Krásy počítačové grafiky 2021

Stránka předmětu http://afrodita.zcu.cz/~kolinger/vyukaZCU.html#KPG
Stránka cvičícího https://home.zcu.cz/~manak/
Informace ke cvičení http://kpg.dynu.net - přístup přes GAPPS: **<orion_login>@gapps.zcu.cz**
Vzdálená výuka http://zoom.kpg.dynu.net - bude probíhat přes ZOOM, středa 14:50 - 16:30
Repozitáře GitLab https://gitlab.kiv.zcu.cz

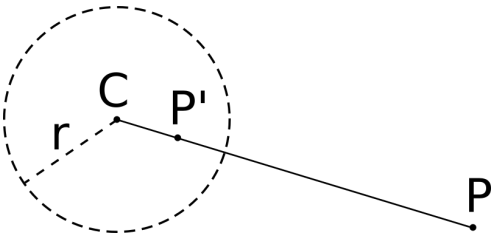
Na zápočet je potřeba získat alespoň **30** bodů - tabulka s body.

- Aktivní účast na cvičení cca až 1 - 2 body / cvičení
- Semestrálka za cca 15 bodů
  - Možnost vlastního zadání
  - Téma symetrie
  - Zadání cca do **konce března**
  - Odevzdání ideálně do konce května až půlky června, v odůvodněných případech lze i později
  - Odevzdání do repozitáře na GitLabu (adresář semestral-work)
  - Součásti:
    - Zdrojové kódy + všechny závislosti. Je vítán přenositelný kód, cvičící zpravidla nepracuje pod Windows.
    - Návod jak to přeložit, spustit a vyzkoušet (README.txt)
    - Dokumentace
      - PDF
      - Co za problém jste řešili?
      - Jak jste ho řešili?
      - Jakých jste dosáhli výsledků?
      - Jaké cizí knihovny nebo zdroje, či inspiraci z webu jste použili (reference)?
- Bonusové body

13) Závěrečná hodina

- kontrola a bodování zbývajících úkolů, které třeba původně nešly přeložit
- ukázka point-cloud dat z laserového skenování krajiny a městské zástavby, možnost spolupráce na tématech z této oblasti v dalším semestru nebo v rámci kvalifikační práce
- zpětná vazba od studentů na probíraná témata

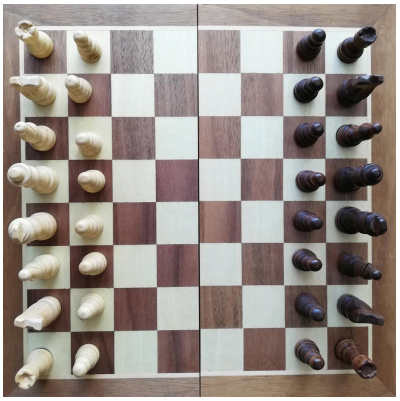
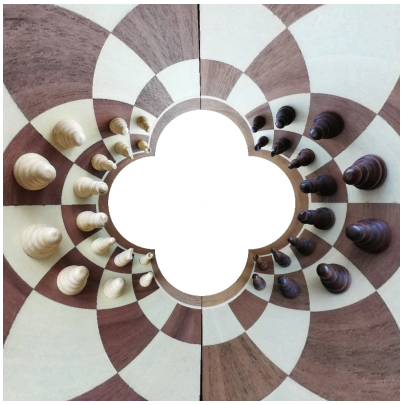
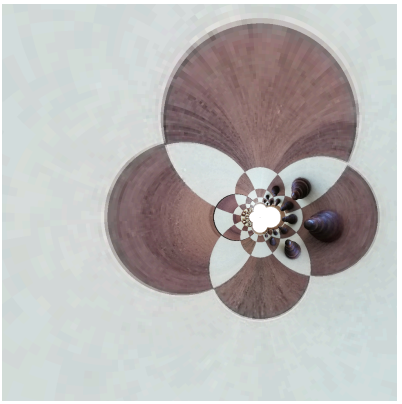
12) Kruhová inverze

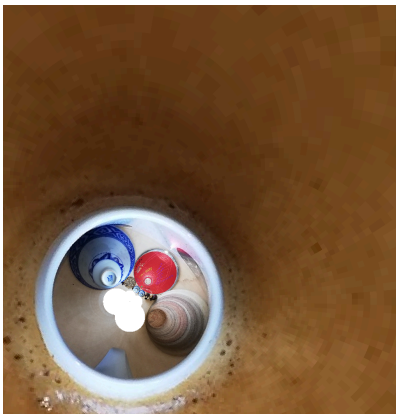


Mějme zadaný kruh pomocí středu **C** a poloměru r. Zobrazení, které každý bod **P** ≠ **C** zobrazí na bod **P'**, přitom splňuje vztah |**CP**| * |**CP'**| = r^2, bod v nekonečnu zobrazí na střed **C**, a střed **C** na bod v nekonečnu, se nazývá *kruhová inverze*. Výpočet.

Toto zobrazení zobrazuje kružnice a přímky na kružnice nebo přímky, zachovává úhly protínajících se křivek, ale nemusí zachovávat vzdálenosti.

Lze tak například transformovat šachovnici na něco “neskutečného” nebo transformovat kruhové objekty mezi sebou. Obrázky k dnešnímu cvičení jsou k dispozici zde.




Úkol

Napište program, který načte zadaný obrázek a aplikuje na něj kruhovou inverzi. Střed a poloměr inverze půjde v programu měnit. Program vyzkoušejte na fotografie, které sami pořídíte a pokuste se na nich vytvořit nějaký zajímavý efekt nebo scénu. Dále svůj program aplikujte na "*questové*" obrázky q1 a q2, pokuste se co nejlépe obnovit jejich původní vzhled a na základě toho uhodnout, co se na nich nachází.

11) Malé 3D

Vizuální reprezentaci virtuálního trojrozměrného světa si lze představit jako scénu, ve které jsou nějaké **3D objekty**, **kamera** a další věci jako např. světla. Tvar 3D objektu může být popsán např. nějakou povrchovou reprezentací (zpravidla trojúhelníková síť). Součástí vizuální reprezentace objektu je také popis materiálu povrchu (zpravidla textury a programy pro výpočet barvy a osvětlení při dopadu paprsku světla), a další věci (stíny, post-procesingové efekty, ...).

Kamera definuje pohled do virtuálního světa, který se zobrazuje na rovinu průmětny. Část světa, který kamera zabírá, je tzv. pohledový objem (frustum). Scénu je potřeba ještě zobrazit uživateli - probíhá rasterizace do uživatelského okna, tedy tvorba samotných pixelů výsledného obrázku.

Důležité parametry kamery

- pozice pozorovatele ve virtuálním světě (position)
- směr pohledu (dir-vector)
- orientace (up-vector), abychom tzv. věděli "kde je nahoře"
- typ projekce (perspektiva / rovnoběžná)
- viewport (šířka a výška obdélníkového okna v pixelech (případně ještě offset x a y))

V následujícím textu používám zvyklosti z OpenGL (dobou prověřené, viz historie > 25 let). Pěkné ilustrace k těmto pojmům lze najít např. v tomto tutoriálu.

Virtuální svět má svůj globální souřadný systém, tzv. **world-space**.

Každý objekt má svůj lokální souřadný systém, tzv. **object-space**. To, jak je objekt natočený a jakou má pozici ve world-space, je popsáno příslušnou transformací - maticí **M** rozměru 4x4 popisující třeba nějakou složitější sekvenci transformací aplikovaných za sebou (různé rotace, posuny, změny měřítka, atd). Pro rotace a měřítko by nám stačily matice 3x3, ale díky velikosti 4x4 můžeme popsat i posunutí

(translaci), můžeme pracovat s homogenními souřadnicemi (a potažmo body v nekonečnu), provádět perspektivní projekci, atd. Matice M, která transformuje object-space do world-space, se jmenuje **model-matrix**. Transformace bodu z object-space do world-space je pak pouhé násobení:

$$(x', y', z', w')^T = \mathbf{M} * (x, y, z, 1)^T.$$

Pro potřeby zobrazení virtuálního světa z pohledu kamery se používá ještě několik dalších prostorů a transformací:

World-space je potřeba transformovat do lokálního prostoru kamery, tzv. **camera-space**, a to tak, aby se pozice kamery převedla do počátku soustavy souřadnic (0, 0, 0), dir-vector odpovídal záporné ose Z (0, 0, -1), vektor "pravé ruky", který je kolmý rovině vektorů dir a up, odpovídal ose X (1, 0, 0) a up-vector odpovídal ose Y. Z vektorů kamery dir-vector a up-vector lze sestavit ortonormální systém souřadnic (3 vektory délky 1, které jsou na sebe kolmé), a sestavit matici **V** rozměru 4x4 pro změnu systému souřadnic. Řádky matice odpovídají těmto 3 vektorům a poslední sloupec matice obsahuje posun pozice kamery do počátku. Matici **V** převádějící world-space -> camera-space se říká **view-matrix**.

Kamera dělá nějakou projekci (rovnoběžnou / perspektivní). Tím vzniká pohledový objem (frustum - u rovnoběžné projekce se jedná o kvádr, u perspektivní o komolý jehlan). Tato projekce je popsaná maticí **P** rozměru 4x4, tzv. **projection-matrix**. Po této transformaci už zpravidla nemáme čtvrtou souřadnici (w) rovnu jedné, ale můžeme provést dělení (x/w, y/w, z/w, w/w) a získat tak příslušný bod v 3D prostoru. Této operaci se říká **perspektivní dělení** a důležité je, že transformuje pohledový objem na kostičku velikosti 2x2x2. Prostoru této kostičky se říká **normalized-device-coordinates**, zkráceně **NDC**. Její levý dolní roh má souřadnice (-1, -1, -1) a pravý horní roh (1, 1, 1). Přední stěna (z = -1) kostičky odpovídá přední ořezávací rovině, zadná stěna (z=1) odpovídá zadní ořezávací rovině pohledového objemu.

Vše, co je mimo NDC, se ořízne (**clipping**), protože je to mimo pohledový objem.

Přední stěna kostičky NDC se roztáhne do okna **viewport** a rozsah hloubky kostičky (od -1, do 1) se namapuje do rozsahu (0, 1), aby se lépe využily bity pro informaci o vzdálenosti bodů od pozorovatele (pro z-buffer). Tím se transformuje NDC do prostoru okna, tzv. **window-space**, a zde už se dělá rasterizace a zobrazení uživateli.

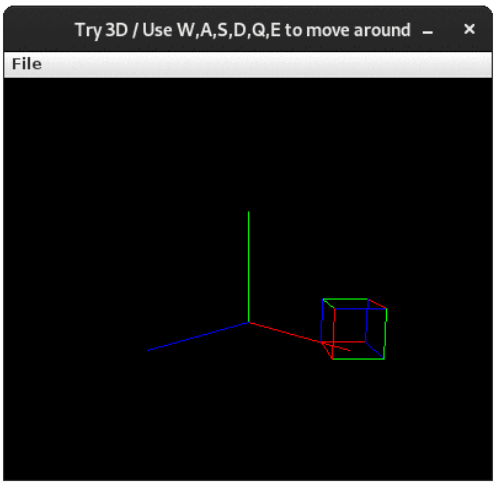
Všechny tyto transformace lze spojit dohromady pomocí násobení matic. Takže výsledná transformace do homogenních souřadnic je $(x', y', z', w') = (\mathbf{P} * \mathbf{V} * \mathbf{M}) * (x, y, z, 1)^T$. Výsledné matici se říká **model-view-projection matrix**. Tímto postupem se transformovaly body, normálové vektory se řeší trochu jinak (některé z matic budou inverzní resp. transponované).

Matici **P** pro **perspektivní projekci** lze definovat buď na základě field-of-view, poměru stran okna a pozice přední a zadní ořezávací roviny (gluPerspective), nebo na základě definice ořezávacích rovin (glFrustum).

Matici **P** pro **rovnoběžnou projekci** lze definovat na základě ořezávacích rovin (glOrtho).

Úkol

Vyzkoušejte si výše uvedenou teorii v praxi. V repository k 11 cvičení máte k dispozici aplikaci Try 3D, která implementuje velmi základní zobrazení jednoduché 3D scény (pouze zobrazování úseček). Zobrazí se osa X, Y, Z a hrany krychličky po transformaci (změna měřítka, rotace a posun). Ve scéně je kamera, kterou lze ovládat pomocí klávesnice: W pohyb rovně, S vzad, A otoč vlevo, D otoč vpravo, Q pohyb vlevo a E vpravo.



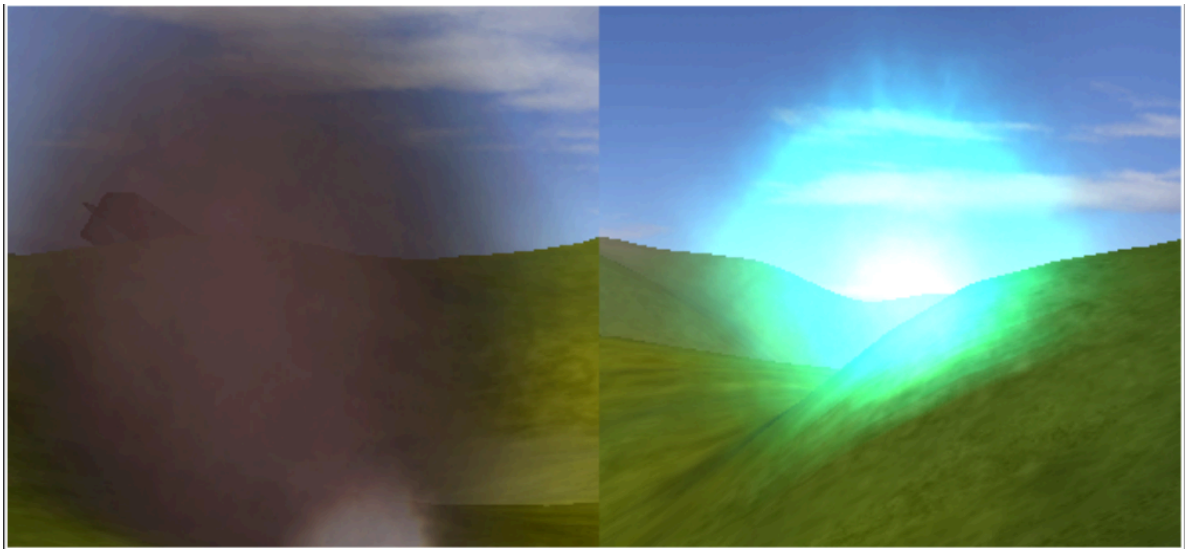
Projekt byl vytvořen v NetBeans 11.3 / Maven / Java 1.8. Pokud máte nainstalovaný Maven, tak projekt jde zkompilovat i jen z příkazové řádky. Automaticky se stáhne a přibalí závislost javax.vecmath 1.5.2.

Update stavu repository: `git pull origin main`
Kompilace: `mvn package`
Spuštění: `java -jar target/try-3d-1.0-SNAPSHOT-jar-with-dependencies.jar`

Aplikaci rozšiřte nějakým kreativním způsobem a vyzkoušejte si skládání transformací objektů pomocí násobení matic. Lze například implementovat import objektů, nebo do scény vygenerovat nějaký základní fraktál pomocí technik z předchozích cvičení, použít částicový systém pro ohňostroj, implementovat plynulejší pohyb, lepší ovládání, atd.

10) Efekty - částicové systémy

Jedno z mnoha využití částicových systémů jsou například efekty explozí a kouře. Na “motivačním” příkladu níže (projekt BJS) je vidět efekt kouře, exploze radioaktivního zeleného slizu a jeden odvážnější výbuch. Všechny tyto efekty byly vytvořeny s využitím jedné jediné poloprůhledné textury, několika strategií míchání barev a několika málo částicových systémů, bez využití shaderů.



Ačkoliv jde s částicovými systémy dosahovat skvělých efektů, velmi snadno se takový systém může stát problematickým z hlediska výkonu aplikace. Pokud je například 50 hráčů ve hře, všichni střílí, pak i částic k simulaci bude velmi mnoho. Pokud je navíc každá částice zobrazována jako poloprůhledný sprite, vizualizace bude velmi náročná z hlediska vyššího faktoru překreslování. Někdy to lze vyřešit např. renderingem do textury s nižším rozlišením a následnou aplikací výsledného efektu. S částicovými systémy je radno nakládat opatrně.

Na procvičení tvorby částicového systému ve 2D je k dispozici základní aplikace particle-app. Zdrojové kódy projektu jsou uvnitř JAR (Java 11, NetBeans 11.3, Maven). V aplikaci je definována struktura částic, částicový systém pak simuluje jejich pohyb a aplikuje gravitační sílu, která částice táhne dolů k zemi. Na klik myši se vytvoří exploze barevných částic (100 částic na náhodných pozicích v roztaženém a náhodně pootočeném obdélníku v oblasti kliku, trochu náhodných barev, rychlosti, atd.) Pokud bychom chtěli vytvořit např. efekt exploze, aplikovali bychom aditivní alpha-blending - skládáním barev překrývajících se částic by došlo k zesvětlení jejich barev do bílé.

Simulace částicového systému byla vytvořena podle článku:
[An Introduction to Physically Based Modeling: Particle System Dynamics]

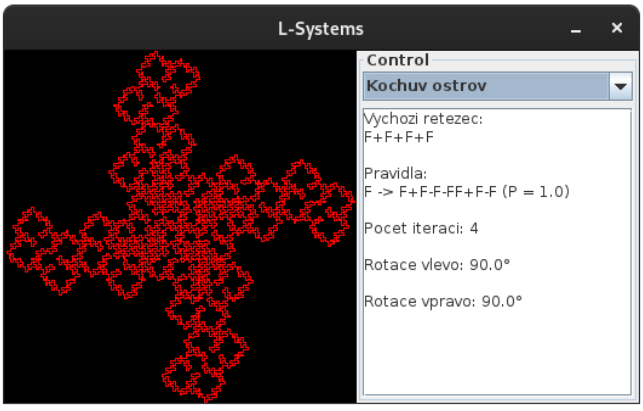
Úkol: Ohňostroj

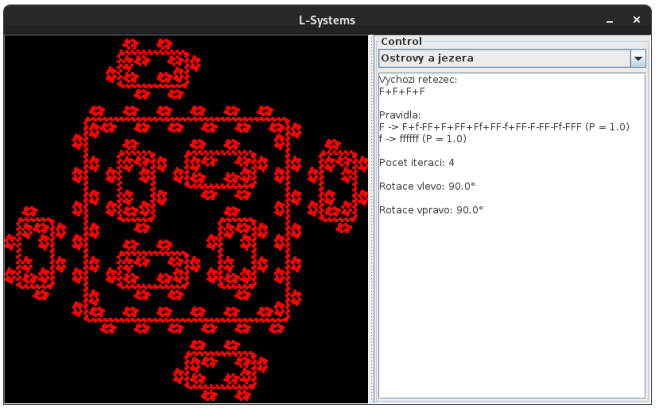
Vytvořte 2D ohňostroj nebo nějaký jiný vlastní zajímavý efekt pomocí částicového systému. K tvorbě efektu lze využít výše uvedenou aplikaci nebo si napsat vlastní. Součástí efektu ohňostroje mohou (ale nemusí) být např. trajektorie raket, záblesky, kouř, problikávání částic (třeba intenzita podle postupně tlumené sinusoidy), efekt větru, nějaký tmavý obrázek na pozadí, který budou exploze osvětlovat, ale hlavně, spektakulární exploze. Efekt by měl být zajímavější než ta animace výše.

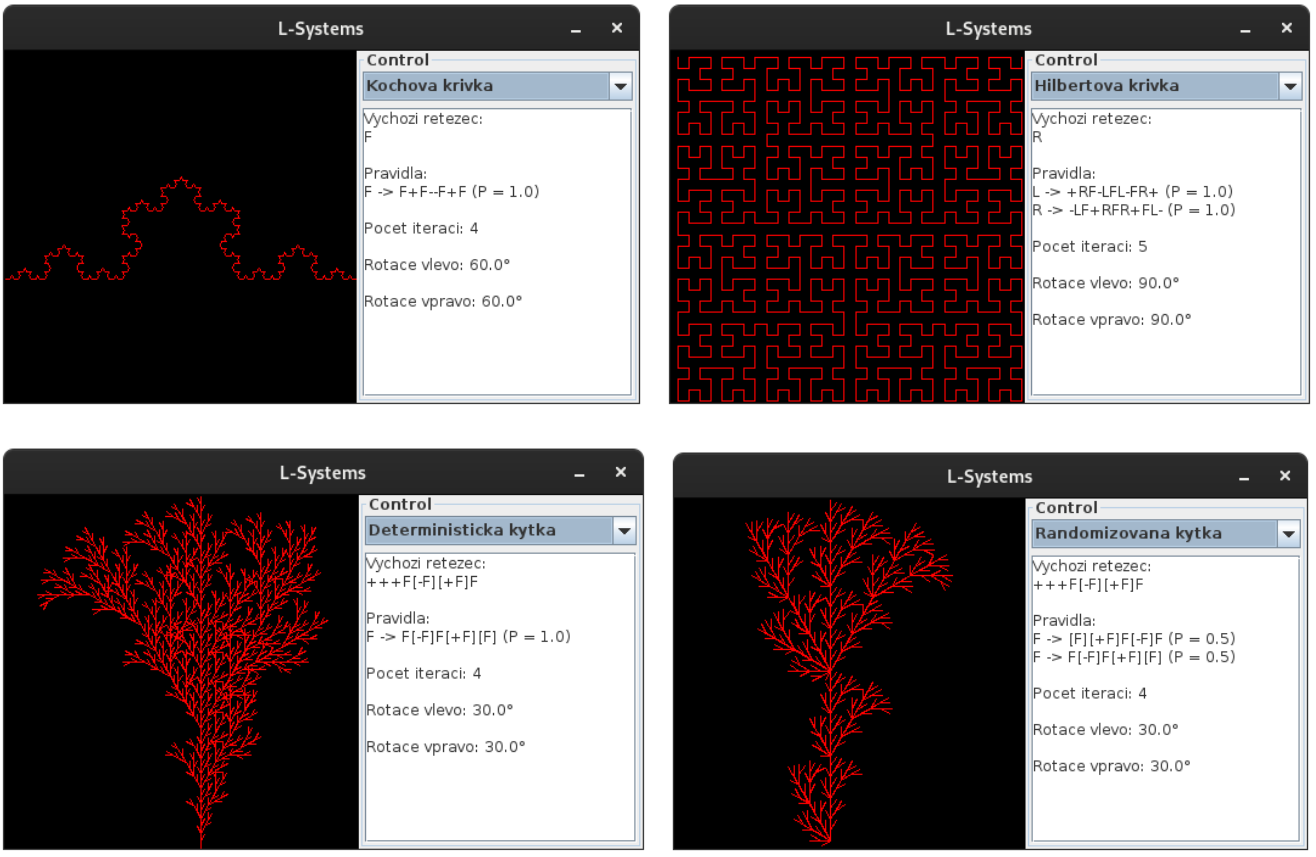
9) L-Systémy - 14. 4. 221

Pozor, nezaměňovat [L-Systém] a [H-Systém].

Na procvičení L-systémů je k dispozici jednoduchá aplikace (zdrojová kód máte v repository v adresáři se cvičením 09, Maven projekt v NetBeans 11), která implementuje výpočet a zobrazení některých základních předdefinovaných L-systémů. Vyzkoušejte si jednotlivé L-systémy, zkuste si definovat vlastní pravidla (zatím je to nutné udělat v kódu třídy `ApplicationFrame`).







Definice L-systému se provádí na základě gramatiky přepisovacích pravidel “symbol -> řetězec”. Známe výchozí slovo, symboly jazyka, přepisovací pravidla, ve stochastických L-systémech i pravděpodobnosti výběru pravidel, pokud volba pravidla není jednoznačná (např. pro 1 symbol více pravidel).

Aplikací přepisovacích pravidel na výchozí slovo se vytvoří další slovo jazyka, na ně lze opět aplikovat přepisovací pravidla a získat tak další slovo, atd. Pokud lze na jeden symbol aplikovat více přepisovacích pravidel a u pravidel máme pravděpodobnost jejich výběru, pak vybereme pravidlo náhodně (výše zmíněná aplikace to zatím neumí). Tato iterativní aplikace pravidel odvozuje (*derivuje*) slova jazyka popsaného L-systémem.

Vizualizace slov gramatiky L-systému se provádí interpretací znaků slova jako grafických příkazů. Používá se tzv. “*želví grafika*”, kdy jakoby ovládáme pohyb želvy (posun vpřed, otočení, ...) a kreslíme trasu, po které se želva pohybuje. Ve výše uvedených příkladech symbol ‘F’ znamená pohyb vpřed, symbol ‘+’ je otočení proti směru hodinových ručiček (o nějaký definovaný úhel), symbol ‘-’ je otočení želvy po směru hodinových ručiček, symbol ‘[’ ukládá stav (pozici a orientaci) želvy na zásobník a symbol ‘]’ znamená vyzvednutí stavu želvy ze zásobníku. Pomocí zásobníku lze např. tvořit větve kytek.

Mohlo by se zdát, že při vizualizaci nevíme, jak rozsáhlý bude výsledný obrázek, protože nedokážeme triviálně vyhodnotit, kudy se želva vydá a kam až dojde. Řešení je jednoduché. Vizualizace slova L-systému se udělá nadvakrát:

- V první iteraci se simuluje pohyb želvy a počítá se ohraničující obdélník (minima a maxima v jednotlivých osách). Potom se spočítá best-fit transformace ohraničujícího obdélníku do vizualizačního okna (1. vycentrování obdélníku, 2. roztáhnutí horizontálně / vertikálně na velikost okna a převrácení osy Y, a 3. translace do středu okna).
- Ve druhé iteraci se simuluje pohyb želvy a kreslí se její trajektorie. Souřadnice, na kterých se želva nachází, se ale transformují pomocí výše zmíněné transformace, a tak se vejdou do okna.

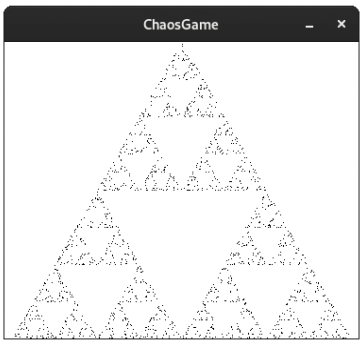
8) Chaos Game & IFS 7. 4. 2021

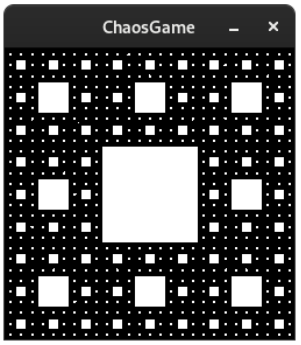
Na dnešním cvičení si vyzkoušíme vygenerovat několik základních fraktálů stylem Chaos game a IFS, jak bylo probíráno na přednášce.

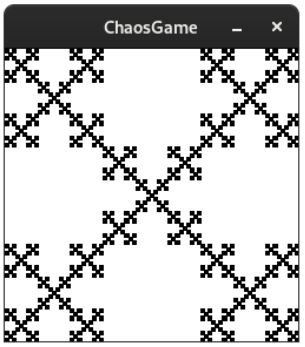
- V případě Chaos game jde o nastavení souřadnic několika málo rohových bodů $\mathbf{C}_i$ a váhy w, na základě které se (zprvu náhodně vygenerovaný) bod **P** přitahuje k náhodně zvolenému rohovému bodu $\mathbf{C}_k$ (k je náhodně zvolený index rohového bodu). Provádí se lineární interpolace:
  $\mathbf{P}' = \mathbf{P} * (1 - w) + \mathbf{C}_k * w$
  Tuto operaci opakujeme po zvolený počet iterací N (v každé volíme náhodný index k) a pro každou iteraci tento bod vykreslíme. Můžeme přeskočit prvních pár iterací, aby byl výsledek méně chaotický. Bod by měl náhodně skákat po přibližných pozicích na fraktálu.
- V případě IFS jde o to specifikovat koeficienty (a_i b_i c_i d_i e_i f_i) několika afinních transformací:
  $x' = a_i * x + b_i * y + e_i$
  $y' = c_i * x + d_i * y + f_i$
  a pravděpodobnost p_i výběru i-té transformace ze systému.
  Začneme s bodem na pozici (0,0). Na tento bod následně aplikujeme náhodně vybranou transformaci podle pravděpodobností, vykreslíme ho, pak zase náhodně vybereme a aplikujeme transformaci podle pravděpodobností, vykreslíme bod, atd.

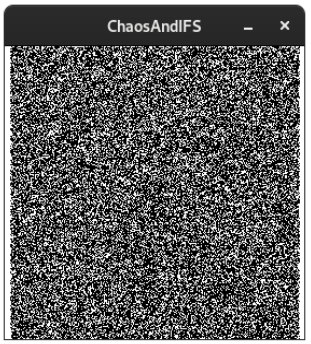
Úkol 1 Chaos game

Realizujte výpočet a vizualizaci Sierpinského trojúhelníku (první obrázek), Sierpinského koberce (druhý obrázek) a Viscekova fraktálu (třetí obrázek) pomocí Chaos game. Čtvrtý obrázek ukazuje “square” - případ, kde výsledkem Chaos game není fraktál, ale chaos. Můžete si vyzkoušet i další fraktály případně obarvení dle svého uvážení.



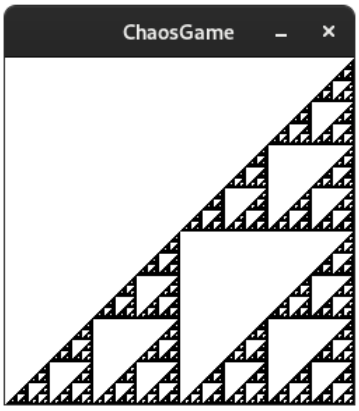






Úkol 2 Sierpinského trojúhelník a kapradí pomocí IFS

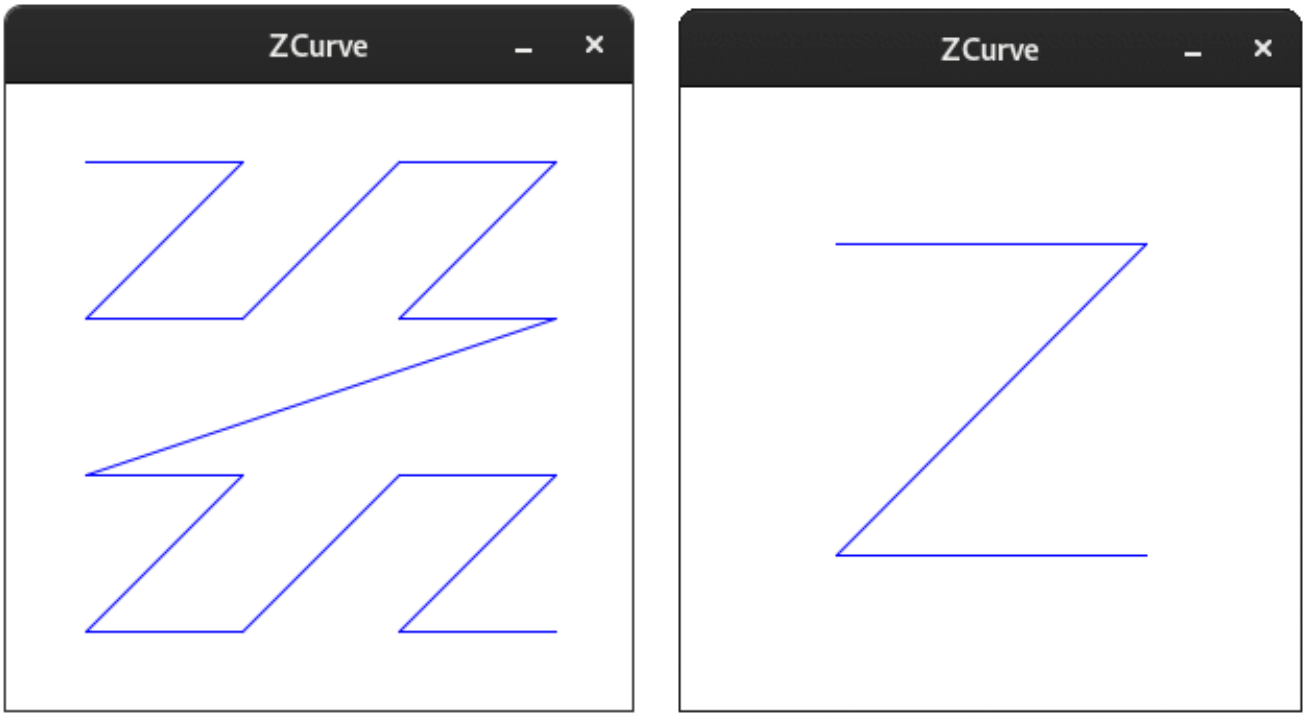
Realizujte výpočet a vizualizaci Sierpinského trojúhelníku a kapradiny pomocí systému iterovaných funkcí. Můžete si vyzkoušet i další fraktály případně obarvení dle svého uvážení.



7) Fraktály 31. 3. 2021

Příklad 1

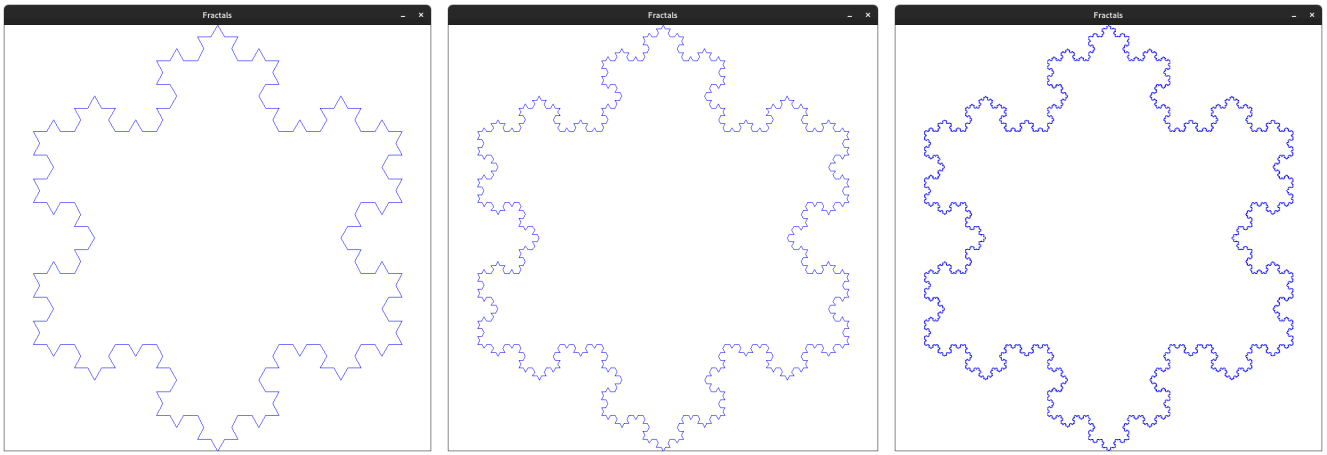
Vyzkoušejte si vykreslení Z-křivky. Vstupním parametrem bude počet iterací. K dispozici je soubor ZCurve.java, kde stačí dokončit implementaci metody `genZCurve(...)`.



Z-křivky jsou poněkud jednodušší než Hilbertovy křivky. Jedna z hlavních výhod Z-křivek je v tom, že pro zadané (celočíselné) souřadnice X a Y dokážeme určit souřadnici na Z-křivce prolnutím bitů ($...y_2x_2y_1x_1y_0x_0$). To se hodí v případech, kdy chceme uspořádat množinu bodů a zachovat jim do jisté míry lokalitu (mnohé body, které jsou prostorově blízko sebe, budou "blízko" sebe i na křivce).

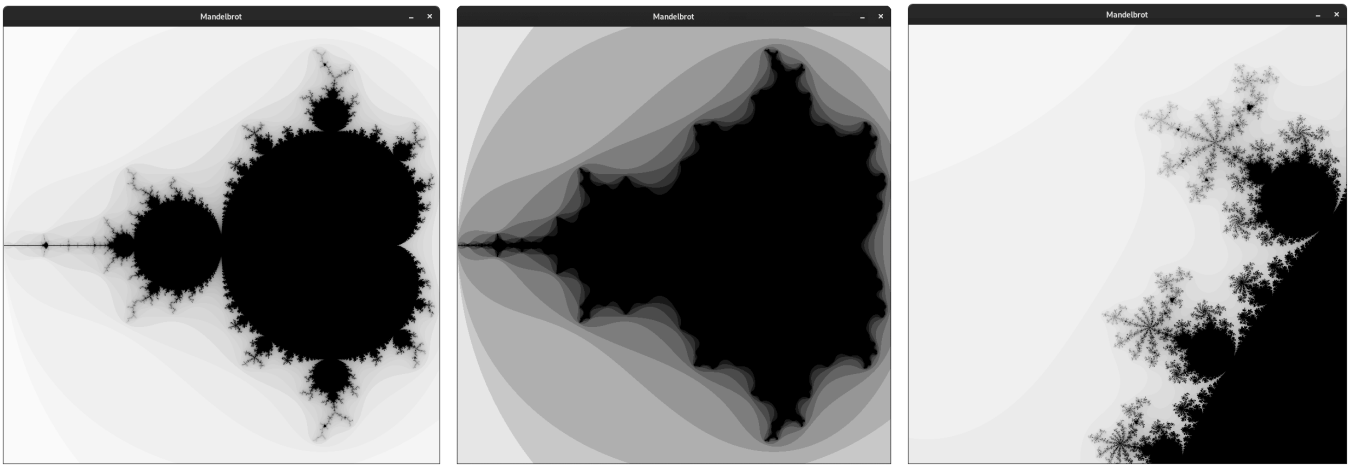
Příklad 2

Napište program pro generování Kochovy vločky. K dispozici je soubor Koch.java, kde stačí implementovat metodu `nextKochIteration()`. Dokážete se obejít bez goniometrických funkcí?



Příklad 3

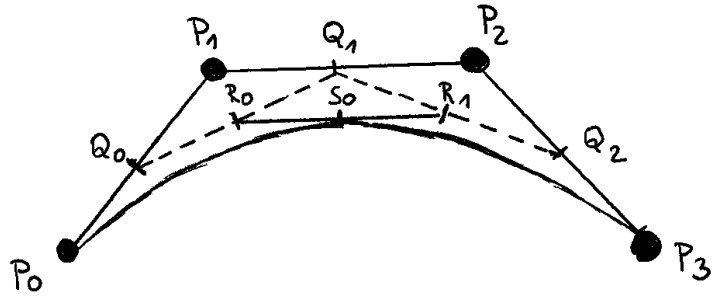
Napište program pro zobrazení Mandelbrotovy množiny (Pro komplexní číslo c uvažujeme předpis $z = z^2 + c$, iterujeme výpočet z po zadaný počet iterací (např. 100). Pokud se po zadaném počtu iterací číslo z nedostane do "kruhu konvergence" (nebude platit $re(z)^2 + im(z)^2 < 4$), vykreslíme bílou barvu pozadí, jinak vykreslíme černou barvu popředí. Možný je i barevný přechod nebo stupně šedi podle toho, ve které iteraci jsme detekovali divergenci.



6) Křivky II - 24. 3. 2021

Příklad 1 - ještě k tématu Bézierových křivek

KUBICKÁ BÉZIEROVA KŘIVKA - ADAPTIVNÍ DĚLENÍ



P_0, P_1, P_2, P_3 - řídící body Bézierovy křivky

$$C(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3, \quad t \in [0,1]$$

VÝPOČET BODU NA KŘIVCE PRO $t = \frac{1}{2}$ de Casteljau

$$Q_0 = \frac{1}{2}(P_0 + P_1), \quad Q_1 = \frac{1}{2}(P_1 + P_2), \quad Q_2 = \frac{1}{2}(P_2 + P_3)$$

$$R_0 = \frac{1}{2}(Q_0 + Q_1), \quad R_1 = \frac{1}{2}(Q_1 + Q_2)$$

$$S_0 = \frac{1}{2}(R_0 + R_1)$$

REKURZIVNÍ DĚLENÍ

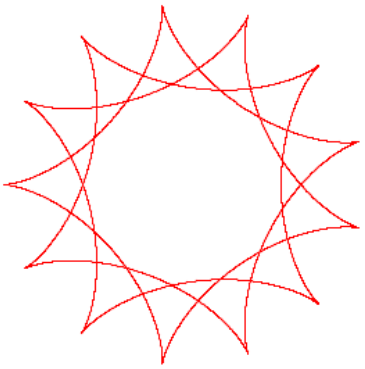
Křivka $C(t)$ pro $P_0, \dots, P_3$ $\Rightarrow$
- Křivka $C_a(t)$ pro P_0, Q_0, R_0, S_0
- Křivka $C_b(t)$ pro S_0, R_1, Q_2, P_3

ZASTAVENÍ REKURZE

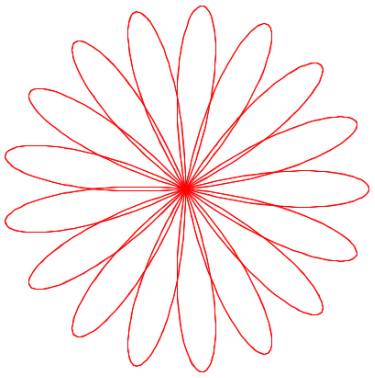
Pokud $|P_1 - P_0| + |P_2 - P_1| + |P_3 - P_2| \leq (1+\varepsilon)|P_3 - P_0|$,
křivku nahradíme úsečkou $P_0 P_1$.

Příklad 2 - Cyklické křivky

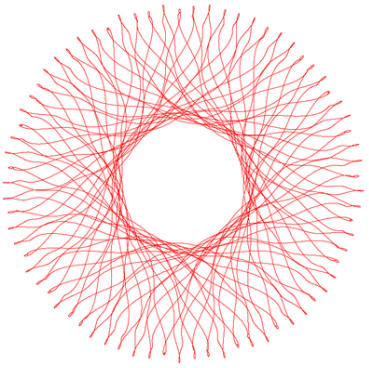
Existují i křivky založené na pohybu bodu po kružnici. Zajímavým zdrojem je např. Matematika křivek od Jarešové a Volfa (str. 43 - 50). Najdete tam nejen definici těchto křivek, ale také stručný historický kontext.



Hypocykloida



Rhodonea



Kombinace obou

Vytvořte program, který bude umět vizualizovat 2 až 3 cyklické křivky (vyberte si jakou chcete, nemusí to být právě ty křivky zde zobrazené). Vyzkoušejte různé hodnoty parametrů a třeba i složené některých těchto křivek dohromady. Může být zajímavé vykreslit si i pomocnou kružnici, na základě jejíhož pohybu křivka vznikla. Zamyslete se nad periodicitou těchto křivek. Pro některé hodnoty parametrů lze perioda spočítat, pro některé hodnoty může být opravdu velká, a pro některé hodnoty může být křivka neperiodická (nikdy se neuzavře). Křivku můžete obarvit - barevný přechod podle hodnoty parametru t.

K úloze odevzdejte i několik screenshotů a vysvětlující text (co je to za křivku, jaké hodnoty parametrů jste použili), nějakou zmínku o periodicitě křivky, případně i reference odkud jste čerpali informace.

5) Křivky I - 17. 3. 2021

Uživatelská práce s křivkami, zejména s kubickou Bezierovou

Ve vektorové grafice se pro reprezentaci tvarů velmi často používají kvadratické a kubické Bézierovy křivky. Např. formát SVG pro vektorovou grafiku podporuje úsečky, eliptické oblouky a Bézierovy křivky do stupně 3. Podobně je na tom kreslící program Inkscape pro vektorovou grafiku.

S křivkami manipulujeme pomocí bodů řídícího polygonu. U Bézierovy kubiky máme 4 řídící body P_0, P_1, P_2 a P_3. První a poslední, P_0 a P_3, leží na křivce, takže máme pod kontrolou odkud kam křivka vede. Vektor $P_1 - P_0$ určuje tečnu v bodě P_0 a vektor $P_3 - P_2$ určuje tečnu v bodě P_3. Manipulací P_1 a P_2 tedy ovládáme směr křivky v počátečním a koncovém bodě. Tato kontrola se hodí při spojování křivek, abychom mohli vytvořit složitější tvary. Kreslící program pak u řídících bodů nabízí různé varianty, např. ostrý zlom (corner, sousední tečné vektory nemusí být rovnoběžné) nebo hladké napojení (smooth, sousední tečné vektory jsou rovnoběžné) či symetrické napojení (sousední tečné vektory jsou rovnoběžné a navíc stejně dlouhé).

Obyčejná Bézierova kubika na všechno nestačí

- Pokud chceme křivku víc přimknout k nějakému bodu, můžeme k řídícím bodům přidat váhu a použít racionální variantu křivky. Zvyšováním váhy se křivka víc přimyká k řídícímu bodu. Váha lze nastavit i záporná (vyzkoušet).
- Pokud chceme křivkou popsat kružnici (nebo obecně kuželosečku), lze to udělat mnoha způsoby, ale ne obyčejným Bézierem. Jedna z možností je využití kvadratické racionální Bézierovy křivky. Tím ale popíšeme jen část oblouku. Chybějící část lze popsat s využitím negativní váhy u stejné křivky nebo použitím více (např. 3) křivek.

Výše zmíněné si lze vyzkoušet např. v programu Inkscape, Bézierovy křivky, NURBS a B-Spliny si lze vyzkoušet ve webovém nástroji NURBS Calculator. Kružnici pomocí křivky si lze vyzkoušet na http://demofox.org/bezquadrational.html.

Zajímavost k vizualizaci křivek: Knihovna OpenGL sama o sobě neumí kvadratické racionální Bézierovy křivky, nicméně místo práce v 3D prostoru lze pracovat v homogenních souřadnicích (4D). Až GPU provede perspektivní dělení, z obyčejné křivky ve 4D se stane racionální křivka ve 3D.

Pochopení algoritmu de Casteljau

http://www.malinc.se/m/DeCasteljauAndBezier.php

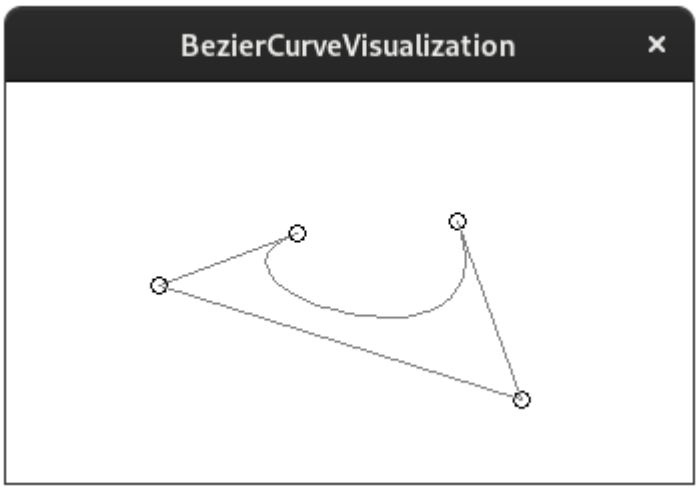
Pokud chceme určit pro zadaný parametr t bod na Bezierově křivce, nemusíme vyhodnocovat polynom přímo, lze na to jít přes hrany řídícího polygonu. Jednotlivé hrany “rozdělíme” v poměru t : (1-t). Spojíme body dělení, vznikne polygon který bude mít o jednu hranu méně. Hrany tohoto polygonu opět rozdělíme, a tak postupujeme dál, až zbyde poslední hrana. Tu rozdělíme ve stejném poměru. Bod v tomto posledním dělení je bod křivky pro zadanou hodnotu parametru t.

Algoritmus se používá nejen k výpočtu pozice “bodu na křivce”, ale také k **rozdělení jednoho segmentu křivky na dva** v bodě zadaném uživatelem. Viz např. práce s křivkou v Inkscape, kdy lze křivku editovat a dvojklikem na ni přidat bod. Některé body tohoto dělení totiž definují řídící polygony těchto dvou segmentů křivky (viz obr. na wikipedii).

Další využití algoritmu je pro **převod křivky na lomenou čáru** (např. pro potřebu vizualizace). Adaptivní dělení stylem rozděl-a-panuj. Čím hlouběji jsme v rekurzi, tím víc se přibližuje řídící polygon tvaru křivky. Lze stanovit míru podobnosti (součet stran lomené čáry řídícího polygonu / vzdálenost strany prvního a posledního řídícího bodu) a dělení zastavit, až je dosaženo uspokojivé prahové hodnoty.

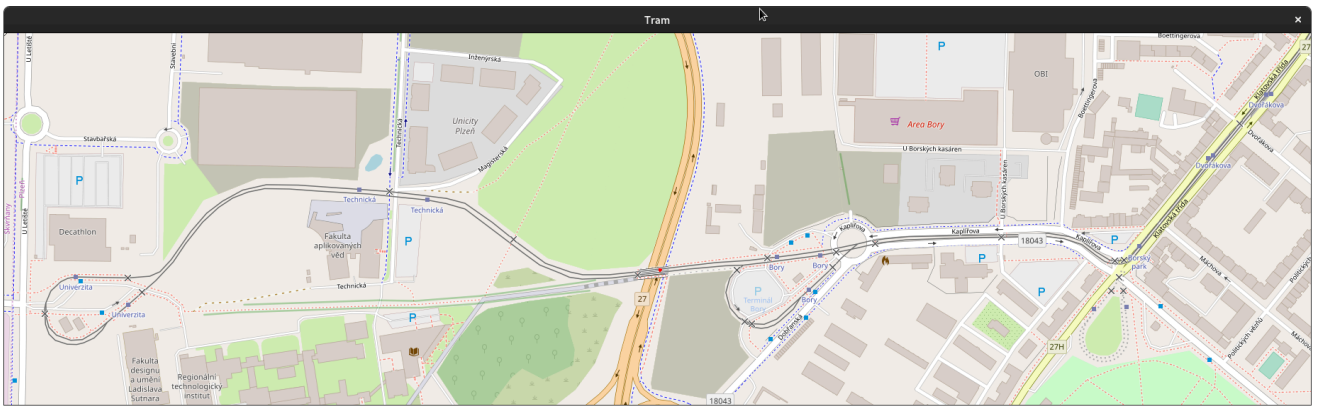
Příklad 1 - Implementace alg. de Casteljau pro vizualizaci Bézierovy křivky

V repository labs/05 máte základ aplikace pro vizualizaci Bézierovy křivky. Doplňte implementaci metody `drawBezierCurve()` o vlastní kód pro vykreslení křivky pomocí bodů nebo segmentů úsečky. Pro kontrolu se tam vykresluje křivka pomocí knihovní funkce.



Příklad 2 - Tramvaj až k univerzitě

Nedávno byla rozšířena trasa tramvaje č. 4 k Západočeské univerzitě. Vytvořte simulaci jízdy tramvaje jako pohyb bodu nebo složitějšího obrázku podél křivky. Jako podkladové pozadí použijte obrázek z Open Street Map. Koleje namodelujte pomocí sekvence (kubických) Bézierových křivek např. v programu Inkscape. Dokážete tramvaj přinutit, aby zastavila na zastávce a nabrala studenty?



Mapa a křivka jsou ke stažení stažení zde: Resources

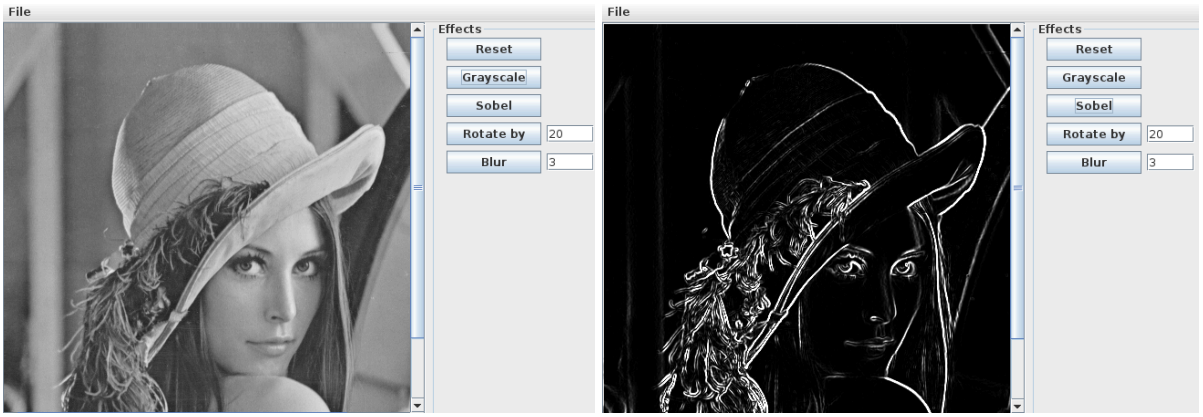
Další užitečné informace:

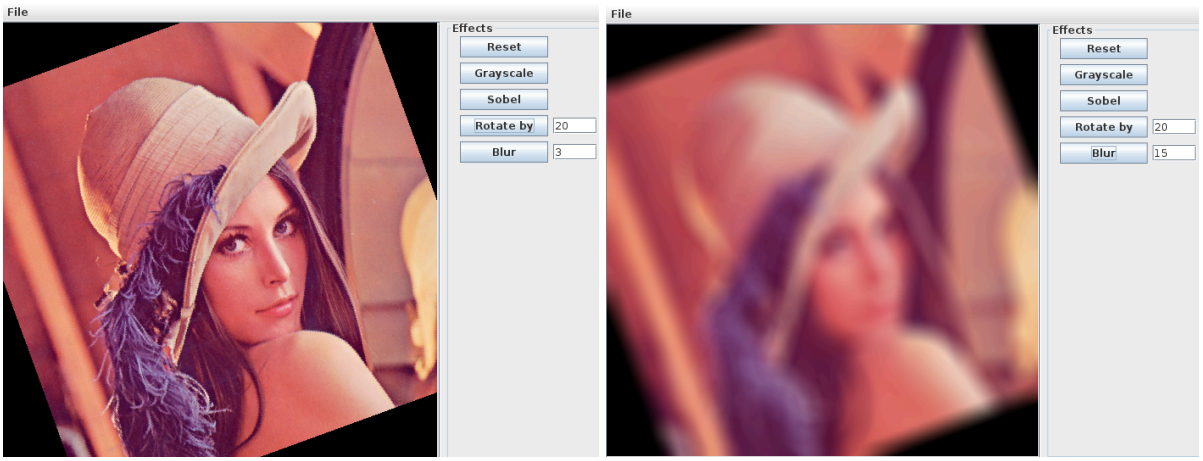
- https://docs.oracle.com/javase/tutorial/2d/overview/index.html
- SVG je textový soubor. Popis křivky z něj lze jednoduše vytáhnout a zpracovat textově. https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/Paths
- Obrázek na pozadí lze načíst pomocí třídy javax.imageio.ImageIO. Obrázek ale nenačítejte opakovaně každý snímek (v metodě paintComponent), ale jen jednou na začátku programu.

4) Úpravy obrazu 10. 3. 2021

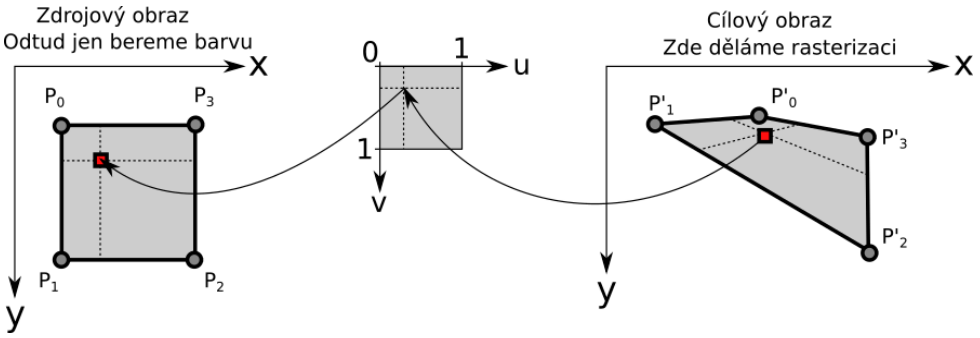
Ve svých repozitářích v adresáři `labs/04` najdete kostru aplikace (image-transform), do které stačí doimplementovat jednotlivé efekty. Jedná se o Maven projekt, který jde otevřít například v NetBeans IDE, ale build lze provést třeba i jen s pomocí Apache Maven (příkaz `mvn install`). Aplikace byla napsána pro Javu 8, ale měla by fungovat i v pozdějších verzích Javy.

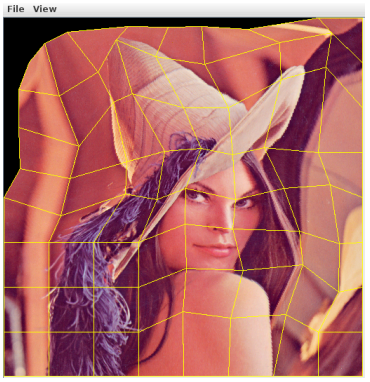
- **Převod barevného obrázku na odstíny šedi**
  Pro každý pixel načtěte jeho barvu (RGB), zprůměrujte (R+G+B) / 3, a zapište výsledek. Lidské oko je různě citlivé na různé barvy a tak lze použít i vážený součet, který toto zohledňuje.
- **Detekce hran v obraze - Sobelův operátor**
  Jde o aplikaci konvolučního kernelu velikosti 3x3 pro každý pixel. Sobelův operátor používá dva kernely, díky kterým dokáže odhadnout gradient funkce obrazu (gx, gy). Velikost gradientu, $(gx^2+gy^2)^{1/2}$, pak lze použít pro zvýraznění hran. Lze udělat i pokročilejší detekci hran, třeba Canny detektor.
- **Rotace obrazu kolem jeho středu**
  Projděte pixely cílového obrazu, souřadnice každého z nich promítněte do prostoru zdrojového obrazu, načtěte barvu a zapište ji do cílového obrazu.
- **Jednoduché rozmazání obrazu**
  Lze aplikovat konvoluční kernel, který pro každý pixel spočítá průměrnou hodnotu přes všechny pixely, které nejsou dál než uvedený poloměr (parametr rozmazání). Lze vypozorovat, že pro větší poloměry tato operace může trvat relativně dlouhou dobu. Lze to řešit efektivně přes Fourierovu transformaci (není třeba implementovat, jen o tom vědět). Obrázek převedeme do jiného prostoru, ve kterém jsou patrné frekvence, které se v obrázku vyskytují. Pracná konvoluce v prostoru obrazu zde funguje jako prosté násobení. Lze např. utlumit vysoké frekvence (detaily) a ponechat nízké frekvence. Po vynásobení provedeme inverzní transformaci a získáme obrázek po konvoluci. Pro Fourierovu transformaci existuje rychlý algoritmus, který běží v čase O(W * H * log W * log H) místo $O((W * H)^2)$. Lze dělat i dekonvoluci (zaostření), a různé další zajímavé věci. Další informace viz např. tento tutoriál.





- **Warping obrazu**
  Obraz se rozdělí na čtvercové dlaždice. S vrcholy této sítě lze manipulovat tažením myši a tím síť deformovat, jak je vidět na obrázku vpravo. Jde o to projít všechny deformované dlaždice, rasterizovat je nějakým jednoduchým způsobem (bounding box), a pro každý pixel v rasterizaci cílové dlaždice dohledat příslušný pixel v nedeformované dlaždici zdrojového (výchozího) obrazu. Mapování mezi cílovou a zdrojovou dlaždicí lze udělat vícero způsoby. Vhodný studijní materiál lze najít např. zde. Potřebné koeficienty pro transformaci lze snadno spočítat dosazením hodnot souřadnic U a V: (0,0), (0,1), (1,1), (1,0). Alternativně lze použít i algoritmus z přednášky, který deformaci provádí nejdříve podél osy X a potom podél osy Y.





3) Hexagonální dlaždice, Voroného diagram 3. 3. 2021

Zdrojové kódy kostry aplikace máte ve svém repository na GitLabu. Můžete ale nemusíte je použít.

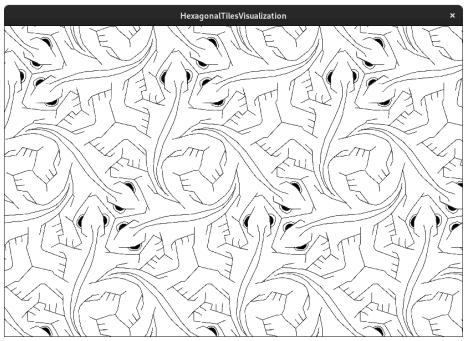
Příklad 1 - Hexagonální dlaždice

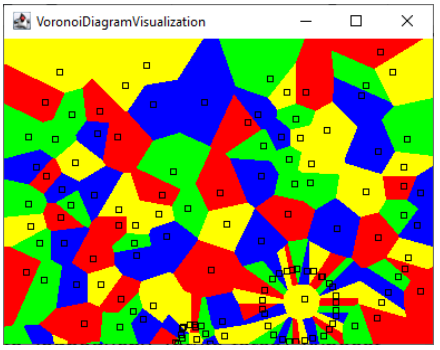
Vyzkoušejte si dláždění hexagonálními dlaždicemi. Nejdříve jednoduché (dlaždice stačí jen obarvit), poté s texturou. Použijte texturu Escherovy hexagonální dlaždice s motivem ještěrky. Vhodnou rotací dlaždic jde docílit toho, aby na sebe ještěrky navazovaly. Nápověda.

Příklad 2 - Voroného diagram

Pro zadané vstupní body rozdělíme prostor na regiony obsahující všechny body roviny, které jsou danému vstupnímu bodu nejblíž. Implementujte jednoduchý (brute-force) výpočet Voroného diagramu v diskrétním prostoru obrázku. Stačí projít všechny pixely a pro každý určit, ke kterému vstupnímu bodu to má tento pixel nejblíž. Pokuste se obarvit regiony tak, aby žádné dva sousední regiony neměly stejnou barvu. Aplikace by měla podporovat přidávání bodů, mazání bodů a posun bodů tažením myší.

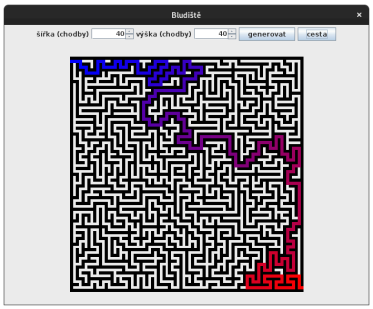
Poznámka: Čtyři barvy stačí pro jakýkoliv planární graf. Vyzkoušejte si barvení států čtyřmi barvami na stránkách Mathigon a projděte si zbytek tutoriálu (“skip to the next step” / “reveal all steps”).



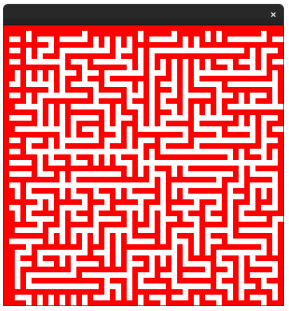


2) Dokončení bludišť a základů grafiky 23. 2. 2021

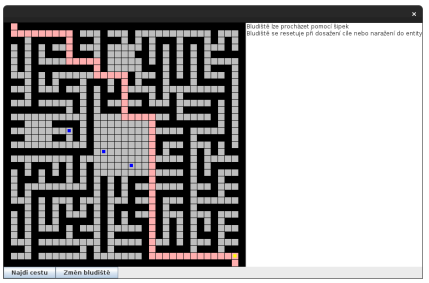
Z odevzdaných úloh vznikla malá Galerie bludišť a Game of Life.



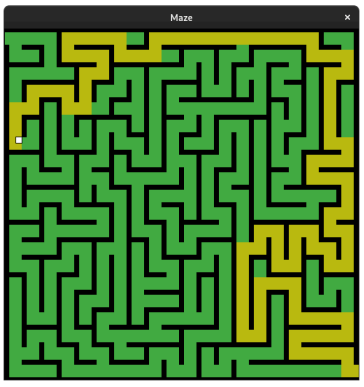
Ondřej Matura



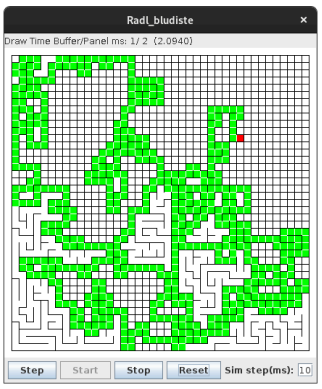
Jakub Vítek



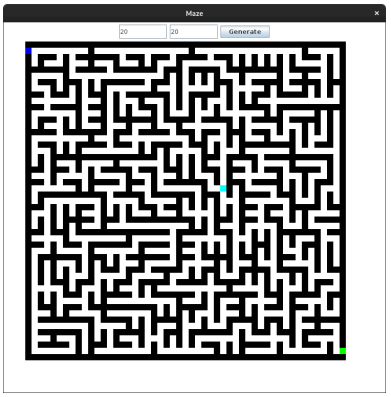
Martin Jakubašek



Jakub Hejman



Jan Rádl



Jan Kubice

Zde je odkaz na povedený vizualizátor od pana Rádla, který ukazuje, jak pracuje generátor bludiště.

Příklad 1 - Grafické okno

Vytvořte jednoduchý program, který zobrazí grafické okno s rozlišením 800x600 a obarví ho zeleně.

Nápověda (Java): metoda `main`, komponenty `JFrame` a `JPanel`, metody `setPreferredSize`, `setSize`, pack, `setVisible` a `paintComponent`.

Příklad 2 - Zobrazení čtverce na pozici myši

Rozšiřte předchozí úkol o pohyb čtverce nebo jiného objektu po obrazovce. Objekt se bude zobrazovat na pozici kurzoru myši. Vyzkoušejte si vyplnění objektu, vykreslení jeho hranice. Pokud si chcete vyzkoušet další primitiva, nahraďte čtverec nějakým složitějším obrazcem, například klasický “domeček jedním tahem”.

Nápověda: `MouseMotionListener`, `MouseAdapter`, `Color`, metody `mouseMoved`, `fillRect`, `drawRect`, `drawLine`.

Příklad 3 - Pravidelné překreslování plátna a rotace objektu

Rozšiřte Příklad 2 o rotaci objektu (čtverce nebo složitějšího obrazce) v čase. Seznamte se s třídou `Graphics2D` (stačí “natvrdo” přetypovat objekt typu `Graphics` na `Graphics2D`). Tato třída umožňuje kreslení složitějších objektů (Obdélníky, ovály, části oválů, úsečky, Bézierovy kvadratické a kubické

křivky, skládání křivek (Path2D) do složitějších útvarů, a lze definovat transformaci, která se na objekty má aplikovat). Pro rotaci lze využít metodu `rotate` třídy `Graphics2D`. Periodického překreslování lze dosáhnout s využitím třídy `javax.swing.Timer`. Nastavte mu periodu např 16 milisekund (odpovídá frekvenci 62.5 snímků za sekundu). Při vyvolání časové události změřte, kolik času uběhlo od minula (`System.nanoTime()` nebo `System.currentTimeMillis()`), na základě času a rychlosti rotace spočítejte výslednou rotaci v daný okamžik, proveďte vykreslení scény (metoda `paintImmediately` vizualizačního panelu) a nakonec proveďte synchronizaci pomocí metody `Toolkit.getDefaultToolkit().sync()`) Synchronizace pomáhá zabránit zhoršení snímkové frekvence na některých operačních systémech.

Úkol 4

Přidejte nějakou bitmapu do svého bludiště. Nápověda: Třída `ImageIO`.

Úkol 5

Vyzkoušejte si Game of Life a zkuste si tuto simulaci naprogramovat. Aplikujte Game of life na vaše skvělé bludiště nebo ho rozšiřte nějakým kreativním způsobem. Uvažujte periodické podmínky.

1) Úvodní hodina 16. 2. 2021

Osnova hodiny

- Úvodní informace o předmětu a podobě cvičení
- Za co bude zápočet a jak se bude bodovat aktivita na cvičení
- Oblasti témat a požadovaná podoba semestrální práce
- Přidělení repository na GitLabu KIV (proč a jak)
    - udělejte si registraci na gitlab.kiv.zcu.cz
    - repository dostanete přiděleno a inicializováno podle šablony
- Vyzkoušení si zanesení zdrojáku do repository (nějaký Hello World)
- Jednoduchý generátor bludiště (jen textový výstup)
    - Pozn: Kód předvedl cvičící a nasdílel do repozitářů na GitLabu
- Základní práce s grafikou (příště)
- Domácí úkoly (dobrovolná aktivita)
    - rozšiřte generátor bludiště do grafiky
    - přidejte možnost průchodu hráče bludištěm
    - nalezněte a zobrazte (nejkratší) cestu do cíle
    - místnosti a pohyblivé entity
    - měnící se mapa
    - příště: Galerie vytvořených bludišť
    - Mathigon - motivační lekce do tématu symetrie

Úvodní administrativa

Zpřístupnění GitLabu KIV podle návodu a přiřazení privátních repozitářů jednotlivým studentům. V tomto repozitáři budete uchovávat kód svých úloh, které jste vytvořili na cvičení, případně cokoliv dalšího, co s předmětem souvisí a nebude vám vadit, že do toho cvičící bude vidět.

Příklad 1 - Generování bludiště (forma tutoriálu - naprogramoval cvičící)

Napište program pro generování bludiště pomocí DFS. Výstup stačí udělat na standardní textový výstup. Zeď bludiště reprezentujte znakem '#' a volná políčka označte znakem '.' (tečka). Třeba něco takového:

```
####################
#.................#
###.#####.#.#######.#
#.#...#...#.#.#.#...#
#.#.#######.#.#.###.#
#...#...#...#.#.#...#
#.#.#.###.#.#.#.#.###
#.#.#.#.#.#.....#...#
#.#.#.#.###.###.#####
```

```
#.#.....#.#...#.....#
####################
```

Nabídka semestrálek

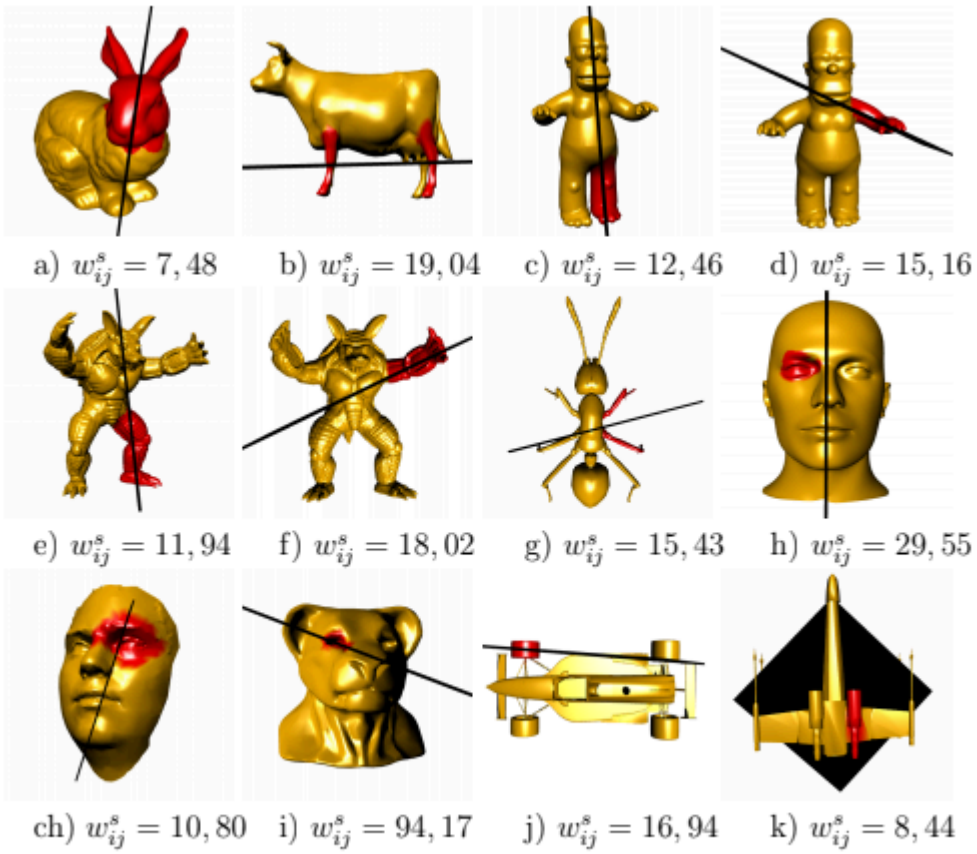
Detekce symetrie s pomocí point-cloud deskriptorů

Jde o to najít body nebo skupiny bodů s podobným deskriptorem. Deskriptory si bude nutné nastudovat (vlastní research ohledně state of the art). Toto má potenciál pro výpočet rovin symetrie, protože z těchto kandidátů (bodů nebo množin bodů) pak dokážeme nastartovat algoritmus hledání rovin symetrie. Codebase od Ing. Lukáše Hrudy (načítání a manipulace s objekty).

Lze se i zaměřit na hledání a zvýraznění lokálních extrémů (minima a maxima) na objektu, který je modelován jako point-cloud (mračno bodů bez informace o povrchové reprezentaci, bez trojúhelníkové sítě). Hledání lokální roviny, detekce hlavních směrů v datech.

Příklady některých datasetů pro detekci rovin symetrie:
https://sites.google.com/view/symcomp17/challenges/datasets?authuser=0

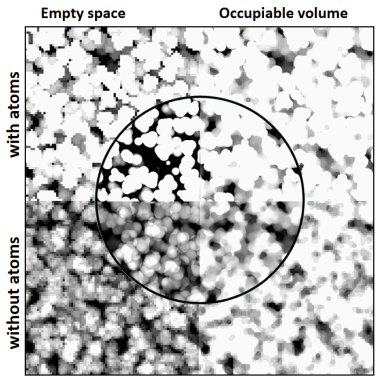
Jiný příklad z práce na téma symetrie

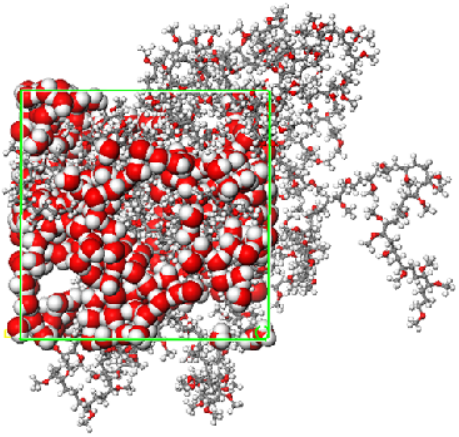


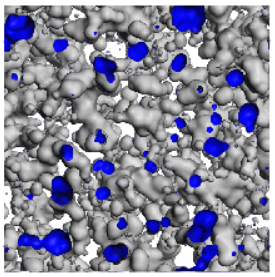
Obr. 3.3: Výsledná rovina symetrie s různou významností vstupních bodů pomocí rov.3.2. Červeně jsou označeny významné body. Hodnota w_{ij}^s udává váhu významného bodu.

Periodické nelineární Voroného diagramy (2D nebo 3D)

Motivace: Výpočet volného prostoru a dutin v granolových materiálech a molekulárních strukturách.







Free volume scan (polypropyleneglycol), focused with a raster 0.1 Å, R_P = 0,53 Å

Vizualizace stěn nelineárního Voroného diagramu

Stěna je geometricky hyperboloid ohraničený kuželosečkami. Cílem je udělat efektivní vizualizaci těchto stěn, nejlépe na GPU. Ilustrační obrázek stěn Voroného diagramu (vlevo) pochází z domácích stránek VDRC. Vizualizace hran (vpravo) pak z jedné bakalářské práce na ZČU (O. Šťáva).