ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta stavební

Stavební inženýrství

Konstrukce pozemních staveb

Diplomová práce

Využití grafických karet pro modelování konstrukcí

Autor: Bc. Ivan Kimák

Vedúci práce: prof. Dr. Ing. Bořek Patzák

Akademický rok: 2018/2019

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta stavební
Thákurova 7, 166 29 Praha 6

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: Kimák Jméno: Ivan Osobní číslo: 423013

Zadávající katedra: Katedra mechaniky (K132)

Studijní program: (N3607) Stavební inženýrství

Studijní obor: (3608T008) Konstrukce pozemních staveb

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce: Využití grafických karet pro modelování konstrukcí

Název diplomové práce anglicky: Structural modeling using graphical processing units

Pokyny pro vypracování:

Formulace a implementace 2D numerického modelu konstrukce využívajícího procesor grafické karty. Diskuse implementačních aspektů aplikace na řešení vybraných problémů, vyhodnocení efektivity použité metody a implementace.

Seznam doporučené literatury:

Jméno vedoucího diplomové práce: prof. Dr. Ing. Bořek Patzák

Datum zadání diplomové práce: 21.2.2019 Termín odevzdání diplomové práce: 19.5.2019

Údaj uveďte v souladu s datem v časovém plánu příslušného ak. roku

Podpis vedoucího práce Podpis vedoucího katedry

III. PŘEVZETÍ ZADÁNÍ

Beru na vědomí, že jsem povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je nutné uvést v diplomové práci a při citování postupovat v souladu s metodickou příručkou ČVUT „Jak psát vysokoškolské závěrečné práce“ a metodickým pokynem ČVUT „O dodržování etických principů při přípravě vysokoškolských závěrečných prací“.

Datum převzetí zadání Podpis studenta(ky)

Čestné prehlásenie:
Prehlasujem, že som diplomovú prácu na tému Využití grafických karet pro modelování konstrukcí, spracoval samostatne za pomoci uvedenej literatúry. Ďalej prehlasujem, že nemám závažný dôvod proti použitiu tohto školského diela v zmysle § 60 zákona č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon).
V Prahe dňa 19.5.2019 Ivan Kimák

Poďakovanie:
Ďakujem môjmu vedúcemu práce prof. Dr. Ing. Bořkovi Patzákovi za trpezlivosť, ochotu a cenné rady pri vedení mojej diplomovej práce. Chcel by som sa poďakovať mojej rodine, ktorá mi umožnila štúdium a podporovala ma počas celého jeho trvania. Na záver ďakujem mojej priateľke, ktorá mi bola po celú dobu oporou.

Názov práce:
Využití grafických karet pro modelování konstrukcí
Autor: Bc. Ivan Kimák
Štúdijný obor: Stavební inženýrství
Druh práce: Diplomová práca
Vedúci práce: prof. Dr. Ing. Bořek Patzák
Abstrakt: Táto práca sa zaoberá numerickým modelovaním dvojrozmerných konštrukcií. Skúma efektivitu riešenia pohybových rovníc, pomocou metódy priamej integrácie. K riešeniu bola využitá paralelizácia výpočtov, ktoré boli realizované na grafickom procesore počítača. Výsledky boli posúdené na základe časovej efektivity riešenia v porovnaní s CPU.
Kľúčové slová: NVIDA CUDA, modelovanie konštrukcií, paralelné výpočty, priama integrácia, metóda konečných prvkov, GPU programovanie, Newmarkova metóda

Title:
Structural modeling using graphical processing units
Author: Bc. Ivan Kimák
Abstract: Basic concept of this study is structural modeling of two-dimensional objects, using direct integration method to find solution of movement equations. Furthermore parallel algorithms were used to examine its effects on solution efficiency. Calculations were handled by graphical processor unit. Efficiency of solution was compared to traditional computing on CPU.
Key words: NVIDA CUDA, structural modeling, parallel computing, direct integration, finite elements method, GPU computing, Newmark's method

Obsah

Kapitola 1

Úvod

1.1 Ciele práce

Cieľom práce bolo vytvorenie numerického modelu pre riešenie problémov stavebnej mechaniky. Základným aspektom bolo využitie grafického procesoru počítača pre urýchlenie riešenia implementovaním paralelizácie výpočtov. A následné vyhodnotenie efektivity riešenia.

1.2 Popis problému

Základným námetom tejto práce je skúmanie efektivity riešenia pohybových rovníc metódou priamej integrácie, ktoré je sprostredkované grafickým procesorom počítača, skrátene GPU. Použité metódy boli aplikované na dvojrozmernej konštrukcii v rovine, ktorá bola zaťažená vonkajším zaťažením, a to buď statickou alebo periodicky sa meniacou hodnotou. Cieľom práce je nájsť riešenie, ktoré bude popisovať odozvu konštrukcie v čase. Odozvou konštrukcie sa rozumejú vzniknuté posuny konštrukcie a z nich vyplývajúce deformácie a napätia pôsobiace na konštrukciu. Diferenciálne rovnice popisujúce túto závislosť sú odvodené z podmienok rovnováhy a sú diskretizované v priestore metódou konečných prvkov a v čase metódou konečných diferencií. Problém vedie k postupnosti riešení v diskrétnych časových krokoch. Výpočty potom boli spracované s využitím grafického procesoru. Výsledkom práce je vyhodnotenie efektivity takéhoto riešenia v závislosti od času potrebného pre výpočty. Pre porovnanie boli výsledky porovnané aj s výsledkami spracovanými na základnej procesnej jednotke, skrat. CPU.

Kapitola 2

Grafické karty

2.1 História grafických kariet

Koncom 20. storočia došlo vývojom počítačových technológií k vytvoreniu prvých grafických kariet firmou IBM. Tá vytvorila prvé dva typy kariet MDA (Monochrome Display Adapter) a CGA(Color Graphics Adapter). Vynálezom zbernice PCI firmou Intel došlo k pokroku vo vývoji grafických procesorov. Táto zbernica umožňovala rýchlejší presun infomácií medzi CPU a GPU. Hlavnou hnacou silou bolo zvýšenie výkonu pre videoherný priemysel. Od začiatku 21. storočia sa stali hlavnými hráčmi na trhu spoločnosti AMD a NVIDIA, ich karty boli určené hlavne pre 3D rendering. Práve NVIDIA v roku 2001 uviedla na trh prvé karty, ktoré umožňovali GPGPU (General-purpose computing on graphical processor unit) alebo vo voľnom preklade Univerzálne počítanie pomocou grafického procesoru. Čo umožňuje, že matematické výpočty bolo možné riešiť pomocou GPU namiesto tradičného riešenia prostredníctvom CPU. To viedlo k využitiu GPGPU aj v akademickej sfére. Pomocou GPU sa podarilo dosiahnuť rýchlejší výpočet problémov ako násobenie vektorov alebo matíc [8] . V dnešnej dobe sa využitie GPU presunulo hlbšie do teoretickej sféry, ako implementácia pre strojové učenie v oblasti umelej inteligencie.

2.2 Porovnanie CPU a GPU

Konštrukčne sú si CPU a GPU podobné v tom, že oba procesory sú zostavené z veľkého počtu tranzistorov, ktoré prevádzajú veľké množstvo operácií každú sekundu. Moderné CPU sa skladajú z určitého počtu výpočtových jadier, bežne

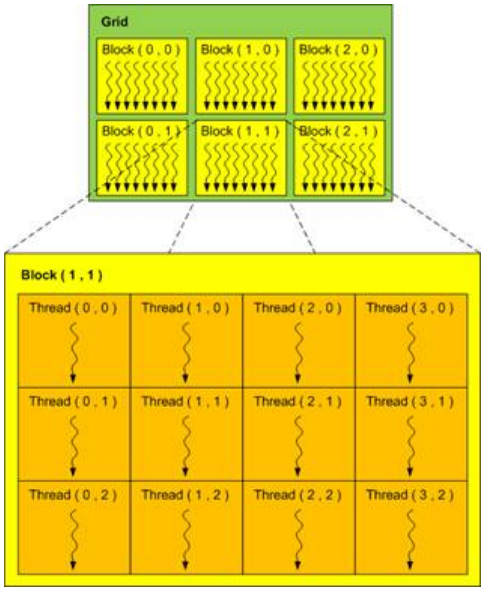
medzi jedným až štyrmi, ale určité procesory môžu obsahovať až desiatky jadier. Výhodou CPU je, že môže prevádzať širokú škálu rôznych operácií. Na rozdiel od CPU je grafický čip zložený z obrovského množstva výpočtových jadier, dnes v rádoch stoviek až tisícov jednotlivých jadier. Na všetkých sa dajú paralelne prevádzať operácie, teda aj grafický procesor použitý pre riešenie tohto problému dokázal počítať stovky operácií zároveň. Táto vlastnosť sa využíva hlavne pre vykresľovanie zložitých 3D grafík. Hlavnou nevýhodou GPU je to, že jeho jadrá môžu vykonávať iba jednoduché funkcie, čiže nie sú tak všestranné ako klasické CPU. Ďalšou nevýhodou je spôsob paralelných výpočtov, ktoré sa môžu vykonávať iba súčasným spustením bloku operácií a je potrebné čakať na ukončenie každej operácie, kým sa môže spustiť ďalší blok. Preto je dôležité správne členenie výpočtov, aby sa minimalizovala strata výpočtového času. Vo výsledku GPU poskytuje veľký výpočtový potenciál vďaka paralelnosti výpočtov, na druhej strane však umožňuje len výpočet jednoduchých operácií, ktoré musia byť čo najlepšie optimalizované.

2.3 Práca s GPU

Pre výpočty bolo potrebné zostaviť program, ktorý sa bude spúšťať priamo na grafickej karte, namiesto bežného spracovania na CPU. K napísaniu takéhoto kódu slúži prostredie CUDA Toolkit od spoločnosti NVIDIA, ktoré umožňuje písanie a exekuovanie počítačového kódu na GPU. Pre väčšie priblíženie problému je potrebné uviesť podrobnejšie vysvetlenie vnútorných procesov na GPU. Tieto procesy sa dajú ilustrovať popisom programovacieho modelu prostredia CUDA. Vstupom pre výpočty je sériový kód, ktorý sa následne rozdelí na paralelný kód zvaný kernel. Kernel sa virtuálne skladá z vláken (z ang. thread), ktoré predstavujú jednotlivé operácie prevádzané na jadrách. Celý výpočet sa takto rozdelí na určitý počet kernelov, ktoré sa postupne vykonávajú na tzv. Streaming multiprocesoroch, z ktorých sa skladajú čipy grafickej karty. Každý multiprocesor obsahuje určitý počet CUDA jadier, ktoré slúžia k paralelným výpočtom. Multiprocesory môžu pracovať nezávisle na sebe, čím sa uľahčuje problém synchronizácie výpočtov. Na jednom multiprocesore sa môže naraz vykonávať niekoľko tisíc samostatných vláken výpočtov. Tie pracujú po 32 v skupinách, zvaných warpy. Každému warpu sú priradené registre pre ukladanie dát, v prípade nedostatku miesta im je priradený priestor v globálnej pamäti procesora.

Obr. 2.1: Schéma fyzického usporiadania multiprocesoru prevzané z [6]

Práve správny manažment pamäte je jedným z kľúčových faktorov ovplyvňujúcich efektivitu výpočtov. Jedným z dôvodov je obmedzená rýchlosť prenosu dát medzi CPU a GPU, takže je účinné koncipovať program tak, aby sa všetky potrebné dáta nahrali do pamäte GPU a nebolo potrebné prenášať dielčie výsledky medzi zariadeniami. Výhodou je aj zarovnanie pamäte warpov s prvkami zarovnanými vedľa seba, aby vlákna nepristupovali do pamäte náhodne, ale združene. Na koniec je potrebné definovať spôsob identifikácie vláken, potrebný pre prerozdelenie jednotlivých činností na vlákna. Pred spustením kernelu sa zadefinuje mriežka, zostavená z blokov vláken. Každému bloku v mriežke je priradené unikátne identifikačné označenie skr. id a takisto je id priradené každému vláknu v bloku. Kernel je potom vykonávaný po blokoch v dopredu neurčenom poradí. Každý blok naraz spustí beh všetkých jeho vláken simultánne. Kedže ide iba o virtuálne delenie, je možné kernel rozdeliť až na trojdimenzionálnu sieť blokov, kde každý z nich môže opäť byť trojdimenzionálny.



Obr. 2.2: Delenie siete na bloky vláken [6]

Kapitola 3

Popis riešenia

Táto kapitola sa zaoberá rozborom použitých metód a taktiež popisom použitého počítačového algoritmu. Matematické rovnice a teoretické poznatky boli odvodené z použitej literatúry [11] [1].

3.1 Použitý hardware a software

Z dôvodu možnosti budúcej replikácie experimentu alebo porovnania efektivity riešenia problému uvádzam popis použitého softwaru a hardwaru.

3.1.1 Hardware

Výpočty boli vykonávané na prenosnom počítači "DELL(TM) Inspirion 15 7000 Gaming" [3] s integrovaným procesorom Ïntel(R) Core(TM) i7/7700HQ" [4] a grafickým procesorom "NVIDIA GeForce GTX 1050 Ti" [7], ktorý má 768 výpočtových jadier CUDA. Táto informácia je dôležitá vzhľadom na to, že počet týchto jadier určuje výpočtovú kapacitu GPU (z ang. Graphical processing unit alebo grafická výpočtová jednotka), čo má priamy vplyv na výpočtovú kapacitu riešenia. Toto GPU je v nižšej triede výkonu medzi dostupným hardwarom, keďže má len 768 jadier, zatiaľ čo najvýkonnejšie grafické karty ako napr. "GeForce GTX TITAN Z"majú k dispozícii až 5760 výpočtových jadier.

3.1.2 Software

Základ použitého algoritmu bol napísaný a kompilovaný ako konzolová aplikácia v prostredí "Microsoft Visual Studio"využitím programovacieho jazyka C++. GPU bolo využité pre riešenie sústav lineárnych rovníc, s pridaním knižnice "CUDA Toolkit 10.1", ktorá je nadstavbou prostredia Microsoft Visual Studio, takisto postavenou na jazyku C++. Pre úspešnú kompiláciu kódu je potrebné integrovať do prostredia aj knižnicu cuBLAS. Vyhodnotenie výsledkov bolo implementované v prostredí "Mathworks MATLAB". Tento postup bol potrebný pre grafické znázornenie výsledkov. Pracovné prostredie CUDA taktiež umožňuje grafický výstup, ale z dôvodu zložitosti grafického programovania a z nej vyplývajúcej časovej náročnosti, bolo použité jednoduchšie riešenie pomocou softwaru MATLAB.

3.2 Metóda konečných prvkov

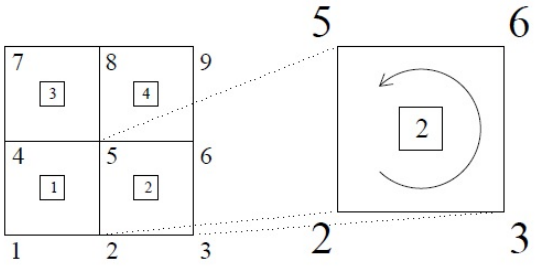
Táto časť je venovaná základnému popisu riešenia diskretizácie dvojrozmerného problému použitím metódy konečných prvkov (MKP). Diskretizácia bola prevedená pomocou dvojrozmerných izoparametrických štvoruholníkových prvkov. Pre integráciu výsledných pohybových rovníc bola použitá Newmarkova metóda.

3.3 Formulácia problému

Úloha je zadaná pomocou geometrických a materiálových charakteristík problému a zadaním okrajových (podoprenie a pôsobiaca sila) a počiatočných podmienok. Geometria je zadaná priamo ako vektor súradníc uzlov siete konečných prvkov, kedže zostrojenie generátora sietí predstavuje vlastný problém, ktorého riešenie môže byť veľmi komplexné v prípade všeobecných oblastí. Vektor súradníc uzlov siete je

$$X(n,2) = \begin{Bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ & . \\ & . \\ x_n & y_n \end{Bmatrix}.$$

Je potrebné zostrojiť aj vektor kódových čísel prvkov. Ten priradí každému prvku zoradenú štvoricu bodov, ktoré definujú uzly daného prvku. Metodika číslovania je zobrazená na schéme.



Obr. 3.1: Číslovanie prvkov v metóde konečných prvkov

Vektor kódových čísel prvkov je

$$V_{nodes}(m,4) = \begin{Bmatrix} X(1) & X(2) & X(5) & X(4) \\ X(2) & X(3) & X(6) & X(5) \\ X(4) & X(5) & X(8) & X(7) \\ X(5) & X(6) & X(9) & X(8) \end{Bmatrix}.$$

V práci sa uvažuje s izotropným, elastickým materiálom. Jeho charakteristiky sú definované pomocou dvoch parametrov a to, Youngovym modulom pružnosti **E**, Poissonovou konštantou **v**. Okrem toho je zadaná hrúbka **t** a hustota $\boldsymbol{\rho}$. Popis zadávania síl je podrobnejšie priblížený až pri popise použitého algoritmu, pretože pri algoritmizácii je neefektívne zadávanie síl ako celého vektoru. Potrebným vstupom je ešte počet časových krokov, ktoré majú byť vykonané a ich dĺžka.

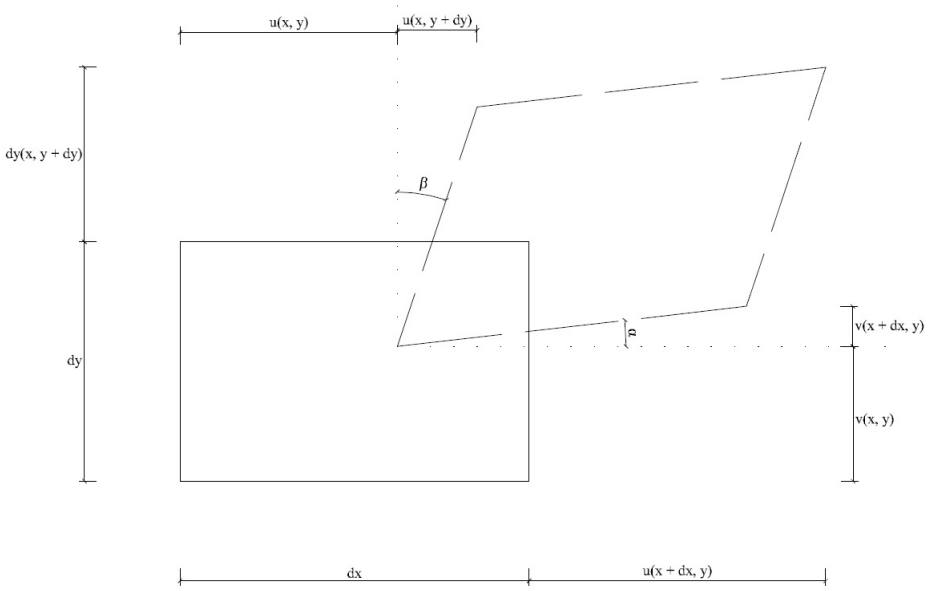
3.3.1 Premenné

Úlohou teórie pružnosti je určiť na telese vypĺňajúcom oblasť Ω a ohraničenom hranicou Γ sadu troch polí, a to vektorové pole posunov $\mathbf{u}$, pole deformácie $\boldsymbol{\varepsilon}$ a pole napätí $\boldsymbol{\sigma}$. Pre dvojrozmerný problém majú tieto polia nasledujúce zložky

$$\begin{aligned} \mathbf{u} &= \{u, v\}^T, \\ \boldsymbol{\varepsilon} &= \{\varepsilon_x, \varepsilon_y, \gamma_{xy}\}^T, \\ \boldsymbol{\sigma} &= \{\sigma_x, \sigma_y, \tau_{xy}\}^T. \end{aligned}$$

3.3.2 Geometrické rovnice

Geometrické rovnice popisujú vzťah medzi posunom $\mathbf{u}$(x) a zložkami vektora deformácie $\boldsymbol{\varepsilon}$(x). Daný je dvojrozmerný prvok s dĺžkou hrany dx v smere osi x a dĺžkou dy v smere osi y. Prvok je potom premiestnený v smere osi x a y, o vzdialenosť dx resp. dy. Predpokladá sa, že prvok sa deformoval voči pôvodnému tvaru ako v smere osi x tak aj osi y.



Obr. 3.2: Odvodenie geometrických rovníc

Pre nekonečne malý posun dx a dy môžeme písať rovnice v tvare

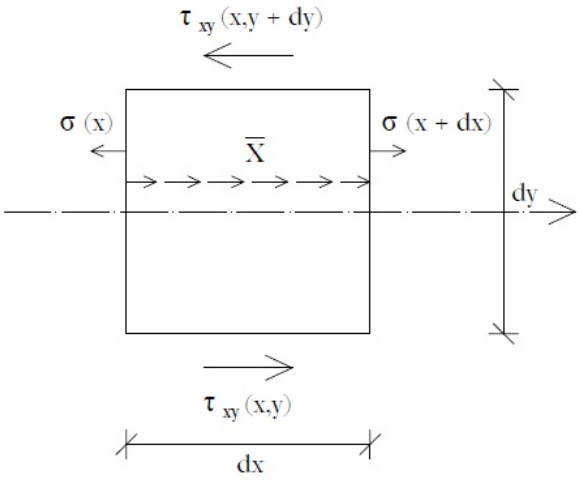
$$\varepsilon_x = \lim_{dx\to\infty} \frac{dx + u(x+dx) - u(x) - dx}{dx} = \frac{du(x)}{dx},$$
$$\varepsilon_y = \lim_{dy\to\infty} \frac{dy + v(y+dy) - v(y) - dy}{dy} = \frac{dv(y)}{dy},$$
$$\gamma_{xy} = \alpha + \beta = \lim_{dx,y\to\infty} \frac{u(x, y+dy)}{dy + v(x, y+dy) - v(x, y)} + \frac{v(x, y+dy)}{dx + u(x+dx, y) - u(x, y)} =$$
$$= \frac{dv(x)}{dx} + \frac{du(y)}{dx}.$$

$$\begin{Bmatrix} \varepsilon_x(\mathbf{x}) \\ \varepsilon_y(\mathbf{x}) \\ \gamma_{xy}(\mathbf{x}) \end{Bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix} \begin{Bmatrix} u(\mathbf{x}) \\ v(\mathbf{x}) \end{Bmatrix}, \tag{3.1}$$

$$\varepsilon(\mathbf{x}) = \partial^T u(\mathbf{x}). \tag{3.2}$$

3.3.3 Podmienky rovnováhy

Pre odvodenie Cauchyho rovníc rovnováhy je opäť uvažovaný dvojrozmerný prvok ohraničený v rovine, o rozmeroch dx a dy a hrúbky t. Je definovaný objemový vektor síl $\overline{\mathbf{X}}$ ako funkcia premennej x. Pre odvodenie vnútorných síl v smere y sa predpokladá zdroj sily v smere tejto osi. Na hranici prvku Γ je potom definovaná aj funkcia normálového napätia $\sigma(x)$ v smere normály na hranu prvku a šmykového napätia τ_{xy}. Rovnice rovnováhy sú potom odvodené takto



Obr. 3.3: Odvodenie podmienok rovnováhy

$$-\sigma_x(x)\Delta y t + \sigma_x(x+\Delta x)\Delta y t - \tau_{xy}(x,y)\Delta x t + \tau_{xy}(x,y+\Delta y)\Delta x t + \overline{X}\Delta x\Delta y = 0,$$
$$-d\sigma_x(x)\Delta y + d\tau_{xy}(x,y)\Delta x + \overline{X}\Delta x\Delta y = 0,$$
$$\frac{d\sigma_x}{dx} + \frac{d\tau_{xy}}{dy} + \overline{X} = 0.$$

$$\mathbf{x} \in \Omega : \begin{bmatrix} \frac{\partial}{\partial x} & 0 & \frac{\partial}{\partial y} \\ 0 & \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix} \begin{Bmatrix} \sigma_x(\mathbf{x}) \\ \sigma_y(\mathbf{x}) \\ \tau_{xy}(\mathbf{x}) \end{Bmatrix} + \begin{Bmatrix} \overline{X}(\mathbf{x}) \\ \overline{Y}(\mathbf{x}) \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix}, \tag{3.3}$$

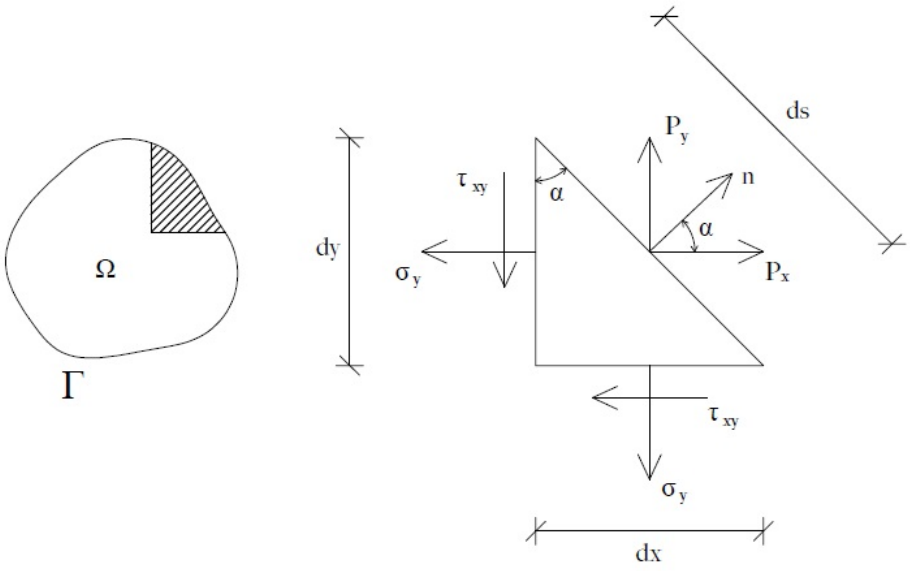
$$\partial^T \sigma(\mathbf{x}) + \overline{X} = 0. \tag{3.4}$$

3.3.4 Odvodenie okrajových podmienok

Kinematické okrajové podmienky
Okrajové podmienky vyplývajú z predpísaných hodnôt posunov $\overline{u}(\Gamma_u)$ na hrane $\Gamma = \Gamma_u \cup \Gamma_p$ prvku. Takto je možné definovať

$$\mathbf{x} \in \Gamma_u : \mathbf{u}(\mathbf{x}) = \overline{\mathbf{u}}(\mathbf{x}). \tag{3.5}$$

Statické okrajové podmienky
Uvádzaný príklad platí pre statické okrajové podmienky, kde na prvok pôsobí vonkajšia sila $\overline{p}(x)$ s predpísanou hodnotou. Je potrebné definovať aj funkciu $n(x)$ ktorá určuje smer normály v bodoch na hranici prvku. Na hranici prvku teda platí



Obr. 3.4: Schéma pre odvodenie okrajových podmienok

$$\begin{aligned}
\sigma_x dy + \tau_{xy} dx - p_x ds &= 0, \\
\sigma_x \frac{dy}{ds} + \tau_{xy} \frac{dx}{ds} - p_x &= 0, \\
\sigma_x cos\alpha + \tau_{xy} sin\alpha - p_x &= 0, \\
\sigma_x n_x + \tau_{xy} n_y - p_x &= 0.
\end{aligned}$$

$$\mathbf{x} \in \Gamma_u : \begin{bmatrix} n_x(\mathbf{x}) & 0 & n_y(\mathbf{x}) \\ 0 & n_y(\mathbf{x}) & n_x(\mathbf{x}) \end{bmatrix} \begin{Bmatrix} \sigma_x(\mathbf{x}) \\ \sigma_y(\mathbf{x}) \\ \tau_{xz}(\mathbf{x}) \end{Bmatrix} - \begin{Bmatrix} \overline{p}_x(\mathbf{x}) \\ \overline{p}_y(\mathbf{x}) \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix}, \tag{3.6}$$

$$\mathbf{n}(\mathbf{x})\sigma(\mathbf{x}) - \overline{\mathbf{p}}(\mathbf{x}) = 0. \tag{3.7}$$

3.3.5 Konštitutívne rovnice

Poslednými známymi vzťahmi sú konštitutívne (fyzikálne) rovnice vyjadrujúce vzťah medzi vektorom napätia a deformácie [9]. Pre dvojrozmerný problém rovinnej napätosti za predpokladu lineárneho, izotropného materiálu majú tieto rovnice tvar

$$\begin{Bmatrix} \sigma_x(\mathbf{x}) \\ \sigma_y(\mathbf{x}) \\ \tau_{xz}(\mathbf{x}) \end{Bmatrix} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \begin{Bmatrix} \varepsilon_x(\mathbf{x}) \\ \varepsilon_y(\mathbf{x}) \\ \gamma_{xy}(\mathbf{x}) \end{Bmatrix}, \tag{3.8}$$

$$\sigma(\mathbf{x}) = D\varepsilon(x). \tag{3.9}$$

Pre úlohu rovinnej napätosti taktiež platí, že nevzniká napätie v smere kolmom na rovinu prvku, teda $\sigma_z = 0$. Pre pomernú deformáciu v tomto smere platí

$$\varepsilon_z = -\frac{\nu}{E}(\varepsilon_x + \varepsilon_y). \tag{3.10}$$

3.3.6 Vyjadrenie diferenciálnej rovnice

Pre diferenciálnu rovnicu a okrajové podmienky v tvare

$$\begin{aligned}
\partial^T D \partial u(x) \overline{X} &= 0, \\
\mathbf{u}(\mathbf{x}) = \overline{\mathbf{u}}(\mathbf{x}) &= 0, \\
\mathbf{n}(\mathbf{x}) D \partial u(x)(\mathbf{x}) - \overline{\mathbf{p}}(\mathbf{x}) &= 0,
\end{aligned}$$

platí, že každé $u(x)$, ktoré spĺňa homogénne okrajové podmienky je riešením diferenciálnej rovnice. Toto riešenie sa nazýva silné riešenie diferencialnej rovnice, no nájsť takéto riešenie analyticky pre všeobecnú oblasť je výpočtovo nemožné. Riešenie je ale možné nájsť v takzvanej slabej forme. Takéto riešenie s dostatočnou presnosťou sa dá nájsť použitím Galerkinovej metódy vážených reziduí r, prípadne vychádzať z princípu virtuálnych posunov. Metóda vážených reziduí pre náš problém má tvar

$$\int_{\Omega} \delta u(\mathbf{x})^T \underbrace{\left(\partial\sigma(\mathbf{x}) + \overline{X}\right)}_{r} d\Omega = 0, \tag{3.11}$$

kde $\delta u(x)$ je tzv. testovacia funkcia, ktorá spĺňa homogénne okrajové podmienky. Úpravou integrálu sa dá tento výraz vyjadriť v tvare

$$\int_{\Gamma_u} \overbrace{\delta u(x)^T}^{0} n(x)\sigma(x)d\Gamma_u + \int_{\Gamma_p} \delta u(x)^T \overbrace{n(x)\sigma(x)}^{\overline{p}} d\Gamma_p - \int_{\Omega} \left(\partial^T \delta u(x)\right)^T \sigma(x)d\Omega + \int_{\Omega} \delta u(x)^T \overline{X}(x)d\Omega = 0, \tag{3.12}$$

ktorý je vyjadrením slabého riešenia rovnice. To je charakteristické menšími nárokmi na spojitosť riešenia $[C^2(\Omega \to C^0(\Omega)]$ a vyžaduje, aby bola len testovacia funkcia $u(x)$ dostatočne integrovateľná. Ďalej za predpokladu, že váhovej funkcii $\delta u(x)$ bude prisúdený fyzikálny význam virtuálneho posunu, člen $\partial^T \delta u(x)$ môže byť definovaný ako virtuálna deformácia $\delta\varepsilon(x)$. Potom sa dá úpravou rovnice odvodiť vzťah

$$\int_{\Omega} \delta\varepsilon(x)^T \sigma(x)d\Omega = \int_{\Gamma_p} \delta u(x)^T \overline{p}(x)d\Gamma_p + \int_{\Omega} \delta u(x)^T \overline{X}(x)d\Omega, \tag{3.13}$$

$$\delta W_{int} = \delta W_{ext}, \tag{3.14}$$

ktorý sa dá interpretovať ako rovnosť medzi prácou vnútorných a vonkajších síl.

3.3.7 Aproximácia testovacích a váhových funkcií v 1D

Testovacie a váhové funkcie boli zostrojené na základe Galerkinovskej aproximácie, ktorá tvrdí, že Galerkinovské riešenie $u_n(x) \in V_N$ je tak isto priblížené k riešeniu $u(x) \in V_N$ ako ktorýkoľvek iný vektor v priestore polygonických funkcií V_N ohraničenom bodmi (0,1). Aproximácia sa volí vo forme lineárnych kombinácií bázových funkcií, ktoré spĺňajú dve základné podmienky knocker-delta

$\sum N_i = 1$. Bázové a testovacie funkcie sa často volia v tvare Lagrangeových polynómov

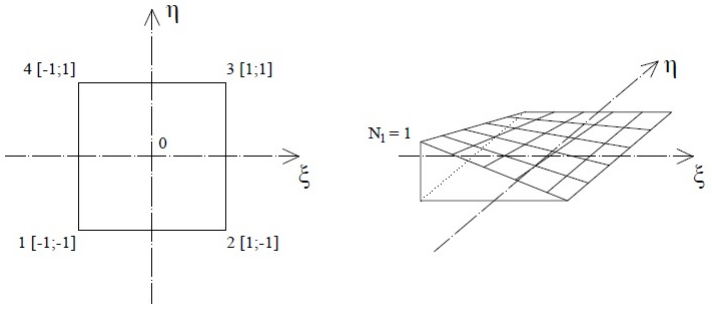
$$N_i^e = \frac{(x - X_1)...(x - x_{i-1})(x - x_{i+1})...(x - x_n)}{(x_i - x_1)...(x_i - x_{i-1})(x_i - x_{1+i})...(x_i - x_n)}. \tag{3.15}$$

Čitateľ zlomku zaručuje, že i-ta funkcia bude rovná nule v každom uzle okrem uzlu i-teho. Menovateľ zlomku potom normalizuje výraz v čitateli do takého tvaru, aby platilo, že súčet všetkých bázových funkcií je v každom uzle rovný práve jednej. To zaručuje spojitosť testovacích funkcií $u(x)$ a aproximačných funkcií N^e na hranách prvkov. Veľmi výhodné je vyjadriť tieto súradnice v prirodzených súradniciach $\xi, \eta \in < -1, 1 >$. To síce vedie k problému transformácie koordinát medzi prirodzenými súradnicami a súradnicovým systémom prvku, ale riešenie tohto problému je triviálnejšie ako zostavovať aproximačné funkcie špecificky pre každý prvok. Pre riešenie problému tejto práce boli použité prvky s lineárnou aproximáciou testovacích funkcií, pre popísanie dvojrozmerného prvku vzniknú aproximácie ako kombinácie jednorozmerných aproximačných funkcií. Lineárne aproximačné funkcie premenných ξ a η sú

$$N_i(\xi) = \frac{1}{2}(1 \pm \xi)$$
$$N_i(\eta) = \frac{1}{2}(1 \pm \eta)$$

3.3.8 Aproximačné funkcie v 2D

Číslovanie funkcií je závislé na číslovaní uzlov prvku, použité číslovanie je znázornené na nasledujúcom obrázku



Obr. 3.5: Číslovanie prvku (vľavo), grafické znázornenie 2D aproximačnej funkcie (vpravo)

$$\begin{bmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \end{bmatrix} = \begin{bmatrix} \frac{1}{4}(1-\xi)(1-\eta) \\ \frac{1}{4}(1+\xi)(1-\eta) \\ \frac{1}{4}(1+\xi)(1+\eta) \\ \frac{1}{4}(1-\xi)(1+\eta) \end{bmatrix}. \tag{3.16}$$

Pre dvojrozmerný problém sa používa zápis aproximačných rovníc a ich derivácií v tvare matice

$$N^e = \begin{bmatrix} N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4 \end{bmatrix} \tag{3.17}$$

$$B^e = \begin{bmatrix} \frac{\partial N_1}{\partial \xi} & 0 & \frac{\partial N_2}{\partial \xi} & 0 & \frac{\partial N_3}{\partial \xi} & 0 & \frac{\partial N_4}{\partial \xi} & 0 \\ 0 & \frac{\partial N_1}{\partial \eta} & 0 & \frac{\partial N_2}{\partial \eta} & 0 & \frac{\partial N_3}{\partial \eta} & 0 & \frac{\partial N_4}{\partial \eta} \\ \frac{\partial N_1}{\partial \eta} & \frac{\partial N_1}{\partial \xi} & \frac{\partial N_2}{\partial \eta} & \frac{\partial N_2}{\partial \xi} & \frac{\partial N_3}{\partial \eta} & \frac{\partial N_3}{\partial \xi} & \frac{\partial N_4}{\partial \eta} & \frac{\partial N_4}{\partial \xi} \end{bmatrix} \tag{3.18}$$

Po diskretizácii problému je možné na prvku aproximovať vektor posunov $u^e(x)$ a polia deformácií $\varepsilon^e(x)$ a napätí $\varepsilon^e(x)$ vektorom interpolačných funkcií $N^e(x)$ a ich derivácií $B^e(x)$ v tvare

$$u^e(x) \approx N^e(x) r^e, \tag{3.19}$$

$$\varepsilon^e(x) \approx B^e(x) r^e, \tag{3.20}$$

$$\sigma^e(x) \approx D^e(x)(B^e(x) r^e). \tag{3.21}$$

Rovnakým postupom sa dajú aproximovať váhové funkcie $\delta u^e(x)$ a $\delta \varepsilon^e(x)$

$$\delta u^e(x) \approx N^e(x) \delta r^e, \tag{3.22}$$

$$\delta \varepsilon^e(x) \approx B^e(x) \delta r^e. \tag{3.23}$$

Dosadením týchto vzťahov do rovnice (3.13) sa získa vzťah

$$\delta r^t \left\{ \sum_{e=1}^{n} \left[\overbrace{\int_\Omega B^e(x)^T D^e(x) B^e(x) d\Omega}^{K^e} r^e - \overbrace{\int_\Omega N^e(x)^T \overline{X} d\Omega}^{f^e_\Omega} - \underbrace{\int_{\Gamma_p} N^e(x)^T \overline{p}(x) d\Gamma_p}_{f^e_\Gamma} \right] \right\} = 0. \tag{3.24}$$

Z tejto formulácie je zrejmé, že vektor uzlových posunov je riešením rovnice statického problému

$$Kr = f = f_\Omega + f_\Gamma. \tag{3.25}$$

Pre riešenie problému dynamického systému je potrebné doplniť podmienky rovnováhy a okrajové podmienky o zotrvačné a tlmiace sily. Touto úpravou sa dá rovnakým postupom prísť k riešeniu pohybovej rovnice v tvare

$$M\ddot{r}_{t+\Delta t} + C\dot{r}_{t+\Delta t} + Kr_{t+\Delta t} = R_{t+\Delta t}, \tag{3.26}$$

kde M a C predstavujú matice hmotnosti resp. tlmenia. Pre pohybovej rovnici sa používa formálny zápis vektoru síl f ako $R_{t+\Delta t}$. Matica hmotnosti sa dá vyjadriť v tvare

$$M^e = \int_\Omega N^{eT}(x)\rho N^e(x)d\Omega. \tag{3.27}$$

Matica C potom predstavuje vplyv tlmenia na kmitavý pohyb. Obvykle sa vyjadruje vo forme Raylegovho útlmu ako lineárna kombinácia matíc K a M, teda

$$C = \alpha M + \beta K. \tag{3.28}$$

3.4 Newmarkova iteračná metóda

3.4.1 Popis metódy

Newmarkova metóda je implicitná metóda numerickej integrácie, ktorá sa dá použiť pre riešenie pohybových diferenciálných rovníc. Metóda predpokladá, že riešenie je hľadané pre konečný počet bodov $m + 1$ v časových okamihoch $t_0, t_1, ..., t_m$. Rozdiel medzi dvoma okamihmi v čase sa nazýva dĺžka integračného kroku Δt. Metóda popisuje vzťahy medzi posunom, rýchlosťou a zrýchlením hmotného bodu v čase t a v čase $t+\Delta t$. Táto závislosť je vyjadrená ako [11]

$$\dot{r}_{t+\Delta t} = \dot{r}_t + [(1-\delta)\ddot{r}_t + \delta\ddot{r}_{t+\Delta t}]\Delta t, \tag{3.29}$$

$$r_{t+\Delta t} = r_t + \Delta t\dot{r}_t + \left[\left(\frac{1}{2} - \alpha\right)\ddot{r}_t + \alpha\ddot{r}_{t+\Delta t}\right]\Delta t^2. \tag{3.30}$$

Vhodným výberom parametrov α a δ sa dá docieliť stabilita metódy. Použitím $\delta = 1/2$ a $\alpha = 1/4$, obdržíme metódu konštantného priemerného zrýchlenia. Dosadením týchto závislostí do rovnice (3.26) sa odvodí vzťah

$$\begin{aligned}\left(M + \delta\Delta tC + \alpha\Delta t^2K\right)\ddot{r}_{t+\Delta t} &= -Kr_t - (C + \Delta tK)\dot{r}_t \\ &- \left[(1-\delta)C + \left(\frac{1}{2} - \alpha\right)\Delta tK\right]\Delta t\ddot{r}_t + R_{t+\Delta t}.\end{aligned} \tag{3.31}$$

Výpočtom zrýchlenia $\ddot{r}_{t+\Delta t}$, sa dá následne vypočítať rýchlosť bodu $\dot{r}_{t+\Delta t}$ a vektor jeho posunov $r_{t+\Delta t}$, použitím (3.29) (3.30). Opakovaním tohto procesu m-krát

sa získa riešenie rovnice na intervale $< 0, t_m >$. O výraze na ľavej strane rovnice (3.31)

$$\hat{K} = \left(M + \delta\Delta t C + \alpha\Delta t^2 K\right) \ddot{r}_{t+\Delta t}, \tag{3.32}$$

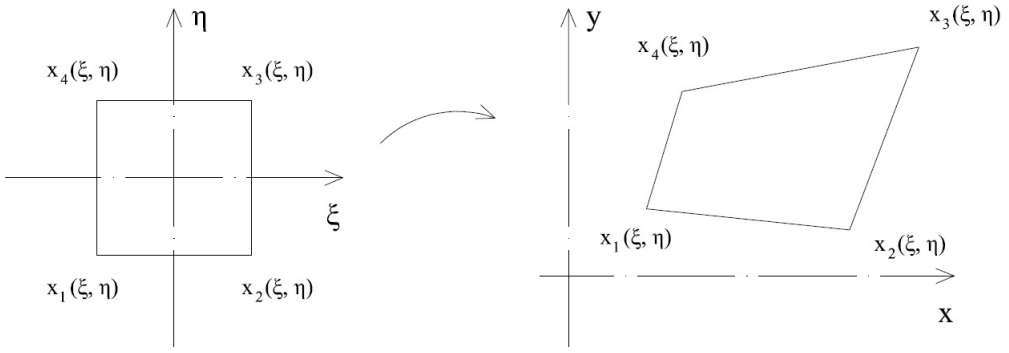
sa dá povedať, že ak časový krok Δt zostane konštantný počas procesu výpočtov, tak aj matica $\hat{K}$ zostáva konštantná. Táto vlastnosť je veľmi výhodná pre ušetrenie výpočtového času, a to hlavne pri použití GPU. Práve z tohto dôvodu bol počas výpočtov časový krok použitý vždy ako nemenný.

3.4.2 Transformácia súradníc

Ako bolo spomenuté v 3.3.8 je výhodné použitie aproximačných funkcií v prirodzených súradniciach. Vzniká tak požiadavka na nájdenie spôsobu, ako definovať prirodzené súradnice (ξ, η) ako funkciu (x, y). To sa dá dosiahnuť pomocou predpokladu, že geometria prvku bude aproximovaná podobne ako neznáme posuny

$$x(\xi, \eta) = \sum_{i=1} N_i(\xi, \eta) x_i^e, \quad y(\xi, \eta) = \sum_{i=1} N_i(\xi, \eta) y_i^e. \tag{3.33}$$

Aproximačné funkcie vystupujú v riešení aj vo forme derivácií. Určiť ich v prirodzených súradniciach je triviálne, ale je potrebné vyjadriť ich aj v závislosti na x a y.



Obr. 3.6: Jakobián slúži k vyjadreniu skutočnej geometrie pomocou prirodzených súradníc

Pomocou pravidla o derivácii zložených funkcií sa dá odvodiť vzťah

$$\begin{Bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \end{Bmatrix} = \overbrace{\begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix}}^{J} \begin{Bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{Bmatrix}, \tag{3.34}$$

kde J predstavuje jakobián transformácie, ktorý určuje vzťah medzi deriváciami v oboch súradnicových systémoch. Inverziou jakobiánu sa dá odvodiť vzťah pre deriváciu bázových funkcií podľa x a y

$$\begin{Bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{Bmatrix} = J^{-1} \begin{Bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \end{Bmatrix}, \tag{3.35}$$

kde sa dá jakobián spočítať zo vzťahu

$$J = \begin{bmatrix} \sum \frac{\partial N_i}{\partial \xi} x_i^e & \sum \frac{\partial N_i}{\partial \xi} y_i^e \\ \sum \frac{\partial N_i}{\partial \eta} x_i^e & \sum \frac{\partial N_i}{\partial \eta} y_i^e \end{bmatrix} = N^e X^e. \tag{3.36}$$

3.4.3 Numerická integrácia

Keďže slabé riešenie vyžaduje integráciu, ale tá sa nedá previesť analyticky pre väčšinu prípadov, bola pre získanie riešenia použitá numerická integrácia. Presnejšie Gaussova integrácia, a to z dôvodu jej vhodnej formulácie pre integrovanie polynómov. Základná myšlienka môže byť vyjadrená pomocou nasledujúceho integrálu a následnou aproximáciou ako

$$\hat{I} = \int_{-1}^{1} f(\xi) d\xi \approx \sum_{i=1}^{n} f(\xi_i) w_i, \tag{3.37}$$

Funkcia $\hat{I}$ je polynóm $(n-1)$ - vého stupňa s $2n$ parametrami. Ďalšou úpravou sa dá navýšiť stupeň aproximačného polynómu. To vedie k metóde, ktorá rieši integrály s vysokou presnosťou. Podmienkou takéhoto riešenia je použitie dostatočného počtu integračných bodov a ich váh. Pre riešenie problému bola využitá integrácia pomocou dvoch integračných bodov a im zodpovedajúcich váh. V prípade výpočtu matice tuhosti K^e bola použitá aj jednobodová integrácia, aby bolo možné znížiť vplyv šmykového uzamknutia prvku. Pre ukážku je uvedený príklad zápisu riešenia tejto matice

$$K^e = \sum_{i=1}^{n} \sum_{j=1}^{n} w_i w_j B^{eT}(\xi_i, \eta_j) D B^e(\xi_i, \eta_j) t |J(\xi_i, \eta_j)|. \tag{3.38}$$

3.5 Metóda združených gradientov

3.5.1 Popis

Kedže rovnica (3.31) je zložená z riedkych matíc, ktoré môžu byť veľkých rozmerov, je priame riešenie problému príliš neefektívne. Z tohto dôvodu bolo použité nepriame riešenie pomocou metódy združených gradientov. Zápis problému môže byť uvedený ako riešenie rovnice

$$Ax = b. \tag{3.39}$$

Iteračné metódy sú založené na myšlienke, že pre sústavu v špeciálnom tvare $(I - A)x = b$, platí pre $\forall b, x_0$, že postupnosť vektorov $x_k = \{k = 0, 1, 2, ...\}$ daná predpisom $x_{k+1} = Ax_k + b$ konverguje po súradniciach k vektoru $\hat{x}$, ktorý je riešením sústavy $(I - A)x = b$. [5]Metóda združených gradientov predpokladá symetricky pozitívne definitnú maticu A rozmeru nxn a dvojicu vektorov $u, v \in \mathbb{R}^n$, ktoré sú združené (konjungované), teda ich skalárny súčin je vzhľadom na maticu A rovný nule. Platí

$$\begin{aligned} u, v \in &\mathbb{R}^n \\ (Au, v) \equiv v^T Au &= 0. \end{aligned}$$

Definovaním vektorov $p_1, p_2, ..., p_n$, vzájomne združených sa dá vytvoriť báza preistoru $\mathbb{R}^n$. Teda riešenie $\hat{x}$ sústavy, $Ax = b$ sa dá vyjadriť ako lineárna kombinácia týchto vektorov p

$$\hat{x} = \sum_{i=1}^{n} a_i p_i. \tag{3.40}$$

Z rovností $A\hat{x} = b$ a $(p_k, p_i)_A = 0$ sa dá dosadiť do rovnice a pomocou úprav odvodiť vzťahy

$$\begin{aligned} A\hat{x} &= \sum_{i=1}^{n} a_i A p_i, \\ p_k^T A\hat{x} &= \sum_{i=1}^{n} a_i p_k^T A p_i, \\ p_k^T b &= \sum_{i=1}^{n} a_i (p_k, p_i)_A, \\ (p_k, b) &= a_k (p_k, p_k)_A. \end{aligned}$$

Z toho dostaneme

$$a_k = \frac{(p_k, b)}{(p_k, p_k)_A}. \tag{3.41}$$

Pre x_0 ako počiatočnú aproximáciu $Ax_0 \neq b$, a $p_0 = r_0 = b - Ax_0$, a $k = 0, 1, ..., n-1$ sa dá použiť predpis

$$a_k = \frac{(r_k, r_k)}{(Ap_k, p_k)}, \tag{3.42}$$

$$x_{k+1} = x_k + a_k p_k, \tag{3.43}$$

$$r_{k+1} = r_k - a_k A p_k, \tag{3.44}$$

$$b_k = \frac{(r_{k+1}, r_{k+1})}{(r_k, r_k)}, \tag{3.45}$$

$$p_{k+1} = r_{k+1} + b_k p_k, \tag{3.46}$$

kde pre dostatočne malé r_{k+1} je x_{k+1} riešením sústavy.

3.5.2 Využitý algoritmus

Na základe obmedzených skúseností s grafickým programovaním a časových obmedzení bol pre riešenie problému použitý už vytvorený algoritmus. Presnejšie, software vytvorený Christopherom Fougnerom [2]. Algoritmus využíva metódu združených gradientov.

3.6 Popis použitého algoritmu

Táto kapitola sa venuje bližšiemu popisu zostrojeného algoritmu. V texte sa budú nachádzať len ukážky zdrojového kódu, celý okomentovaný kód je priložený k práci.

3.6.1 Hardwarové a softwarové nároky

Pre úspešné spustenie kódu je nevyhnutné, aby mal počítač grafickú kartu NVIDIA podporujúcu technológiu CUDA. Teda musí mať dedikované CUDA jadrá. Softwarovo je potrebné prostredie CUDA Toolkit so zahrnutou knižnicou cuBLAS, v niektorých prípadoch je nutné ručne začleniť potrebné adresáre do samotnej knižnice cuBLAS. Z dôvodu negatívnych skúseností s implementáciou

prostredia CUDA sú k práci priložené stručné poznámky ako zahrnúť všetky potrebné knižnice. Ako prekladač jazyka C++ bol použitý Microsoft Visual Studio 2017 (MVS2017), kedže je predvoleným prekladačom softwaru CUDA. MVS musí obsahovať knižnicu stdlib.h, ktorá nie je automaticky nainštalovaná s najnovšiou verziou programu. Riešenie je tak isto popísané v prílohe.

3.6.2 Štruktúra kódu

Zdrojový kód sa skladá zo súboru konzolovej aplikácie Dynamics.cpp, v ktorom sú zadefinované všetky pomocné algoritmy, a jemu prislúchajúceho hlavičkového súboru. Telom algoritmu je aplikácia prostredia CUDA s názvom kernel.cu. V hlavičkovom súbore cgls.cuh je zadefinovaný algoritmus metódy združených gradientov.

3.6.3 Práca s maticami

Pre základné operácie s maticami a vektormi boli v prostredí MVS zostrojené pomocné funkcie, ktoré definujú operácie ako súčet matíc, násobenie matice vektorom a iné. Táto alternatíva bola zvolená pre nadobudnutie programovacích skúseností a pre lepší prehľad nad vnútorným fungovaním programu. Základným problémom týchto operácií bola práca s dynamickou pamäťou. Preto boli matice aj vektory definované podľa šablony (template) vector. Tie na rozdiel od radov (array) nevyžadujú predom definované rozmery dimenzií matíc. Pri lokalizácii matice ako vektor z vektorov sa dá dimenzia matice upraviť dynamicky. Nedostatočné skúsenosti s pokročilou algoritmizáciou mali za následok zlú optimalizáciu týchto výpočtov. Tento faktor ovplyvnil aj rozsah prevedených výpočtov.

3.6.4 Vstupné údaje

Vstupy do výpočtu už boli vymenované v časti3.3. Tento odsek sa venuje iba popisu vstupných parametrov geometrie a zaťaženia. Súradnice je nutné zadávať ako dvojice hodnôt x a y, takisto je potrebné ručne zadávať jednotlivé uzly každého prvku. Čo vedie k veľkej zložitosti zadávania konštrukcií komplikovaných tvarov. Pre generáciu štvoruholníkových prvkov bol zostrojený jednoduchý algoritmus založený na myšlienke delenia dĺžok strán na požadovaný počet prvkov. Pomocou algoritmu je možné zostrojiť aj vektor uzlov prvkov. Podpory sú zadávané do jednotlivých uzlov. Zadávanie síl funguje na rovnakom princípe, ale

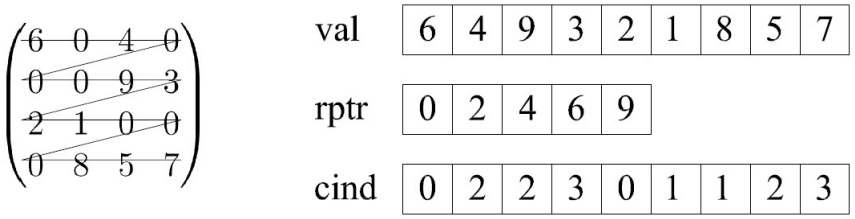
je ešte potrebné určiť ich smer a veľkosť. Toto riešenie zjednodušuje užívateľovi prístup k aplikácii, ale vyžaduje znalosť číslovania siete.

3.6.5 Výpočty na CPU

Pre zostavenie matíc tuhosti K a hmotnosti M je potrebné veľké množstvo dielčich výpočtov. Preto boli výpočty sprostredkované pomocou CPU. Proces pozostáva z výpočtu matíc K^e a M^e každého prvku a ich následnou alokáciou do globálnej matice K resp. M. Z matíc boli následne vytvorené submatice zložené iba z rovníc zodpovedajúcich neznámym posunom. To sa docielilo odobratím riadkov a stĺpcov prislúchajúcich uzlom, v ktorých su predpísané podpory. Rovnakým spôsobom bol upravený aj vektor zaťaženia. Použitím submatíc bola zostavená matica pravej strany $\hat{K}$. Vďaka konštantnému integračnému kroku Δt zostáva táto matica v priebehu výpočtov nemenná, a preto ju nie je potrebné opäť prepočítavať. Ďaším krokom bolo určenie pravej strany rovnice (3.26). Ako počiatočné hodnoty posunov, rýchlostí a zrýchlení boli zvolené nulové vektory. Výpočet pravej strany je potrebné zopakovať v každom časovom intervale. Aj tieto výpočty boli vykonané na CPU. Pred presunutím dát z CPU na grafickú kartu bola potrebná kompresia matíc K a M do formátu CSR.

3.6.6 CSR formát matice

CSR alebo compressed row storage formát predstavuje zápis riedkych matíc do tvaru troch polí informácií. V poli *val* sú uložené hodnoty všetkých nenulových prvkov matice usporiadaných hierarchicky od $a_{ii}...a_{ij}$, až $a_{ji}...a_{jj}$. Pole *rptr* obsahuje ukazovatele na tie prvky poľa *val*, ktoré sú prvým nenulovým členom v každom riadku. Posledné pole *cind* priradzuje každému pvku z *val* jeho stĺpcovú súradnicu. Pri kompresii sú spočítané aj čísla m, n a nnz, ktoré určujú dimenzie matice a celkový počet nenulových prvkov. Tieto údaje sú potrebné pre prealokáciu pamäte na GPU.



Obr. 3.7: Ukážka tvaru matice v CSR formáte

3.6.7 Využitie GPU

Na GPU sa vykonával algoritmus metódy združených gradientov, teda výpočet riešenia rovnice (3.31). Ako už bolo uvedené, algoritmus bol prevzatý z [2], ktorý bol poskytnutý k voľnému použitiu. Práca s GPU sa zásadne odlišuje od CPU hlavne v manažmente pamäte grafickej jednotky. Pred podrobnejším vysvetlením je potrebné definovať pojmy Host (hostiteľ) a Device(zariadenie). V tomto prípade predstavuje Host CPU počítača a Device grafický procesor. Pre spustenie výpočtov na GPU je potrebné najprv alokovať priestor v pamäti karty. Tento priestor bude využitý pre ukladanie vstupných a výstupných dát, jeho veľkosť je počas behu algoritmu nemenná. Práve z tohto dôvodu bolo potrebné získať hodnotu nnz, na základe ktorej sa určí veľkosť potrebnej pamäte. Predpokladá sa, že pre uloženie matice a vektrov b a x bude potrebných $val_d = (nnz+m+n)*double$ bitov, kde double predstavuje typ zápisu čísla s desatinnou čiarkou. Vektor x je síce neznámy, ale pre uloženie výsledkov je potrebné mať prealokvaný priestor. Vstupné dáta je potom potrebné skopírovať z Host na Device. Následne sa môže vykonať spustenie programu na GPU, ten ukladá výsledky do taktiež predalokovanej pamäte. Výsledky treba následne opäť skopírovať z Device na Host a vymazať predalokovanú pamäť karty. Pri bežných počtoch na CPU nie je problém ukladania matice prítomný, pretože sa stačí odvolávať na hodnoty matice priamo. GPU má však obmedzený priestor pre ukladanie dát a priamy prístup k dátam na CPU je nerealizovateľný kvôli obmedzeniam zbernice. Fakt, že matica $\hat{K}$ je počas výpočtov konštantná je pre výpočet veľmi výhodný. Jeho využitím je to, že alokácia a kopírovanie matice prebehne iba raz počas prvého kroku výpočtu. Následne sa v pamäti prepisujú iba hodnoty vektora pravej strany, v zdrojovom kóde označovaného b. Ten sa po každom časovom kroku prepočíta využitím Newmarkovej metódy. Do algoritmu bola pridaná aj funkcia merania času medzi prepočtami, ktorá slúži k vyhodnoteniu výsledkov. Pre názornú ukážku sa uvádza časť zo zdrojového kódu.

```
// Inicializácia dát
real_t* val_h = val;
int* cind_h = cind;
int* rptr_h = rptr;
real_t* b_h = b;
real_t* x_h = x_;
real_t *val_d, *b_d, *x_d;
int *cind_d, *rptr_d;

// Alokácia priestoru na grafickej karte
cudaMalloc((void**)&val_d, (nnz + m + n) * sizeof(real_t));
cudaMalloc((void**)&cind_d, (nnz + m + 1) * sizeof(int));

rptr_d = cind_d + nnz;

  // Alokácia priestoru na grafickej karte
cudaMemcpy(val_d, val_h, nnz * sizeof(real_t), cudaMemcpyHostToDevice);
cudaMemcpy(cind_d, cind_h, nnz * sizeof(int), cudaMemcpyHostToDevice);
cudaMemcpy(rptr_d, rptr_h, (m + 1) * sizeof(int), cudaMemcpyHostToDevice);

for (int it = 0; it < Num_ts; it++) {

  // Spustenie timeru
float elapsed = 0;
cudaEvent_t start, stop;

HANDLE_ERROR(cudaEventCreate(&start));
HANDLE_ERROR(cudaEventCreate(&stop));

HANDLE_ERROR(cudaEventRecord(start, 0));

b_d = val_d + nnz;
x_d = b_d + m;

// Kopírovanie dát z HOST na DEVICE, teda z CPU na GPU
cudaMemcpy(b_d, b_h, m * sizeof(real_t), cudaMemcpyHostToDevice);
cudaMemcpy(x_d, x_h, n * sizeof(real_t), cudaMemcpyHostToDevice);

// Spustenie iteračného riešiča
int flag = cgls::Solve<real_t, cgls::CSR>(val_d, rptr_d, cind_d, m, n,
```

```
nnz, b_d, x_d, shift, tol, maxit, quiet);

// Kopírovanie dát z DEVICE na HOST, teda z GPU na CPU
cudaMemcpy(x_h, x_d, n * sizeof(real_t), cudaMemcpyDeviceToHost);

// Ukončenie timeru
HANDLE_ERROR(cudaEventRecord(stop, 0));
HANDLE_ERROR(cudaEventSynchronize(stop));

HANDLE_ERROR(cudaEventElapsedTime(&elapsed, start, stop));

HANDLE_ERROR(cudaEventDestroy(start));
HANDLE_ERROR(cudaEventDestroy(stop));

}

// Vymazanie pamäti na konci iteračného procesu
cudaFree(val_d);
cudaFree(cind_d);
```

3.6.8 Grafický výstup

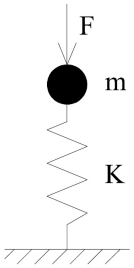
Grafické vyobrazenie výsledkov sa prevádzalo pomocou softwaru Matlab. Aby sa uľahčila práca s výsledkami bol konzolový výstup aplikácie nastavený tak, aby vypisoval výsledky priamo v tvare zápisu vykresľovacích funkcií Matlabu. Takýto výpis nebol aktivovaný počas merania výkonu GPU, aby neskresľoval výsledky. Prostredie Matlab bolo tiež použité pre kontrolu správnosti riešení aplikácie.

Kapitola 4

Výsledky

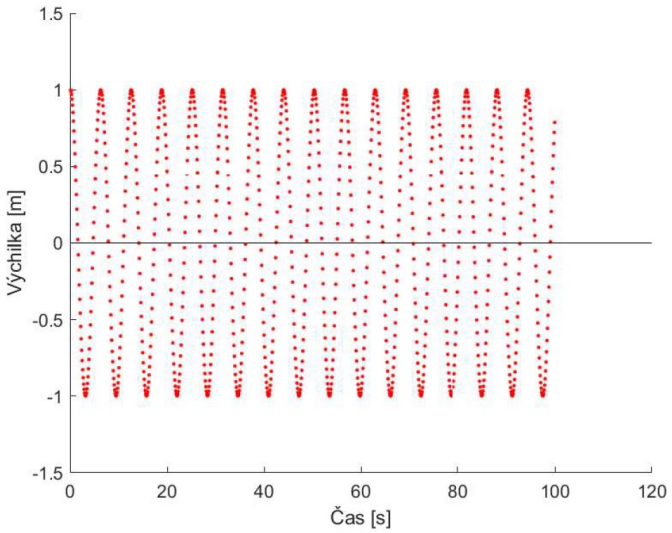
4.1 Hmotný bod

Ako prvý príklad pre demonštráciu funkčnosti programu bol vytvorený model hmotného bodu s jedným stupňom voľnosti. Bol umiestnený do roviny so súradnicami [1,0]. Bodu bola definovaná hmotnosť $m = 1$ $[kg]$ a tuhosť K s hodnotu 1 $[kN/m]$, tá abstraktne predstavuje pružinu, na ktorej je bod zavesený.



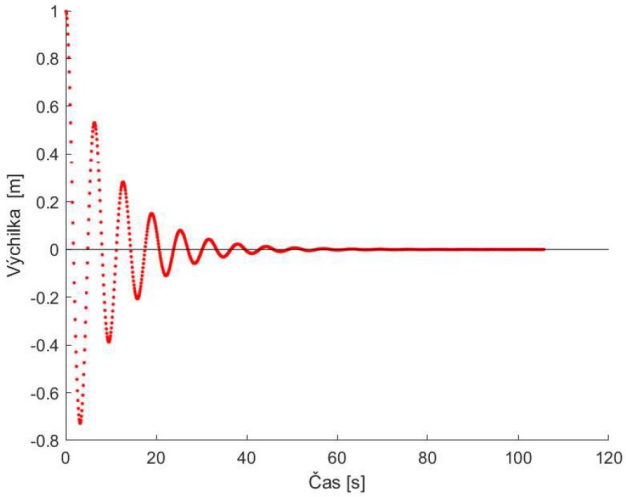
Obr. 4.1: Grafická schéma zadania

Bod bol následne zaťažný silou o veľkosti $F = 1$ $[kN]$. Odozva bodu sa merala počas 1000 časových krokov s dĺžkou Δt 0.1 sekundy. Na konci každého kroku sa do grafu zakreslila jeho nová poloha spočítaná ako suma výsledného posunu $r_{t+\Delta t}$ a jeho pôvodnej polohy y.



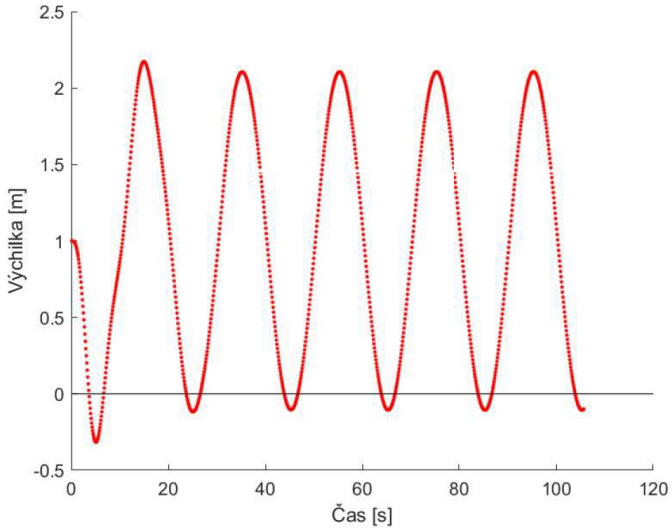
Obr. 4.2: Netlmené kmitanie bodu

Z obrázku je zrejmé, že bod osciluje okolo hodnoty statickej výchylky, teda osy x. Pre ukážku vplyvu tlmiacich síl boli určené aj konštanty $\alpha = 0.1$ a $\beta = 0.1$, ktoré definujú tlmenie C ako lineárnu kombináciu $\alpha K + \beta M$. Ostatné charakteristiky zostali nemenné.



Obr. 4.3: Tlmené kmitanie bodu

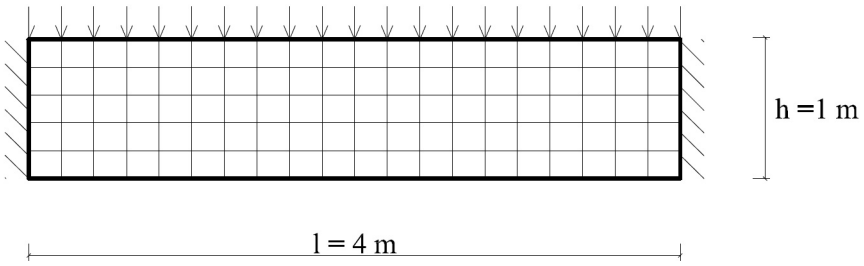
Ako ďalšia ukážka bol bod zaťažený budiacou silou s rovnakou veľkosťou, no so zadanou budiacou frekvenciou $\omega = 1$ $[Hz]$. Rýchlosť jednotlivých operácií pri každom výpočte je príliš malá na to, aby sa dalo uviesť porovnanie medzi výpočtom na CPU a GPU.



Obr. 4.4: Tlmené kmitanie bodu s budiacou silou

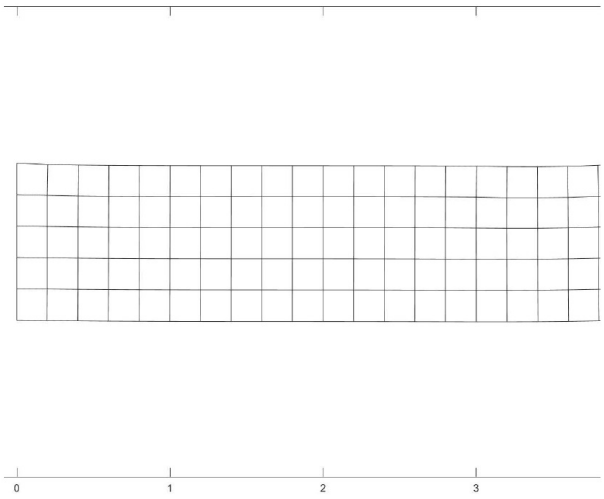
4.1.1 Votknutý nosník

Pre ilustráciu väčšieho problému bol uvažovaný votknutý nosník s rozponom $l = 2.4$ $[m]$, výškou $h = 0.8$ $[m]$ a hrúbkou $t = 0.1$ $[m]$. Jeho materiálové charakteristiky boli zvolené tak, aby odpovedali charakteristikám ocele. Na nosníku bola zostrojená sieť bodov, ktorá ho rozdeľovala na 48 rovnakých štvorcových prvkov s dĺžkou hrany 0.2 m. Nosník bol zaťažený bodovými silami $F = -100$ $[kN]$ vo zvislom smere v každom z bodov hornej hrany. Pre lepšiu predstavu sa uvádza grafické interpretovanie problému

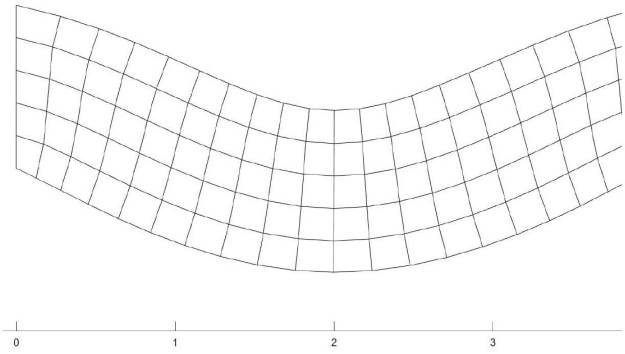


Obr. 4.5: Schéma votknutého nosníku

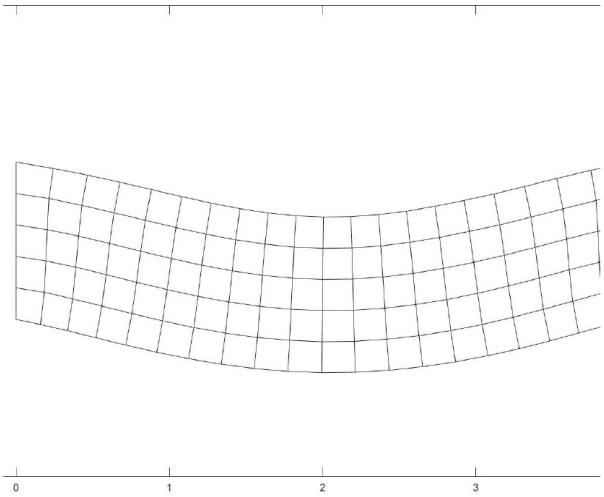
Merania opäť prebehli počas 250 krokov s dĺžkou $\Delta t = 0.1$ $[s]$. Počas výpočtov sa počítalo aj s tlmením konštrukcie. Kedže sa jedná o dynamický problém, jeho grafické znázornenie v práci je obmedzené. Na obrázkoch je možné vidieť kmitanie nosníka.



Obr. 4.6: Okamih zaťaženia nosníka

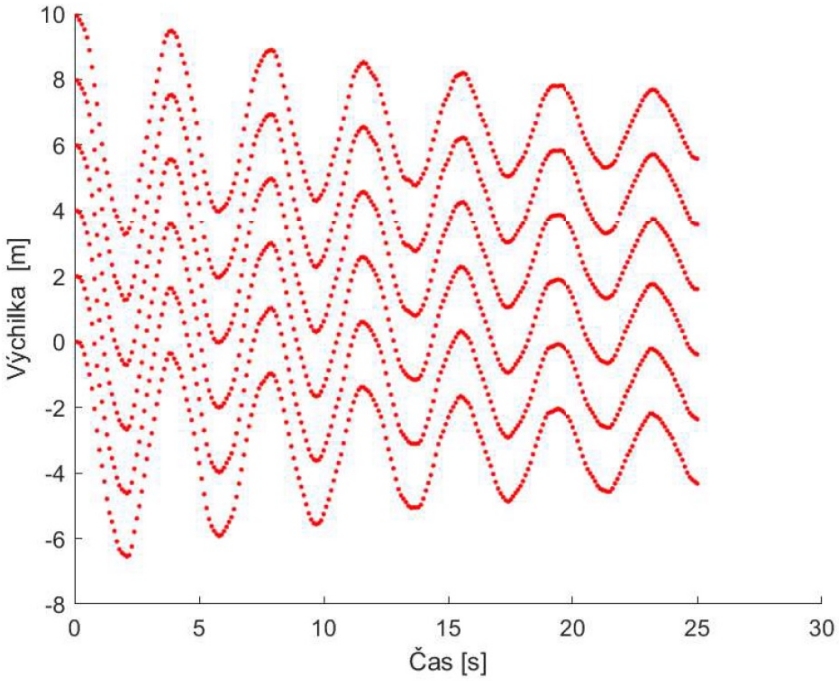


Obr. 4.7: Amplitúda výchylky nosníka



Obr. 4.8: Ustálený nosník v tvare statického priehybu

Pre zretelnejšie vyobrazenie kmitania boli do grafu vynesené výchylky uzlov stredového prierezu nosníka.



Obr. 4.9: Kmitanie uzlov prierezu

Pri tomto príklade bol prvýkrát porovnaný čas výpočtu CPU a GPU. Porovnané sú doby výpočtov časových krokov. Tie boli získané z 20 meraní na každom zariadení. Uvedené sú iba priemerné hodnoty všetkých meraní. A to doba výpočtu jedného kroku na CPU $t_{CPU} = 1.34\ [ms]$ a pre GPU $t_{GPU} = 1.34\ [ms]$.

4.1.2 Metodika merania a obmedzenia výpočtov

Prvé výpočty prebehli len na malých konštrukciách s menej ako 200 stupňami voľnosti. Po experimentoch s vačším počtom sa začali objavovať problémy s dĺžkou výpočtového času. Výpočet modelu s vyše 20 000 stupňami voľnosti musel byť zastavený po vyše hodine bez toho, aby bol získaný jediný výstup. Metódou pokusu a omylu bola stanovená pomyselná hranica veľkosti problému s časovo vyhovujúcimi nárokmi. Aby bolo možné nahromadiť dostatok dát, bola táto hranica určená pre konštrukcie skladajúce sa z menej ako 500 prvkov, teda s 1000 stupňami voľnosti. Pri tejto veľkosti je doba riešenia cyklu s 10 časovými krokmi približne 20 minút. Prvým predpokladom tejto limitácie bol slabý výkon grafickej karty počítača. Bližším skúmaním parametrov výpočtu sa však ukázalo, že

problém vychádza zo zlej optimalizácie algoritmu bežiaceho na CPU. Celý kód je veľmi neefektívny a práve výpočty matíc K a M zaberajú vyše 95% času potrebného pre výpočet. Z toho vychádza problematika metodiky vyhodnocovania výsledkov. Pôvodné dáta boli namerané ako čas od nakopírovania matice $\hat{K}$ do konca výpočtu kroku, kedy prebieha algoritmus metódy združených gradientov, ale prebieha aj prepočet vektora pravej strany $\boldsymbol{b}$. Tento proces je sprostredkovaný pomocou CPU, keďže aj táto časť kódu je slabo optimalizovaná, dochádza k spomaľovaniu jednotlivých krokov. Toto spomalenie je v porovnaní so samotnou rýchlosťou GPU razantné. Priemerná doba výpočtu jedného kroku sa zväčší približne 60-krát. Novou metodikou bolo meranie času od uloženia matice $\hat{K}$ do okamihu skopírovania výsledkov z GPU naspäť na CPU. Pôvodne bolo zamýšľané porovnávať výsledky GPU s výsledkami získanými pomocou toho istého algoritmu, ale s funkciou združených gradientov sprostredkovanou pomocou CPU. Neefektívnosť tohto algoritmu, ale viedla k tomu, že bol pre porovnanie použitý software Matlab. Tieto výsledky sú považované za relevantnejšie, pretože presnejšie znázorňujú výkon CPU. Avšak aj v prostredí Matlab je práca s veľkým počtom dát zdĺhavá.

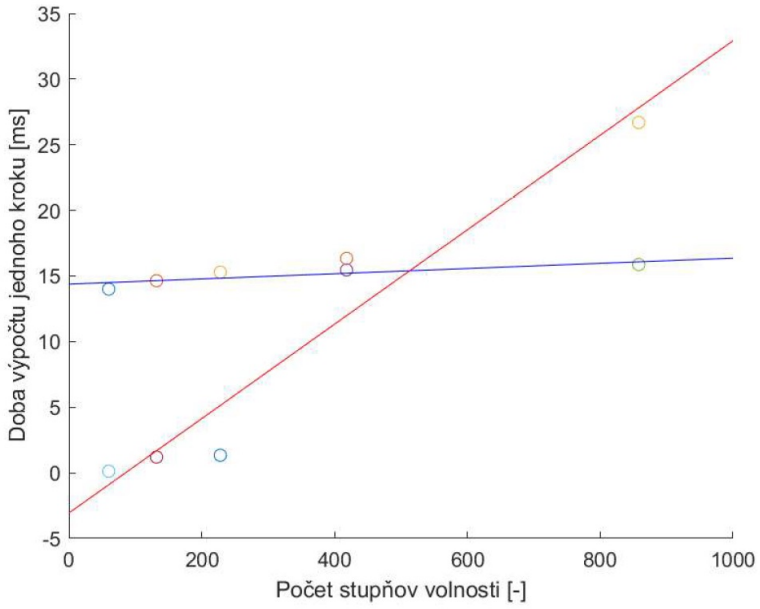
4.1.3 Výsledky meraní

Výsledky boli získané použitím algoritmu GPU a výsledky pre porovnanie pomocou CPU. Všetky merania boli vykonané na rovnakej konštrukcii, a to na votknutom nosníku s rozponom $l = 4$ $[m]$, výškou $h = 1$ $[m]$ a hrúbkou $t = 1$ $[m]$. Ten bol postupne delený na väčšie množstvá konečných prvkov. Tie sa rámcovo pohybovali v desiatkach až stovkách. Algoritmus potom previedol desať krokov s dĺžkou $\Delta t = 0.1$ $[s]$. Každý príklad bol prepočítaný päťkrát, aby sa získal väčší počet výsledkov, ktoré sa následne spriemerovali. Pri každej operácii na GPU boli merané hodnoty, a to čas potrebný pre vykonanie výpočtu riešenia, celkový čas riešenia a čas výpočtu prvého časového kroku. Doba prvého výpočtu sa v každom meraní vymykala z priemerných hodnôt času potrebného pre vykonanie výpočtu. Časové oneskorenie je pravdepodobne zapríčinené ešte prebiehajúcim kopírovaním matice ľavej strany. Preto nebol prvý výpočet zahrnutý do výpočtu priemerného trvania kroku. Predpokladá sa, že pri väčšom počte opakovaní, napr. presahujúcom 1000 cyklov, by sa táto hodnota stala v priemere zanedbateľnou. Naviac by skresľovala výsledky, ktoré boli zmerané na pozorovanie rýchlosti vykonávania opakovaných procesov. Na CPU bola taktiež meraná priemerná doba kroku a celkový čas riešenia. V tomto prípade sa ale doba prvého výpočtu nelíši od ostatných, teda odpadá nutnosť vyčleniť ju z výsledkov. Namerané hodnoty sú zobrazené v tabuľke 4.1.3.

	Výsledky CPU [ms]		Výsledky GPU [ms]	
p.s.v.	**Priem. kroku**	**Celk. čas**	**Priem. kroku**	**Celk. čas**
60	14.00	725.42	0.127	2.3
132	14.64	770.28	1.2	11.8
228	15.294	748.2	1.34	12.8
418	15.46	744.14	6.34	63.5
858	15.87	757.39	26.7	266.5

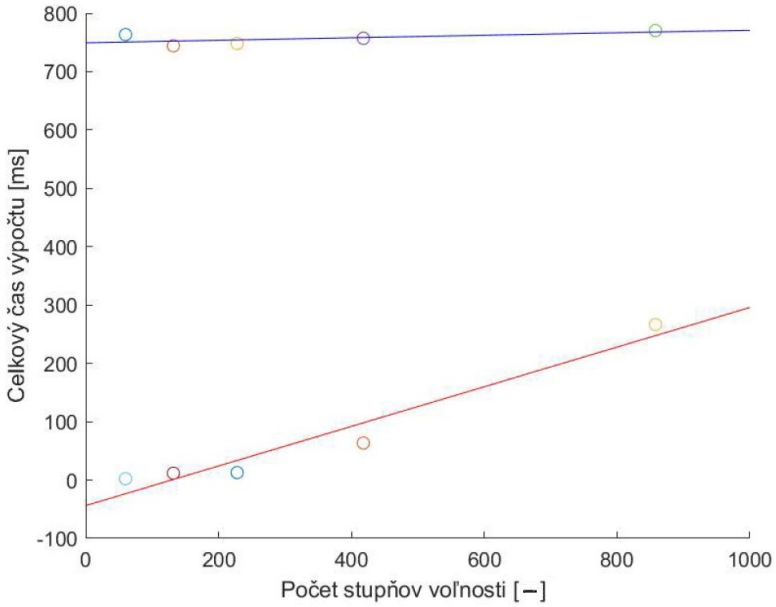
Tabuľka 4.1: Tabuľka výsledkov, Uvádza dobu výpočtov a riešení jednotlivých krokov v závislosti od počtu stupňov voľnosti p.s.v.

Výsledky sa vyniesli do grafu závislosti počtu stupňov voľnosti konštrukcie a priemernej doby riešenia kroku. Bodmi bol preložený polynóm prvého stupňa. Z dôvodu nízkeho počtu riešení veľkých problémov sa graf nedal preložiť polynómom vyššieho stupňa. Pri použití druhého alebo tretieho stupňa sa tvar krivky aproximoval do nuly alebo prudko konvergoval k nekonečnu. Graf bol extrapolovaný do hodnoty 1000 stupňov voľnosti.



Obr. 4.10: Priemerný čas jednoho kroku na GPU (modrá priamka), priemerný čas jednoho kroku na CPU (červená priamka)

V druhom grafe je uvedená závisloť medzi počtom stupňov voľnosti a celkovým časom procesu výpočtu. Od výsledkov GPU neboli odčítané časy prvých krokov. V tomto prípade sa vnímajú ako nevyhnutná súčasť riešenia, ktorá charakterizuje použite GPU. Nastáva to preto, že každé riešenie je ovplyvnené obmedzeniami pri prenose dát medzi zariadeniami.



Obr. 4.11: Celkový čas procesu výpočtu GPU (modrá priamka), celkový čas procesu výpočtu CPU (červená priamka)

Kvôli nedostatočnému počtu dát sa dá o výsledkoch hovoriť len ako o odhadoch. Zo získaných výsledkov vyplýva, že s narastajúcou komplexitou problému je využitie grafickej akcelerácie výhodným riešením. Pre riešenie menších problémov je však využitie GPU nevhodné, kedže vzniká nevyhnutné spomalenie výpočtov, ktoré je spôsobené neefektívnym prenosom dát medzi zariadeniami. Tieto tvrdenia sú podporené aj výsledkami štúdií využitia GPU pre riešenie komplexných problémov [5] [10]. S narastajúcou zložitosťou problémov sa zdá byť implementácia paralelného počítania na grafickej karte výhodným riešením.

4.1.4 Diskusia

Na základe nadobudnutých skúseností s paralelným programovaním je zrejmé, že využitie GPU má veľký potenciál pre riešenie komplexných problémov. Je potrebné dôkladnejšie štúdium grafického programovania, aby sa dali takéto riešenia implementovať. Výsledkom práce sú hlavne teoretické poznatky o fungovaní procesov na GPU a z nich vychádzajú rôzne námety na budúce rozšírenie práce. Hlavným problémom je optimalizácia riešenia pre prácu s veľkým množstvom informácií, a to napríklad použitím vhodných formátov kompresie matíc. Obmedziť ukladanie nepotrebných medzivýpočtov a zlepšiť manažment dát minimalizovaním počtu potrebných medzikrokov výpočtu ako napríklad alokácia matíc prvku do globálnej matice konštrukcie. Ďalším krokom by mala byť snaha o prenesenie čo najväčieho počtu výpočtov z CPU priamo na GPU. Rozdelenie algoritmov čo najefektívnejšie do kernelov, ktoré by mohli pracovať simultánne, ako napríklad, zostavovanie matíc K a M zároveň na rôznych multiprocesoroch karty. Zmeniť systém vkladania vstupov a vyhnúť sa tak prílišnej spotrebe pamäte. Napríklad tvorbou algoritmu, ktorý by odvodzoval kódové čísla uzlov prvku počas chodu aplikácie, namiesto ukladania veľkých objemov dát, ktoré slúžia len pre identifikáciu prvku. Po vyriešení základnej optimalizácie by sa mohlo rozšíriť spektrum využitia riešiča, a to napríklad zahrnutím riešení trojrozmerných problémov, alebo riešenia problémov iných odvetví mechaniky, ako napríklad šírenie tepla v priestore. Pre riešenie väčších problémov by bolo vhodné implementovať generátor sietí konečných prvkov. Kedže najväčšou predispozíciou grafických procesorov je vykresľovanie grafických modelov, bolo by užitočné implementovať tvorbu grafických výstupov priamo na GPU. Na druhú stranu sú riešenia týchto problémov zložité a vyžadujú veľké množstvo práce, preto je potrebné prehĺbenie znalostí danej problematiky.

Kapitola 5

Záver

Výsledkom práce je algoritmus určený pre modelovanie dvojrozmerných konštrukcií. Zahrňuje v sebe aj riešič založený na paralelizácii výpočtov pomocou grafického procesoru počítača. Dôsledkom nedostatočnej optimalizácie je neefektivita pre riešenie komplexných problémov, ktorá viedla k obmedzenému počtu získaných dát. Získané výsledky však poukazujú na výhody riešenia veľkých sústav lineárnych rovníc pomocou GPU. Práca hlavne pomohla vybudovať teoretický základ pre nasledovnú prácu v oblasti paralelizácie výpočtov.

Literatúra

[1] Bořek Patzák přednášky z numerické analýzy konstrukcí 1 a 2. https://mech.fsv.cvut.cz/student/. Accessed: 2019-16-05.

[2] Christopher Fougner cuda conjugate gradient solver. https://github.com/foges/cgls_cuda. Accessed: 2019-16-05.

[3] DELLspecification of used pc hardware. https://www.dell.com/en-au/shop/dell-laptops/inspiron-15-7000-gaming/spd/inspiron-15-7567-laptop. Accessed: 2019-16-05.

[4] HPspecification of cpu. https://ark.intel.com/content/www/us/en/ark/products/97185/intel-core-i7-7700hq-processor-6m-cache-up-to-3-80-ghz.html. Accessed: 2019-16-05.

[5] Ján Chleboun přednášky z matematiky 4. https://mat.fsv.cvut.cz/chleboun/JCh_vyuka/101MA4/MA4_odkazy.htm. Accessed: 2019-16-05.

[6] NVIDIAnvidia tesla p100 whitpaper. https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf. Accessed: 2019-17-05.

[7] NVIDIAspecification of gpu. https://www.nvidia.com/en-gb/geforce/products/10series/geforce-gtx-1050/. Accessed: 2019-16-05.

[8] vlask20 let vývoje grafických karet: od cga až po konec 3dfx. https://www.cnews.cz/20-let-vyvoje-grafickych-karet-od-cga-az-po-konec-3dfx-dil-i/. Accessed: 2019-17-05.

[9] Clifford Truesdell & Walter Noll; Stuart S. Antman. *The Non-linear Field Theories of Mechanics*. Springer, 2004.

[10] Hongyu Li, Jun Teng, Z.-H Li, and Lu Zhang. Nonlinear dynamic analysis efficiency by using a gpu parallelization. *Engineering Letters*, 23:232–238, 11 2015.

[11] Ing Jiří Šejnoha Zdeněk Bittnar. *Numerické metody mechaniky 1.* Vydavatelství ČVUT, Zikova 4, 16635 Praha 6, 1992.