VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra aplikované matematiky

Vizualizace diskretizovaných polí ve 2 dimenzích
Visualisation of Discretized Vector Fields on Surfaces

2010 Milan Váhala

vložit zadání

Prohlášení:

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Datum a vlastnoruční podpis

Poděkování:

Chtěl bych touto cestou poděkovat vedoucímu své diplomové práce Ing. Daliborovi Lukášovi, Ph. D. za odbornou pomoc.

Abstrakt

Tato práce se zabývá vizualizací skalárních a vektorových polí na oblastech v rovině, které jsou rozděleny na malé dílky. Tato pole získáme jako výsledky mnoha různých úloh z praxe. K řešení těchto úloh se často používá metoda konečných prvků. Cílem této práce je vytvořit software, který skalární a vektorová pole efektivně vizualizuje.

Klíčová slova

Metoda konečných prvků, Qt, GNU Octave.

Abstract

This thesis deals with visualisation of scalar and vector fields on regions in plane divided into small pieces. This fields we get as results of many various practical problems. Finite elements method is often used to solve this problems. Goal of this thesis is to develop a software for effective visualization of scalar and vector fields.

Keywords

Finite element method, Qt, GNU Octave.

Seznam použitých symbolů a zkratek

MKP	Metoda konečných prvků
$C^n(\Omega)$	Množina všech funkcí z Ω do $\mathbb{R}$ majících na Ω spojité všechny derivace až do n-tého řádu včetně
GUI	Grafické uživatelské rozhraní (Graphical user interface)

Obsah

Úvod.

Cílem této práce je vytvořit software, který vizualizuje rovinná skalární a vektorová pole, která získáme jako řešení různých praktických úloh.

První kapitola pojednává o metodě konečných prvků, jejíž výsledky jsou skalární a vektovová pole, která hodláme vizualizovat. V této kapitole zavádíme pojem triangulace, tj. rozdělení oblasti na malé dílky, na kterých hledaná skalární a vektorová pole aproximujeme jednoduchou funkcí.

Druhá kapitola pojednává způsobech, kterými lze skalární a vektorová pole vizualizovat prostředky počítačové grafiky.

Ve třetí kapitole si ukážeme vizualizaci výsledků různých praktických úloh. K vizualizaci použijeme prostředky popsané v předchozí kapitole, které jsou implementovány programem Varan, jehož vytvoření je cílem této práce.

V posledních dvou kapitolách popíšeme vlastnosti programu Varan a některé postupy, které jsou použity při jeho implementaci v prostředí frameworku Qt.

1. Metoda konečných prvků

Metoda konečných prvků (MKP) je numerická metoda pro řešení parciálních diferenciálních rovnic. Potřeba vizualizovat výsledky např. této metody je motivací k vytvoření programu Varan, o kterém pojednává tato práce.

Nejprve si ukážeme MKP v jednorozměrném případě a poté analogicky ve dvou rozměrech. V obou případech použijeme Dirichletovu okrajovou podmínku na hranici.

1.1. Jednorozměrný problém.

Uvažujme okrajovou úlohu

$$\begin{cases} -u''(x) = f(x), \quad x \in (0,1), \\ u(0) = u(1) = 0, \end{cases} \tag{1.1}$$

kde $f \in L^2(0,1)$ (viz [4]). Odvodíme slabou (variační) formulaci této úlohy. Pro každou funkci $v \in C^1(\langle 0,1\rangle)$ takovou, že $v(0) = v(1) = 0$ platí

$$\int_0^1 -u''(x)v(x)\ dx = \int_0^1 f(x)v(x)\ dx.$$

Odtud integrací per partes dostáváme

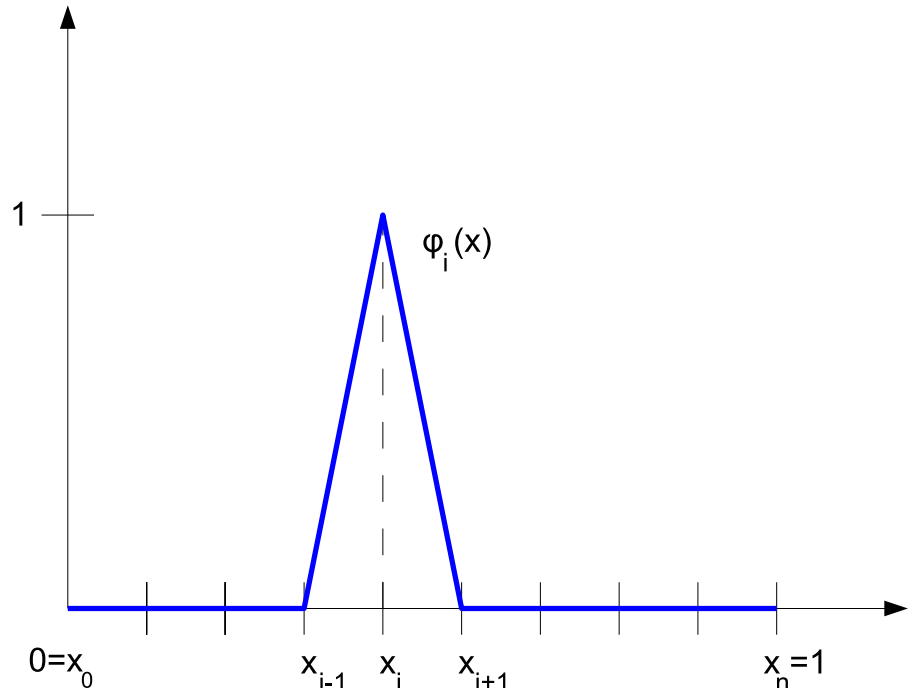
$$\int_0^1 -u''(x)v(x)\ dx = [-u'(x)v(x)]_0^1 + \int_0^1 u'(x)v'(x)\ dx = \int_0^1 u'(x)v'(x)\ dx.$$

Poslední rovnost platí i pro u, v patřící do prostoru $H_0^1(0,1)$ (viz [1]), na kterém definujeme slabé řešení úlohy (1.1) jako $u \in H_0^1(0,1)$ takové, že pro všechna $v \in H_0^1(0,1)$ platí

$$\int_0^1 u'(x)v'(x)\ dx = \int_0^1 f(x)v(x)\ dx. \tag{1.2}$$

Úlohu nalezení tohoto slabého řešení na prostoru $H_0^1(0,1)$ aproximujeme úlohou, ve které nahradíme prostor nekonečné dimenze $H_0^1(0,1)$ některým jeho podprostorem V konečné dimenze. Prostor V definujme následujícím způsobem. Zvolme body $x_0, x_1, \ldots, x_n$ tak, že $0 = x_0 < x_1 < \cdots < x_n = 1$. Interval $\langle 0,1\rangle$ rozdělíme na n dílků $\langle x_i, x_{i+1}\rangle$, $i = 0, 1, \ldots, n-1$. Nechť mají, pro jednoduchost, všechny intervaly $\langle x_i, x_{i+1}\rangle$ stejnou délku $h = 1/n$. Zavedeme $\varphi_1(x), \varphi_2(x), \ldots, \varphi_{n-1}(x)$ bázové funkce prostoru V tak, že $\varphi_i(x_i) = 1$, $\varphi_i(x_j) = 0$ pro $j \neq i$ a $\varphi(x)$ je lineární na každém intervalu $\langle x_i, x_{i+1}\rangle$, kde $i = 0, 1, \ldots, n-1$. Obecně tímto způsobem zavádíme i bázové funkce $\varphi_0(x)$ a $\varphi_n(x)$, ale v tomto případě je vynecháme kvůli

splnění Dirichletovy okrajové podmínky $u(0) = u(1) = 0$. Jak vypadá taková funkce $\varphi_i(x)$ můžeme vidět na obr. 1.1.



Obrázek 1.1: *Příklad bázové funkce pro jednorozměrnou úlohu.*

Libovolný prvek $u(x)$ prostoru V vyjádříme jako

$$u(x) = \sum_{i=1}^{n-1} \alpha_i \varphi_i(x).$$

Snadno si rozmyslíme, že graf takovéto funkce $u(x)$ je lineární interpolací bodů $(0,0), (x_1, \alpha_1), \ldots, (x_i, \alpha_i), \ldots, (1,0)$.

Metodou konečných prvků hledáme $u \in V$ takové, že pro všechna $v \in V$ platí (1.2). Označíme

$$u(x) = \sum_{i=1}^{n-1} \alpha_i \varphi_i(x), \quad v(x) = \sum_{i=1}^{n-1} \beta_i \varphi_i(x).$$

Rovnost (1.2) je pro takto zvolená $u(x)$ a $v(x)$ ekvivalentní s úlohou

$$\forall j \in \{1, 2, \ldots, n-1\} : \int_0^1 u'(x)\varphi_j'(x)\, dx = \int_0^1 f(x)\varphi_j(x)\, dx. \tag{1.3}$$

Implikaci „⇒“ dostaneme vhodným zvolením koeficientů β_i tak, že $\beta_j = 1$ a $\beta_i = 0$ pro $i \neq j$. Opačnou implikaci „⇐“ dostaneme takto:

$$\int_0^1 u'(x)v'(x)\ dx = \sum_{i=1}^{n-1} \beta_i \int_0^1 u'(x)\varphi_i'(x)\ dx =$$
$$\sum_{i=1}^{n-1} \beta_i \int_0^1 f(x)\varphi_i(x)\ dx = \int_0^1 f(x)v(x)\ dx.$$

Dosadíme za $u(x)$ do (1.3) a po úpravě dostaneme

$$\forall j \in \{1, 2, \ldots, n-1\} : \sum_{i=1}^{n-1} \alpha_i \int_0^1 \varphi_i'(x)\varphi_j'(x)\ dx = \int_0^1 f(x)\varphi_j(x)\ dx. \tag{1.4}$$

Po označení

$$a_{ji} = \int_0^1 \varphi_i'(x)\varphi_j'(x)\ dx, \qquad \forall i, j \in \{1, 2, \ldots, n-1\},$$

$$F_j = \int_0^1 f(x)\varphi_j(x)\ dx, \qquad \forall j \in \{1, 2, \ldots, n-1\},$$

dostáváme (1.4) ve tvaru

$$A\alpha = F,$$

kde $A = (a_{ji})$, $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_{n-1})^T$, $F = (F_1, F_2, \ldots, F_{n-1})^T$. Vyřešením této rovnice dostáváme čísla α_i, která představují souřadnice hledané funkce $u(x)$ v bázi, která je definovaná bázovými funkcemi $\varphi_i(x)$.

1.2. Dvojrozměrný problém.

Zabývejme se nyní okrajovou úlohou

$$\begin{cases} -\Delta u = f, & \text{v } \Omega, \\ u = 0, & \text{na } \partial\Omega, \end{cases} \tag{1.5}$$

kde $\emptyset \neq \Omega \subset \mathbb{R}^2$ je omezená oblast s lipschitzovskou hranicí [4] a $f \in L^2(\Omega)$. Podobně jako v předchozím jednorozměrném případě odvodíme slabou (variační) formulaci této úlohy. Pro každou funkci $v \in H_0^1(\Omega)$ platí

$$\int_\Omega -\Delta u v\ dx = \int_\Omega fv\ dx.$$

Odtud pomocí Greenovy věty [4] a skutečnosti, že pro všechna $x \in \partial\Omega$ je $v(x) = 0$ ve smyslu stop [4], dostáváme

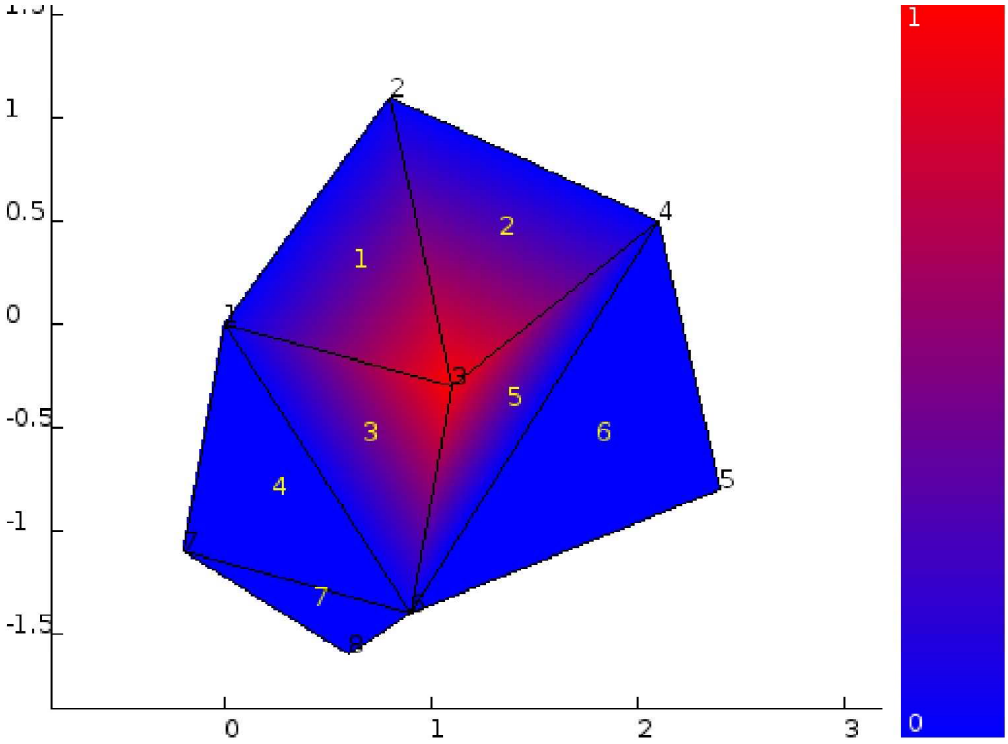
$$\int_\Omega -\Delta u v \, dx = \sum_{i=1}^{2} \int_\Omega -\frac{\partial^2 u}{\partial x_i^2} v \, dx = \sum_{i=1}^{2} \int_\Omega \frac{\partial u}{\partial x_i} \frac{\partial v}{\partial x_i} \, dx + \sum_{i=1}^{2} - \int_{\partial\Omega} \frac{\partial u}{\partial x_i} v \nu_i \, ds =$$
$$= \int_\Omega \nabla u \nabla v \, dx - \int_{\partial\Omega} \frac{du}{d\nu} v \, dx = \int_\Omega \nabla u \nabla v,$$

kde $\nu(x) = (\nu_1(x), \nu_2(x))$ je jednotkový vektor vnější normály k $\partial\Omega$ v x. Definujeme slabé řešení úlohy (1.5) jako $u \in H_0^1(\Omega)$ takové, že pro všechna $v \in H_0^1(\Omega)$ platí

$$\int_\Omega \nabla u \nabla v \, dx = \int_\Omega f v \, dx. \tag{1.6}$$

Podobně jako v jednorozměrném případě nahradíme prostor nekonečné dimenze $H_0^1(\Omega)$, na kterém hledáme slabé řešení, prostorem V konečné dimenze. Zavedeme triangulaci oblasti Ω, tj. rozdělení oblasti Ω na trojúhelníky tak, že dva trojúhelníky, které nejsou shodné nebo disjunktní, mají společnou buď celou hranu, nebo jeden vrchol [5].

Mějme triangulaci oblasti Ω. Vrcholy trojúhelníků dané triangulace nazýváme uzly a trojúhelníky nazýváme prvky. Každému vnitřnímu uzlu, tj. uzlu, který nepatří do $\partial\Omega$, přiřadíme bázovou funkci φ takovou, že φ je rovna 1 v daném uzlu a ve všech ostatních uzlech je rovna 0. Funkce φ je spojitá na $\overline{\Omega}$ a lineární na všech trojúhelnících dané triangulace. V obecném případě stejně definujeme i bázové funkce v uzlech na hranici $\partial\Omega$, ale v našem případě je nezavádíme proto, aby byla splňena Dirichletova okrajová podmínka $u = 0$ na $\partial\Omega$. Na obr. 1.2 vidíme zobrazení jedné bázové funkce na triangulaci, kde počet uzlů $n = 8$ a počet prvků $m = 7$. Také na něm vidíme černými čísly označené indexy uzlů a žlutými čísly indexy prvků.



Obrázek 1.2: *Příklad bázové funkce pro dvojrozměrnou úlohu.*

Nechť $\mathcal{I}^I \subset \{1, 2, \ldots, n\}$ je množina indexů vnitřních uzlů. Prvek u prostoru $V \subset H_0^1(\Omega)$ vyjádříme jako

$$u = \sum_{i \in \mathcal{I}^I} \alpha_i \varphi_i, \quad \alpha_i \in \mathbb{R}.$$

Hledáme takto definované $u \in V$, které splňuje slabou formulaci (1.6). Podobně jako v jednorozměrném případě lze snadno odvodit ekvivalentní tvrzení

$$\forall j \in \mathcal{I}^I : \int_\Omega \nabla u \nabla \varphi_j \, dx = \int_\Omega f \varphi_j \, dx.$$

Dosadíme u a dostáváme

$$\forall j \in \mathcal{I}^I : \sum_{i=1}^{n-1} \alpha_i \int_\Omega \nabla \varphi_i \nabla \varphi_j \, dx = \int_\Omega f \varphi_j \, dx,$$

tedy je soustavu $n-1$ lineárních rovnic pro $n-1$ neznámých $\alpha_1, \alpha_2, \ldots, \alpha_{n-1}$, což jsou souřadnice hledaného řešení u v konečněrozměrné bázi prostoru V.

Pomocí MKP můžeme řešit různé úlohy z praxe jako např. vedení tepla, průhyb membrány, nebo úlohy elektromagnetismu.

Budeme se zabývat vizualizací skalárních a vektorových polí v rovině. Uvažujme oblast $\Omega \subset \mathbb{R}^2$, na které je dána triangulace. Pojmem pole zde rozumíme takovou oblast Ω, pro kterou máme navíc zadanou buď skalární funkci $h : \Omega \to \mathbb{R}$, nebo vektorovou funkci $v : \Omega \to \mathbb{R}^2$.

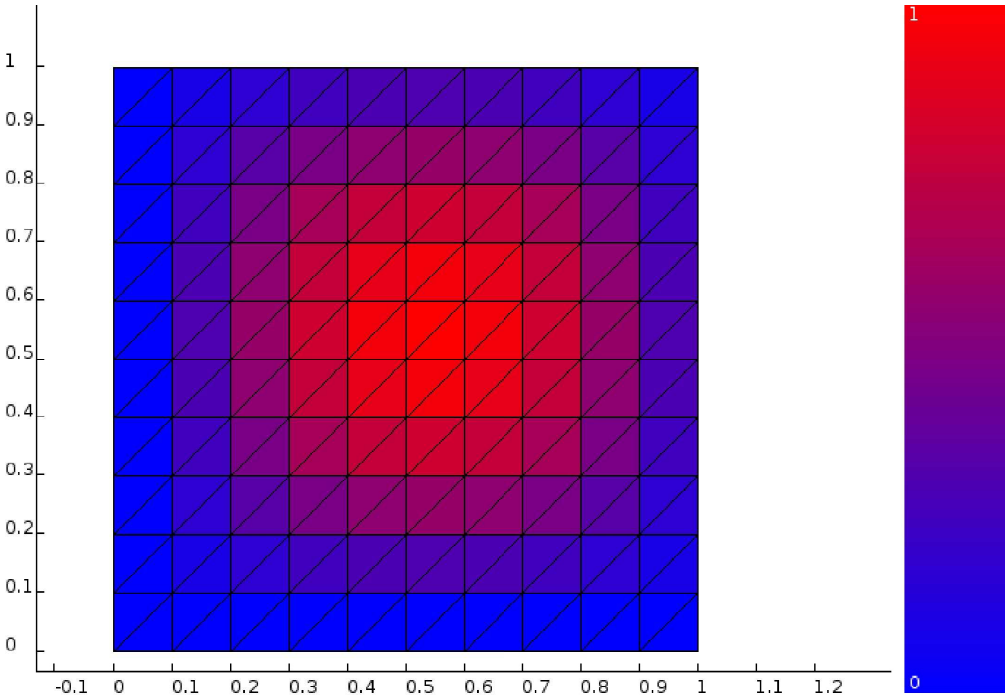
2.1. Skalární pole.

Uvažujeme dva typy skalárních polí nad danou triangulací. Skalární pole konstantní na trojúhelnících a skalární pole lineární na trojúhelnících.

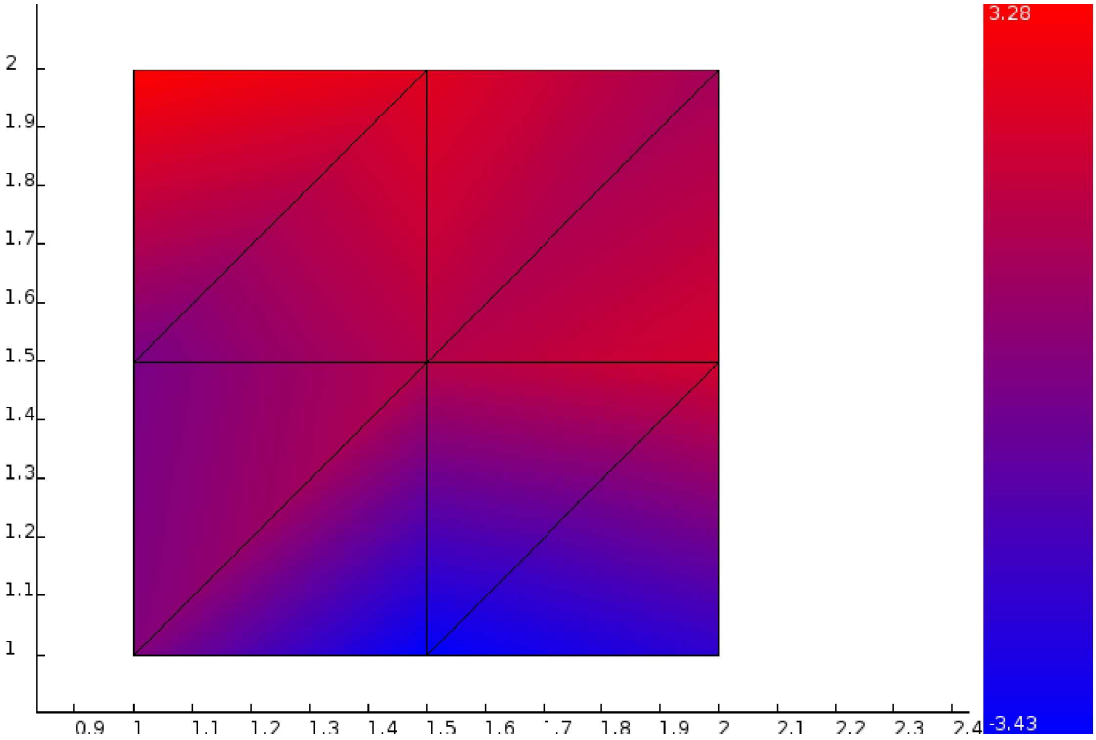
V prvním případě skalárního pole konstantního na trojúhelnících každému trojúhelníku triangulace přiřadíme reálné číslo (skalár) představující hodnotu skalární funkce ve všech bodech tohoto trojúhelníku.

V případě skalárních polí lineárních na trojúhelnících máme jednotlivým uzlům triangulace přiřazeno reálné číslo. Nazývejme hodnotu zobrazení $h(x)$ v bodě $x \in \Omega$ výškou nad bodem x. Každý trojúhelník triangulace má určenou výšku všech svých vrcholů. Výšky jednotlivých bodů patřících do daného trojúhelníku jsou jednoznačně určeny tak, že leží v rovině určené vrcholy daného trojúhelníka.

Skalární pole konstantní na trojúhelnících budeme zkráceně nazývat typ $S0$, skalární pole lineární na trojúhelnících označíme jako typ $S1$. Jak vypadá skalární pole typu $S0$ vidíme na obr. 2.1 a příklad skalárního pole typu $S1$ vidíme na obr. 2.2.



Obrázek 2.1: *Příklad pole typu S0.*



Obrázek 2.2: *Příklad pole typu* $S1$.

Skalární pole jsme zobrazili jako projekci na rovinu, kde barvou zobrazujeme příslušnou výšku v daném bodě. Určíme maximální a minimální výšku zadanou pro jednotlivé trojúhelníky pole typu $S0$. V případě pole typu $S1$ zjistíme maximum a minimum výšek v jednotlivých uzlech. Přiřadíme maximální a minimální výšce různou barvu a vytvoříme barevný přechod od barvy minima k barvě maxima tak, aby každé výšce mezi maximem a minimem byla jednoznačně přiřazena barva.

Uvažujme barvu určenou uspořádanou trojicí (r, g, b), kde $r, g, b \in \langle 0, 1 \rangle$ jsou složky červené, zelené a modré barvy. Označme $h_{\min}$ minimální výšku a $h_{\max}$ maximální výšku v daném poli. Nechť $h_{\min} \leq h \leq h_{\max}$ je výška, které přiřazujeme barvu. Definujme

$$c(h) = \frac{h - h_{\min}}{h_{\max} - h_{\min}},$$

a přiřaďme výšce h barvu $(c(h), 0, 1 - c(h))$. Pak je výšce $h_{\min}$ přiřazena modrá barva, výšce $h_{\max}$ červená barva a barvy výšek mezi $h_{\min}$ a $h_{\max}$ tvoří plynulý přechod od modré k červené barvě (viz např. obr. 2.2).

Každému trojúhelníku pole typu $S0$ je jednoznačně podle jeho výšky přiřazena příslušná barva. V případě $S1$ máme pro každý trojúhelník zadány výšky v každém jeho vrcholu a výšky vnitřních bodů trojúhelníku dopočítáme lineární interpolací.

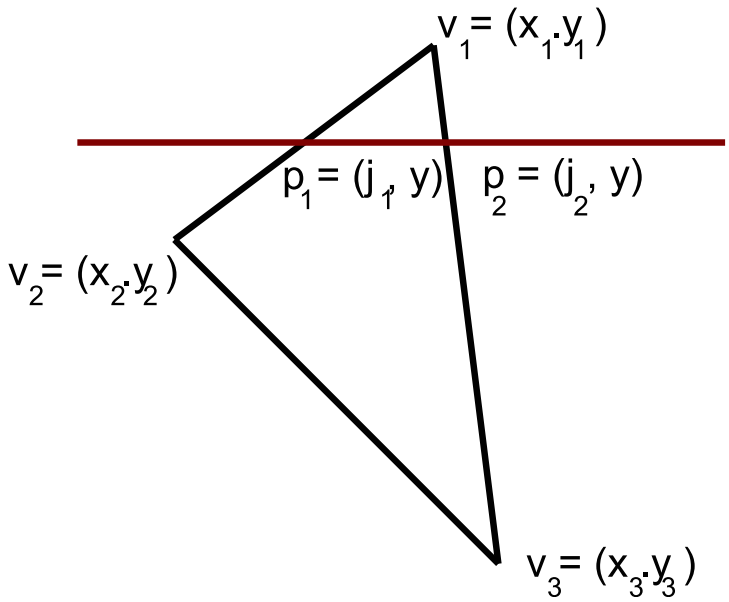
Známe tedy výšku i příslušnou barvu libovolného bodu trojúhelníku. Jak takový trojúhelník nakreslíme metodami počítačové grafiky popisuje následující podkapitola.

2.1.1. Kreslení po řezech.

Princip vykreslení spočívá v tom, že projdeme všechny body patřící do daného trojúhelníku, pro každý bod vypočítáme jeho výšku a podle ní určíme jeho barvu. Budeme postupovat po řezech trojúhelníku rovnoběžných s osou x. Protože vykreslujeme trojúhelník prostředky počítačové grafiky používáme pixely místo bodů v rovině a jejich souřadnice jsou tedy celá čísla.

Označme vrcholy uvažovaného trojúhelníku $v_1 = (x_1, y_1)$, $v_2 = (x_2, y_2)$, $v_3 = (x_3, y_3)$ tak, aby platilo $y_1 \leq y_2 \leq y_3$, kde $x_i, y_i \in \mathbb{N}$, $i = 1, 2, 3$ jsou souřadnice pixelů, které zobrazují dané vrcholy. Dále označíme $h_1, h_2, h_3 \in \mathbb{R}$ výšky ve vrcholech v_1, v_2 a v_3. Pomocí řezů trojúhelníku nalezneme nejprve dva body na stranách trojúhelníku a výšky v těchto bodech a poté najděme výšky ve všech bodech úsečky spojující tyto dva body. Protože zobrazujeme rovinu v prostoru, využíváme toho, že pokud známe výšky ve dvou krajních bodech úsečky, tak výšky ve vnitřních bodech této úsečky lineárně přecházejí z výšky v jednom krajním bodě do výšky v druhém.

Provedeme postupně řezy trojúhelníku přímkou $y = k$, $k = y_1, y_1 + 1, y_1 + 2, \ldots, y_2$ a nalezneme průsečíky $p_1 = (j_1, y)$ a $p_2 = (j_2, y)$ s úsečkami v_1v_2 a v_1v_3. Viz obr. 2.3.



Obrázek 2.3: *Řez trojúhelníku.*

Souřadnici j_1 bodu p_1 dostaneme z rovnice

$$\frac{j_1 - x_1}{x_2 - x_1} = \frac{y - y_1}{y_2 - y_1},$$

podobně vypočítáme také j_2. Označme $h(p_1)$ výšku v bodě p_1. Tuto výšku vyjádříme z rovnice

$$\frac{h(p_1) - h_1}{h_2 - h_1} = \frac{y - y_1}{y_2 - y_1}.$$

Obdobně vyjádříme výšku v bodě p_2. Protože známe výšky v bodech p_1 a p_2 dopočítáme i body řezu, tj. úsečky p_1p_2 tak, že volíme $j = j_1, j_1 + 1, j_1 + 2, \ldots, j_2$ a výšku $h(p)$ v bodě $p = (j, y)$ dostaneme z rovnice

$$\frac{h(p) - h(p_1)}{h(p_2) - h(p_1)} = \frac{j - j_1}{j_2 - j_1}.$$

Stejně postupujeme i řezy ve výšce $y = y_2, y_2 + 1, y_2 + 2, \ldots, y_3$, kde hledáme průsečíky s úsečkami v_2v_3 a v_1v_3.

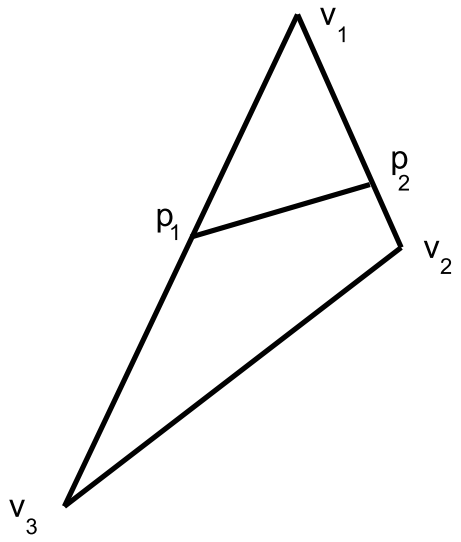
2.1.2. Vrstevnice.

V případě polí typu $S1$ bývá užitečné zobrazovat také vrstevnice, tj. množiny bodů, které mají stejnou výšku. Vrstevnice zobrazujeme v pravidelných intervalech od sebe následujícím způsobem. Nechť $h_{\min}$ je nejmenší a $h_{\max}$ je největší výška daného pole. Tyto hodnoty získáme jako minimum a maximum výšek zadaných v jednotlivých uzlech. Označme m počet vrstevnic, které chceme zobrazit. Hledané vrstevnice jsou množiny bodů s výškou

$$h_i = h_{\min} + i \cdot \frac{h_{\max} - h_{\min}}{m - 1}, \quad i = 0, 1, \ldots m - 1.$$

Pro každý trojúhelník triangulace zjišťujeme, zda jím prochází vrstevnice výšky h_i, $i = 0, 1, \ldots m-1$. Nejprve zjistíme, zda se výška vrstevnice nachází mezi nejmenší a největší výškou jednotlivých vrcholů trojúhelníku. Pokud ano, tak to znamená, že vrstevnice této výšky prochází daným trojúhelníkem. V tom případě najdeme dva body na stranách trojúhelníku, kterými vrstevnice prochází, a ostatní body vrstevnice dostaneme jako úsečku spojující tyto dva body.

Předchozí postup si projdeme podrobněji. Označme si vrcholy uvažovaného trojúhelníku $v_1, v_2, v_1 \in \mathbb{R}^2$ tak, aby pro jejich výšky $h(v_1), h(v_2), h(v_1) \in \mathbb{R}$ platilo $h(v_1) \leq h(v_2) \leq h(v_3)$. Označme $h = h_i$ konkrétní zkoumanou výšku vrstevnice. Vrstevnice této výšky daným trojúhelníkem prochází, pokud platí $h(v_1) \leq h \leq h(v_3)$. Pokud vrstevnice trojúhelníkem prochází hledáme body p_1 a p_2 na stranách trojúhelníku tak, aby jejich výška byla h. Situaci vidíme na obr. 2.4.



Obrázek 2.4: *Vrstevnice.*

Speciální případ dostaneme, pokud $h(v_1) = h(v_3) = h$. Pak patří do vrstevnice celý trojúhelník a vybarvíme jej barvou odpovídající výšce h. Dále předpokládejme, že platí $h(v_1) < h(v_3)$. Jeden z bodů p_1, p_2 musí ležet na úsečce v_1v_3 a druhý leží buď na úsečce v_1v_2, nebo na úsečce v_2v_3. Nechť p_1 je bod ležící na úsečce v_1v_3, takže platí

$$p_1 = v_1 + k(v_3 - v_1), \quad k \in \langle 0, 1 \rangle.$$

Koeficient k dostaneme tak, aby poloha bodu p_1 mezi krajními body úsečky v_1v_2 byla ve stejném poměru jako výška h v bodě p_1 k výškám v krajních bodech $h(v_1)$, $h(v_2)$. Odtud dostáváme

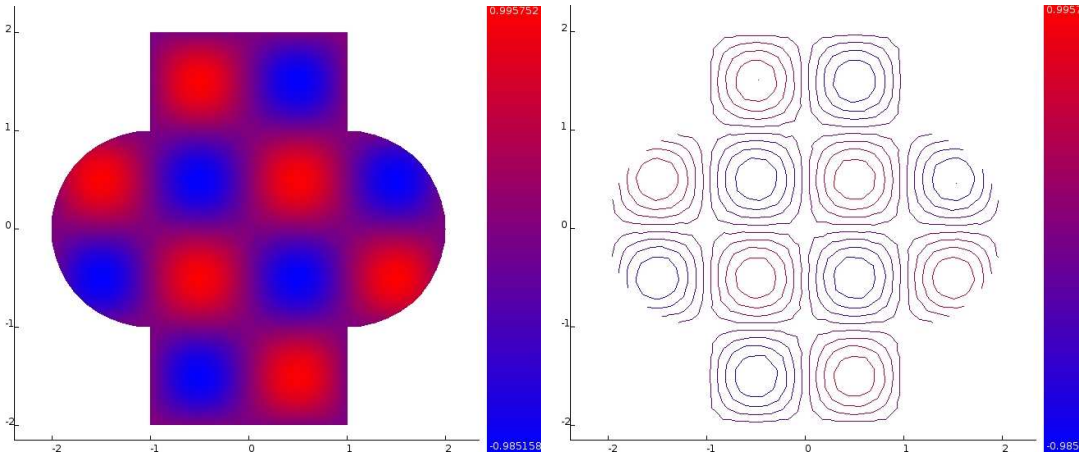
$$k = \frac{|p_1 - v_1|}{|v_3 - v_1|} = \frac{h - h(v_1)}{h(v_3) - h(v_1)}.$$

Podobným postupem odvodíme, že

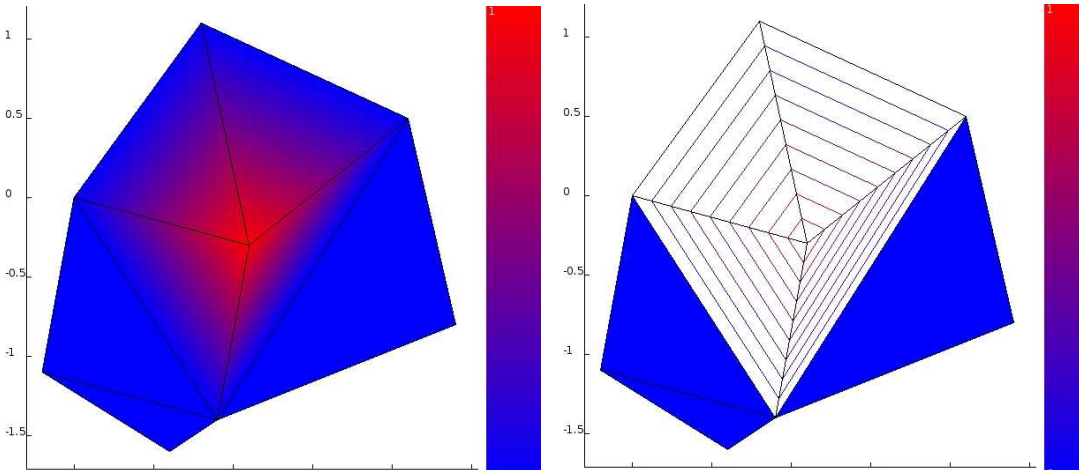
$$p_2 = \begin{cases} v_1 + \dfrac{h - h(v_1)}{h(v_2) - h(v_1)}(v_2 - v_1), & h \in \langle h(v_1), h(v_2)) \\ v_2 + \dfrac{h - h(v_2)}{h(v_3) - h(v_2)}(v_3 - v_2), & h \in \langle h(v_2), h(v_3)\rangle, h(v_2) \neq h(v_3) \\ v_2, & h = h(v_2) = h(v_3). \end{cases}$$

Podmínky pro h jsou voleny tak, abychom nikdy nedostali nulu do jmenovatele. Nakonec nakreslíme úsečku p_1p_2 barvou odpovídající výšce h.

Na obrázku 2.5 je příklad pole typu $S1$ a jeho vrstevnic. Na obrázku 2.6 vidíme, jak vypadají vrstevnice bázové funkce z první kapitoly. Tento příklad obsahuje vrstevnice, které tvoří celé trojúhelníky.



Obrázek 2.5: *Zobrazení vrstevnic skalárního pole.*



Obrázek 2.6 *Vrstevnice bázové funkce pro metodu konečných prvků.*

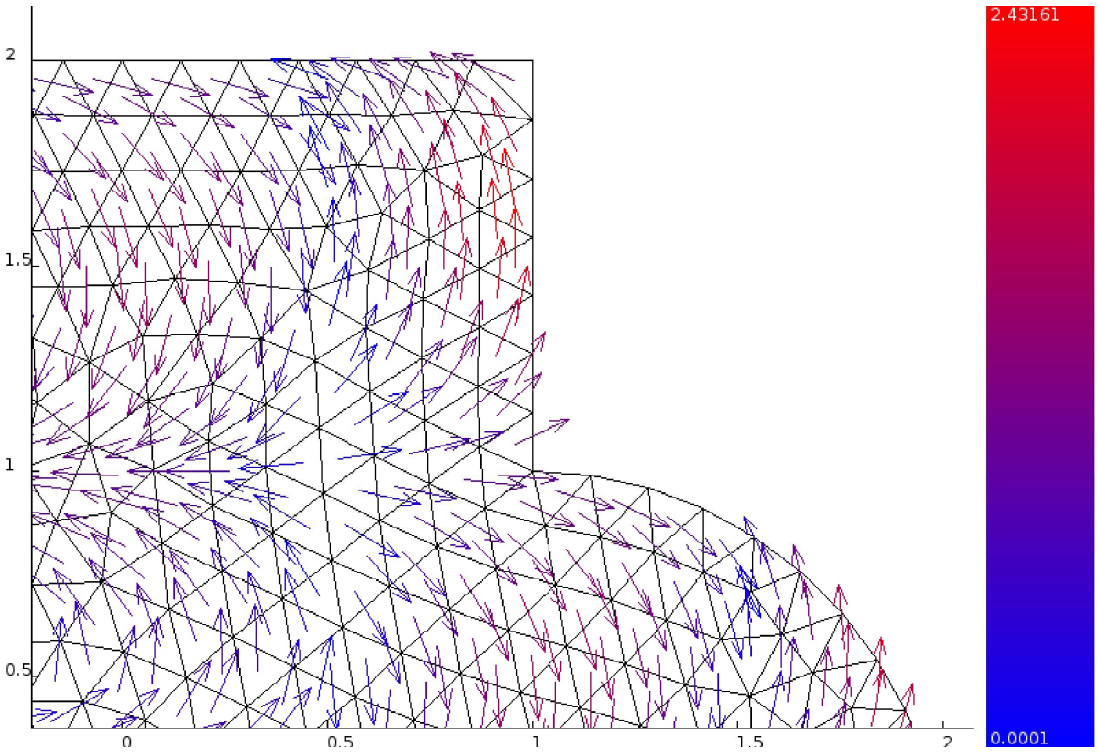
2.2. Vektorová pole.

Podobně jako u skalárních polí nás zajímají vektorová pole konstantní na trojúhelnících, která nazýváme pole typu $V0$, a vektorová pole konstantní na uzlech, která zkráceně označujeme jako pole typu $V1$. V případě vektorových polí máme

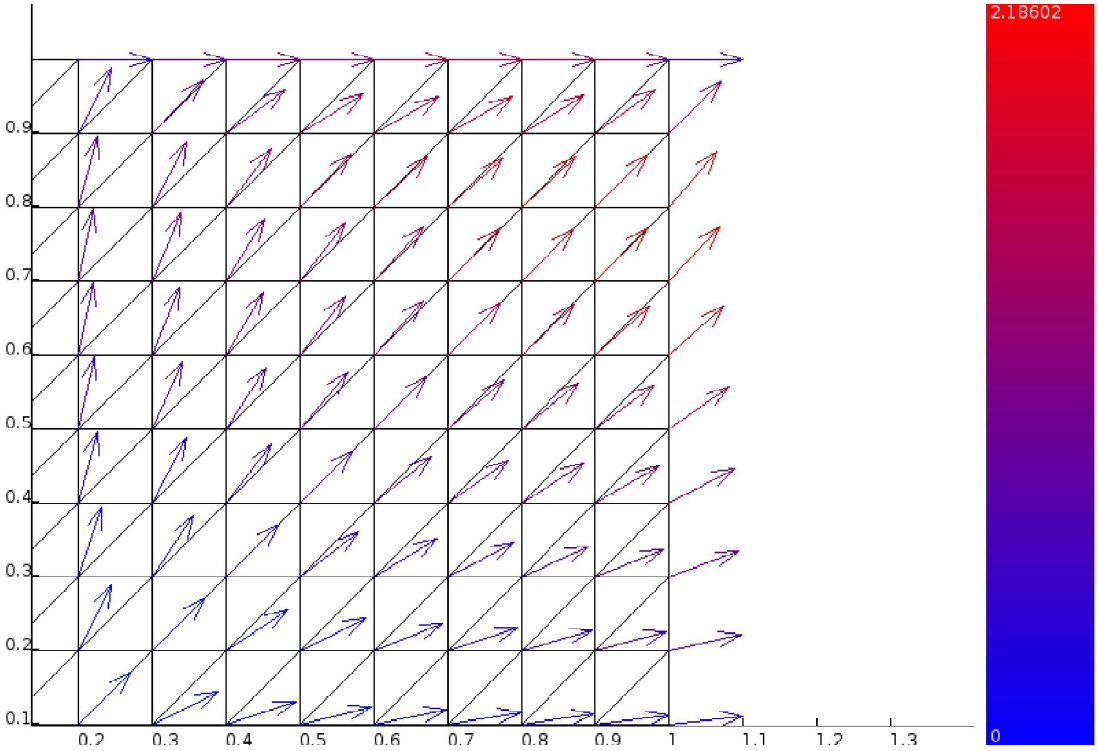
pro každý trojúhelník nebo uzel triangulace zadán vektor $u = (u_x, u_y)$. Vektor znázorníme šipkou, která má směr zobrazovaného vektoru. Velikost vektoru zobrazíme barvou šipky. Barvu šipce přiřadíme stejně, jako jsme v případě skalárních polí přiřadili barvu výšce. U vektorových polí typu $V0$ umístíme počátek vektoru do těžiště trojúhelníku. Těžiště trojúhelníku o vrcholech $v_1 = (x_1, y_1)$,$v_2 = (x_2, y_2)$, $v_3 = (x_3, y_4)$ je bod

$$t = (\frac{x_1 + x_2 + x_3}{3}, \frac{y_1 + y_2 + y_3}{3}).$$

U vektorových polí typu $V1$ umístíme počátek vektoru do příslušného uzlu. Příklad pole typu $V0$ je na obr. 2.7 a příklad pole typu $V1$ je na obr. 2.8.



Obrázek. 2.7: *Příklad vektorového pole typu $V0$.*



Obrázek 2.8: *Příklad vektorového pole typu $V1$.*

2.3. Výpočet gradientu.

Při řešení praktických úloh je často užitečné znát hodnoty gradientu v bodech pole. Nechť $h : \Omega \to \mathbb{R}$, $\Omega \subset \mathbb{R}^2$ je skalární funkce definovaná na oblasti Ω. Pak gradient funkce h je vektorová funkce $\nabla h : \Omega \to \mathbb{R}^2$ definovaná předpisem

$$\nabla h(x, y) = \left(\frac{\partial h(x, y)}{\partial x}, \frac{\partial h(x, y)}{\partial x} \right).$$

Ukážeme si, jak ze skalárního pole typu $S1$ dostaneme vektorové pole gradientů typu $V0$.

Uvažujme trojúhelník triangulace pole typu $S1$. Skalární funkce h je na tomto trojúhelníku lineární, a proto je gradient ∇h konstantní ve všech bodech uvažovaného trojúhelníku. Označme vrcholy uvažovaného trojúhelníku $v_1 = (x_1, y_1)$, $v_2 = (x_2, y_2)$ a $v_1 = (x_3, y_3)$. Zaveďme body $p_1, p_2, p_3 \in \mathbb{R}^3$ takto:

$$p_1 = (x_1, y_1, h(x_1, y_1)), \quad p_2 = (x_2, y_2, h(x_2, y_2)), \quad p_3 = (x_3, y_3, h(x_3, y_3)).$$

Bod $p = (x, y, h(x, y))$, kde (x, y) je bodem uvažovaného trojúhelníku, leží v rovině, která je určena body p_1, p_2, p_3. Pomocí vektorového součinu získáme normálový vektor této roviny

$$\mathbf{n} = (p_2 - p_1) \times (p_3 - p_1).$$

Označíme $\mathbf{n} = (a, b, c)$. Obecná rovnice hledané roviny je

$$ax + by + cz + d = 0,$$

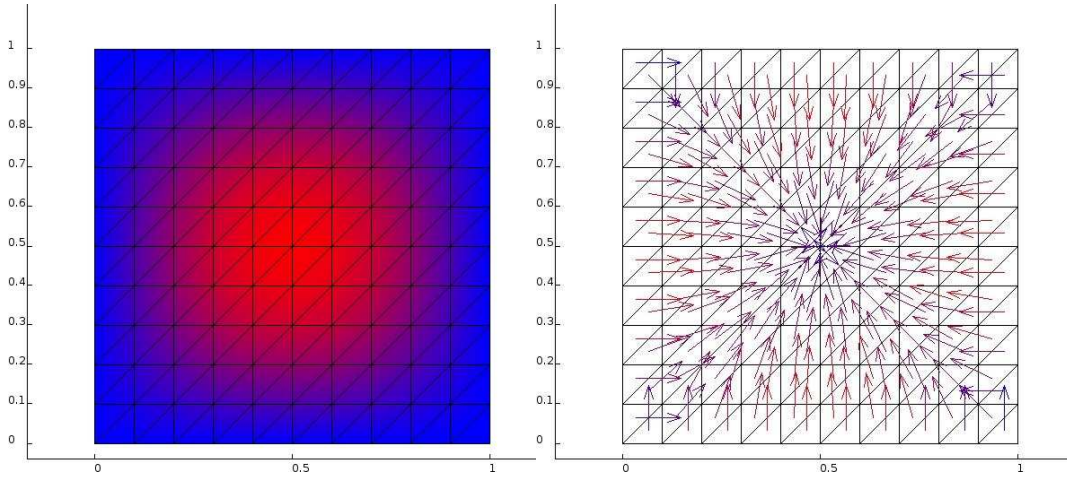
kde $d \in \mathbb{R}$. Vyjádříme předpis pro funkci h jako

$$h(x, y) = z = -\frac{1}{c}(ax + by + d).$$

Odtud dostáváme

$$\nabla h(x, y) = -\frac{1}{c}(a, b).$$

Takto pro všechny trojúhelníky triangulace pole typu $S1$ získáme vektory gradientu, které nad touto triangulací tvoří pole typu $V0$. Příklad takových polí vidíme na obr. 2.9.



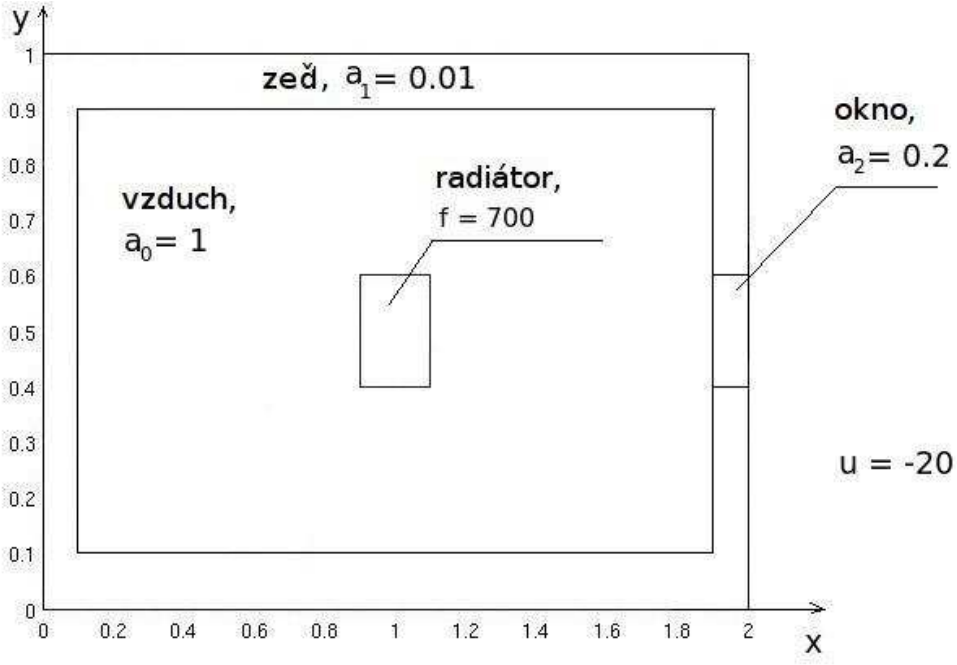
Obrázek 2.9: *Skalární pole a odpovídající vektorové pole gradientů.*

3. Vizualizace výsledků úloh z praxe

V této kapitole si ukážeme použití metod z předchozí kapitoly při vizualizaci výsledků konkrétních úloh. K zobrazení výsledků byl použit program Varan, který je popsán v následujících kapitolách 4 a 5.

Příklad 1. Úloha rozložení tepla v místnosti.

Najděte rozložení teploty $u(x, y)$ v místnosti, kde uvažujeme zdi, okno a radiátor, viz obr. 3.1.



Obrázek 3.1: *Zadání příkladu 1.*

Matematický model této úlohy je

$$\begin{cases} -\mathrm{div}(a(x,y)\nabla u(x,y)) = f(x,y) & \text{v } \Omega = (0,2)\times(0,1), \\ u(x,y) = -20 & \text{na } \partial\Omega, \end{cases}$$

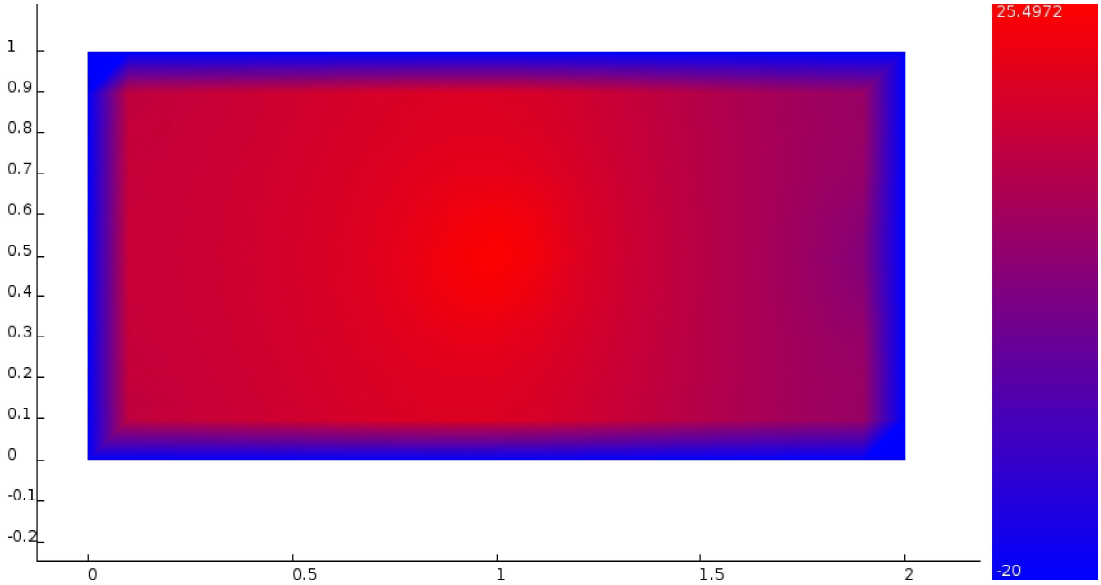
kde koeficient tepelné vodivosti

$$a(x,y) = \begin{cases} 1 & \text{ve vzduchu,} \\ 0.01 & \text{ve zdi,} \\ 0.2 & \text{v okně,} \end{cases}$$

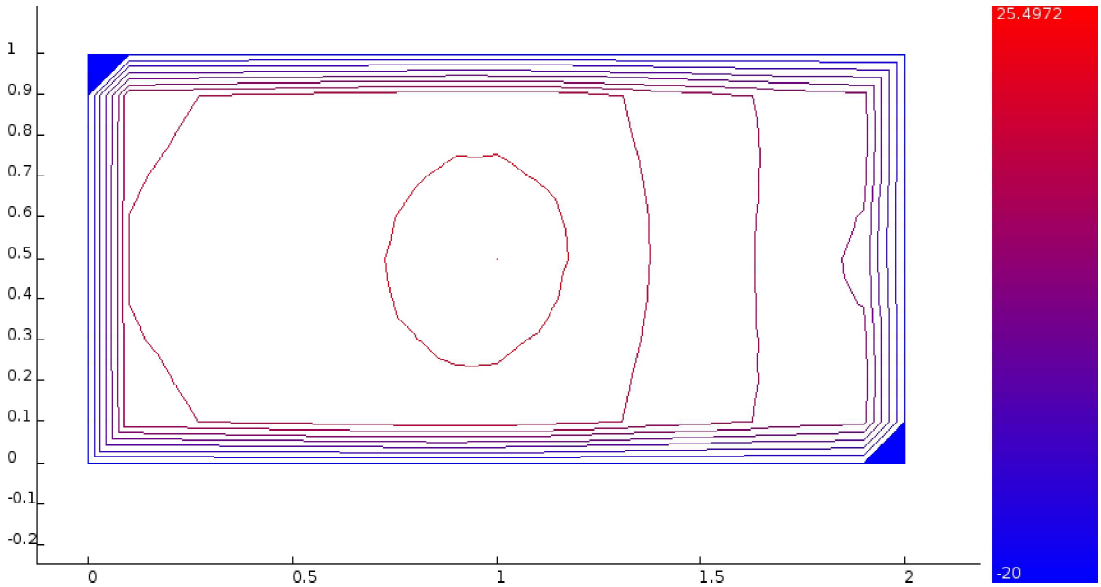
a zdroj tepla

$$f(x,y) = \begin{cases} 700 & \text{radiátor,} \\ 0 & \text{jinde.} \end{cases}$$

Výsledek získaný metodou konečných prvků zobrazený a programem Varan vidíme na obr. 3.2. Na obrázku 3.3 můžeme vidět vrstevnice tohoto pole.



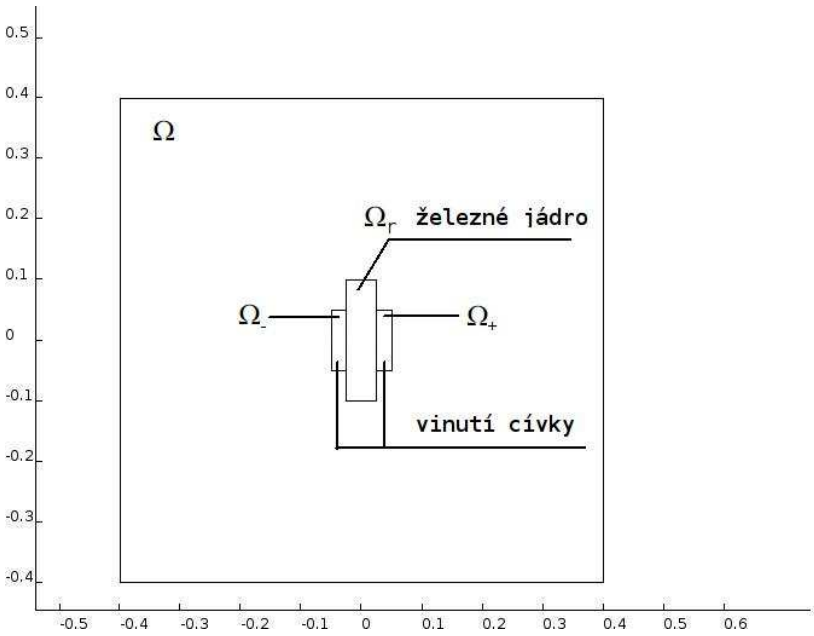
Obrázek 3.2: *Výsledek příkladu 1.*



Obrázek 3.3: *Vrstevnice výsledku příkladu 1.*

Příklad 2. Magnetické pole cívky.

Vypočtěte rozložení pole magnetické indukce cívky s železným jádrem buzenou stejnosměrným proudem protékajícím vinutím (viz obr 3.4).



Obrázek 3.4: *Schéma k zadání příkladu 2.*

Matematická formulace této úlohy je: najděte magnetický potenciál $u(x, y)$, pro který platí

$$\begin{cases} -\mathrm{div}(\nu(x)\nabla(u(x,y))) = \mu_0 J(x) & \text{v } \Omega = (-0.4, 0.4) \times (-0.4, 0.4), \\ u(x,y) = 0 & \text{na } \partial\Omega, \end{cases}$$

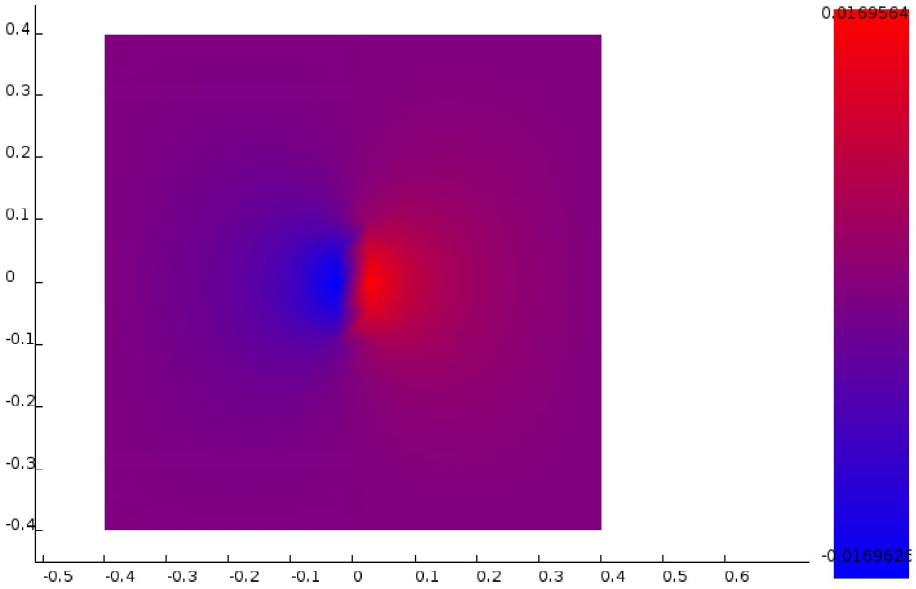
kde reluktivita prosředí

$$\nu(x,y) = \begin{cases} 1/5100 & \text{v } \Omega_r \text{ (železné jádro)}, \\ 1 & \text{jinde}, \end{cases}$$

proudová hustota

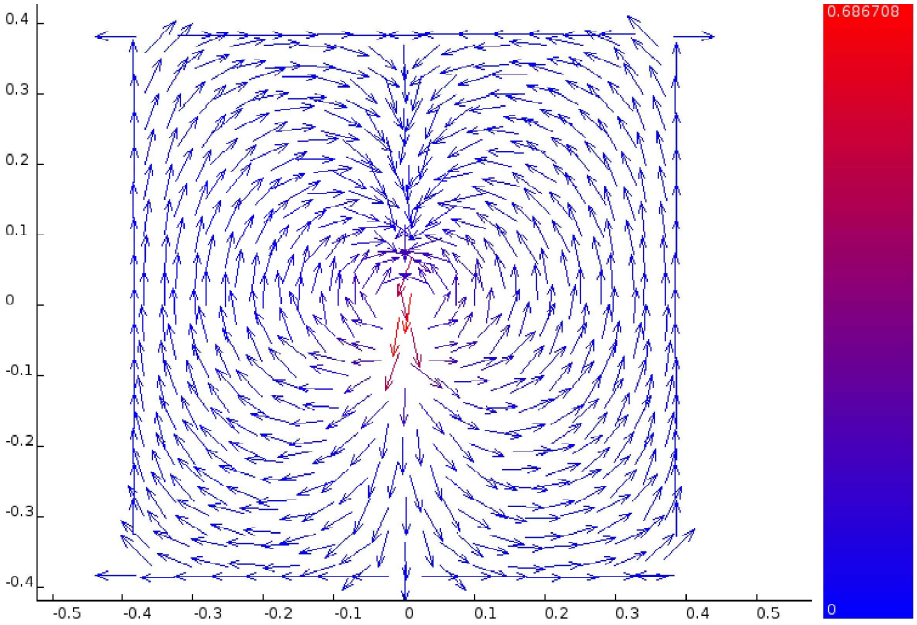
$$J(x,y) = \begin{cases} -K & \text{v } \Omega_- \text{ (levá část vinutí)}, \\ +K & \text{v } \Omega_+ \text{ (pravá část vinutí)}, \end{cases}$$

kde $K = 8 \cdot 10^6$ a permeabilita vzduchu $\mu_0 = 4 \cdot \pi \cdot 10^{-7}$. Výsledek této úlohy můžeme vidět na obr. 3.5.



Obrázek 3.5: *Řešení příkladu 2. - magnetické pole cívky.*

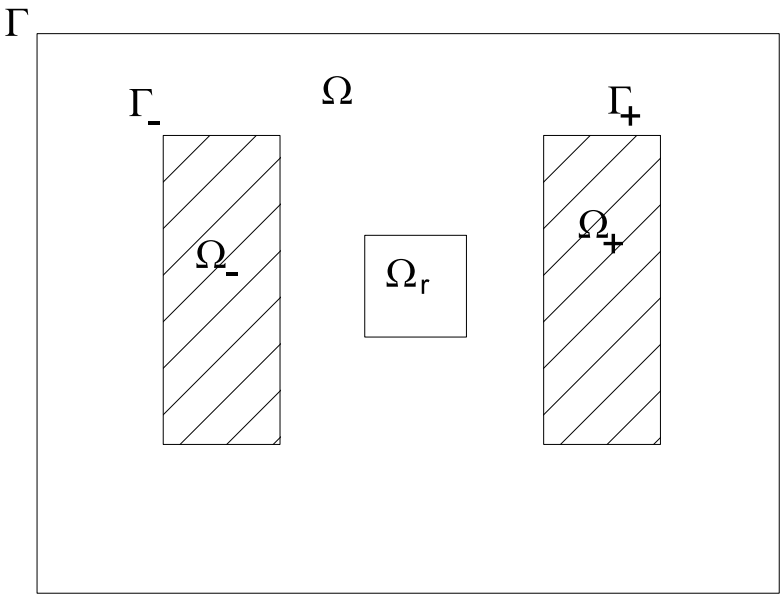
Postupem uvedeným v minulé kapitole vypočítáme gradient magnetického potenciálu $\nabla u(x,y) = (\frac{\partial u(x,y)}{\partial x}, \frac{\partial u(x,y)}{\partial y})$, pomocí kterého si vyjádříme vektor magnetické indukce $B = (\frac{\partial u(x,y)}{\partial y}, -\frac{\partial u(x,y)}{\partial x})$. Pole vektorů magnetické indukce vidíme na obr. 3.6.



Obrázek 3.6: *Řešení příkladu 2. - vektory magnetické indukce.*

Příklad 3. Elektrostatické pole kondenzátoru..

Vypočítejte rozložení elektrostatického potenciálu $u(x, y)$ na oblasti Ω vyplněné vzduchem, ve které jsou v oblastech Ω_- a Ω_+ vloženy dvě nabité desky, které mají na povrchu konstantní potenciál $U > 0$, resp. $-U$. Mezi těmito deskami je v oblasti Ω_r vloženo dielektrikum o relativní permitivitě $\varepsilon_r > 1$. Viz obr. 3.7.



Obrázek 3.7: *Zadání příkladu 3.*

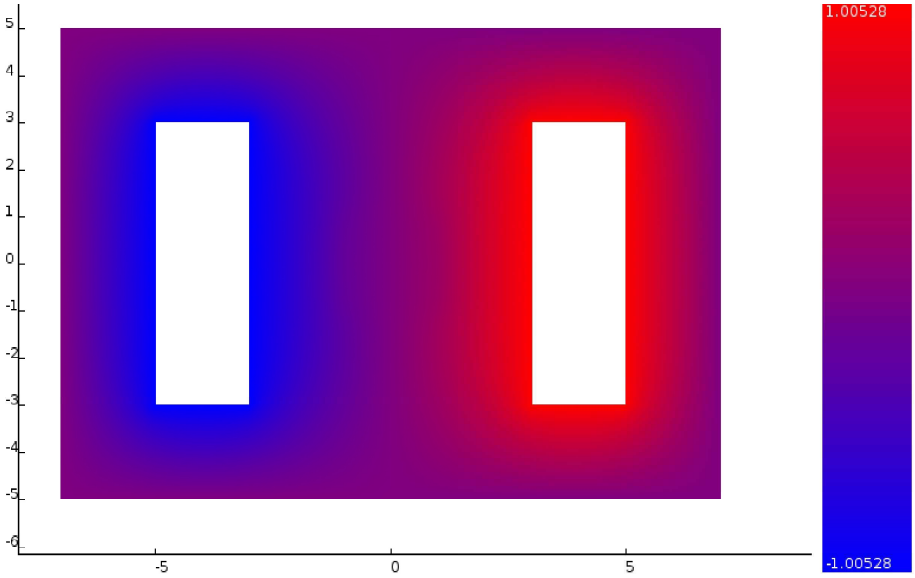
Matematická formulace této úlohy je následující. Najděte elektrostatická potenciál $u(x, y)$, pro který platí

$$\begin{cases} -\mathrm{div}(\varepsilon(x,y)\nabla u(x,y)) = 0 & \text{v } \Omega, \\ u(x,y) = U & \text{na } \Gamma_+, \\ u(x,y) = -U & \text{na } \Gamma_-, \\ u(x,y) = 0 & \text{na } \Gamma, \end{cases}$$

kde permitivita prostředí

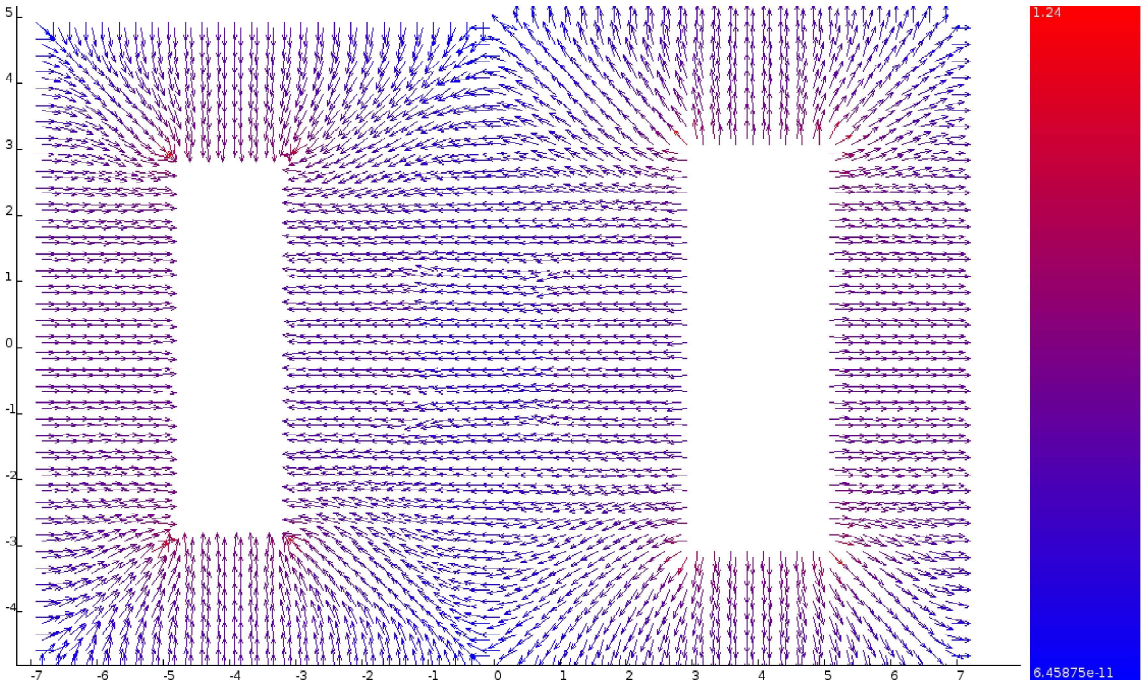
$$\varepsilon_r(x,y) = \begin{cases} 2 & \text{v } \Omega_r, \\ 1 & \text{jinde.} \end{cases}$$

Výsledek vidíme na obrázku 3.8.



Obrázek 3.8: *Výsledné rozložení potenciálu $u(x, y)$ z příkladu 3.*

Opět vypočítáme gradient, pomocí kterého zobrazíme vektory intenzity elektrického pole $E(x, y) = -\nabla u(x, y)$. Výsledek vidíme na obr 3.9.

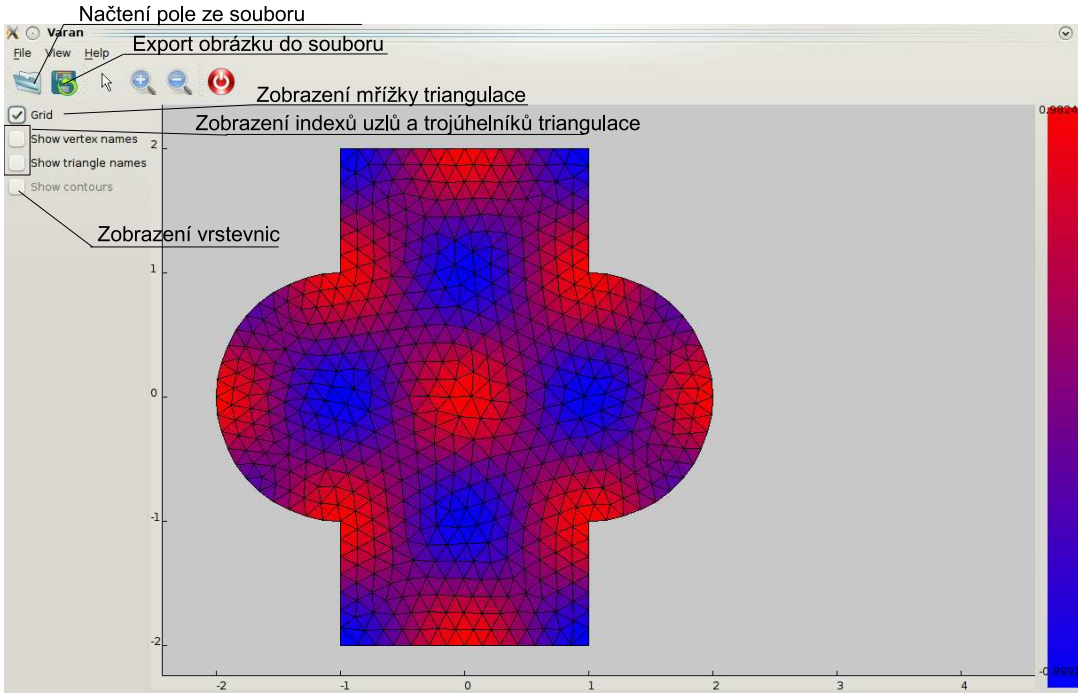


Obrázek 3.9: *Vektory intenzity elektrického pole z příkladu 3.*

4. Program Varan

Cílem této práce je vytvořit program pro vizualizaci rovinných skalárních a vektorových polí. Program se jmenuje Varan a vizualizuje skalární a vektorová pole prostředky popsanými v kapitole 3. Program Varan je naprogramován v jazyce C++ s použitím frameworku Qt verze 4.6.2, takže je možné jej použít na různých operačních systémech. Program byl testován na systémech Linux (Kubuntu) a Microsoft Windows XP.

V této kapitole popíšeme vzhled a vlastnosti tohoto programu. Na obr. 4.1 vidíme, jak vypadá program Varan a popis jeho ovládacích prvků.



Obrázek 4.1: *Program Varan.*

4.1. Funkce programu Varan.

- Vizualizuje triangulaci.
- Vizualizuje pole typu $S0$, $S1$, $V0$, $V1$.
- Zobrazuje indexy trojúhelníků a uzlů triangulace.
- Umožňuje posouvat, přiblížit a oddálit aktuální zobrazení.
- Zobrazuje vrstevnice polí typu $S1$.
- Načítá data ze souboru.

- Exportuje aktuální zobrazení do formátů PostScript a JPEG.

4.2. Načítání dat ze souboru.

Program Varan načítá data ze dvou typů souborů. Jsou to textové soubory námi definované struktury a soubory obsahující matice uložené programem GNU Octave. V souborech jsou uloženy informace o triangulaci a hodnoty pole na trojúhelnících nebo v uzlech dané triangulace.

4.2.1. Struktura textového souboru s daty.

Nejprve jsou v souboru uloženy souřadnice jednotlivých uzlů. Na prvním řádku je celé číslo udávající počet uzlů n. Na následujícíh n řádcích jsou jednotlivé souřadnice tak, že každý řádek obsahuje souřadnice jednoho bodu, tedy dvojici reálných čísel oddělených mezerou. Uzly jsou indexovány od jedničky podle pořadí, ve kterém jsou napsány. Například následujícím způsobem definujeme souřadnice uzlů triangulace na obr. 1.2.

```
8
0 0
0.8 1.1
1.1 -0.3
2.1 0.5
2.4 -0.8
0.9 -1.4
-0.2 -1.1
0.6 -1.6
```

Následují informace o trojúhelnících. Jeden řádek obsahuje celé číslo udávající počet trojúhelníků m. Následujících m řádků definuje trojúhelníky triangulace. Kařdý z těchto m řádků obsahuje trojici celých čísel z rozmezí 1 až n, která udávají indexy uzlů které tvoří vrcholy trojúhelníku. Čísla jsou opět oddělena mezerami. Následuje příklad, jak definujeme indexy trojúhelníků v triangulaci na obr. 1.2.

```
7
1 2 3
2 3 4
1 3 6
1 7 6
3 4 6
4 5 6
6 7 8
```

Dále jsou uvedeny informace o poli. Pokud tyto informace chybí, program Varan zobrazí pouze síť triangulace. Na prvním řádku je označení typu pole a to

buď `S0`, nebo `S1`, nebo `V0`, nebo `V1`. V případě pole typu $S0$ následují hodnoty pole na m trojúhelnících, to znamená m řádků, kde na každém je jedno reálné číslo. Podobně pro pole typu $S1$ máme n řádků s jednotlivými čísly. Pole typu $V0$ a $V1$ jsou definována podobně, s tím rozdílem, že na každém řádku jsou dvě reálná čísla oddělená mezerami, která udávají velikosti složek vektoru.

Hodnoty v uzlech pole typu $S1$, které je na obr. 1.2 definujeme takto:

```
S1
0
0
1
0
0
0
0
0
```

4.4.2. Soubory dat uložených programem GNU Octave.

Často používaným nástrojem pro provádění výpočtů je program MATLAB. Jeho volně šířitelná alternativa je program GNU Octave, který se snaží být s programem MATLAB co nejvíce kompatibilní. Výhodou GNU Octave je, že se jedná o svobodný software a je zdarma. Data používaná v programu GNU Octave snadno uložíme do souboru a zobrazíme v programu Varan.

Struktura dat je podobná jako v předchozím případě dat uložených v textovém souboru. Označme n počet uzlů triangulace. Pro uložení souřadnic uzlů použijeme matici typu $n \times 2$, jejíž řádky obsahují souřadnice jednotlivých uzlů. Matici označíme například P.

Dále definujeme matici T typu $m \times 3$, kde m je počet trojúhelníků triangulace. Každý řádek matice T obsahuje tři celá čísla v rozmezí 1 až n udávající indexy uzlů, které tvoří daný trojúhelník. Indexy uzlů představují čísla řádků matice P.

Jména matic P a T můžeme volit libovolně. Jméno třetí matice však musí být buď $S0$, nebo $S1$, nebo $V0$, nebo $V1$ podle toho, o jaký typ pole se jedná. Matice $S0$ je typu $m \times 1$ a obsahuje hodnoty pole v trojúhelnících. Podobně definujeme matici $S1$ typu $n \times 1$, matici $V0$ typu $m \times 2$ nebo matici $V1$ typu $n \times 2$.

Tyto matice v GNU Octave uložíme do souboru pomocí příkazu `save`. Například pole typu $S1$ uložíme do souboru `grid.dat` pomocí příkazu

```
save grid.dat P T S1
```

Pokud uložíme pouze matice P a T, program Varan zobrazí síť triangulace.

4.3. Překlad zdrojových kódů.

Snadný způsob, kterým přeložíme zdrojové kódy programu Varan, je pomocí vývojového prostředí frameworku Qt – programu Qt Creator. Projekt programu Varan naimportujeme do prostředí Qt Creator otevřením souboru `varan.pro`, který je součást zdrojových kódů. Poté v prostředí Qt Creator spustíme překlad projektu.

Pro překlad z příkazového řádku poskytuje framework Qt nástroj `qmake`, který generuje *Makefile* projektu. Program varan pomocí `qmake` přeložíme takto:

```
qmake varan.pro
make
```

V této kapitole si ukážeme, jak je program Varan implementován pomocí frameworku Qt. Framework Qt poskytuje dva koncepty pro efektivní programování grafického rozhraní. Jsou to tzv. *widgety* a mechanismus *signály-sloty*.

Widgety jsou objekty, které dědí z třídy `QWidget`, a představují obdelníkový prvek grafického uživatelského rozhraní (GUI). Na widgety lze vykreslit požadovaný grafický výstup a také je možné na ně umístit jiné widgety. Widgety také reagují na tzv. eventy, což může být například kliknutí myší, vstup z klávesnice a podobně. Framework Qt poskytuje hotové widgety jako např. okno, tlačítko nebo checkbox. Je také možné definovat vlastní widgety jako třídy, které dědí buď přímo ze třídy `QWidget`, nebo z některého jiného widgetu.

Signály-sloty jsou mechanismus, pomocí kterého mohou objekty komunikovat mezi sebou. Tento mechanismus se často používá ke komunikaci widgetů. Signál je speciální *member function* třídy deklarovaný v bloku za klíčovým slovem `signals`. Podobně slot je speciální *member function* třídy deklarovaný v bloku za klíčovým slovem `public slots`. `signals` a `public slots` jsou klíčová slova frameworku Qt. Pomocí funkce `connect`, propojíme signál jednoho objektu se slotem druhého objektu. Zavoláním funkce signálu (před volání funkce přidáme klíčové slovo `emit`) se vyvolá funkce slotu, se kterým byl daný signál propojen. Na jeden signál může reagovat i více slotů. Pokud mají funkce signálu i slotu argumenty stejného typu ve stejném pořadí, pak se tyto argumenty předávají. Příklad použití funkce `connect` uvedeme později.

Qt obsahuje také mnoho tříd objektů, které nejsou widgety, a které poskytují různou funkcionalitu, jako například podporu pro počítačové síťe, databáze a podobně. To dělá z Qt něco více, než jen knihovnu pro GUI, a proto je Qt označován jako framework. Třídy objektů (včetně widgetů), které definuje Qt poznáme podle toho, že začínají velkým písmenem Q, jako např. `QButton`, `QPainter`, apod. Abychom tyto třídy odlišili od námi definovných tříd, je dobré pojmenovat naše třídy tak, aby začínaly jiným písmenem než Q. Tuto konvenci budeme dále dodržovat.

Začátek Qt programu se nachází ve funkci `main`. Zdrojový kód funkce `main` programu Varan vypadá takto:

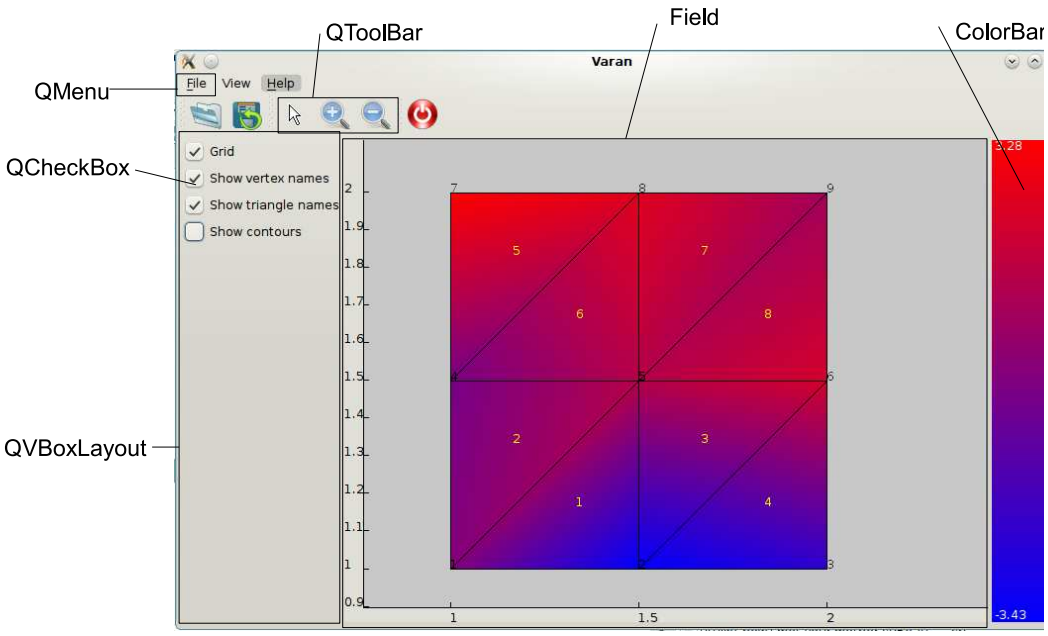
```
#include <QApplication>
#include "MainWindow.h"

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    MainWindow window;
    window.show();
    return app.exec();
```

}

Objekt třídy `QApplication` řídí běh celé GUI aplikace. Třída `MainWindow` je třída programu Varan (nemá na začátku jména Q) a představuje widget hlavního okna, na kterém jsou umístěny všechny ostatní widgety. Definice widgetů umístěných na `MainWindow` a definice signálů a slotů, kterými objekty mezi sebou komunikují, se provádí v konstruktoru třídy `MainWindow`. Jak, to popisuje následující podkapitola.

5.1. Struktura hlavního okna.

Třída `MainWindow` dědí z `QMainWindow`. Objekt třídy `QMainWindow` je widget, který představuje hlavní okno aplikace, které může obsahovat menu a lištu s tlačítky, tzv. *toolbar*. Objekt typu `QMainWindow` navíc musí obsahovat widget označený jako *Central Widget*, na který je umístěn vlastní obsah okna. Některé widgety, které jsou umístěny na `MainWindow` vidíme na obr. 5.1. Na tomto obrázku také vidíme, že kromě `MainWindow` program Varan definuje dva nové widgety `Field` a `ColorBar`.



Obrázek 5.1: *Umístění widgetů na hlavním okně.*

Dříve, než vytvoříme menu a toolbary, musíme vytvořit tzv. akce, tj. objekty třídy `QAction`. Objekty třídy `QAction` jsou příkladem objektů frameworku Qt, které nejsou widgety (nezobrazují se). Následuje příklad vytvoření akce a nastavení vlastností této akce.

```
exitAct = new QAction(QIcon(":/images/exit.png"), "Exit", this);
connect(exitAct, SIGNAL(triggered()), this, SLOT(close()));
```

V konstruktoru je této akci přiřazena ikona, název a nadřazený objekt (v našem případě objekt třídy `MainWindow`). Ve druhém řádku vidíme použití signálů a slotů. `triggered()` je signál definovaný ve třídě `QAction`, který se zavolá, když je daná akce spuštěna. Spuštění akce se provede např. kliknutím na položku menu, které tuto akci přiřadíme, jak uvedeme později. `close()` je slot patřící objektu třídě `MainWindow` a způsobí zavření hlavního okna a ukončení celé aplikace.

Nyní můžeme definovat menu a toolbary. Nejprve uvedeme příklad definice menu:

```
fileMenu = menuBar()->addMenu("File");
fileMenu->addAction(openAct);
fileMenu->addAction(exportImageAct);
fileMenu->addSeparator();
fileMenu->addAction(exitAct);
```

`fileMenu` je objekt třídy `QMenu`. Funkce `QMainWindow::menuBar` vrací místo, na kterém se v hlavním okně vytvoří menu. Na tomto místě vytvoříme nové menu označené **File**. Jednotlivé položky menu vytvoříme tak, že jim přiřadíme předem definované akce. Název položky menu, její ikona a funkcionalita jsou určeny přiřazenou akcí. Na posledním řádku tohoto příkladu vidíme, jak je položce menu přiřazena akce `exitAct`, jejíž definici jsme uvedli dříve. Proto je této položce přiřazeno jméno **Exit**, ikona ze souboru `:/images/exit.png` a vybráním této položky se ukončí celá aplikace.

Podobně definujeme toolbary. Všechny položky toolbarů tvoří stejné akce, které jsou již definovány pro položky menu, a není třeba vytvářet nové.

Pro umístění dalších widgetů do hlavního okna vytvoříme nový widget a pomocí funkce `QMainWindow::setCentralWidget` jej nastavíme jako Central Widget. To provede následující kód.

```
QWidget *widget = new QWidget;
setCentralWidget(widget);
```

Na tento widget již můžeme umístit ostatní widgety. O rozvržení widgetů se stará tzv. *layout*. Jedná se o „neviditelný“ widget, na který umísťujeme další widgety a layout se postará o jejich umístění na dané ploše. Qt nabízí různé druhy layoutů, které zarovnávají widgety např. vertikálně, horizontálně, nebo do mřížky. Na layout můžeme také umístit jiný layout a vytvářet tak složitější struktury rozvržení widgetů.

Dále uvedeme příklad, jak pomocí signálů a slotů propojíme widget třídy `QCheckBox` s widgetem třídy `Field`. Widget třídy `Field` zobrazuje vizualizaci pole.

```
field = new Field;
⋮
QCheckBox *grid = new QCheckBox("Grid");
connect(grid, SIGNAL(toggled(bool)), field, SLOT(grid(bool)));
```

V tomto příkladě jsme propojili signál `QCheckBox::toggled(bool)` se slotem `Field::grid(bool)`. Signál `QCheckBox::toggled(bool)` představuje změnu stavu checkboxu. Argument typu `bool` signálu `QCheckBox::toggled(bool)` udává, zda je checkbox po změně stavu zapnutý, nebo vypnutý. Ve slotu `Field::grid(bool)` reagujeme na stav checkboxu zobrazením, nebo vypnutím zobrazení sítě triangulace v právě zobrazovaného poli. Podobně jsou implementovány i ostatní checkboxy.

Widget třídy `MainWindow` se také stará o zobrazení dialogu pro vybrání souboru s daty k vizualizaci. Následující příklad ukazuje, jak pomocí funkce `QFileDialog::getOpenFileName` zobrazíme požadovaný dialog. Pomocí argumentů funkce `QFileDialog::getOpenFileName` nastavíme dva možné filtry zobrazování souborů a to buď soubory končící příponou **.dat**, nebo všechny soubory. Návratová hodnota funkce `QFileDialog::getOpenFileName` je textový řetězec, který obsahuje cestu k vybranému souboru. Tento textový řetězec pak předáme widgetu třídy `Field`, který načte data z tohoto souboru a zobrazí je.

```
QString fileName = QFileDialog::getOpenFileName(this, "Open File",
         ".", "Dat Files (*.dat);;All Files (*.*)");
if (!fileName.isNull() ){
    field->loadFile(fileName);
}
```

Podobně je implementován i dialog pro export aktuálního zobrazení do souboru s obrázkem.

5.2. Načítání dat ze souboru.

Třída `fileParser` slouží k načítání dat ze souboru a jejich uložení v paměti. Struktura dat v souboru je popsána v předchozí kapitole. Soubor se postupně načítá po řádcích a v každém řádku vyhledáváme data požadované struktury pomocí regulárních výrazů. Pro práci s regulárními výrazy nabízí Qt třídu `QRegExp`. Příklad použití regulárního výrazu, který hledá tři kladná celá čísla oddělená mezerou vypadá takto:

```
QRegExp intsRx("^\\s*(\\d+)\\s+(\\d+)\\s+(\\d+)");
int pos=intsRx.indexIn(line);
    if (pos>=0){ //maching pattern found
        int vert1=intsRx.cap(1).toInt();
        int vert2=intsRx.cap(2).toInt();
        int vert3=intsRx.cap(3).toInt();
        ⋮
```

Regulární výraz `^` představuje začátek textu (v našem případě začátek řádku), `\\s` představuje tzv. *whitespace*, neboli neviditelný znak jako mezera nebo tabulátor. Znak `*` za regulárním výrazem znamená nula nebo více výskytů tohoto regulárního výrazu a znak `+` za regulárním výrazem znamená jeden nebo více výskytů.

Funkce `QRegExp::indexIn` vrací celé číslo větší než -1, pokud text, zadaný v argumentu této funkce, obsahuje hledaný regulární výraz. Pomocí funkce `QRegExp::cap` dostáváme text odpovídající části regulárního výrazu oddělené závorkami. V našem případě tuto funkci používáme k načítání jednotlivých čísel, což používáme při načítání indexů vrcholů jednotlivých trojůhelníků. Podobně používáme regulární výrazy při čtení většiny dat ze souboru.

5.3. Zobrazení vizualizace.

Pro zobrazení vizualizace používáme widget třídy `Field` a widget třídy `colorBar`. Widget třídy `Field` představuje obdelník, do kterého se vykresluje zobrazovaná vizualizace. Widget třídy `colorBar` představuje barevnou stupnici, na které je vyznačena hodnota najmenší a největší výšky ve skalárním poli, případně délka nejkratšího a nejdelšího vektoru vektorového pole.

Widget třídy `Field` dostává informaci o jménu souboru, ve kterém jsou uložena data, která má zobrazit. Toto jméno předá objektu třídy `fileParser`, který ze souboru načte požadované informace. `fileParser` rovněž zjistí hodnoty, které se zobrazují na barevné stupnici. Tyto hodnoty se pomocí signálů a slotů předají widgetu třídy `colorBar`, který je zobrazí.

Widget třídy `Field` se stará o vykreslení vizualizace dat načtených pomocí objektu třídy `fileParser`. Pro kreslení na widget se používá objekt třídy `QPainter`. Objekt třídy `QPainter` poskytuje např. funkce pro kreslení čar a bodů, nebo vkládání textových řetězců na dané souřadnice. Pomocí těchto funkcí se vykresluje požadovaná vizualizace. Třída `QPainter` navíc poskytuje užitečné funkce pro transformace souřadnicového systému. Například `QPainter::translate` posouvá systém souřadnic a funkce `QPainter::scale` mění měřítko na souřadnicových osách. Pomocí těchto dvou metod provádíme posouvání zobrazované vizualizace a také přibližování a oddalování pohledu. Objekt třídy `QPainter` můžeme nastavit tak, aby místo kreslení na widget kreslil na oběkt třídy `QImage` představující obrázek. Tento obrázek pak můžeme uložit do souboru. Takto je řešen export

zobrazované vizualizace do souboru s obrázkem.

Třída `QWidget` nám poskytuje virtuální funkce, které se volají při různých událostech. Například funkce `QWidget::mouseMoveEvent` se volá při pohnutí kurzoru myši a jako argument obsahuje informace o stavu, ve kterém se myš nachází jako souřadnice kurzoru, nebo zda je stisknuté nějaké tlačítko myši. Implementací této virtuální funkce ve třídě `Field` řešíme posouvání aktuálního zobrazení jako reakci na pohyb myši se stisknutým levým tlačítkem (tzv. drag) po widgetu.

Qt také postkytuje podporu pro programování vláken (threads) pomocí oběktů dědících ze třídy `QThread`. Program Varan vytváří vlákno pro výpočty při vizualizaci polí typu $S1$. Tyto výpočty jsou pro triangulace, které pokrývají větší oblasti, příliš dlouhé a mají vliv na plynulost ovládání aplikace. Proto se například při přiblížení vizualizace pole typu $S1$ spustí výpočet ve speciálním vlákně a program vykreslí pouze síť triangulace bez vybarvení. Vybarvení se zobrazí až poté, co vlákno dokončí svůj výpočet.

Závěr.

Podařilo se nám vytvořit program pro vizualizaci diskretizovaných polí a na zobrazení výsledků praktických úloh jsme ukázali jeho použití. Použití frameworku Qt se ukázalo jako vhodné k řešení jak grafického rozhraní programu, tak i zobrazení samotné vizualizace.

Literatura

[1] Süli, Endre: *Finite Element Methods for Partial Differential Equations*

[2] Uppadhay, C. S.: *Mechanical - Finite Element Method*, video z přednášek `http://www.youtube.com/user/nptelhrd#g/c/A4CBD0C55B9C3878`

[3] Článek na wikipedii o metodě konečných prvků: `http://en.wikipedia.org/wiki/Finite_element_method`

[4] Bouchala, Jiří: *Variační metody*, skripta ve vývoji

[5] Blaheta, Radim: *Matematické modelování metoda konečných prvků*, skripta ve vývoji

[6] Blanchette, J., Summerfield, M.: *C++ GUI Programming with Qt 4*, Prentice Hall PTR, 2006

Seznam příloh.

- CD se zdrojovými kódy programu Varan a soubory s daty k vizualizaci.