Vizualizace funkcí Analytického programování pro potřeby prezentací

Visualization of functions in Analytic programming for presentations

Martin Papež

Bakalářská práce
2010

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2009/2010

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Martin PAPEŽ**
Osobní číslo:	**A07588**
Studijní program:	**B 3902 Inženýrská informatika**
Studijní obor:	**Informační a řídicí technologie**
Téma práce:	**Vizualizace funkcí Analytického programování pro potřeby prezentací**

Zásady pro vypracování:

Cílem bakalářské práce je vytvořit animace pro potřeby prezentací Analytického programování ve standardních prezentovacích programech.

1. **Seznamte se s Analytickým programováním.**
2. **Vytvořte animace vlastního procesu Analytického programování.**
3. **Vytvořte animace příkladů pro využití Analytického programování.**

Rozsah bakalářské práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. **ZELINKA, I., OPLATKOVÁ, Z., OŠMERA, P., ŠEDA, M., VČELAŘ, F. Evoluční výpočetní techniky – principy a aplikace. BEN – technická literatura, Praha, 2008, ISBN 80-7300-218-3.**
2. **ZELINKA, I., OPLATKOVÁ, Z., NOLLE, L., Boolean Symmetry Function Synthesis by Means of Arbitrary Evolutionary Algorithms-Comparative Study, International Journal of Simulation Systems, Science and Technology, Volume 6, Number 9, August 2005, pages 44 – 56, ISSN: 1473-8031, online http://ducati.doc.ntu.ac.uk/uksim/journal/Vol-6/No.9/cover.htm, ISSN: 1473-804x.**
3. **OPLATKOVÁ, Z.: Metaevolution – Synthesis of Optimization Algorithms by means of Symbolic Regression and Evolutionary Algorithms, Lambert-Publishing, 2009, ISBN 978-8383-1808-0.**
4. **POKORNÝ, P. Blender: Naučte se 3D grafiku. 2. vyd.: BEN – technická literatura, 2009, ISBN 80-7300-244-2.**
5. **MEYER, T. Adobe After Effects, Computer Press, 2010, ISBN: 978-80-251-2500-7.**

Vedoucí bakalářské práce: **Ing. Zuzana Oplatková, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **5. března 2010**

Termín odevzdání bakalářské práce: **1. června 2010**

Ve Zlíně dne 5. března 2010

L.S.

prof. Ing. Vladimír Vašek, CSc.
děkan

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

ABSTRAKT

Cílem bakalářské práce je vytvoření vizualizace algoritmu Analytického programování. V teoretické části jsou předvedeny základní principy evolučních algoritmů, které jsou nutné pro vlastní činnost Analytického programování. V animacích je ukázán proces tvorby optimalizované funkce a využití AP v konkrétních situacích. Animace názorně a jednoduše přibližují divákovi vybrané úseky z této problematiky. Vytvořené projekty jsou uloženy na přiloženém CD-ROM.

Klíčová slova: Analytické programování, Evoluční algoritmy, vizualizace, animace

ABSTRACT

The aim of the bachelor thesis is to elaborate a visialisation of an algorithm in Analytic programming. In the theoretical part are presented the essential principles of the Evolutional algorithms, which are necessary for the Analytic programming. The animations describe the creative process of the optimized function and the usage of AP in particular situations. Consequently, the animations focus on the particular elements of the chosen issue. Both of the generated projects are saved on the added CD-ROM.

Key words: Analytic programming, Evolutional algorithms, visualisation, animation

Tímto bych chtěl poděkovat mé vedoucí bakalářské práce Ing. Zuzaně Oplatkové Ph.D. za cenné rady, trpělivost a pohodový přístup, se kterým se mi věnovala.

Můj dík samozřejmě také patří rodině, která mě jak psychicky tak i materiálně podporovala ve studiu.

„Think rich, look poor“ – Andy Warhol

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo –bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu
  využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně …….…….

podpis diplomanta

OBSAH

ÚVOD

Problematiku optimalizace můžeme řešit pomocí klasického matematického aparátu. Nicméně tento způsob umožňuje nalezení řešení pouze pro problémy jednoduššího charakteru. Řešení není zpravidla optimální. Výpočetní náročnost se zvyšuje společně se složitostí algoritmu, kde svou roli hraje i druh argumentů účelové funkce. [10]

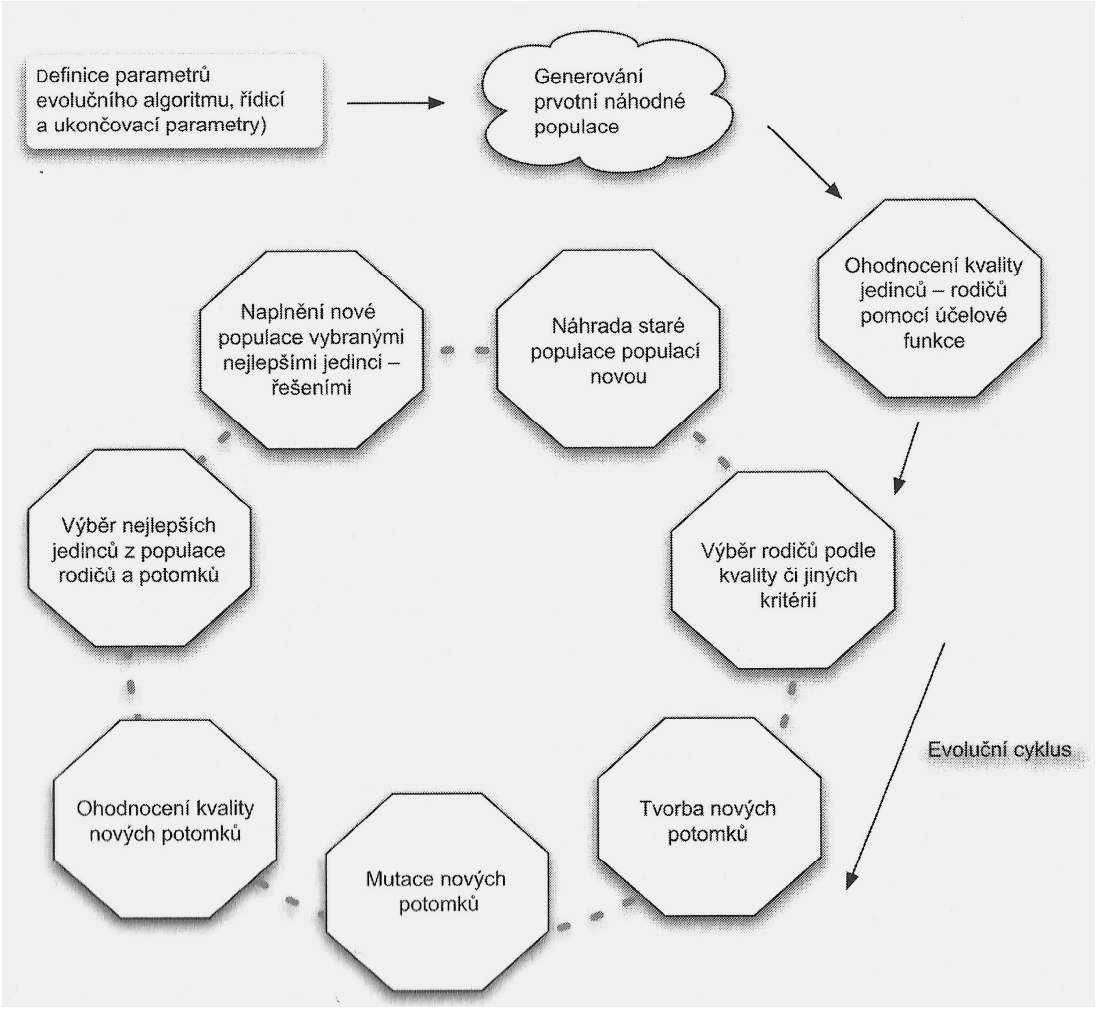
V průběhu posledních třiceti let vznikly nové metody řešení problémů optimalizace založené na tzv. Evolučních algoritmech. Informace o těchto algoritmech jsou zpracovány v teoretické části této práce. Výhodou těchto metod je, že oproti matematickému modelu se zaměřují na hledání globálního extrému. Jednou z těchto metod je i Analytické programování. AP potřebuje ke svému běhu evoluční algoritmus, na základě kterého se snaží přiblížit k optimu. Algoritmy pracují s náhodnými veličinami, tudíž jejich výsledek nelze předpovídat. Pouze můžeme porovnat zkušenosti s jednotlivými z nich, kde hodnotíme jejich životaschopnost a použitelnost v praxi.

Rozvoj techniky jde ruku v ruce s rozvojem civilizace. Jedním způsobem jak můžeme hodnotit úspěšnost civilizace je úroveň vzdělání. Je jen na nás jak budeme techniku využívat v tomto směru. Výsledkem této práce jsou animace, které přibližují princip a využití jedné z nových výpočetních metod. V praktické části bakalářské práce jsou pak uvedeny informace o způsobu výroby vizuálního díla. Jsou zde nastíněny postupy výroby od otevření programu až po výsledný rendering. Zmíněny jsou také programy, ve kterých samotná animace vznikala. Modely byly vyrobeny v open-sourcovém programu Blender (verze 2.49). Postprodukce a efekty jsou dodělány ve víceúčelovém programu Adobe After Effects.

I. TEORETICKÁ ČÁST

1 EVOLUČNÍ ALGORITMY

Evoluční algoritmy jsou odvozeny od vývojových procesů, které probíhají v přírodě už miliony let. Toho si všiml v 19. století Charles Darwin a na základě dlouhodobého výzkumu rozvinul myšlenku o přirozeném výběru. Ta se stala základem pro vývoj umělé inteligence, jež poznala světlo světa až o sto let později. Začátek historie evolučních algoritmů se obvykle datuje do poloviny 70.let. Duchovními otci evolučních algoritmů jsou A. Turing a A. Barricelli. Již v této době formulovaly přesné principy evolučních algoritmů, které však nemohli realizovat.



Obrázek 1: Schéma Evolučního algoritmu[10]

Podle Darwinovy teorie je uznáváno evoluční dogma, podle něhož se jednotlivé druhy vyvíjejí tak, že jsou z rodičů plozeni potomci, kteří podléhají při svém vzniku mutacím. Rodiče a potomci nevhodní pro aktuální životní prostředí vymírají aktuální životní prostředí vymírají „cyklicky" po tzv. generacích, čímž uvolňují místo novým lepším potomkům, kteří se stávají novými rodiči. Schéma takovéto evoluce je zobrazeno na *obr. č. 4.* [10]

Z této teorie vycházeli i A.M.Turing a A.N. Barricelli, kteří tyto principy přenesli ve zjednodušené podobě do výpočetních metod. Jimi prezentované metody daly možnost vzniknout dalším optimalizačním algoritmům. Rozvoj evolučních algoritmů závisí na rozvoji počítačů a pokrokách v informatice. [8] [10]

Pro správnou funkci evolučního algoritmu je nutné na začátku vymezit parametry evoluce. Tyto parametry řídí běh algoritmu případně ho regulérně ukončí - při splnění/dosažení stanoveného kritéria ukončení. Nejčastěji používaná kritéria pro ukončení jsou - počet určitých generací nebo dosažení požadované kvality optimalizace. Součástí nastavení parametrů je též stanovení účelové funkce, která bývá také označována jako funkce vhodnosti (**fitness**). Účelová funkce je obvykle představována matematickým modelem, který pomocí hledání extrému vede k řešení problému. Omezující parametry společně s vyhodnocováním účelové funkce slouží jako prostředky pro vyhodnocení kvality jedinců.

Množina jedinců se nazývá populace, kde první populace bývá generována na základě nastavení základních parametrů. Jedinci se rozumí jako vektoru čísel, který má tolik složek, kolik je optimalizovaných parametrů účelová funkce. Každý jedinec představuje jedno možné konkrétní řešení problému. Dalším krokem je ohodnocení jedinců na základě účelové funkce. Na základě tohoto vyhodnocení nastává výběr nových rodičů podle kritérií kvality. Noví jedinci se do generace dostávají pomocí reprodukce stávajících prvků. Prvky se můžou reprodukovat pomocí křížení. **Proces křížení je u každého evolučního algoritmu odlišný**. Dalším způsob jak rozšířit populaci o nové jedince je pomocí tzv. mutování. Jedná se metodu rozšiřování populace pomocí náhodného procesu. Každý nový jedinec

je opět ohodnocen a zařazen do civilazace. Výběrem nejlepších jedinců zaplníme nově vzniklou populaci. Stará civilizace je zapomenuta. Tyto kroky se opakují až do splnění ukončovacích podmínek, které algoritmus ukončí v případě ukončení posledního generačního cyklu nebo v případě dosažení určité kvality optimalizované funkci.

Evoluční algoritmy se staly jednou z nejpoužívanějších metod pro řešení hledání optimalizace funkce. V současné době mají své uplatnění při řešení složitých matematicko-ekonomických problémů. Na základě jejich rozvoje vznikají nové algoritmy pracující na jejich vylepšených principech.

2 ANALYTICKÉ PROGRAMOVÁNÍ

Analytické programování (AP) bylo inspirováno numerickými metodami v Hilbertově funkcionálním prostoru a genetickým programováním. Na rozdíl od jiných metod není AP vázané s žádnou gramatikou ani stromovou strukturou. Jedná se o experimentální metodu, kterou lze chápat jako alternativní přístup právě vzhledem ke GP a GE. AP je postaveno na množině funkcí, operátorů a tzv. terminálů. Terminál si představme jako konstantu nebo nezávislou proměnnou.[10]

1, funkce {sin, cos, and, log, …}

2, operátory {+, - , * , / }

3, terminály {2, 23, t, …}

Vlastní princip analytického programování spočívá v práci s těmito množinami, přičemž se snažíme syntetizovat nejvhodnější možné řešení. Hlavní výhodou AP je, že může pracovat s různými evolučními algoritmy. Tento fakt je založen na práci s diskrétními množinami (DSH – discrete set handling), které nám umožní práci jak s celočíselnými tak i diskrétními hodnotami argumentů. V případě celočíselných argumentů nastává problém v momentě, kde je chtěno rozšiřovat populaci. Jedná o moment, ve kterém bude AP provádět celočíselné zaokrouhlování. V zásadě můžou nastat dva případy[10]

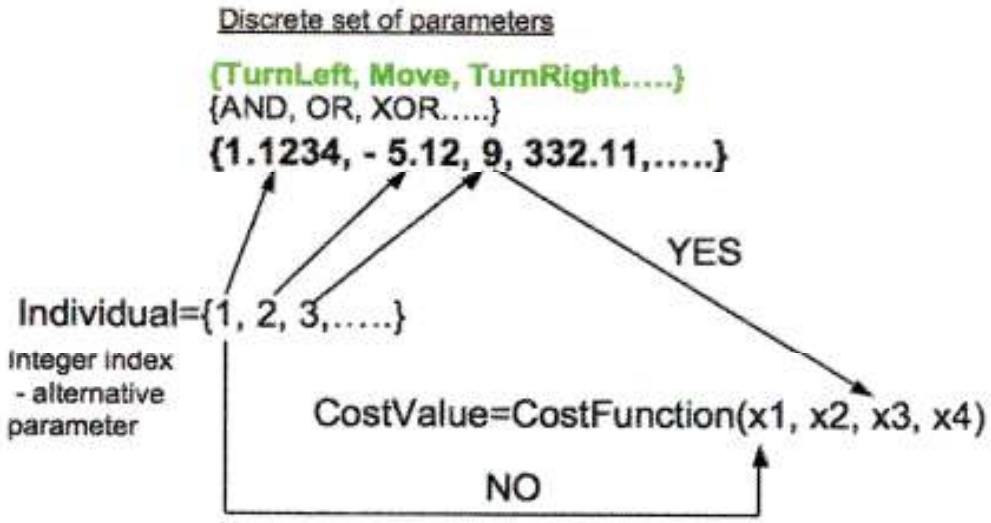
a, vybraného jedince z populace zaokrouhlujeme přímo v populaci

b, zaokrouhlování je provedeno před dosazováním do účelové funkce

Kde lepších výsledků dosahujeme při variantě **b.** Dosahujeme větší diverzibility a robustnosti evolučního procesu. To je zapříčiněno tím, že díky zaokrouhlování je větší počet pozic, ze kterých můžeme startovat.

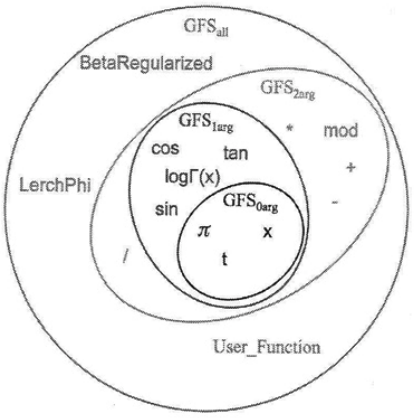
V případě použití diskrétních hodnot bude postup relativně obdobný. Při množině diskrétních hodnot, např. { -1, 2.5, -20, 5.68,…} se vytváří celočíselný

index, který nahradí daný parametr jedince a nabývá hodnot {1 ,2, 3, 4,…}. S tímto náhradním parametrem se pak pracuje namísto s jedincem diskrétní množiny. Jediný rozdíl je, že při ohodnocování účelové funkce se nedosazuje daný argument aktuálního čísla, ale dosadí se hodnota z původní diskrétní množiny. [10]



Obrázek 2: Výběr diskrétních hodnot

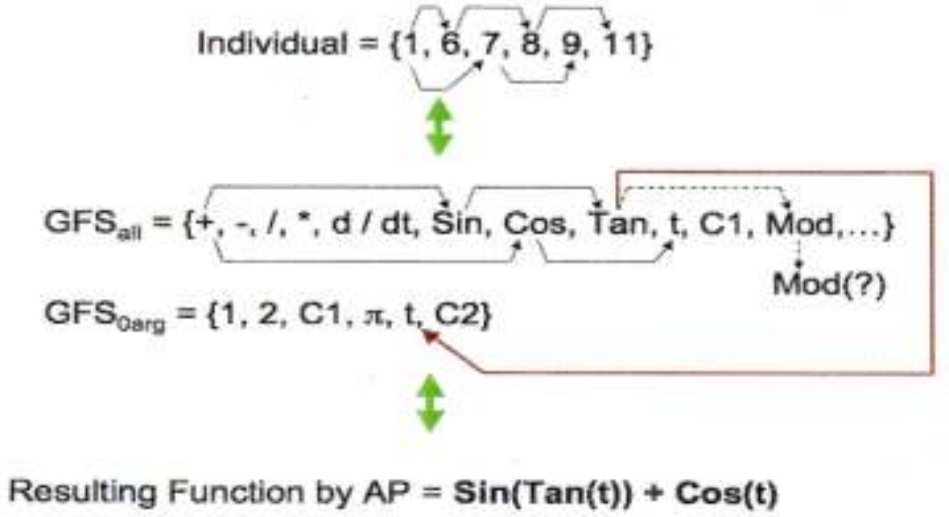
Základní množina - GFS (general function set) je tvořena pomocí jednotlivých množin funkcí, operátorů a terminálů. Analytické programování dělí GFS na několik podmnožin, které jsou dány počtem argumentů jednotlivých funkcí (např. $GFS_{1\arg} = \{\sin, \cos\}$, $\mathrm{GFS}_{0\arg} = \{x, t\}$). Zpravidla se vytváří podmnožina $\mathrm{GFS}_{\mathrm{all}} = \{x, \sin, \mathrm{mod}, +\}$, která je sjednocením všech ostatních $\mathrm{GFS}_{?\arg}$ (obr. č. 2)., která je využita k tomu, aby při syntéze nevznikaly patologické programy. To jsou programy, které např. volají neexistující funkce, dělí nulou atd.



$$GFS_{all} = GFS_{3arg} \cup GFS_{2arg} \cup GFS_{1arg} \cup GFS_{0arg}$$
$$GFS_{3arg} = \{LerchPhi, BetaRegularized, +, -, /, *, cos, tan, sin, log\Gamma, x, t, \pi\}$$
$$GFS_{2arg} = \{+, -, /, *, cos, tan, sin, log\Gamma, x, t, \pi\}$$
$$GFS_{1arg} = \{cos, tan, sin, log\Gamma, x, t, \pi\}$$
$$GFS_{0arg} = \{x, t, \pi\}$$

Obrázek 3: Schématická ukázka množiny GFS

Jak už bylo naznačeno v úvodu, AP se snaží najít co nejoptimálnější řešení daného problému na základě syntézy funkcí. Jádrem AP je zobrazení množiny základních symbolických objektů do množiny možných programů. Zobrazení je prováděno pomocí techniky DSH (discrete set handling). Jehož součástí je i tzv. „bezpečnostní prodecura“, která neustále kontroluje vzdálenost od celočíselného jedince a na jejím základě určuje, ze které podmnožiny GFS budou funkce vybírány. Princip bezpečnostní procedury můžeme vidět na *obrázku č. 3*, kde máme naindexovaného jedince, který se váže na jednotlivé programy z množiny GFS_{All}. Samotný cyklus evoluce probíhá stejně jako u jiných evolučních algoritmů za pomoci operátorů. Na rozdíl od jiných metod Analytické programování s nimi nepracuje. Jejich užití záleží vždy konkrétně na užitém algoritmu. Úspěšnost hledání můžeme ovlivnit vhodnou volbou základního rozdělení množiny (GFS). [8] [10]



Obrázek 4: Princip bezpečnostní procedury

Na *obr. č. 3* je zobrazen princip bezpečnostní producery. Podívejme se na samotné skládání funkce, kde problém nastává až pro dosazování argumentů do funkce Tan. Problém je ošetřen právě užitím BP, kde při kroku výběru jedince docházím k zjištění, že daná funkce nebude mít vstupní argumenty . Do třetí funkce jedince (č. 7 , Tan) dosazuji dvouargumentový program (Mod) s tím, že už nemám žádné jedince v populaci. Mod spadá do podmnožiny GFS_{2arg}, takže pokud bych funkci chtěl použít ve funkci výsledné musel bych mít ještě dva jedince v podobě dvou proměnných nebo terminálů. Bezpečnostní procedura zakročila a doporučila vybrat program z podmnožiny GFS_{0arg}, protože množství jedinců je konečné číslo. Po syntéze vzniká výsledná funkce $F = Sin(Tan(\omega) + Cos(t)$.

V průběhu simulací nevykazovala první verze Analytického programování uspokojivých výsledků. Proto byla použita technika posíleného hledání „**reinforced search**“, která spočívá v myšlence zařazení aktuálně syntetizovaného programu do množiny symbolických objektů GFS_{0arg} . Do této množiny ho zařazuji právě když bude jeho vhodnost pod stanovenými parametry. V případě nalezení programu vhodnějšího, nahradím program zavedený do základní množiny. To vede k zvyšování kvality a zároveň se nezvětšuje množina GFS_{0arg}.

2.1 GENETICKÝ ALGORITMUS

Zakladatelem genetického algoritmu (GA) byl v 60. letech minulého století John Holland. Jeho myšlenkou bylo využít evoluční principy (založené na metodách optimalizace funkcí a umělé inteligenci) pro hledání řešení konkrétních úloh. Studoval elementární procesy v populacích, které jsou z hlediska evoluce nepostradatelné. Na základě tohoto výzkumu navrhl genetický algoritmus jako abstrakci daných biologických procesů. [1] Vycházel z Darwinovy teorie přirozeného výběru, kdy přežívají pouze nejlépe přizpůsobení jedinci. Míru přizpůsobení nazval „fitness" jedince. Fitness nám slouží jako kvalitativní metoda vyhodnocování daného jedince. Podle této kvality jsou stochaicky vybráni jedinci, kteří jsou reprodukováni pomocí mutací a křížení čímž, vznikne nová populace. Jedinci s vyšší fitness mají vetší pravděpodobnost přežití a vetší pravděpodobnost reprodukce svých genu do generace potomků.

GA se snaží analogicky srovnat s evolučními procesy probíhající samostatně v jednotlivých biologických systémech. Základ GA je odvozen od biologické genetiky a dvoupohlavního rozmnožování. Všechny živé organismy mají zakódované veškeré informace o jedinci v podobě DNA, u genetického algoritmu je to obdobné. Máme uložen charakteristický popis jedince v podobě chromozmomu. Chromozom si můžeme představit jako řetězec složený ze symbolů 0 a 1 – genů, který obsahuje charakteristicky zakódované informace o jedinci (tzv. fenotyp). Tyto informace jsou použity jako měřítko kvality pro další vývoj v evoluci.

Schéma genetického algoritmu je následující :

1, Navržení genetické struktury

2, Inicializace

3, Ohodnocení

3.1, Konverze genotypu na fenotyp

3.2, Ohodnocení objektivní funkce

3.3, Konverze objektivní funkce na vhodnost

3.4, Konverze vhodnosti na selekci rodičů

4, Volba rodičů

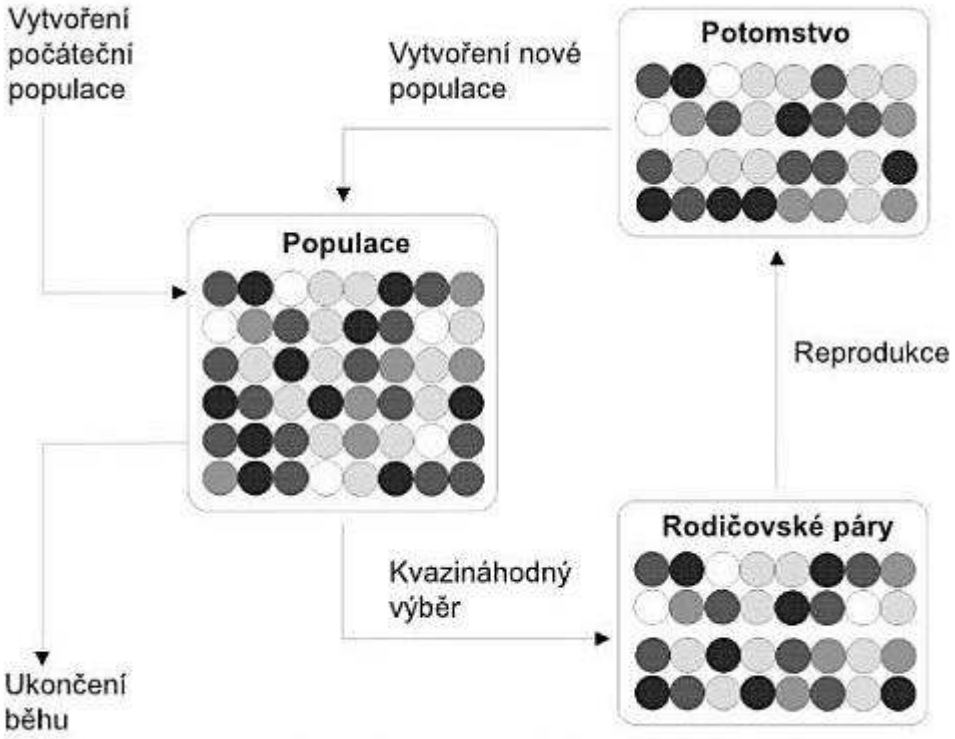
5, Reprodukce

6, Mutace

V některých publikacích se můžeme setkat s trochu modifikovaným schématem GA. [1][9]

Vlastní algoritmus genetické optimalizace je cyklus, ve kterém jsou vytvářeni noví potomci, kteří jsou použiti jako rodiče v dalším cyklu. Potomek sdílí po svých rodičích genovou výbavu a s tím částečně i jejich schopnosti. Podobně je tomu tak v přírodě, kde přežívají jen ti nejschopnější. Nevyhovující potomci umírají velmi rychle, dříve než stačí předat svým potomků „špatnou" genetickou výbavu. [9] Na začátku algoritmu získáme první rodiče pomocí náhodného vygenerování . V každém cyklu vyhodnocujeme vhodnost potomka na základě účelové funkce (fitness function). Takto procházíme celou populací jedinců a hledáme největší přiblížení k optimu. Nezbytnou součástí rozvoje genetických algoritmů je vlastní cyklus reprodukce nových potomků. V samotné reprodukci dochází k dělení chromozomů a vytváření potomků díky křížení a mutacím. Pomocí mutací nacházíme „nové-lepší" jedince, díky čemuž zkvalitníme

genetický proces. Aby byl vliv mutací na průběh genetického algoritmu důrazný, je potřeba správně volil jejich četnost výskytu. Vlastní mutace se provádí v cyklu pomocí náhodného generátoru čísel, tak že se vygenerované číslo porovnává s požadovanou četností výskytu. Pokud je náhodné číslo menší, pak dojde k mutaci genu – jedná se o součin vygenerovaného čísla a dalšího náhodného čísla s tím, že součin se přičte k již existujícímu genu jedince.

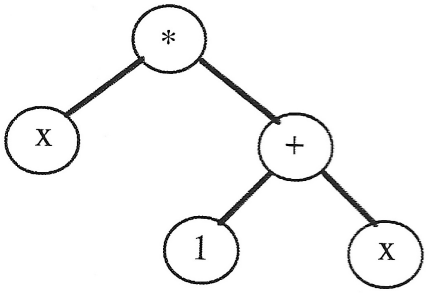


Obrázek 5: Znázornění principu funkce genetického algoritmu

Genetické algoritmy představují mohutný zdroj inspirace pro další evoluční algoritmy, pro zkoumání umělé inteligence a pro rozvoj soft-computing. Jejich podstata se stala základem mnohých jiných evolučních algoritmů.

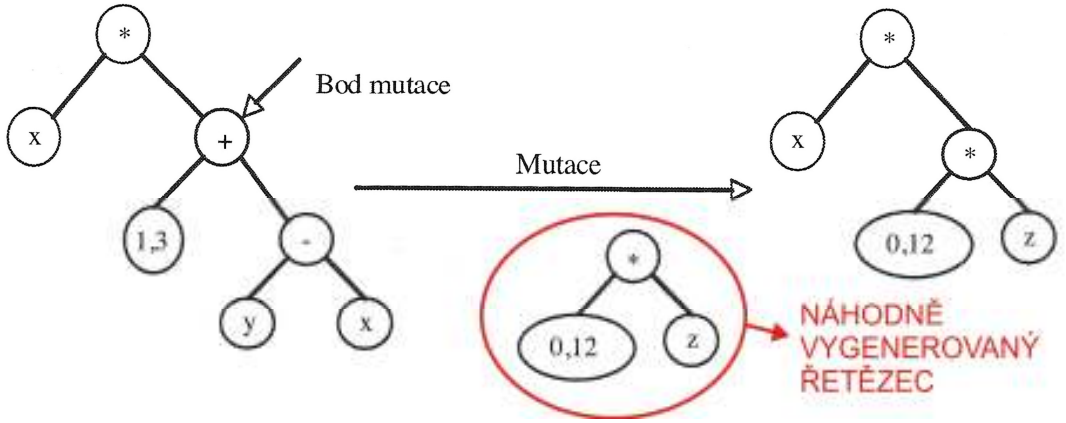
2.2 GENETICKÉ PROGRAMOVÁNÍ

První experimenty v tomto směru prováděli už 80. letech S.F. Smith a N. L. Clamer. Za zakladatele této metody se přesto považuje až americký informatik John Koza. Na konci 80. let navrhl modifikaci genetických algoritmů, kde propagoval šlechtění jedinců na symbolické úrovni. Na základě tohoto principu pracuje velké množství evolučních algoritmů. Genetické programování (GP) se liší od jiných GA především v reprezentaci jedinců. GP hledá řešení problémů reprezentované algoritmem tak, že jedinci ze kterých je složena populace jsou programy. Programy bývají zobrazeny do grafů nejčastěji typu strom, který bývá označován „parse tree" (*obr. č. x*).



Obrázek 6: Stromová struktura

S tímto způsobem programování se je možné setkat například u funkcionálních programovacích jazyků jako je např. LISP. Noví jedinci se vytvářejí na základě kombinace stávajících jedinců zapomoci rekombinačních operátorů. Ty přetvářejí jedince na základě nástrojů reprodukce jako jsou např. mutace a křížení (obr. č. x). V průběhu evoluce dochází ke změnám v syntaktickém stromu, kdy jsou jednotlivé části stromu jsou postupně nahrazovány složitějšími funkcemi. Pak hovoříme o tzv.dělení stromu.

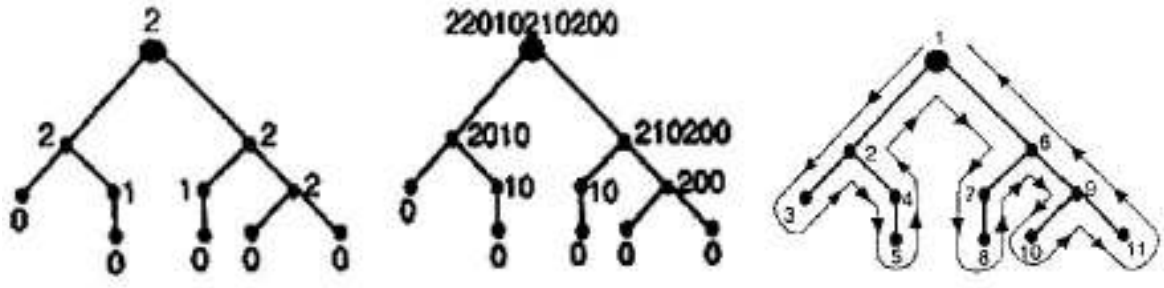


Obrázek 7: Mutace v Genetickém programování

Samotné GP je pak reprezentováno pomocí funkcí a terminálů. Množina všech možných programů je tedy dána konečnou rekurzivní kombinací funkcí těchto dvou množin (F – množina funkcí, T – množina terminálů). Množina funkcí obsahuje obvykle funkce aritmetické (+, - , *, /), matematické (sin, cos, exp), logické (and or, not) a fce podmíněné vyhodnocení (if). Terminály obsahují pouze formální parametry těchto funkcí, tedy konstanty proměnné.

Cílem je najít funkci reprezentovanou syntaktickým stromem, která minimalizuje rozdíl vypočítaných hodnot. John Koza nazval tento postup jako metodu symbolické regrese. Důležitým faktorem pro manipulaci se stromem je způsob kódování. Zde se můžeme setkat s tzv. **Readovým kódem** (RK), který umožní popis standartních procesů pomocí programovacích jazyků Pascal, C++ a Fortran. Užitím Readova kódu se stane GP mnohem efektivnějším.

Princip RK spočívá procházení stromu, kde se ohodnotí každý uzel hodnotou s kolika jinými uzly jsem musel k danému uzlu projít. Započítání každého uzlu probíhá pouze jednou a to při prvním průchodu. Hlavním přínosem Readova kódu je, že každý uzel stromu je jednoznačné identifikován.



Obrázek 8: Schématické znázornění Readova kódu

Totožně jako v případě genetických algoritmů se i u GP používá parametr pravděpodobnost mutace a křížení. V případě GP se ještě uvádí parametr pravděpodobnosti výběru jednotlivých uzlů po mutaci.

Schéma algoritmu GP [2]

1, vygeneruj výchozí populaci o velikosti N
2. je-li splněna podmínka pro ukončení algoritmu, vrať výsledek, jinak pokračuj
3. proveď ohodnocení jedinců v populaci
4. proveď reprodukci
5. kroky 5 až 8 proveď n^2 *– krát*
6. vyber dva jedince z populace.
7. proveď křížení jedinců a potomky zařaď do populace.
8. proveď mutaci potomků
9. aplikuj ostatní evoluční operátory
10. jdi na krok 2.

U genetického programování se během evoluce objevuje efekt **BLOAT,** který povoluje, že potomci mohou mít delší řetězec než-li jejich rodiče. Obvykle průměrná délka jedince narůstá s počtem iterací lineárně.Mezi možná řešení tohoto problému je potřeba zavést penalizaci dlouhých řešení, případně uzpůsobit operátory pomocí mutací či křížení. [10]

Ke genetickému programování existuje několik přístupů, které se v zásadě liší pouze v podobě kódování jedince a striktně se odlišují genotypem a fenotypem. Více informacích těchto přístupech naleznete v publikacích J. Kozy. [2] [3]

2.3 DIFERINCIÁLNÍ EVOLUCE

Diferenciální evoluce (DE) je poměrně nový typ evolučního algoritmu, který vyvinuli a poprvé použili Ken Price a Rainer Storm v roce 1995. Pracuje s podobnými schématy jako algoritmy genetické. Využívají principů tzv. **Genetického žíhání**, kde změnou reprezentace z binární do dekadické se zároveň změní logické operace na vektorové. Tyto změny udělaly z genetického žíhání algoritmus vhodný pro numerickou optimalizaci.

Chvíli po provedení těchto změn byla objevena tzv. „diferenciální mutace“, která spočívala v generaci tzv. **Zkušebních řešení** - pomocí přičtení diference dvou náhodně vybraných vektorů (jedinců) ke třetímu jedinci z populace. Zkombinováním DE a metody selekce z Genetického žíhání vznikla první verze diferenciální evoluce. [8]

Na testovacích funkcích byla první verze algoritmu velice úspěšná. Její experimentální výsledky konvergovaly rychleji než-li u jiných stochaických metod. První publikace se dočkal v roce 1996, nicméně se ukázalo, že DE je pro řešení komplexních optimalizačních problémů nedostačující. Nakonec se, ale ujala třetí verze algoritmu (1997), která se dostala rychle do podvědomí vědců a nadále se tento algoritmus vyvíjel. Touto problematikou se zabývá také prof. Ivan Zelinka, jež je autorem mnohých publikací. [1] [8]

Diferenční evoluce má oproti jiným evolučním algoritmů hnedka několik výhod.

- jednoduchost,
- možnost použít různé datové typy
- náhodný výběr rodičů vede takřka vždy k nalezení extrému
- schopnost dát vícenásobná řešení a najít extrém plochých funkcí

Cílem DE je vyšlechtit co nejlepší populaci jedinců ve smyslu hodnot účelové funkce. Celá generace pracuje s populací a jejími jedinci se může rozčlenit do následujících aktivit. [8]

1, Stanovení parametrů

2, Tvorba populace

3, Započetí cyklu generace

4, Evoluční cyklus

5, Testování naplnění ukončovacích parametrů

6, Vyhodnocení

Princip algoritmu diferenciální evoluce si zjednodušeně můžeme představit následovně. Algoritmus na základě stanovených parametrů vytvoří populaci. Populace je množina jedinců, kde si každý jedinec znázorňuje jedno aktuální řešení problému. Populaci si můžeme představit jako matici NxM, kde každý sloupcový vektor je jedinec. Každý jedinec se skládá z konečného počtu prvků, které s účelovou funkci udávají dimenzi, ve které probíhá evoluce. Započetí nové generace si můžeme představit jako cílené evoluční šlechtění každého jedince z populace.

Samotný **evoluční cyklus** vypadá asi takto: [15]

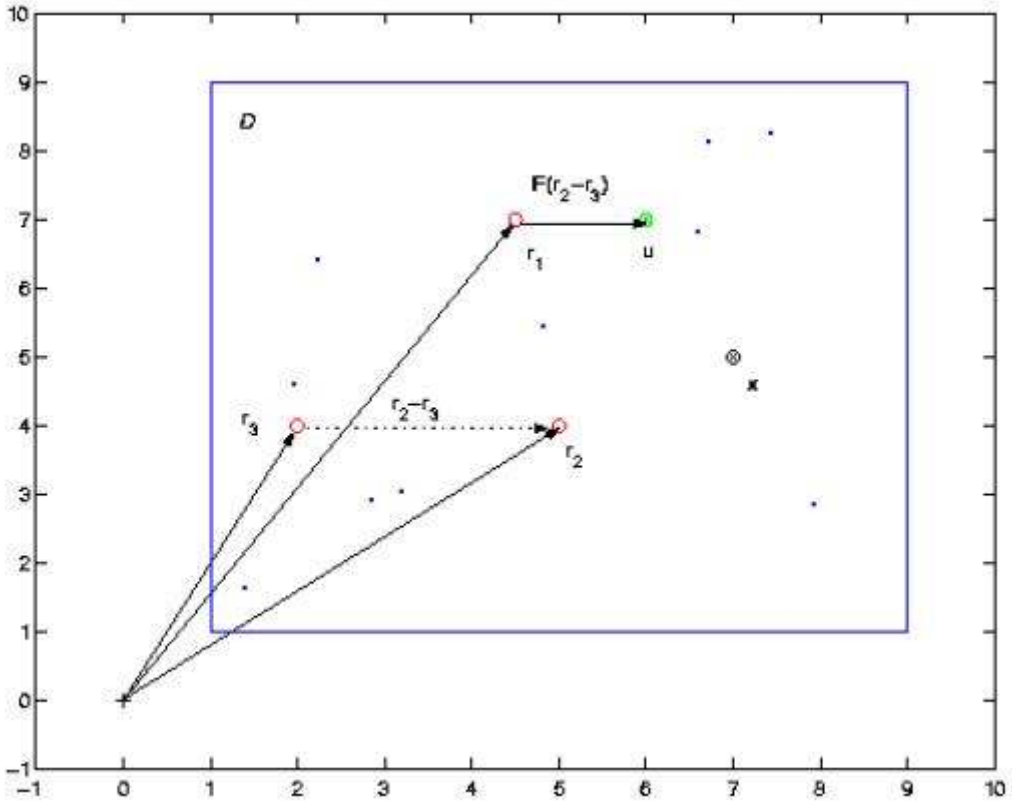
generuj P (N bodů náhodně v D)
repeat
    for i := to N **do**
        generuj vektor u
        vytvoř vektor y křížením u a x_i
            if f(y) < $f(x_i)$ **then** Q zařaď y
        else do Q zařaď do x_i
endfor
P := Q
until podmínka ukončení

Q –populace
P – stará populace
x_i -jedinec
y – zkušební vektor
u – nový jedinec

Nového jedince je možno vytvořit několika způsoby. Uvedeme si ten nejznámější, postup označovaný **RAND**. Spočívá v generování nového jedince **u** ze tří náhodně vybraných jedinců z populace ($r_{1,2,3}$**)** viz *vzorec č.1***,** kde F je mutační konstanta.

$$u = r_1 + F(r_2 - r_3) \tag{1}$$

Pro lepší představu uvádím evoluční postup **RAND** i v grafické podobně*(obr č. 8)*



Obrázek 9: Grafická ukázka principu diferenciální evoluce

Kvalitu optimalizace a průběh celého šlechtění ovlivňujeme nastavením vstupních faktorů algoritmu. Jsou to:

- nastavení řídících parametrů
- velikostí populace
- počtem generací
- definicí účelové funkce
- deficitní omezení

Více informací ohledně nastavení algoritmu můžeme nalézt v kapitole (11.18.2, Zelinka, Oplatková Evoluční teorie). [8] [10]

2.4 SOMA

SOMA – Self-Organization Migration Algrorithm spatřil svělto světa roku 1999. Jeho činnost je založena podobně jako u Diferenční evoluce nebo Genetického algoritmu na práci s vektorovými operacemi. SOMA patří mezi evoluční algoritmy i navzdory faktu, že během jednoho cyklu nejsou z hlediska filozofie algoritmu vytvářeni noví potomci. Přesnější je jej zařadit do skupin algoritmů memetických či hejnocých (**Swarm intelligence**).

SOMA byl vyvinut na principech, které lze odpozorovat v přírodě a kterými se v sociálně-biologickém prostředí řídí inteligentní jedinci, jenž kooperují na řešení společného úkolu. [8] SOMA je založena na principech spolupráce inteligentních jedinců jež hledají společně v prostoru optimální řešení. Příklady takového chování lze nalézt v reálném světě (např. včely, vlci, mravenci).

Biologická realita	Počítačová implementace
členové smečky, společenství	jedinci v populaci, parameter PopSize
člen společenství s nejlepším zdrojem potravy	Leader, vedoucí aktuálního migračního kola
potrava	vhodnost, hodnota účelové (kriteriální, cenové) funkce, geometricky je to lokální či globální extrém na *N* rozměrné hyperploše
životní prostor společenství	hyperplocha daná účelovou funkcí
migrace členů společenství v životním prostředí	migrační kola v algoritmu SOMA

Tabulka 1: Význam biologické terminologie v algoritmu SOMA

Algoritmus pracuje podobně jako jiné evoluční algoritmy v cyklech zvaných migrační kola. Opět se zde setkáváme s jedinci z předem dané množiny - populace. Každý jedinec je ohodnocen účelovou funkcí (**fitness**). Ze všech jedinců populace vybíráme toho s nejvyšší hodnotou fitness funkce. Tomuto jedinci říkáme **leader.** Průběh evoluce je založen na kooperativním prohledávání prostoru možných řešení. Nemůžeme zde hovořit o vzniku nových jedinců, protože jedinec se pouze přemisťuje do „optimálnější pozice“ vzhledem k pozici

leadra. Noví jedinci se vygenerují na základě spercimenu a hned se pouští do prohledávaní plochy. Vydají směrem k leadrovi po krocích (STEP). V každém novém kroku vypočítáme hodnotu účelové funkce, kterou srovnáváme s dosavadní nejlepší hodnotu a zjišťujeme zda-li je nová pozice vhodnějším krokem evoluce. Pokud ano, uložíme tuto hodnotu do paměti jedince (pBest) případně rovnou do paměti populace (gBest). Po skončení migračního kola se jedinci vydají na pozice s nejvyšší hodnotou fitness funkce. Dle těchto hodnot určíme nového leadra a opakujeme migrační kolo. [8] [10]

Parametry určující chod algoritmu můžeme rozdělit na tzv. řídící a ukončovací. Řídící parametry mají vliv na kvalitu běhu algoritmu (z hlediska hodnoty účelové funkce) a ukončovací jsou ty které za předem nadefinovaných podmínek algoritmus ukončují.

Parametrem určujícím nám kvalitu posuvu (tzv. „zrnitost“) je krok (STEP). Kvalita tzv. mapování množiny všech přípustných možností je dána nastavením kroku, prostor podrobněji prozkoumáme při menších hodnotách. Vhodné na hledání lokálních extrémů, v praxi používáme STEP > 1. [10] Dalším kritériem je parametr (MASS), který nám určuje relativní velikost překročení vůdce. Ten nám určuje jak daleko se aktivní jedinec zastaví od leadra. Posledním řídícím parametrem je (PRT), tzv. pertubace. Ten nám zavádí stochaickou složku algoritmu, díky které se nechová celý algoritmus deterministický. Pertubaci si můžeme představit jako vektor jež nám bude určovat směr posuvu vůči leadrovi.

S ukončovacími parametry jsme se setkaly už v předešlých algoritmech, proto si je pouze popíšeme a bližší informace nalezneme v. [8] Jsou jimi parametry určující velikost populace, počet migračních kol a počet argumentů účelové funkce.

V současné době existuje několik variací základního algoritmu SOMA, pro jejich obecné označení používáme výraz strategie. Všechny verze jsou navzájem prakticky plně porovnatelně ve smyslu hledání globálního extrému. Popíšeme si základní strategie:

1, All to one (viz. výše popsaný algoritmus)

2, All to All (není žádný leader, algoritmus vychází z každého z jedinců jako vůdce, to umožní prozkoumat větší prostor.)

3, All to All Adaptive (pracuje podobně jako alg. All to All jen s tím rozdílem, že migrující jedinec se přesouvá do nové pozice až po všech migracích)

4, All to One Random (pracuje jako zmíněná strategie 1. ale s tím rozdílem že leader se nevybírá na základě účelové funkce, ale náhodně)

5, Cluster (vytváření svazků můžeme použít pro jakoukoli z těchto strategií. Princip spočívá v rozdělení jedinců do jednotlivých svazků. V každém tomto svazku probíhá právě ještě jeden samostatný alg. SOMA, díky čemuž je možné vytváření a rozpadání nových svazků)

2.5 ROJENÍ ČÁSTIC

Také se označuje Particle swarm optimalization (PSO), jedná se o optimalizační techniku pracující podobně jako algoritmy SOMA či DE . Tato technika byla vyvinuta R. Eberhartem a J. Kennedym v roce 1995. Algoritmus se inspiroval chováním živočišných společenstev - ptačích a rybích hejn. Pro jeho lepší pochopení si představme hejno ptáků, kteří hledají vrchol hory. Každý pták se nachází v určité nadmořské výšce. Společně hledají vrchol hory, který je v nejvyšší výšce.

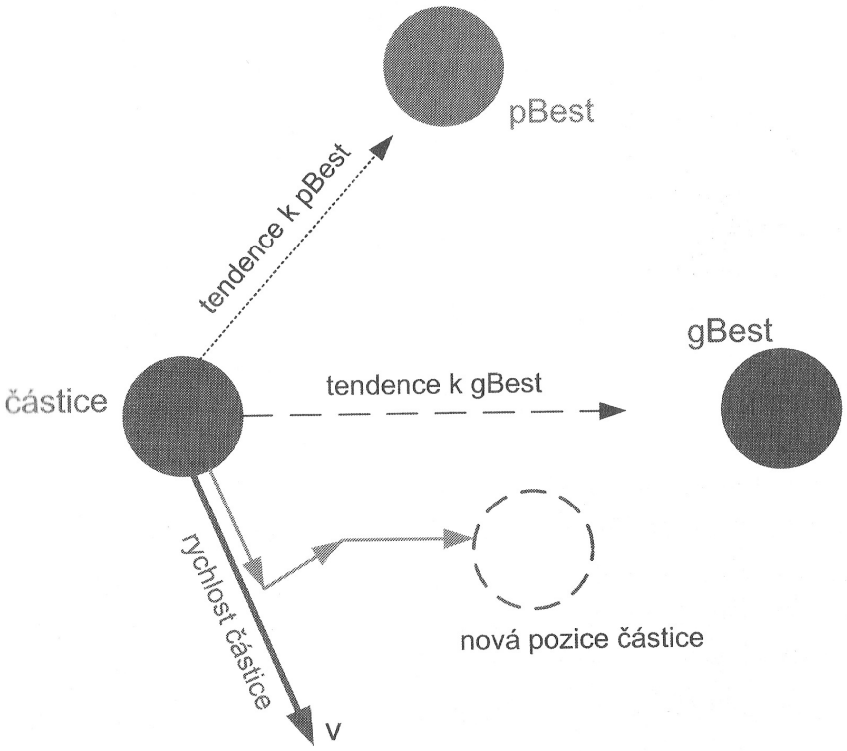
Pták je vlastně členem populace, v algoritmu PSO se jedinec nazývá částicí. Každá částice je reprezentována vektorem rychlosti, souřadnicí ve vyhledávacím prostoru a svou zatím nejlépe dosaženou pozicí – o které hovoříme jako (pBest). Můžeme si ji představit jako zjištěnou nejvhodnější variantu řešení, která závisí na hledaném globálním extrému. Jedinec s nejvyšší hodnotou účelové funkce uloží svou současnou pozici do společné paměti populace. Každý jedinec pak má přehled, kde se tato pozice nachází – v praxi bývá označena gBest (globalBest). V průběhu každého posuvu porovnává jedinec hodnotu účelové funkce se svým dosaženým maximem z předešlých pozic. Při jeho překonání ukládá aktuální pozici jako pBest (personalBest). Trajektrorie částic se určuje dle *vzorce č. 3* , vektor rychlosti posuvu zjišťuji na základě dosazení hodnot pBest a gBest.

$$v_d(t+1) = v_d(t) + c_1 \cdot rand \cdot (pBest_{i,d} - x_{i,d}(t)) + c_2 \cdot rand \cdot (gBest_d - x_{i,d}(t)) \quad (2)$$

$$x_{i,d}(t+1) = x_{i,d}(t) + v_d(t+1) \quad (3)$$

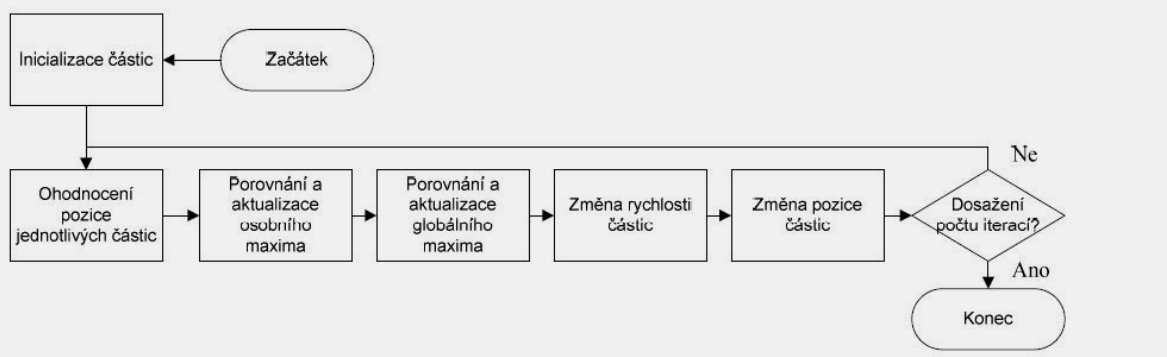
$v_d(t)$ - rychlost jedince v tomto kroku ; $v_d(t+1)$ - rychlost jedince v následujícím kroku ; $rand$ - náhodné číslo v intervalu (0,1) ; c_1, c_2 - učící faktory ; $x_{i,d}(t)$ -pozice jedince v tomto kroku ; $x_{i,d}(t+1)$ - pozice jedince v následujícím kroku. [10]

Pro lepší představu je princip hledání nové částice znázorněn na *obrázku číslo 9*.



Obrázek 10: Princip rojení částic

Princip algoritmu je vlastně jednoduchý - spočívá v náhodném výběru jedinců z populace. Těmto částicím (jedincům) je vygenerován a přidělen vektor rychlosti (*vzorec č. 2*). Na základě jedincovy komunikace s populací je zjištěna informace o nejvhodnější pozici částice (gBest). Nyní jedinec zná vše potřebné (vektor rychlosti, pBest a právě gBest), aby mohl přistoupit k další iteraci. Ta vychází z přesunu jedinců na nově vyhledané pozice, kde zkouší její vhodnost pomoci fitness funkce. Celý tento proces se aplikuje na všechny částice. Porovnání hodnot nám vniká nova generace jedinců. Po dosažení požadovaného množství cyklů algoritmus ukončíme. Vývojový diagram je znázorněn na obrázku č. 10.



Obrázek 11: Diagram procesu optimalizace PSO [13]

K správné funkci PSO je třeba nastavit základní parametry, se kterými tato metoda pracuje. Jmenovitě nastavujeme parametry - práh příjmení (CR – [0,1] ; dimenze systému (D) ; velikost populace (NP) ; mutační konstantu (F – [0,2]) ; generation . Celý algoritmus má smysl pouze při podmínce Generations > 0.

Algoritmus byl nadále vyvíjen a dostal se několika modifikací, s nejznámějšími se seznámíme. Zejména užívanou je varianta PSO –neighborhood (sousedství)– více nejen o této variantě nalezneme v publikacích prof. Zelinky. [10]

1,PSO- neighborhood (tato modifikace algoritmu přišla chvíli po jeho prezentování, Snaží se snížit riziko předčasné konverze, která byla jednou z hlavních nevýhod PSO. Bylo nutné přidat další parametr, počet sousedů – pracují nejčastěji v kruhové topologii. Každý jedinec sousedí právě s těmi jedinci, kteří jsou mu svým číslováním nejblíže. Sousedství jsou navzájem propojená a všichni nepřímo ovlivňují všechny)

2,Speciation PSO (pracuje s izolovanými skupinami jedinců v průběhu iterací se volí vůdci jednotlivých skupin, v průběhu dalších iterací se můžou stát vůdci i jiní jedinci)

3,Niching PSO (spočívá ve vygenerování náhodného hejna které je rozptýleno po prohledávaném prostoru. Částice která najde nejlepší řešení vytvoří skupinku z okolních částic a důkladněji prohledají okolí)

4,Dynamic Neighbourhood PSO (částice tvoří nezávislé sousedství, toto sousedství se vyvíjí dle dané pozice jedince, vytváří se nová sousedství stará zanikají)

II. PRAKTICKÁ ČÁST

3 TVORBA ANIMACE

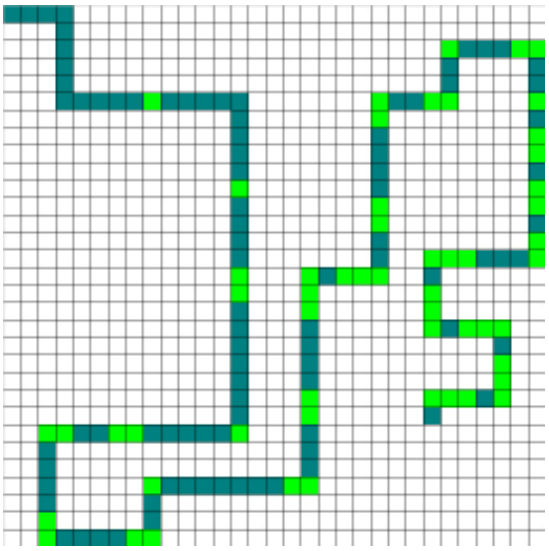
Mým úkolem bylo vytvoření dvou animací pro možnosti prezentování analytického programování. První z nich se týká samotným principem AP. Pro realizaci jsem zvolil plošnou animaci. Druhou animací je vizualizace stezky Santa Fe, která je prezentována v mnohých publikacích o evolučních algoritmech. Na této stezce je ukázán algoritmus Analytického programování prezentovaný v publikaci Evoluční výpočetní techniky - Oplatková, Zelinka. [8]

Animace jsou vytvořeny v různém softwaru (Adobe After Effects, Make Human, Corel a Blender), kde jsem se snažil upřednostnit open source. Výsledná animace bude syntézou práce v jednotlivých programech. V úvodu praktické části bakalářské práce se nejprve seznámíme s problémy, které jsou ztvárněny v animacích a až poté přistoupíme k jejich samotné realizaci.

3.1 TRAIL SANTA FE „(Stezka SANTA FÉ)“

Stezka SANTA FÉ vznikla pro účely porovnání analytického programování s genetickým. Je to stezka pro umělého mravence, kterého [Koza, 1998] definoval ve svých simulacích v genetickém programování. Úloha umělého mravence se zabývá pohybem v prostoru a sbíráním jídla na určité stezce a vyhýbáním se překážek. Tento přístup může být aplikován i na robota, který se má v určitém prostoru pohybovat.[8]

Santa Fe stezka je definovaná jako pole 31 x 32 políček (obr č. 11), modrá políčka obsahují potravu (tzn. něco, co má být sebráno). Ostatní políčka nic neobsahují, slouží pouze jako překážky. Zelená políčka slouží pro vykreslení výsledné dráhy robota. Robot se pohybuje tak, že se podívá před sebe a pokud je před ním potrava vykoná pohyb vpřed. V případě, že před ním jídlo není tak se otáčí ve směru hodinových ručiček. Po dosažení původní pozice se začne celý algoritmus hledání znova.



Obrázek 12: Stezka Santa Fe

Analytické programování má tak před sebou úkol najít systém pravidel, který umožní přeskakovat i takové překážky. Do základní sady byly použity následující funkce.[6]

1, Patřící do podmnožiny funkcionálního prostoru s 0 argumenty – GFS_{0arg}

Move – postup o pole vpřed

Left - otoč se vlevo

Right - otoč se vpravo

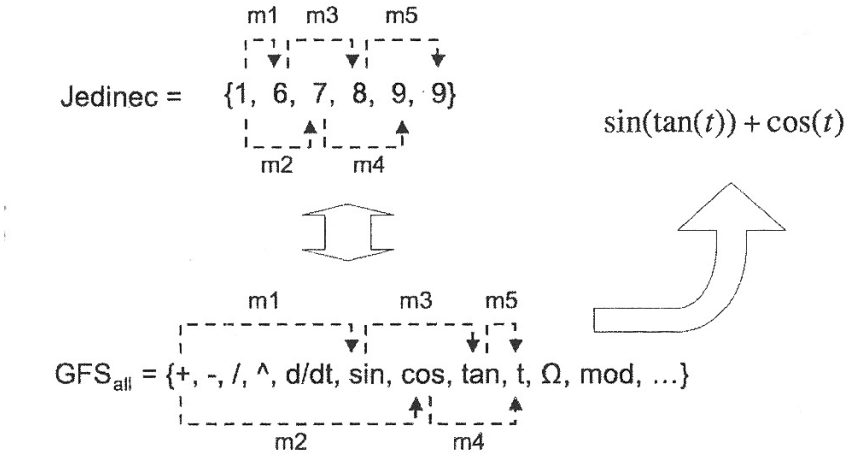
2, patřící do podmnožiny funkcionálního prostoru se 2 argumenty – GFS_{2arg}

IfFoodAhead - podívej se na pole před sebe, pokud se tam nachází potrava, udělej to, co je na pozici True, jinak udělej, co je na pozici False. Před spuštěním evoluce, nemůžeme vědět, co se na pozici True nebo False bude nacházet. Funkce se tam doplňují až v průběhu Analytického programování.

Prog2 a **Prog3** -slouží pro rozvíjení programu. Tyto příkazy jsou dodávány až v průběhu evoluce obdobně jako pro IfFoodAhead.

3.2 ANIMACE PRINCIPU AP

Vizualizace principu Analytického programování je v teoretické části práce na (*obr. č. 3)* , kde skládáním optimalizované funkce nacházím řešení na konkrétnímu jedinci. Úkolem je názorné vytvoření výsledné funkce pomocí jednotlivých kroků optimalizace. Po konzultaci s vedoucí práce byly navrženy příklady na kterých byla princip prezentována. Na základě komunikace s vedoucí práce o daném problému byla dolazena konkrétní představa o zobrazení tohoto problému.



Obrázek 13:Princip Analytického programování

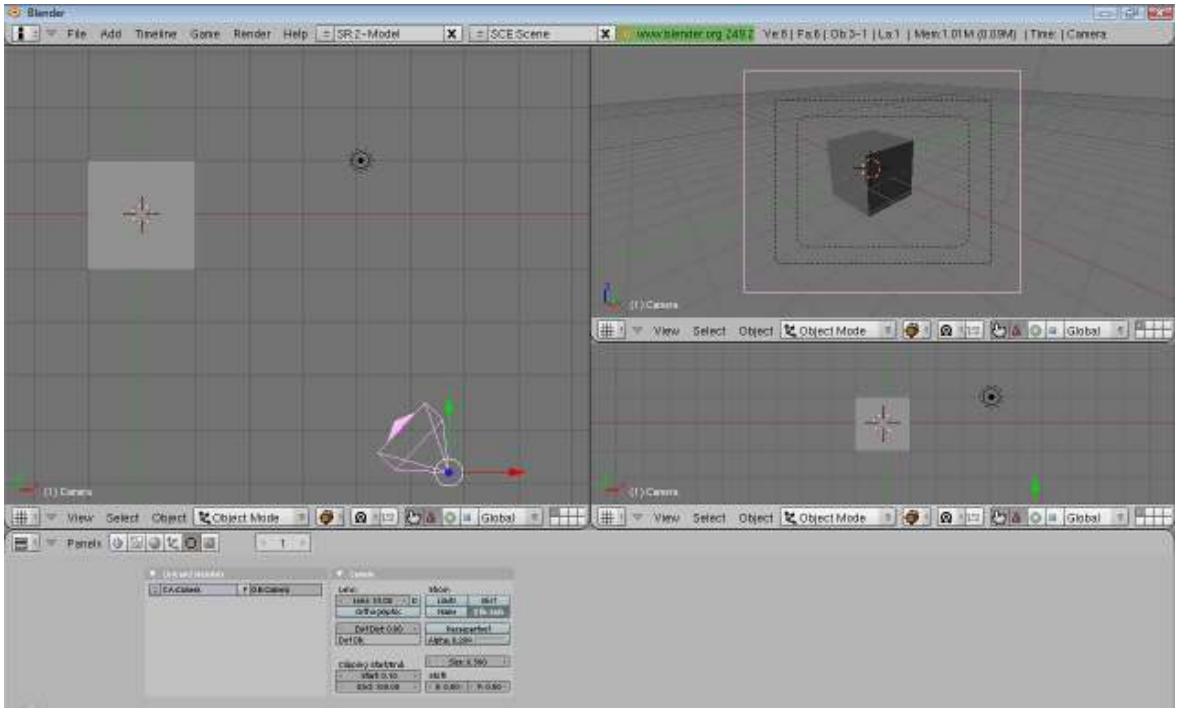
Princip je zobrazen ve dvou částech animace, kde jsou ukázány vhodné i nevhodné příklady dosazení do optimalizované funkce.. V úvodu animací jsou zobrazeny jednotlivé množiny funkcí **(GFS)** a z nich je vytvořena množina všech funkcí GFS_{ALL}. Do množiny jedince je vylosován náhodný obsah, který pak indexově ukazuje na použité funkce z celkové množiny funkcí. Výslednou syntézou vzniká optimalizovaná funkce.

4 BLENDER

Blender je software pro 3D modelovaní, animaci, tvorbu her, rendering a přehrávání. Je to program, který je založen na knihovně ***OpenGL***. [7] OpenGL umožňuje výhodnou hardwarovou akceleraci při vykreslování 2D a 3D objektů, ale především zaručuje snadnou přenositelnost na všechny podporované platformy. Jedná se o multiplatformní program, což znamená, že jde nainstalovat na různé operační systémy (Windows, Linux, Mac OS nebo OS X).

Hlavním důvod proč jsem si zvolil právě Blender byl, jelikož se jedná o bezplatný software. Patří mezi volně šířitelný software do skupiny programů nazvaných Open source (OS či OS software). První zkušenosti s tvorbou 3D grafiky jsem získal na střední škole, kde jsme pracoval s program 3D MAX (Autodesk). S Blenderem jsem prvně setkal zde na fakultě ve 4.tém semestru v předmětu Grafika. Přišel mi jako jako vhodná nekomerční náhrada s širokou paletou nástrojů - od tvorby modelů přes UV mapping až po vlastní animaci. Po důkladnějším prozkoumání, na mě čekalo nemilé zjištění, že nastavení klávesových zkratek je defaultně nastaveno jinak než-li u jiných programů. Bohužel v této verzi nebylo možno je editovat. Nezbylo nic jiného než-li si zvyknout, protože licence Autodesk 3D MAX stojí řádově desetitisíce. Pracoval jsem s verzí Blender 2.49b, kde oproti starším verzím nalezneme propracovanější práci se Python skripty a řadu jiných vylepšení. V podstatě je tato verze srovnatelná s tou, se kterou jsem prvně pracoval na fakultě. Rozdíl je markantní až u nově zveřejněné verze 2.5 Alpha 2, ve které se celkem podstatně změnil základní interface (více naleznete na www.blender.org).

K samotnému prostředí verze 2.49b – jedná se o velice variabilní interface, kde je možno si rozvrhnout pracovní plochu dle individuálních přání uživatele (obrázek č. 12). Rozdělením vytváříme samostatná okna, kde každému oknu můžeme nastavit různé funkce zobrazení. Volíme mezi



Obrázek 14: Ukázka prostředí v blendru 2.49b

4.1 MODELOVÁNÍ

Blender nám nabízí široké spektrum předdefinovaných 3D objektů, které můžeme použít při modelování (např. krychle, koule, kužel, aj.). Tyto objekty se označují jako **mesh**. Ke každému objektu typu **mesh**, můžeme přistupovat na různých úrovních, a to - pomocí jednotlivých bodů (vertexů) či znázorněním celých hran a ploch. To nám poskytne dobrou možnost editování objektu při samotném modelování.

Dalším typem objektů se kterými můžeme pracovat jsou křivky a texty. Křivky nám umožní přesné modelování 2D objektů, ze kterých mohou vzniknout oblé tvary ve výsledné scéně.

Samotné modelování může vznikat různými způsoby, od vkládání jednoduchých objektů až po tvorbu složitých tvarů pomocí různých nástrojů. V základu bych rád představil modelování pomocí základních nástrojů v EDIT MODU (TAB). Pak bych Vás rád seznámil s funkcí tzv. MODIFIKÁTORŮ (Modifiers).

4.1.1 NÁSTROJE PRO MODELOVÁNÍ V PROGRAMU BLENDER

Nástroje můžeme použít pro tvorbu rozmanitých objektů. Pro jejich užití musíme být v **EDIT MODU**, do kterého se přepínáme pomocí klávesy **TAB**. Po přepnutí dostáváme možnost pracovat jednotlivými vertexy/hranami/plochami. Pomocí klávesy **B,** si můžeme jednoduše vybrat jednotlivé body, se kterými budeme chtít ovlivňovat. Pro samotnou editaci vybraných vertexů nám slouží tyto nástroje.

Extrude - Tento nástroj můžeme volně přeložit jako tažení. Nalezneme jej pod klávesovou zkratkou **E**. Nástroj vytvoří kopii všech vybraných vertexů a vazeb mezi nimi (hran, ploch) a každé dva odpovídající si vertexy spojí hranou (případně uzavřené čtyři hrany plochou). Je to nejpoužívanější nástroj v blenderu , potřebný prakticky při každém modelování. Vybíráme z různých voleb extrudování – táhnout můžeme jak pouhé vertexy tak i celé hrany.

Subdivide - je funkce, která vytvoří v objektu nové vertexy, a to tak, že každou vybranou hranu rozdělí na dvě části. Nalezneme ji v nabídce pod se klávesovou zkratkou **W** (nebo záložce Mesh tools). Dané dělení můžeme měnit dle našich potřeb, tzn. může být o daném kroku aj. Defaultně je nastaveno na rovnoměrný dělení – tzn. vkládání přesně mezi vybranými vertexy.

Bevel - Tento nástroj slouží pro zaoblení ostrých hran objektu. Zaoblení se vytvoří přidáním nových hran/vertexů. Kvalitu zaoblení zadáváme parametrem Recursion. Bevel najdeme v nabídce vyvolané stiskem klávesy **W**.

Boolean – vychází z klasických operací používaných v Boolově algebře. Setkáváme s nástroji ***Intersect, Union*** a ***Difference*** (průnik, sloučí, rozdíl).

K samotnému užití tohoto nástroje potřebujeme vždy dva objekty. Výsledná operace bude provedena na základě pořadí vybrání objektů. Nástroj je k nalezení opět v nabídce pod kl. zkratkou **W**.

4.1.2 MODIFIERS

Další kapitolou o které jsem se chtěl zmínit jsou modifikátory. Chtěl bych vyzdvihnout tuto možnost práce s objekty v prostředí Blenderu. Při jejich použití ovlivňujeme výslednou podobu objektu a můžeme vytvářet co si naše fantazie zamane. Modifikátory jsou s námi od verze blenderu 2.40, kde byly přidány pouze ty základní. V současné verzi se můžeme setkat s celou řadou nových, které jsou nyní sjednoceny pod záložkou **Editor buttons (F9)** v tzv. ***„Modifier Stack".***

Největší výhoda modifikátorů je v tom, že kdykoliv můžeme libovolný modifikátor přidat či odebrat. Pořadí, v jakém budeme jednotlivé modifikátory na daný objekt aplikovat, ovlivní výsledný efekt. Pořadí můžeme libovolně ovlivnit tlačítky šipek vedle názvu. Seznámím Vás alespoň s těmi základními.

Subsurf - tento modifikátor nám slouží k zaoblení objektu, které se daleko více vykreslí než-li použitím pouhého „Set Smooth". Jeho výhodou je že nepřidává nové vertexy, ale pouze několikanásobně rozdělí plochu objektu a poté teprve zaobluje. Při užití tohoto modifikátoru si musíme uvědomit, že velikost výsledného objektu bude menší než-li toho původního. Nastavením parametrů (**level**) lze ovlivnit výslednou kvalitu vykreslení.

Armature – neboli nástroj pro vytváření kostry, provazování jejích jednotlivých částí, ringing a následnou animaci prvků. Pro vytvoření kostry potřebuje přiřadit rodičovský objekt, který bude kostra ovlivňovat. Toho dosáhneme pomocí vybráním obj a kostry - **CTRL + P** (Make Parent). Výhodou rodičovské vazby je, že jednoduchým přiřazením

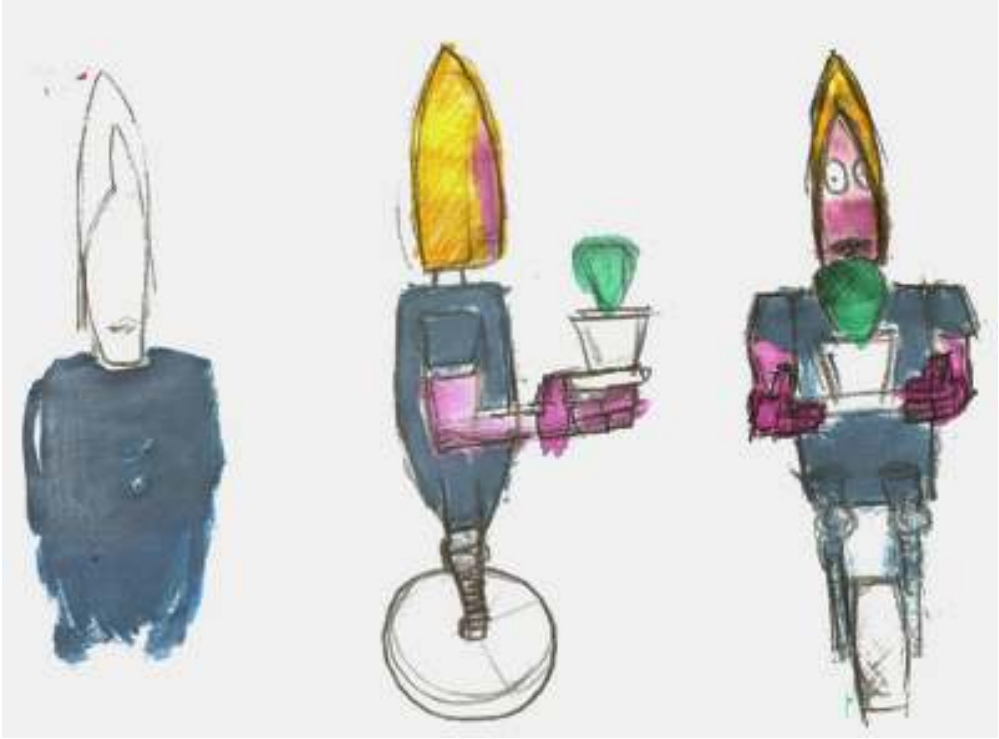
potomka zajistíme, že potomek se bude řídit dle kosti rodičovské. Každé kosti můžeme nastavit oblast meshe, kterou má ovlivňovat. Toto nastavení děláme pomocí funkce **Weight Paint.** Nástroj armature by stál za hlubší prozkoumání, ale bohužel to není možné. S tímto nástrojem se ještě setkáme při ukázce tvorbě modelu robota. Více o něm naleznete na (3Dgrafika.cz).

Mirror – při výrobě osově souměrného modelu, jistě každý ocení právě tento modifikátor. Základním parametrem je nastavení osy kolem které chceme zrcadlit. (Pozn. tento modifikátor byl použil při tvorbě armatury Dummyho,ale o tom až za chvíli)

Curve – slouží pro deformaci objektu křivkou. Samotné použití je jednoduché, výběrem si objekt který má být ovlivněn - přidáme mu modifikátor Curve a do položky **Ob:** napíšeme jméno křivky. Příkladem použití tohoto modifikátoru budiž nastavení pohybu kamer po křivce.

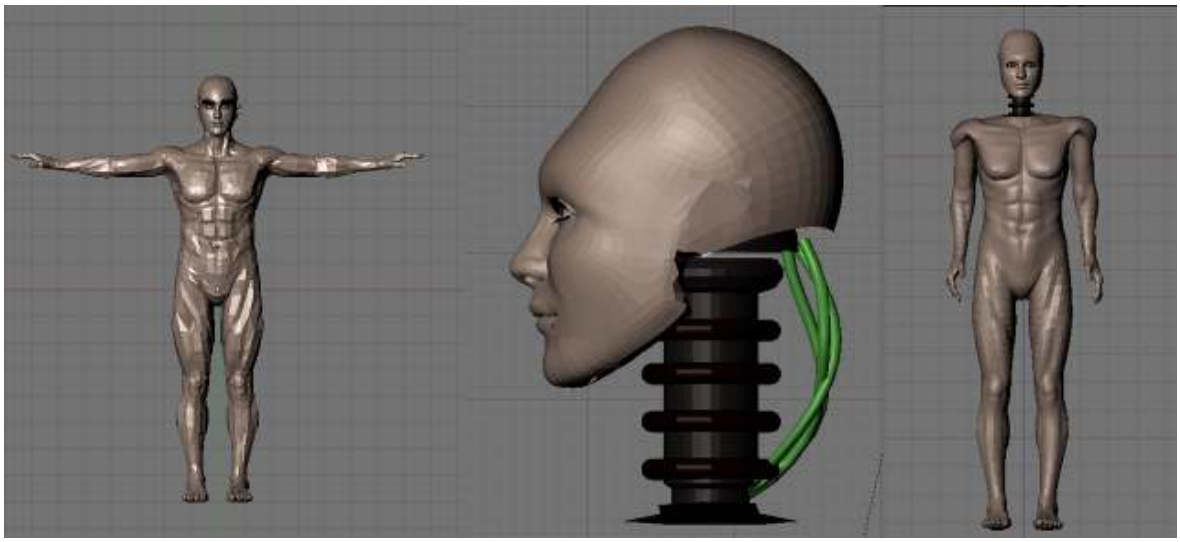
4.2 MODEL DUMMY

Po prvních shlédnutých tutorialech jsem zjistil, že Blender mi umožní vytvoření postavy přesně dle mých představ. Celkem dlouho mi trvalo, než jsem vymyslel konkrétní podobu mého modelu. Pro animaci stezky „Santa Fe“ (oplatková, zelinka) jsem se rozhodl použít model robota.



Obrázek 15: První výtvarné návrhy

Vytvoření robota bylo realizováno v programu Make Human (MH). Je to další z programů rodiny open source, takže jeho lincence je freeware. Tento program přináší možnost velice snadno generoverovat lidské postavy. MK umožňuje výborné možnosti editace vygenerované postavy. Je možné nastavit pohlaví, výšku, váhu, věk a spoustu parametrů obličeje a hlavy. Z programu MK je možnost exportovat v různých formátech. V tomto případě byl model exportován ve formátu **.OBJ** . S tímto formátem spolupracuje i Blender. [17] [20]



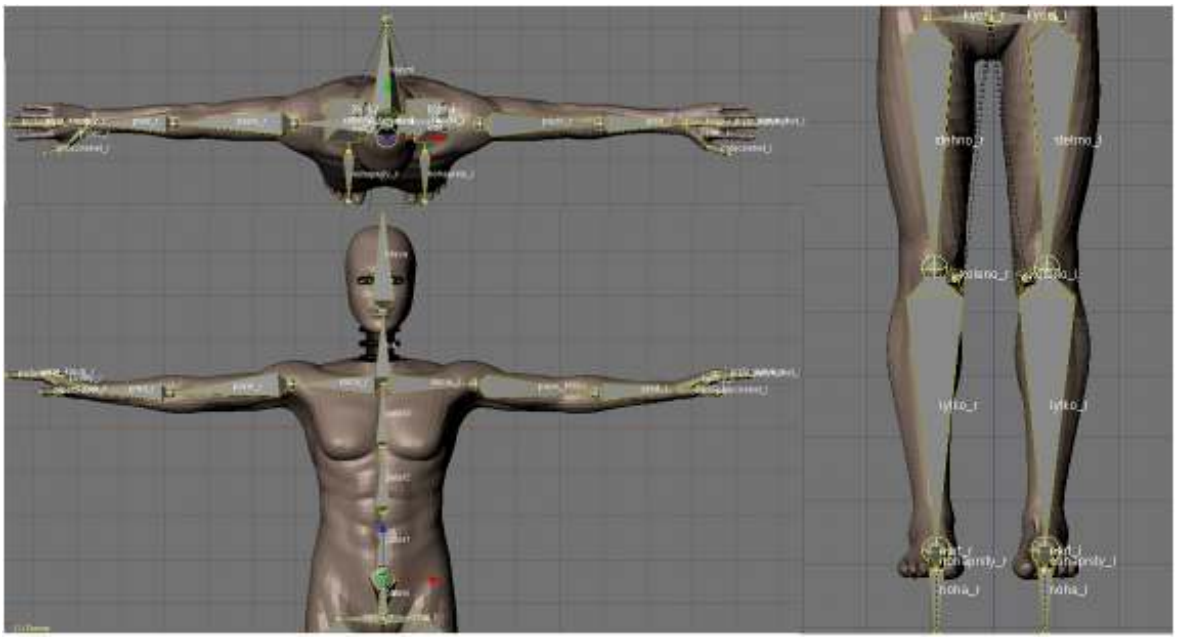
Obrázek 16: Jednotlivé fáze při tvorbě robota Dummyho

Na obrázku č. 14 jsou zobrazeny jednotlivé fáze výroby modelu. Na první fázi je zobrazen robot při importu do programu Blender. Na dalších jsou zobrazeny další kroky výroby robota.

První úpravy byly uskutečněny na hlavě, kde bylo upraveno obočí a odstraněny uši (*obr č. 14 - fáze 2*). Tyto úpravy byly uskutečněny v Editačním modu (**Edit Mode**) za pomoci základních nástrojů. Nevhodné vertexy byly smazány (Výběr + **X**) a vymazané plochy byly doplněny pomocí funkce Make faces (**F**, případně **CTRL + F**). Přesto robot vypadal „moc lidsky". Volba padla na vytvoření mechanického krku, kde byla snaha přiblížit se představě robotů jež testují automobily. Tento krok byl realizován pomocí sloučení základních objektů typu mesh (**CTRL + J –** join meshes). Samotná náhrada krku probíhala podobně jako při editaci ucha. Vymazáním nevhodných vertexů se vytvoří prostor pro vložení vymodelovaného krku. U něj pouze upraví velikost (**SCALE**) a může se vložit na dané místo. Krk, hlava a tělo nemusí být spojeny, protože model bude ovládán pomocí Armatury. Výsledný model je zobrazen na třetí fázi *obr. č. 14.*

4.2.1 ARMATURE

Dalším krokem je samotné rozpohybování robota Dummyho. To se uskutečňuje pomocí vytvoření armatury, která ovládá jeho pohyb. Na dané téma byla zpracována bakalářská práce Tomáše Vilímka (UTB – FAI 2008, Animace chůze ženy v blenderu, [17] Ve zkratce si ukážeme nejdůležitější kroky, kterými probíhá výroba .



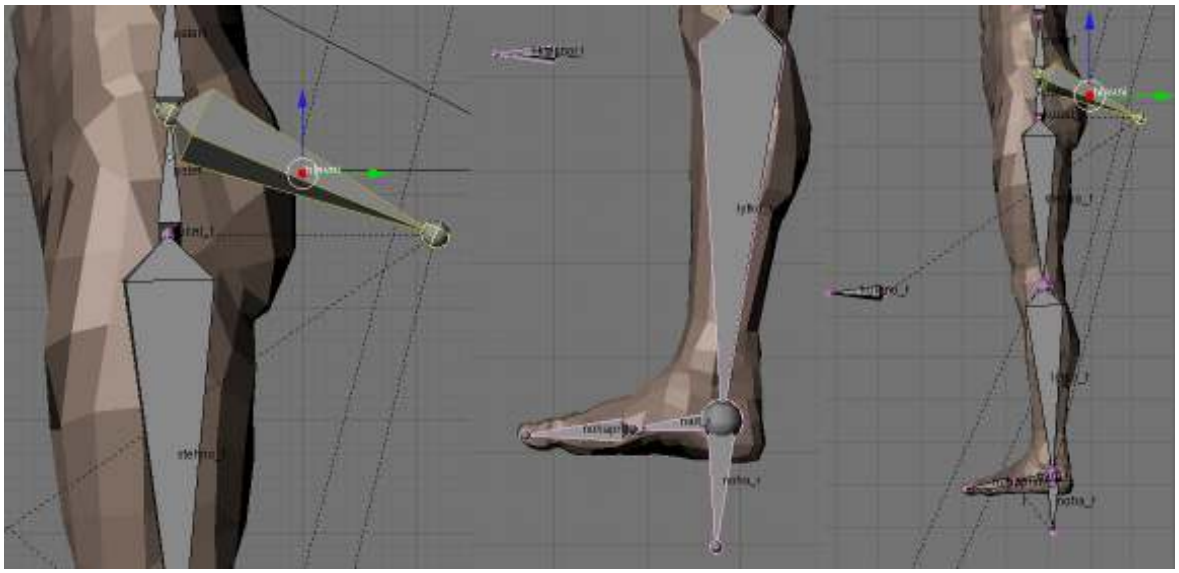
Obrázek 17: Ukázka armatury a jejích vazeb

S vytváření armatury se začíná od pánve, kde se pokládá první kost. Další části armatury jsou vytvářeny pomocí nástroje **Extrude** (**E**), který lze používat pouze v Edit modu. Tímto postupem se rozšiřuje objekt o jednotlivé kosti. Výhodné je využít modifikátor **Mirror**, který umožní vytváření osově souměrných kostí při modelování. Stačí vytvořit jednu párovou končetinu a druhá se zduplikuje dle středu otáčení. Při tomto typu rozšiřování modelu dochází k jejich provázání rodičovskou vazbou. Prvek z nějž jsme extrudovali se stává rodičem prvku extrudovanému. Rodičovskou vazbu si můžeme volně představit jako spojení dvou prvků, ve kterých dle jednoho prvku se řídí ten druhý. (Tzn. při

pohybu prvního prvku se druhý dostává do koncové pozice prvního a z ní pokračuje ve svém vykonávaném pohybu.)

Obrázek 18: Armature bones - nastavovaní rodičovské vazby v blenderu

Po vytvoření jednotlivých vazeb mezi kostmi už jen zbývá dodělat řídící kosti nohou a hlavní kost. Ty zjednoduší a zkvalitní rozpohybování robota. Hlavní kost je vytvořena extrudováním z prvního článku páteře (*obr 17. první fáze*). Tato kost zastupuje pohyb celé armatury robota. Proto se musí přiřadit jako rodič kosti, ze které jsme ji extrudovali (pouze stačí prohodit vazbu rodič-potomek). Dalším krokem je vytvoření řídících kostí nohou, které jsou vytvořeny obdobně. Byla extrudovány z „kotníku" a tím byla vytvořena řídící kost nohy (**noha_r)** *(fáze 2 obr. 17)*. Bude sloužit pro ovládání pohybu dolní končetiny. Dále kost byla provázana se stehenní kostí, aby pohyb dolních končetin by vypadal nepřirozeně. Toho bylo dosáženo pomocí tzv. **Constrains**, které zaručí provázanost vybraných kostí armatury. Pomocí řídících kostí (noha_r, noha_l) je teď možné ovládat pohyb celé končetiny. [20]

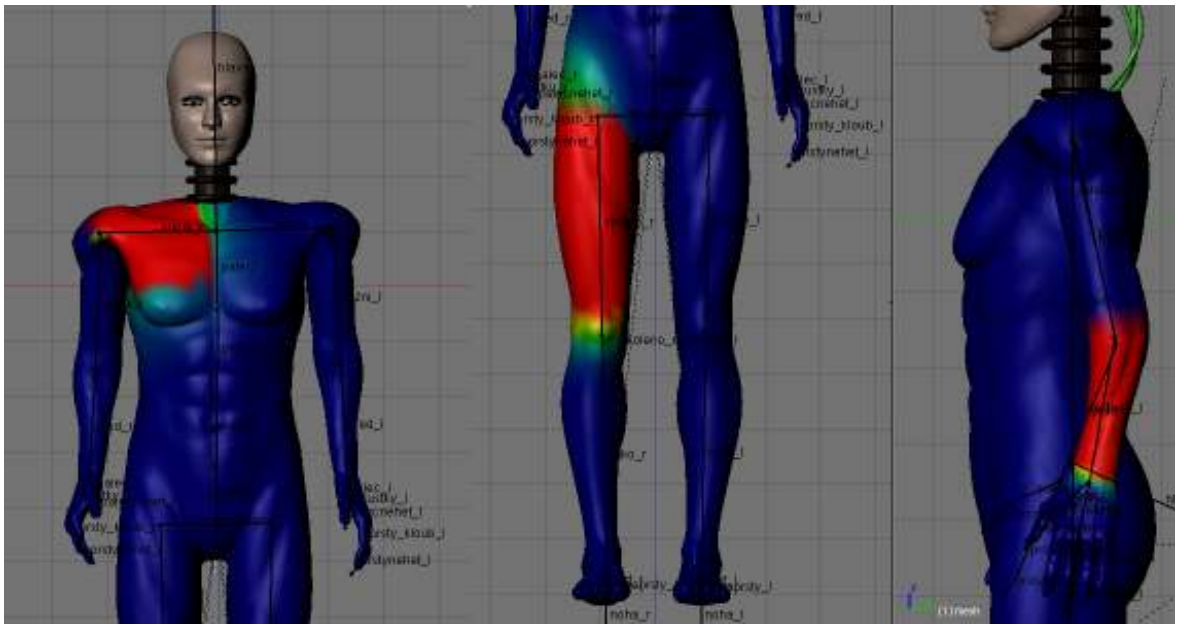


Obrázek 19: Řídící kosti pro pohyb Dummyho

V tento moment jsou všechny potřebné kosti pro ovládání pohybu robota připraveny. Společně s kostrou byly vybrány jednotlivé části Dummyho (hlavu, krk, trup) a pomocí nástroje **Make parent (CTRL + P)** byla vytvořena vazba mezi kostrou a robotem.

4.2.2 WEIGHT PAINTING

Posledním krokem pro úspěšné rozanimování robota je nastavení váhy jednotlivým kostem. Lidé používají kosti pouze jako oporu a celý pohyb vykonávají svaly, v 3D grafice to tak není. Jednotlivým kostem je potřeba přiřadit váhu, na základě toho budou dané kosti ovlivňovat deformaci postavy.



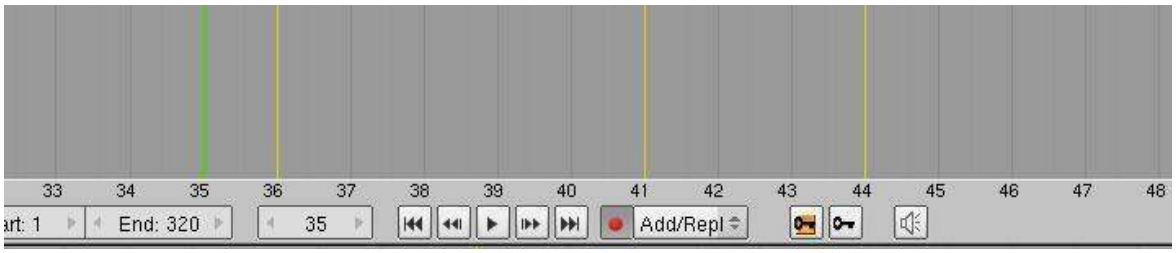
Obrázek 20: Přiřazování váhy jednotlivým kostem (Weight painting)

Blender po provázání postavy s kostrou přiřadil automaticky každé kosti nějakou váhu. Toto přiřazení bylo nepřesné, proto bylo nutné ho vyrobit znovu. Každé části armatury je přiřazena část modelu, kterou by měla ovlivňovat. Tohoto efektu se dosahuje ve **Weight paint modu**, kde pomocí nástroje štětec se přiřazuje jednotlivým kostem váhu. Ta se projeví zobrazení dané úrovně barev (modrá – 0, červená - 1) na daném místě.

4.3 ANIMACE V BLENDERU

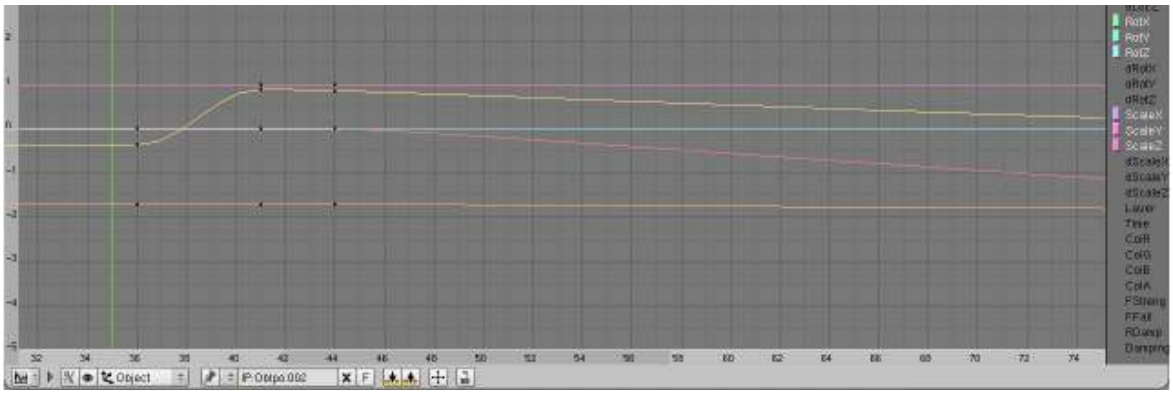
Podstatou animace je ukládání daných nastavení pro objekty pomocí animačních klíčů pro jednotlivé snímky animace. Blender umožňuje vkládat různé klíče pro objekty, materiály, textury, kameru nebo tvary objektů. Pro samotný pohyb objektu slouží klíč "Loc". Tento klíč je nutné vložit pro objekt na začátku pohybu pro první snímek, pak se nastaví konečná snímek kdy bude pohyb ukončen a definuje se místo, kde bude pohyb ukončen. Stejným způsobem se vkládají i klíče pro rotaci - "Rot" nebo zvětšení - "Scale". Blender automaticky dopočítá změnu hodnot daného parametru ve snímcích, které se nacházejí mezi počáteční a konečnou.

Nejjednodušší způsob jak animovat je pomocí zobrazeného okna **TIMELINE.** Jedná se okno zobrazující čas, do kterého můžeme vkládat jednotlivé animační klíče pomocí klávesy **I (Insert key),** kde můžeme vybrat mezi (LOC, ROT,SCALE). Druhým způsobem jak takto vkládat klíče, je pomocí **AUTO-KEY-MODE**). V případě změny parametrů objektu v rozdílu čase (Δt) se vloží animační klíč s uloženou hodnotou změny na aktuální pozici v TIMELINE (zelená čára)



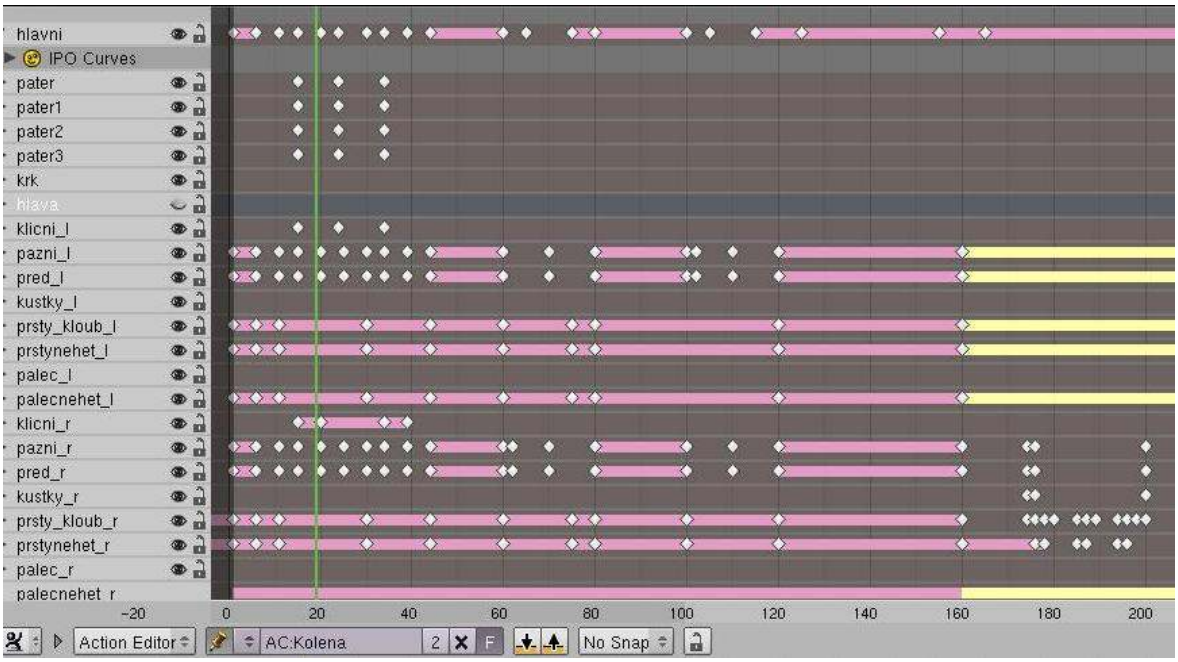
Obrázek 21: Timeline

V případě vložení většího množství klíčů by se stala scéna špatně čitelná, proto se Blenderu nachází **IPO editor**. V něm můžeme daleko efektivněji pracovat s animačními klíči. Do IPO editoru se dostaneme **SHIFT + F6** nebo pouhým přepnutím okna na typ **Ipo Curve Editor**.



Obrázek 22: Ipo Curve Editor

V IPO editoru můžeme měnit charakteristické hodnoty pohybu daného objektu. Křivku můžeme pomocí řídících bodů upravovat a dosáhnout tak nejideálnějšího posuvu mezi dvěma animačními klíči. Běžně se této úpravy používá při rozjezdu kamery a při dynamických záběrech.



Obrázek 23: Zobrazení pohybu v Action modu

V případě použití armatury se pohyb nejlépe animuje v tzv. **ACTION EDITOR**. Zde je výhodou, že se může pohybovat s každou kostí zvlášť a tak dosáhnout požadovaného pohybu. Práci v Action editoru usnadňuje užití

klávesových zkratek. Pohyb s jednotlivými key framy se zajistí pomocí klávesy **G**, frame se duplikuje pomocí **SHIFT + D**, případně jej můžu vymazat pomocí **DEL**.

4.3.1 RENDERING

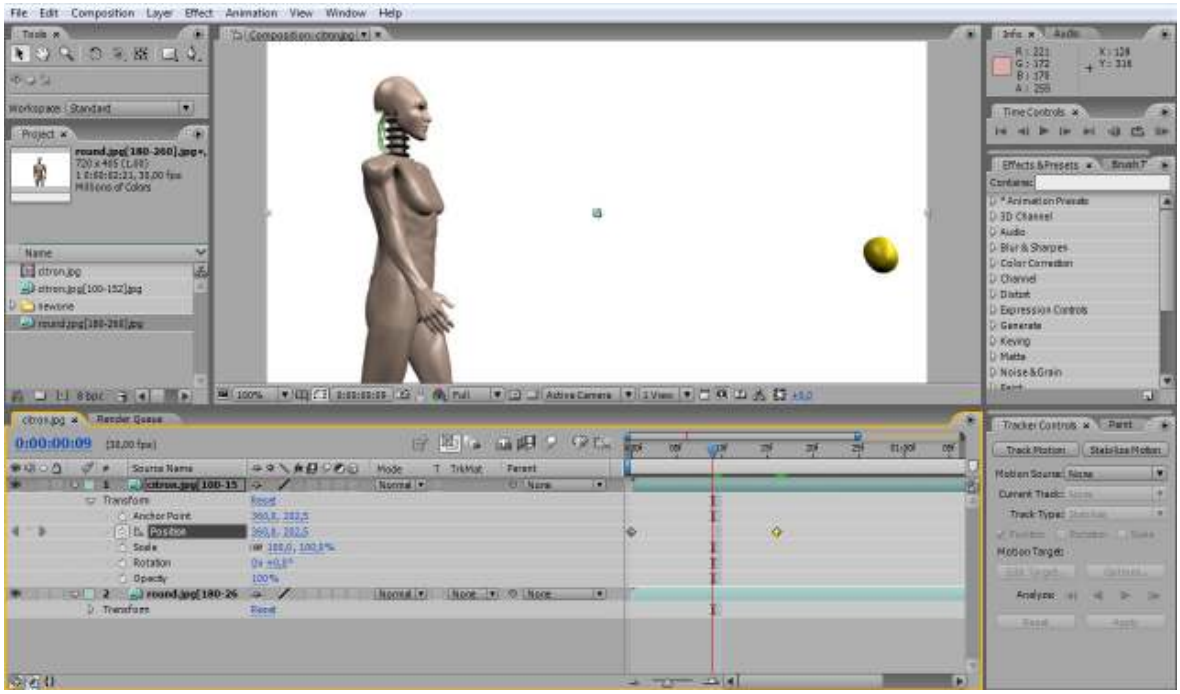
Rendering, tedy samotný export obrazu z programu. Na jeho nastavení bude záviset kvalita výstupu. Jedná se o nejnáročnější proces v průběhu chodu celého programu. Doba renderování přímo závisí na vstupních parametrech, které jsou nastavitelné v nabídce **render settings (F10).** Hlavními parametry jsou délka renderovaného úseku, rozlišení a množství vertexů na scéně. Nesmí se také opomenout nastavení vykreslení stínů, případně užití modifikátorů – všechny tyto parametry hrají roli při výsledném exportu obrazu.

Obrázek 24: Render settings

Rozlišení v němž byl projekt vyrenderován, bylo potřeba uzpůsobit náročnosti této operace. Původní myšlenka tvorby v HD rozlišení vzala brzy za své. Kompromisem se jevilo rozlišení PAL (720 x 405), kdy jeden frame se renderoval cca 20s. Výstupem byly sekvence obrázků (.jpeg). Obrázky byly zvoleny, protože se s nimi lépe pracuje v postprodukci , kde je nutné přizpůsobovat délku záběrů individuálním potřebám střihu. V případě zvolení kvalitnějšího renderingu, dosáhneme i vyšší kvality obrazu.

5 ADOBE AFTER EFFECTS

Adobe After Effects (AE) je počítačový program od softwarové firmy Adobe Systems. Tento software je použitelný pro širokou paletu „operací s obrazem“ , např. tvorbu 2D animací, speciálních filmových efektů či postprodukci audiovizuálních děl. Program je velice srozumitelný a práci v něm zvládne opravdu každý. Výhodou AE je, že podporuje import velkého množství formátů. [5]



Obrázek 25: Ukázka prostředí Adobe After Effects cs3

Na *obr. č. 23* je zobrazeno prostředí nemž byly vytvořeny animace. Samotná pracovní plocha je rozdělena na několik různých oken, kde každé z nich má svou specifickou funkci. Pokud má uživatel zkušenosti s produkty od firmy Adobe tak si na prostředí zvykne velice rychle. Pro vytvoření nového projektu je nutné vytvořit tzv. **Kompozici** (composition). Do této kompozice mohou být v daném pořadí nataženy jednotlivé vrstvy, které pak vytvoří požadovaný sled záběrů. Vrstvy jsou do projektu přidány jednoduchým přetažením z okna Projektu do okna Timeliny. V každé vrstvě jsou pod položkou Transform znázorněny konkrétní atributy vrstvy (**Duration, Rotation, Scale).**

Obrázek 26: Ukázka Timeline s jednotlivými vrstvami

Vhodným nastavením těchto parametrů může být vrstva změněna na požadované hodnoty. Každé vrstvě tak může být jednoduchým způsobem přiřazen efekt, který ovlivní výsledné její vykreslení. Tyto efekty jsou reprezentovány pomocí key framů a mohou být jednodušše kopírovány pomocí klávesových zkratek (**CTRL + V, CTRL + C).** Práce v okně Timeline je pak obdobná jako v prostředí Blenderu. Pomocí vkládání jednotlivých animačních klíčů, je možné vytvořit požadové nastavení scény. Na *obrázku č. 26* jsou zobrazena okna projektu a timeliny.

5.1.1 STŘIH

Neboli skládání jednotlivých záběrů - určující celkový děj na scéně. Jeho návaznost musí být logická. Divákovi by měla v každém záběru ukázat něco nového. Složení záběrů má svá pravidla, která se musí dodržovat, jinak se snadno divák mystifikuje. Mezi základní z nich patří střih přes osu či cílený rozvoj děje (gradace), která má také svá pravidla.

Na *obr č. 26* vidíme vytvořenou kompozici ve které jsou vloženy vrstvy. Každá vrstva má zvý razněmu svou duration (vpravo na *obr. č. 26*). Vhodným střídáním vrstev dosáhneme požadovaného střihu.

5.1.2 RENDERING A EXPORT

Vytvářené kompozice byly rendrovány cca po 30 sekundových úsecích. To jednak z důvodů náročnosti této operace, ale také protože tomto úseku se nachází kolem tří desítek vrstev a pak se kompozice stává práce nepřehlednou.

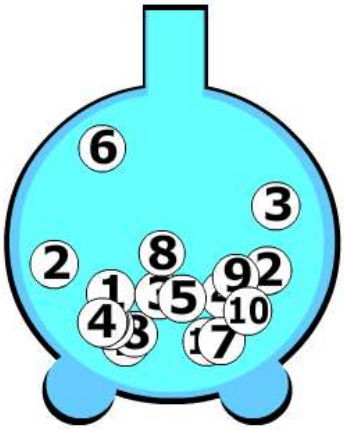
Renderingem v AE rozumíme dopočítání veškerých grafických operací, s nimiž počítáme pro výsledný export videa (např. prolínačky, zrychlení obrazu, freeze frame aj.). Poté následuje fáze přepočítání obrazu a vytvoření výsledného díla. Poslední krok je sloučení všech úseků. Takto seskupené dílo je už připraveno na zápis na CD - ROM.

5.2 PRINCIP AP

Pro samotnou realizaci této animace bylo nejprve nutné nastudovat informace o činnosti algororitmu Analytického programování. Na základě zjištěných faktů byly vytvořeny dvě animace popisující princip chování se algoritmu AP. Zobrazení této problematiky je provedeno pomocí 2D animace, která byla vytvořena v programu AdobeAfter Effects.

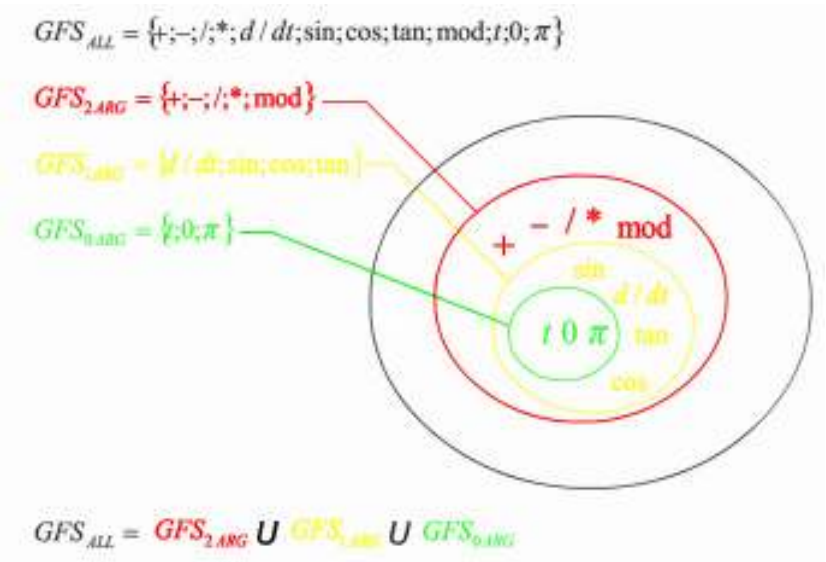
Plošná animace byla zvolena, protože lépe schématicky znázorňuje princip algoritmu. 2D animace bude působit na diváka jednoduššeji a bude mu lépe přibližovat jednotlivé kroky postupu. V první řadě bylo nutné samotný proces rozzáběrovat a stanovit si, které fáze jsou potřeba zdůraznit.

Pro diváka neznalého této problematiky, je potřeba zajistit, aby mu byly dostatečně vysvětleny principy AP. To spočívá už ve vytváření **jedince** a množiny všech funkcí **GFS.** Jak bylo v úvodu teoretické části zmíněno, tak vznik jedince je podmíněn náhodnému výběru. Pro jeho realizaci byla zvolena metoda podobná v televizní soutěži sportka – tzn. Losování. Prvky jedince jsou losovány po jednom a postupně zapisovány na uktuální pozici jedince. Náhodný výběr je zobrazen na *obr. č. 27*



Obrázek 27: Znázornění náhodného výběru

Další důležitou částí animace je samotný reprezentativní funkční výběr z množiny GFS. V animaci je prezentována syntéza funkčních množin GFS_{2ARG} , , GFS_{1ARG} a GFS_{0ARG}. Sloučením těchto množin vzniká sjednocená množina všech funkcí GFS_{ALL}. Princip jejího vzniku je zobrazen na *obr. č. 2.* V animaci je postup vzniku GFS_{ALL} znázorněn podobně jako na obrázku užitém v teoretické části.



Obrázek 28: Znázornění výroby funkční množiny $\mathrm{GFS}_{\mathrm{ALL}}$

Po předvedení vzniku symbolických objektů následuje už samotné zobrazení krokování optimalizované funkce. To funguje na základě spojení výběru jedince a celkové množiny funkcí. (tzv. indexace vybraných fcí). Prvním krokem postupu je vybrání prvního jedince, kde hodnou na prvním místě množiny jedince používám pro jako indexaci výběr funkce z množiny **GFS**. Vybranou funkci, terminál či operátor vložím do optimalizované funkce. Dalším krokem je vybrání následujícího prvku jedince a dosazení do množiny fci **GFS**. Z něj opět je vybrána funkce na kterou ukazuje index. U dalších kroků algoritmu se musíme zamyslet, protože syntéza optimalizované funkce musí být prováděna v přesně daném pořadí. Pokaždé je prvně volena další vybraná fce jako argument fce předešlé. Tento postup je opakován až do vyčerpání množiny jedince. V zásadě se můžeme setkat s dvěmi různými řešeními. První je případě, že veškeré prvky jedince jsou vyčerpány. Tento způsob hodnotíme za správně vytvořenou optimalizovanou funkci. Druhý způsob je takový, že při vyčerpání prvků jedince zbydou nevyplněny některé argumenty optimalizované funkce. Jelikož se pomocí metody AP reprezentované konkrétním jedincem nepodařilo vyplnit všechny vylosované argumenty, daný jedinec není vhodným řešením pro optimalizaci funkce. Oba způsoby vytváření jedince jsou prezentovány v animacích.

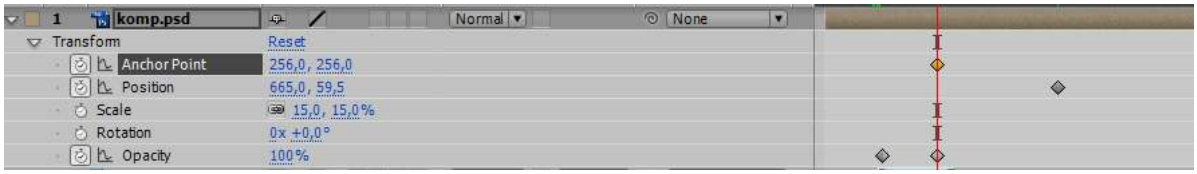
5.2.1 REALIZACE AMIMACE

Různé etapy výroby této animace jsou vidět na obrázcích v předešlé kapitole. Samotný postup výroby probíhal v různých fázích. První z nich byla příprava příkladů na nichž bude princip prezentován. Byly zvoleny dva odlišní jedinci, kteří se snaží pomocí vhodného funkcí výběru z množíny GFS snaží dosáhnout optimalizovaného řešení. Animace jsou si dost podobné, proto si ukážeme pouze postup výroby na jedné z nich.

Náhodný výběr je reprezentován pomocí losovacího zařízení. Toto zařízení je zobrazeno na obr. č. x. Losované objekty, čísla z předem dané z populace, jsou každý vložen jako samostatné vrstva v AE. Jejich pohyb je realizován pomocí vložených keyframů, které v průběhu času určují změnu polohy vrstvy. Výběrem

konktrétního prvku, jej zařadíme do náhodně vybraných jedinců z populace. Tento výběr je závislý na pořadí. Zaplněním množiny jedince zamezím dalšímu výběru z populace.

Druhá část animace se věnuje vzniku funčkní množiny GFS. Jak je uvedeno v teoretické části GFS_{ALL} je průnikem všech ostatních množin $GFS_{?ARG}$. To vytvoření množiny jsem se rozhodl vytvořit dle publikace Evoluční výpočetní techniky (Oplatková, zelinka) str. 272. Jedná se o nám známý obrázek, který je uveden jakou součást teoretické části této práce. Pro animaci jednotlivých množin je použit nástroj **Opacity,** tedy nastavení průhlednosti dané vrstvy. Vložením keyframů s hodnotou opacity 0 % a na rozdílnou pozici uložení hodnoty 100% dosáhneme efektu prolínačky. Kdy se obraz buďto přidá nebo mizí.



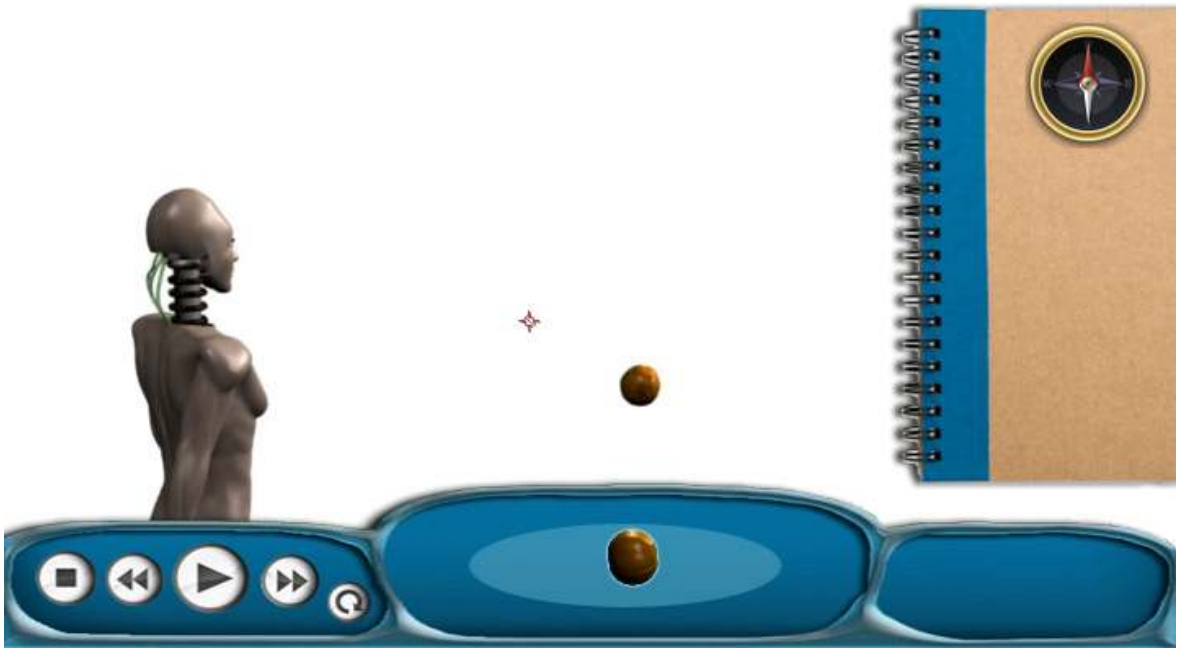
Obrázek 29: Ukázka práce v s opacity vrstvy

Na základě práce s průhledností jednotlivých vrstev byly nastaveny jednotlivým vrstvám keyframy tak, aby se zobrazovaly v námi chtěném okamžiku. Prvky dané množiny jsou zobrazeny v nových vrstvách, kterým je nakopírován efekt zobrazení a zmizení. Na základě vzniku GFS je možné přikročit k samotnému principu Analytického programování.

V poslední částí animace je zobrazen princip výroby optimalizované funkce, která zobrazena pomocí vrstvy textu. Tento text je změněn v závislosti výběru na určité funkce. Vytvářená funkce se postupně doplňuje o nově vybrané funkce z množiny GFS. Změna textu je realizována pomocí vložení nového statického textu, který je uložen v nové vrstvě. V případě vložení statického obrázku je možnost tomuto obr. přiřadit určitou duration. Pokud chci změnit layer neboli střihnout – stačí zkrátit duration dané vrsty a vložit pod ni vrstvu novou.

MODEL DUMMY POSTPRODUKCE

Již od začátku bylo naplánováno vyrenderované sekvence obrázků z Blenderu doplnit o výrazové prostředky, které mají za úkol podtrhnout prezentovaný princip. Hlavními požadavky kladené pro vytvoření tohoto výukového materiálu jsou přehlednost a pochopitelnost. Scénu jsem se rozhodl udělat podobně jako u starých počítačových her. Kde budou přidána různá menu, která budou přibližovat jednotlivé pohyby robota. Každý prvek menu je do projektu importován jako nový layer, což umožní je nezávisle na sobě animovat.



Obrázek 30: Ukázka interaktivního menu

Pro vytvoření efektu vyjíždějícího menu byly vloženy keyframy na pozice, na kterých se menu má vyskytovat (tzn. v obraze a schované). Animaci AE dopočítá obdobně jako blender a menu se nám zobrazí na požadované pozici. Pro animaci rotujícího ovoce byly použity objekty, které byly předem vymodelovány v Blendru. Do AE se vložily je jako nové vrstvy. Následně se jim upravil **Scale (měřítko)** a vyklíčoval se layer pomocí keying effectu. Klíčování spočívá na odebrání pozadí a ponechání pouze požadované části objektu na scéně. Jelikož pozadí ovoce bylo bílé, nebylo těžké záběr vyklíčovat. Poté se ovoci nastavila pozice, na které má setrvat a pomocí nástroje **Parent tool** se layer připojil

k vyjížděcímu menu. Tím bylo zajištěno, že při vyjetí menu se zobrazí i dané ovoce.

Kompas se na scénu přidal, protože Dummy se otáčí do všech světových stran. Užitím kompasu se zvýší přehlednosti při otáčkách robota. Byl vytvořen v programu Adobe Photoshop, z něhož byl vyexportován v souboru .png, čímž bylo zajištěno, že jednotlivé vrsty budou odděleny.

Obrázek 31: Návrh kompasu

Jak je zmíněno víše, jedná se o více vrstev - konkrétně dvě - střelku a ukazatel světových stran (pozadí). Pohyb střelky se animuje pomocí nástroje **Rotate.** Obdobně jako u animace menu, se vloží jednotlivé keyframy, na pozice mezi kterýma má střelka vykonávat pohyb.

ZÁVĚR

Cílem této práce bylo vytvoření animací pro potřeby prezentování algoritmu Analytického programování. S touto optimalizační metodou jsem se seznámil a vytvořil vizualizaci jejího principu na určitém příkladu. Všech stanovených cílů se mi podařilo dosáhnout.

V teoretické části práce byly popsány principy evolučních algoritmů, se kterými pracují nové optimalizační metody. Hlavní pozornost je soustředěna kolem algoritmu Analytického programování. AP je jednou z neustále vyvíjených metod optimalizace. Její rozvoj je také spjat s FAI-UTB, kde vzniklo velké množství publikací o této problematice. Autor práce se dále v teoretické části věnuje rozebíráním evolučních algoritmů, které jsou potřeba pro běh algoritmu. Zmiňuje jejich vznik a popisuje princip jejich evoluce.

V praktické části práce byla popsána tvorba animací pro potřeby prezentací. Důležitou součástí výukových materiálů je jejich samotná pochopitelnost. Po shlédnutí obou animací by měl divák porozumět základnímu principu algoritmu.

Animace zobrazující průběh algoritmu Analytického programování byly vytvořeny pomocí různých grafických programů. Vizualizace byla vytvořena v jednotné jednoduché grafické úpravě. Divák by neměl být rozptylován nepodstatnými předměty, které nepopisují samotný princip algoritmu. Důraz byl kladen na názornost vytvořených prezentací, a proto byly animace doplněny o prostředky vysvětlující dané kroky průběhu algoritmu.

V první animaci je schematicky znázorněn princip Analytického programování. Tento princip je demonstrován na vytváření optimalizované funkce. Druhá animace zobrazuje konkrétní případ využití Analytického programování na stezce Santa Fe (Trail Santa Fe).

ZÁVĚR V ANGLIČTINĚ

The goal of the thesis was to create the animation for the presenting purpose of algorithm Analytic programming. I explored this optimization method and created a visualization of it's principle on the certain example. I have reached all the given goals.

Principals of evolution algorithms were described in the theoretical part of the thesis. Optimization method works with these principles. The main attention has been concentrated around algorithm of analytic programming. AP is one of the continually developed methods of optimization. AP's development is also closely connected with the FAI-UTB, where many publications aimed on this topic have been created. Furthermore, in the theoretical part author describes evolution algorithms, which are necessary for the algorithm process, he mentions their creation and describes the principal of their evolution.

The creation of animations for presenting has been described in the practical part. Important part of the study materials is their 'understandability'. The viewer should understand the basic principle of algorithm after watching both animations.

The principal of Analytic programming is displayed on a scheme in the first animation. This principal is explained on the creation of optimized function. The second animation shows an example of analytic programming on Santa Fe path (Trail Santa Fe).

SEZNAM POUŽITÉ LITERATURY

[1] HYNEK, J. Genetické algoritmy a genetické programování. Grada. vyd. 1, ISBN 978-80-247-2695-3.

[2] KOZA, J. R. Genetic Programming: On the Programming of Computers by Means of Natural Selection, The MIT Press, 1992.

[3] KOZA, J.R. Genetické programování: Na programování počítačů prostředky k přirozenému výběru, MIT tisk, 1992.

[4] KOZA, J.R. Genetické programování: Vzor pro geneticky množící se populace počítačových programů vyřešit problémy, Stanford univerzita oddělení informatiky technický hlásí Stan-CS-90-1314. Důkladná zpráva, možná použitý jako návrh k jeho 1992 knize. 1990.

[5] MEYER, T. Adobe After Effects Computer Press 2010, ISBN: 978-80-251-2500-7.

[6] OPLATKOVÁ, Z. Metaevolution – Synthesis of Optimization Algorithms by means of Symbolic Regression and Evolutionary Algorithms, Lambert-Publishing, 2009, ISBN 978-8383-1808-0.

[7] POKORNÝ, P. Blender: Naučte se 3D grafiku. 2. vyd.: BEN – technická literatura, 2009, ISBN 80-7300-244-2.

[8] ZELINKA, I., OPLATKOVÁ, Z., OŠMERA, P., ŠEDA, M., VČELAŘ, F. Evoluční výpočetní techniky – principy a aplikace, BEN – technická literatura, Praha, 2008, ISBN 80-7300-218-3.

[9] ZELINKA, J.R. Umělá inteligence I. Neuronové funkce a genetické algoritmy. Vysoké učení technické v Brně, 1998, Vutium, ISBN 80-214-1163-5

[10] ZELINKA, J.R., OPÁLKOVÁ, Z., ŠENKEŘÍK, R. Aplikace umělé inteligence (aneb vybrané statě z evolučních algoritmů), Univerzita Tomáše Bati ve Zlíně, 2010, ISBN 978-80-7318-898-6.

ZDROJE ONLINE

[11] DOSTÁL, M. Evoluční výpočetní techniky. Univerzita Palackého, Olomouc, 2007. [cit. 2010-05-01]. Dostupné z WWW: http://phoenix.inf.upol.cz/esf/ucebni/evt.pdf

[12] KALÁTOVÁ, E., DOBIÁŠ, R. Evoluční algoritmy. Západočeská univerzita v Plzni, 2000. [cit. 2010-05-01]. Dostupné z WWW: http://www.kiv.zcu.cz/studies/predmety/uir/gen_alg2/E_alg.htm

[13] KRESTA, A. Využití algoritmu PSO při hledání portfolia s nejnižším rizikem. 7. mezinárodní konference Finanční řízení podniků a finančních institucí. Ostrava. 2009. [cit. 2010-05-01]. Dostupné z WWW: http://www.ekf.vsb.cz/shared/uploadedfiles/cul33/Kresta.Ales.pdf)

[14] POSPÍCHAL, J., KVASNIČKA, V. Evoluční algoritmy. Computerworld, 1995. [cit. 2010-04-22]. Dostupné z WWW: http://www.kiv.zcu.cz/studies/predmety/uir/gen_alg2/E_alg.htm

[15] TVRDÍK, J. Evoluční algoritmy. Učební texty ostravské univerzity. Ostrava, 2004. [cit. 2010-05-01]. Dostupné z WWW: http://prf.osu.cz/doktorske_studium/dokumenty/Evolutionary_Algorithms.pdf

[16] ŠTĚRBA, O. Genetické a evoluční algoritmy. Darvinova evoluční teorie a umělá inteligence. 2006. [cit. 2010-04-22]. Dostupné z WWW: http://www.elreste.cz/evolution_algorithm.html

[17] VILÍMEK, T. Blender tutoriál: Animace chůze ženy v Blenderu. Tvorba armatury. 2009. [cit. 2010-03-11]. Dostupné z WWW: http://www.3dscena.cz/art/3dscena/bleder-chuzezeny1.html

[18] ZELINKA, I., OPLATKOVÁ, Z., NOLLE, L. Boolean Symmetry Function Synthesis by Means of Arbitrary Evolutionary Algorithms-Comparative Study, International Journal of Simulation Systems, Science and Technology, Volume 6, Number 9, August 2005, pages 44 – 56, ISSN: 1473-8031, online http://ducati.doc.ntu.ac.uk/uksim/journal/Vol-6/No.9/cover.htm

[19] *3Dscéna* [online]. 2002 - 2003 [cit. 2010-05-01]. Dostupné z WWW: <www.3Dscena.cz>.

[20] *Blender* [online]. 2005 [cit. 2010-06-01]. Dostupné z WWW: <http://www.blender.org/>.

[21] *Grafika on-line* [online]. 2003 [cit. 2010-05-01]. Dostupné z WWW: <http://www.grafika.cz/>.

[22] *Creative Cow Tutorials* [online]. 2005 [cit. 2010-05-01]. Dostupné z WWW: <http://library.creativecow.net/tutorials/adobeaftereffects>.

[23] *Blender tutorials* [online]. 2005 [cit. 2010-05-01]. Dostupné z WWW: <http://www.tutorialsforblender3d.com/>

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

AE	Adobe After Effects
AP	Analytické programování
DSH	Diskrétní množina
GA	Genetický algoritmus
GFS	Základní množina Analytického programování
GP	Genetické programování
HD	High Definition
LISP	Programovací jazyk
MH	Make Human
PRT	Pertubace
PSO	Particle swarm optimalization
SOMA	Samoorganizující se Migrační Algoritmus

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH