3D vizualizace vybraných procesů umělé inteligence

3D visualization of processes of artificial intelligence

Zbyněk Molnár

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:	**Zbyněk MOLNÁR**
Studijní program:	**B 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**3D vizualizace vybraných procesů umělé inteligence**

Zásady pro vypracování:

1. **Nastudujte základy teorie vybraných procesů umělé inteligence.**
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. **KVASNIČKA V., POSPÍCHAL J., TIŇO P. 2002 Evoluční algoritmy, STU Brati-slava, 2000, ISBN 85-246-2000.**
3. **VAŘACHA, Pavel. Syntéza neuronových sítí metodou symbolické regrese. Zlín, 2006. 83 s. UTB. Vedoucí diplomové práce doc. Ing. Ivan Zelinka, Ph.D.**
4. **ZELINKA, I. 2002. Umělá inteligence v problémech globální optimalizace. Praha : BEN, 2002, 189 s. ISBN 80-7300-069-5.**
5. **ŠNOREK, M. 2004 Neuronové sítě a neuropočítače, Praha : ČVUT, 2004, 156 s.**
6. **POKORNÝ, Pavel. Blender – Naučte se 3D grafiku. Praha : BEN, 2006. 248 s. ISBN 80-7300-203-5.**
7. **3D scéna [online]. 2003 [cit. 2008-01-28]. Dostupný z WWW: http://www.3dscena.cz/3dshowks.php?xuid=207.**
8. **Blender [online]. 2001 [cit. 2008-01-28]. Dostupný z WWW: http://www.blender.org/education-help/tutorials/.**

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

L.S.

prof. Ing. Vladimír Vašek, CSc.
děkan

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

ABSTRAKT

Tato práce se zabývá především pochopením a vizualizací vybraných procesů v umělé inteligenci. Hlavním úkolem je vytvoření jednoduchých 3D animací, které názorně popíší procesy v neuronových sítích a Samo Organizujícího se Migračního Algoritmu. Práce obsahuje teoretický popis jednotlivých procesů a všechny animace a zdrojové soubory vytvořených modelů a animací programu Blender 2.45 jsou na přiloženém CD.

Klíčová slova: umělá inteligence, neuronová síť, Samo Organizující se Migrační algoritmus, 3D animace

ABSTRACT

This thesis deals understanding of visualization of selected processes of artificial intelligence. The very task was to create elementary 3D animations which by visual demonstration describe processes of neural network and Self-Organizing Migration Algorithm. Thesis includes primarily theoretical account of particular processes and all animation and source codes created models and animation of program Blender 2.45 can be found on attached CD.

Keywords: artificial intelligence, neural network, Self-Organizing Migration Algorithm, 3D animation

V rámci této práce, bych chtěl na tomto místě poděkovat Ing. Pavlu Vařachovi za všechny rady, názory a připomínky, které mě vedli při tvorbě této práce. Dále bych chtěl poděkovat svým rodičům za podporu po celou dobu mého dosavadního studia a v neposlední řadě svým přátelům, kteří mi poskytli cenné rady i kritiku.

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

Počítačová grafika se jako obor začala rozvíjet asi před padesáti lety. Její vývoj byl ovlivněn zejména možnostmi jejího uplatnění. V dnešním době se můžeme setkat s počítačovou grafikou v mnoha oblastech – např. odvětví reklamy, speciální efekty ve filmu, umělecká tvorba, příprava publikací (DTP), konstrukční návrhy (CAD), grafická komunikace (ikony), grafické znázornění údajů (grafy, diagramy, prezentace), zpracování a úprava obrazu (úprava fotografií apod.). U 3D grafiky to bylo daleko obtížnější, je závislá na výkonu počítače. A to nejen v oblasti již zmíněných konstrukčních návrhu, ale i v oblastech počítačových her nebo filmové tvorby.

Pro vytvoření konkrétních 3D objektů a následných animací se používají specializované softwarové nástroje. Práce s tímto softwarem bývá pro začátečníky velmi složitá, a i po zvládnutí pokročilých grafických technik nebývá zaručeno, že budeme vytvářet dokonalé obrázky a filmové scény. K tomu je potřeba znát spoustu jiných věcí, jako je kompozice a koncepce obrazu.

Stejný nástup, jaký zažila 3D grafika s příchodem rychlých mikroprocesorů, zažili i neuronové sítě. O ty se začalo zajímat opět v 80 letech, kdy byla naplno vyvrácena fáma o tom, že jednoduchý preceptor nedokáže vypočítat vylučovací funkci XOR. Výkonné počítače byli i u zrodu nové oblasti umělé inteligence a to evolučních algoritmů.

Úkolem této práce je tedy vytvoření jednoduchých animací, které názorně popíší všechny zákonitosti ve vybraných procesech umělé inteligence. K tomu ovšem potřebujeme jisté teoretické nastínění problému, které je popsáno v teoretické části této práce. Praktická část se zabývá hlavně popisem programu na 3D grafiku Blender a popis modelování objektů a vytváření animací v tomto programu. Všechny vytvořené animace i zdrojové soubory programu Blender jsou na přiloženém CD – ROM.

I. TEORETICKÁ ČÁST

1 OPTIMALIZAČNÍ ALGORITMY

Optimalizační algoritmy se používají tam, kde analogické řešení problému není vhodné či realizovatelné. Při vhodném nastavení parametrů algoritmu, můžeme maximálně minimalizovat uživatelské zásahy v činnosti příslušného zařízení. V inženýrské praxi se občas setkáme s problémy, které jsou definovány jako optimalizační. Jinými slovy, řešený problém lze převést na matematický problém daný vhodným předpisem, jehož optimalizace vede k nalezení argumentů tzv. účelové funkce, což je cílem optimalizace. [4] Tu můžeme použít pro nalezení optimální trajektorie robota, optimální tloušťky stěny tlakové nádoby atd. Tyto optimalizované funkce pracují s argumenty, které mohou být různorodého charakteru (obor celočíselný, reálný, komplexní, diskrétní apod.). Navíc se mohou objevovat požadavky na nejrůznější omezení a penalizace, a to nejen na dané argumenty, ale také na funkční hodnotu optimalizované funkce. Samozřejmě většinu řešení lze provést analytickou cestou, ale může to být značně zdlouhavé a komplikované.

Pro zefektivnění tohoto problému bylo za posledních dvacet let vyvinuto množství algoritmů, se kterými se dají řešit i velmi složité problémy. Těmto algoritmům se souhrnně říká „evoluční algoritmy". Typickým rysem pro evoluční algoritmy je, že pracují s tzv. populacemi možných řešení, jimž se říká jedinci. [4] Jedinci se snaží na základě evolučních principů v jednotlivých cyklech nalézt nejlepší řešení.

1.1 Rozdělení optimalizačních algoritmů

Optimalizační algoritmy můžeme rozdělit do skupin podle jejich principů a činnosti. Existuje daleko víc možností rozdělení, takže následující rozdělení je jeden z možných pohledů na klasické, ale i moderní optimalizační metody.

- **Enumerativní.** Obecný postup tohoto přístupu by potřeboval k úspěšnému dokončení čas, který by byl delší než existence našeho vesmíru. Tento princip spočívá ve výpočtu všech možných kombinací daného problému.
- **Deterministické.** Tato metoda je založena na předběžných předpokladech, které umožní podávat algoritmu efektivní výsledky. Tyto předpoklady jsou:
  - Problém je lineární
  - Problém je konvexní

- Prohledávaný prostor možných řešení je malý
- Prohledávaný prostor možných řešení je spojitý
- Účelová funkce je unimodální
- Nelineární iterakce mezi parametry
- Informace o gradientu
- Problém v analytickém tvaru

- **Stochastické.** Tento princip je založen na využití náhody. Hodnoty argumentů účelové funkce jsou hledány čistě náhodně s tím, že vždy vybereme tu nejlepší hodnotu.
  - Pomalé
  - Malý rozsah argumentů účelové funkce prohledávaného prostoru
  - Vhodné pro hrubý odhad
- **Smíšené.** V tomto principu jde o spolupráci mezi stochastickými a deterministickými metodami, jejichž kooperací dosahujeme překvapivě dobrých výsledků. Mezi tuto skupinu patří i evoluční algoritmy.
  - Robustní – nezávisí na počátečních podmínkách. Řešení bývá obvykle reprezentováno více globálními extrémy.
  - Efektivní a výkonné – Tato skupina dokáže nalézat kvalitní řešení během malého počtu ohodnocení účelové funkce.
  - Odlišné od stochastických metod
  - Minimální nebo žádné požadavky na předběžné informace
  - Černá skříňka – nepotřebují k činnosti analytický popis problému
  - Více řešení během jednoho spuštění

V této práci je podrobně popsán jeden algoritmus z oblasti smíšených algoritmů a to konkrétně algoritmus SOMA.

1.2 Účelová funkce

Účelová funkce je funkce, jejíž optimalizací dojde k nalezení optimálních hodnot jejich argumentů. Na tuto funkci můžeme nahlížet jako na geometrický problém, který hledá nejnižší (minimum) či nejvyšší (maximum) pozici na N rozměrné ploše. Pro tuto plochu můžeme mít označení „hyperplocha" nebo „prostor možných řešení". Má-li například optimalizovaná funkce pět argumentů, pak extrém hledáme na pětirozměrné ploše a v šestirozměrném prostoru. Šestá dimenze reprezentuje návratovou hodnotu účelové funkce. Prvních pět dimenzí zobrazuje typ os „x" a „y", šestá má charakter osy „z".

2 SOMA

Tato zkratka představuje **S**amo-**O**rganizující se **M**igrační **A**lgoritmus (*Self-Organizing Migration Algorithm*). Tento algoritmus vytvořil I. Zelinka v roce 1999. Vzhledem k tomu, že pracuje s populacemi podobně jako např. genetické algoritmy a výsledkem po jednom evolučním cyklu (migračním kole) je totožný s genetickými algoritmy či diferenciální evolucí, lze jej řadit např. mezi evoluční algoritmy navzdory faktu, že během jeho běhu nejsou vytvářeni noví potomci, jak je tomu u jiných evolučních algoritmů. [4] Jak již bylo zmíněno, tento algoritmus nevytváří nové jedince, ale stojí spíš na prohledávání prostoru možných řešení, proto může být označován i jako algoritmus memetický.

2.1 Princip algoritmu SOMA

Princip použit pro algoritmus SOMA, můžeme pozorovat kdekoliv v přírodě a to konkrétně u inteligentních jedinců, kteří se snaží spolupracovat při řešení společného úkolu. Jako příklad spolupráce inteligentních jedinců si můžeme vzít smečku vlků. Jednotlivý jedinci putují krajinou a jejich společným úkolem je najít nejlepší zdroj potravy pro smečku. I když pracují v rámci smečky, tak mezi s sebou i soutěží. Každý jedinec se snaží najít nejlepší zdroj potravu. To označujeme jako fázi soutěžení. Hned po fázi soutěžení následuje fáze spolupráce jedinců. Ta spočívá ve sdělování informací o tom, který jedinec má zatím nejlepší zdroj potravy. Jedinec, který našel nejlepší zdroj potravy zůstává na svém místě a ostatní jedinci opustí své zdroje potravy a migrují po krajině směrem k jedinci s nejlepším zdrojem potravy. Během jejich putování ovšem znovu hledají ještě lepší zdroj potravy. Tato celá fáze se opakuje, dokud se všichni jedinci nesejdou u nejlepšího zdroje potravy. Z tohoto principu lze usoudit, že pomocí tohoto algoritmu můžeme nalézat velmi hluboké a často i globální extrémy(zdroje potravy).

Tab. 1. Význam biologické terminologie v algoritmu SOMA

Biologická realita	Počítačové implementace
členové smečky, společenství	jedinci v populaci, parametr NP
člen společenství s nejlepším zdrojem	Leader, vedoucí aktuálního migračního
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	Migrační kola v algoritmu SOMA

2.2 Parametry SOMA

Běh algoritmu SOMA je ovlivňován dvěma druhy parametrů.

- řídící parametry
- ukončovací parametry

2.2.1 Řídící parametry

Jsou to parametry, které mají vliv na kvalitu běhu algoritmu z hlediska hodnoty účelové funkce. Tyto parametry musí být zvoleny uživatelem ještě než započne běh algoritmu.

Tab. 2. Řídící parametry SOMA

Parametr	Doporučený rozsah
Mass	<1.1, 5>
Step	<0.11, Mass>
RPT	<0, 1>
D	dáno problémem
NP	<10, definuje uživatel>

1) **Mass.** Tento parametr určuje jak daleko se aktivní jedinec zastaví od tzv. vedoucího jedince. [4] Nastavení parametru je doporučeno dávat od 1.1 do 5 . Hodnota menší jak jedna by mohla vést k degradaci migračního procesu, protože se jedinec zastaví před vedoucím jedince. Pokud nastavíme hodnotu Mass = 1, pak se náš aktivní jedinec zastaví na pozici vedoucího jedince.

2) **Step.** Tento parametr určuje podrobnost, s jakou je mapována cesta aktivního jedince k vedoucímu jedinci. Proto u účelové funkce, kde neznáme geometrii, která

ji reprezentuje, nastavujeme nízkou hodnotu. Tento prostor tak bude prohledán podrobněji. Naopak pro účelovou funkci s méně lokálními extrémy použijeme velkou hodnotu parametru a tím algoritmus výrazně urychlíme.

3) **RPT.** Parametr důležitý pro nastavení perturbační vektoru. To je vektor, který určuje, zda půjde jedinec přímou cestou k vedoucímu nebo ne. Nejoptimálnější je hodnota 0.1. Se zvyšující se hodnotou parametru roste konvergence k lokálním extrémům. Při hodnotě PRT=1 zaniká stochastická složka a tím se algoritmus chová podle deterministických pravidel.

4) **D.** Parametr udávající počet argumentů účelové funkce

5) **NP.** Tento řídící parametr určuje, kolik jedinců bude tvořit populaci. [4] Jeho hodnota se obvykle nastavuje jako 0.2 až 0.5 parametru **D**.

2.2.2 Ukončovací parametry

Jsou to parametry, které ukončují algoritmus za předem nadefinovaných podmínek. Tyto parametry musí být zvoleny uživatelem ještě než započne běh algoritmu.

Tab. 3. Ukončovací parametry SOMA

Parametr	Doporučený rozsah
Migrace	<10, definuje uživatel >
AcceptedError	< ± libovolný, definuje uživatel >

1) **Migrace.** Parametr Migrace udává kolikrát se populace jedinců přeorganizuje.

2) **AcceptedError.** Tento parametr definuje maximální rozdíl, který je povolen mezi nejlepším a nejhorším jedincem v aktuální populaci. Pokud se hodnota AcceptedError dostane pod hodnotu nastavenou na začátku, tak se algoritmus ukončí. Optimální nastavení je kolem AcceptedError = 1.

2.3 Strategie SOMA algoritmu

V současné době existuje několik verzí algoritmu SOMA. Tyto verze nazýváme strategie

„Všichni k jednomu“ (AllToOne). Tato strategie spočívá v tom, že všichni jedinci migrují směrem k Leaderovi. (viz kap. 2.4)

„Všichni ke všem" (AllToAll). V této strategii neexistuje Leader. Všichni jedinci migrují ke všem ostatním tak jako ve verzi „Všichni k jednomu" s tím rozdílem, že po dokončení migrací aktuálního jedince se daný jedinec vrací na pozici, kde byl nalezen nejlepší extrém během jeho NP-1 migračních cest vykonaných v jednom migračním kole. [4] Touto strategií se prohledá větší část prostoru a tím máme větší pravděpodobnost, že najdeme globální extrém.

„Adaptivně všichni ke všem" (AllToAllAdaptive). Jde o analogii „všichni ke všem" s tím, že aktuálně migrující jedinec se přesouvá po každé aktuálně dokončené migraci ke každému jedinci a vybere se vždy ta nejlepší poloha.

„Všichni k jednomu náhodně" (AllToOneRand). Všichni jedinci se pohybují k Leaderovi, který ovšem nereprezentuje nejnižší místo, ale je náhodně vybrán ze zbývajících jedinců.

„Svazky" (Clusters). Svazky využívají všechny předchozí strategie. Svazek spočívá v tom, že jedinci jsou roztřídění do skupin „svazků" a v každé z nich probíhá samostatný SOMA algoritmus. Jedinci se pohybují po hyperploše a tím se mohou svazky rozpadat nebo naopak spojovat.

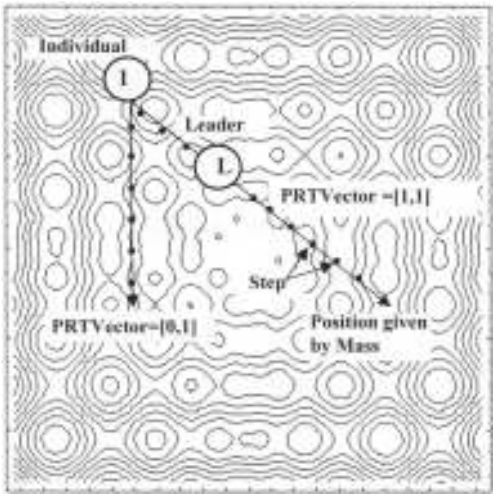
2.4 Podrobný popis strategie „Všichni k jednomu"

Zde bude podrobně popsána krok za krokem strategie „Všichni k jednomu" algoritmu SOMA. Jeden cyklus algoritmu SOMA se nazývá Migrační kolo. V tomto Migračním kole se nevytváří noví jedinci, ti se jen přesunou do nejnižší pozice pomocí sekvence pozic vypočítaných vzhledem k pozici Leadera. Samozřejmě můžeme matematicky demonstrovat algoritmus SOMA jako harémový styl tvorby nových potomků, kdy Leader je rodič - samec a ostatní putující jedinci jsou samice. Ovšem tento popis je méně elegantní než soutěživě – kooperativní filozofie. Základní strategii algoritmu SOMA je „Všichni k jednomu" a postupuje podle následujících kroků:

1) **Definice parametrů.** Ještě před startem SOMA je nutné nastavit řídící (viz. kap. 2.2.1) a ukončovací (viz. kap. 2.2.2) parametry. Musíme ovšem vytvořit i účelovou funkci, která reprezentuje životní prostředí jedinců. Skalár účelové funkce je použit jako měřítko kvality daného jedince. Účelová funkce je přenesený problém z reálného světa.

2) **Tvorba populace.** Pomocí generátoru náhodných čísel vytvořím prvopočáteční generaci.

3) **Migrační kola.** Jedinci na účelové funkci najdou nejníže uloženého jedince. Toho zvolí za Leadera Migračního kola. Potom se začnou ostatní jedinci pohybovat směrem k Leaderovi pomocí skoků. Počet skoků určuje parametr Step. (viz. kap. 2.2.1) Při každém skoku jedinec přepočítá, zda jeho aktuální poloha na účelové funkci není lepší než předchozí. Pokud ano tak si ji zapamatuje. Pokud ještě nebyli vyčerpány všechny skoky, pak pokračuje až do posledního skoku, který je dán parametrem Mass (viz. kap. 2.2.1) a zapamatuje si jen tu nejvýhodnější polohu, kterou našel po cestě k Leaderovi. Po ukončení skoku se jedinec přesune právě do této polohy. Ještě před tím než začne jedinec skákat směrem k Leaderovi musíme zvolit směr, kterým se k němu bude ubírat. Tento směr určuje PRTVector. (viz. kap. 2.2.1) Dále jsou vygenerovány náhodná čísla, která porovnáváme s PRT parametrem. Jejich počet je určen parametrem D (viz. kap. 2.2.1) . Jestliže je n-té vygenerované číslo větší nežli PRT parametr, pak je n-tý parametr PRTVectoru nastaven na 0 a v opačném případě na 1. [4] Pokud je hodnota 0 pak se vyruší druhý člen z rovnice (*Rovnice 1.*) a tím jsou parametry nepřepočítány a zůstanou na svých předcházejících hodnotách. Ale každé migrační kolo se tyto parametry mohou měnit. Takže pokud jsou parametry v jednom Migračním kole zmraženy, tak v dalším Migračním kole už být nemusí.



Obr. 1. PRTVector

Rovnice 1. $\vec{r} = \vec{r}_0 + \vec{m}.t.\vec{PRTVector}$

4) **Testování naplnění ukončovacích parametrů.** V tomto kroku se kontroluje rozdíl mezi nejlepším a nejhorším jedincem. Pokud je rozdíl větší než parametr AcceptedError (viz. kap. 2.2.2) a nebyl vyčerpán počet cyklů, opakujeme předešlý krok. Pokud nejsou obě podmínky splněny, algoritmus je ukončen.

5) **Stop.** V tomto kroku dojde k vyhodnocení algoritmu. Vrátí se nejlepší poloha jedince po posledním migračním kole.

3 NEURONOVÉ SÍTĚ

Již mnoho let přitahuje pozornost laiků i odborníků umělý informační systém, který principiálně napodobuje nervové soustavy a procesy v mozku živých organizmů. Ty totiž mohou pracovat podstatně dokonalejším způsobem než dosavadní konvenční výpočetní technika.

3.1 Historie neoronových sítí

Pokud pomineme část historie, která vychází z fyziologie, a která spadá do devatenáctého století, lze říci, že počátek neuroinformatiky sahá do dvacátých let dvacátého století. První práci o modelech neuronu v této době publikuje Američan W. S. McCulloch. Ten v roce 1943 se svým sedmnáctiletým studentem Waltrem Pittsem vytvořil práci, ve které popisuje velmi jednoduchý matematický model neuronu, což je základná buňka neuronové soustavy. Tento model pracoval s bipolárními parametry (tj. z množiny {-1,0,1}). Svou prací ukázali, že nejjednodušší typ neuronové sítě dokáže počítat jakékoliv aritmetické či logické funkce. I když nepočítali s využitím svého jednoduchého modelu, tak se stal inspirací pro Norberta Wienera při řešení podobnosti nervové soustavy a systémů výpočetní techniky. Dokonce i autor projektu elektronických počítačů, američan John von Neumann, si pohrával s myšlenkou počítačů které pracují na základě činností mozku. Donald Hebb ve své práci z roku 1949 „The Organization of Behaviour“ navrhl učící pravidlo. Domníval se že podmíněné reflexy, které můžeme vidět u všech živých organizmů, jsou vlastností jednotlivých neuronů. V roce 1951 spatři světlo světa první neuropočítač „Snark”, který zkonstruoval Marvin Minsky. Tento počítač již automaticky adaptoval váhy (míra synoptické propustnosti). V roce 1957 byl stvořen perceptor což bylo zobecnění McCullochova a Pittsova modelu Frankem Rossenblattem, který navrhl učící algoritmus, který po konečném počtu kroků nalezne váhový vektor. Tento princip použil o rok později pro sestrojení neuropočítače Mark I Perceptor. Pak došlo k jistému uvíznutí na mrtvém bodě. Zapříčinili to hlavně dvě věci. První bylo že se k neuronovým sítím přistupovalo spíše experimentálně a zanedbávali se analytické výzkumy neuronových modelů. Druhou byla fáma, která se rozšiřovala o neuronových sítích. A to konkrétně, že již do dvou let sestrojí umělý mozek. To odradilo mnohé vědce, kteří se do té doby zabývali neuroinformatikou. Nakonec v roce 1969 publikoval Minsky a Papert rukopis Perceptors ve kterém argumentovali, že jeden preceptor nedokáže vypočítat jednoduchou vylučovací

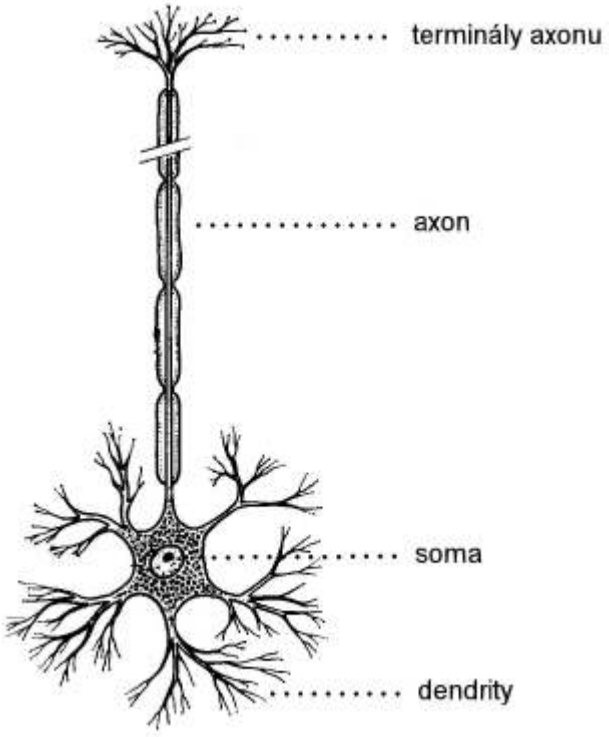
funkci XOR. Tento rukopis měl velký vliv a ve Spojených státech pokračoval výzkum neuronových sítí jen ojediněle a izolovaně. Z tohoto stavu se vědci osmělili až v 80. letech. V roce 1986 vydala skupina kolem Davida Rumelharta, Geoffraye Hintona a Ronalda Williamse sborník, ve kterém popsali učící algoritmus zpětného řízení chyby pro vícevrstvou neuronovou síť. Tím vyvrátili tvrzení Minského a Paperta o nepoužitelnosti neuronových sítí pro složitější funkce. Tento algoritmus je doposud nejpoužívanější učící metodou neuronových sítí. [9] Terrence Sejmowski a Charles Rosenberg demonstrovali sílu tohoto algoritmu pomocí systému NETtalk, který v krátké době učení neuronové sítě z příkladu, dokázal konvertovat anglicky psaný text na mluvený. V roce 1987 se v San Diegu konala první větší konference specializovaná na neuronové sítě. (IEEE International Conference on Neural Networks) [9] Hlavní rozdíl mezi neuronovými sítěmi a konvenčními počítači je, že konvenční počítače pracují podle předem daného postupu – algoritmu. Naproti tomu neuronové systémy uskutečňují velmi vysoký počet dílčích operací současně a pracují bez algoritmu. Jejich činnost je založena na procesu učení, při kterém se neuronová síť postupně co nejlépe adaptuje k řešení dané úlohy. [5] Neuronovými sítěmi dnes můžeme řešit tyto úlohy :

- Rozpoznání písma
- Identifikace podpisů
- Převod mluvené řeči na psaný text
- Predikce v pojišťovnictví a bankovnictví
- Identifikace radarových údajů
- Zjišťování cíle v sonarových obrazech
- Identifikace státních poznávacích značek

3.2 Biologický neuron

Neuron neboli nervová buňka je základním stavebním prvkem nervové soustavy živých organizmů. Jen mozková kůra člověka je tvořena asi 13 až 15 miliardami neuronů, z nichž každý může být spojen s 5000 jinými neurony. [9] Jde tedy o živou buňku, která má za úkol zprostředkovat příslušnou reakci na vnější podměty i na vnitřní stavy organizmu a na co nejúčelnější zpracování, uchování a přenos informací. Tyto informace se přenášejí

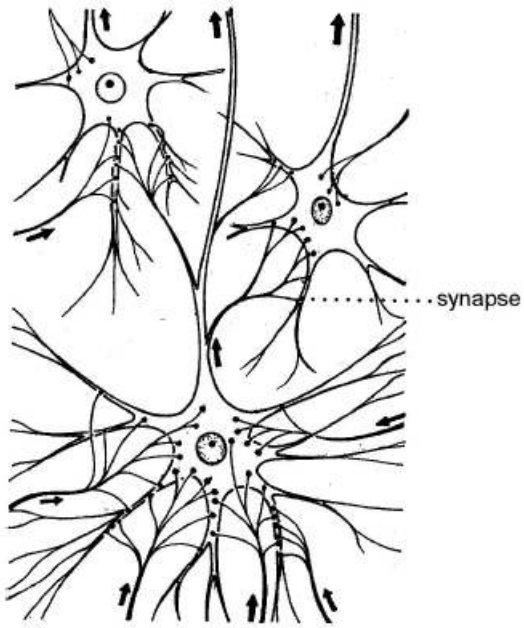
pomocí čidel, tzv. receptorů, které dokáží přijímat mechanické, tepelné, chemické a světelné podměty. Ty putují k efektorům, které tyto podměty zpracují. Do mozku se dostanou pomocí projekčních drah. V těch dochází ke kompresy a filtraci informací a pokračují do mozkové kúry. V té se nachází šest projekčních oblastí, které odpovídají jednotlivým smyslům. Fyziologové objevili a popsali řadu druhů neuronů. Všechny jsou ale složeny ze stejných částí. Struktura je tvořena buněčným jádrem zvaným soma, z kterého vychází jediný výstup zvaný axon avšak dále z axonu odbočuje řada větví tzv. terminálů. Terminály jsou ukončeny blánou, které slouží k přenosu informací na výběžky jiných dentritů. Naopak do somy vstupuje několik bohatě rozvětvených vstupů tzv. dentritů.



Obr. 2. Biologický neuron

Přenos informací mezi jednotlivými neurony zprostředkovávají mezineuronové rozhraní tzv. chemická synapse. Synapse můžeme rozdělit na excitační, které umožňují rozšíření vzruchu, a na inhibiční které tento vzruch tlumí. Vědci se domnívají že paměťová stopa neuronu vzniká právě zakódováním synapse mezi receptorem a efektorem. Díky tomu že jsou soma a axom obaleny speciální membránou, která generuje impulsy, je umožněno šíření informace mezi jednotlivými neurony. Tyto impulsy se přenáší pomocí synaptických bran, a tím dochází k podráždění jiných dentritů. Z dentritů se tento vzruch šíří do dalších

neuronů. Pokud do takto podrážděných neuronů dosadíme nějaké hraniční meze, které označujeme jako práh, tak jsou schopny samy generovat impuls a zajišťují tím přenos příslušné informace. Po každém průchodu signálu se synoptická propustnost mění, což je předpokladem paměťové schopnosti neuronů. [9]



Obr. 3. Synapse neuronové sítě

I propojení mezi neurony se v průběhu života radikálně mění. Pokud je spojení hodně používané, vytváří se paměťové stopy. Naopak pokud se spojení dlouho nepoužívá, může dojít k zapomenutí a přerušení spojení. I když během života dochází k obnově buněk, pro neurony toto neplatí. Obnovují se jen trny na dentritech.

3.3 Učení biologických neuronových sítí

Nejdůležitější poznatek procesu učení se nazývá Hebbovská hypotéza. Popisuje, jak se chovají dvě biologické buňky spojené axonem, právě když se obě buňky nacházejí v excitovaném stavu současně. Průchod vzruchů axonem je ovlivněn frekvencí. Čím větší je frekvence, tím lepší podmínky průchodu dostaneme. Pro maximální propustnost musí být oba neurony v místě příslušného synaptického styku aktivovány nadprahově.

3.4 Umělé neuronové sítě

Výstup „Y" umělých neuronových sítí můžeme brát jako určitou transformaci „T" vstupního signálu „X".

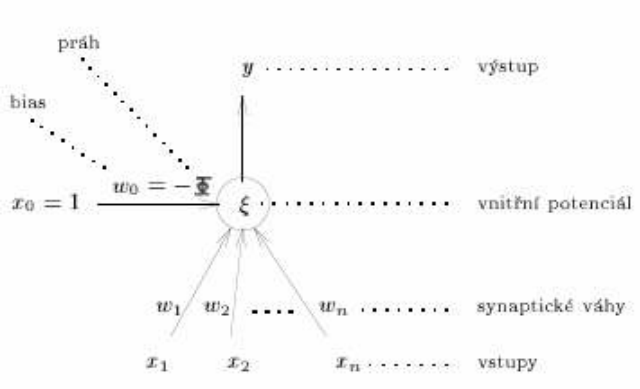
Rovnice 2. $\vec{Y} = T(\vec{X})$

Pokusy, které se zaměřovali na to, jaké transformace je možno pomocí neuronových sítí realizovat, se řeší už od začátku jejich poznání.

Základem pro umělý neuron je formální neuron. To je v podstatě přeformulování zjednodušené funkce neurofyziologického neuronu do matematické řeči. Formální neuron je prvkem umělé neuronové sítě perceptronovského typu. Formální neuron nahrazuje dentrity biologického neuronu pomocí vstupů x_1, x_2,…, x_n, které jsou ohodnoceny reálnými synaptickými váhami w_1, w_2,…, w_n. Inhibiční charakter nastavíme tím že dáme zápornou hodnotu váhy. Pro zjištění vnitřního potenciálu neuronu musíme udělat zváženou sumu vstupních hodnot.

Rovnice 3. $\xi = \sum_{i=1}^{N} w_i x_i.$

Výstupní hodnota „y" se potom indikuje po dosažení prahové hodnoty „Φ" vnitřního potenciálu „ξ".



Obr. 4. Formální neuron

Podle povahy vstupních dat rozlišujeme neurony na binární nebo spojité. Binární zpracovávají dvouhodnotová data. S daty nesoucí vícebitovou informaci si poradí spojité neurony. To že se tyto neurony nazývají spojité, nemá žádnou souvislost s jejich

implementací nebo způsobem jejich činnosti. Ty bývají nejčastěji digitální jako program pro nějaký procesor. V poslední době vzrůstá význam implementací optických.

3.5 Pracovní fáze umělé neuronové sítě

Pracovní fáze umělé neuronové sítě jsou dvě :

- Adaptivní
- Aktivní

3.5.1 Učení a jeho typy

V adaptivní části se síť učí, v aktivní vykonává naučenou činnost, vybavuje. [5] Při učení dochází v síti k adaptaci na daný problém a to především nastavováním synoptických vah prahů. Při učení nedochází ke změně topologie sítě. Jedinou výjimkou je síť GMDH . Při učení nastavíme váhy na počáteční hodnoty, které mohou být náhodné nebo vybrané podle nějakého případu. Potom přivedeme do sítě trénovací vstup a síť nám na výstupu poskytne odezvu. Učení rozlišujeme na :

- **S učitelem.** U tohoto druhu se váhy nastavují podle zpětné vazby. Ta hlídá rozdíl mezi žádaným a skutečným výstupem. Váhy se nastavují podle algoritmu, který zabezpečuje snižování chyby mezi skutečným a žádaným výstupem. Velikost rozdílu mezi výstupy v následujících krocích se postupně zmenšuje. Po provedení velkého počtu kroků se síť naučí vydávat stabilní výstup.
- **Bez učitele.** Algoritmus je navržen tak, že hledá ve vzorcích určité společné vlastnosti. Tomuto učení se říká samoorganizující.

Než se dosáhne učícího procesu u živých organizmů, musí se provádět opakovaně. Existují i případy jednorázového učení, při kterém si hned napoprvé předložený vzor zapamatuje. Takové vlastnosti mají jen některé umělé sítě. Jmenovitě Hopfieldova síť a jedna varianta Neocognitronu.

3.5.2 Vybavování

Hlavní částí aktivní fáze je vybavování. Následuje za fází adaptivní a zpracovávají se v ní vstupní data. [5]

Každý člověk je schopen, si pomocí určité asociace přiřadit známé tváři jméno, poznávat lidi kolem sebe atd. Spojujeme si tvary, místa, objekty, tváře, lidi. Z tohoto tvrzení lze formulovat že lidský mozek pracuje asociativním způsobem.

Umělá síť je taková asociativní paměť, ve které jsou uloženy vzory, které si potom neuronová síť vybavuje. I vybavování Neuronových sítí má dvě varianty:

- **Autoasociativní.** v paměti jsou uloženy vektory ($\vec{X}_1,...,\vec{X}_m$)
- **Heteroasociativní.** v paměti jsou uloženy dvojice vzor-obraz ($\vec{X}_1,\vec{Y}_1$),…,($\vec{X}_m,\vec{Y}_m$).

Může nám připadat zbytečná neuronová síť, která dává na výstupu stejný tvar jako jsme přivedli na vstup. Hlavní přínos nacházíme, když na vstup přivedeme neúplný tvar.

3.6 Metody vyhodnocování dvou vektorů

U neuronových sítí můžeme předkládat na vstup více vzorů a výstup nám bude rozhodovat, do které třídy vzorů právě předkládaný vzor patří. Dále popsané metody spadají do geometrické představy.

Častou metodou bývá určení podobnosti na základě Euklidovské vzdálenosti. Každý vektor $\vec{X}_i$ můžeme zapsat takto:

Rovnice 4. $$\vec{X}_i = [x_{i1}, x_{i2}, ..., x_{iN}]^T$$

Horní index T znamená maticovou transpozicí. [5]

Vzdálenost dvojice vektorů z Euklidovského prostoru určíme ze vztahu:

Rovnice 5. $$d_{ij} = \left\| \vec{X}_i - \vec{X}_j \right\| = \left[\sum_{n=1}^{N} \left(x_{in} - x_{jn} \right)^2 \right]^{1/2}$$

Dalším měřítkem podobnosti je vnitřní součin.

Jsou-li dány dva vektory(s indexy i a j), pak jejich vnitřní součin určíme vyčíslením vztahu. [5]

Rovnice 6. $$\vec{X}_i^T \vec{X}_j = \sum_{n=1}^{N} x_{in} x_{jn}$$

3.7 Trénovací množina

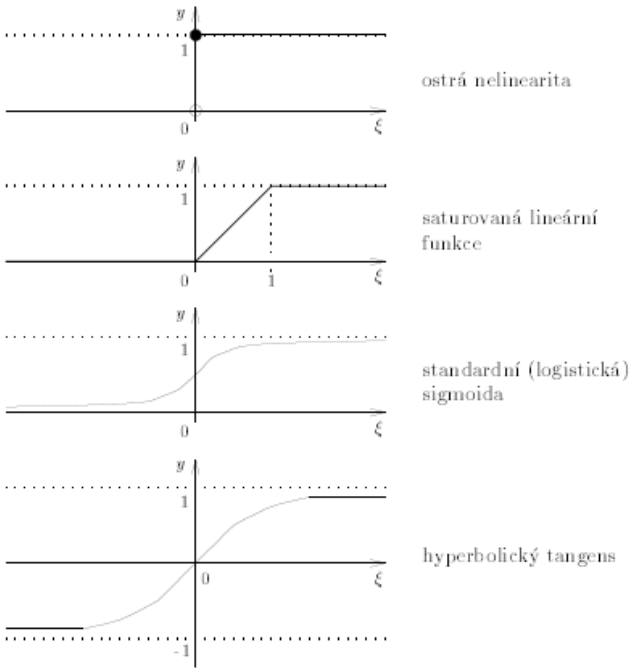
Všechny vstupní vektory tvoří vstupní množinu. Ta může být malá, ale jindy může mít i nekonečný počet prvků. Rozhodující význam pro proces učení má samozřejmě vstupní množina. Pokud nejsme schopni z nějakého důvodu dodat celou vstupní množinu, musíme aspoň vybrat reprezentativní data, která vystihují vstupní množinu. Této reprezentativní množině říkáme trénovací množina. Požadavek na to, aby učící množina vystihovala množinu vstupní reprezentativním způsobem, je jedním z největších oříšků, se kterým se v aplikaci neuronových sítí setkáme. [5] Pro výběr vzorů učící množiny můžeme použít jak sekvenční tak náhodný výběr. Některé vzory mohou být předkládány daleko častěji než jiné. Časový interval, při kterém síti předložíme každý vzor alespoň jednou, nazýváme epocha. Těchto epoch potřebujeme k procesu učení stovky až tisíce.

3.8 Kdy je síť naučena?

Za naučenou síť bereme ten případ, když je splněna zadaná podmínka. Za tuto podmínku můžeme brát dosažení předepsaného počtu trénovacích epoch, nebo dosažení shody chování sítě s nějakým exaktně zadaným kritériem. Kritériem často bývá i pokles globální chyby pod předem stanovenou mez. Stav naučenost sítě poskytuje i časový průběh chyby. Nezávislou proměnou zde nemusí být jen čas, ale můžeme použít i počet epoch učení. Obvyklý průběh globální chyby je, že okamžitá hodnota klesá. Ovšem neklesá konstantně, ale můžeme pozorovat úseky, kde okamžitá hodnota je určitou dobu ustálena na jedné hodnotě. Tento stav se objevuje nejčastěji kolem lokálních minim. Aby nedošlo k uváznutí (místo, kde prodlužování učení nemá smysl), používáme různé techniky proti uváznutí. Další možností je tzv. přeučení sítě . To se projevuje zvyšováním hodnoty globální chyby následující za jejím poklesem. Nejlepší východisko z této situace je ukončení učení a pokud je to možné, vrátit se k hodnotám které vykazovali nejlepší výsledky (nejmenší globální chybu). V případě, že jsme nemohli při učení využít celou množinu vstupních vektorů a museli jsme ji nahradit podmnožinou učicí, kontrolujeme kvalitu naučení testem na množině testování. [5] Tato množina musí být reprezentativní a počet prvků této množiny je několikrát menší, než prvky množiny učící.

3.9 Přenosové funkce

Přenosová funkce převádí vnitřní potenciál neuronu na nadefinovanou výstupní hodnotu. Tato přenosová funkce se občas označuje jako aktivační. Přenosová funkce dále určuje zda je výkonný prvek binární nebo spojitý. Jedním ze zástupců neuronové sítě, který zpracovává binární signál je klasický perceptor. Aktivační funkce preceptoru tedy pracuje jen s jednou ze dvou hodnot 0 a 1. Graf přenosové funkce je popsán na *Obr. 5*. (ostrá linearita). V některých případech je ale z implementačních důvodů výhodnější, je-li oborem hodnot neuronové sítě dvojice 1 a -1 (Hopfieldova síť). V tomto případě se jako aktivační funkce použije ta druhá. [5] Nejčastěji je ovšem obor vstupních a výstupních hodnot spojitý interval [0, 1]. *Obr. 5*. (standardní logistická sigmoida) Musí být ovšem zajištěny spojité vstupy a výstupy. Další analogií spojité funkce je hyperbolická tangenta. *Obr. 5*.



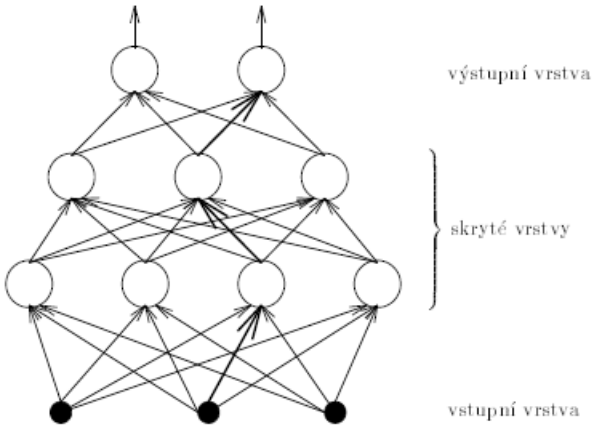
Obr. 5. Možné varianty aktivačních funkcí

3.10 Rosenblattův perceptron

Byl realizován v roce 1958 F. Rosenblattem. Inspiraci pro tento preceptor našel u lidského oka. Na povrchu oka jsou totiž světlo citlivá čidla, které jsou uspořádány do matice. Možnost rozpoznávat typy vzorů má geneticky specializovaná sada buněk tzv. démonů. Na

vstup démonů přivádíme signály z čidel. Naopak výstup z démonů dále zpracujeme v buňkách s prahovým chováním.

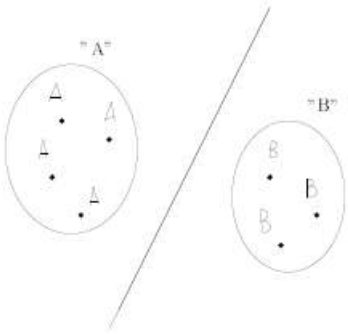
Samotná perceptronová síť je třívrstvá. Vstupní vrstva plní funkci rozvětvovací. Převádí tím vstupní hodnoty matice na jednorozměrný vektor procesorových elementů. Ve skryté vrstvě jsou detektory rysů, které mají za úkol detekovat specifické příznaky. Detektory jsou náhodně spojeny se vstupní vrstvou. Váhy vstupní a skryté vrstvy jsou nastaveny konstantně. Při procesu učení se mění pouze váhy u vstupů do výstupní vrstvy. Proto se bere zdánlivě třívrstvý Rosenblattův perceptron jako jednovrstvá síť.



Obr. 6. Vícevrstvá neuronová síť

3.11 MP – perceptron

McCulloch a Pitts zjednodušili Rosenblattův perceptron. Výsledkem je jednovrstvá neuronová síť, jejíž předností je, že se v ní dá velmi snadno geometricky interpretovat učící proces.



Obr. 7. Hraniční přímka

PRAKTICKÁ ČÁST

4 BLENDER

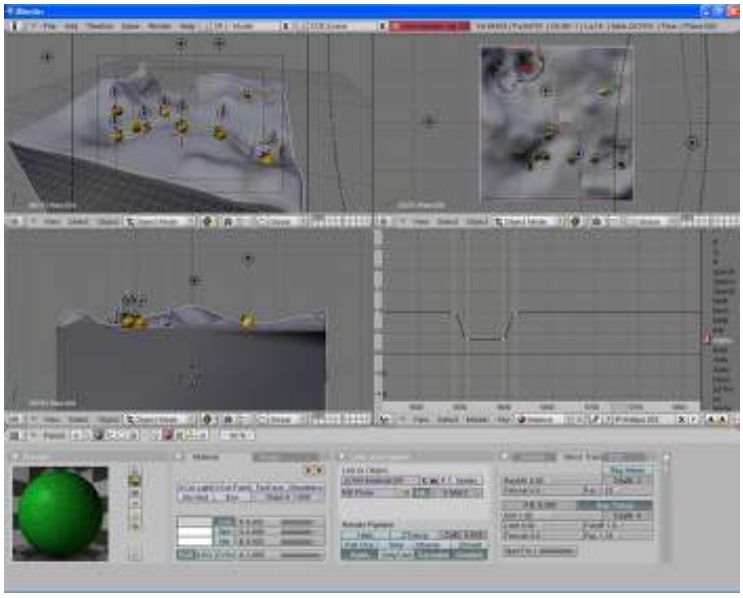
Než se podrobně podíváme na vytvoření konkrétních animací, bylo by dobré seznámit se s programem, ve kterém byly vytvořeny. Vzhledem k mnoha hlediskům jsem si vybral program na 3D grafiku Blender.

Obr. 8. Logo programu Blender

Blender je software pro 3D modelování, animaci, tvorbu her, rendering a přehrávání. Je to program, který je založen na knihovně *OpenGL*. [6] Díky tomu je multiplatformní, což znamená, že jde spustit nejen na operačním systému Windows, ale i pod Linuxem, Irisem, Sun Solarisem, Macem nebo OS X. Jeho největší výhodou je, že spadává do skupiny programů *Open source*. *Open source* znamená, že je zcela zdarma. A to i pro komerční využití. To je velká výhoda, protože licence jiných nástrojů např. (3D Studio Max, Maya, Cinema 4D) stojí desetitisíce až statisíce. Další výhodou *Open source* je v možnosti stáhnutí zdrojových kódů, které si můžeme zkompilovat nebo upravit na svojí sestavě a tím maximalizovat výkon. Dalším příjemným zjištěním je Blenderova hardwarová nenáročnost a velikost. Nainstalovaný program zabírá na harddisku přibližně 25 MB.

4.1 Rozhraní

Rozhraní Blenderu je vytvořeno jako plně nastavitelné prostředí, ve kterém je možno si vytvořit nespočet překrývajících se oken. Tyto okna se mohou lišit podle jejich funkce. Můžeme mít okno pro modelování, animační křivky, editace nelineárních video sekvencí, editace, obrazový/UV editor, textový editor, Python skripty atd. Dále můžeme nastavit několik jazyků a vektorových fontů písmen s podporou mezinárodních znakových sad. Na *Obr. 9*. můžeme vidět rozhraní se třemi pohledy (nárys, bokorys a půdorys), dále editor animačních křivek a nabídka materiálového nastavení.



Obr. 9. Rozhraní Blenderu

4.2 Modelování

V Blenderu máme možnost práce s různými 3D objekty např. krychle, koule, válec, kužel atd. Tyto objekty se v Blenderu označují jako *mesh*. S *mesh* objekty můžeme pracovat na úrovni *vertexů* (jednotlivé body), hran a ploch. Blender poskytuje 2D objekty jako je *Beziérovy* a *B-spline* křivky a texty. Zvláštní kategorií jsou meta objekty. Tyto objekty mají strukturu, kterou můžeme chápat jako zdroj statického pole, takže se mohou přitahovat nebo odpuzovat. Tím vytváříme hlavně oblé tvary. Pro modelování se používá mnoho editačních nástrojů např. *extrude*, *bevel*, *cut*, *spin*, *screw*, *warp*, *subdivide*, *noise* nebo *smooth*.

4.3 Animace

Pro tvorbu animace využívá Blender vkládání animačních klíčů do jednotlivých snímků. Animační klíče jsou vlastně transformace jako změna polohy, rotace a měřítka. Pomocí okna *Ipo* můžeme tyto klíče upravovat pomocí animačních křivek. Pro složitější animace můžeme deformovat daný model pomocí transformaci jednotlivých *vertexů*. To nám umožňuje funkce *Shape keys*. Pro animaci živých postav se používají deformační kosti s podporou dopředné i inverzní kinematiky, dále *pose editor* a editor nelineárních animaci s automatickou možností zacyklení. Blender obsahuje taky čističové efekty, které využívají deformátorů větru, gravitace, magnetické přitažlivosti/odpuzování a detekcí kolizí. Pro

pohyb kamery v animaci se nejčastěji používá funkce *constraints*. Touto funkcí dokážeme přesně sledovat pohybující se předmět.

4.4 Rendering

Renderování představuje souhrn matematických výpočtů fyzikálních zákonitostí, které ve scéně simulujeme. Výsledkem takových výpočtů pak je bitmapový obraz nebo animace, které by měli co nejvíce odpovídat našim představám. [6] V Blenderu je implementován velmi rychlý *ray tracer*. Navíc je integrována i podpora externího *ray traceru Yafray*. Na modely můžeme uplatnit několik druhů materiálových shaderů pro stínování povrchů. Dále *Ambient Occlusion* a možnost zvýraznění hran. Pro vzhled velmi reálných povrchů můžeme využít procedurální textury.

4.5 Soubory

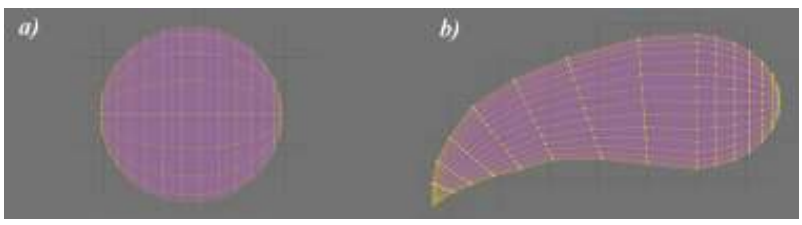
Všechno co vytvoříme v programu Blender se ukládá do souboru s příponou *.blend*. Ale výstup po renderování se zapisuje do formátu *tga*, *jpg*, *png*, *gif*, *tiff*, *psd*, *mov*, *Iris(+ Zbuffer*), *SGI Movie*, *iff*, *avi* a *Quicktime*. Dále má nativní podporu importu a exportu formátů *dxf*, Invertor a *vmrl* souborů.

5 VYTVÁŘENÍ MODELŮ

Pokud do scény vložíme některé ze základních *mesh* objektů, můžeme je dále upravovat.To nám dovolují dva módy. V objektovém módu se nejčastěji řeší různé transformace objektu jako celku. Druhý mód se nazývá editační a dovoluje nám pracovat s jednotlivými hranami, plochami a *vertexy*. Vybraný objekt se v tomto módu zobrazuje s fialovým obrysem. Přepínání mezi jednotlivými módy se provádí klávesou **Tab**.

5.1 Modelování pomocí transformací

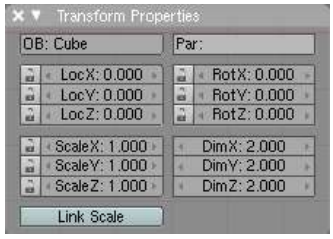
Transformace patří mezi nejzákladnější operace, které s objekty, nebo s jejich částmi provádíme. Patří sem posunutí, otočení, změna měřítka a zrcadlení. [6] Popis modelování pomocí transformací si popíšeme na jednoduchém objektu, který je používán v animaci SOMA jako obočí pro jedince. Do prázdné scény si vložíme *mesh* objekt koule. Tu pomocí transformace otočení otočíme o 90 stupňů kolem osy Y. Přepneme se do editačního módu, který znázorňuje *Obr. 10. a)*. Vybereme jednu část koule, která vypadá jako poledník na zeměkouli a provedeme na ní transformaci změny polohy. Tu aktivujeme tlačítkem **G**. Tento proces uplatníme na všechny části za polovinou objektu a posuneme je po ose X. Tím nám vznikne tvar podobný kapce. Tuto novou polohu potvrdíme klepnutím levého tlačítka myši. Další transformací je změna měřítka. Ta se vyvolá tlačítkem **S**. Opět vybereme příslušný objekt, nebo část objektu, v našem případě posunuté části. Po stisknutí klávesy **S**, pohybem myši určíme velikost zvětšení. Tuto změnu měřítka opět potvrdíme kliknutím levého tlačítka na myši. Poslední transformací je otočení. Rotace objektu se vyvolá pomocí tlačítka **R**. Označíme zvětšené a posunuté části koule a uplatníme na ně transformaci otočení. Výsledkem použitých transformací je tvar z *Obr. 10*. b).



Obr. 10. Editační mód modelování pomocí transformací

Pokud model vyžaduje přesné rozměry, můžeme tyto transformace zadávat i numericky. Vyvoláme-li nabídku *Object-Transform Properties* stiskem klávesy **N**. Dostaneme

nabídku, z *Obr. 11.* kde nastavíme základní parametry objektu. Změnou hodnot uvedených u polohy, rotace a velikosti můžeme objekty transformovat v jednotlivých osách X, Y, a Z.

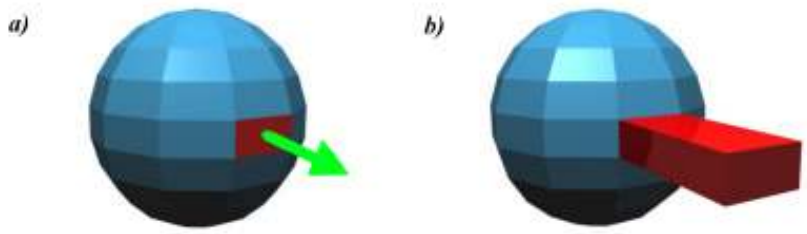


Obr. 11. Číselné zadávání transformací

Poslední transformací je zrcadlení. To umožňuje u symetrických objektů, jako např. tváře, automobily, hlava zvířete atd. že stačí vytvořit polovinu objektu a tu potom podle příslušné osy zrcadlit. Pokud se rozhodneme zrcadlit objekt, máme dvě možnosti. První je, že objekt jen zkopírujeme a ten se chová jako samostatná část. Tuto možnost vyvoláme klávesovou zkratkou **Alt + D**. Druhou možností je, že vytvoříme závislou kopii. V objektovém módu se chová samostatně, ale v editačním módu je spojená s originálem,tj. pokud provedeme nějakou operaci v editačním módu jednoho objektu, provede se totéž i u objektu druhého. Tuto možnost zvolíme klávesovou zkratkou **Alt + D**. U našeho modelu obočí tedy použijeme první možnost a zrcadlíme podle příslušné osy. Potom jen posuneme druhé obočí do polohy, do které potřebujeme.

5.2 Tažení

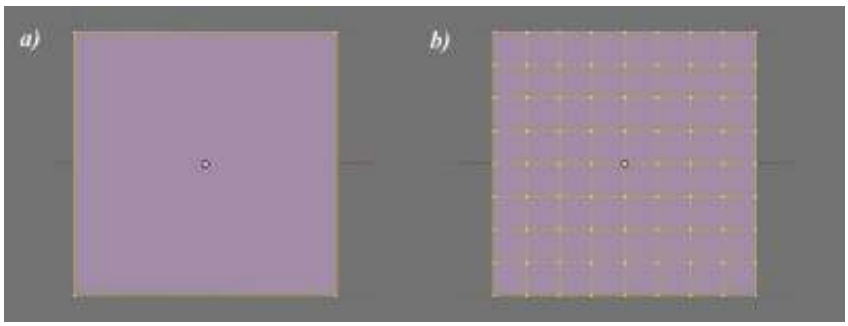
Tažení je metoda modelování, která je velmi oblíbená. Používá se pro tažení bodů, hran a ploch. Pokud si třeba vytvoříme obdélníkovou plochu, do které vytvoříme čtvercový otvor a následně tuto plochu vytáhnu, dostanu 3D objekt podobný stěně s oknem. Tažení aktivujeme tlačítkem **Extrude** na kartě *Mesh Tools*. Pro animaci jsem použil tažení u modelu neuronu, kde jsem z *mesh* objektu koule *Obr. 12. a)* „vytáhl" pomocí funkce *Extrude* červenou plochu. To ukazuje *Obr. 12. b)*.



Obr. 12. Ukázka taženi plochy z koule

5.3 Rozdělení

Každé složitější modely se zpravidla vytváří tak, že vytvoříme celkový tvar objektu a teprve poté se postupně dodělávají detaily. Funkce rozdělení (*subdivide*) přidává k vybraným částem objektu nové *vertexy*, které nám tyto detaily umožňují vytvářet. [6] Princip funkce *Subdivide* je takový, že každou vybranou hranu rozdělí na polovinu a přidá tím jeden *vertex*. Na *Obr. 13. a)* je do scény vložená plocha. Na tuto plochu použijeme třikrát funkci *Subdivide*. To vidíme na *Obr. 13.* v části *b)*. Funkci *Subdivide* najdeme na kartě *Mesh Tools*, nebo stačí stisknout klávesu **W**.



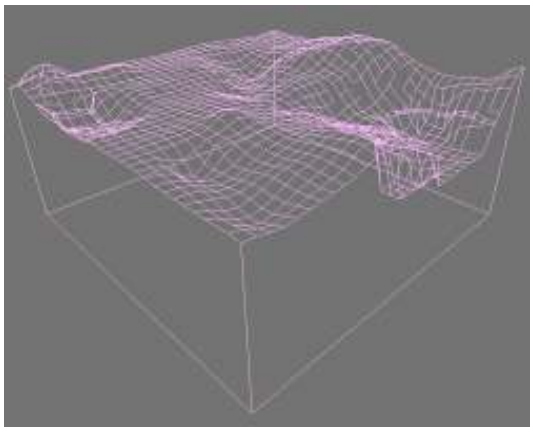
Obr. 13. Funkce Subdivide použitá na plochu

5.4 Magnetické pole

Magnetické pole je modelovací nástroj, který se používá pro vytvoření členité krajiny. Do této chvíle jsme se zabývali jen funkcemi, které ovlivňují jen označené *vertexy* a jim odpovídající hrany a plochy. Funkce magnetického pole ovšem umožňuje měnit polohu *vertexů*, které označeny nebyly. Kolem vybraného *vertexu* se objeví pole a všechny body uvnitř tohoto pole budou ovlivněny. Tato funkce se vyvolá ikonou mezikruží v pohledu *3D View*, nebo pomocí klávesy **O**. Pokud je funkce aktivovaná, vybereme *vertex* a nastavíme

intenzitu pole. Potom na tento *vertex* aktivujeme transformaci posunu. Při posunu *vertexu* sledujeme že se přesouvají i body uvnitř pole.

Tuto funkci jsem využil při vytváření modelu krajiny pro animaci SOMA. Do scény byla vložena krychle, jejíž horní plocha byla několikrát rozdělena nástrojem *Subdivide* (viz. kap. 5.3). Vybereme vždy *vertex*, který bude představovat vrchol kopce, nebo nejnižší místo v údolí, aktivujeme funkci magnetického pole a pomocí transformace přesunu změníme jeho polohu. Tím byla vytvořena krajina, kterou můžeme vidět jako drátoví model na *Obr. 14*. Stejný vyrenderovaný model s texturou trávy a skály ukazuje *Obr. 15*.



Obr. 14.Drátový model terénu

Obr. 15.Vyrenderovaný objekt terénu

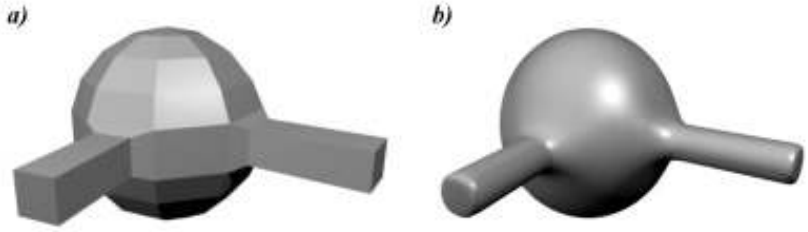
5.5 Subsurf

Tento nástroj najdeme mezi modifikátory. Zjednodušeně řečeno, vytváří z objektů plochých hladké s tím, že vyhlazenost si můžeme volit. [6] Musíme ovšem počítat s tím, že

pokud na objekt aplikujeme tento modifikátor, objekt se proti původnímu zmenší a naroste počet jeho *vertexů*. To je dáno změnou sítě v editačním módu. Na *Obr. 16*. je ukázka panelu se zvoleným modifikátorem *Subsurf*. Má několik nastavitelných parametrů, z nichž si uveďme tlačítko *Levels*, jehož hodnota reprezentuje mír vyhlazení.

Obr. 16.Nabídka pro modifikátor Subsurf

Tento modifikátor jsem použil pro vytvoření modelu neuronové sítě. Vytvořil jsem si jednoduchý model, který měl minimum *vertexů*, ten ukazuje *Obr. 17. a)*. Na něj jsem uplatnil *Subsurf* s úrovní vyhlazeností „*Levels : 4*". Výsledný objekt je vyobrazen na *Obr. 17. b)*



Obr. 17. Použití modifikátoru Subsurf na jednoduchý model

5.6 Rodičovská vazba

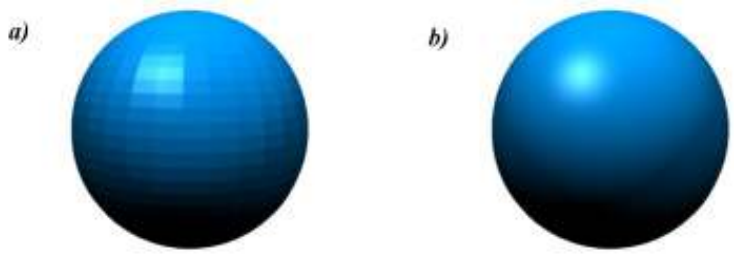
Rodičovská vazba není žádná modelovací technika, ale je to velmi užitečná funkce, která se ve scénách hodně využívá. Do teď jsme vše prováděli jen pro jeden objekt. V některých případech je potřeba použít více objektů, které budou nějakým způsobem seskupeny. Rodičovská vazba nám dovoluje vytvořit skupinu objektů, které mají mezi sebou určité vztahy. Tyto vztahy určují dvě vlastnosti objektu. První vlastnost je rodič a druhá je potomek. Pokud seskupíme objekty rodičovskou vazbou, pak když provedeme jakoukoliv transformaci s rodičovským objektem, projeví se tato změna i na potomcích. Opačně tato vazba neplatí. Blender dokáže vytvořit vícenásobnou rodičovskou vazbu. To znamená, že

například objekt, který je potomkem nějakého rodiče, může být pro jiné objekty sám rodičem. [6] Rodičovskou vazbu vytvoříme v Blenderu tak, že vybereme všechny objekty, které chceme seskupit a ten co vybereme jako poslední bude rodič. Potom stiskneme kombinaci kláves **Ctrl + P** a z nabídky vybereme *Make parent*.

Rodičovskou vazbu jsem uplatnil např. na modely jedinců v animaci SOMA, kde rodič je objekt žluté koule a potomci jsou objekty obočí,očí a úst. Všechny transformace v animaci tedy tvořím s objektem žluté koule a Blender si sám hlídá, aby se poloha potomků vůči rodiči nezměnila.

5.7 Způsoby zobrazení a materiálové nastavení

Všechny objekty v Blenderu můžeme zobrazovat ve dvojím vzhledu. První z nich je hrubý (deskový - *solid*) *Obr. 18. a)*. Druhým způsobem zobrazení je vyhlazený (*smooth*) *Obr. 18. b)*. Nastavení způsobu zobrazení se provádí v okně *Buttons Windows* v editačním menu tlačítky *Set Smooth* nebo *Set Solid*.



Obr. 18. Způsoby zobrazení Solid a Smooth

Materiálové nastavení, udává našemu objektu jeho vzhled. Obvykle nastavení optimálního vzhledu zabere daleko více času,než vytvoření modelu. Při renderu Blender využívá zpětné sledování světelného paprsku z osvětlení ve scéně. To znamená že určí barvu každého bodu (*pixelu*) pomocí průsečíku světla s libovolným objektem ve scéně. Vliv na výslednou barvu mají vlastnosti natavené u objektu. Mezi tyto vlastnosti patří barva, hrubost, průhlednost, zrcadlové odrazy, reflexy atd. Každý individuální paprsek, který dopadá na povrch nějakého objektu se může zachovat dvěma způsoby – může se okamžitě zrcadlově odrazit (tento jev se nazývá specularita - *specular*) nebo dochází k vícenásobnému odrazu a lomu (tento jev se nazývá difúze - *diffuse*). [6] Spekulární složka je příčinou odlesků na zobrazovaných objektech. Tyto odlesky jsou závislé na hladkosti povrchu. Difúzní složka závisí pouze na úhlu dopadu světelného paprsku a přináší pouze informaci o barvě

povrchů. Na *Obr. 19*. můžeme vidět povrch žluté koule na které je aplikována hrubost a oči s nastavením zrcadlového odrazu.

Další možností je vytvoření realistických povrchů pomocí textur. Textura je obrázek, který se obalí na povrch objektu. Blender používá pro textury formátu *jpg*, *bmp*, *gif* a *tga*. Nejčastěji se pro textury používají digitální fotografie. Ovšem můžeme použít i nastavitelné textury, které najdeme v okně *Buttons Window*. Jde o textury *Noise*, *Clouds*, *Marble*, *Stucci*, *Wood* atd. Texturu *Noise* jsem použil pro vytvoření povrchu, který připomíná vzhled trávy na *Obr. 19*. pro animaci SOMA.

Obr. 19. Materiálové nastavení na objektech

6 ANIMACE

Animace je způsob, jak dokážeme zdánlivě rozpohybovat námi vytvořené objekty. Princip animace spočívá v sekvenci statických obrázků, které se od sebe liší jen minimálně. Důležité je tyto snímky přehrávat takovou rychlostí, jakou už oko nepostřehne. Potom máme dojem, že se díváme na záznam reálného pohybu. V Blenderu můžeme tvořit jednoduché animace pomocí změny určitých vlastností v čase (například transformace polohy, otočení či změna měřítka). Tyto druhy animací se provádí tzv. klíčováním, tj. vkládáním animačních klíčů. [6] Tyto klíče můžeme editovat v nástroji zvaném *Ipo editor*, který nám hodně usnadní práci a manipulaci s animačními klíči. Pokud chceme, aby se animace projevila na jednotlivých *vertexech* objektu, použijeme pro animaci nástroj deformace objektů *Shape key*. Pro výstup animace se nejčastěji používá formát *AVI Codec*. Ten nám povolí použít kodeky, které máme nainstalované v našem počítači. Dále můžeme nastavit počet snímků za sekundu. Další variantou výstupu je formát bitmapového obrázku (například *PNG*) ze kterého se vytváří posloupnost souborů, z nichž každý přestavuje jeden snímek animace.

6.1 Jednoduchá animace

Jednoduchá animace se vytváří tzv. klíčováním. To znamená, že si vystačíme pouze s vkládáním animačních klíčů do scény. Aktuální hodnotu snímku nám ukazuje číslo v tlačítkovém oknu *Button window*. Pro jeho změnu můžeme číslo jednoduše přepsat, nebo stačí stisknout kurzorových kláves. Poslední věcí, kterou k vytvoření jednoduché animace potřebujeme, je znát vkládání animačního klíče. Ten vložíme klávesou **I**. Potom se objeví nabídka s možnostmi transformací polohy, otočení, změny měřítka atd. (*Loc*, *Rot*, *Size*, *LocRot*, *LocRotSize*, *Layer* a *Avail)*. Hodně záleží na jakém místě je kurzor myši. Pokud vložíme animační klíč s kurzorem myši na materiálovém menu, objeví se nám nabídka s novými možnostmi jako změna barvy, průhlednost atd. Poslední možností, je zadání animačního klíče nad nastavením pozadí scény. To umožní změnu *Horizontu*, *Zenithu*, *mlhy* a *hvězd*. Všechny nabídky vložení animačních klíčů ukazuje *Obr. 20*.

Obr. 20. Možné nabídky animačních klíčů

Samotná tvorba animace je taková, že v prvním snímku vložíme na objekt animační klíč. Potom změníme číslo snímku a provedeme příslušnou transformaci, která odpovídá animačnímu klíči který jsme vybrali. Nakonec vložíme opět stejný animační klíč. Blender sám vytvoří pohyb objektu, a to tak, že příslušná vlastnost animačního klíče se bude rovnoměrně provádět v rozmezí nastavených snímků.

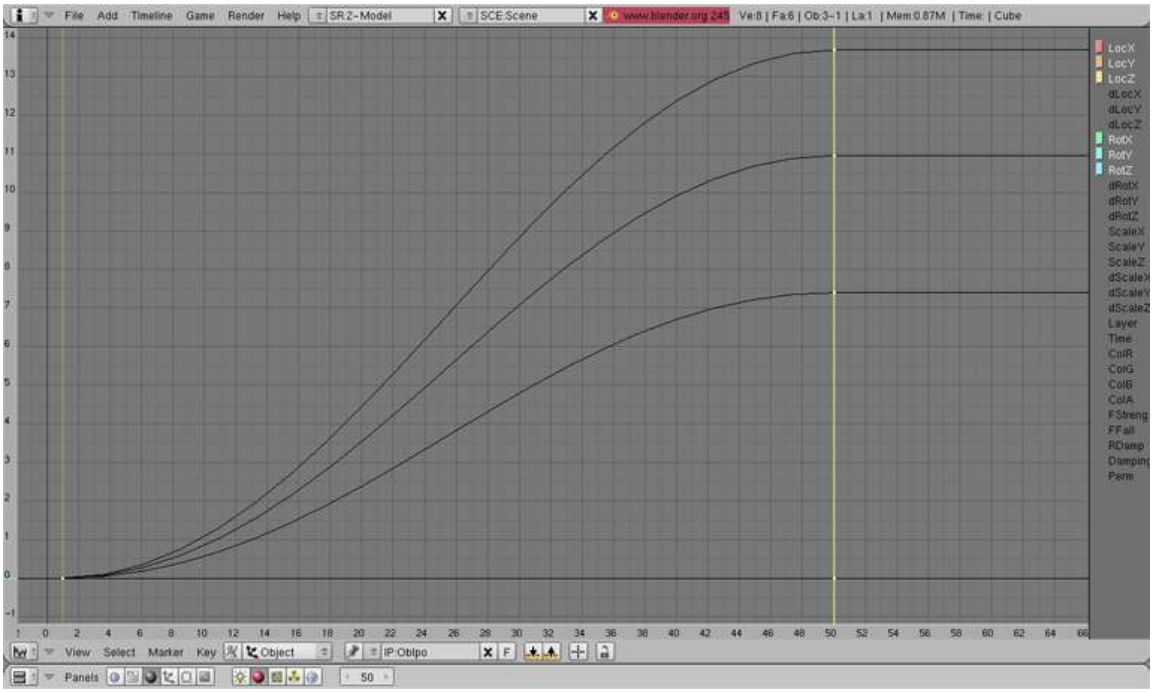
Tuto možnost animace jsem použil pro všechny pohybující se objekty ve scéně. Jako názorný příklad uveďme např. pohyb jedinců v animaci SOMA, pohyb kamery, pohyb světelné informace v animaci neuronových sítí atd.

Popišme si konkrétní ukázku tvorby jednoduché animace pro neuronovou síť. Označím si objekt koule. Stisknu tlačítko **I** a z nabídky vyberu možnost *Loc*. Potom změním hodnotu snímku na hodnotu 50. Objekt přesunu do pozice, do které se potřebuji dostat. Znovu vložím animační klíč *Loc*. Protože počet snímků za sekundu je na hodnotě 25, bude mít tato animace pohybu koule délku 2 sekundy.

6.2 Animace v IPO editoru

Tento editor se používá pro správu animačních klíčů při rozsáhlých animacích. Je tvořen dvěma osami. Na ose Y se zobrazují obvykle transformace, změny barvy, průhlednosti atd. Osa X představuje časovou osu animace. *Ipo editor* vyvoláme změnou okna *3D view* na *Ipo Curve Editor*. Pokud si zobrazíme animaci, kterou jsme popsali v kap. 6.1., uvidíme v *Ipo Editoru* tři křivky. Každá křivka zobrazuje změnu polohy v určité ose. Na každé křivce jsou 2 tečky. Ty představují vložené animační klíče. Když stiskneme klávesu

K, proloží se tyto body přímkou. Tento stav ukazuje zobrazený *Ipo editor* na *Obr. 21*. Pokud na tuto přímku klikneme a přepneme se do editačního módu, můžeme ji posunovat po časové ose. Tím vlastně změníme délku animace vložených klíčů. Pro tuto krátkou animaci tento editor nemá význam, ale pro situace, kdy máme více objektů, které mají několik desítek vložených animačních klíčů, je nepostradatelný.



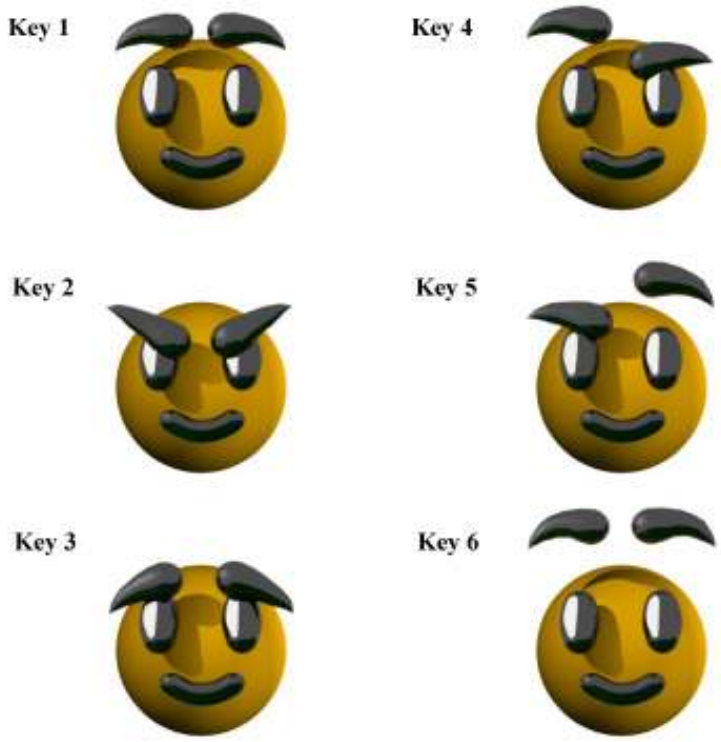
Obr. 21. Ipo editor pro jednoduchou animaci

6.3 Animace deformací objektů Shape key

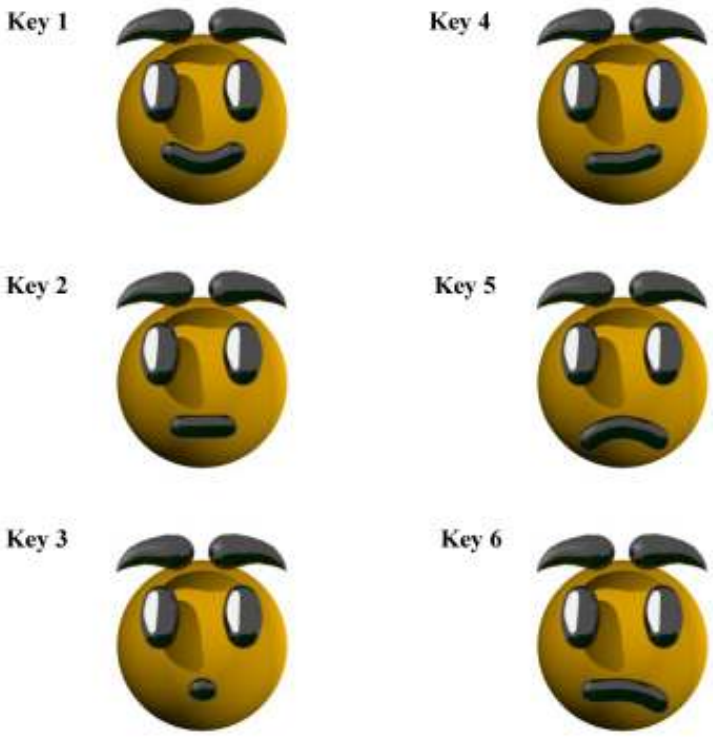
Další hodně používanou animační metodou v Blenderu je deformace objektů. To nám dovoluje měnit tvar objektu v reálném čase. To se používá hlavně pro animaci mimiky. V praxi to potom vypadá tak, že si vytvoříme několik různých tvarů modelů (například úsměv, zamračení, mrknutí okem apod.), které označíme jako tzv. klíče. [6] Tyto klíče potom můžeme vkládat na časovou osu a tím docílit přechodu od úsměvu k zamračení.

Tuto metodu jsem použil pro model neuronové sítě a pro mimiku jedinců v animaci SOMA. Ukažme si jednoduchý příklad na jedinci z animace SOMA. Označíme objekt obočí a přepneme se do editačního módu. Zde označíme všechny *vertexy* a vložíme animační klíč pomocí klávesy **I**. Z nabídky si tentokrát vybereme možnost *Mesh*. Tím vložíme základní tvar modelu. Teď provedeme nějakou transformaci. Pootočíme vybrané obočí a následně ho posuneme níž. Dále označíme jednotlivé kruhy v *mesh* síti obočí, a

pomocí pootočení a rotace je vyrovnáme do tvaru kapky. Opět označíme všechny *vertexy* a vložíme animační klíč *Mesh*. Tím vytvoříme změnou tvaru a polohy *mesh* první klíč (*Shape key*), který znázorňuje zamračení. Stejným způsobem vytvoříme klíče pro mrknutí, údiv, smích atd. (viz. *Obr. 22*.). Podobně jsem postupoval u vytvoření klíčů pro objekt pusy (viz. *Obr. 23*.). Kombinací těchto klíčů můžeme vytvořit výrazy obličeje, které jsou obsažené v animaci SOMA. U animace neuronové sítě jsem jako klíče (*Shape key*) vkládal změny velikostí jednotlivých *mesh* částí válců a koulí.



Obr. 22. Klíče (Shape key) pro objekt obočí



Obr. 23. Klíče (Shape key) pro objekt pusy

ZÁVĚR

Všechny úkoly, které jsme si v zadaní vytýčili, byli splněny. Hlavním úkolem práce bylo vytvoření 3D modelů neuronových sítích, terénu a jedinců algoritmu SOMA. Dále tyto modely použít pro krátké názorné animace, které popíší procesy v neuronových sítích a algoritmu SOMA.

Pro vytvoření modelů a animací jsem si zvolil program na 3D grafiku Blender. Výstup animací byl nastaven na 25 snímků za vteřinu, aby byl zachován plynulý chod animace. Jako formát animace jsem zvolil *AVI Codec*, který zaručí, že výstupní soubor nebude mít enormně velkou kapacitu.

Pro algoritmus SOMA byla použita kvůli názornosti animace nejjednodušší metoda *All to One*. Model neuronové sítě byl vytvořen jako síť se vstupní, skrytou a výstupní vrstvou, kde impulsy byli nahrazeny barevnými svítícími kuličkami.

Samotné animace by se mohly stát názornou pomůckou pro výuku studijního předmětu na univerzitě a pomoci tím studentům k pochopení tohoto problému.

ZÁVĚR V ANGLIČTINĚ

All targets which has been to trace, has been completed. The main target of task has been to create 3D models of neural network, landscape and individuals of algorithm SOMA. Also these models should be used for short animations, which should be to describe processes in neural networks and algorithm SOMA.

I select 3D programme Blender to create a models and animations. The output has been set up with 25 frames per second to keep the smooth run of animation. I select an *AVI Codec* like the format of animation, which guarantees that output file has not had an excessive capacity.

For SOMA algorithm has been used for plasticity animations of basic method *All to One*. Model of neural network has been built like network with input, hidden and output layer, where the impulse has been replaced by the coloured shining bullets.

Single animations should be become objective utilities for part of study program on university and it should be helped to students for understanding this problem.

SEZNAM POUŽITÉ LITERATURY

[1] ZELINKA I. 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] KVASNIČKA V., POSPÍCHAL J., TIŇO P. 2002 *Evoluční algoritmy*, STU Bratislava, 2000, ISBN 85-246-2000.

[3] VAŘACHA, Pavel. Syntéza neuronových sítí metodou symbolické regese. Zlín, 2006. 83s. UTB. Vedoucí Diplomové práce doc. Ing. Ivan Zelinka, Ph.D.

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

[5] ŠNOREK, I. 2002 *Neuronové sítě a neuropočítače*, Praha : ČVUT, 2002, 156 s.

[6] POKORNÝ, Pavel. BLENDER – Naučte se 3D grafiku. Praha : BEN, 2006. 248 s. ISBN 80-7300-203-5.

[7] 3D scéna [online]. 2003 [cit. 2008-01-28]. Dostupný z WWW: http://www.3dscena.cz/3dshowks.php?xuid=207.

[8] Blender [online]. 2001 [cit. 2008-01-28]. Dostupný z WWW: http://www.blender.org/education-help/tutorials/.

[9] ŠÍMA, Jiří, NERUDA, Roman. *Teoretické otázky neuronových sítí*. Praha : MATFYZPRESS, 1996. 390s. ISBN 80-85863-18-9.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

Bevel	Zaoblení
Cut	Rozříznutí
D	Řídící parametr algoritmu SOMA určující počet argumentů účelové funkce
Extrude	Vytažení
Ipo	Editor, pro usnadnění editace animačních klíčů v animaci
Level	Úroveň
Mass	Řídící parametr algoritmu SOMA určující jak daleko se aktivní jedinec zastaví
Mesh	Síťový povrch objektu
NP	Řídící parametr algoritmu SOMA určující počet jedinců v populaci
pixel	Obrazový bod
Rendering	Vykreslování
RPT	Řídící parametr algoritmu SOMA určující perturbační vektor
Screw	Rotace po šroubovici
Smooth	Hladký
SOMA	Samo-Organizující se Migrační Algoritmus
Spin	Rotace kolem osy
Step	Řídící parametr algoritmu SOMA určující podrobnost prohledávání
Subdivide	Rozdělení
Subsurf	Vyhlazení
Vertex	Bod nebo vrchol
w	Synaptická váha neuronu
x	Vstupy neuronu
XOR	Logická funkce
y	Výstup neuronu

3D	Zobrazení v $\mathbb{E}^3$
ϕ	Práh neuronu
ξ	Vnitřní potenciál neuronu

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: ADRESÁŘOVÁ STRUKTURA PŘILOŽENÉHO CD

Pro lepší orientaci v adresářové struktuře přiloženého CD, přikládám tento stručný popis.

- **Prace** – tento adresář obsahuje tuto práci ve formátu doc a pdf
- **Animace** – Kořenový adresář pro všechny animace
- **Animace****Soma** – Obsahuje animaci SOMA.avi a zdrojové soubory Blenderu, SOMA.blend
- **Animace****Biological neuron** – Obsahuje animaci Biological.avi a zdrojové soubory Blenderu, Biological.blend
- **Animace****Neural network** – Obsahuje animaci Neural.avi a zdrojové soubory Blenderu, Neural.blend
- **Animace****Neural learning** – Obsahuje animaci SOMA.avi a zdrojové soubory Blenderu, SOMA.blend