|  |
| --- |
| OSTRAVSKÁ UNIVERZITA  PŘÍRODOVĚDECKÁ FAKULTA  KATEDRA INFORMATIKY A POČÍTAČŮ |
| Algoritmy kvantifikace propustnosti materiálu pro 3D rastr  DIPLOMOVÁ PRÁCE |
| Autor práce: Bc. Jan Mrógala  Vedoucí práce: Mgr. Alexej Kolcun, CSc. |
| 2022 |

|  |
| --- |
| UNIVERSITY OF OSTRAVA  FACULTY OF SCIENCE  DEPARTMENT OF INFORMATICS AND COMPUTERS |
| Material permeability quantification algorithms for 3D raster  MASTER THESIS |
| Author:  Bc. Jan Mrógala  Supervisor:  Mgr. Alexej Kolcun, CSc. |
| 2022 |

(Zadání vysokoškolské kvalifikační práce)

ABSTRAKT

Český text abstraktu

*Klíčová slova:*

*(klíčová slova vypsaná na řádku, oddělená od sebe čárkami)*

ABSTRACT

The text of the abstract.

*Keywords:*

čestné prohlášení

Já, níže podepsaný/á student/ka, tímto čestně prohlašuji, že text mnou odevzdané závěrečné práce v písemné podobě je totožný s textem závěrečné práce vloženým v databázi DIPL2.

Ostrava dne

………………………………

podpis studenta/ky

|  |
| --- |
| Poděkování |
| Prohlašuji, že předložená práce je mým původním autorským dílem, které jsem vypracoval/a samostatně. Veškerou literaturu a další zdroje, z nichž jsem při zpracování čerpal/a, v práci řádně cituji a jsou uvedeny v seznamu použité literatury.  V Ostravě dne . . . . . . . . . . . .  . . . . . . . . . . . . . . . . . .  (podpis) |

OBSAH

[ÚVOD 10](#_Toc135195259)

[1 Cíle práce 11](#_Toc135195260)

[2 Definice pojmů 11](#_Toc135195261)

[2.1 Propustnost 12](#_Toc135195262)

[2.2 Úplná propustnost 12](#_Toc135195263)

[2.3 Částečná propustnost 13](#_Toc135195264)

[2.4 Kritické místo propustnosti 13](#_Toc135195265)

[3 teorie grafů 14](#_Toc135195266)

[3.1 Cesta v grafu 14](#_Toc135195267)

[3.2 Propustnost grafu 15](#_Toc135195268)

[3.3 Úplná propustnost grafu 15](#_Toc135195269)

[3.4 Částečná propustnost grafu 16](#_Toc135195270)

[3.5 Kritické místo propustnosti grafu 16](#_Toc135195271)

[3.6 Dijkstrův algoritmus 17](#_Toc135195272)

[3.7 Maximální tok a minimální řez 18](#_Toc135195273)

[3.7.1 Maximální tok 19](#_Toc135195274)

[3.7.2 Minimální řez 20](#_Toc135195275)

[4 Algoritmy hledání cesty v bludišti 21](#_Toc135195276)

[4.1 Vyplňování slepých konců (breadth-first search) 22](#_Toc135195277)

[4.2 Prohledávání do hloubky (depth-first search) 23](#_Toc135195278)

[4.3 Podobnost mezi algoritmy vyplňování a řešení bludiště 23](#_Toc135195279)

[5 Kvalita propustnosti 24](#_Toc135195280)

[5.1 Matematická morfologie 24](#_Toc135195281)

[5.1.1 Dilatace 25](#_Toc135195282)

[5.1.2 Eroze 25](#_Toc135195283)

[6 Další přístup k řešení propustnosti materiálu 27](#_Toc135195284)

[7 Otázka grafické reprezentace 28](#_Toc135195285)

[8 analýza kritických míst v materiálu 30](#_Toc135195286)

[8.1 Další možné řešení problému 30](#_Toc135195287)

[9 Algoritmus pro analýzu permeability 31](#_Toc135195288)

[9.1 Optimalizace algoritmu 36](#_Toc135195289)

[9.2 Zobecnění pro 3D 37](#_Toc135195290)

[9.3 Možnost pokračování a zlepšení algoritmu 38](#_Toc135195291)

[ZÁVĚR 39](#_Toc135195292)

[RESUMÉ 40](#_Toc135195293)

[SUMMARY 41](#_Toc135195294)

[SEZNAM POUŽITÉ LITERATURY 42](#_Toc135195295)

[SEZNAM OBRÁZKŮ 44](#_Toc135195296)

[SEZNAM TABULEK 45](#_Toc135195297)

[SEZNAM PŘÍLOH 46](#_Toc135195298)

ÚVOD

Tato práce navazuje na mou bakalářskou práci, která se věnovala algoritmům pro vyplňování 3D rastrového obrazu [1]. Předchozí práce se zaměřila na popis a porovnání různých vyplňovacích algoritmů, zatímco tato práce se soustředí na jejich využití při řešení konkrétního problému – permeability materiálu.

Tento problém je velmi zajímavý, protože právě pomocí vyplňovacích algoritmů můžeme detekovat permeabilitu materiálu. Konkrétněji, můžeme posoudit, zda je materiál v určitém stavu propustný, či nikoliv.

Pro praktickou aplikaci algoritmů je podobně jako v předešlé práci zvolena sada CT snímků, znázorňujících 3D pórovitý propustný materiál.

1. Cíle práce

Hlavním cílem této práce je nalezení způsobu, jak vyhodnotit míru propustnosti permeabilního materiálu. Důležitou roli pro vyřešení tohoto problému hrají vyplňovací algoritmy.

Vedlejším stanoveným cílem je porovnání algoritmů pro řešení bludiště a vyplňovacích algoritmů.

Pro aplikaci algoritmů je zvolen program Octave, ve kterém budou zdrojové kódy pro algoritmy napsány.

1. Definice pojmů

Pro pochopení problému permeability materiálu a jaký problém se konkrétně snažíme vyřešit si nejdříve nadefinujeme několik pojmů.

* Pro zjednodušení se budeme zatím bavit pouze o 2D oblasti.
* Pro 3D oblast se základní myšlenky těchto pojmů nemění.
* Nejmenším obrazovým bodem, se kterým pracujeme a bereme jej v úvahu, je pixel.
  1. Propustnost

Chápejme propustnost materiálu v našem případě jako cestu z bodu **A** do bodu **B** přes oblast **C**. Pro ujasnění chápejme tyto body jako pixely v rastrové mřížce, zde je takto znázorňujeme pouze pro zjednodušení.

A

B

C

Obrázek 1- znázornění propustnosti materiálu

* 1. Úplná propustnost

Pro úplnou propustnost platí, že z bodu **A** existuje v dané oblasti cesta jak do bodu **B,** tak do bodu **C**.

A

B

C

Obrázek 2- znázornění úplné propustnosti

* 1. Částečná propustnost

Mějme hlavní oblast **G** obsahující podoblasti **E** a **F**, které nejsou propojeny. Částečná propustnost nastává, když z bodu **A** existuje cesta do bodu **B**, ovšem nikoli do bodu **D**.

G

A

B

C

D

E

F

Obrázek 3- znázornění částečné propustnosti

* 1. Kritické místo propustnosti

Pokud v oblasti **C** existuje místo **D**, které po uzavření znemožňuje propustnost z bodu **A** do bodu **B**, jedná se o kritické místo propustnosti v dané oblasti.

A

B

C

D

Obrázek 4- znázornění kritického místa propustnosti D

I přes to, že se tato práce na obor teorie grafů nezaměřuje, je dobré si zde uvést širší souvislosti a uvědomit určité paralely, které jsou přítomny jak v problematice teorie grafů, tak hledání propustnosti v rastrovém obraze (ať už 2D či 3D).

1. teorie grafů

Otázku, jak se dostat z jednoho bodu do druhého do jisté míry v této práci řešíme. Více než konkrétní cesta, nás ovšem zajímá samotná existence cesty neboli propustnost. Podívejme se ale, jakým způsobem k tomuto problému přistupuje teorie grafů.

V teorii grafů nalézt několik algoritmů, které se zaobírají buď nalezením cesty z jednoho vrcholu do druhého (Dijkstrův algoritmus), nebo největším tokem sítí (maximální tok). Tyto problémy jsou analogické k našemu řešení problému hledání permeability v obrazovém rastru. V úvahu zde ovšem berme to, že se jedná o odlišný kontext: Práce s rastrovou mřížkou a odpovídajícími algoritmy, které pracují s pixely, oproti grafovému prostoru se svými uzly a hranami.

* 1. Cesta v grafu

Když se bavíme o pojmu cesta v grafu, myslíme tím podgraf v grafu G, který je vůči němu izomorfní. Někdy můžeme za cestu P v grafu G považovat posloupnost vrcholů   
(x0, x1, …, xk) pro které platí, že . O cestě P říkáme, že vede mezi vrcholy x0 a xk. Z tohoto důvodu ji občas značíme jako x0Pxk. Díky této definici můžeme jednoduše značit podcesty pro :

[4]

* 1. Propustnost grafu

Stejně, jako jsme si nadefinovali druhy propustnosti pro 2D rastrový obraz, tak si je nadefinujeme pro grafový kontext.

Existuje-li v grafu G sled mezi uzly x a y, kde x a y odpovídají krajním uzlům grafu, můžeme graf G považovat za propustný. Jinak řečeno, v grafu existuje cesta mezi dvěma vrcholy.

V našem příkladu můžeme vidět, že existuje sled uzlů mezi vrcholy x0 a x3

A picture containing barbell, circle, sport

Description automatically generated

Obrázek 5- příklad propustného grafu

V rastrové mřížce chápeme krajní body jako body, jejichž souřadnice jsou extrémní na ose X. Například veškeré body, jejichž souřadnice mají hodnotu X nulovou, nebo maximální vůči šířce obrazu. Grafy takto jednoznačně značeny nejsou, takže zda je nějaký vrchol krajní, či nikoliv, musíme posoudit buď podle zraku, nebo algoritmicky při případném převodu z rastru na grafovou síť.

* 1. Úplná propustnost grafu

Existuje-li v grafu G sled mezi uzly x0 x3 a x0 x4, je graf plně propustný.

A picture containing circle, line

Description automatically generated

Obrázek 6- příklad úplné propustnosti grafu

* 1. Částečná propustnost grafu

Tato situace nastává, když máme nesouvislý graf G. Z vrcholu x0 existuje cesta do x3, ale nikoliv do x7.

A picture containing circle, line

Description automatically generated

Obrázek 7- příklad částečné propustnosti grafu

V kontextu grafů se částečná propustnost nedá nalézt v souvislém grafu. Pouze v souvislém.

* 1. Kritické místo propustnosti grafu

U grafů kritickým místem myslíme vrchol, po jehož odstranění dojde k přerušení propustnosti. Jinak řečeno přestane existovat cesta mezi krajními vrcholy grafu G. V našem příkladě představuje kritické místo vrchol x4.

A picture containing circle, line, diagram, design

Description automatically generated

Obrázek 8- příklad kritického místa propustnosti grafu

* 1. Dijkstrův algoritmus

Dijkstrův algoritmus je algoritmus pro hledání nejkratší cesty v grafu s pozitivně ohodnocenými hranami. Byl vytvořen nizozemským počítačovým vědcem Edsgerem Dijkstrou v roce 1956 a publikován v roce 1959.

Algoritmus funguje tak, že přiřadí dočasné vzdálenosti ke všem vrcholům v grafu a poté iterativně aktualizuje tyto vzdálenosti, dokud nenajde nejkratší cestu ze zdrojového vrcholu ke všem ostatním vrcholům. [3]

Pseudokód pro Dijkstrův algoritmus:

funkce Dijkstra(*Graf*, *zdroj, cíl*):

1. pro každý vrchol grafu v:
   1. vzdálenost[v] = NEKONEČNO
   2. předchozí[*v*] = NEDEF
   3. přidej v do Q
2. vzdálenost[*zdroj*] = 0
3. dokud Q není prázdné:
   1. u = vrchol z Q, který má nejmenší vzdálenost
   2. odstraň u z Q
   3. pokud u = cíl:
      1. S = prázdný seznam
      2. pokud předchozí[u] je DEF nebo u = zdroj:
         1. dokud u je DEF:
            1. vlož u na začátek S
            2. u = předchozí[u]
      3. ukonči cyklus
   4. pro každého souseda v od u z Q:
      1. temp = vzdálenost[u] + Graf.hrana(u, v)
      2. pokud temp < vzdálenost[v]:
         1. vzdálenost[v] = temp
         2. předchozí[v] = u
4. vrať seznam S

převzato z <https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm>

Standardně by Dijkstrův algoritmus vracel všechny vrcholy a jejich nejkratší cesty, ovšem pokud nám stačí nalézt jedinou cestu ze zdroje do cíle, můžeme hlavní cyklus ukončit předčasně, a to tak, že zkontrolujeme, zda zkoumaný vrchol odpovídá vrcholu cílovému (podmínka 3c). Následně zpětně projdeme předešlými hranami, které uložíme do seznamu S, což nám dává celou cestu ze zdroje do cíle.

* 1. Maximální tok a minimální řez

Co se naší analogie týče, v tuhle chvíli již máme způsob, jak ověřit propustnost grafu pomocí Dijkstrova algoritmu. Následně nám zbývá nalezení cesty s největším průtokem. V kontextu rastru zde chápejme propustnou cestu s nejmenším kritickým místem.

Pracujeme se sítí *S(G, s, t, c)*, která se skládá z následujících prvků:

* **G** – orientovaný graf *G = (V, E)*, skládající se z vrcholů *V* a hran *E*
* **s** – počáteční vrchol, nazýván zdroj (source)
* **t** – konečný vrchol, nazýván spotřebič (target)
* **c** – kapacita hran, *c(E)* pro konkrétní hranu daná funkcí *c: E → R+*

kromě kapacity má každý vrchol také hodnotu toku, ta se znázorňuje funkcí: *f: E → R+*. Zároveň musí platit následující 2 pravidla:



První pravidlo nám říká, že tok mezi vrcholy nemůže být záporný, nebo větší, než je kapacita hrany. Druhé pravidlo se také nazývá zákon zachování toku, protože tok v hranách ani nepřibývá, ani neubývá.

Číslo *f(e)* se nazývá tok po hraně *e*. Ve výsledku značíme hrany tímto způsobem: *x/y*, kde x je hodnota toku *f(e)* a y kapacita hrany *c(e)*. Tok přes hranu nemůže přesáhnout její kapacitu.

Řezem nazýváme množinu orientovaných hran , kde a je doplňkem.(s, t) – řezem myslíme takový řez, který odděluje zdroj od spotřebiče. [4]

A picture containing watch

Description automatically generated

Obrázek 9- příklad sítě, převzato z [4]

Věta o maximálním toku a minimálním řezu: Když existuje maximální (s, t) – tok, potom platí:

Jednoduše řečeno, maximální velikost toku v síti je dána kapacitou minimálního řezu. [4]

* + 1. Maximální tok

Maximální tok v síti představuje maximální množství "nákladu" nebo "informace", které může projít ze zdroje do cíle přes danou síť. Tento tok může být omezen kapacitou jednotlivých hran v síti. Cílem je najít největší tok, který může být přepraven z jednoho bodu do druhého.

Máme-li neorientovaný graf, nahrazujeme každou jeho hranu dvojicí orientovaných hran v opačných směrech, se stejnou hodnotou kapacity na obou.

Z věty o maximálním toku a minimálním řezu vychází Ford-Fulkersonův algoritmus, což je jeden ze základních algoritmů vylepšující cesty. Je to algoritmus pro nalezení maximálního toku a to tak, že začínáme s nulovým tokem a postupně hledáme vylepšující cesty, ve kterých tok zvětšujeme. Když už nelze tok dále zvětšit, nalezli jsme maximální tok. [4]

Pseudokód pro Ford-Fulkersonův algoritmus:

funkce Ford-Fulkerson:

1. f = 0;
2. dokud existuje vylepšující cesta P z s do t:
   1. vylepši tok f podél cesty P
   2. f++
3. vrať f

Mějme síť *a)*, na které znázorníme průběh algoritmu. Začínáme s nulovým tokem a postupně vylepšujeme cesty. Každý obrázek ukazuje stav po navýšení toku podél červenou barvou označené cesty. [4]

A picture containing diagram

Description automatically generated

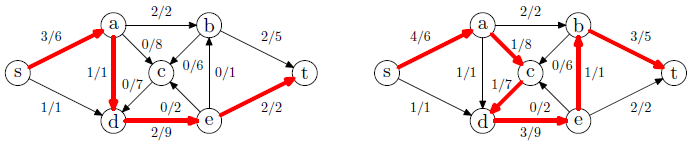
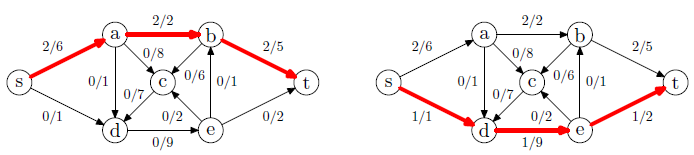
a)

d)

b)

c)

e)



Obrázek 10- Příklad postupu pro nalezení maximálního toku, převzato z [8]

Z příkladu můžeme vidět, že po čtvrtém kroku – e), již neexistuje další vylepšující cesta. Hodnota maximálního toku je v tomto případě 5.

* + 1. Minimální řez

Minimální řez je nejmenší množina hran, kterou je nutné odstranit ze sítě, aby bylo dosaženo omezení maximálního toku. Odděluje zdrojový uzel od cílového uzlu a zabraňuje toku mezi nimi. Minimální řez může být také definován jako hranová množina, která rozděluje síť na dvě disjunktní podmnožiny, z nichž jedna obsahuje zdrojový uzel a druhá obsahuje cílový uzel. [4]

1. Algoritmy hledání cesty v bludišti

Zásadním pojmem, který jsme si definovali, bylo kritické místo propustnosti. Abychom mohli analyzovat vlastnosti tohoto místa, musíme nejdříve nalézt cestu mezi body A B. V našem případě využíváme vyplňovací algoritmy, ovšem můžeme na to taktéž nahlížet jako na problém hledání cesty bludištěm.

Za vyřešení bludiště se dá považovat nalezení cesty z počáteční buňky bludiště do cílové buňky.

Základní rozdělení těchto algoritmů je podle toho, zda řešící algoritmus pracuje s celým bludištěm, nebo pouze „simuluje“ průchod bludištěm za určitých pravidel.

Bludiště se skládá z následujících prvků:

* zeď
* buňka
* cesta
* trasa

základní prvek bludiště je buňka, která se rozděluje na čtyři typy: izolovaná, slepý konec, křižovatka a buňka, která je součástí cesty.

Cesta je souvislá posloupnost propojených buněk mezi křižovatkami a slepými konci.

Trasou se rozumí posloupnost po sobě jdoucích buněk mezi dvěma libovolnými buňkami.

Bludiště zde chápeme jako graf. K vytvoření tohoto grafu reprezentující bludiště nám postačují právě předtím zmíněné základní prvky bludiště, kde cesty odpovídají hranám a slepé konce a křižovatky představují vrcholy v grafu. [5]

A picture containing diagram, line, plan, rectangle

Description automatically generated

Obrázek 11- bludiště reprezentované jako graf, převzato z [5]

* 1. Vyplňování slepých konců (breadth-first search)

Algoritmus pracuje tak, že začne od vstupu do bludiště a postupně prochází jednotlivé chodby. Algoritmus označuje (vyplňuje) slepé cesty, které nevedou k cíli. Následně se vrací na poslední navštívenou křižovatku, ze které se na slepou cestu vydal. Poté pokračuje po jiné nevyzkoušené cestě a opakuje tento postup, dokud nenajde cestu k východu z bludiště. [5]

Jedná se o jednoduchý algoritmus, za jehož největší nevýhodu můžeme pokládat nutnost prohledat celé bludiště.

U tohoto algoritmu můžeme pozorovat podobnost s řádkovým semínkovým vyplňováním právě ve využití zapamatování křižovatky (kritického pixelu), ke kterému se algoritmus vrací. Můžeme porovnat pseudokódy těchto dvou algoritmů:

Pseudokód pro řádkové semínkové vyplňování:

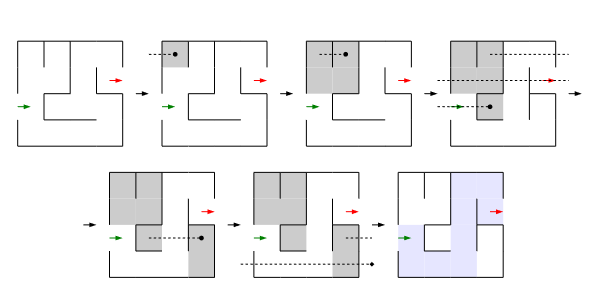
funkce vyplň (x, y, barva\_vyplnění, barva\_hranice):

1. vytvoř nový zásobník
2. přidej do zásobníku pixel[x, y]
3. dokud není zásobník prázdný, proveď:
   1. vyjmi z vrcholu zásobníku pixel
   2. dokud pixel[x-1, y] leží uvnitř oblasti, proveď:
      1. x--
   3. přidej do zásobníku horní nebo spodní pixel, pokud se nachází v oblasti, je nevybarvený a levý sousední pixel není hraniční
   4. dokud se pixel nachází uvnitř oblasti:
      1. vybarvi pixel[x, y]
      2. přidej do zásobníku horní nebo spodní pixel, pokud se nachází uvnitř oblasti, není vybarvený a levý sousední pixel je hraniční
      3. x++

Pseudokód pro breadth-first search algoritmus:

**funkce** BFS(*G*, *root*) **is**

1. let *Q* be a queue
2. label *root* as explored
3. *Q*.enqueue(*root*)
4. **while** *Q* is not empty **do**
   1. *v* := *Q*.dequeue()
   2. **if** *v* is the goal **then**
      1. return v
   3. **for all** edges from *v* to *w* **in** *G*.adjacentEdges(*v*) **do**
      1. **if** *w* is not labeled as explored **then**
         1. label *w* as explored
         2. *w*.parent := *v*
         3. *Q*.enqueue(*w*)



Obrázek 12- průběh algoritmu slepých konců, převzato z [5]

* 1. Prohledávání do hloubky (depth-first search)

Prohledávání do hloubky prohledává graf postupně podle jednotlivých větví, tedy prochází do hloubky jednu větev, dokud nenarazí na uzel bez nezpracovaných sousedů. Poté se vrací zpět k poslednímu uzlu, u kterého existuje nezpracovaný soused a opakuje tento proces znovu.

Prohledávání do hloubky nemusí najít nejkratší cestu k cílovému uzlu, ale může najít řešení rychleji než vyplňování slepých konců, pokud je cílový uzel hlouběji v grafu. [5]

Diagram

Description automatically generated

Obrázek 13- Postup prohledávání do hloubky, převzato z

<https://www.simplilearn.com/tutorials/data-structure-tutorial/dfs-algorithm>

* 1. Podobnost mezi algoritmy vyplňování a řešení bludiště

Oba algoritmy, řešení bludiště i výplně vybrané oblasti, zahrnují průchod dvourozměrným prostorem a prozkoumávání sousedních buněk nebo pixelů. Oba také zahrnují použití datových struktur, jako jsou zásobníky nebo fronty, pro udržování navštívených buněk nebo pixelů a pro řízení procesu prohledávání nebo výplně.

Existují však také některé klíčové rozdíly mezi algoritmy řešení bludiště a výplně vybrané oblasti. Algoritmy řešení bludiště se obvykle zaměřují na nalezení jediné cesty z počátečního bodu do koncového bodu, zatímco algoritmy výplně vybrané oblasti se zaměřují na vyplnění celé oblasti zájmu.

1. Kvalita propustnosti

Pod otázkou kvality propustnosti permeabilního materiálu si představíme především výskyt kritických míst, a jak silná je propustnost, respektive velikost kritického místa. Pro vyřešení těchto problémů využijeme přístupy využívané v matematické morfologii.

* 1. Matematická morfologie

Matematická morfologie je matematická teorie, která se zabývá analýzou geometrických transformací, jako jsou eroze, dilatace, otevření a uzavření, aplikovaných na obrazy nebo jiné geometrické objekty. Tato teorie byla poprvé navržena Georgesem Matheronem v roce 1962 a později rozvinuta Jeanem Serra v 70. letech 20. století.

Základní myšlenkou matematické morfologie je aplikace geometrických transformací na obrazová data za účelem získání informací o struktuře, tvaru a topologii objektů v obraze. Eroze a dilatace jsou základními operacemi matematické morfologie, které slouží k úpravě tvaru objektů v obraze.

Morfologické operace vychází z myšlenky, že binární obraz můžeme vyjádřit jako množinu 2D bodů. Této množině říkejme X. Každý bod v této množině reprezentuje pixel v obraze s hodnotou jedna. Jako příklad uveďme následující množinu X = {(1, 1), (1, 2), (2, 1),  
(2, 2)}.

Transformace obrazu je relace mezi dvěma takovými množinami bodů, kde ta druhá je obvykle menší a nazývá se strukturní element. Transformace můžeme provádět jak na 2D tak i 3D obraze. [6]

A picture containing shape

Description automatically generated

Obrázek 14 - příklad strukturních elementů, převzato z [6]

* + 1. Dilatace

Pro tuto práci je dilatace nejužitečnější operace matematické morfologie. Za pomocí vektorového součtu mezi množinami X a B expandujeme objekty v obraze. [6]   
Předpis dilatace je následující:



Obrázek 15- Příklad dilatace, převzato z [6]

* + 1. Eroze

Eroze je duální operace k dilataci. Dilatace a eroze nejsou invertovatelné, tzn. eroze se nedá považovat za opak dilatace. Eroze způsobuje ztenčování objektů v obraze. [6]   
Předpis pro erozi:

B = {}

A picture containing text, clipart

Description automatically generated

Obrázek 16- Příklad eroze, převzato z [6]

1. Další přístup k řešení propustnosti materiálu

Problém propustnosti materiálu je ve vědecké komunitě většinou řešen odlišnými způsoby. Konkrétně je nejčastější laboratorní, či fyzikální přístup. Případně modelování chování proudění kapaliny v propustném materiále.

Jedna z prací, kterou můžeme na toto téma nalézt, kombinuje laboratorní přístup s modelováním.

Uveďme si zde, o čem se vědecký článek [7] zabývá:

Autoři kombinovali pokročilé techniky charakterizace a modelování k analýze mikrostruktury materiálu porézních pásek, litých ve velmi tenké vrstvě ve tvaru desky z keramické kaše, a jejich vlivu na propustnost. Byly vyrobeny čtyři vzorky s různým obsahem porogenu pomocí metody páskového lití. Následné vypalování vyrobených zelených pásek umožnilo získat otevřeně porézní struktury s porozitou v rozmezí 45–50%.

Byla provedena kvantitativní analýza obrazu 3D mikro-počítačové tomografie a byly získány podrobné charakteristiky porézní mikrostruktury. Na základě získaných dat byl vyvinut reprezentativní model otevřeně porézní mikrostruktury. Metoda konečných objemů byla použita k výpočtu propustnosti různých scénářů mikrostruktury.

Získané výsledky ukazují, že porozita, průměrná velikost pórů a konstriktivita přímo ovlivňují vlastnosti proudění kapalin. Významné změny v konstriktivitě a výsledné propustnosti mohou být způsobeny významným přídavkem porogenu, pro který se mění propojení pórů a tvorba (nebo její absence) volných cest ovlivňuje proudění kapalin. [7]

Je tedy vidět, že přístupů pro zjištění permeability materiálu je více. Tato metoda je praktičtější a případně užitečnější pro použití v praxi, ovšem vyžaduje nestandardní výbavu jak softwarovou, tak laboratorní.

1. Otázka grafické reprezentace

Jelikož tato práce manipuluje s obrazovými daty, bylo by vhodné prozkoumat možnosti, jak grafický výstup, v podobě vyplněné oblasti, rozumně uživateli zobrazit. Ideální by byl interaktivní 3D pohled na extrahovanou cestu materiálem. Do jisté míry řešila podobnou tématiku Petra Grossmanová ve své práci [8].

Cílem práce je navržení vhodného počítačového modelu pro reprezentaci 3D prostoru. Autorka popisuje vytvoření aplikace, ve které by se měl být uživatel schopný pohybovat a orientovat.

V zásadě téměř každé řešení pro 3D reprezentaci vede k vytvoření sítě trojúhelníkových polygonů. Nejzajímavějším z popisovaných přístupů je tzv. Marching cubes.

Tento velmi známý algoritmus s počátkem v roce 1987 je hojně využíván pro rekonstrukci povrchu z 3D objemových dat. Pro svou práci algoritmus využívá data strukturovaná v mřížce. Body mřížky jsou ohodnoceny binárně čísly 0 a 1, kde 0 do tělesa nepatří a 1 patří. Pro každou osmici bodů v mřížce poté nastavuje algoritmus hodnotu, která určuje konfiguraci triangulací. Každá krychle má 15 různých kombinací. [8]

Diagram

Description automatically generated

Obrázek 17- Triangulační tabulka, převzato z [8]

Po ohodnocení všech osmi bodů v mřížce, kdy bod do krychle buď patří, nebo nepatří, se přechází k následující krychli.

K zjištění souřadnic vrcholů trojúhelníků se využívá lineární interpolace. Následně jsou vypočítány normály v těchto vrcholech. Postup je takový, že se nejdříve vypočítá hodnota normály pro každý trojúhelník, a následně je normála pro každý bod vypočítána jako průměr normál nejbližších dotýkajících se trojúhelníků. Po zjištění normál můžeme aplikovat libovolné stínování, jako je Gouraudovo, nebo konstantní.

Je třeba si uvědomit slabá místa tohoto algoritmu. První problém je značně velký počet vytvořených trojúhelníků, neboť každá krychle může obsahovat až 4 trojúhelníky. Druhým problémem je vytváření děr, za což může nejednoznačnost při zvolení triangulace sousedních krychlí. Pro první problém existuje několik řešení, které počet polygonů dokáže snížit, a přesto zachovat původní tvar objektu. Problém děr řešíme jejich zalepením, případně odstraněním pravidla reflexivní symetrie.

Existují ještě další možnosti reprezentace 3D obrazu, jako je reprezentace pomocí ploch, kde se využívá řídících bodů a bázových funkcí. Náleží zde například NURBS plochy, nebo Beziérovy plochy. Tyto metody se odlišují hlavně tím, jak velký vliv má změna jednoho bodu na tvar okolní plochy. [8]

1. analýza kritických míst v materiálu

Způsoby, jak vyhledat kritická místa propustnosti můžeme vymyslet více. Nejjednodušší algoritmus, který by se dal použít, by mohl při vyplňování zároveň prohledávat okolní čtyři sousední pixely zrovna vyplňovaného pixelového bodu. Narážíme zde ovšem na více problémů. Ten první je, že by nestačilo prosté kontrolování sousedního okolí pixelu, ale bylo by nutné prohledávat větší oblast, jelikož nemůžeme předpokládat, že v materiálu je kritické místo propustnosti o velikosti jednoho pixelu. Další problematická situace nastává při určité konfiguraci hraniční oblasti.

Velký problém, na který narážíme u jednoduchého prohledávání sousedních pixelů je ten, že existují případy, kdy je kritické místo těžko detekovatelné. Pokud bychom vyhledávali pouze do čtyř směrů, tak nedetekujeme kritické místo mezi diagonálně protilehlými pixely. Tento případ je tedy důvodem, proč využití této jednoduché metody není dobrý nápad. Další způsoby, jak prohledávat okolí jednoho bodu, jistě existují, ovšem komplexnost začíná růst jak z implementačního hlediska, tak výpočetního. Přesuneme se tedy na další možnost řešení daného problému.

* 1. Další možné řešení problému

Pokud nejsme reálně schopni využít prohledávání sousedního okolí kolem pixelového bodu, což by byl nejjednodušší přístup, vymyslíme jiný.

Následující přístup využívá teorie matematické morfologie, a to konkrétně dilatace. Myšlenka spočívá v tom, že místo toho, abychom zkoumali jednotlivé pixely, zda se nejedná o kritické místa, raději postupně dilatujeme hraniční body. Tento přístup je méně výhodný v tom, že bude nutné opakované vyplňování zkoumané oblasti, abychom zjistili, zda je materiál stále propustný. Tato operace rozšiřování bude opakována tak dlouho, až dojde k uzavření. Jakmile zjistíme, že již nelze materiálem prostoupit na jeho druhý konec, víme, že jsme zaplnili místo, které je kritické.

Algoritmus pro analýzu propustnosti materiálu si nejdříve nadefinujeme pro 2D, následně pro 3D

1. Algoritmus pro analýzu permeability

Cílem algoritmu je nalézt následující parametry propustnosti:

* velikost kritického místa propustnosti
* počet propustných cest materiálem
* vizualizace výsledků

Základním přístupem pro nalezení velikosti kritického místa je postupná dilatace hraniční oblasti, dokud nedojde k uzavření materiálu. Počet těchto dilatací nám poté logicky dává velikost kritického místa.

Operace dilatace je již dostupná funkce v Octave, která je velmi dobře optimalizovaná, takže nemá smysl vytvářet dilataci vlastní. Vstupními parametry pro tuto funkci jsou černobílý obraz a strukturní element ve formě matice. Pro 2D si nadefinujeme následující strukturní elementy:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | 0 | 0 | 0 | | 0 | 1 | 1 | | 0 | 1 | 1 |   Obrázek 18- strukturní element a) | |  |  |  | | --- | --- | --- | | 1 | 1 | 0 | | 1 | 1 | 0 | | 0 | 0 | 0 |   Obrázek 19- strukturní element b) |

V našem řešení střídáme každou druhou dilataci s opačným strukturním elementem. Důvodem je rovnoměrnější rozšíření hraniční oblasti. Použitím pouze jednoho ze strukturních elementů by nemělo mít vliv na velikost kritického místa. Také můžeme vidět, že směry, do kterých provádíme dilataci jsou opačné, čímž zabráníme, abychom do jednoho směru rozšiřovali více než do směrů ostatních. Později si ukážeme výhodu tohoto střídání strukturních elementů na vizualizaci cesty materiálem, kdy se tato cesta formuje hezky uprostřed propustné cesty.

Základní myšlenka fungování algoritmu je znázorněna následujícím vývojovým diagramem:

Diagram

Description automatically generated

Obrázek 20- Vývojový diagram základního fungování algoritmu

Je třeba poznamenat, že před dilatací je nutné invertovat celý obraz, jelikož funkce z Octave provádí dilataci na nejsvětlejších pixelech. My ovšem chceme dilatovat pixely černé.

Pro vyplnění je využito obyčejného 2D řádkového semínkového vyplňování, jehož kód je převzat z [1]. Octave vlastní vyplňovací metodu splňující naše požadavky na vyplňování nemá. Originální kód byl napsán v C++, takže bylo nutné přepsání do jazyka Octave.

Pseudokód pro řádkové semínkové vyplňování převzatý z [1]:

funkce vyplň (x, y, barva\_vyplnění, barva\_hranice):

1. vytvoř nový zásobník
2. přidej do zásobníku pixel[x, y]
3. dokud není zásobník prázdný, proveď:
   1. vyjmi z vrcholu zásobníku pixel
   2. dokud pixel[x-1, y] leží uvnitř oblasti, proveď:
      1. x--
   3. přidej do zásobníku horní nebo spodní pixel, pokud se nachází v oblasti, je nevybarvený a levý sousední pixel není hraniční
   4. dokud se pixel nachází uvnitř oblasti:
      1. vybarvi pixel[x, y]
      2. přidej do zásobníku horní nebo spodní pixel, pokud se nachází uvnitř oblasti, není vybarvený a levý sousední pixel je hraniční
      3. x++

Ukažme a vysvětleme si fungování algoritmu na několika jednoduchých testovacích obrázcích.



Obrázek 21- Jednoduchá oblast s velikostí kritického místa 3

Podívejme se na průběh algoritmu v oblasti, kde předem víme, že velikost kritického místa je tři. Obraz je předem binarizovaný, takže algoritmus začne rovnou postupně dilatovat obraz střídajícími se strukturními elementy. Propustnost zjišťujeme od levé strany po pravou. Můžeme vidět, že po třetí dilataci již oblast není propustná. Dalším krokem je vizualizace nalezené propustné cesty. Když zobrazíme vyplněnou cestu před posledním uzavírajícím krokem dilatace do původního obrazu, cesta nám pasuje přesně do středu.



Obrázek 22- Postup řešení algoritmem

Table

Description automatically generated

Obrázek 23- Výstup z konzole

Výstup z okna konzole nám dává informaci o velikosti kritického místa v obraze, v tomhle případě 3. Následně počet cest se stejnou velikostí tohoto parametru. Výstup také obsahuje informace z profileru, který Octave poskytuje. Zde vidíme prvních deset časově nejnáročnějších operací a celkové trvání běhu programu. V tomto případě se jedná o oblast 10x10, tudíž nic zvlášť užitečného z tohoto výstupu nevyčteme.

Pro znázornění, jak je chápána propustnost materiálem si uveďme ještě jeden příklad:



Obrázek 24- Jednoduchá oblast se dvěma cestami

Tato oblast obsahuje dvě cesty obrazem. Velikost kritického místa pro první je 2, druhá cesta má velikost 1. V tomto případě je cesta s menším kritickým místem ignorována, jelikož se nejedná o cestu s největší kapacitou průchodu. V úvahu se bere vždy cesta s největším kritickým místem, případně více cest se stejnou hodnotou tohoto parametru.



Obrázek 25- Postup řešení oblasti se dvěma cestami

Můžeme vidět, že na maximální propustnost materiálem nemá druhá cesta žádný vliv, je to samozřejmě dáno samotným přístupem dilatace. Aby byla brána v úvahu každá cesta, musel by se použít jiný přístup.

Uveďme si ještě příklad, kdy cest se stejnou hodnotou kritického místa je v materiále více. Každou cestu odlišíme jinou barvou výplně.



Obrázek 26- Příklad více cest v materiálu

* 1. Optimalizace algoritmu

Pro menší oblasti není rychlost algoritmu příliš velký problém. Ovšem s většími oblastmi se musíme začít zamýšlet nad tím, zda neexistuje lepší přístup k řešení tohoto problému. Hlavním problémem algoritmu, který postupně obraz dilatuje je ten, že mnohokrát opakujeme operaci vyplnění. Operace vyplnění je nejnáročnější operací, takže má cenu se zamyslet, jak snížit množství volání této operace, aniž bychom narušili výsledek algoritmu.

Jedna možnost, jak tento postup lépe optimalizovat, je dilatovat od konce. Tímto způsobem voláme vyplnění pro menší oblasti než při způsobu dilatace od původního obrazu, až po uzavřený. Tento přístup nám sice způsobí navýšení množství dilatací, ovšem ty jsou mnohem méně výpočetně náročné než u vyplňování.

Průběh algoritmu zůstane prakticky stejný, akorát si na začátku určíme nejvyšší počet dilatací, který na obraze můžeme provést. Můžeme předpokládat, že obraz nemůže zůstat propustný po počtu dilatací rovných největší hodnotě z šířky nebo výšky obrazu. Mění se nám také podmínka ukončení hlavního cyklu, kdy v předchozím přístupu jsme prováděli dilataci, dokud nedošlo k uzavření materiálu, naopak zde budeme ubírat počet dilatací, dokud se materiál nestane propustným. Tímto způsobem by oba algoritmy měly dojít ke stejným výsledkům.

Abychom zbytečně neprováděli velké množství dilatací, tak definujeme dynamicky se měnící krok pro dilataci. Nazývejme tuto dilataci jako hrubou, jejím cílem je co nejrychleji najít mez, kdy se v materiále objeví propustná cesta. Počet dilatací si určíme tak, že z velikosti výšky a šířky vybíráme tu větší hodnotu, kterou dělíme polovinou v každém kroku. Takže bychom měli například tuto sekvenci: 100 50 25 13…

Průběh algoritmu můžeme lépe vidět na následujícím obrázku. Díky hrubé dilataci se dozvíme, že k otevření materiálu dochází mezi šesti a třemi dilatacemi. To znamená že postupnou dilatací začneme od vyššího čísla tohoto rozsahu, kdy již dilatujeme krok po kroku, dokud nenarazíme na přesný počet dilatací.



Obrázek 27- Příklad zpětného algoritmu

Oproti předchozímu algoritmu je tento značně rychlejší. Obyčejný algoritmus nad tímto obrazem stráví 6.3 sekund, zatímco optimalizovaný běží skoro dvojnásobně rychleji s časem 3.8 sekund.

* 1. Zobecnění pro 3D

Převedení výše popsaného algoritmu do 3D je poměrně přímočarý proces, kdy nemusíme měnit hlavní myšlenky. Pro správné fungování je nutné upravit pouze několik věcí. Například nejdříve vytvoření nového strukturního elementu. Funkce dilatace v Octave funguje také pro 3D matice, takže akorát musíme vytvořit odpovídající 3D strukturní element. Taktéž musíme pro 3D přizpůsobit funkci vyplňování. Nakonec je nutný odlišný přístup k zobrazení průběhu algoritmu. Ideální by bylo 3D zobrazení, ovšem Octave není pro 3D zobrazení ideální. Vystačíme si tedy se zobrazením přední a zadní stěny, mezi kterými sledujeme propustnost. Vše je jinak analogické s 2D přístupem.

Stejně, jako jsme optimalizovali algoritmus ve 2D, tak učiníme i u 3D. V tomto případě je jednoduchý přístup od nejméně dilatovaného obrazu po nejvíce dilatovaný velice náročný. Jednoduchý algoritmus na tomto příkladě probíhá 169 sekund, zatímco zpětný 40 sekund, skoro čtyřikrát rychleji! Tato optimalizace se tedy stává velmi důležitou.



Obrázek 28- Průběh 3D algoritmu na 100x100x100 oblasti

Vizualizace zde je bohužel méně přehledná, ovšem můžeme i tak vidět, které cesty z jedné stěny přecházejí do druhé. Každý krok je znázorněn stěnou F1 a F2, které jsou vzájemně protilehlé.

* 1. Možnost pokračování a zlepšení algoritmu

Zajímavou myšlenkou je další optimalizace algoritmu pomocí paralelizace některých procesů na více vláknech. V tuhle chvíli je průběh algoritmu sekvenční, ovšem existuje možnost provádět operace dilatace a vyplňování na více vláknech naráz. Tyto operace nejsou závislé na průběhu ostatních operací v průběhu programu.

Následně by se dalo přemýšlet o převodu tohoto problému z rastrové grafiky do oboru teorie grafů. Hlavní problém by byl samotný převod, který se jeví jako velmi náročný. Bylo by nutné ověření tohoto tvrzení, ovšem je možné, že jakmile bychom měli k dispozici graf našeho materiálu, operace na něm prováděné by mohly být rychlejší a efektivnější.

ZÁVĚR

RESUMÉ

SUMMARY

SEZNAM POUŽITÉ LITERATURY

1. **Mrógala J.:** *Algoritmy vyplňování pro 3D rastr*. Ostrava, 2021. Bakalářská práce. Ostravská univerzita. Přírodovědecká fakulta. Katedra informatiky a počítačů.
2. **Sedláček J.:** *Úvod do teorie grafů*. Academia, Praha, 1977. ISBN 510-21-826.
3. **Bělohlávek R., Vychodil V.:** *Diskrétní matematika pro informatiky II.* Olomouc: PřF UP Olomouc, 2006.
4. **Černý J.:** *Základní grafové algoritmy*. Praha, 2010. KAM, MFF UK.
5. **Matějka P.:** *Algoritmy pro generování a řešení bludišť*. Brno, 2012. Diplomová práce. Masarykova univerzita. Fakulta informatiky.
6. **Šonka M., Hlaváč V., Boyle R.:** *Image Processing, Analysis and Machine Vision*. Springer 2015.
7. **Haj Ibrahim, S., Skibinski J., Oliver G. J., Wejrzanowski T.:** *Microstructure effect on the permeability of the tape-cast open-porous materials*. Materials & Design Vol. 167, April 2019.
8. **Grossmanová P.:** *Model 3D prostoru*. Ostrava, 2019. Diplomová práce. Ostravská univerzita. Přírodovědecká fakulta. Katedra informatiky a počítačů.
9. SEZNAM POUŽITÝCH SYMBOLŮ

|  |  |  |  |
| --- | --- | --- | --- |
| ABC |  |  | Význam první zkratky. |
| B |  |  | Význam druhé zkratky. |
| C |  |  | Význam třetí zkratky. |
|  |  |  |  |

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH