Výsledná funkcia má tvar:

$$Sin(Tan(t))+Cos(t) \tag{4}$$

1.3.4 Evolučné operácie

V priebehu evolúcie danej populácie možných riešení sa používa rôznych evolučných operátorov ako sú napríklad kríženie alebo mutácia. Analytické programovanie je však zostavené tak robustne, že vo svojej štruktúre žiadne tieto operácie nepoužíva. Jednotlivé kroky

evolúcie – mutácia, kríženie, súťaženie, selekcia sú plne v kompetencii použitého evolučného algoritmu. Evolučný algoritmus tak medzi sebou premiešava jednotlivé prvky GFS [5], [6].

Z toho vyplýva, že úspešnosť pri hľadaní riešenia pomocou analytického programovania ovplyvňuje len voľba vhodnej GFS. Na strane evolučného algoritmu je potom množina užívateľom nastavených parametrov závislá na konkrétnom evolučnom algoritme. Jeden kľúčový argument však musia mať všetky evolučné algoritmy rovnaký. Týmto argumentom je účelová funkcia (Cost Function, Fitness Function, CF), ktorú je potrebné minimalizovať. Kľúčovou dvojicou parametrov analytického programovania je GFS a CF [6].

1.3.5 Posilnená evolúcia

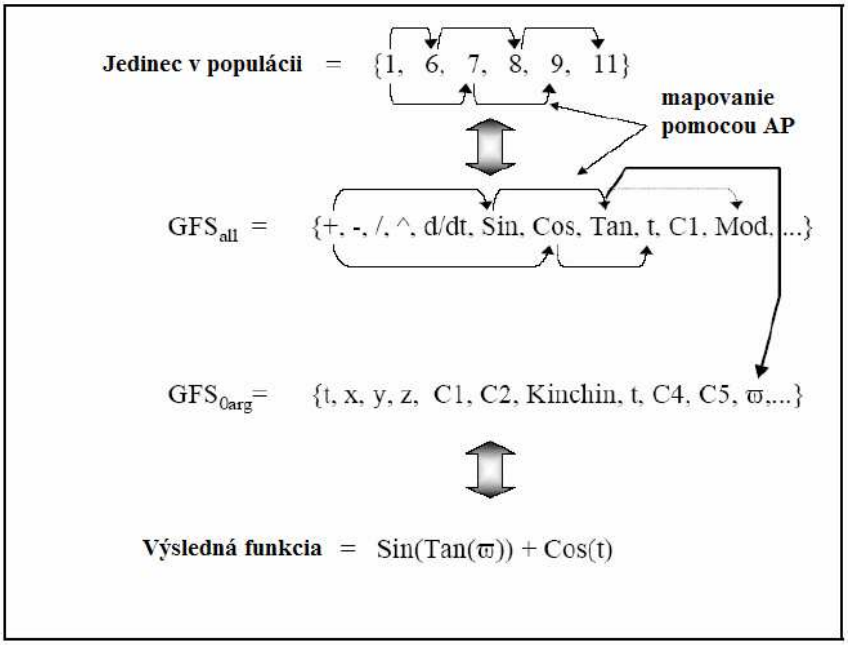
V priebehu evolúcie sú syntetizovaní viac alebo menej vhodní jedinci. Niektorých jedincov je možné použiť k posilneniu evolúcie, aby sme získali ešte lepšie riešenie. Hlavnou myšlienkou je pridanie už vytvoreného a čiastočne úspešného jedinca do množiny terminálov. Posilnenie je založené na užívateľom zvolenom prahu, ktorý bude používaný pri rozhodovaní, ktorý jedinec bude pridaný do množiny terminálov.

Napríklad, keď bude užívateľom zvolený prah 5 a všetci jedinci v populácii majú hodnotu účelovej funkcie vyššiu ako 5, evolúcia beží ďalej na základnej GFS. Avšak pokiaľ je najlepší jedinec v aktuálnej populácii menší ako 5, potom je kompletne pridaný do GFS a označený ako terminál. Od tejto chvíle pracuje analytické programovanie s rozšírenou GFS obsahujúcou čiastočne úspešné riešenia. Vďaka tomuto faktu je posilnená evolúcia schopná syntetizovať riešenie omnoho rýchlejšie ako analytické programovanie bez posilnenej evolúcie. Po pridaní programu do GFS dochádza k zníženiu hodnoty prahu na hodnotu účelovej funkcie daného jedinca. Ak je hodnota účelovej funkcie ďalšieho jedinca menšia ako nastavený prah, dochádza znova k jeho zníženiu a predtým pridaný jedinec je premazaný novým a úspešnejším [5].

1.3.6 Bezpečnostné procedúry

Aby sa zabránilo syntéze patologických jedincov, analytické programovanie obsahuje niekoľko bezpečnostných opatrení. Prvým pravidlom je, že GFS je rozdelená do podmnožín obsahujúcich funkcie s rovnakým počtom argumentov. Existencia týchto štruktúr je použitá v špeciálnych bezpečnostných podprogramoch, ktoré zisťujú ako ďaleko sa pri mapovaní

nachádza koniec jedinca a podľa toho volia vhodné podmnožiny GFS, aby tak predišli patologickej – neuzavretej funkcii. Ak je požadovaných viac parametrov ako je indexov v jedincovi, funkcia je nahradená inou funkciou s rovnakým indexom z podmnožiny s menším počtom argumentov [5]. Takýto prípad je zobrazený na nasledujúcom obrázku.



Obr. 6. Bezpečnostná procedúra AP [5]

Pri mapovaní jedinca je posledným indexom číslo 11, ktoré je nahradené funkciou "Mod". Tá však potrebuje dva parametre, aby dávala správny výsledok. Keďže už nie sú k dispozícii žiadne ďalšie indexy a došlo by k syntéze patologickej funkcie, funkcia "Mod" je nahradená inou, na pozícii 11 v podmnožine GFS_{0arg}.

S bezpečnosťou je potrebné počítať aj pri definícii účelovej funkcie a v prípade potreby definovať ďalšie bezpečnostné funkcie priamo ako jej súčasť.

1.3.7 Verzie analytického programovania

Existuje niekoľko verzii analytického programovania. V pokročilých verziách analytického programovania je upustené od definície konkrétnych konštánt ako terminálov (pokiaľ to nie je konkrétne vyžadované) a miesto nich sú definované obecné konštanty K_1, K_2, ... K_n, ktorých hodnota je určená pomocou nelineárneho prekladania (non-linear fitting) až tesne pred

hodnotením účelovej funkcie. Tento postup ďalším zefektívnením algoritmu analytického programovania [5], [6].

2 EVOLUČNÉ ALGORITMY

Evolučné algoritmy slúžia ako základ pre analytické programovanie. Pokiaľ sa nepoužije kvalitne nastavený evolučný algoritmus, nebude ani analytické programovanie pracovať správne.

2.1 SOMA

Samo - Organizujúci sa Migračný Algoritmus (Self-Organizing Migration Algorithm) je algoritmus existujúci od roku 1999, ktorého činnosť je založená na geometrických princípoch. Vzhľadom k tomu, že pracuje s populáciami podobne ako genetické algoritmy a výsledok je po jednom evolučnom cykle (migračnom kole) totožný s genetickými algoritmami či diferenciálnou evolúciou, možno ho radiť medzi evolučné algoritmy, aj keď nevyvára počas jeho behu potomkov.

Pôvodná myšlienka, ktorá viedla k jeho vytvoreniu, spočíva v napodobnení chovania skupiny inteligentných jedincov, ktorí spolupracujú pri riešení spoločného problému ako je napríklad hľadanie zdroja potravy. SOMA sa svojou robustnosťou v zmysle nájdenia globálneho extrému vyrovná algoritmom ako je napríklad diferenciálna evolúcia.

Tento algoritmus, ktorý pracuje ako ostatné evolučné algoritmy s populáciou jedincov, bol vyvinutý na princípoch, ktoré je možno odpozorovať v prírode, a ktorými sa v sociálno-biologickom prostredí riadia inteligentní jedinci, ktorí kooperujú na riešení určitého problému. Filozofia algoritmu je založená na kooperatívnom prehľadávaní (migrácii) priestoru možných riešení daného problému.

Vzhľadom k tomu, že hlavná myšlienka algoritmu SOMA nie je založená na princípe evolúcie ako takej, ale je založená na princípe skupiny, nie je klasifikovaný ako algoritmus evolučný ale tzv. memetický.

Vlastnosť samoorganizácie u SOMA algoritmu plynie z faktu, že sa jedinci ovplyvňujú navzájom behom hľadania lepšieho riešenia. Mnohokrát to vedie k tomu, že v priestore možných riešení vznikajú skupiny jedincov, ktoré sa rozpadajú alebo spojujú a putujú cez prehľadávaný priestor. Skupina jedincov (populácia) teda organizuje vzájomný pohyb jedincov [3].

2.1.1 Parametre algoritmu

Beh algoritmu SOMA je ovplyvňovaný špeciálnou množinou parametrov, rozdelených na dva druhy: riadiace a ukončovacie. Riadiace parametre majú vplyv na kvalitu behu algoritmu a ukončovacie parametre za dopredu nadefinovaných podmienok ukončujú beh algoritmu.

Parameter Mass určuje ako ďaleko sa aktívny jedinec zastaví od vedúceho jedinca. Parameter Step určuje „zrnitosť“ s akou bude mapovaná cesta aktívneho jedinca. Parameter PRT znamená perturbáciu. Podľa tohto parametru sa tvorí perturbační vektor (PRTVector), ktorý ovplyvňuje, či sa aktívny jedinec bude pohybovať priamo k vedúcemu jedincovi alebo nie. Parameter D udáva počet argumentov účelovej funkcie. Parameter NP určuje počet jedincov v populácii. Parameter Migrácia udáva koľkokrát sa populácia preorganizuje. Parameter AcceptedError je ukončovacím parametrom a definuje aký maximálny rozdiel medzi najhorším a najlepším jedincom v aktuálnej populácii je povolený [3].

2.1.2 Populácia

Typickým znakom evolučných algoritmov je, že sú založené na práci s populáciami jedincov. Toto platí aj pre algoritmus SOMA. Každý jedinec predstavuje aktuálne riešenie daného problému. V podstate sa jedná o množinu argumentov účelovej funkcie, ktorej optimálna číselná kombinácia sa hľadá. S každým jedincom je taktiež spojená hodnota účelovej funkcie, ktorá určuje ako je daný jedinec vhodný pre ďalší vývoj populácie. Táto hodnota sa neúčastní samotného evolučného procesu, ale nesie iba informáciu o kvalite jedinca.

Zmyslom evolučných algoritmov je cyklické vytváranie nových populácii a náhrada starých pomocou novo vytvorenej na základe presne definovaných matematických pravidiel. To aké pravidlá sa sú definované, závisí na evolučnom algoritme.

Tvorba populácie je náhodné rozloženie jedincov v priestore možných riešení. Pri behu evolučného algoritmu dochádza k zhromažďovaniu jedincov okolo jedného (obvykle globálneho extrému) alebo viacerých extrémov. Vývoj populácie musí byť vždy konvergentný k lepším hodnotám, čo znamená, že nemôže nikdy vykazovať divergenciu. Do novej populácie sa prepúšťajú len riešenia, ktoré sú lepšie alebo rovnako dobré ako tie zo starej populácie [3].

2.1.3 Mutácia

V evolučných algoritmoch hrá životne dôležitú úlohu mutácia. Je to proces, pri ktorom dochádza k náhodnej zmene niektorých vlastností - parametrov daného jedinca. Mutačné procedúry využívajú generátor náhodných čísel. V prípade algoritmu SOMA je mutácia nazývaná pojmom perturbácia. Pri pohybe jedincov priestorom možných riešení je ich pohyb perturbovaný a nie mutovaný [3].

2.1.4 Kríženie

U evolučných algoritmov znamená pojem kríženie tvorbu nového potomka alebo vygenerovanie novej pozície na danej hyperploche pomocou dvoch existujúcich pozícii (rodičov). SOMA je v tomto prípade odlišný. Pri putovaní aktuálneho jedinca po hyperploche dochádza k mapovaniu tohto priestoru po diskrétnych skokoch, čo sa dá považovať ako generovanie sekvencie potomkov, z ktorých prežije len ten najlepší. Pri tomto pohybe si každý jedinec pamätá súradnice pozície, na ktorej našiel najlepšie riešenie v rámci svojej cesty a toto riešenie postupuje do ďalších migračných kôl [3].

2.1.5 Princíp algoritmu

Tab. 1. Význam biologickej terminológie v algoritme SOMA

Biologická realita	Počítačová implementácia
členovia svorky, spoločenstvo	jedinci v populácii, parameter NP
člen spoločenstva s najlepším zdrojom potravy	Leader, vedúci aktuálneho migračného kola
potrava	vhodnosť, hodnota účelovej funkcie, geometricky je to lokálny alebo globálny extrém na N – rozmernej hyperploche
životný priestor spoločenstva	hyperplocha daná účelovou funkciou
migrácia členov spoločenstva	migračné kolá v algoritme SOMA

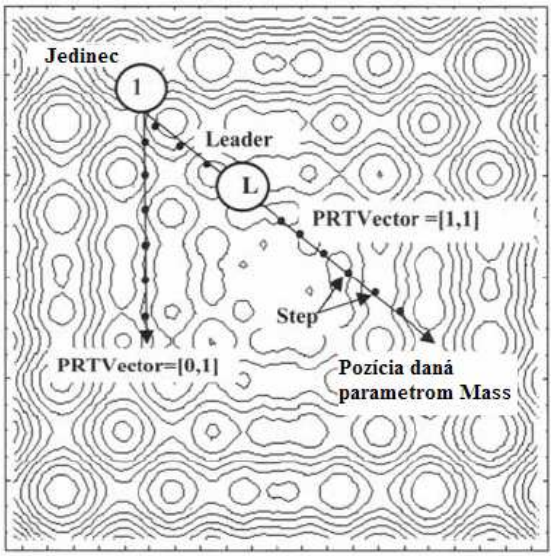
SOMA pracuje v cykloch nazývaných migračné kolá. Počas migračných kôl nie sú tvorení noví jedinci, sú iba premiestňovaní do finálnych pozícii pomocou sekvencie pozícii vypočítaných vzhľadom k pozícii Leadera – migrujú naprieč priestorom možných riešení. Vzhľadom k tomu, že je SOMA založená na súťaživo-kooperačnom princípe, sú jeho va-

rianty nazývané stratégiami. Základná verzia algoritmu SOMA AllToOne sa skladá z nasledujúcich krokov:

Definícia parametrov – Pred štartom SOMA je nutné nadefinovať riadiace a ukončovacie parametre. Taktiež je nutné nadefinovať účelovú funkciu, ktorá bude optimalizovaná. Táto poslúži pre budúcich jedincov ako akési životné prostredie. Užívateľom daná účelová funkcia by mala vracať skalár, ktorý bude použitý ako miera kvality daného jedinca.

Tvorba populácie – V tomto kroku je vytvorená počiatočná populácia pomocou generátora náhodných čísel. Pomocou Specimenu (vzor, podľa ktorého bude generovaná počiatočná populácia) a generátora je pre každý parameter jedinca generované náhodné číslo.

Migračné kolá – Každý jedinec je ohodnotený účelovou funkciou a je zvolený Leader (jedinec s najlepšou hodnotou účelovej funkcie) pre nasledujúce migračné kolo. V tomto okamihu sa začnú ostatní jedinci pohybovať smerom k Leaderovi pomocou skokov, ktorých veľkosť je daná parametrom Step. Po každom skoku si každý jedinec na takto získanej novej pozícii prepočíta svoju hodnotu účelovej funkcie a pokiaľ je lepšia, tak si ju zapamätá. Pohyb jedinca k Leaderovi po skokoch pokračuje tak dlho, pokiaľ nie je dosiahnuté pozície danej parametrom Mass. Po ukončení behu sa jedinec vracia na pozíciu, kde bola nájdená najlepšia hodnota účelovej funkcie počas jeho cesty. Pred tým ako sa začne jedinec pohybovať smerom k Leaderovi, je vygenerovaný prázdny vektor PRTVector o rozmere rovnajúcemu sa hodnote parametru D včetne vygenerovanej sekvencie náhodných čísel, ktorých počet je rovný D. Tie sú porovnané s parametrom PRT. Ak je n-té vygenerované číslo väčšie ako PRT parameter, potom je n-tý parameter PRTVectoru nastavený na 0, v opačnom prípade na 1. Parametre jedinca, ktoré sú nastavené na 0 sa neprepočítavajú , sú zmrazené a znižuje sa počet stupňov voľnosti pohybu jedinca. Tento proces nahrádza mutáciu známu u iných evolučných algoritmov a zvyšuje robustnosť algoritmu pri hľadaní globálneho extrému.



Obr. 7. PRTVector [3]

Testovanie ukončovacích parametrov – V tomto kroku je kontrolované, či je rozdiel medzi Leaderom a najhorším jedincom menší ako parameter AcceptedError. Taktiež je kontrolovaná zhodnosť počtu migračných kôl s parametrom Migrácia.

Stop – Návrat najlepšieho nájdeného jedinca po poslednom migračnom kole [3].

2.1.6 Stratégie SOMA algoritmu

„Všetci k jednému“ (AllToOne). Táto stratégia bola popísaná v predchádzajúcej sekcii. Všetci jedinci v populácii migrujú k Leaderovi.

„Všetci ku všetkým“ (AllToAll). V tejto stratégii neexistuje Leader. Všetci jedinci migrujú ku všetkým ostatným tak ako vo verzii „Všetci k jednému“ s tým rozdielom, že po dokončení migrácii aktuálneho jedinca, sa daný jedinec vracia na pozíciu, kde bol nájdený najväčší extrém počas jeho NP-1 migračných ciest vykonaných v jednom migračnom kole. Táto stratégia je výpočetne náročnejšia, ale je vyššia pravdepodobnosť, že bude nájdený globálny extrém.

„Adaptívne všetci ku všetkým“ (AllToAllAdaptive). Táto stratégia je totožná so stratégiou „Všetci ku všetkým“ s rozdielom, že aktuálne migrujúci jedinec sa nepresúva do novej pozície až po všetkých migráciách ku všetkým ostatným, ale po každej, aktuálne dokončenej migrácii ku každému z NP-1 jedincov sa presunie na najlepšiu pozíciu nájdenú na tejto aktuálnej migrácii.

„Všetci k jednému náhodne“ (AllToOneRand). Pri tejto stratégii sa všetci jedinci pohybujú k jednému Leaderovi, ktorý však nie je určený najhlbšou pozíciou na hyperploche, ale je pre každého jedinca náhodne určený z populácie.

„Zväzky“ (Clusters). SOMA s vytváraním zväzkov je úprava, ktorá sa dá použiť na ktorúkoľvek predchádzajúcu stratégiu. Jedinci zúčastňujúci sa migračného procesu sú rozdelení do zväzkov. V každom tomto zväzku potom prebieha samostatný SOMA. Vzhľadom na to, že sa jedinci pohybujú, môžu sa zväzky spájať a rozpadať. Pokiaľ sa stane, že nejaký jedinec je príliš ďaleko od ostatných a teda nemôže byť zahrnutý do už existujúcich zväzkov, je zväzkom sám o sebe a migruje ku všetkým ostatným ako v stratégii „Všetci ku všetkým“ [3].

2.2 Diferenciálna evolúcia

Diferenciálna evolúcia je pomerne nový typ evolučného algoritmu uvedený v roku 1995 Kenom Priceom a Rainerom Stornom. Jeho schéma je dosť podobné genetickým algoritmom, s ktorými má niekoľko podobných rysov ako je tvorba potomkov, používanie generácii a pod. [3].

2.2.1 Parametre algoritmu

Činnosť a kvalita diferenciálnej evolúcie je ako u ostatných evolučných algoritmov ovplyvnená riadiacimi parametrami.

Parametrom CR sa nastavuje prah kríženia. Parameter NP nastavuje veľkosť populácie. Parameter F označuje mutačnú konštantu. Parametrom D sa nastavuje počet argumentov účelovej funkcie [3].

2.2.2 Populácia

Diferenciálna evolúcia pracuje s populáciami rovnako ako algoritmus SOMA. Detailný popis v sa nachádza v podkapitole Populácia [3].

2.2.3 Mutácia

Diferenciálna evolúcia sa líši od ostatných algoritmov tým, že k vytvoreniu ďalšieho potomka je potrebné štyroch rodičov. Pre každého jedinca sú náhodne vybraní traja ďalší je-

dinci z populácie. Pomocou týchto troch jedincov sa potom vytvorí šumový vektor „v“, ktorý predstavuje mutáciu kombinácie troch náhodne vybraných rodičov. Mutácia je vykonaná tak, že rozdiel dvoch náhodne vybraných rodičov je vynásobený mutačnou konštantou F a výsledný vektor sa pričíta ku zostávajúcemu tretiemu [3].

2.2.4 Kríženie

V diferenciálnej evolúcii nastáva kríženie až po mutačnom procese. Proces kríženia vytvorí zo štvrtého, zatiaľ nepoužitého, rodiča a šumového vektoru skúšobný vektor. Ten sa vytvára za pomoci parametra CR (prah kríženia) tak, že sa v cykle vyberajú korešpondujúce parametre zo štvrtého a šumového jedinca a pre každú túto vybranú dvojicu je generované náhodné číslo. Ak je menšie ako CR, do príslušného parametru v skúšobnom vektore sa presunie parameter z jedinca šumového a v opačnom prípade z vektoru cieľového.

Týmto spôsobom sa vytvorí nový jedinec, ktorý potom súťaží o miesto v novej populácii s aktuálnym – štvrtým jedincom z populácie starej [3].

2.2.5 Princíp algoritmu

Cieľom diferenciálnej evolúcie je v cykloch nazývaných „generácie“, vyšľachtiť populáciu jedincov s čo najlepšími hodnotami účelovej funkcie spojenej s každým z nich.

Definícia parametrov – Je nutné nadefinovať množinu parametrov a prototyp jedinca (Specimen).

Tvorba populácie – Populácia sa tvorí vygenerovaním množiny jedincov podľa prototypového (Specimen) vektoru.

Začiatok cyklu generácie – Počas každej generácie sa vykonáva ešte cyklus, ktorý zabezpečuje postupné evolučné šľachtenie každého jedinca z populácie. V tomto cykle sa postupne vyberajú jedinci za sebou až do konca populácie a pre každého z nich je vykonaný následný evolučný cyklus.

Evolučný cyklus – V tomto cykle dochádza k mutácii a kríženiu ako už bolo písané vyššie v časti Mutácia a v časti Kríženie. Hodnota účelovej funkcie skúšobného vektoru sa porovná s hodnotou funkcie cieľového vektoru. Na pozíciu cieľového vektoru v novej populácii sa vyberá ten vektor (jedinec), ktorý má menšiu hodnotu účelovej funkcie.

Testovanie splnenia ukončovacích parametrov – Diferenciálna evolúcia je ukončená len za podmienky, že bol vykonaný užívateľom zadaný počet generácii.

Vyhodnotenie – Celý proces generácii sa opakuje, pokiaľ nie je splnený zadaný počet generácii. Počas každej generácie sa uchováva hodnota účelovej funkcie najlepšieho jedinca do vektoru histórie, ktorý po ukončení znázorňuje priebeh evolučného procesu [3].

Nevýhodou algoritmu diferenciálnej evolúcie je jav stagnácie, pri ktorom dochádza k zastaveniu vývoja hodnoty účelovej funkcie smerom k nižším hodnotám ešte pred dosiahnutím globálneho extrému.

2.3 Predstierané žíhanie (Simulated Annealing)

Tento algoritmus patrí medzi staršie v porovnaní s algoritmom SOMA. Slovo žíhanie bolo prevzaté z odvetvia metalurgie ako analógia k žíhaniu oceli (proces zahrievania a ochladzovania materiálu za účelom získania lepších fyzikálnych vlastností). Tento princíp využíva algoritmus predstieraného žíhania v zmysle hľadania globálneho extrému.

Algoritmus začína vygenerovaním náhodnej pozície. Následne je vhodne vygenerovaná pozícia v okolí štartovacieho bodu. V tomto bode je vyčíslená účelová funkcia. Ak je jej hodnota lepšia ako hodnota účelovej funkcie v predchádzajúcom bode (rozdiel účelových funkcii je záporný), je táto účelová funkcia uložená. Ak je účelová funkcia horšia ako v predchádzajúcom kroku, je vygenerované číslo z intervalu <0,1>. Ak je toto číslo menšie ako hodnota pravdepodobnosti *p(T)*, potom je toto číslo akceptované ako nový bod, v opačnom prípade sa pokračuje v starom bode.

$$p(T) = e^{-\frac{\Delta E}{T}} \tag{5}$$

p(T) - pravdepodobnosť zmeny teploty

ΔE - rozdiel účelových funkcii v predchádzajúcom a aktuálnom bode

T - aktuálna teplota

Algoritmus začína s nastavenou teplotou, ktorá sa s postupným pribúdaním krokov znižuje [8], [4].

2.4 Genetické algoritmy

Genetické algoritmy sú skupinou metód vhodných na riešenie optimalizačných a vyhľadávacích problémov. Sú založené na prírodných princípoch evolúcie, popísanej Charlesom Darwinom. Genetické algoritmy pracujú s populáciou jedincov. Každý jedinec sa skladá z binárnych parametrov nazývaných gény. Všetky gény, ktoré patria jednému jedincovi, vytvárajú reťazec nazývaný chromozóm. V genetike je množina parametrov jednotlivých chromozómov uvádzaná ako genotyp. Genotyp obsahuje informácie potrebné k vytvoreniu organizmu uvádzané ako fenotyp. Rovnaké pojmy sú použité aj v genetických algoritmoch. Chromozóm je genotyp a jeho hodnota účelovej funkcie je fenotyp. Táto hodnota určuje úspešnosť riešenia.

Evolúcia začína z populácie náhodne vygenerovaných jedincov stanúcich sa generáciou. V každej generácii je vyčíslená hodnota účelovej funkcie každého jedinca. Rôzni jedinci sú vybraní na základe ich hodnoty účelovej funkcie z aktuálnej populácie a mutovaní. Pomocou týchto jedincov je vytvorená nová populácia. Nová populácia je potom použitá v ďalšom cykle algoritmu. Algoritmus je ukončený, ak je dosiahnutý počet generácii alebo bola dosiahnutá optimálna hodnota účelovej funkcie. Ak je algoritmus ukončený po dosiahnutí maximálneho počtu generácii, vhodné riešenie nemuselo byt nájdené [7], [4].

II. PRAKTICKÁ ČÁST

3 GRAFICKÉ APLIKÁCIE

Ako grafický software sme sa rozhodli použiť nekomerčné, voľne dostupné grafické aplikácie. Animácie boli vytvorené vo freewarovom programe Blender (verzia 2.45). Pomocou programu PhotoFiltre boli upravované textúry, ktoré boli použité v animácii alebo bol tento program použitý na tvorbu obrázkov použitých v animáciách.

Nie vždy bolo však vhodné vizualizovať dané procesy v 3D priestore. V niektorých prípadoch bolo potrebné zobraziť priebehy algoritmov v textovej podobe. Aj tieto textové 2D animácii boli vytvorené v programe Blender. Postup pri vytváraní animícii je uvedený v nasledujúcich podkapitolách.

3.1 Blender

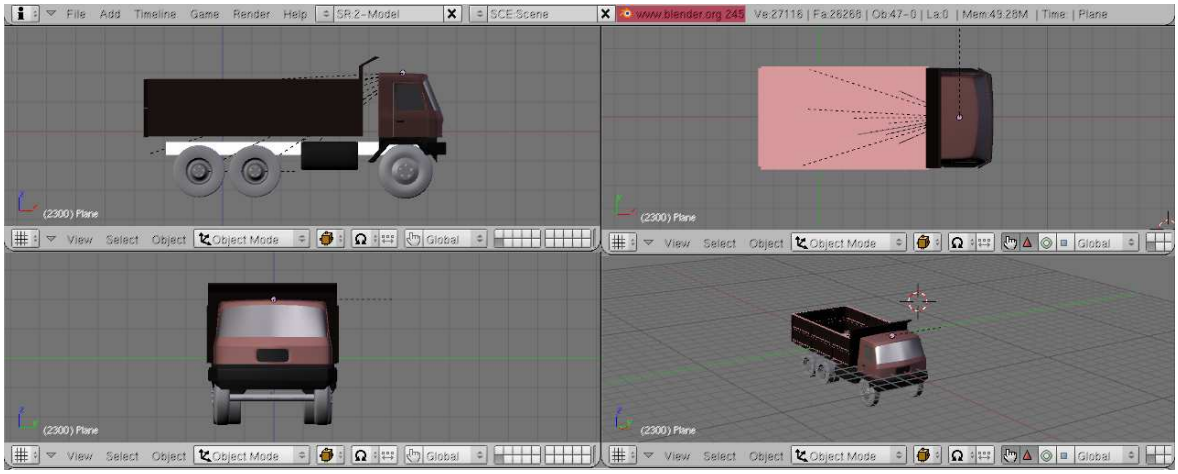
Blender je open source, multiplatformný software určený k tvorbe 3D grafiky a hier.

Obr. 8. Logo Blender

Obsahuje veľa implementovaných nástrojov určených k tvorbe animácii. V prípade, že je potrebná určitá funkcia, ktorú software neponúka, je možné si ju doprogramovať pomocou Python skriptovacieho jazyka a tieto skripty potom používať počas animácie alebo pri modelovaní.

3.1.1 Tvorba scény

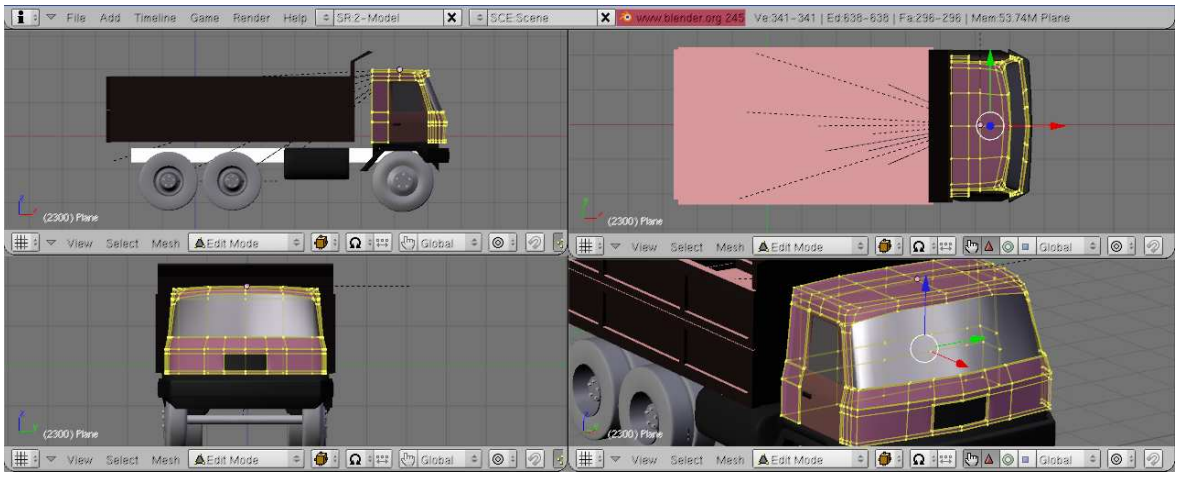
Vytváranie scény a modelovanie prebiehalo v 3D okne.



Obr. 9. 3D okno – pohľad spredu, zboku, zhora a pohľad z kamery

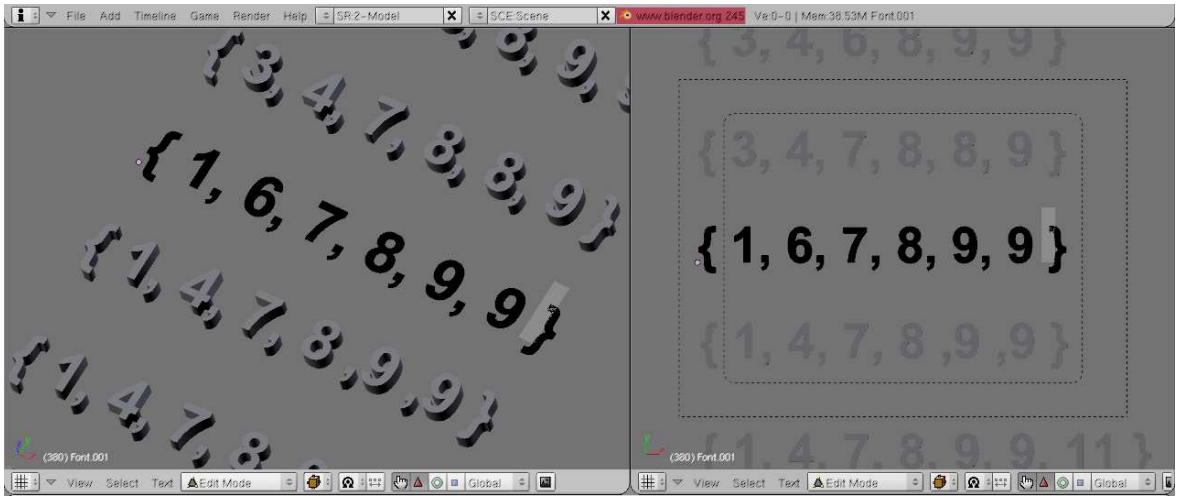
V 3D okne je možné nastaviť pohľad spredu, zboku , zhora alebo pohľad z kamery, alebo je možné vybrať si akýkoľvek iný pohľad v priestore pomocou myši.

Samotné modelovanie prebiehalo v editovacom móde, kde sa z vloženého primitíva (doska alebo kocka) pomocou riadiacich bodov – vertexov vytvárala konečná podoba objektu. Vertexy je možné pridávať, meniť ich polohu, aby sa nami tvorený model podobal čo najviac žiadanému tvaru.



Obr. 10. Objekt v editovacom móde

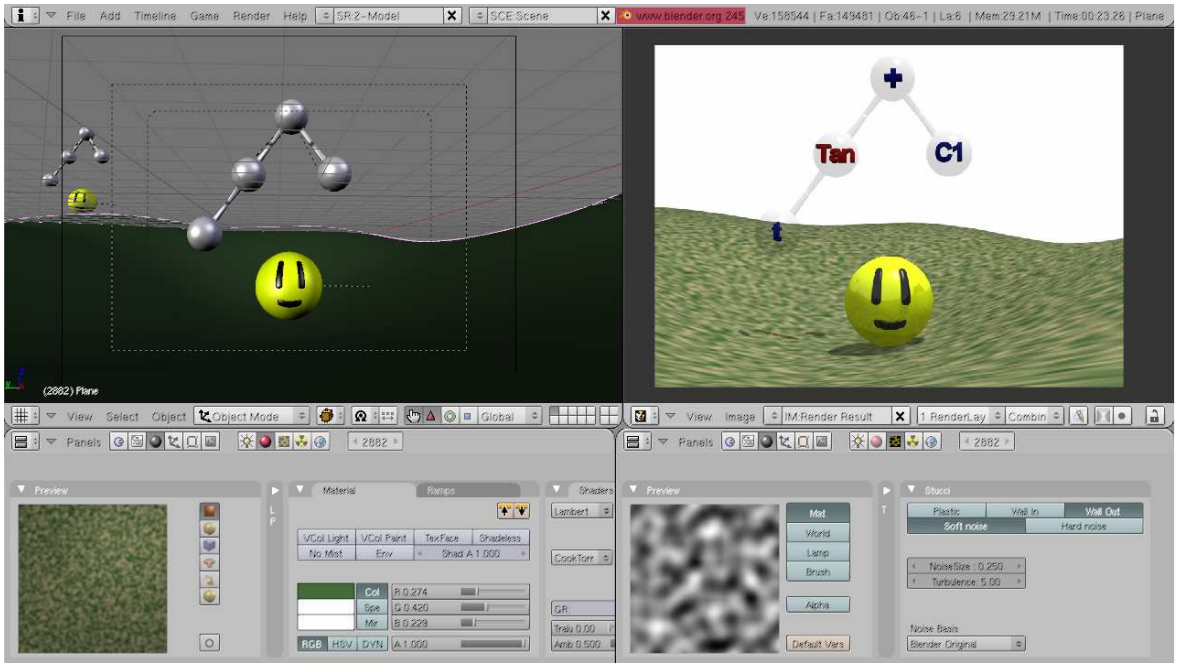
Text bol vytváraný odlišným spôsobom. Text vložený do 3D okna nemal charakter mesh, čiže sa neskladal z vertexov, ale bol to objekt editovateľný pomocou špeciálnych modifikátorov určených len pre prácu s textom. V priestore bol zobrazený ako 3D objekt avšak špeciálnym nastaveným snímania kamery bolo dosiahnuté len 2D vnímanie potrebné do animácie.



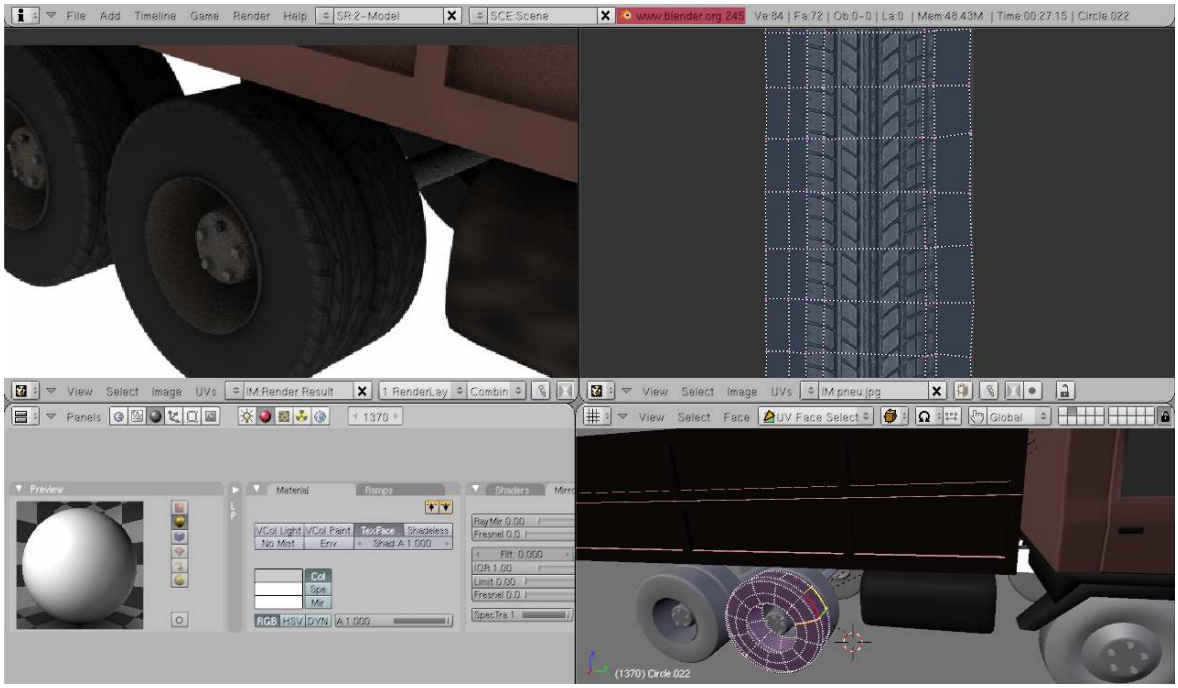
Obr. 11. Text v editovacom móde – vľavo, ortografické nastavenie kamery - vpravo

3.1.2 Materiály a textúry

Dôležitou vlastnosťou objektov v scéne sú ich materiály. Po vymodelovaní objektu bola naň namapovaná textúra. V niektorých prípadoch bola mapovaná procedurálna textúra (textúra založená na matematickej funkcii vstavaná do Blenderu). V iných prípadoch bola textúra vytvorená v 2D editore a namapovaná pomocou UV Image Editoru. V takomto prípade bolo potrebné objekt (mesh) „odbaliť“ a na jeho jednotlivé plochy ručne namapovať textúru.

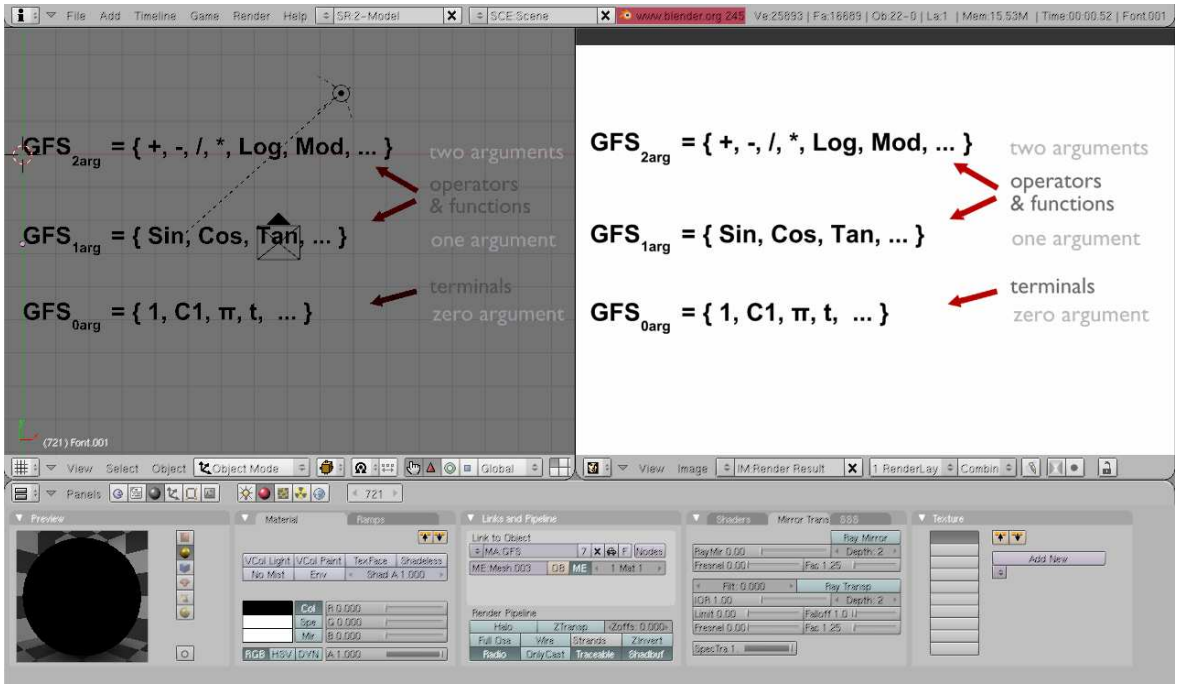


Obr. 12. Materiál s procedurálnou textúrou



Obr. 13. Materiál s textúrou nampovanou v UV Image Editore

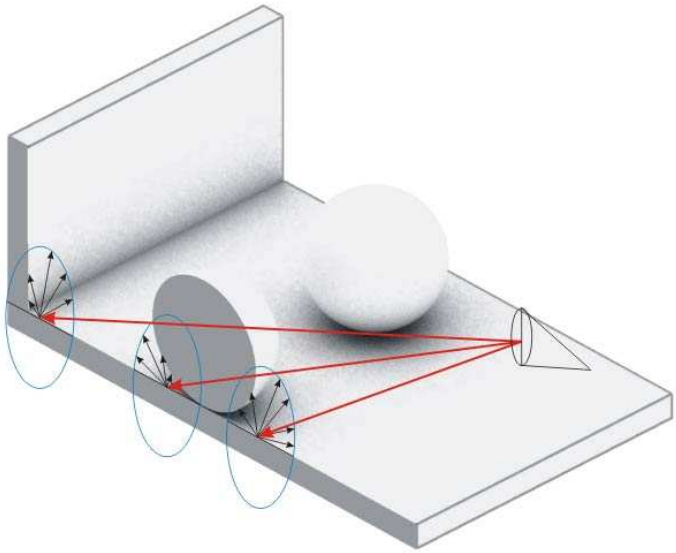
Pri animácii textu neboli potrebné textúry. Bol použitý základný materiál rôznych farieb bez lesklosti alebo reflexivity.



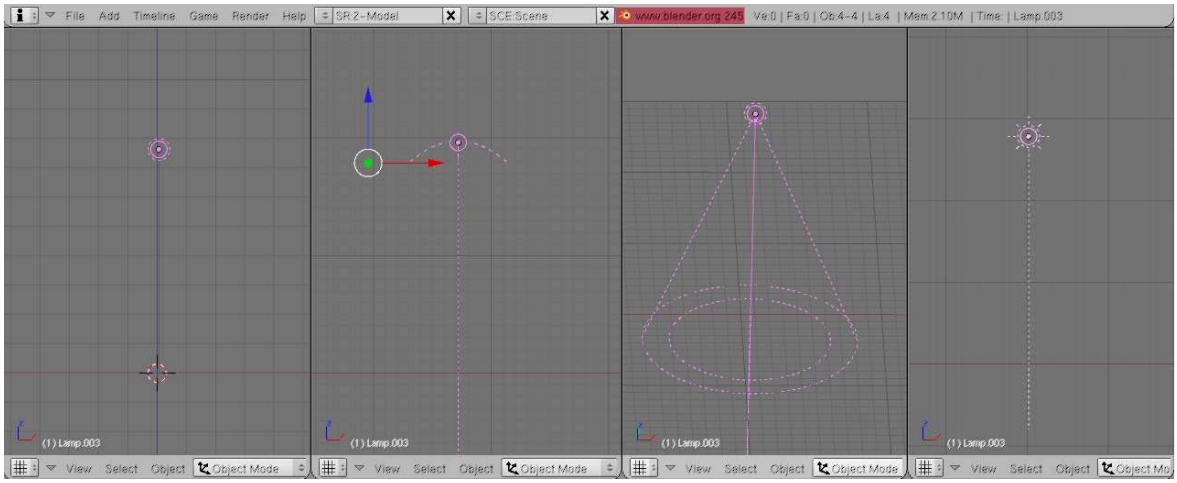
Obr. 14. Materiál bez textúry

3.1.3 Osvetlenie

Medzi základné objekty v scéne patria zdroje osvetlenia. Bez zdrojov osvetlenia by nebolo možné dosiahnuť očakávané výsledky. Pri animovaní textu boli použité jednoduché osvetlenia bez ray tracingu (RT), ktoré nevrhali tiene. Pri priestorových animáciách boli tiene potrebné pre dosiahnutie dojmu priestorovosti. Boli použité jednoduché ray tracingové lampy simulujúce svetlo zo žiarovky vychádzajúce z jedného malého bodu a rozchádzajúce sa do priestoru, bodové svetlá simulujúce svetelný kužeľ reflektoru a svetlo simulujúce slnko s rovnobežnými lúčmi. Problémom pri tomto druhu osvetlenia je, že svetlo vychádzajúce len z jedného zdroja svetla nie je schopné dostatočne osvetliť scénu. V reálnom svete sa svetlo od rôznych povrchov odráža a tým nepriamo osvetľuje povrchy, ktoré nie sú priamo osvetlené z tohto zdroja svetla. Tento efekt je možné dosiahnuť špeciálnym typom osvetlenia, ktoré je emitované z prostredia a nie len z jedného zdroja svetla. Tento typ osvetlenia sa nazýva Ambient Occlusion a ide o nepriame osvetľovanie scény. Z každého viditeľného pixelu sa vyšle zväzok lúčov. Zmeria sa celková dĺžka lúčov a na základe tejto dĺžky sa určí čiernobiely odtieň pixelu pre ktorý sa tento krok počítal. Lúče, ktoré sú dlhšie ako predpísaná dĺžka sa nezapočítavajú do výpočtu.



Obr. 15. Nepriame osvetľovanie – Ambient Occlusion [14]

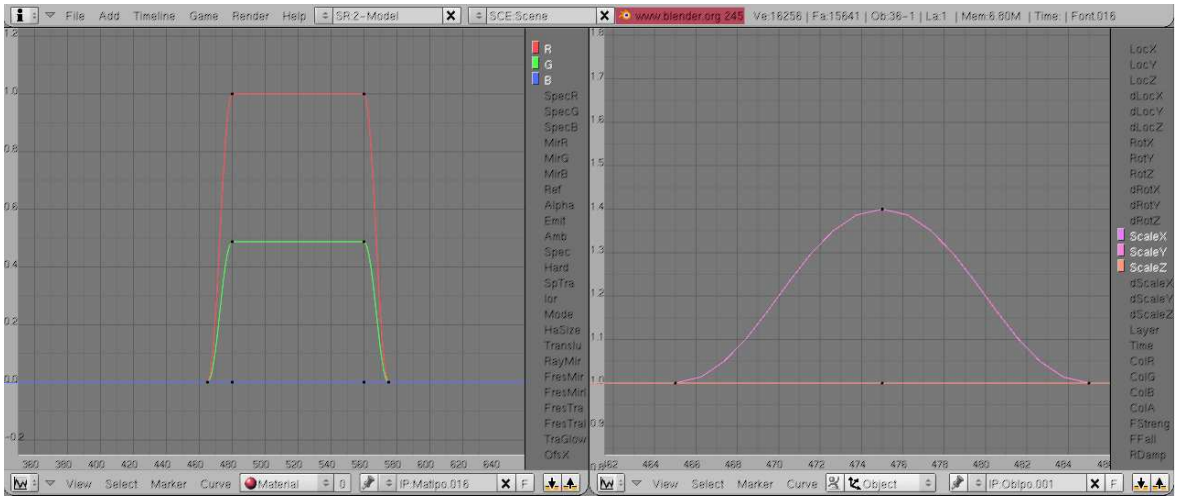


Obr. 16. Zdroje svetla – žiarovka, svetlo bez RT, reflektor, slnko

3.1.4 Animácia

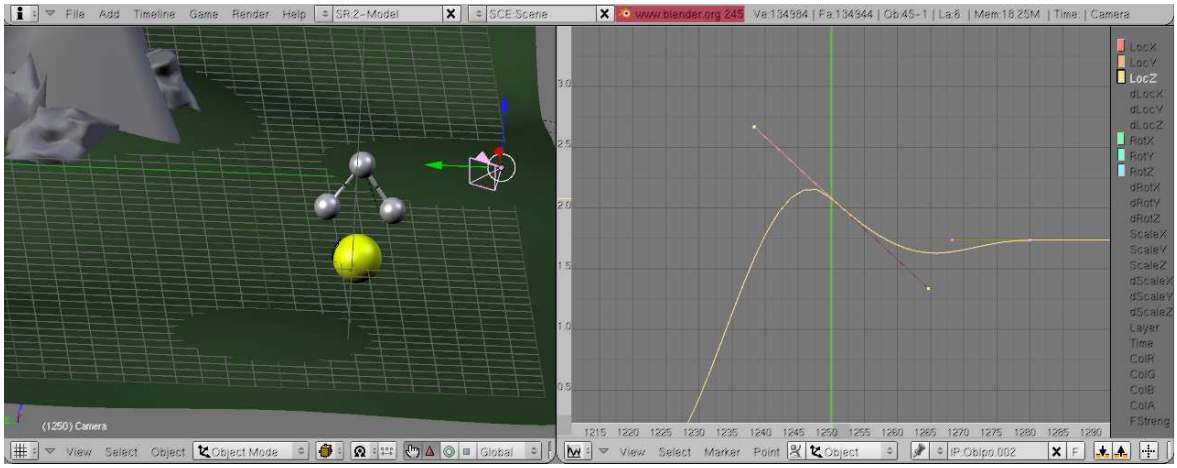
Podstatou animácie je ukladanie daných nastavení pre objekty alebo materiály pomocou animačných kľúčov pre jednotlivé snímky animácie . Animované objekty boli vytvorené pomocou týchto animačných kľúčov. Blender umožňuje vkladať rôzne animačné kľúče pre objekty, materiály, textúry, kameru alebo tvary objektov. Pre pohyb objektu slúži kľúč „Loc“. Tento kľúč je potrebné vložiť pre objekt na začiatku pohybu pre prvú snímku, potom sa nastaví konečná snímka kedy bude pohyb ukončený a definuje sa miesto, kde bude pohyb ukončený. Následne sa vkladá ďalší kľúč „Loc“. Týmto spôsobom bol naprogramovaný pohyb počas daných snímok. Pre ďalší pohyb sa tento postup opakuje. Rovnakým spôsobom sa vkladajú aj kľúče pre rotáciu – „Rot“ alebo zväčšenie – „Scale“.

Podobný princíp platí aj pri animácii materiálu. Prvý kľúč sa nastavuje v počiatočnej snímke pre dané nastavenie materiálu, nasleduje presun na konečnú snímku a zmena želaného parametru nastavenia materiálu alebo textúry. Tieto hodnoty sa zapíšu pomocou vloženia ďalšieho príslušného animačného kľúča.

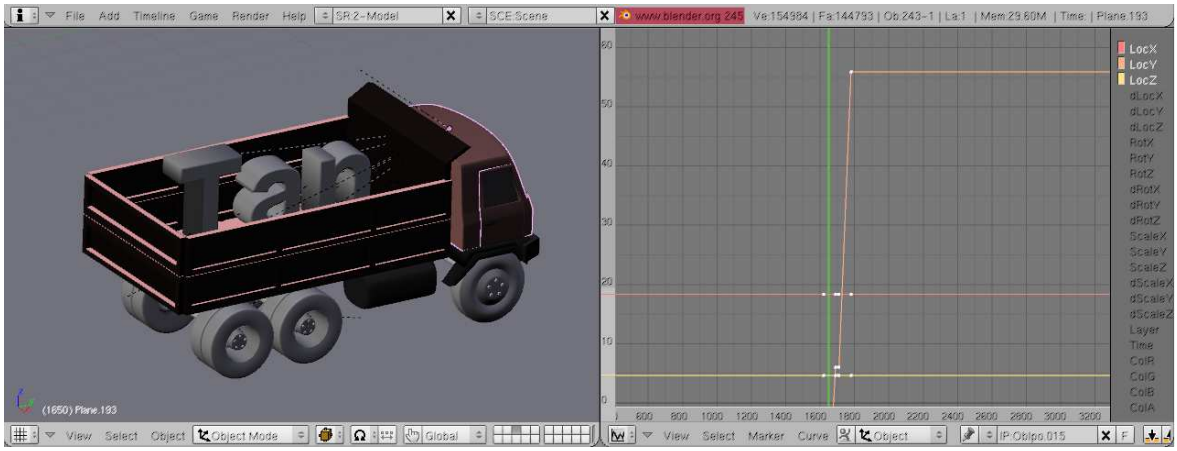


Obr. 17. Prechodové charakteristiky materiálu a objektu v IPO Curve Editore

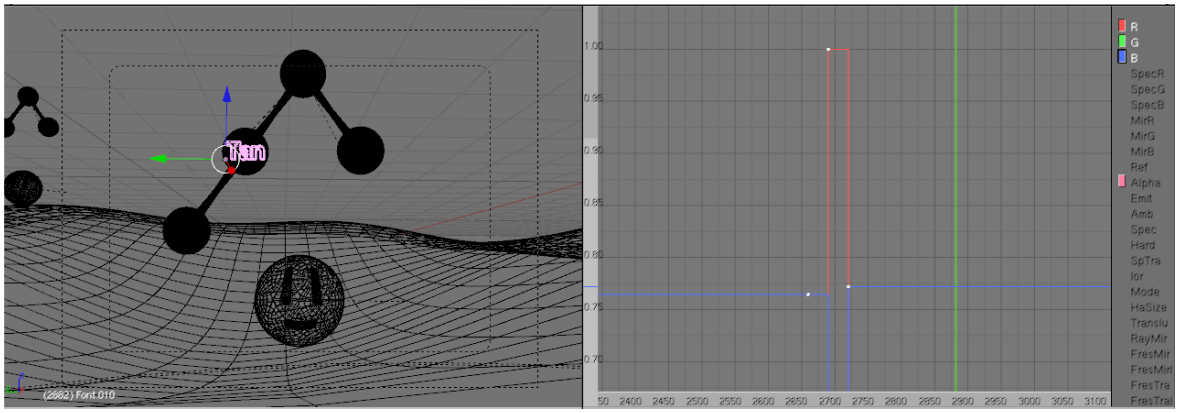
Blender automaticky dopočíta zmenu hodnôt daného parametru v snímkach, ktoré sa nachádzajú medzi počiatočnou a konečnou. Tu je možné nastaviť tri druhy dopočítavania hodnôt. Blender automaticky počíta hodnoty medzi dvoma kľúčmi z bezierovej krivky, ktorou sú tieto dva body spojené. Nie vždy je však bezierova krivka ideálne generovaná. V IPO curve editor je možné krivku pomocou riadiacich bodov upravovať, aby boli hodnoty dopočítavané medzi dvoma animačnými kľúčmi čo najideálnejšie. V niektorých prípadoch je však takýto prechod medzi snímkami nežiaduci. V takomto prípade je možné zvoliť prechod lineárny, kedy v každej snímke bude pripočítaná rovnaká zmena alebo prechod skokom, kde bude medzi dvoma snímkami skoková zmena hodnoty.



Obr. 18. Prechod medzi kľúčmi – bezierova krivka (IPO Curve Editor)

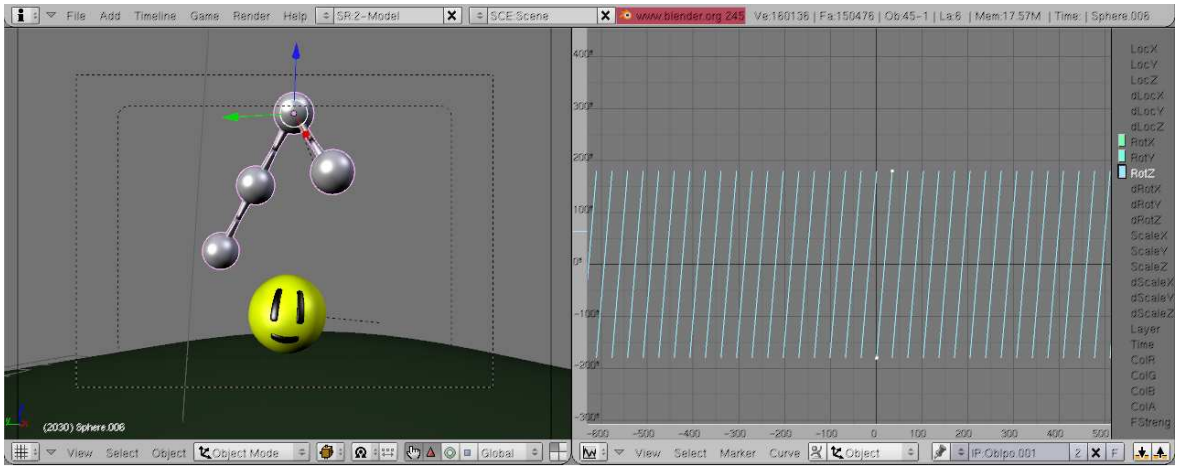


Obr. 19. Prechod medzi kľúčmi – lineárny (IPO Curve Editor)



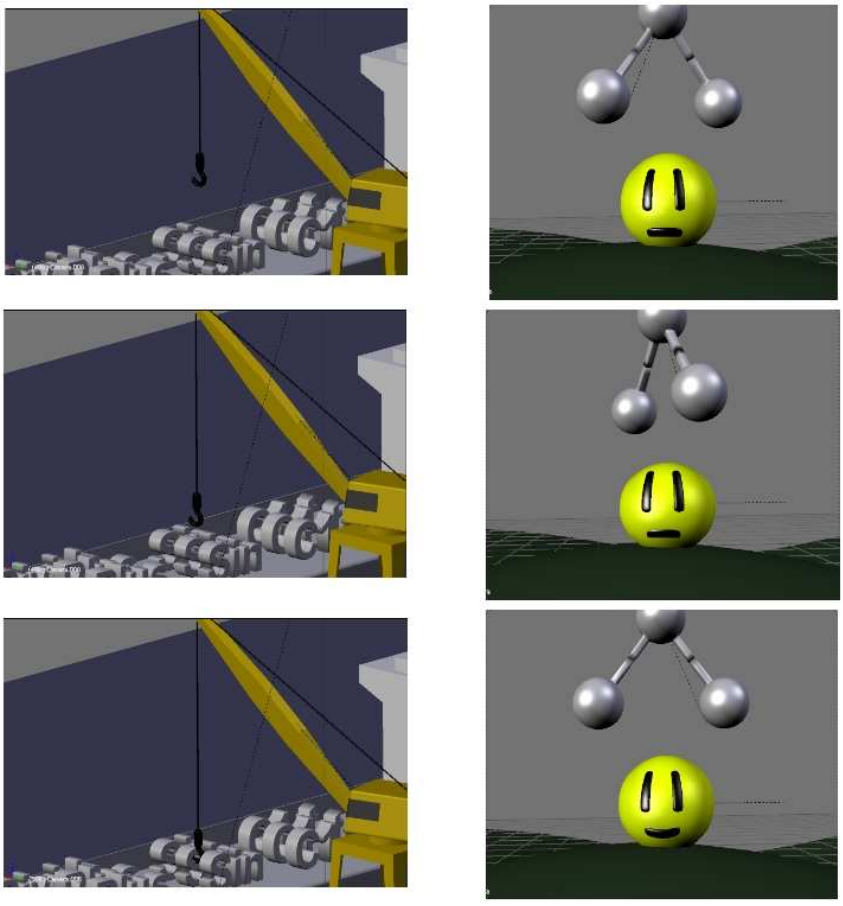
Obr. 20. Prechod medzi kľúčmi – skokom (IPO Curve Editor)

Niekedy je potrebné animačné sekvencie opakovať cyklicky donekonečna. V takomto prípade sa nastaví jeden cyklus a Blender automaticky vygeneruje v IPO Curve Editore nekonečný cyklus podľa nastaveného originálu.

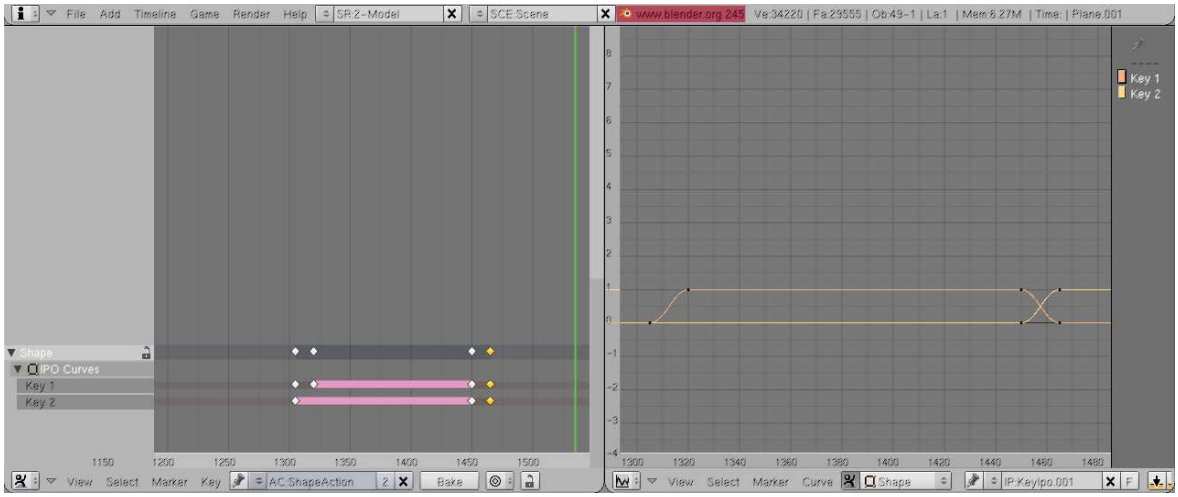


Obr. 21. Cyklická sekvencia (IPO Curve Editor)

Počas animácie bolo v niektorých prípadoch potrebné meniť vzhľad daných objektov. Blender toto umožňuje pomocou Relative vertex keys (shape keys). Pomocou tejto funkcie si program ukladal pozície vertexov a ich zmenu polohy v určitom časovom okamihu. Najprv bolo potrebné zadefinovať počiatočný objekt, ktorý sa mal počas animácie meniť pomocou kľúča „mesh“. V editačnom móde bol následne premodelovaný charakter objektu do odpovedajúceho tvaru. Takto boli definované dva stavy, medzi ktorými program vypočítal prechod. Ako v prípade zmeny materiálu alebo polohy objektu aj tu program počítal zmenu pomocou bezierovej krivky. Pomocou riadiacich bodov bolo možné krivku upraviť alebo zmeniť na lineárnu, poprípade na skokovú. Kľúče pre už vytvorenú zmenu objektu boli vkladané v Action Editore.



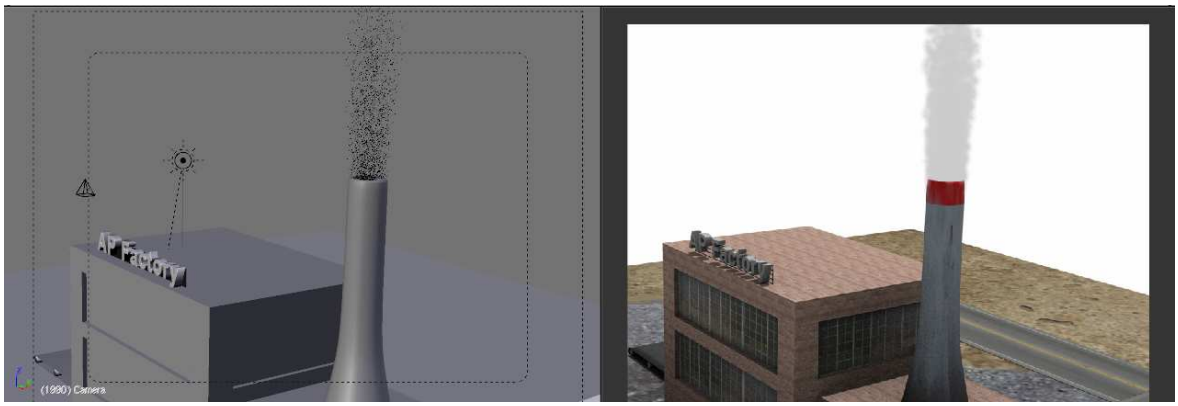
Obr. 22. Modifikácia objektu pomocou Shape keys



Obr. 23. Modifikácia objektu pomocou Shape keys – Action Editor a IPO Curve Editor

3.1.5 Efekty

Pre znázornenie efektu dymu vychádzajúceho z komína bol použitý časticový systém implementovaný do Blenderu. Dym bol simulovaný časticami s nastavením materiálom. Tieto častice po vyrenderovaní scény pripomínali dym. Časticový systém pracuje na princípe emitora, ktorý vyžaruje tieto častice do prostredia. Ako emitor bola použitá guľa s množstvom vetrxov umiestnená v komíne. Ich počet, rozptyľovanie, náhodnosť alebo rýchlosť sa dá dosiahnuť pomocou nastavení určených pre prácu s časticami.

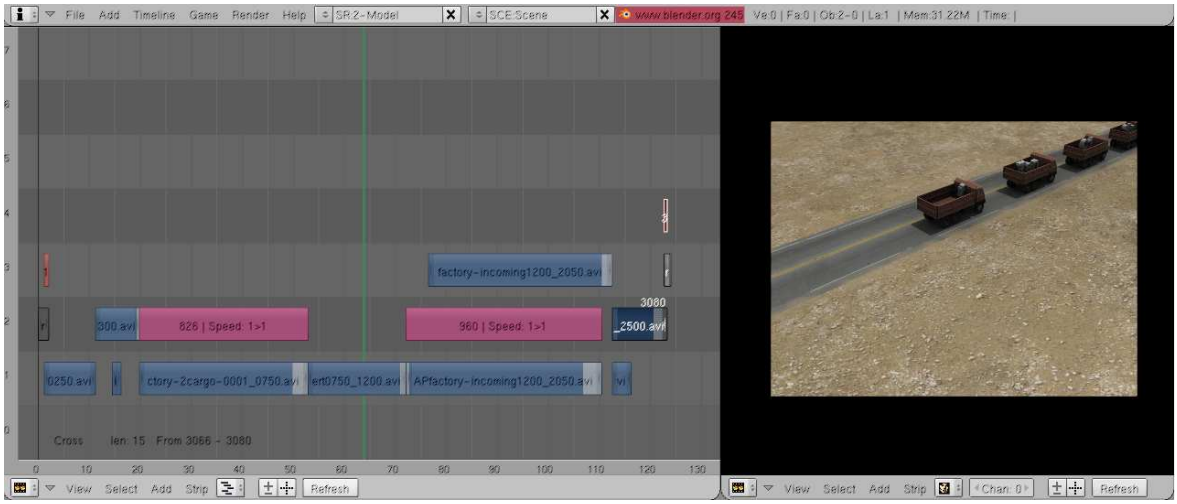


Obr. 24. Častice

Ďalším efektom bolo zjavenie objektu v scéne v danej snímke. Tento efekt bol využívaný prevažne v textových animáciách. Bol dosiahnutý nastavením priehľadnosti na maximum (alfa kanál = 0), čím sa dosiahlo úplné zneviditeľnenie objektu v scéne.

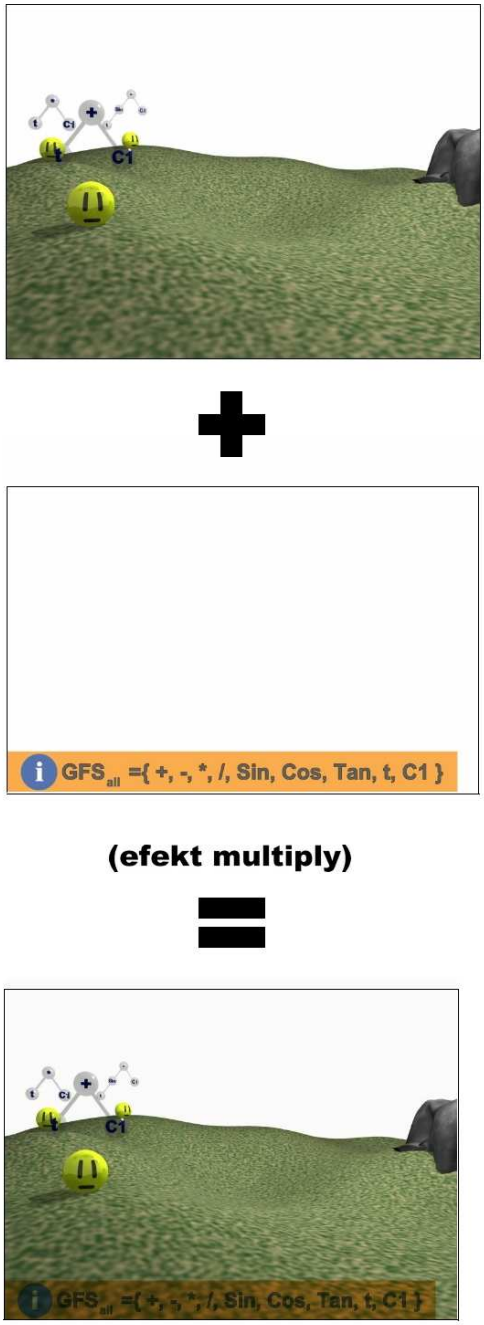
3.1.6 Strih a úprava animácii

Po vytvorení objektov, nastavení materiálov a textúr, nasvetlení scény a nastavení pohybov daných objektov bola animácia renderovaná. Pre náročnosť niektorých scén alebo pre dlhý renderovací čas neboli všetky animácie počítané vcelku. Na úpravu animácii, ich spájanie, rozdeľovanie, prechody medzi scénami bol použitý Video Sequence Editor.



Obr. 25. Video Sequence Editor

Pomocou vstavaných funkcii bolo docieleného napríklad efektu prelínania dvoch scén efektom „Cross“. Tento efekt bol použitý na úvodné a záverečné scény, kde bolo potrebné vytvoriť efekt zotmenia alebo rozjasnenia scény. V takomto prípade bol efekt počítaný ako prechod medzi čiernym alebo bielim obrázkom a danou scénou. Na spomalenie alebo zrýchlenie priebehu animácie bol použitý efekt „Speed control“. V niektorých prípadoch bolo potrebné spojiť dve animácie do jedného obrazu. Tento efekt bol docielený pomocou funkcie „Multiply“.

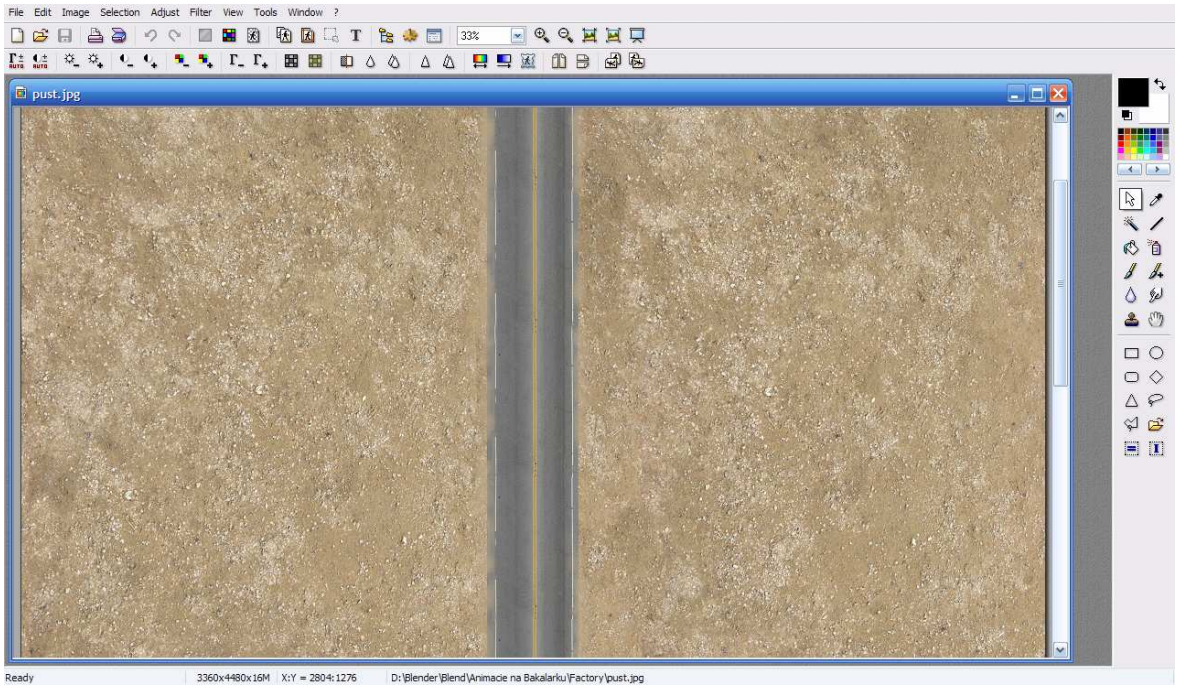


Obr. 26. Aplikácia efektu Multiply

3.2 PhotoFiltre

Photofiltre je freewarový grafický editor vhodný na úpravu obrázkov alebo tvorbu textúr. Pre potreby animácie boli vytvorené alebo upravené textúry pomocou jednoduchých nástrojov ako je klonovacie razítko. Veľmi užitočným nástrojom bolo orezávanie obrázka a úprava kontrastu alebo farby, poprípade spájanie viacerých textúr do jednej.

Textúry použité v animáciách pochádzali z internetovej databáze textúr CG – Textures (http://www.cgtextures.com/).



Obr. 27. Prostredie programu PhotoFiltre

4 VIZUALIZÁCIA ANALYTICKÉHO PROGRAMOVANIA

Vysvetlenia autorov článkov alebo kníh o analytickom programovaní ako jednej z metód symbolickej regresie nemusí byť pre každého dostatočne pochopiteľné. Táto práca bola vytvorená za účelom čo najlepšieho osvetlenia fungovania algoritmu pomocou 2D a 3D grafiky.

4.1 Vizualizované operácie

Vybrali sme kľúčové a podstatné časti a zvolili vhodnú formu vizualizácie, ktorá osvetľuje názorne a jednoducho algoritmus analytického programovania počas jeho priebehu od výberu jedinca až po nájdenie najvhodnejšieho riešenia daného problému.

4.1.1 Základná množina (General Functional Set)

Animácia osvetľuje vznik a delenie základnej množiny (GFS_{all}) na podmnožiny na základe počtu argumentov daných funkcii, operátorov alebo terminálov.

4.1.2 Jedinec

Animácia zobrazuje populáciu jedincov – vektorov, ktoré obsahujú celočíselné indexy ukazujúce do základnej množiny. Veľkosť tejto populácie je nastavená užívateľom. Zobrazuje sa náhodný výber jedinca, ktorý bude mapovaný pomocou základnej množiny na výslednú funkciu.

4.1.3 Operácie mapovania

Jedinec - vektor ukazateľov do základnej množiny a je namapovaný na jej prvky. Ako výsledok je syntetizovaný program (účelová funkcia). Animácia ukazuje spôsob mapovania jedinca na základnú množinu a vytvorenie účelovej funkcie.

4.1.4 Posilnená evolúcia

Každý jedinec má určenú svoju vhodnosť, ktorú určuje hodnota jeho účelovej funkcie. V animácii je zobrazené akým spôsobom je možné zvýšiť efektívnosť algoritmu v zmysle nájdenia čo najideálnejšieho riešenia v čo najkratšom čase. V novej populácii je možné použiť jedinca s najlepšou hodnotou účelovej funkcie v predchádzajúcom evolučnom cyk-

le, pokiaľ je jej hodnota menšia ako nastavený prah. Ak je táto podmienka splnená, je jedinec pridaný do základnej množiny ako terminál.

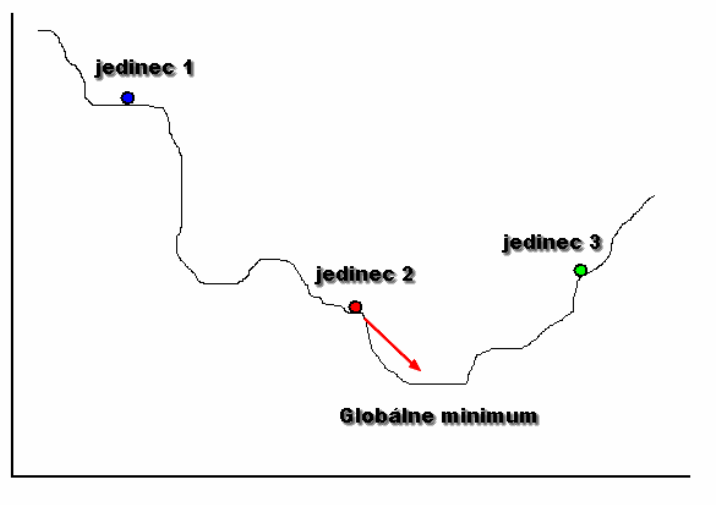
4.1.5 Bezpečnostné procedúry

Počas mapovania môže dôjsť k situácii, kedy už nie je vo vektore jedinca žiaden ukazateľ na ďalší prvok v základnej množine. V takom prípade by došlo k syntéze patologickej (neuzavretej) funkcie. Bezpečnostné procedúry zabraňujú vzniku takejto funkcie výberom funkcie s príslušným počtom argumentov, aby došlo k uzatvoreniu funkcie. Animácia zobrazuje toto mapovanie a uzatvorenie funkcie bezpečnostnou procedúrou a mapovaním z podmnožiny funkcii s menším počtom argumentov.

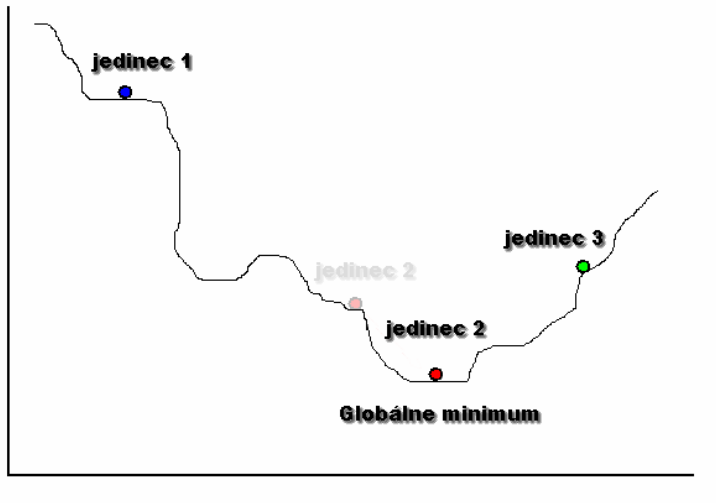
4.1.6 Mutácia

Animácia ukazuje alternatívny zápis syntetizovanej funkcie pomocou stromu. Na vykreslenom strome je zobrazená mutácia, kedy dochádza k zmene vo vektore jedinca, čo sa prejaví zmenou výslednej funkcie. Boli zobrazené dva prípady. V prvom prípade mutácia jedinca neovplyvní štruktúru stromu, v druhom prípade však už dochádza k zmene štruktúry.

Ďalším zobrazením princípu mutácie je mutácia jedinca v evolučnom algoritme SOMA. Plocha predstavuje rez n-rozmernej plochy ako priestor, ktorý si dokáže človek predstaviť. Jedinci sú symbolizovaný guľôčkami. Pri zmene polohy jedinca dochádza k jeho mutácii, čiže sa mení vektor jedinca. Pre daný prípad dochádza k mutácii len tých indexov, ktorých sa týka daný rez plochou. Ostatné indexy ostávajú pre daný rez zdanlivo nezmenené, avšak dochádza k ich mutácii, len nie je pre dané rozmery viditeľná. Jedinec sa snaží konvergovať ku globálnemu minimu, z čoho vyplýva, že takýto jedinec bude mať najlepšiu hodnotu účelovej funkcie, a teda bude aj predstavovať najlepšie riešenie daného problému.



Obr. 28. Rez n-rozmernou plochou pred mutáciou



Obr. 29. Rez n-rozmernou plochou po mutácii

4.1.7 Nelineárne prekladanie konštánt

Účelom animácie bolo vysvetlenie priebehu nelineárneho prekladania konštánt, kedy sú konkrétne konštanty vypustené a nahradené obecnými konštantami. Obecné konštanty sú nahrádzané konkrétnymi náhodne generovanými konštantami z určitého intervalu až tesne pred výpočtom samotnej účelovej funkcie. Najnižšia hodnota účelovej funkcie znamená najvhodnejšie zvolené konštanty.

4.1.8 AP Factory

Analytické programovanie je znázornené ako továreň, ktorá príjme na vstupe jedinca a na výstupe je výsledná funkcia po syntéze. Jedná sa v podstate o názorne zjednodušené mapovanie jedinca na základnú množinu zobrazené z nevedeckého hľadiska ako výrobný proces v továrni, kde jedinec a základná množina sú chápané ako vstupy do výroby, algoritmus analytického programovania je výrobný proces a na výstupe je výsledná funkcia ako výrobok .

ZÁVER

Podarilo sa nám úspešne dosiahnuť všetkých cieľov, ktoré sme si na začiatku našej práce určili. Pomocou veľmi výkonných freewarových grafických aplikácii sa nám podarilo vytvoriť súbor krátkych animácii zobrazujúcich priebeh algoritmu analytického programovania. Spojením týchto animácii do jedného celku bola vytvorená komplexná vizualizácia, ktorá by mala pomôcť divákovi pochopiť základné deje a postupy v algoritme.

Kľúčovým prvkom bol správny výber procesov schopných osvetliť fungovanie algoritmu. Nemenej dôležitá bola taktiež samotná vizualizácia, od ktorej záviselo správne pochopenie zo strany diváka. Ukázalo sa, že aj freewarové grafické nástroje sú schopné dobre poslúžiť nášmu účelu a je možné v nich vytvoriť vhodné vizualizačné materiály. Úroveň takýchto prác vytvorených vo freewarových aplikáciách môže byť takmer na úrovni profesionálnych grafických aplikácii, ktoré majú neraz aj stotisícové hodnoty. Jediným obmedzením sú schopnosti a skúsenosti autora, ktorý pracuje v takomto grafickom prostredí.

Zvolili sme jednoduchšiu grafickú úpravu animácii bez zbytočných rušivých grafických efektov. Taktiež aj v trojdimenzionálnych animáciách bola scéna upravená v duchu jednoduchosti a názornosti, bez akýchkoľvek rušivých elementov. Ďalším dôvodom k vytvoreniu jednoduchých animácii bola aj veľká výpočtová náročnosť spojená s ich vytváraním. Niektoré časti animácie bolo nutné renderovať aj viac ako 12 hodín, pričom vo výsledku bolo za tento čas vytvorených len niekoľko sekúnd animácie.

Je potrebné dodať, že animácie boli navrhnuté ako pozadie k prezentácii, čiže pre úplné pochopenie danej problematiky je potrebné názorné vysvetlenie prednášajúceho.

Rôzne vizualizačné metódy slúžiace ako nástroje propagácie množstva vedeckých prác verejnosti budú aj naďalej neodmysliteľným pomocníkom veľkých vedeckých objavov. Už je len otázkou budúcnosti aké vzrušujúce veci nás čakajú na poli vedeckom, ale i grafickom.

ZÁVER V ANGLIČTINE

We have successfully achieved all goals that we have defined at the very beginning of our work. Using very powerful freeware graphical applications we were able to create a complex of short animations showing the running of analytical programming algorithm. Connecting of these animations into one part was created complex visualization that should help viewer to understand the fundamental processes in algorithm.

The key factor was the right selection of processes able to make the function of algorithm clear. The visualization itself, on which depended the right understanding from viewer's point, was also important. It has shown, that even freeware graphic tools are capable to serve good to our purpose and is possible to create suitable visualization materials on their basis. The level of such works created with freeware applications might be almost on the same level in comparison with professional graphic applications that can cost sometimes an enormous amount of money. The only limitations are the abilities and experiences of author working in such environment.

We have chosen simple graphical arrangement of animations without disturbing graphical effects. The scenes in 3D animations were also arranged in a spirit of simplicity without any disturbing elements. Other reason for the creation of simple animations was also quite time consuming rendering time connected with its creation. Some parts of animation had to be rendered more than 12 hours, whereas in the result were created only few seconds of animation during this time.

It is necessary to note that animation were designed as a background for presentation, therefore the transparent explanation of lecturer is necessary.

Various visualization methods, serving as instruments for promotion of number of scientific works to public, will continue to be an essential helper of huge scientific discoveries. It is just a question of future how fascinating things can we expect on the scientific as well as graphical field.

ZOZNAM POUŽITEJ LITERATÚRY

[1] ZELINKA, Ivan. 2004, SOMA – *Self Organizing Migrating Algorithm*, kap.7, str. 33, in B.V.Batu, G. Onwubolu (eds), New Optimization Techniques in Engineering, Springer-Verlag.

[2] ZELINKA, I.,OPLÁTKOVÁ, Z. 2003, *Analytic programming – Comparative Study*. CIRAS '03, The second International Conference on Computional Intelligence, Robotics and Autonomous Systems, Singapore, 2003, ISSN 0219-6131.

[3] ZELINKA, Ivan. 2002. *Umělá inteligence v problémech globální optimalizace*. Praha: BEN, 2002 189 s. ISBN 80-7300-069-5.

[4] OPLÁTKOVÁ, Zuzana. *Metaevolution – Synthesis of Evolutionary Algorithms by means of Symbolic Regression*. [s.l.], 2007. 198 s. Zlín. Vedoucí dizertační práce doc. Ing. Ivan Zelinka Ph.D.

[5] ZELINKA, I., OPLÁTKOVÁ, Z., NOLLE, L. *Analytic programming – Symbolic regression by means of arbitrary evolutionary algorithms*. International Journal of Simulation, Systems, Science & Technology : Intelligent Systems [online]. 2005 [cit. 2008-04-10]. Dostupný z WWW:

<http://ducati.doc.ntu.ac.uk/uksim/journal/Vol-6/No.9/cover.htm>.

ISSN 1473-804x.

[6] VAŘACHA, Pavel. *Syntéza neuronových sítí metodou symbolické regrese*. [s.l.], 2006. 83 s. Zlín Vedoucí diplomové práce doc. Ing. Ivan Zelinka Ph.D.

[7] Genetic algorithm. *Wikipedia* [online]. 2008 [cit. 2008-04-10]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Genetic_algorithm>.

[8] Simulaten Annealing. *Wikipedia* [online]. 2008 [cit. 2008-04-10]. Dostupný z WWW: http://en.wikipedia.org/wiki/Simulated_annealing.

[9] POKORNÝ, Pavel. *Blender - Naučte se 3D grafiku*. Praha : BEN, 2006. 248 s. ISBN 80-7300-203-5.

[10] *3D scéna* [online]. 2008. Dostupný z WWW:

<http://www.3dscena.cz/3dshowks.php?xuid=207>.

[11] *Blender* [online]. 2008. Dostupný z WWW:

<http://www.blender.org/education-help/tutorials/>.

[12] *Blenderartists* [online]. 2008. Dostupný z WWW:

< http://blenderartists.org/cms/content/view/17/53/>.

[13] *Blendernation* [online]. 2008. Dostupný z WWW:

< http://www.blendernation.com/category/tutorials/>.

[14] Ambient Occlusion. *Blender3d* [online]. 2008 [cit. 2008-04-10]. Dostupný z WWW: < http://blender3d.cz/others/tnt/index.html>.

ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK

AP	Analytické programovanie (Analytic Programming)
arg	argument funkcie
CR	prah kríženia v DE
D	počet argumentov účelovej funkcie v SOMA a DE
DE	Diferenciálna evolúcia (Differential Evolution)
DSH	diskrétna množina v AP (Discrete Handling Set)
DVD	priložené záznamové médium
CF	účelová funkcia (Cost Function, Fitness Function)
F	mutačná konštanta v DE
GE	Gramatická evolúcia (Grammatical Evolution)
GFS	základná množina v AP (General Functional Set)
GP	Genetické programovanie (Genetic Programming)
K	konštantná funkcia, K_i je neskôr determinované napr. pomocou nelineárneho prekladania
NP	počet jedincov v populácii SOMA
PRT	parameter SOMA
RT	ray tracing, metóda výpočtu osvetlenia v grafických programoch
SOMA	Samo - organizujúci sa migračný algoritmus (Self – Organizing Migration Algorithm)
2D	dvojdimenzionálny priestor
3D	trojdimenzionálny priestor

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK

ZOZNAM PRÍLOH

Príloha PI: Adresárová štruktúra priloženého DVD

PRÍLOHA P I: ADRESÁROVÁ ŠTRUKTÚRA PRILOŽENÉHO DVD

Rozhodli sme sa pre jednoduchšiu orientáciu uviesť popis adresárovej štruktúry priloženého záznamového média. Najprv je uvedená cesta ku konkrétnemu adresáru a potom stručný popis jeho obsahu.

- /praca

  Obsahuje text tejto bakalárskej práce.

- /prezentacia

  Obsahuje prezentáciu vizualizácii AP v programe MS PowerPoint vhodnú ako sprievodnú prezentáciu k prednáškam.

- /vizualizacie/AP

  Obsahuje vizulizácie AP vo vysokej kvalite.

- /vizualizacie/AP_komprimovane

  Obsahuje vizualizácie AP v nižšej kvalite vhodné k umiestneniu na internet.

- /vizualizacie/AP_animacie

  Obsahuje vizualizácie AP rozdelené podľa podkapitol.