VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

KARTÉZSKÉ GENETICKÉ PROGRAMOVÁNÍ V EVOLUČNÍM UMĚNÍ

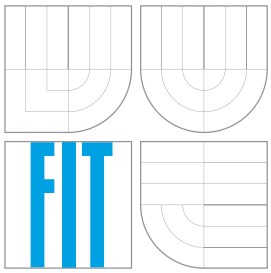
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PAVEL VESELÝ

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

KARTÉZSKÉ GENETICKÉ PROGRAMOVÁNÍ V EVOLUČNÍM UMĚNÍ

CARTESIAN GENETIC PROGRAMMING IN EVOLUTIONARY ART

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PAVEL VESELÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JIŘÍ PETRLÍK

BRNO 2015

Abstrakt

Tato práce se zabývá využitím kartézského genetického programování (CGP) v evolučním umění (EvoArt). Text předkládá úvod do problematiky. Dále se zaměřuje na proces návrhu, implementace a testování nové metody, jak CGP v EvoArtu využít. Navržená metoda využívá CGP pro generování 2D vektorové grafiky. Praktickou částí je webová aplikace pro tvorbu EvoArt, která metodu implementuje. V závěru jsou zhodnoceny dosažené výsledky.

Abstract

This thesis deals with use of Cartesian Genetic Programming (CGP) in Evolutionary Art (EvoArt). Text presents introduction to the topic. The rest of the thesis focuses on the process of design, implementation and testing of new method of application CGP in EvoArt. The proposed method uses CGP to create 2D vector images. Web application for EvoArt creation is made to demonstrate this method. Achieved results are presented and evaluated.

Klíčová slova

kartézské genetické programování, CGP, evoluční umění, genetické programování, evoluční algoritmy

Keywords

cartesian genetic programming, CGP, evolutionary art, genetic programming, evolutionary algorithms

Citace

Pavel Veselý: Kartézské genetické programování
v evolučním umění, bakalářská práce, Brno, FIT VUT v Brně, 2015

Kartézské genetické programování v evolučním umění

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana inženýra Jiřího Petrlíka.

.....
Pavel Veselý
19. května 2015

Poděkování

Na tomto místě bych chtěl poděkovat vedoucímu své práce Ing. Jiřímu Petrlíkovi za cenné rady a pomoc při tvorbě této práce. Dále bych chtěl poděkovat své rodině za podporu v průběhu celého studia.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Evoluční umění je generované programově s využitím evolučních algoritmů. Uživatel vytváří objekt tak, že směřuje jeho postupný vývoj od náhodného, nezajímavého začátku k uměleckému dílu. Dílem může být 2D obraz, 3D objekt nebo dokonce hudba.

V první kapitole popisuji problematiku evolučních algoritmů a evolučního umění. Čtenář se seznámí s principy evolučních algoritmů. Text se zaměřuje především na kartézské genetické programování a jeho využití v evolučním umění.

Zbytek textu se zabývá procesem vývoje nové metody jak kartézské genetické programování aplikovat v oblasti evolučního umění, konkrétně pro tvorbu vektorové 2D grafiky.

Druhá kapitola popisuje návrh metody, zvažované alternativy a výslednou metodu. Třetí kapitola se zabývá implementací webové aplikace, která navrženou metodu aplikuje. Čtvrtá kapitola se věnuje testování metody. V páté kapitole jsou zhodnoceny výsledky navržené metody ve srovnání s metodou využití CGP pro tvorbu rastrových obrázků. V závěru jsou dosažené výsledky shrnuty.

Kapitola 2

Teoretický základ

2.1 Optimalizace

2.1.1 Optimalizační problém

Optimalizační problém je problém nalezení nejlepšího ze všech možných řešení. Nechť je funkce f : D→R, přičemž D představuje n-rozměrnou kostku. Hledáme pak takové x∈D, pro které bude f(x) minimální, resp. maximální. Dále uvažujeme hledání globálního minima.

Nalezení globální minima použitím klasických optimalizačních metod patří mezi těžko řešitelné problémy pro funkce, které nejsou ohraničeny dalšími podmínkami. Z toho důvodu se používají evoluční optimalizační metody, které poskytují řešení blízké globálnímu minimu nebo s ním totožné.[9]

2.1.2 Slepý algoritmus

Nejjednodušším stochastickým algoritmem je slepý algoritmus. Ten opakovaně generuje náhodné řešení z oblasti D a zapamatuje si jej pouze v případě, že je lepší než řešení už dříve zaznamenané.

Stochastický optimalizační algoritmus poskytuje korektní globální minimum v případě, že doba běhu programu roste k nekonečnu. Slepý algoritmus neobsahuje žádnou strategii konstrukce řešení na základě předcházejícího běhu algoritmu. Každé řešení je naprosto nezávislé. Jeho reálná použitelnost při konečné době běhu algoritmu tedy není ideální.[9]

2.1.3 Horolezecký algoritmus

Horolezecký algoritmus iterativně hledá nové řešení pouze v určitém okolí již nalezeného řešení. Pokud narazí na lepší, stává se toto novým středem prohledávané oblasti. Bude-li struktura řešení bitový řetězec, můžeme okolní řešení definovat jako řetězce, které se liší pouze v malém počtu bitů. V ten okamžik odpovídá krok vytvoření nového řešení mutaci v genetickém algoritmu, jak je definovaná později.

Hlavním nedostatkem horolezeckého algoritmu je jeho náchylnost k uvíznutí v lokálním minimu. Horolezecký algoritmus následuje gradient funkce a v případě, že existuje globální minimum mimo okolí středu prohledávání, ale všechny hodnoty v okolí středu prohledávání jsou vyšší, nikdy neopustí nalezené lokální minimum.[9]

2.2 Evoluce

Evoluce je proces změny zděděných znaků v populaci v průběhu mnoha po sobě následujících generací. Současné poznání tohoto procesu vychází z teorie, kterou představil v roce 1859 Charles Darwin ve své knize O původu druhů.

Darwinova teorie je založená na přirozeném výběru. Každá populace existuje v prostředí s omezenými zdroji. Po dosažení limitu daného množstvím zdrojů vzniká konkurence a boj o přežití. Pokud v populaci existují dědičné odchylky, potom přežijí jedinci s výhodnějšími odchylkami a budou se rozmnožovat na úkor jedinců s méně výhodnými odchylkami. Přežijí nejsilnější jedinci a jejich potomci zdědí jejich výhodnější rysy. Celá populace se tak v průběhu generací adaptuje na okolí.

Další část teorie doplnil Gregor Mendel, zakladatel genetiky. Podle genetického determinismu, který je spojením těchto dvou teorií, probíhá proces evoluce na základě přirozeného výběru a náhodných genetických mutací.[11]

2.3 Evoluční algoritmy

Evoluční algoritmy jsou založené na metafoře evoluce. Hledání globálního minima (maxima) je zde převedeno na proces evoluce náhodně vygenerovaných řešení. Každé řešení je zakódováno do řetězce symbolů (parametrů). V terminologii evoluce pak mluvíme o jedinci, případně chromozomu.

Proces evoluce pak představuje cyklus ohodnocení jedinců populace hodnotou fitness a vygenerování nové generace na jejich základě. Pravděpodobnost, že se jedinec stane rodičem nové generace, závisí na jeho fitness hodnotě. Do tvorby nových jedinců také vstupuje náhodný prvek mutace. Stejně jako v biologické evoluci, i zde kombinace reprodukce, která vnáší rozmanitost nových řešení, a selekce nejlepších jedinců vede k obecnému zlepšení fitness nové generace.

Evoluční algoritmy jsou výrazně robustnější než například horolezecký algoritmus. Stochastický vliv reprodukce brání rychlému uvíznutí v lokálním minimu. I zde je ale riziko předčasné konvergence algoritmu.

2.4 Dále použité pojmy

Genotyp je soubor všech znaků, projevených i neprojevených, zakódovaných v genetické informaci jedince.

Fenotyp je soubor všech pozorovatelných znaků jedince. U velkého množství genetických algoritmů není rozdíl mezi genotypem a fenotypem a v některé literatuře se výraz fenotyp používá pro jedince. V kontextu tohoto textu ale fenotyp představuje tu část genotypu, která se přímo projevuje na chování jedince a ovlivňuje jeho fitness.

Fitness je relativní schopnost jedince přežít v prostředí a reprodukovat se. V kontextu genetického programování se chápe jako relativní schopnost jedince plnit požadované úlohy.

2.5 Genetický algoritmus

Myšlenku genetického algoritmu poprvé zformuloval R. Holland, který jej použil při studiu adaptivního chování.

Genetický algoritmus je asi nejpoužívanější evoluční algoritmus. Jeho hlavní využití je v optimalizačních úlohách, především při hledání globálního extrému v jedno a vícedimenzionálních funkcích. Chromozomy pak představují binární řetězec fixní délky, ve kterém jsou zakódované hodnoty jednotlivých vstupních proměnných funkce.

GA velmi blízce napodobuje biologickou evoluci. Při selekci je typicky pravděpodobnost, že se jedinec stane rodičem pro novou generaci, přímo úměrná jeho fitness relativně k ostatním jedincům generace. Reprodukce stojí především na (typicky jednobodovém) křížení, mutace probíhá s relativně malou pravděpodobností. Nová populace může být složena čistě z nových jedinců, velmi často ale část jedinců staré generace přechází do nové nezměněna.[11]

2.5.1 Algoritmus

Velmi obecně je možné genetický algoritmus zapsat následujícím způsobem:

1. Vynuluj hodnotu počítadla generací t=0.
2. Náhodně vygeneruj počáteční populaci P(0).
3. Vypočítej hodnotu fitness každého jedince populace P(t).
4. Vyber jedince z P(t) a z nich vygeneruj novou populaci P(t+1) použitím operátorů křížení a mutace.
5. Zvyš hodnotu počítadla generací t=t+1.
6. Pokud je t rovno maximálnímu počtu generací nebo je splněné jiné ukončovací pravidlo, vrať populaci P(t); jinak pokračuj krokem číslo 3.

[5]

2.5.2 Hodnocení kandidátních řešení

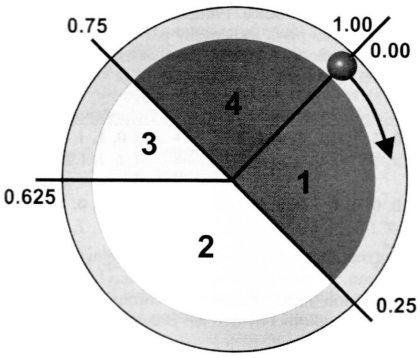
Fitness funkce je funkce hodnotící kvalitu jedince vzhledem k zadanému problému. Z tohoto důvodu je třeba pro každý reálný problém navrhnout specifickou funkci. Při návrhu je navíc potřeba najít kompromis mezi přesností fitness funkce, tedy jak podrobně bude popisovat problém, a její výpočetní náročností a počtem argumentů. Pokud bude funkce příliš komplikovaná, roste extrémně výpočetní čas evolučního algoritmu.

2.5.3 Selekce

Po vyhodnocení fitness hodnoty jednotlivých jedinců v populaci je potřeba rozhodnout, kteří z nich se stanou rodiči pro novou generaci, případně které přejdou do nové generace beze změny. Tato část evolučního procesu imituje přirozený výběr v biologické evoluci.

Lze očekávat, že potomci kvalitních jedinců se budou vyznačovat ještě lepšími vlastnostmi a ještě vyšším hodnocením. Zároveň je ale nutné zachovat v populaci dostatečnou rozmanitost. Pokud je výběrové kritérium příliš úzce zaměřené na nejlepší jedince, může algoritmus předčasně konvergovat k lokálnímu minimu. Pokud je naopak příliš volné, bude postup algoritmu velmi pomalý.

V generativním GA je stará populace plně nahrazena potomky. V případě částečné obnovy se nahradí vždy pouze nejslabší člen populace. Velmi běžné jsou smíšené varianty, kdy je 20-50% populace nahrazeno potomky.



Obrázek 2.1: Schéma ruletové metody selekce s pravděpodobností výběru přímo úměrnou fitness [5]

Ať už se vybraný jedinec stane rodičem nebo přechází do nové populace beze změny, je potřeba jej v první fázi vybrat. Některé typické mechanismy selekce jsou tyto:

Ruletový (proporcionální) mechanismus

Jedná se o zřejmě nejběžnější implementaci přirozeného výběru. Pro každého jedince nové populace se náhodně vybírá rodičovský jedinec. Každý jednotlivý výběr je nezávislý. Pravděpodobnost výběru jedince je závislá na jeho hodnotě fitness.

V nejjednodušší variantě je pravděpodobnost výběru přímo úměrná jeho hodnotě fitness relativně ke zbytku populace. Další běžnou variantou je exponenciální uspořádání, kdy je pravděpodobnost rozložena exponenciálně v závislosti na fitness.

Turnajová selekce

Z populace se vybere N jedinců, kteří přímo přejdou do další generace nebo se stanou rodiči. Pro R náhodných dvojic realizuje „turnaj“, kdy se porovnají jejich fitness hodnocení. Vítěznému jedince se zvýší skóre vítězství o 1. Na konci se jedinci seřadí podle skóre a vybere se N nejlepších.

2.5.4 Genetické operátory

Jednotlivé evoluční algoritmy se liší politikou výběru rodiče pro novou generaci. Nicméně způsoby, jak z jednoho nebo více vybraných jedinců vygenerovat nové jedince, jsou dva: křížení a mutace.

Křížení

Operátor křížení je charakteristický pro genetické algoritmy. Vychází z biologického principu sexuální reprodukce. Výsledný jedinec nebo jedinci tedy mají standardně dva rodiče, mezi kterými dojde k výměně (rekombinaci) genetické informace.

Křížení je potenciálně schopné vytvářet potomky s výrazně vyšší fitness než oba rodiče. Oba rodičovské chromozomy se ale musí lišit ve velkých částech genetické informace. Pokud jsou si z tohoto pohledu velmi podobní, budou i hodnoty fitness potomků velmi podobné fitness rodičů.

Při jednobodovém křížení se náhodně určí bod, ve kterém se oba rodičovské chromozomy rozdělí a vymění si část za bodem rozdělení.

$AA + BB \rightarrow AB + BA$

U vícebodového křížení se pouze použije více bodů, ve kterých se rodiče rozdělí.

$AAAA + BBBB \rightarrow ABAB + BABA$

Někteří z nově vzniklých jedinců budou mít pravděpodobně vyšší fitness než oba jejich rodiče. Nicméně rekombinace genetické informace nevnese do populace znaky, které nemá žádný jedinec v populaci. Jedno a vícebodové křížení tedy vede k poměrně rychlé konvergenci algoritmu a může být náchylné k uvíznutí v lokálních maximech.

Při uniformním křížení se provádí záměna jednotlivých genů s určitou pravděpodobností.

Oproti jednobodovému křížení je potenciálně genotyp potomků výrazně odlišnější od obou rodičů. To vede k větší rozmanitosti v nové generaci a obecně pomalejší konvergenci algoritmu.

Mutace

Při mutaci se náhodně vybere gen a nastaví se mu jiná, náhodná hodnota. Při bitové reprezentaci chromozomu je novou hodnotou velmi často inverzní hodnota bitu.

Mutace je schopná vnést do populace naprosto novou informaci. Na druhou stranu může do populace vnášet nestabilitu a zvláště u některých reprezentací (exponent reálných čísel) může naprosto srazit fitness hodnotu nového jedince.

2.6 Genetické programování

Genetické programování (GP) je evoluční algoritmus, o jehož rozvoj se nejvíce zasloužil John Koza koncem 80. let 20. století. Genetické programování navazuje na jiné evoluční algoritmy, ale zvyšuje komplexitu datových struktur, nad kterými evoluce probíhá. Namísto optimalizace funkce pro několik parametrů představuje GP evoluci spustitelných struktur. Konkrétně se jedná o obecné, hierarchické počítačové programy proměnné délky a tvaru, v některých implementacích pak elektronické obvody.

Je široká paleta problémů, které nelze zformulovat jako přímočaré hledání optimálních hodnot parametrů, ale hledání ideálního programu, resp. algoritmu řešení, už možné je. GP k těmto problémům přistupuje jako k hledání ideálního programu v množině všech možných počítačových programů.

2.6.1 Reprezentace jedince

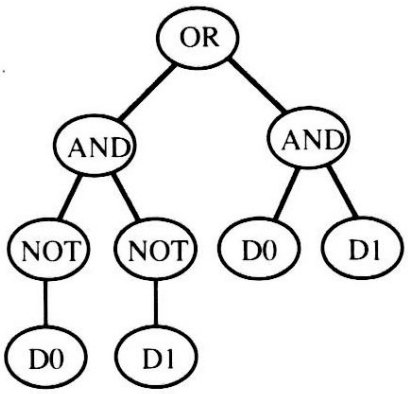
Genetické programování pracuje se spustitelnými hierarchickými programy. Ty jsou reprezentovány jako stromové struktury. John Koza používá S-výrazy jazyka LISP, GP jako takové je ale nezávislé na konkrétním jazyce.

GP pracuje s přímo spustitelnými programy. Veškeré vstupy a výstupy a následné hodnocení fitness tak mohou přímo odpovídat doméně řešeného problému.

2.6.2 Reprodukce

Vzhledem k hierarchické stavbě evolvovaných struktur je potřeba přizpůsobit i operátory reprodukce.

Křížení probíhá náhodným výběrem uzlů v obou rodičovských stromech a výměnou podstromů určených těmito uzly.



Obrázek 2.2: Stromová reprezentace jedince v GP [6]

Mutace může nabírat několika podob. Uzlová mutace nahradí neterminální uzel neterminálem se stejným počtem argumentů nebo terminál jiným terminálem. Jiné druhy pak mění tvar stromu.

2.6.3 Fitness funkce

Je běžné, že fitness každého jedince je měřena několikrát za různých podmínek. Výsledná fitness hodnota je pak sumou těchto měření. Jednotlivé případy podmínek měření mohou například představovat různé hodnoty nezávislé proměnné nebo počáteční stav systému.

Například v případě, že je jedincem program přijímající symboly na vstupu, budou jednotlivé měřené případy představovat různé vzorky těchto vstupů. Tyto vzorky mohou být generovány buď náhodně nebo mohou být nějakým způsobem strukturované. Volba opět závisí na znalosti reálného problému.

2.6.4 Algoritmus

Princip genetického programování se dá zjednodušeně zapsat takto:

1. Vytvoř počáteční populaci náhodných stromů složených z funkcí a terminálů z domény řešeného problému (počítačových programů).
2. Dokud nejsou splněny ukončující podmínky, iterativně prováděj tyto kroky
   (a) Spusť každý program v populaci a ohodnoť jeho fitness v závislosti na tom, jak dobře plní zadaný úkol
   (b) Vyber programy z populace s pravděpodobností přímo úměrnou jejich fitness. Vytvoř novou populaci použití následujících operátorů na vybrané programy
      i. Zkopíruj program do nové populace
      ii. Vytvoř nový program genetickou rekombinací náhodně vybraných částí dvou existujících programů
3. Nejlepší program každé generace je uložen jako výsledek GP. Tento výsledek může být řešením nebo částečným řešením řešeného problému.

[6]

2.7 Kartézské genetické programování

Kartézské genetické programování poprvé představili Julian F. Miller a Peter Thomson v roce 1999. Jedná se o variantu genetického programování, kde jsou kandidátní řešení reprezentovaná jako obecné orientované grafy.

Kartézské genetické programování (CGP) vychází z genetického programování. Odlišuje se od něj ale v několika velmi důležitých ohledech.

Prvotní a stále velmi časté užití CGP je návrh kombinačních obvodů. CGP si ale našlo i řadu dalších uplatnění.[12]

2.7.1 Reprezentace jedince

Genetickou informaci jedince v CGP představuje orientovaný, typicky acyklický graf (Obr.2.3). Genotyp fixní délky (počtu uzlů) se skládá ze vstupních uzlů umístěných na začátku, běžných funkčních uzlů a výstupních uzlů umístěných na konci.

Vstupní uzly sémanticky představují vstupy pro reálné řešení, ale v genotypu nemají proměnný parametr. Výsledné uzly obsahují pouze jeden parametr a sice odkaz na dřívější uzel představující hranu grafu. Funkční uzly obsahují několik parametrů: odkazy na jiné (typicky předcházející) uzly představující vstupy funkce (jejich počet je fixní pro implementaci CGP), odkaz na (matematickou) funkci a případný parametr (nebo parametry) funkce.

V genotypu se mohou vyskytovat (a prakticky vždy vyskytují) uzly, které nejsou připojené na výstupní uzly. Tyto uzly tedy neovlivňují chování jedince a nejsou součástí fenotypu. Existuje tedy uzlová redundance.

Dalším druhem redundance je redundance vstupů, která znamená, že ne všechny vstupní uzly musí být připojené ke všem výstupům, a funkcionální redundance, kdy výsledná funkce může být zapsána jiným jednodušším způsobem (r = NOT(NOT(x)) vs. r = x).

2.7.2 Neutralita

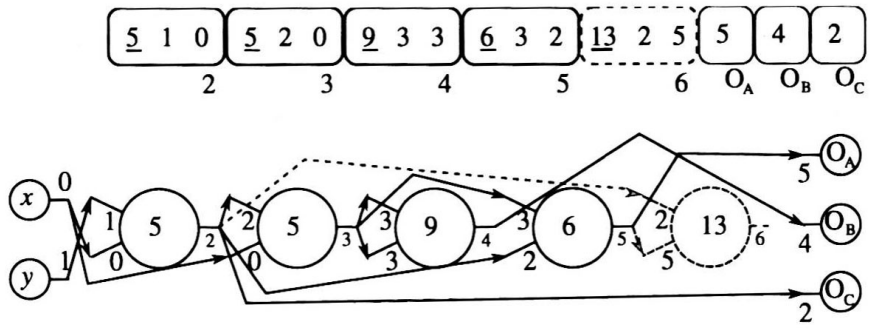
Redundance obsažená v CGP explicitně dovoluje neutralitu, tedy přítomnost různých genotypů se stejnou hodnotou fitness. Neutrální prohledávání prostoru možných řešení znamená, že jedinec může být v populaci nahrazen jiným jedincem o stejné fitness.

Tento princip dovoluje genetický drift, tedy postupnou změnu v neprojevené části genotypu za mnoho generací. Ten umožňuje posun v prostoru možných řešení a možnost uniknout z lokálního minima.

2.7.3 Reprodukce

Kartézské genetické programování používá typicky operátor mutace. Ta mění některý z výše zmíněných parametrů uzlu. Operátor křížení je typicky velmi destruktivní. U aplikace CGP na některé konkrétní problémy (např. evoluční umění) ale najde využití.

Orientovaný graf představující genotyp CGP je acyklický. Dodržení této podmínky je zaručeno omezujícími podmínkami v implementaci genetického operátoru (mutace). Díky acykličnosti je možné zaručit, že všechna nová řešení budou korektní. Tím se CGP liší od standardního genetického programování, ve kterém zdaleka ne všechny řešení ve formě programů musí být spustitelné bez chyb.[8]



Obrázek 2.3: Schéma CGP genotypu. Uzel 6 (čárkovaně) není připojený k výstupu a není tudíž součástí fenotypu [7]

2.7.4 Algoritmus

Typická evoluční strategie Kartézského genetického programování. Hodnota λ je typicky 4.

1. Vytvoř náhodnou počáteční populaci velikosti $1+\lambda$
2. Ohodnoť všechny členy populace, nejlepšího ulož jako rodiče
3. Dokud není nalezeno řešení nebo není dosažen generační limit, opakuj:
   (a) Pro λ opakování
      i. Pomocí mutace rodiče vytvoř potomka
   (b) Podle těchto pravidel vyber nejlepšího jedince v populaci
      i. Pokud má 1 potomek lepší nebo stejnou hodnotu fitness jako rodič, ulož jej jako nového rodiče
      ii. Pokud má více potomků lepší nebo stejnou hodnotu fitness jako rodič, vyber z nich jednoho náhodně a ulož jej jako nového rodiče
   (c) Jinak ponech současného rodiče

[7]

2.8 Evoluční umění

Evoluční umění využívá počítač a evoluční algoritmy k vytvoření esteticky zajímavých objektů. Může se jednat o 2D obrazce, 3D objekty nebo dokonce hudbu. Tento text se ale zaměřuje na 2D obrazy.

Prvním zásadním rozdílem mezi evolučním uměním a jinými problémy prohledávání prostoru řešení je absence jednoznačné fitness funkce. Fitness hodnotu obrazu určuje jeho estetická kvalita, vlastnost jen těžko uchopitelná. Estetická kvalita je navíc závislá na pozorovateli. Nemůžeme ani s jistotou prohlásit, že by jeden obraz měl vyšší estetickou hodnotu než druhý.

Probíhá vývoj na poli čistě algoritmicky vytvořeného evolučního umění, ale valná většina existujícího software používá interakci uživatele pro hodnocení fitness a směřování evoluce.

Obrázek 2.4: Karl Sims, 1991 [10]

Druhý rozdíl evolučního umění přímo vyplývá z prvního. Absence jasně definované fitness funkce implikuje i absenci jejího globálního minima.

Cílem evoluce tedy není nalezení optimálního bodu v prostoru řešení, ale průzkum tohoto prostoru a hledání různých zajímavých řešení. Účelem evolučního umění je tedy experimentace a explorace, ne optimalizace.

2.8.1 Metody tvorby 2D obrazů

Vykreslování založené na výrazech

V roce 1991 představil Karl Sims přístup k tvorbě obrazů založený na výrazech (expression-based).

V tomto přístupu je genotypem jedince matematický výraz, typicky reprezentovaný ve formě stromu, kde funkce představují vnitřní uzly a konstanty a proměnné pak listy stromu. Tento přístup je vlastní genetickému programování.

Proměnné pak typicky představují souřadnice pixelu. Výraz je vyhodnocen pro každý pixel zvlášť a pixelu je přiřazena výsledná barva.

Použití neuronových sítí

Několik projektů vyvinulo neuronové sítě pro vykreslování obrázků. Například Artificial Painter používá neuronové sítě, na jejichž vstupu jsou směr a vzdálenost orientačního bodu. Vývoj abstraktních obrazců může probíhat automaticky nebo interaktivně.

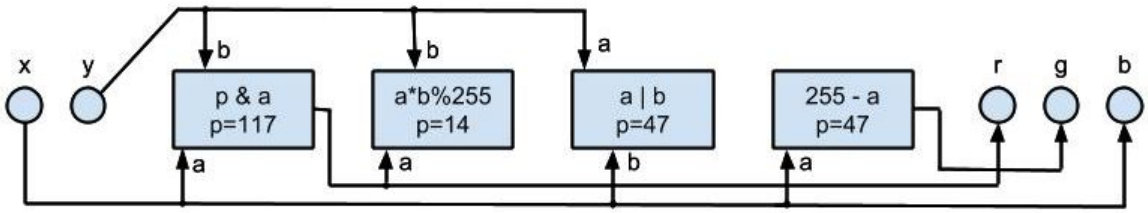
Zpracování obrazu

Existuje poměrně hodně systémů, které používají genetické techniky ke zpracování zdrojového obrazu. Několik systémů založených na vykreslování založeném na výrazech používá zpracování obrazu jako zdroj barevných palet.

Objevily se dokonce komerční produkty, které používají tuto metodu pro interaktivní úpravu obrazu pomocí filtrů.[10]

2.9 Kartézské genetické programování v evolučním umění

Přístup kartézského genetického programování odpovídá způsobu vykreslování založeném na výrazech. Genotyp CGP je sice obecný orientovaný graf, jeho vyhodnocením ale pro ka-



Obrázek 2.5: Schéma CGP genotypu

ždý výstup vzniká matematický výraz, který definuje výstup jako funkci vstupů. Rozdíl je především v tom, že každý výstup (typicky jsou potřeba 3, jeden pro každý barevný kanál) nemá oddělený genotyp, ale výstupy mohou navzájem sdílet uzly. Výsledkem jsou typicky funkčně bližší výstupní výrazy.

Ve své práci jsem přímo vycházel z článku Lawrence Ashmora a Juliana F. Millera Evolutionary Art with Cartesian Genetic Programming[4].

První částí článku je přiblížení problematiky evolučního umění. Druhá se zabývá podobnými systémy v problematice evolučního umění. Třetí část přibližuje kartézské genetické programování. Tato témata jsem už popsal výše. Samotný přístup k problému a implementaci se pokusím rozebrat podrobněji.

Při implementaci jakéhokoli evolučního algoritmu velmi záleží na způsobu reprezentace kandidátního řešení. Výběr metodiky, v tomto případě kartézského genetického programování, ji do jisté míry určuje. Stále ale zbývá prostor pro přizpůsobení konkrétnímu problému.

Orientovaný graf představující genotyp jedince v CGP má vstupní a výstupní uzly. Výsledné chování jedince pak vytváří funkce závislosti výstupu na vstupech. Vstupy a výstupy je tedy třeba definovat v závislosti na doméně problému.

Ashmore a Miller vytvářejí rastrový obrázek. Vstupem jsou x,y souřadnice konkrétního pixelu, výstupem pak barva tohoto pixelu. Testují barevnou reprezentaci RGB(červená, zelená, modrá) i HSB(odstín, sytost, jas). V obou případech jsou výstupní uzly 3.

Použité matematické funkce pracují až se dvěmi proměnnými a jedním parametrem. Funkční uzly tedy představují 4 celočíselné hodnoty: odkaz na vstup 1, odkaz na vstup 2, odkaz do tabulky funkcí a hodnota parametru. Pro zaručení acykličnosti grafu a korektnosti všech výstupních stromů jsou jako vstup povoleny pouze odkazy na uzel s nižším indexem.

Ashomore a Miller implementují operátory mutace i křížení. Při mutaci se mění hodnoty jednotlivých celočíselných hodnot v uzlech, a to tak, aby byly stále zachovány výše zmíněné omezující podmínky. Rychlost mutace udává počet hodnot, které se změní.

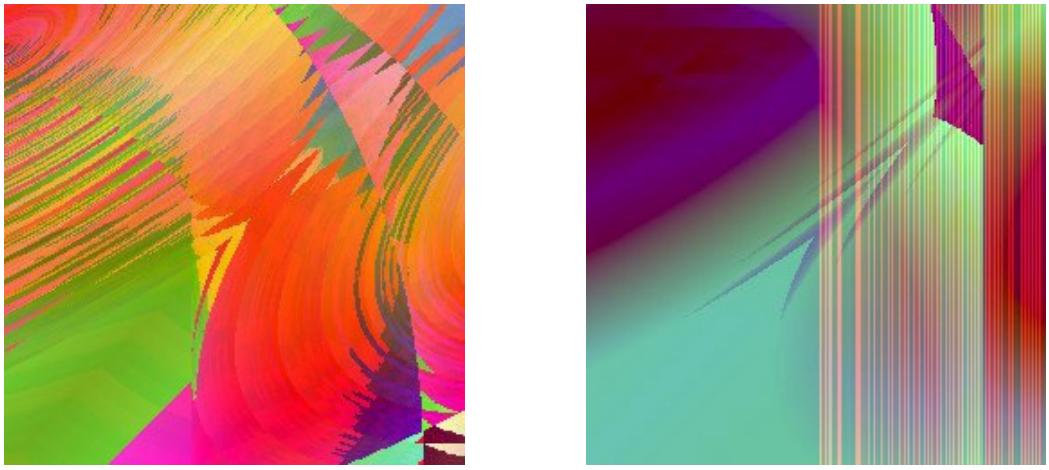
Bod pro křížení genotypu se vybírá vždy mezi uzly. Výsledkem křížení jsou často obrázky, které mají tvar jednoho rodiče a barevnou skladbu druhého.

Směr prohledávání a fitness jedinců je zcela na uživateli. Má proto možnost ovládat rychlost mutace a případnou přítomnost křížení. Může také zamknout jednotlivé barevné kanály. V uzlech, které jsou připojeny k vybranému výstupu, nedojde ke změně.

V článku jsou popisovány i pokusy s autonomní evolucí za použití předdefinované fitness funkce. Experimentuje se s evolucí k maximální diverzitě v obrázku nebo například k přítomnosti kulatých objektů.

Výsledky autonomní evoluce nevytváří obrázky, které by byly přímo esteticky skvělé, ale mohou vytvořit zajímavý výchozí bod pro vývoj řízený uživatelem.

Aplikace také nevytváří nultou generaci náhodně, ale rekombinuje genotypy, které byly



Obrázek 2.6: Ashmore, Miller: výsledné obrázky [4]

dříve označené jako esteticky kvalitní. To zkracuje čas průchodu relativně nezajímavou částí prostoru řešení.[4]

Kapitola 3

Návrh metody

Zadáním mé bakalářské práce je navrhnout metodu, jak použít kartézské genetické programování v evolučním umění, a tuto metodu implementovat a otestovat. Samotný algoritmus CGP jsem tedy implementoval v relativně jednoduché formě a zaměřil se na experimentaci se způsoby grafické reprezentace vyvinutých stromů funkcí.

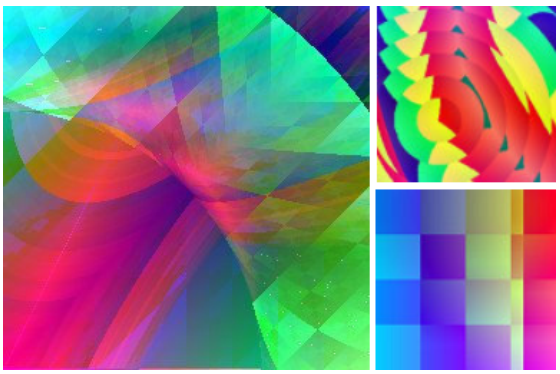
Samotný proces návrhu metody probíhal iterativně. Průběžně jsem experimentoval s různými možnostmi, a spolu s rostoucí znalostí problematiky se měnily i mé představy o výsledné metodě.

V této části se pokusím přiblížit nejen navrženou metodu, ale i myšlenkový proces, který k jejímu zrodu vedl.

3.1 Motivace

Směr, kterým se práce ubírala, je aplikace evolučního umění v užitém umění. Měl jsem představu aplikace schopné evolučním přístupem vytvořit například logo firmy. Pro člověka, který nemá žádné předešlé znalosti v designu, by pak přestavovala alternativu ke konvenčním editorům loga.

Pro účel užitého umění je potřeba generovat graficky jednoduchý, konzistentní obraz. Evolučně generovaná rastrová grafika toto kritérium nesplňuje dle mých představ. Proto jsem se rozhodl experimentovat s vektorovou grafikou.



(a) Ashmore a Miller: Rastrové CGP [4]

(b) Výsledek finální aplikace

Obrázek 3.1: Srovnání rastrového a vektorového obrázku

V návaznosti na tento požadavek jsem definoval některé rysy, které by měla cílová aplikace splňovat. A přestože je její účel nakonec poněkud jiný, tyto rysy si zachovala.

Cílovým uživatelem cílové aplikace není nadšenec pro evoluční umění, ale člověk, který s ním má minimální zkušenosti. I důraz mé bakalářské práce je spíše na aplikaci evolučních metod než experimentaci s nimi.

Požadavky na cílovou aplikaci jsou tedy následující:

1. Jednoduché ovládání: Způsob chování jádra evolučního je relativně málo nastavitelný. Veškeré uživatelské rozhraní je maximálně intuitivní.
2. Rychlý přechod mezi generacemi: Důraz programu je na co nejrychlejší a nejzábavnější tvorbu výsledných obrázků. Doba výpočtu mezi generacemi musí být minimální, průchod prostorem řešení pro uživatele co nejplynulejší.
3. Okamžitá použitelnost: Uživatel nepotřebuje studovat návody ani rozumět procesu evolučních algoritmů. Aplikaci nemusí stahovat a instalovat.

3.2 Napodobení už existující implementace

Dříve než jsem se pustil do skutečného návrhu vlastní vykreslovací metody, jsem se potřeboval skutečně seznámit s CGP a vytvořit funkční jádro evolučního algoritmu. K tomuto účelu jsem se pokusil zreprodukovat metodu, kterou použili Ashmore a Miller.

Vzhledem k výše zmíněným požadavkům na výslednou aplikaci jsem ale pro svou implementaci jejich metody zjednodušil. Samotné jádro CGP jsem pak použil i v ostatních iteracích své aplikace.

Pro reprodukci jsem implementoval pouze mutaci, jak je pro obecné CGP běžné. Uživatel z každé generace také vybírá pouze jednoho jedince, který se stane rodičem pro další generaci. To odpovídá požadavkům na jednoduché ovládání a rychlý přechod mezi generacemi. Evoluční krok ze strany uživatele představuje pouze kliknout na vybraný obrázek.

Oproti typickému CGP (jak ho popisují Miller a Thomson) jsem se rozhodl nenechávat vybraného rodiče v populaci. Jakkoli je vhodné zachovat nejlepší dosavadní řešení v optimalizačních problémech, důraz mé aplikace je na exploraci. Často se vyskytnou potomci stejného fenotypu, ale tímto způsobem je garantován posun v prostoru řešení při každé generaci.

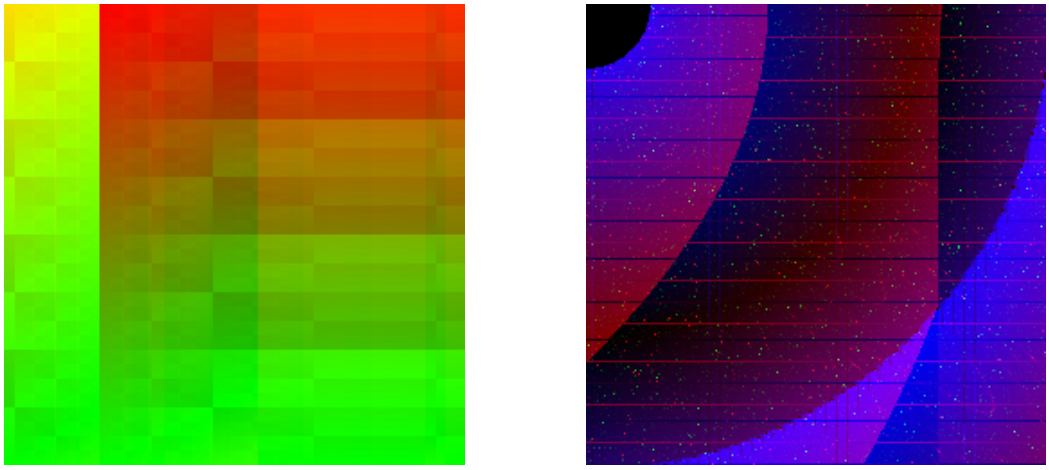
Ve výsledné aplikaci jsem tento rys do jisté míry omezil přidáním možnosti vrátit se o jeden evoluční krok zpět. Jde částečně o krok proti filozofii aplikace, ale testující uživatelé ho přímo požadovali.

3.3 Navržené metody

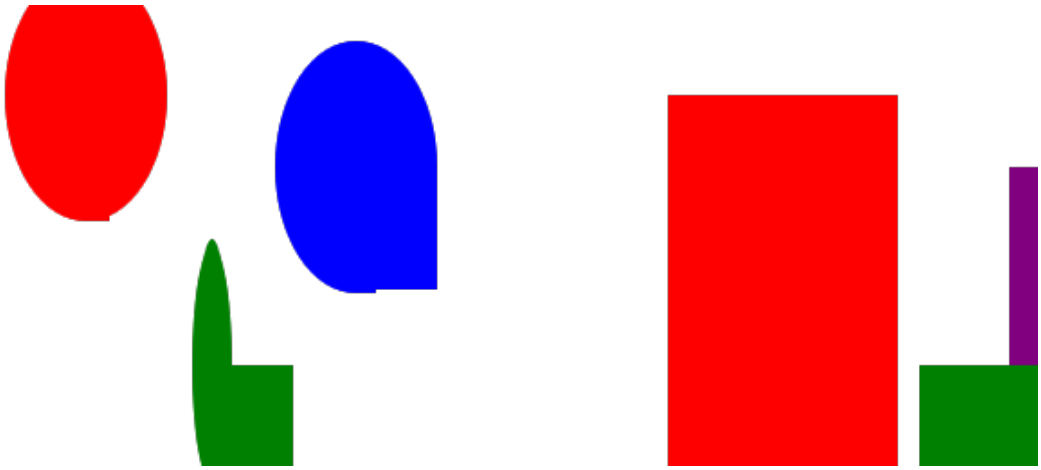
Poté, co jsem implementoval jádro CGP algoritmu, jsem přistoupil k samotnému návrhu alternativní metody aplikace CGP.

Již jsem zmiňoval zájem o užité umění a vektorovou grafiku. Ale v okamžiku, kdy jsem CGP skutečně implementoval, jsem si uvědomil praktické implikace některých jeho vlastností.

Výstupem CGP jsou funkce závislosti na vstupech. Právě vstupy a rozmanitost hodnot, kterých nabývají, se pro mě stala hlavním omezujícím faktorem. Pokud je totiž počet kombinací vstupů příliš nízký, výsledný obrázek přestává být vizuálně podobný svému rodiči a uživatel ztrácí kontrolu nad evolucí.



Obrázek 3.2: Výsledné obrázky mojí implementace rastrového CGP



Obrázek 3.3: Obrázky vygenerované metodou superobjektů

Konkrétní počet kombinací hodnot, kterých musí vstupy nabývat, aby byla aplikace ovladatelná, je závislý na způsobu vykreslování. Pokud se ale tento počet pohybuje řádově v desítkách, je nová generace z vizuálního hlediska prakticky náhodná.

Jednoduché kompaktní vektorové obrázky, o kterých jsem uvažoval na začátku vývoje, se skládaly z jednotek, maximálně desítek vektorových objektů.

Pokusil jsem se navrhnout několik možných směrů, kterými se mohl vývoj dále ubírat:

3.3.1 Superobjekty

Tato metoda je nejblíže mojí původní představě několika málo vektorových objektů, které dohromady tvoří abstraktní obrazec.

Každý z těchto několika objektů je superobjektem, celkem složeným z více vektorových objektů. Ty mají společnou barvu a přibližné umístění. Vytváří tak vizuální dojem jednoho celku s komplikovaným tvarem.

Tato metoda kombinuje CGP a standardní genetický algoritmus. Každý superobjekt představuje chromozom v genetickém algoritmu. Jeho výstupem je barva, střed a velikost superobjektu.

Superobjekt definuje oblast, ve které se vykreslí standardní vektorové objekty. Vytváří hrubý obrys, který je pak přesněji definován jednotlivými vykreslovanými tvary.

Vektorové objekty jsou vytvářeny funkcemi, které jsou produktem CGP. Jediným vstupem funkce je index vykreslovaného objektu. Výstupy jsou relativní posun od středu superobjektu, velikost a tvar.

Aby index objektu byl smysluplným vstupem CGP, musí nabývat alespoň desítek, lépe stovek možných hodnot. V tom okamžiku už vykreslované objekty vyplní plochu superobjektu prakticky beze zbytku a výsledkem je jednolitý barevný obdélník nebo elipsa, v závislosti na definici omezení obrysu.

Metodu superobjektů jsem tedy po krátké experimentaci opustil jako neperspektivní.

3.3.2 Vektorové objekty s rastrovou CGP texturou

Druhá metoda, kterou jsem zvažoval, je opět kombinací kartézského genetického programování se standardním GA.

Výsledný obraz se skládá z několika vektorových objektů, jejichž vlastnosti (umístění, velikost, tvar) nese chromozom v genetickém algoritmu. Namísto jednolité barvy jsou ale vyplněny rastrovou texturou vygenerovanou pomocí CGP.

Metoda kombinuje kartézské genetické programování s vektorovým přístupem, což je něco, co bylo mým cílem. Navíc je zcela určitě realizovatelná.

Další výhodou této metody je jednoduchost implementace. Generování rastrového obrázku pomocí CGP jsem už implementoval a implementování jednoduché formy GA je poměrně přímočaré.

Mým cílem ale bylo vytvořit kompaktnější obrázek než při použití CGP pro tvorbu rastrového obrázku. Tato metoda vytváří koláž rastrových obrázků, která je naopak z principu méně vizuálně celistvá než původní metoda. Z tohoto důvodu jsem se rozhodl metodu neimplementovat.

3.3.3 Přímá generace vektorů pomocí CGP

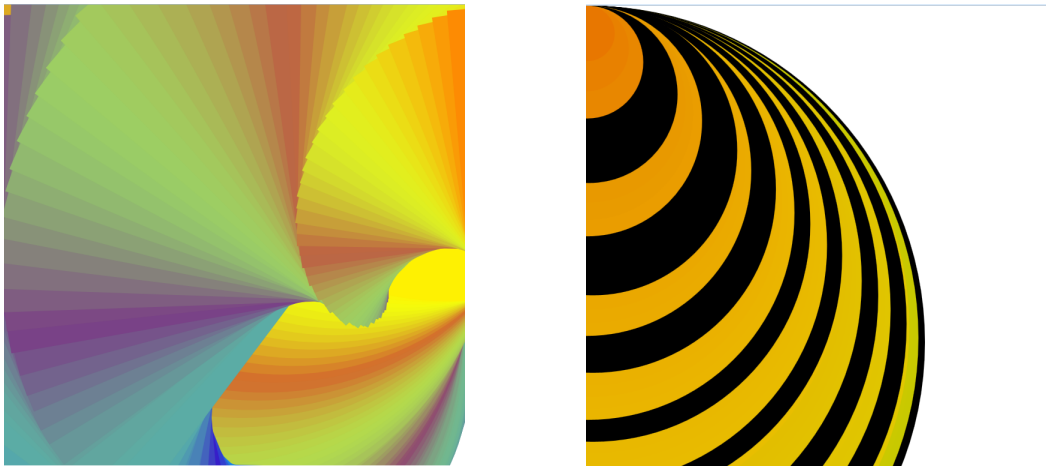
Při použití této metody jsou všechny vlastnosti (umístění, velikost, tvar, barva) vektorových objektů výsledkem funkcí generovaných pomocí CGP. Vstupem těchto funkcí je index objektu.

Rozdílem oproti metodě superobjektů je počet generovaných objektů, který je v řádech stovek až tisíc. Toto množství je už dostatečné pro to, aby generované obrázky nové generace byly příbuzné svému rodiči i po vizuální stránce a uživatel měl kontrolu nad směrem evoluce.

Dalším zásadním rozdílem je to, že barva každého objektu je opět generována z jeho indexu. I když se tedy objekty překreslují přes sebe a zakrývají celé vykreslovací pole, neslijí se do jednoho celku. Pouze se překreslují přes sebe a výsledný obrázek představuje pohled na nejvyšší (nejpozději generované) objekty, nezávisle na množství objektů v nižších vrstvách.

Při použití této metody vznikají nejvíce kompaktní obrázky ze všech navržených metod. Ani v tomto případě ale nejsou dostatečně jednoduché pro přímé využití v užitém umění.

Obrázky vzniklé použitím této metody jsou ale (dle mého názoru) esteticky zajímavé a často velmi vizuálně diametrálně odlišné od rastrových.



Obrázek 3.4: Obrázky vygenerované přímou genenerací vekrorů

Obrázek 3.5: Obrázky vygenerované vybranou metodou: přímou genenerací vekrorů

3.4 Výběr metody

Po krátké experimentaci s navrženými metodami jsem dospěl k závěru, že vytvoření obrazu natolik jednoduchého, aby odpovídal mým prvotním očekáváním, není za použití vykreslování pomocí výrazů možné, nebo alespoň není v mých silách.

Při použití metody, ve které je generovaný obraz vyjádřen pomocí matematických výrazů (a použití CGP, které je přímo zadáním mé práce, je taková metoda), je nutnou podmínkou rozmanitost vstupů. Při práci s vektory je jedinou proměnnou, která se přirozeně nabízí, index objektu. Tento fakt mě limitoval v celém průběhu mé práce.

Metoda přímé generace vektorů pomocí CGP se mi jevila z navržených metod nejslibnější. Jakkoli se generované obrázky odchylují od prvotních představ, jsou (dle mého názoru) vizuálně zajímavé. Metoda tak představuje alternativu k rastrovému CGP popisovanému Ashmorem a Millerem.

Výsledné obrázky nejsou použitelné v užitém umění, ale vzory a motivy, které se v nich objevují, mohou alespoň představovat inspiraci pro člověka, který chce vytvořit logo nebo jiný designový návrh.

Kapitola 4

Implementace

4.1 Výběr platformy

Požadavek na okamžitou použitelnost aplikace nejlépe splňuje webová aplikace. Cílovým uživatelem je člověk, který nemá přílišné povědomí o evolučním umění. Pokud mu stačí kliknout na odkaz a okamžitě může tvořit obrázky, je šance, že bude chvilku experimentovat a třeba ho problematika zaujme. Stahování a instalace těžké aplikace představují bariéru, přes kterou by se mnoho potenciálních uživatelů nikdy nedostalo.

Další rozhodnutí je mezi výpočtem na straně serveru a klienta. Hlavním kritériem je zde čekací čas mezi generacemi. Pokud by přesáhl stovky milisekund, je aplikace prakticky nepoužitelná.

Při výpočtu na straně serveru je čekací čas delší o komunikaci mezi serverem a klientem. Výpočetní výkon serveru bude pravděpodobně řádově vyšší než u klienta, ale záleží na jeho zatížení a počtu uživatelů.

Rozhodl jsem se provádět zpracování algoritmu na straně klienta. Jednak kvůli tomu, že nemám jiné serverové zázemí než školní servery, jednak kvůli větší znalosti technologie na straně klienta.

Na klientské straně webové aplikace není příliš široký výběr možné technologie. Jediným nativním programovacím jazykem je JavaScript (dále JS).

Pro jádro CGP, samotný evoluční algoritmus a tvorbu stromu funkcí, jsem se ale rozhodl použít programovací jazyk Dart. Dart je silně typovaný třídně objektově orientovaný jazyk, který je možné kompilovat do JS.[1]

Jádro CGP jsem chtěl co nejvíce robustní. Výpočetní čas nutný na provedení CGP algoritmu je relativně malý, alespoň v porovnání s časem vykreslení. Naopak algoritmus je výrazně komplexnější a pracuje se složitějšími datovými strukturami. Silná typová kontrola a třídní objektový přístup dávají daleko menší prostor pro chybu než přístup JavaScriptu, který nemá prakticky žádnou typovou kontrolu a pracuje s prototypy.

Druhý a možná stejně důležitý důvod pro použití jazyka Dart byl můj osobní zájem a chuť tento jazyk vyzkoušet.

Na vykreslování obrazu a funkce uživatelského rozhraní jsem použil JavaScript s knihovnou jQuery.

4.2 Technologie

4.2.1 Dart

Dart je open-source programovací jazyk určený pro web. Za jeho vývojem stojí společnost Google. Jedná se o velmi mladý programovací jazyk. Veřejnosti byl jeho vývoj oznámen v roce 2011, verze 1.0 byla vytvořena na konci roku 2013.

Dart je silně typovaný, třídně objektově orientovaný programovací jazyk. Strukturou vytvořených programů i syntaxí se podobá Javě. Některé své prvky ale přebírá z funkcionálních programovacích jazyků.

Dart je stejně jako Javu možné spustit na virtuálním stroji. Webový prohlížeč Dartium, varianta Google Chrome, jej umožňuje spouštět tímto způsobem na straně klienta. Stejně tak je možné ho spustit i na straně serveru.

Pro většinu aplikací na straně klienta je ale vhodné Dart kompilovat do JavaScriptu. V tom okamžiku je výsledná webová aplikace spustitelná libovolným webovým prohlížečem. Zkompilovaný kód ve většině případů není výpočetně náročnější než přímo ručně vytvořený JS skript. Vzhledem k heuristikám, které kompilátor nasazuje, je naopak často rychlejší.

Hlavní výhodou Dartu je robustnost zdrojového kódu. Silná typová kontrola je prvkem, který umožňuje najít chybu v kódu potenciálně rychleji. Kód se při spuštění na virtuálním stroji překládá do pseudokódu. Na rozdíl od čistě interpretovaného JavaScriptu tedy kompilátor kontroluje i ty větve programu, kterými se při konkrétním spuštění neprojde.

Moje osobní zkušenost s Dartem je velmi kladná. Se složitými datovými strukturami se mi pracovalo výrazně lépe než v JS.[1]

4.2.2 jQuery

jQuery je multiplatformní knihovna pro JavaScript. V současnosti je nejrozšířenější JS knihovna na světě.

jQuery výrazně zjednodušuje práci s JS. Umožňuje rychlou navigaci v DOM HTML a manipulaci s ním. Zpřístupňuje jednoduché rozhraní pro AJAX. Zjednodušuje přístup k CSS a implementuje animace.

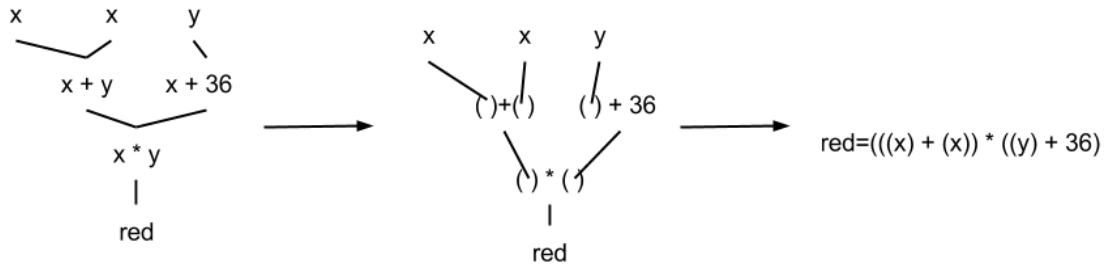
Dále obaluje přístup k atributům objektů. Na jednu stranu k nim zrychluje přístup, na druhou výrazně snižuje možnost chyby, například omylem zapsat do atributu, který má být pouze čten.

Při použití jQuery také klesá nutnost psát stejnou operaci několikrát pro různé webové prohlížeče. jQuery sjednocuje rozhraní manipulace s CSS a DOM pro všechny prohlížeče (včetně IE6).[2][3]

4.3 Napodobení existující implementace

Předtím, než jsem začal s návrhem mých vlastních metod, jsem se pokusil napodobit implementaci, kterou použili Ashmore a Miller, jak jsem již výše popsal. Samotná implementace evolučního jádra CGP ale zůstala zachována i ve výsledné aplikaci. Pokusím se ji tedy přiblížit detailněji.

Výpočetní náročnost algoritmu CGP oproti vykreslení je relativně malá (zpočátku odhadováno, později ověřeno). Volba datového modelu CGP grafu tedy nezáležela až tak na výkonu jako spíše na použité technologii a co nejjednodušší mutaci a resoluci.



Obrázek 4.1: Schéma resoluce stromu

Ashmore a Miller pracují s genotypem ve formě celočíselného pole. Já se ale vzhledem k použité technologii a důrazu na robustnost rozhodl použít objektovou strukturu. Genotyp je reprezentován objektem, který zapouzdřuje operace jako mutace, iniciace a resoluce (vytvoření výsledné funkce pro každý výstup) nad genotypem.

Vstupní, výstupní a funkční uzly jsou pak objekty, na které má genotyp uložené odkazy. Tyto odkazy jsou seřazené do pole. Každý typ uzlu má vlastní třídu, sdílejí ale rozhraní. Iniciace a mutace jsou implementovány jako metody uzlů poměrně přímočaře, objekt genotypu pak vybírá cílové uzly a volá jejích metody.

Objekt vstupu má pouze jeden řetězcový parametr, který se po resoluci stane jménem proměnné při vyhodnocování výsledné matematické funkce. Vstup není možné mutovat, iniciace není potřeba.

Objekt výstupu má obdobně řetězový parametr, který se po resoluci stane jménem výsledné funkce. Dále obsahuje jeden odkaz na předchozí uzel. Ten získá náhodnou hodnotu při iniciaci a při mutaci se opět nahradí novou korektní náhodnou hodnotou.

Funkční uzly obsahují dva odkazy na dřívější uzly, odkaz do pole matematických funkcí a hodnotu parametru, která nabývá hodnot 0 až 255. Při iniciaci se jim přiřadí náhodné hodnoty. Při mutaci se náhodně vybírá, který parametr se změní, a tomu se přiřadí nová náhodná hodnota.

Orientovaný graf, který tato objektová struktura vytváří, je v každém okamžiku po iniciaci v korektní podobě a je možná jeho resoluce.

Resoluce samotná představuje složitější problém. Ta se provádí zpětným průchodem grafu od výstupů, kdy se pro každý z nich tvoří stromová struktura. Tuto datovou strukturu je ale nutné převést do formy, kterou bude možné zpracovat co nejrychleji. Zpracování matematických funkcí, které jsou interpretací stromů a vlastním výsledkem CGP, totiž i ve formě nativního kódu spotřebuje nejvíce výpočetního času z celého evolučního kroku.

Východiskem pro mě byla nativní funkce JavaScriptu eval(). Ta zpracuje řetězec, interpretuje jej jako zdrojový kód JS a vykoná jej. Při průchodu binárním stromem mi tedy stačí sestavit řetězec s JS syntaxí. Dart pak umožňuje propojení s JS a není problém volat tuto JS funkci z Dart kódu.

Resoluci implementuji rekurzivně, průchodem od konkrétního výstupu. Převod matematických funkcí CGP na řetězec implementuji funkcionálně, tj. pro každou matematickou funkci existuje programová funkce, která vloží do řetězce hodnoty vstupů (řetězce vzniklé resolucí podstromu) a hodnotu parametru.

Tímto postupem získám řetězec v syntaxi JavaScriptu, který představuje přiřazení nového těla pro funkci. Tento řetězec předám jako parametr JS funkci eval(). Od toho okamžiku stačí nově vytvořenou funkci zavolat jako standardní funkci JS a bude vykonána

přímo v nativním kódu, tedy výrazně rychleji, než kdybych implementoval jakýkoli interpret pseudokódu.

Vykreslení samotného obrázku by bylo možné implementovat v rámci Dart skriptu, ale rozhodl jsem se ho oddělit a implementovat v JS. Zejména při použití knihovny jQuery se mi jevila práce s objekty HTML DOM (objekty skládající webovou stránku) jednodušší přímo z kódu v JavaScriptu.

Když jsou výsledné funkce CGP uloženy jako JS funkce, iteruji přes rozlišení obrázku (256x256 pixelů) a každému pixelu přiřadím hodnoty barevných kanálů. Ty získám tak, že jednoduše zavolám nově vytvořenou funkci s aktuálními souřadnicemi pixelu jako argumenty. Samotné vykreslení provádím do objektu HTML canvas pomocí atributu Image.data, která mi dovoluje přímo zkonstruovat rastrový obraz a je k tomuto účelu optimalizována.

4.4 Vykreslování generovaných vektorových obrázků

První implementace metody přímého generování vektorových obrázků byla jednoduchý experiment postavený na testovací aplikaci rastrového CGP, kterou jsem už měl hotovou. I finální verze aplikace vznikla z tohoto jádra.

Zachoval jsem výše popsané jádro CGP včetně toho, že každý uzel genotypu má potenciálně dva vstupy. Převzal jsem i sadu matematických funkcí, jak je původně definovaná v článku Ashmora a Millera.

Jak jsem již zmínil, samotný algoritmus CGP jsem převzal tak, jak je popsaný v minulé kapitole. Změnil jsem pouze množinu výstupů.

Zároveň jsem musel z jediné hodnoty indexu objektu vygenerovat dvě hodnoty pro vstup CGP. V první experimentální implementaci jsem použil tutéž hodnotu dvakrát. Později jsem s tímto problémem poměrně experimentoval. Více je popsaný v podkapitole Evoluční jádro.

Pro samotné vykreslování vektorového obrázku jsem zvolil formát SVG. Je to formát vektorové grafiky, který moderní webové prohlížeče podporují a přímo zobrazují. Zároveň je ho schopné zpracovat široké spektrum software pro editaci počítačové grafiky.

Rozhodl jsem se vykreslovat obrazce pomocí dvou tvarů (objektů SVG): obdélníku a elipsy. Pomocí různě se překrývajících obdélníků a elips je možné generovat velmi široké množství obrázků. Tyto dva tvary také sdílejí argumenty, kterými jsou definované – x a y souřadnice počátku, výška a šířka. Elipsa je v SVG definována prakticky obdélníkem, kterému je vepsána.

Jinými potenciálně použitelnými tvary jsou kruh a čtverec, které jsou ale pouze speciálním případem elipsy a obdélníku. Dále SVG podporuje obecné polygony. Ty jsou ale definované pomocí množiny bodů, což je dost volný způsob definice, který je diametrálně odlišný od obdélníku a elipsy.

Výstupy CGP, které se vyhodnocují pro každý objekt, tedy jsou následující:

1. x souřadnice počátku (levého horního rohu) objektu
2. y souřadnice počátku objektu
3. výška objektu
4. šířka objektu
5. barevný kanál pro červenou

6. barevný kanál pro zelenou
7. barevný kanál pro modrou
8. tvar objektu
9. úhel rotace objektu

Většina hodnot nabývá potenciálně celočíselných hodnot 0-255. To je běžný rozptyl hodnot pro barevné kanály a zároveň odpovídá rozlišení vykreslovaného obrazu (256x256 pixelů). Výstupní hodnoty se převádějí pro tvar objektu (logická hodnota obdélník/elipsa) a úhel rotace (0-360°).

Vykreslení každého obrázku se pak provádí iterativně. Indexem, nad kterým iteruji, a zároveň jediným vstupem CGP, je číslo objektu. Množství vykreslovaných objektů volí uživatel, kvůli tvaru matematických funkcí je ale vždy mocninou dvojky.

V okamžiku, kdy už mám hodnoty všech parametrů pro objekt, je jeho vykreslení otázkou jednoduchého vytvoření SVG tvarového objektu a jeho vložení do objektu SVG, který obrázek zapouzdřuje.

4.5 Evoluční jádro

Konkrétní problémy, které se týkají spíše problematiky evolučního programování než obecné implementace jsem oddělil do této podkapitoly.

4.5.1 Matematické funkce CGP

Když jsem vytvářel svou implementaci rastrového CGP, převzal jsem sadu matematických funkcí pro CGP z článku Ashmora a Millera. Stejně tak jsem ji převzal i při prvních pokusech s vektorovou grafikou.

Později jsem se pokoušel navrhnout vlastní sadu matematických funkcí, ale bez valného úspěchu. Těžiště své práce jsem viděl v jiných oblastech a i vektorová grafika při použití této sady generovala esteticky zajímavé obrázky. Příliš jsem tedy s tímto problémem neexperimentoval.

Sada matematických funkcí, kterou používá Ashmore a Miller, a kterou jsem převzal, je následující:

1. x | y
2. x & p
3. x / (1.0 + y + p)
4. (x * y) % 256
5. (x + y) % 256
6. if (x > y): x - y; else: y - x
7. 255 - x
8. abs(cos(x) * 256)

9. abs(tan(((x % 45) * π) / 180.0) * 256)
10. abs(tan(x) * 256) % 256
11. sqrt((x - p)^2 + (y - p)^2)
12. (x % (p + 1)) + 256 - p
13. (x + y) / 2
14. if (x > y): 256 * (x+1) / (y+1); else: 256 * (y+1) / (x+1)
15. abs(sqrt(x^2 - p^2 + y^2 - p^2) % 256)

x odpovídá vstupu 1 uzlu, y vstupu 2 a p parametru

4.5.2 Generování hodnot vstupů

V okamžiku, kdy jsem převzal sadu matematických funkcí, se vyskytl další problém. CGP ve formě, v jaké jsem jej implementoval, očekává dva vstupy. Jediným vstupem, který jsem měl k dispozici, byl ale index generovaného vektorového objektu.

Tento index zároveň nenabývá vždy stejné množiny hodnot. Množství generovaných objektů může uživatel vybrat. Kvůli tomu, že je sada funkcí přizpůsobená pro hodnoty 0-255, snažil jsem se hodnotami vstupů co nejlépe pokrýt tento interval.

Prvním krokem bylo určení množství generovaných objektů. Empiricky jsem určil dolní hranici 64 a horní na 2048. Při množství objektů nižším než 64 se ztrácí vizuální podobnost geneticky příbuzných obrázků. Při generování většího počtu objektů než 2048 je výpočetní doba pro evoluční krok nepříjemně dlouhá. Větší množství také nepřinášejí větší vizuální rozdíl výsledného obrázku.

Kromě těchto krajních hodnot je počet objektů pevně nastaven na mocniny 2. Díky tomu je interval 0-255, který CGP očekává, rovnoměrně vyplněn.

Prvním vstupem CGP je většinou přímo index objektu, resp. zbytek po celočíselném dělení hodnotou 256. Pouze v případě, že je množství objektů nižší než 256, se index násobí tak, aby pokryl celý interval. Pro 64 vykreslovaných objektů tedy násobím index číslem 4, pro 128 číslem 2.

Generování druhého vstupu je složitější problém. Jeho hodnota je funkcí indexu, případně hodnoty druhého vstupu. Jiný použitelný argument není k dispozici. Pokud je příliš blízký hodnotě prvního vstupu, jsou generované objekty příliš koncentrované kolem diagonály z levého horního rohu. Pokud je naopak příliš náhodný, začnou být generované obrázky vizuálně roztříštěné.

Experimentoval jsem s poměrně širokým množstvím funkcí, ale nejlepších výsledků jsem dosáhl s velmi jednoduchou implementací. Na začátku hodnoty obou vstupů (dále x,y) iniciuji na hodnotu 0. Hodnotu inkrementu pro x nastavím na 1 (resp. 2,4 pro 64 a 128 vykreslovaných objektů). Hodnotu inkrementu pro y pak nastavím na násobek inkrementu pro x. Ve finální aplikaci je hodnota násobitele pevně určena na 3.

Hodnoty x pokryjí interval 0-255 alespoň jednou. Hodnoty y projdou tímto intervalem alespoň třikrát. Vzhledem k tomu, že je číslo 3 nesoudělné s dvojkou, při každém dalším průchodu nabývají jiných hodnot.

Experimentoval jsem i s jinými hodnotami násobitele, ale pro nižší hodnoty je příliš silně těžiště obrázku kolem diagonály a pro vyšší naopak příliš roztříštěný.

4.6 Jiné části aplikace, optimalizace

Ve všech implementovaných verzích jsem omezil počet jedinců v populaci na 9. V tom případě se na většinu monitorů vejde celá populace v nativním rozlišení 256x256 pixelů. Při jejich zmenšování by docházelo k výraznému zkreslení.

Ve finální aplikaci jsem ale pro vektorové obrázky přidal možnost zvětšení, a to na pevnou hodnotu 768x768 pixelů. Díky tomu, že jsou délky přesně trojnásobné, nedochází ke zkreslení. Pokud SVG objektu přiřadím hodnotu atributu viewbox na nativní rozlišení 256x256, provede se transformace pouhou změnou výšky a šířky SVG objektu.

Další důležitou funkcí je uložení obrázku. Současné verze Google Chrome (42.0) a Mozilla Firefox (37.0) neimplementují stažení SVG obrázku přímo v prohlížeči. Stažení jsem tedy implementoval sám pomocí převodu SVG objektu na řetězec a jeho zakódování do base64. Tento převod ale provádím až na vyžádání, tedy po stisku tlačítka Stáhnout. Převod sice trvá řádově desítky milisekund, ale když by se měl provádět při každém evolučním kroku pro všech 9 jedinců v populaci, výrazně se prodlouží odezva systému.

Další operace, která významně prodlužovala výpočetní čas nutný pro evoluční krok, bylo přidání objektu SVG tvaru do DOM HTML stránky. Abych tento problém vyřešil alespoň částečně, nepřidávám objekty po jednom. Místo toho celé SVG sestavím do řetězce s HTML syntaxí a pomocí funkce jQuery append() jej vyhodnotím a vložím na správné místo DOM. Proces vkládání do DOM se tak provádí jednou pro každého jedince populace, tedy 9x za evoluční krok.

Ještě rychlejší reakce by se pravděpodobně dalo dosáhnout vkládáním všech SVG objektů naráz. Kvůli struktuře webové stránky a především tlačítkům Stáhnout a Zvětšit, které se zobrazují pro každé SVG, to ale není dost dobře možné.

Na průměrném stroji (průměrném mezi testovacími uživateli) a standardním nastavení (512 objektů v SVG, 25 funkčních uzlů v CGP) pak trvá evoluční krok přibližně 200 až 350 milisekund. To je zpoždění, které většině tázaných uživatelů nebrání v plynulé práci s aplikací.

4.7 Uživatelské rozhraní

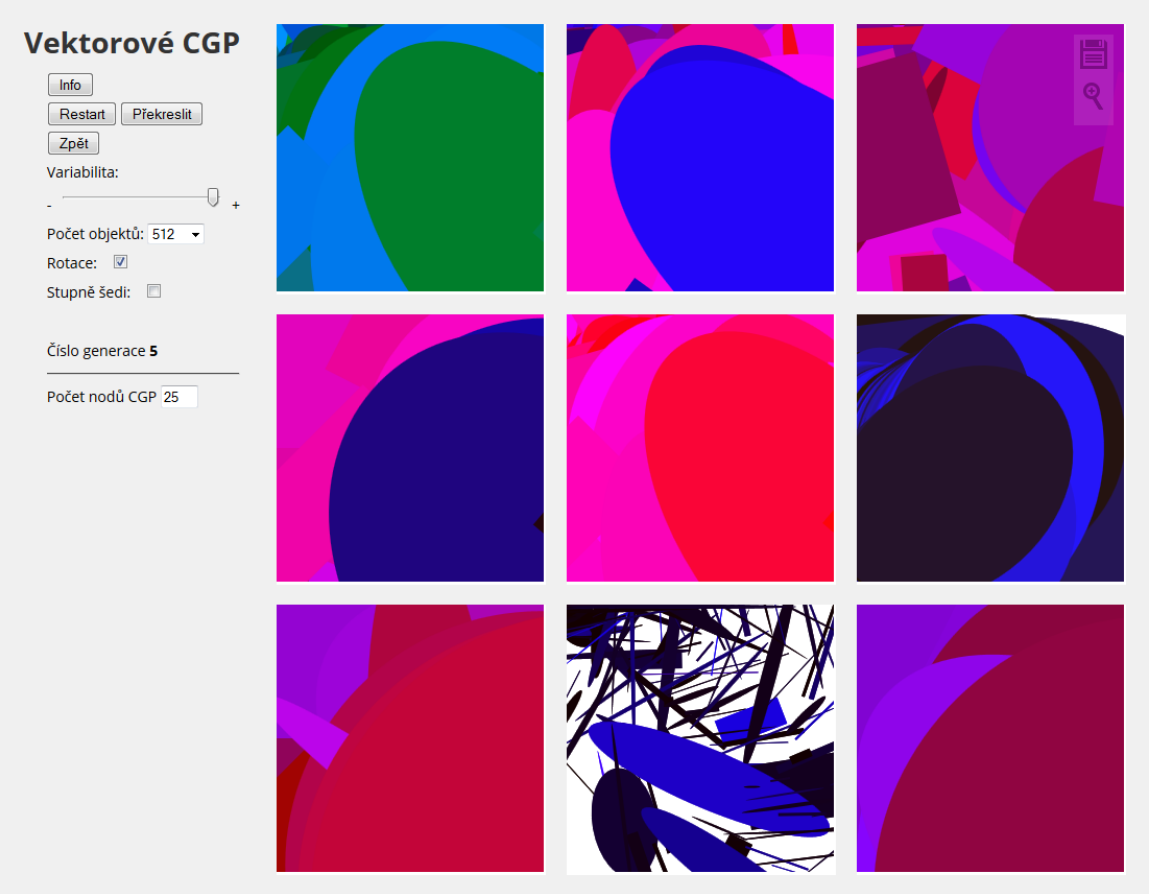
Uživatelské rozhraní pro software pro tvorbu evolučního umění se typicky drží jednomu vzoru. Většinu obrazovky zabírají obrázky, které odpovídají jednotlivým jedincům v populaci. Na straně je pak panel s možnostmi nastavení. I ve své implementaci jsem použil tento návrh.

V aplikaci se snažím o co nejjednodušší ovládání a co nejrychlejší přechod mezi generacemi. I uživatelské rozhraní se podřizuje tomuto požadavku.

Většinu obrazovky zabírají vykreslené obrázky. Uživatel vybírá vždy jen jeden nejlepší obrázek z generace. Výběr rodiče jsem tedy jednoduše nastavil na kliknutí na konkrétní obrázek. Není potřeba žádné potvrzení kroku. Pro případ, že se uživatel překlikl, je přidané tlačítko ZPĚT na bočním panelu.

Při volbě množství obrázků, které se vykreslují v každé generaci, je potřeba zvážit několik aspektů. Více jedinců znamená větší šanci, že uživatel najde obrázek, který se mu líbí. Zároveň je třeba zachovat rozlišení obrázku, při kterém je ještě možné ho dobře posoudit bez přiblížení. Omezuje nás rozlišení obrazovky.

Funkční sada CGP, kterou používám, je optimalizovaná pro rozlišení 256x256 pixelů. Zmenšení SVG obrázku je implementačně jednoduché, ale dochází při něm k optickému zkreslení. Rozhodl jsem se tedy obrázky zobrazovat v nativním rozlišení. Na monitor PC se



Obrázek 4.2: Uživatelské rozhraní aplikace

(a) Skrytá (b) Částečně viditelná (c) Viditelná

Obrázek 4.3: Tlačítka pro stažení a zvětšení

pak pohodlně vejde 9 obrázků. Spolu s bočním panelem zabírají zhruba rozlišení 1000x800 pixelů. Pro větší rozlišení (1250x800, 1250x1050) se už aplikace nemusí vejít na monitor některých uživatelů. Pokud se to stane a uživatel musí obraz posunovat, neúměrně se zpomaluje práce s aplikací.

Panel nastavení jsem umístil standardně na stranu. Pro uživatele zpřístupňuje tyto ovládací prvky:

1. Info: zobrazí nápovědu
2. Restart: zahodí veškerý pokrok a začne evoluci znova
3. Překreslit: zobrazí efekt nově zadaných nastavení
4. Zpět: vrátí evoluci o 1 krok
5. Variabilita: relativní velikost změny za 1 evoluční krok; posuvník
6. Počet objektů: počet objektů ve vykreslovaných obrázcích; volba z možností
7. Rotace: povolí SVG objektům rotovat
8. Stupně šedi
9. Číslo generace
10. Počet nodů CGP: délka CGP genotypu; textový vstup, očekává celé číslo 10-80

Dále implementuji možnost obrázek stáhnout a zvětšit. Tlačítka pro tyto funkce jsou umístěna v levém horním rohu každého obrázku. Aby nebránila v pohledu na obrázek, jsou skrytá. Částečně se zviditelní, najede-li uživatel kurzorem myši nad konkrétní obrázek. Úplně se zobrazí až v okamžiku, kdy je kurzor přímo nad tlačítkem.

4.8 Budoucí rozšíření

Je několik funkcí, které jsem do výsledné aplikace neimplementoval. Jedná se o funkce, které se nedotýkají přímo zadání mé práce, ale které by měly zlepšit komfort uživatele.

Interaktivní galerie je první takovou funkcí. Tématu mé práce se netýká, ale možnost vystavit své obrázky by tázaní uživatelé uvítali. Dodání této funkcionality ale představuje

poměrně velkou časovou investici. Současná aplikace vůbec neobsahuje server, který je pro galerii nezbytný.

Konverze obrázku do PNG formátu je funkce, která není implementačně až tak náročná. Bohužel mě nenapadlo ji implementovat a testující uživatel o ni požádal až v okamžiku, kdy dokončuji tento text.

Sdílení obrázku na sociálních sítích potřebuje první implementovat konverzi do PNG. Opět není až tak náročné ji dodat, ale upozornil mě na ni až stejný uživatel.

Vzhledem k velmi dobrým ohlasům od uživatelů a relativně velkému zájmu o aplikaci počítám s tím, že tyto funkce budu ještě implementovat.

Kapitola 5

Testování

Při návrhu metody bylo testování součástí tvůrčího procesu v každé iteraci. Vždy jsem navrhl úpravu, implementoval ji a otestoval. Aplikace tak prošla velkým množství verzí. Samotné testování hotové metody není možné provést objektivně. Výsledkem metody jsou obrázky, jejichž kvalitu určuje estetická hodnota. Tu samozřejmě není možné objektivně měřit.

Testování tedy bylo naprosto závislé na subjektivním hodnocení testujících uživatelů. S několika jsem spolupracoval po celou dobu vývoje aplikace. Finální aplikaci jsem pak vystavil na web a oslovil o něco širší spektrum přátel a známých. Jejich reakce jsem zpracoval formou dotazníku.

5.1 Dotazník

V dotazníku jsem se ptal na předchozí zkušenosti uživatele s evolučním uměním. Všechny další otázky se týkaly jeho zkušenosti s mou aplikací. Konkrétně se týkaly času stráveného s aplikací a zábavnosti, intuitivnosti uživatelského rozhraní a směrovatelnosti evoluce.

Mezi oslovenými se našlo 12 lidí, které aplikace zaujala a strávili s ní nějaký čas. Odpovědi jsou také znázorněny v grafech (Obr. 5.1).

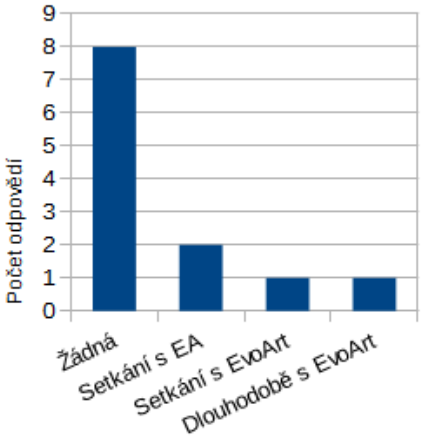
Zábavnost aplikace (5.1c) hodnotí relativně vysoko (průměrně asi 4.2 z 5) a strávili s ní alespoň pět minut, cca polovina více než 15 minut.

Plynulost aplikace (5.1d) hodnotí jako dostatečnou (průměr 4.8 z 5). Jde samozřejmě o subjektivní hodnocení, ale to, jak dobře se uživatelům pracuje je možná relevantnější než přesná výpočetní doba pro evoluční krok.

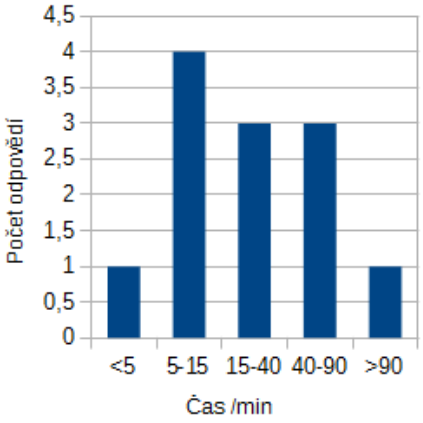
Uživatelské rozhraní (5.1e) hodnotí většinou jako relativně intuitivní (průměr 4 z 5). Jediný uživatel, kterému uživatelské rozhraní přišlo nepřehledné, má zkušenosti s jiným software pro tvorbu evolučního umění. Lidé s minimálními předchozími znalostmi, kteří jsou mou cílovou skupinou, se orientovali velmi dobře.

Směrovatelnost (5.1f) byla hodnocena lehce nadprůměrně (3.8 z 5), v jednotlivých hodnoceních je ale relativně velký rozptyl. Většina uživatelů ale nemá zkušenosti s problematikou a tudíž jim chybí srovnání. Osobně považuji možnost uživatele směřovat evoluci za největší slabinu své aplikace.

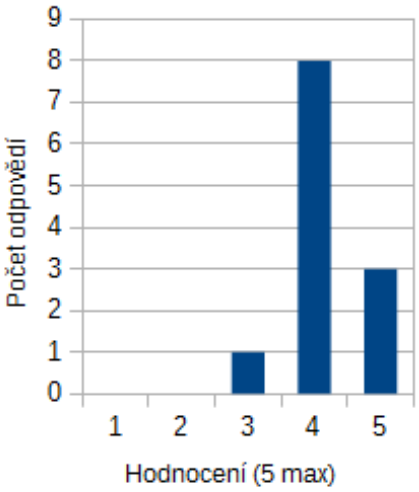
V prostoru pro komentáře jsem získal další zpětnou vazbu. Objevil se požadavek na sdílení na sociálních sítích, ukládání celé historie evoluce, přechod rodiče do nové generace, výběr více rodičů.



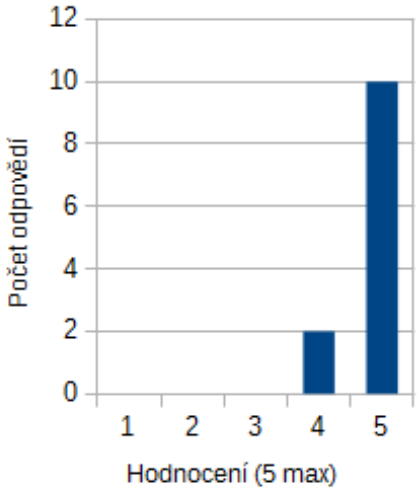
(a) Předchozí zkušenost s EvoArt



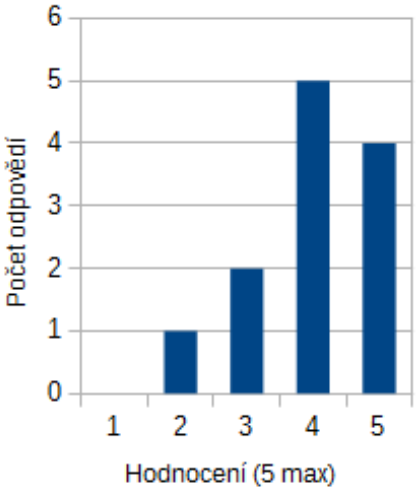
(b) Čas strávený u aplikace



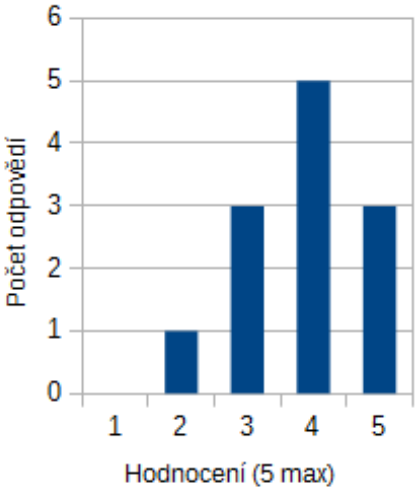
(c) Zábavnost



(d) Plynulost



(e) Intuitivnost UI



(f) Směrovatelnost evoluce

Obrázek 5.1: Výsledky dotazníku

Také jsem získal zpětnou vazbu o příliš nízké směrovatelnosti evoluce, kdy se po dlouhém směřování ke konkrétnímu obrázku naráz objeví naprosto odlišná populace.

Kapitola 6

Výsledky metody

Navržená metoda přímého generování vektorů je schopná vytvářet esteticky zajímavé obrázky. Pro zhodnocení dosažených výsledků používám srovnání s metodou rastrového CGP ve formě, v jaké jsem ho implementoval.

6.1 Průchod prostorem řešení

Standardním chováním CGP je to, že na počátku evoluce je fenotyp jedinců relativně krátký. V průběhu generací pak selekcí délka fenotypu v populaci roste. Relativně složitější výstupní funkce většinou vedou k vyšší hodnotě fitness. Po jisté době se délka fenotypu ustálí.

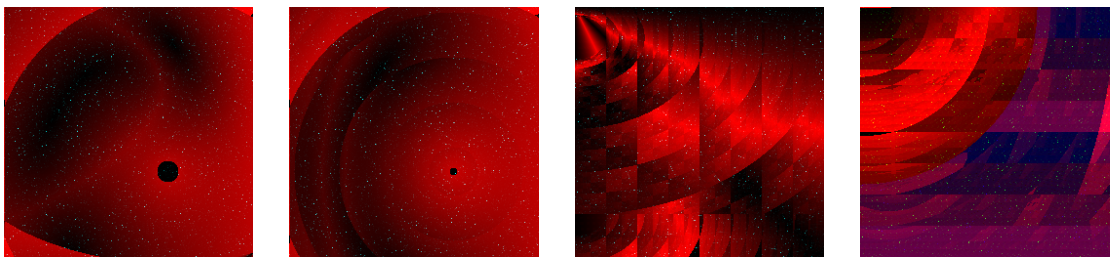
Stejný průběh se dá pozorovat i v obou mých implementacích, rastrové i vektorové, kdy je fitness definována uživatelem.

Rastrové CGP

V mojí aplikaci rastrového CGP má tento proces spíše destruktivní účinky. Po několika málo generacích jsou generované obrázky esteticky zajímavé. Další evolucí a růstem složitosti generovaných matematických funkcí se ale začne obrázek vizuálně tříštit.

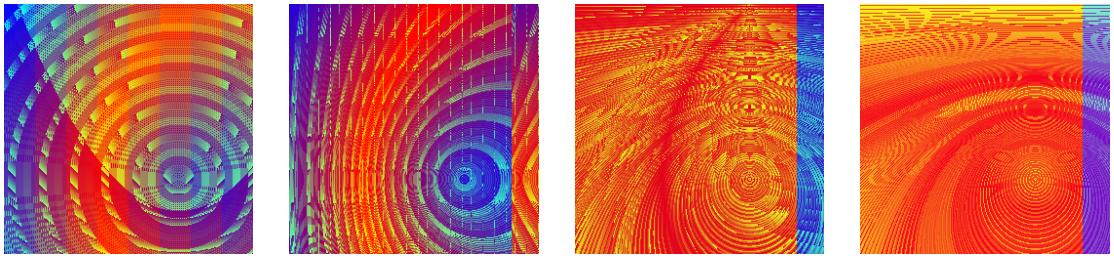
Udržovat směr evoluce požadovaným směrem je většinou relativně jednoduché. Naopak odklon naprosto jiným směrem je v okamžiku, kdy se evoluce relativně ustálí (zhruba 20-40 generací) jen velmi pomalý.

Problém nastává v okamžiku, kdy se chceme od vizuálně složitějšího obrázku dostat k jednoduššímu. Podobná změna je velmi náročná a trvá často řadu generací.



(a) Gen. 14 (b) Gen. 30 (c) Gen. 60 (d) Gen. 90

Obrázek 6.1: Evoluční proces rastrového CGP (1)



(a) Gen. 12 (b) Gen. 25 (c) Gen. 33 (d) Gen. 50

Obrázek 6.2: Evoluční proces rastrového CGP (2)

(a) Gen. 8 (b) Gen. 18 (c) Gen. 27 (d) Gen. 28

Obrázek 6.3: Evoluční proces vektorového CGP

Vektorové CGP

U vektorového CGP je často několik (až několik desítek) generací obrázků relativně nezajímavých. Teprve s vyšší délkou fenotypu se začnou objevovat vizuálně zajímavé struktury. Pokles v kvalitě generovaných obrázků ale s délkou běhu algoritmu nenastává, alespoň ne v několika stech krocích.

Směrovatelnost evoluce je relativně nízká. Udržovat populaci vizuálně podobnou současnému stavu je reálné, především při snížení variability. Vzhledem k tomu, že CGP obsahuje 9 výstupů, se ale většina změn projevu hned v několika z nich. Nalezení potomka, který se od rodiče liší pouze požadovaným způsobem je tak často nereálné.

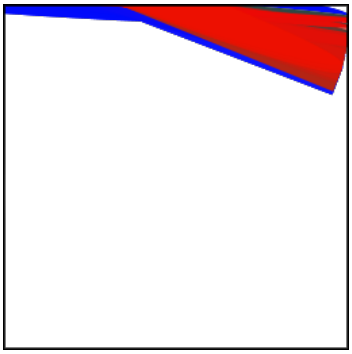
Zamykání těch částí fenotypu, které se váží k některému výstupu, by mohlo pomoci. Na rozdíl od tří výstupů rastrového CGP moje implementace pracuje s devíti výstupy. Při uzamčení více výstupů by se zamkl prakticky celý genotyp.

Naopak potomci, kteří se diametrálně liší od rodiče, ale jsou vizuálně zajímaví, se vyskytují běžně. Celý proces evoluce je méně zaměřený na tvorbu konkrétního obrázku. Namísto toho se důraz posouvá k exploraci potenciálních obrázků a nacházení nečekaných ale zajímavých výsledků.

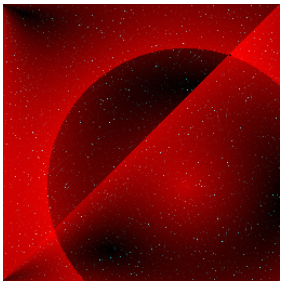
6.2 Prázdné obrázky

Vektorové CGP může potenciálně generovat vizuálně špatné obrázky. Pokud se výška nebo šířka všech objektů zafixuje na nule, vykreslí se prázdný obrázek. Někdy se také veškeré SVG objekty nakupí do jednoho místa a zbytek obrázku je prázdný.

Po opravě implementačních chyb se ale takovéto případy vyskytují jen sporadicky. S výjimkou nulté generace, kdy se občas vyskytne více takových jedinců, se prakticky nikdy neobjeví více než jeden v populaci.

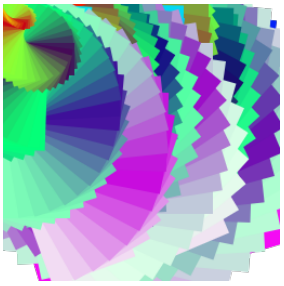


Obrázek 6.4: Příklad téměř prázdného obrázku



(a) Rastrové CGP

(b) VecCGP - jemný přechod



(c) VecCGP - typické

Obrázek 6.5: Gradienty v generovaných obrázcích

Bylo by možné podobné případy detekovat, ale to by zabralo další výpočetní čas a prodloužilo odezvu aplikace. Toto zpomalení by snížilo komfort práce s aplikací výrazně více než občasné prázdné okno.

6.3 Generované obrázky

Implementace rastrového i vektorového CGP používají stejnou sadu matematických funkcí. Rozdíly v generovaných obrázcích jsou důsledkem způsobu vykreslení, tedy přímo rozdílem mezi metodami.

Obě metody generují vizuálně zajímavé obrázky. Ty jsou ale typicky vizuálně odlišné a některé rysy se vyskytují pouze u výsledků jedné z metod.

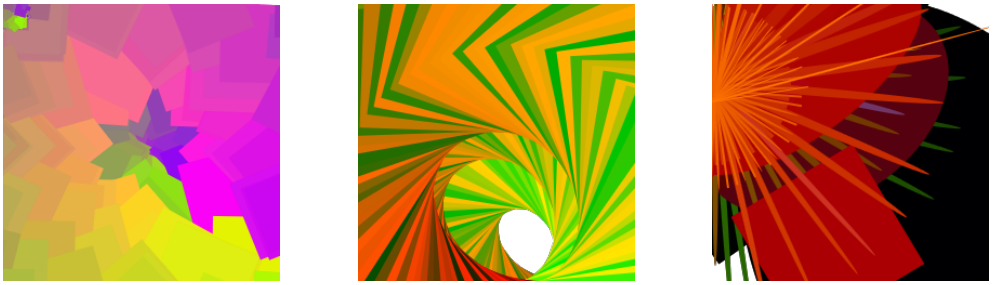
Gradienty

V rastrovém CGP se gradienty vyskytují velmi často. Většina vizuálně kvalitních obrázků, které rastrové CGP generuje, obsahuje jemné barevné přechody.

Ve vektorovém CGP jsou gradienty výrazně vzácnější. Jednotlivé SVG objekty se musí vykreslovat s minimálním přesahem a barevným rozdílem. Daleko častější jsou hrubší stupně přechodu.

Vliv rotace

Při povolení rotace se začnou objevovat vzory, které rastrové CGP není schopné generovat. Jde především o spirály a hvězdice. Tyto obrázky mohou být vizuálně velmi zajímavé.



Obrázek 6.6: Vliv rotace ve vektorových obrázcích

6.4 Využití v užitém umění

V současné implementaci není dost dobře možné směřovat evoluci ke konkrétnímu obrázku. Využití generovaného obrázku přímo v užitém umění tak není dost dobře možné.

Na druhou stranu umožňuje aplikace velmi rychlý průchod prostorem řešení a shlédnutí velkého množství vizuálně zajímavých motivů. Využití vidím především v nalezení motivu, který později uživatel rozvine za použití jiného nástroje. Aplikace tedy může být zdrojem inspirace pro vytvoření loga nebo jiného grafického celku.

Kapitola 7

Závěr

Cílem práce bylo navrhnout a implementovat metodu, jak aplikovat kartézské genetické programování do oblasti evolučního umění.

Seznámil jsem čtenáře s problematikou evolučních algoritmů a vysvětlil vývoj, který stojí za vznikem CGP. Rozebral jsem princip CGP a jeho dřívější aplikaci do oblasti evolučního umění.

Navrhl jsem metodu, jak CGP využít pro generování vektorových obrázků, implementoval ji a otestoval. Obrázky, které metoda generuje, jsou esteticky zajímavé. Obrázky jsou typicky vizuálně odlišné od těch, které generuje rastrové CGP. Některé vizuální rysy, které často obsahují, se v rastrovém CGP prakticky nevyskytují.

Hlavním problémem metody je nízká směrovatelnost. CGP genotyp obsahuje v současné implementaci 9 výstupů. Změna v genotypu tak vyvolává dalekosáhlé změny na vizuální stránce obrázku.

V průběhu evoluce dochází k rychlému průchodu prostorem řešení a uživatel se rychle potkává s novými motivy. Využití metody v užitém umění vidím v hledání inspirace pro následnou tvorbu.

Cílovým uživatelem vytvořené aplikace pro mě byl člověk bez větších zkušeností s evolučním uměním a evolučním programováním. Aplikace mu měla umožnit jednoduše a rychle generovat esteticky zajímavé obrázky.

Věřím, že se mi tohoto cíle podařilo dosáhnout. Aplikace na většině testovaných strojů reaguje rychle a proces tvorby je plynulý. Většina testujících uživatelů hodnotí aplikaci jako zábavnou. Několik jich strávilo tvorbou obrázku opakovaně desítky minut. Ohlas uživatelů mě příjemně překvapil. Aplikaci považuji za úspěšnou a chci se jejímu vývoji dále věnovat.

Literatura

[1] Dart: Oficiální stránky projektu. https://www.dartlang.org/, cit.2015-05-11.

[2] jQuery: Oficiální stránky projektu. http://jquery.com/, cit.2015-05-11.

[3] *JQuery*. Brno: Computer Press, vyd. 1. vydání, 2010, ISBN 9788025131527.

[4] Ashmore, L.; Miller, J.: Evolutionary Art with Cartesian Genetic Programming. *Technical Online Report*, 2004.

[5] Hynek, J.: *Genetické algoritmy a genetické programování*. Praha: Grada, první vydání, 2008, ISBN 9788024726953.

[6] Koza, J. R.: *Genetic programming*. Cambridge : London: Bradford Book, MIT Press, 1992, ISBN 0262111705.

[7] Miller, J.: *Cartesian genetic programming*. New York: Springer Berlin Heidelberg, c2011, ISBN 9783642173097.

[8] Miller, J. F.; Thomson, P.: Cartesian genetic programming. In *Genetic Programming*, Springer, 2000, s. 121–132.

[9] Pospíchal, J.; Kvasnička, V.; Tiňo, P.: *Evolučné algoritmy*. Bratislava: Slovenská technická univerzita v Bratislave, první vydání, 2000, ISBN 8022713775.

[10] Romero, J.; Machado, P.: *The art of artificial evolution*. Berlin: Springer, 2008, ISBN 9783540728764.

[11] Schwarz, J.: *Aplikované evoluční algoritmy*. Brno: Fakulta informačních technologií, 2008.

[12] Sekanina, L.: *Evoluční hardware*. Praha: Academia, vyd. 1. vydání, 2009, ISBN 9788020017291.

Příloha A

Obsah CD

Přiložené CD obsahuje:

- text této práce ve formátu PDF
- zdrojový text práce ve formátu LaTeX
- zdrojové kódy výsledné aplikace vektorového CGP
- spustitelnou verzi aplikace vektorového CGP ve formě WWW stránky
- příklady obrázků vygenerovaných aplikací vektorové CGP
- zdrojové kódy mé implementace rastrového CGP
- spustitelnou verzi mé implementace rastrového CGP ve formě WWW stránky
- příklady obrázků vygenerovaných aplikací rastrové CGP
- uživatelský manuál k aplikaci vektorového CGP
- návod k překladu a spuštění aplikace