VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV MATEMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF MATHEMATICS

VIZUALIZACE DAT NA PLATFORMĚ .NET

DATA VISUALIZATION FOR .NET PLATFORM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ONDŘEJ ŠELIGA
AUTHOR

VEDOUCÍ PRÁCE — RNDr. RUDOLF HLAVIČKA, CSc.
SUPERVISOR

BRNO 2012

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav matematiky
Akademický rok: 2011/2012

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

student(ka): Ondřej Šeliga

který/která studuje v **bakalářském studijním programu**

obor: **Matematické inženýrství (3901R021)**

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Vizualizace dat na platformě .NET

v anglickém jazyce:

Data Visualization for .NET platform

Stručná charakteristika problematiky úkolu:

Pomocí knihovny VTK/Activiz.NET (open source) vytvořit v jazyce C# komponenty pro jednoduché vizualizace sítí a postprocessing metody konečných prvků.

Cíle bakalářské práce:

Pilotní projekt. Jednoduchá knihovna tříd pro vizualizace MKP sítí pro .NET a Matlab.

Seznam odborné literatury:

1) The VTK User's Guide. 11th Edition (March 2010)
2) Pro .NET 2.0 Windows Forms and Custom Controls in C#

Vedoucí bakalářské práce: RNDr. Rudolf Hlavička, CSc.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2011/2012.

V Brně, dne 5.11.2010

L.S.

prof. RNDr. Josef Šlapal, CSc. Ředitel ústavu	prof. RNDr. Miroslav Doupovec, CSc., dr. h. c. Děkan fakulty

Abstrakt

Bakalářská práce pojednává o způsobech a postupech při vizualizaci dat na platformě .NET prostřednictvím vizualizační knihovny ActiViz .NET. Jedná se o průvodní dokument k vyvinuté aplikaci Vizualizace, která demonstruje použití ovládacích prvků Control2D a Control3D. Popisuje způsoby práce a možnosti uvedených programových komponent.

Summary

Bachelor thesis deals with methods and procedures of data visualization for .NET platform using ActiViz .NET visualization library. This is an accompanying document to the developed application Vizualizace, which demonstrates how to use controls Control2D and Control3D. It describes ways of working and possibilities of these program components.

Klíčová slova

Vizualizace sítí, platforma .NET, Windows Forms Control

Keywords

Mesh visualization, .NET platform, Windows Forms Control

Bibliografická citace

ŠELIGA, O. *Vizualizace dat na platformě .NET.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2012. 48 s. Vedoucí bakalářské práce RNDr. Rudolf Hlavička, CSc.

BRNO 2012

Čestné prohlášení

Prohlašuji, že je bakalářská práce „Vizualizace dat na platformě .NET“ mým původním dílem, zpracoval jsem ji samostatně a s použitím odborné literatury a pramenů uvedených na seznamu, jenž je součástí této práce.

V Brně dne 25. května 2012

_____

Ondřej Šeliga

Poděkování

Děkuji tímto panu RNDr. Rudolfu Hlavičkovi, CSc. a Ing. Miloslavu Šeligovi za věnovaný čas, poskytnuté rady a motivaci při vypracování této bakalářské práce. Dále děkuji Mgr. Petře Čaníkové za rady poskytnuté při typologických úpravách tohoto textu.

_____
Ondřej Šeliga

BRNO 2012

Obsah

Úvod

Vizualizace sítí je součástí obecného procesu vědeckého modelování. Tento proces je používán k simulaci abstraktního matematického modelu, který reflektuje reálné chování systému popsané pomocí matematických rovnic. Cílem vědeckého modelování je snaha napodobit reálné chování, simulovat ho na vytvořeném modelu a následně ovlivnit pomocí požadovaných či dostupných prostředků.

Simulovaný model je přitom pouhým přiblížením se skutečnosti, reálný systém bývá natolik složitý, že jeho exaktní simulace není možná. Součástí procesu modelování proto bývají ověřovací experimenty, při kterých se snažíme zpřesňovat parametry modelu, nebo model samotný.

S růstem informační techniky a její dostupnosti se dnes vědecké modelování stává nedílnou součástí při výzkumu v oblastech fyziky, chemie či biologie a přináší nový pohled na chování systému v reálných situacích. Nutným předpokladem k vizualizaci je výpočetní technika a vhodná volba softwarových prostředků, s jejichž pomocí je vizualizace docíleno.

Úkolem mé bakalářské práce je přispět na poli těchto prostředků vytvořením vlastních programových komponent, které zjednoduší použití obsáhlé vizualizační knihovny ActiViz .NET dalším vývojářům. Způsobům programování v prostředí této knihovny je věnována 2. kapitola práce, o volbě prostředků při tvorbě komponent pak pojednává 1. kapitola. Komponenty by měly být vytvořeny především jako prostředek pro vizualizaci fyzikálních modelů, jejichž stavy jsou popsány parciálními diferenciálními rovnicemi, a to jak ve dvoudimenzionální, tak třídimenzionální oblasti (George et al., 2008, s. 11). Jednou z nejpoužívanějších metod pro řešení podobných systémů je metoda konečných prvků.

Závěrečná 3. kapitola práce pojednává o vyvinutých komponentách Control2D a Control3D z hlediska jejich vývojáře. Je zde popsána motivace pro vývoj podobných komponent, problémy, na které jsem osobně při jejich vývoji narazil, a návrhy na možné zdokonalení stávající funkčnosti. Důležitou součástí této kapitoly je také manuál, ve kterém se zájemce dozví, jak komponenty implementovat do svého projektu a jak s nimi následně zacházet.

1 Popis softwarových prostředků

Na začátku vývoje každé aplikace je programátor postaven před stěžejní otázku, jaký programovací jazyk a obecně vývojové prostředí k vytvoření aplikace použije. Při hledání odpovědi je důležité zvážit několik aspektů:

- aktuálnost programovacího jazyka a záruka jeho podpory do budoucna;
- kompatibilita s dalšími zvolenými subsystémy;
- schopnosti programovacího jazyka by měly být alespoň shodné s rozsahem zadání, ideálně by však měly být mnohem širší.

1.1 Platforma .NET Framework v prostředí jazyka C#

Jako základní stavební kámen pro vývoj komponent byl určen jazyk C#, který je dnes hojně využíván k vývoji formulářových aplikací primárně určených pro operační systém Microsoft Windows. Mezi výhody jazyka C# patří mimo kvalitního vývojového prostředí Microsoft Visual Studio a rozsáhlé podpory ze strany společnosti Microsoft také možnost použití funkcionality .NET Framework platformy.

Platforma .NET Framework byla vyvinuta společností Microsoft a uvedena ve verzi 1.0 v roce 2002. Vývojářům nabízí prostředí pro běh aplikace, které zajišťuje např. správu paměťových prostředků a bezpečnosti běhu aplikace (tzv. *Common Language Runtime*) a obsáhlou knihovnu tříd, která poskytuje základní funkcionalitu pro jejich aplikace (tzv. *.NET Framework Class Library*). Platforma .NET nepředepisuje použití žádného programovacího jazyka. Bez ohledu na to, v čem byla aplikace původně napsána, se vždy přeloží do mezi-jazyka Common Intermediate Language[1], čímž je zajištěna jazyková interoperabilita. Mezi výhody platformy .NET patří především:

- úspora času při vývoji díky *.NET Framework Class Library*;
- možnost implementace knihoven třetích stran, které mohou být díky jazykové interoperabilitě napsány v libovolném jazyce podporovaném platformou .NET;
- kvalitní podpora komunitou vývojářů ze strany společnosti Microsoft.

1.2 Vizualizační knihovna ActiViz .NET a systém VTK

Díky popularitě platformy .NET v oblasti vývoje desktopových aplikací, je dnes vyvíjeno mnoho knihoven, které rozšiřují možnosti platformy, případně implementují novou funkcionalitu. Na poli počítačové vizualizace je jedním z nejvýznamnějších přispěvatelů společnost Kitware, která pro platformu .NET vyvíjí knihovnu ActiViz .NET.

[1] Zdroj: Společný jazykový modul runtime (CLR). MSDN – Možnosti vývoje softwaru pro klientské počítače, web, cloud a telefony. MICROSOFT [online]. © 2012 [cit. 2012-05-20]. Dostupné z: <http://msdn.microsoft.com/cs-cz/library/8bs2ecf4.aspx>

Tato knihovna představuje integrační vrstvu pro Visualization Toolkit (VTK), tedy objektově orientovaný systém pro vizualizaci 3D grafiky, dat a zpracování informací (Kol. aut., 2008, s. 4).

Software VTK je vyvíjen již od prosince roku 1993, kdy Will Schroeder, Ken Martin a Bill Lorensen začali pracovat na své knize *The Visualization Toolkit An Object--Oriented Approach to 3D Graphics*, ke které se projekt tehdy vázal. Cílem projektu bylo vytvořit kvalitní open-source software pro vizualizace a 3D grafiku.[2] Vzhledem k otevřenosti kódu vznikla po vytvoření jádra VTK široká komunita vývojářů, kteří projekt postupně rozšiřovali a začali jej používat ve svých aplikacích. V roce 1998 pak vznikla společnost Kitware, která projekt dodnes zaštiťuje.

Celý systém VTK je napsán v jazyce C++ a je možné jej zdarma stáhnout. Jeho implementace do dnešního vývojového prostředí Microsoft se však značně liší od prostředí devadesátých let, je komplikované a vyžaduje široké znalosti jazyka C++ (Kol. aut., 2008, s. 4). V souvislosti se vznikem platformy .NET byl proto společností Kitware vyvinut software ActiViz .NET, který pomocí tzv. obálek (*wrapperů*) implementuje třídy původního projektu VTK a umožňuje tak jejich použití v moderních programovacích jazycích, jako je C# a Visual Basic.

1.3 Ovládací prvky WinForm User Control

Ve vývojovém prostředí Microsoft Visual Studio je formulářová aplikace tvořena pomocí tzv. Windows Forms Designeru, díky kterému je možné interaktivně měnit vzhled a vlastnosti formuláře a jeho komponent (tzv. ovládacích prvků). Interaktivita tvorby formuláře spočívá v přetahování jednotlivých ovládacích prvků z Toolboxu, tedy seznamu všech dostupných prvků, přímo do formulářové aplikace. Zdrojový kód určující vlastnosti komponent a samotného formuláře (např. výška, barva apod.) je pak Designérem generován automaticky.

Jako příklad formulářového ovládacího prvku je často uváděno tlačítko, ovládacím prvkem je ale také například horní menu, vstupní pole pro text, záložka nebo vymezený prostor pro jiné komponenty. Jak uvádí Sells (2006, s. 16), lze ovládací prvky obecně rozdělit na:

- předem definované komponenty, které jsou připraveny tvůrci MS Visual Studia a jsou tak dostupné ihned po své instalaci;
- komponenty definované uživatelem (vývojářem), tedy tzv. uživatelské komponenty.

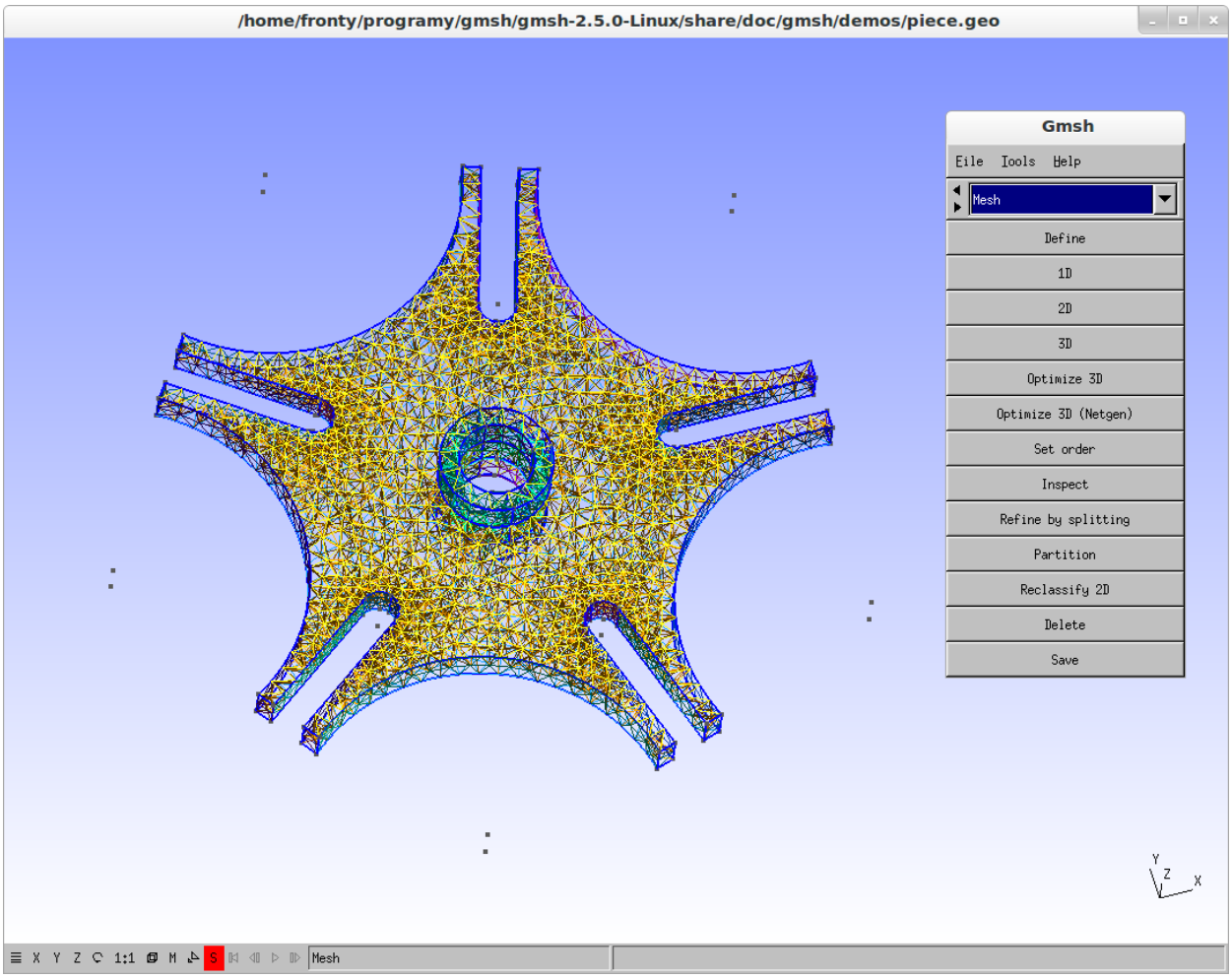
[2] Zdroj: About. *VTK - The Visualization Toolkit.* USA: Kitware, Inc. [online]. © 2012 [cit. 2012-05-20]. Dostupné z <http://www.vtk.org/VTK/project/about.html>

Při tvorbě jednoduché formulářové aplikace si většinou vývojář vystačí s předdefinovanými ovládacími prvky. Vznikne-li však potřeba použít určitou část formuláře na několika rozdílných místech aplikace, případně v rámci zcela jiné aplikace, je vhodné definovat vlastní uživatelskou komponentu.

Jak možná pozorný čtenář zaregistroval, uživatelská komponenta je podobně jako formulářová aplikace složena z dalších předdefinovaných a uživatelských ovládacích prvků. Hlavní rozdíl mezi uživatelskou komponentou a formulářem je tak především ve schopnosti spouštění. Zatímco formulář lze spustit ihned po jeho vytvoření, komponenta musí být nejprve přiřazena do formuláře, a ten je pak možné spustit. V prostředí Visual Studia také není uživatelský ovládací prvek při svém vytváření nijak definován z hlediska vzhledu, zatímco formulář má ve výchozím stavu rámeček a ikony pro zavření, minimalizaci a maximalizaci.

1.4 GMSH - export dat

Při práci na komponentách Control2D a Control3D bylo nutné získat testovací data, na kterých bude funkčnost ovládacích prvků zkoušena. K těmto účelům byl zvolen freeware GMSH, generátor 3D sítí se zabudovaným post-procesorem. Důležitou výhodou, která hrála rozhodující úlohu při volbě této aplikace, je možnost exportu sítě do textového souboru. Vyexportovaný soubor je pak zpracován v aplikaci Vizualizace pomocí třídy GMSHFileParser. Další předností tohoto programu je kompatibilita s operačními systémy Windows, Linux a MacOS.



Obr. 1: Aplikace GMSH v OS Linux Ubuntu

Aplikace GMSH je složena ze čtyř funkčních celků, tzv. modulů, podle kterých je rozděleno také uživatelské menu.

1.4.1 Geometrie

Modul geometrie umožňuje uživateli vytvořit si vlastní geometrický objekt. Geometrické objekty jsou v aplikaci tvořeny elementárními geometrickými entitami, které je možné zadat přímým kliknutím nad kreslící plochou, u vybraných entit také za pomoci parametrů. V příslušné části aplikačního menu jsou k dispozici entity:

- bod,
- úsečka,
- křivka,
- rovinná či zakřivená plocha.

Ze skupiny elementárních entit lze v modulu geometrie dále vytvořit fyzikální skupinu, se kterou je následně možné manipulovat jako s celistvým objektem.

Vedle poněkud zdlouhavého vytváření vlastních geometrických objektů jsou po instalaci aplikace k dispozici také již vytvořené demo objekty, kterých jsem při testování svých ovládacích prvků využíval.

1.4.2 Síť

Po načtení či vytvoření geometrického objektu je skrze tento modul možné aplikovat algoritmus, který nad objektem vytvoří síť. K dispozici je vytvoření sítě pouze na površích (tzv. 2D síť), nebo v celém objemu objektu (tzv. 3D síť). V nastavení aplikace v sekci věnované sítím je mimo jiné možné zvolit algoritmus pro 2D a 3D síťování, jemnost sítě a tzv. shrink faktor, tedy smrštění elementárních geometrických entit sítě.

1.4.3 Řešič diferenciálních rovnic (solver)

Aplikace obsahuje zabudovaný GetDP algoritmus pro řešení diferenciálních rovnic, v případě nedostatku výkonu lokálního stroje je také možné využít až pěti algoritmů připojených přes TCP/IP, nebo Unix socket[3].

1.4.4 Post-procesor

Díky modulu post-procesor je možné v aplikaci GMSH simulovat chování objektu při působení určitých fyzikálních vlivů. Modulu jsou pro každou geometrickou entitu předána skalární či vektorová data vypočítaná řešičem diferenciálních rovnic, jejichž hodnoty jsou pak při vizualizaci použity pro generování barevných map, vrstevnic či povrchů (skalární data) nebo šipek (vektorová data).

[3] GEUZAINE, Christophe; Jean-François REMACLE. Solver: External Solver Interface. *GMSH 2.5* [online]. © 2010 [cit. 2012-05-20]. Dostupné z <http://www.geuz.org/gmsh/doc/texinfo/gmsh.html#Solver>

2 Programování v prostředí knihovny ActiViz .NET

Pro účely vizualizace dat budeme v prostředí knihovny ActiViz .NET pracovat pouze se zlomkem jejich tříd. Vzhledem k objektové orientaci knihovny je vizualizace a funkčnost s ní spojená realizována zakládáním instancí objektů, které jsou následně spojovány podle vhodných vzorů (Kol. aut., 2008, s. 15). Při hledání těchto vzorů je dobré získat představu o filozofii celého vizualizačního procesu, od globálních principů vizualizace se totiž odvíjí konkrétní programátorské postupy. Zároveň je nutné si při práci s třídami knihovny zvyknout na některé nestandardní konvence způsobené implementací původního C++ zdrojového kódu VTK do prostředí .NET. Příkladem může být způsob zakládání instancí objektů, který je realizován podle návrhového vzoru Singleton. Každá třída knihovny tak obsahuje statickou metodu New(), instance je pak získána voláním této metody.

2.1 Filozofie vizualizace

Celý proces vizualizace pomocí knihovny ActiViz .NET je možné si v teoretické rovině představit jako divadelní vystoupení. Tuto asociaci využívá také společnost Kitware ve svých publikacích při pojmenování jednotlivých tříd. Vizualizace je tedy obecně realizována v následujících krocích (dle Kol. aut., 2008, s. 16):

1. Vytvoření jednoho nebo více *herců*, tedy objektů, které budou zobrazeny. V knihovně ActiViz .NET se jedná o třídu **vtkActor**.
2. Aplikace *kamery*, pomocí které je 3D objekt herce zobrazen na 2D monitor. Jedná se o třídu **vtkCamera**.
3. Osvětlení scény pomocí *světla*, tedy třídy **vtkLight**.
4. Materiálové vlastnosti (např. odlesk) herce mohou být upraveny pomocí třídy **vtkProperty**.
5. Příprava *jevišt*ě, na kterém se budou herci zobrazovat pomocí třídy **vtkRenderer**.
6. Jedno nebo více jevišť nakonec tvoří *divadlo*, tedy třídu **vtkRenderWindow**.

Vedle těchto stěžejních procesů mohou další probíhat tzv. v zákulisí, tedy před samotným renderováním. Tak lze např. nastavit interakce uživatele s hercem (možnost ovládání vizualizovaného objektu pomocí klávesnice a myši) nebo přiřazovat hercům textury.

2.2 Vizualizace z hlediska programátora

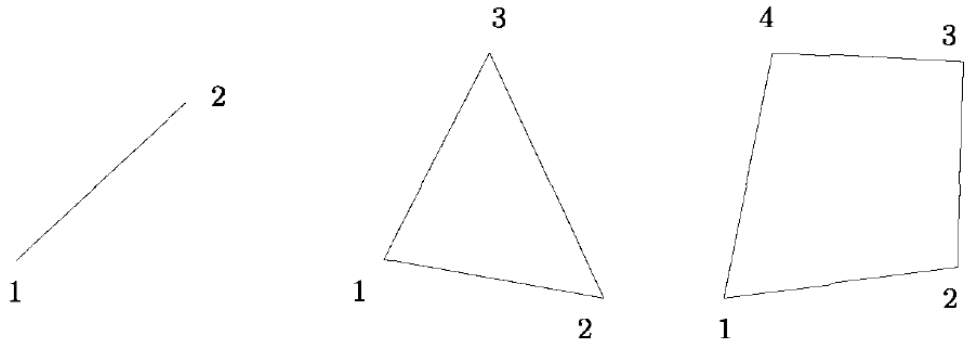
Vizualizační proces realizovaný pomocí knihovny ActiViz .NET je z hlediska aplikace koncepčně rozdělen do dvou kroků. Prvním krokem je získání dat ve formě datových objektů buněk, jejich naplnění do datových souborů a zpracování pomocí filtrů, druhým krokem je pak vytvoření herce, předání herce do renderovacího subsystému a jeho vykreslení (renderování). Pro vytvoření herce v druhém kroku, z datového souboru vytvořeného v prvním kroku, slouží jako mezikrok mapování zpracovaných dat pomocí tzv. mapperu.

Z pohledu objektově orientovaného přístupu se vizualizační proces skládá z objektů reprezentujících data (datové objekty) a objektů, které s daty pracují (procesní objekty) (Schroeder et al., 2002, s. 85).

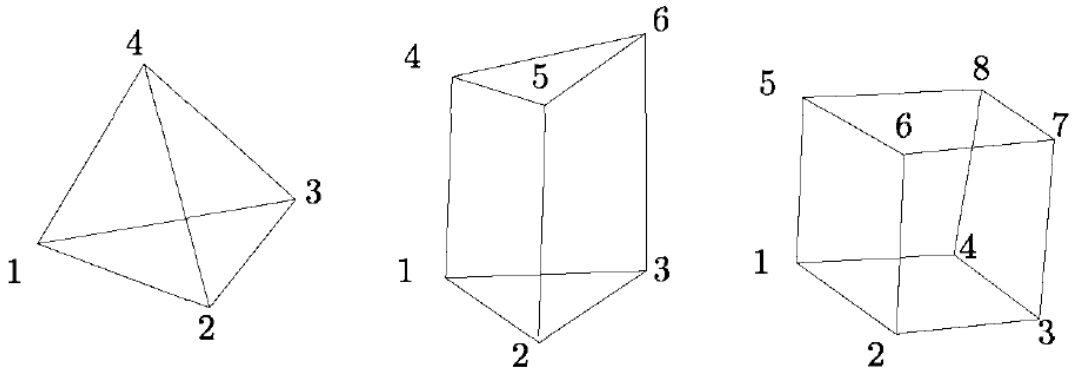
V následujícím textu popíšeme datové objekty, způsoby jejich plnění a práce s nimi, později se budeme věnovat procesním objektům a vykreslení dat. Kapitola je proložena názornými ukázkami komentovaných zdrojových kódů, jejichž výsledkem je aplikace zobrazující jednoduché geometrické útvary, na nichž je aplikováno smrštění pomocí tzv. shrink filtru, proložení skalárním polem, a zobrazení vrstevnic. Celá aplikace včetně zdrojových kódů je k dispozici na přiloženém CD pod názvem mojeVizualizaceActiviz.

2.2.1 Buňky sítě

Síť je z obecného hlediska složena z elementárních geometrických útvarů, které budeme nazývat buňky. „Každá buňka sítě je určena geometrickým tvarem (trojúhelník, čtyřúhelník, atd.) a svými vrcholy. Pro přesné definování buňky včetně jejich hran a stěn (v případě trojrozměrného prostoru) je při určování vrcholů nutné dodržet konvenci jejich pořadí.“ (George et al., 2008, s. 27) Buňky je možné rozdělit na dvojrozměrné, kterými jsou tvořeny sítě povrchů těles, a trojrozměrné, jimiž jsou tvořeny sítě vnitřních částí (objemů) prostorových objektů.



Obr. 2: Základní dvojrozměrné buňky včetně pořadí číslování vrcholů (George et al., 2008, s. 28)



Obr. 3: Základní trojrozměrné buňky včetně pořadí číslování vrcholů (George et al., 2008, s. 28)

Před určením buněk je v knihovně ActiViz .NET nejprve nutné popsat body (vrcholy), kterými jsou definovány všechny buňky v síti. Pro reprezentaci množiny bodů je používána třída vtkPoints, jejímiž stěžejními metodami jsou InsertPoint() pro vložení bodu a GetPoint() pro získání souřadnic bodu. Každý bod je v tomto datovém objektu definován svými souřadnicemi a unikátním celočíselným identifikátorem, kterého se pak využívá při konstrukci buněk sítě. Identifikátor je na rozdíl od běžných programátorských konvencí číslován od 1. V následující ukázce definujeme 4 body umístěné v počátku souřadného systému, na ose *x*, *y* a *z*.

```
/** Definice bodů **/
//založení instance datového objektu vtkPoints
vtkPoints points = vtkPoints.New();
//vložení bodu o souřadnicích [0, 0, 0] na pozici 1
points.InsertPoint(1, 0, 0, 0);
points.InsertPoint(2, 1.0, 0, 0);
points.InsertPoint(3, 0, 1.0, 0);
points.InsertPoint(4, 0, 0, 1.0);
```

Nyní můžeme přistoupit k naplnění datových objektů buněk. Definice vrcholů buněk probíhá v relaci s polem bodů, vrcholy již tedy nebudeme určovat souřadnicemi, ale pomocí identifikátorů bodů z objektu vtkPoints. Pro většinu typů buněk je v knihovně ActiViz .NET připravena reprezentace ve formě tříd.

Plnění každé buňky musí probíhat podle výše zmíněné konvence v pořadí bodů. Často používanými datovými objekty dvojrozměrných a trojrozměrných buněk jsou:

- **vtkLine,** reprezentující úsečku,
- **vtkTriangle,** reprezentující trojúhelník,
- **vtkQuad,** reprezentující obecný čtyřúhelník,
- **vtkPixel,** reprezentující čtverec,
- **vtkTetra,** reprezentující čtyřstěn,
- **vtkWedge,** reprezentující trojboký hranol,
- **vtkHexahedron,** reprezentující obecný šestistěn,
- **vtkVoxel,** reprezentující krychli.

Datové reprezentace buněk plníme pomocí metody GetPointIds(), která vrací objekt se seznamem bodů. Do tohoto seznamu metodou SetId() přidáme vrchol určením jeho pořadí (první vrchol buňky má pořadí 0) a identifikátoru bodu z pole points. Pro vytvoření množiny buněk používáme třídu vtkCellArray(), které předáváme instanci objektu buňky jako parametr metody InsertNextCell(). Vytváření buněk a jejich množin je zřetelné z ukázky zdrojového kódu, ve kterém jsou pomocí identifikátorů bodů z objektu points vytvořeny tři úsečky, dva trojúhelníky a jeden čtyřstěn.

```
/** Definice úseček **/
//pole buněk pro úsečky
vtkCellArray lines = vtkCellArray.New();

//definice první úsečky
vtkLine line1 = vtkLine.New();
//první bod úsečky (index 0), v objektu points na pozici 1
line1.GetPointIds().SetId(0, 1);
//druhý bod úsečky (index 1), v objektu points na pozici 2
line1.GetPointIds().SetId(1, 2);
//vložení úsečky do pole buněk (úseček)
lines.InsertNextCell(line1);

vtkLine line2 = vtkLine.New();
line2.GetPointIds().SetId(0, 1);
line2.GetPointIds().SetId(1, 3);
lines.InsertNextCell(line2);

vtkLine line3 = vtkLine.New();
line3.GetPointIds().SetId(0, 1);
line3.GetPointIds().SetId(1, 4);
lines.InsertNextCell(line3);

/** Definice trojúhelníků **/
vtkCellArray triangles = vtkCellArray.New();

vtkTriangle triangle1 = vtkTriangle.New();
triangle1.GetPointIds().SetId(0, 1);
triangle1.GetPointIds().SetId(1, 2);
triangle1.GetPointIds().SetId(2, 3);
triangles.InsertNextCell(triangle1);

vtkTriangle triangle2 = vtkTriangle.New();
triangle2.GetPointIds().SetId(0, 1);
triangle2.GetPointIds().SetId(1, 3);
triangle2.GetPointIds().SetId(2, 4);
triangles.InsertNextCell(triangle2);
```

```
/** Definice čtyřstěnů **/
vtkCellArray tetrahedrons = vtkCellArray.New();
vtkTetra tetra = vtkTetra.New();
tetra.GetPointIds().SetId(0, 1);
tetra.GetPointIds().SetId(1, 2);
tetra.GetPointIds().SetId(2, 3);
tetra.GetPointIds().SetId(3, 4);
tetrahedrons.InsertNextCell(tetra);
```

2.2.2 Datové soubory

Datové objekty bodů a buněk sítě jsou před dalším zpracováním načítány do jednoho z pěti připravených datových souborů, které se liší svou geometrickou interpretací a způsobem manipulace s daty. Správná volba datového souboru je důležitá především při vizualizaci velkého množství dat, kdy jsou díky optimalizovaným algoritmům jednotlivých souborů minimalizovány paměťové nároky na výpočetní techniku.

vtkImageData

Nejjednodušší a nejkompaktnější je datový soubor, který je definován pouze souřadnicemi počátečního bodu, dimenzemi (počty bodů podél os *x, y* a *z*) a vzdáleností (roztečí) bodů podél jednotlivých os. Body i buňky souboru jsou tak určeny implicitně, jejich souřadnice jsou dopočítány automaticky. Jednotlivé dimenze definují topologii; vzdálenosti a počáteční bod pak geometrii objektu. Identifikátory jsou bodům přiděleny podle rostoucích hodnot souřadnic *x, y, z* (Schroeder et al., 2002, s. 132).

vtkRectilinearGrid

Topologie tohoto datového souboru je pravidelná a je stejně jako v případě souboru *vtkImageData* určena dimenzemi. Geometrie je definována třemi vektory (z hlediska programu se jedná o pole) s hodnotami *x, y,* resp. *z* souřadnic. Kombinací těchto tří polí je možné jednoznačně určit souřadnice kteréhokoli bodu v datovém souboru. Číslování bodů probíhá stejně jako u datového souboru *vtkImageData* (Schroeder et al., 2002, s. 133).

vtkStructuredGrid

Datový soubor je určen pravidelnou topologií definovanou dimenzemi v topologickém souřadném systému. Geometrie souboru je definována explicitně předáním objektu vtkPoints (Schroeder et al., 2002, s. 133).

vtkPolyData

Datový soubor pro libovolné kombinace 2D geometrických objektů. Topologie souboru již není pravidelná, proto je nutné poskytnou body i buňky explicitně. Body jsou souboru předávány v instanci objektu vtkPoints, v případě buněk jsou předávány instance objektů vtkCellArray společně s typem buněk, který se v objektu nachází (Schroeder et al., 2002, s. 133).

vtkUnstructuredGrid

Nejobecnější datový soubor, kterým lze definovat libovolný prostorový či rovinný objekt bez požadavků na jeho pravidelnost. Body i buňky jsou souboru předávány stejným způsobem, jako v případě souboru vtkPolyData (Schroeder et al., 2002, s. 134-135).

Výše popsané datové soubory, slouží k vytvoření sítě obecných geometrických objektů a jejich naplnění je tak nutné provést manuálně. V knihovně ActiViz .NET existují vedle těchto obecných souborů také datové soubory některých geometrických těles, jejichž data jsou naplněna po určení charakteristických rozměrů tělesa. Velice jednoduše tak lze vytvořit např. sféru pouhým zadáním poloměru a požadované jemnosti výsledné sítě.

Datové soubory vtkImageData, vtkRectilinearGrid a vtkStructuredGrid jsou sice nenáročné a zpracování dat v nich obsažených probíhá rychle, jejich použití je však limitováno předpokladem pravidelné struktury sítě a přichází tak v úvahu pouze ve speciálních případech. Nejčastěji proto vývojář sáhne po souboru vtkPolyData, nebo vtkUnstructuredGrid. V následující ukázce zdrojového kódu jsou předávány body, úsečky, trojúhelníky a čtyřstěn z předchozích ukázek do datového souboru typu vtkUnstructuredGrid.

```
/** Datová struktura **/
vtkUnstructuredGrid grid = vtkUnstructuredGrid.New();
//předání bodů
grid.SetPoints(points);
//předání úseček
grid.InsertNextCell(line1.GetCellType(), line1.GetPointIds());
grid.InsertNextCell(line2.GetCellType(), line2.GetPointIds());
grid.InsertNextCell(line3.GetCellType(), line3.GetPointIds());
//předání trojúhelníků
grid.InsertNextCell(triangle1.GetCellType(), triangle1.GetPointIds());
grid.InsertNextCell(triangle2.GetCellType(), triangle2.GetPointIds());
//předání čtyřstěnu
grid.InsertNextCell(tetra.GetCellType(), tetra.GetPointIds());
```

Mimo definici geometrie a topologie vizualizovaného objektu je ke každému bodu (resp. buňce) datového souboru možné přidat tzv. atributy, které představují dodatečná data pro daný bod, resp. buňku. Atributy jsou děleny do kategorií podle typu dat, která poskytují. Nejjednodušším typem jsou **skalární atributy**, které představují číselné hodnoty v bodě nebo buňce. V praxi je jich využito např. pro číselné hodnoty teploty, rychlosti, tlaku, hustoty, případně výšky. **Vektorové atributy** jsou určeny délkou a směrem, použití nacházejí např. při vizualizaci rychlosti proudění, trajektorie částic nebo pohybu větru. **Normálové atributy** jsou na rozdíl od vektorových atributů definovány pouze svým směrem, využívají se tak např. pro určení orientace buněk (Schroeder et al., 2002, s. 120-122).

Zřejmě nejčastějším použitím atributů je přiřazení skalárního pole a následná vizualizace jeho hodnot pomocí barevné škály. Na to je knihovna ActiViz .NET vhodně připravena. Správnou konfigurací mapperu lze totiž generovat zbarvení tělesa pomocí atributů bez nutnosti složitého vytváření vlastní barevné škály. Zmíněný způsob konfigurace mapperu bude popsán později, nyní pouze přiřadíme výsledné hodnoty skalárního pole ve formě atributů k bodům datového souboru.

Skalární atributy jsou nejprve naplněny do objektu vtkDoubleArray, což je obecný datový objekt VTK, reprezentující pole reálných čísel (datový typ double), případně vektorů reálných čísel. Vzhledem k obecnosti tohoto datového objektu je před naplněním nutné určit dimenzi jeho položek pomocí metody SetNumberOfComponents(). Datový objekt je v dalším kroku předán do datového souboru pomocí metody GetPointData(), která vrací objekt s atributy bodů. Metodou SetScalars() tohoto objektu pak pole vtkDoubleArray předáme v parametru.

V následující ukázce zdrojového kódu je použito skalární pole $f(x,y,z)=x+y+z$, jehož výsledné hodnoty jsou počítány ze souřadnic jednotlivých bodů. Souřadnice získáme pro každý bod metodou GetPoint() objektu vtkPoints, jejímž celočíselným parametrem určíme pozici bodu, jehož souřadnice požadujeme. V podmínce cyklu použijeme metodu GetNumberOfPoints() objektu vtkPoints, která vrátí počet bodů v datovém objektu.

```
/** Nastavení skalárních hodnot bodů **/
//vytvoření skalárního pole
vtkDoubleArray scalarField = vtkDoubleArray.New();
//pojmenování pole pro případnou pozdější identifikaci
scalarField.SetName("Scalar field");
//každá položka pole se skládá pouze z jedné hodnoty (skaláru)
scalarField.SetNumberOfComponents(1);

//skalární hodnoty počítáme pro každý bod sítě
for (int i = 1; i <= points.GetNumberOfPoints(); i++)
{
    //získáme souřadnice bodu na pozici i
    double[] point = points.GetPoint(i);
    //skalár počítáme jako součet hodnot souřadnic bodu
    double scalar = point[0] + point[1] + point[2];
    //skalár na pozici i shodnou s pozicí bodu v objektu points
    scalarField.InsertTuple1(i, scalar);
}
//skalární pole přiřadíme k bodům datové struktury
grid.GetPointData().SetScalars(scalarField);
```

V této fázi bychom již mohli přikročit k procesu mapování a následného renderování dat, v následujících podkapitolách však data sítě ještě upravíme pomocí filtrů.

2.2.3 Zpracování dat

Po naplnění vybraného datového souboru daty a případně atributy je možné aplikovat filtr, tedy předat instanci datového souboru do jedné ze tříd filtrů. Filtr určitým způsobem pozmění data souboru, může např. vyjmout část dat, rozdělit data na hrubší části, interpolovat soubor dat na jemnější rozlišení, sloučit více vstupních hodnot do jedné výstupní, nebo rozdělit jednu vstupní hodnotu na několik hodnot výstupních (Bell, 2012). Jak následně ukážeme, modifikace dat filtrem se při vizualizaci projeví např. vykreslením kontur (vrstevnic), nebo smrštěním buněk (tzv. shrink filtr).

Požadujeme-li, aby byla data pozměněna více rozdílnými způsoby, je možné zkombinovat několik filtrů. Realizace probíhá tak, že nejprve předáme do filtru A datový soubor, následně do filtru B předáme instanci filtru A, do filtru C instanci filtru B atd. Příkladem může být vytvoření sféry, kterou pomocí vtkTransformFilter transformujeme v elipsoid a následně pomocí vtkElevationFilter obarvíme podle vzdálenosti bodů od určité roviny.

Smrštění buněk sítě (shrink filtr)

Často používaným filtrem při vizualizaci sítě je tzv. shrink filtr, jehož výsledkem je smrštění, tedy zmenšení objemu buněk sítě bez změny pozice těžiště buňky. Buňky sítě jsou po aplikaci filtru rozpojeny, jejich hrany jsou posunuty směrem k těžišti buňky a zmenšeny.[4] Mezi buňkami tak vzniknou mezery, tvar celé sítě se však nezmění. Díky mezerám je možné pozorovat vnitřní strukturu vizualizované sítě také v povrchovém režimu, při kterém by jinak kvůli neprůhlednosti stěn buněk nebyla viditelná (viz příloha, obr. 19).

Shrink filtr je na pozadí knihovny ActiViz .NET realizován pomocí tzv. Barycentrických souřadnic. Podle Málkové (2006, s. 9) jsou Barycentrické souřadnice definovány následně:

„Nechť $q_1, q_2, ..., q_n$ je n vrcholů konvexního polygonu Q v R^2, kde $n \geq 3$. Dále nechť p je libovolný bod uvnitř Q. Barycentrickými souřadnicemi bodu p vzhledem k $\{q_j\}_{j=1}^{n}$ nazýváme každou n-tici $(\alpha_1, \alpha_2, ..., \alpha_n)$ závisející na Q a p takovou, že platí:

- $p = \sum_{j=1}^{n} \alpha_j q_j$, kde $\sum_{j=1}^{n} \alpha_j = 1$.
- $\{\alpha_j\}_{j=1}^{n}$ musí být číslo nekonečně diferencovatelné vzhledem k p a vrcholům Q. To zajistí hladkost změny koeficientů α_j při změne jakéhokoli vrcholu q_j.
- $q_j \geq 0$ pro $\forall\, j \in [1..n]$. To zajišťuje, že $\alpha_j \in <0,1> \ \forall\, j \in [1..n]$.

Pro účely vizualizace sítí, jejichž základní buňkou je trojúhelník nebo čtyřstěn, jsou zapotřebí Barycentrické souřadnice trojúhelníku. Ty lze spočítat podle následujících rovnic:

$$\alpha_1 = \frac{A(p,q_2,q_3)}{A(q_1,q_2,q_3)}$$
$$\alpha_2 = \frac{A(p,q_3,q_1)}{A(q_1,q_2,q_3)}$$
$$\alpha_3 = \frac{A(p,q_1,q_2)}{A(q_1,q_2,q_3)},$$

kde $A(p,q,r)$ označuje plochu trojúhelníku (p,q,r). Lze dokázat, že uvedené souřadnice splňují podmínky definice Barycentrických souřadnic.” (Málková, 2006, s. 9)

Shrink filtr je v prostředí knihovny ActiViz .NET dostupný v podobě třídy vtkShrinkFilter. Této třídě je nutné předat datový soubor prostřednictvím metody

[4] Zdroj: VTK: vtkShrinkFilter Class Reference. VTK Documentation [online]. © 1997-2012 [cit. 2012-05-19]. Dostupné z: <http://www.vtk.org/doc/release/4.2/html/classvtkShrinkFilter.html#_details>

SetInput() a nastavit koeficient smrštění jako reálné číslo z intervalu <0,1> pomocí metody SetShrinkFactor(). Před mapováním dat je ještě vhodné data filtru obnovit zavoláním metody Update().

```
/** Filtr pro smrštění buněk **/
vtkShrinkFilter shrinkFilter = vtkShrinkFilter.New();
//předání datové struktury do filtru
shrinkFilter.SetInput(grid);
//nastavení koeficientu smrštění
shrinkFilter.SetShrinkFactor(0.6);
//přepočítání po změně výchozí konfigurace
shrinkFilter.Update();
```

Zobrazení vrstevnic (kontur)

V případě vizualizace složitějších objektů, jejichž síť se skládá z mnoha buněk, nemusí být při vizualizaci skalárního pole průběh výsledných skalárů podle barev zřetelný. Pro zvýraznění průběhu funkce je v takových případech vhodné použít vrstevnice, též známé jako kontury nebo isokřivky (případně isoplochy). Vrstevnice jsou tvořeny množinou bodů se stejnou funkční hodnotou, v případě dvojrozměrného zobrazení se jedná o uzavřené křivky, v trojrozměrné oblasti jsou pak vrstevnice tvořeny plochami. Na tělese jsou vrstevnice vykresleny v pravidelných intervalech.

V knihovně ActiViz .NET je postup zobrazení vrstevnic složitější, než např. smrštění dat shrink filtrem. Základním objektem vrstevnic je vtkContourFiltr, kterému je nejprve metodou SetInput() předána instance datového souboru s výsledky skalárního pole v atributech bodů. Následně je pomocí metody GenerateValues() definován počet vrstevnic, minimální a maximální hodnota rozsahu skalárů. Vrstevnice jsou po konfiguraci objektu uvedenými metodami vytvořeny zavoláním metody Update(). Instance objektu vtkContourFiltr je dále předána třídě vtkStripper prostřednictvím metody SetInputConnection(). Objekt vtkStripper pak zavoláním metody Update() z instance vtkContourFilter vytvoří křivky (2D), nebo roviny (3D) vrstevnic.

Na rozdíl od filtru smrštění buněk, který pouze upravuje vzhled existujících dat, filtr kontur do vizualizačního okna přidává nová data. Proto je dále nutné vytvořit, naplnit a vhodně konfigurovat objekt vtkPolyDataMapper, ze kterého je následně vytvořen herec (třída vtkActor) reprezentující kontury. Pro zobrazení herce s vrstevnicemi je v poslední fázi nutné předat objekt vtkActor do původního mapperu vizualizované sítě.

Následující ukázka zdrojového kódu vytváří 5 vrstevnic nad datovým souborem grid. Rozsah hodnot skalárů, tedy minimální a maximální hodnota skalárního pole v proměnné grid, je získán metodou GetScalarRange().

```
/** Filtr pro zobrazení vrstevnic **/
//filtr vrstevnic
vtkContourFilter contourFiltr = vtkContourFilter.New();
//filtr pro vytvoření křivek nebo rovin z trojúhelníkové sítě
vtkStripper stripper = vtkStripper.New();
//mapper vrstevnic
vtkPolyDataMapper contourMapper = vtkPolyDataMapper.New();
//herec reprezentující vrstevnice ve vykreslovacím okně
vtkActor contours = vtkActor.New();

//získání minimální a maximální hodnoty skalárního pole
double[] range = grid.GetScalarRange();

//předání datového souboru sítě do filtru vrstevnic
contourFiltr.SetInput(grid);
//výpočet pěti vrstevnic z dat v uvedeném rozsahu hodnot
contourFiltr.GenerateValues(5, range[0], range[1]);
//provedení výpočtu
contourFiltr.Update();

//předání vypočtených dat vrstevnic sítě pro vytvoření křivek nebo rovin
stripper.SetInputConnection(contourFiltr.GetOutputPort());
//vytvoření křivek nebo rovin
stripper.Update();

//předání vrstevnic mapperu
contourMapper.SetInputConnection(stripper.GetOutputPort());
//barvu vrstevnic neurčovat ze skalárů bodů
contourMapper.ScalarVisibilityOff();
contourMapper.InterpolateScalarsBeforeMappingOff();
//nastavecí barvy vrstevnic skrze herce
contours.GetProperty().SetColor(1, 1, 1);
//předání zmapovaných dat vrstevnic
contours.SetMapper(contourMapper);
```

V ukázce není uvedeno předání herce kontur do mapperu vizualizované sítě, tento krok bude doplněn v ukázce zdrojového kódu renderování sítě.

2.2.4 Mapování

Mapování objektů (instancí filtrů, případně přímo datových souborů) provádíme pomocí tzv. mapperů, které dělíme na dva typy. Prvním jsou **grafické mappery**, které slouží k předání dat do renderovacího subsystému, druhým jsou **zapisovací mappery**,

které uloží data do binárního či textového souboru. Mappery mohou mít jeden či více vstupů, nemají však žádný vizualizační výstup (Schroeder et al., 2002, s. 184).

Vstupní data jsou pomocí mapperů pouze převedena na grafické elementární objekty, kterým je podle atributů dat a konfigurace mapperu určena barva, materiálové charakteristiky apod. Grafické objekty jsou pak v případě použití zapisovacího mapperu převedeny do binárního souboru, tedy vizualizačního výstupu, který v tomto případě proces vizualizace uzavírá. Použitím grafického mapperu jsou elementární grafická data připravena pro vytvoření herců, proces vizualizace tedy v tomto případě pokračuje.

Mapper zpracuje pomocí barevné škály hodnoty skalárních atributů a vygeneruje barvy, které přidělí jednotlivým bodům. Nejsou-li skalární hodnoty prostřednictvím atributů přiřazeny, je místo mapperu a jeho barevné škály k určení barvy použit objekt vlastnosti vtkProperty objektu vtkActor (Schroeder et al., 2002, s. 185-186).

Z hlediska zdrojového kódu naší aplikace budeme pracovat pouze s grafickými mappery. V uvedeném zdrojovém kódu vytvoříme instanci objektu vtkDataSetMapper, kterou použijeme ke zmapování datového souboru upraveného filtrem smrštění dat. Mapperu tak v metodě SetInputConnection() předáme zdroj dat pomocí objektu vtkShrinkFilter. Pokud bychom však chtěli mapovat datový soubor bez úpravy filtrem, předali bychom v této metodě mapperu zdroj dat objektem vtkUnstructuredGrid. Mapper je ve zdrojovém kódu dále konfigurován svými metodami tak, aby generování barev probíhalo z hodnot přiřazeného skalárního pole.

```
/** Mapování dat **/
vtkDataSetMapper mapper = vtkDataSetMapper.New();
//předání datové struktury do mapperu
mapper.SetInputConnection(shrinkFilter.GetOutputPort());
//nastavení viditelnosti skalárů
mapper.ScalarVisibilityOn();
//skalární hodnoty získávat z atributů bodů
mapper.SetScalarModeToUsePointData();
//barvu herce generovat ze získaných skalárních hodnot
mapper.SetColorModeToMapScalars();
//barevnou škálu generovat v celém rozsahu hodnot skalárního pole
mapper.InterpolateScalarsBeforeMappingOn();
```

2.2.5 Vytvoření herce

Herec v podobě třídy vtkActor reprezentuje objekt, který je při vizualizaci zobrazen. Jedná se tedy o souhrn renderovacích vlastností, jako jsou vlastnosti povrchu (okolní, rozptýlené a odražené světlo), způsob reprezentace objektu (povrchový nebo síťový mód), textur a geometrické definice (mapper) (Kol. aut., 2001, s. 58).

Vytvoření herce spočívá v založení instance třídy vtkActor zavoláním jeho metody New() a následně předáním mapperu, v našem případě objektu vtkDataSetMapper, metodou SetMapper(). Aby byla nějaká data viditelná, musí být mapper naplněn neprázdným datovým souborem. Popsané vytvoření herce demonstruje následující ukázka zdrojového kódu:

```
/** Vytvoření herce **/
vtkActor actor = vtkActor.New();
//předání zmapovaných dat v instanci mapperu
actor.SetMapper(mapper);
```

Vzhledem k tomu, že v našich příkladech se o barvu vykreslovaného objektu stará mapper, není potřeba herci z tohoto hlediska nastavovat žádné vlastnosti.

2.2.6 Renderování

Obecně si lze renderovací proces představit jako převod grafických dat do obrázku a jeho následné zobrazení (Schroeder et al., 2002, s. 33). Součástí renderovacího procesu je také interakce uživatele s renderovanými daty, tedy možnost jejich posouvání, otáčení a přibližování. Při vizualizaci jsou vstupem do renderovacího procesu objekty definující světlo, polohu kamery a geometrii zobrazovaného tělesa, které jsou pak spojeny v jeden vizuální výstup.

Jedním z nejdůležitějších faktorů renderování je světlo, bez kterého by byl výsledný obraz černý a kamera by neměla co zobrazit. Jak popisuje Schroeder a kol. (2002, s. 34-36), často používaným a efektivním principem renderování z hlediska 3D počítačové grafiky je tzv. ray-casting. Principem této techniky je simulace interakce světla s objektem sledováním trajektorie paprsků světla. Na rozdíl od reality, kdy paprsek vychází ze světelného zdroje, technika ray-casting sleduje světelné paprsky ve směru od pozorovatele do okolí. Jsou tedy sledovány pouze ty paprsky, které dopadnou do oka pozorovatele, což výrazně snižuje počet sledovaných paprsků a šetří se tak paměťové prostředky. Poté, co paprsek narazí na objekt, je možné určit, zda a jak je místo střetu osvětleno (Schroeder et al., 2002, s. 34-35).

Technika ray-casting počítá s interakcí světelného paprsku a povrchu pozorovaného objektu. Díky tomu nám stačí popsat celý objekt pouze pomocí jeho vnější obálky, tedy povrchu, vnitřní strukturu objektu znát nepotřebujeme. Tento model nazýváme **povrchové renderování**. Mnoho reálných objektů je však průsvitných.

Pro jejich vizualizaci tak musíme poskytnout také popis jejich vnitřní struktury a zjistit, jak ovlivňuje paprsky světla. Jedná se o tzv. **objemové renderování** a oproti povrchovému renderování je tato technika náročnější, protože vyžaduje větší počet sledovaných paprsků (Schroeder et al., 2002, s. 36).

Podle Schroedera a kol. (2002, s. 40-41), při sledování paprsků světla protnou některé z nich vizualizovaný objekt (herce) a vytvoří barvu. Výsledná barva není ovlivněna pouze sledovaným paprskem, ale také paprsky okolního světla, které se odráží od blízkých objektů. Zobrazená barva je výsledkem jednoduché aproximace rozptylu veškerého okolního světla, jenž používá křivku intenzity světelného zdroje (případně zdrojů). Tuto aproximaci lze vyjádřit následující rovnicí:

$$R_C = L_C \cdot O_C$$

Kde R_C je výsledná křivka světelné intenzity, L_C je křivka intenzity okolního světla a O_C je barevná křivka tělesa (Schroeder et al., 2002, s. 41).

Z hlediska programu je renderovací proces složen z několika objektů. V následujícím textu zmíníme jednotlivé třídy podle pořadí, ve kterém je vhodné je v renderovacím procesu použít a nakonec popíšeme způsob, jakým renderovací proces spustíme. Součástí popisu tříd jsou ukázky zdrojových kódů, které demonstrují použití tříd a jejich vybraných metod v praxi.

vtkRenderer

Hlavním objektem renderovací části vizualizačního procesu je vtkRenderer, který celý renderovací proces řídí z hlediska ostatních objektů. Objekt také zajišťuje transformace mezi souřadnicemi reálného světa, souřadnicemi pohledu (souřadný systém počítačové grafiky) a souřadnicemi zobrazení (souřadnice pixelů zobrazovacího zařízení).[5]

V ukázce kódu nejprve objektu vtkRenderer předáme herce vizualizovaného tělesa a kontur jako vstupní parametr metody AddActor(). Následně upravíme pozadí renderovacího okna, aby tvořilo barevný přechod metodou GradientBackgroundOn(). Černou, počáteční barvu přechodu zvolíme pomocí metody SetBackground(), touto metodou bychom bez předchozího zapnutí přechodu definovali také jednolitou barvu pozadí. Bílou, konečnou barvu přechodu pak zvolíme pomocí metody SetBackground2(). Na závěr metodou ResetCamera() upravíme pozici kamery tak, aby těleso zabíralo celý prostor vykreslovacího okna.

[5] Zdroj: VTK: vtkRenderer Class Reference. *VTK Documentation* [online]. © 2012 [cit. 2012-05-20]. Dostupné z <http://www.vtk.org/doc/nightly/html/classvtkRenderer.html#details>

```
/** Renderer **/
//vytvoření rendereru
vtkRenderer renderer = vtkRenderer.New();
//předání herce do rendereru
renderer.AddActor(actor);
//přidání kontur do rendereru
renderer.AddActor(contours);
//nastavení přechodu od černé do bílé barvy na pozadí
renderer.GradientBackgroundOn();
renderer.SetBackground(1, 1, 1);
renderer.SetBackground2(0, 0, 0);
//nastavení kamery tak, aby objekt zabíral celý prostor
renderer.ResetCamera();
```

vtkRenderWindow

Objekt vtkRenderWindow reprezentuje vykreslovací okno, ve kterém jsou zobrazeny vizualizované objekty. Vykreslovací okno je okno v grafickém uživatelské rozhraní, do kterého jsou renderery vykresleny obrázky.[6] Do jednoho vykreslovacího okna může být přiřazeno několik různě nastavených rendererů.

Ukázkový zdrojový kód uvádí vytvoření objektu vtkRenderWindow, nastavení velikosti okna - šířky a výšky v pixelech a definování jediného rendereru okna metodou AddRenderer(), do které předáme instanci objektu vtkRenderer vytvořenou v minulé ukázce.

```
/** Vykreslovací okno **/
vtkRenderWindow renderWindow = vtkRenderWindow.New();
//nastavení velikosti vykreslovacího okna (šířka, výška)
renderWindow.SetSize(1024, 768);
//předání rendereru do vykreslovacího okna
renderWindow.AddRenderer(renderer);
```

vtkRenderWindowInteractor

Způsob interakce uživatele s vizualizovanou sítí lze ovlivnit pomocí objektu vtkRenderWindowInteractor. Tento objekt poskytuje mechanismy interakce klávesnice,

[6] Zdroj: VTK: vtkRenderWindow Class Reference. *VTK Documentation* [online]. © 2012 [cit. 2012-05-20]. Dostupné z <http://www.vtk.org/doc/nightly/html/classvtkRenderWindow.html#details>

myši a časových událostí.[7] Podle publikace ActiViz .NET User's Guide (Kol. aut., 2008, s. 16-17) jsou ve výchozím stavu interakce myši a klávesnice následující:

- **levé tlačítko myši** otáčí kameru;
- **prostřední tlačítko myši** posouvá kameru v rovině;
- **pravé tlačítko myši** a **scrollování** přibližuje/oddaluje objekt;
- **klávesa *j*** zapne režim joystick, kdy při stisku tlačítka myši a posunu myši je zahájen plynulý pohyb;
- **klávesa *t*** zapne režim kruhového ovladače, kdy při stisku tlačítka je sledován pohyb myši;
- **klávesa *f*** stisknutá nad hercem vybere ohnisko kamery, kamera je pak posunuta k tomuto bodu blíže;
- **klávesa *w*** zobrazí herce v síťovém režimu;
- **klávesa *s*** zobrazí herce v režimu povrchů;
- **klávesa *r*** umístí kameru tak, aby byly viditelné všechny objekty herců.

Knihovna ActiViz .NET přichází s několika dalšími připravenými způsoby interakce, vybrání způsobu z hlediska objektu vtkRenderWindowInteractor probíhá předáním instance objektu některého z tzv. pozorovatelů v metodě SetInteractorStyle(). V ukázce zdrojového kódu nejprve vytvoříme pozorovatele objektem vtkInteractorStyleTrackballCameram následně objektu vtkRenderWindowInteractor předáme v metodě SetRenderWindow() instanci vykreslovacího okna, ke kterému se má definovaný způsob interakce vázat.

```
/** Interakce uživatele **/
//vytvoření pozorovatele - způsobu interakce s vizualizovanými daty
vtkInteractorStyleTrackballCamera interactorStyle =
    vtkInteractorStyleTrackballCamera.New();
//vytvoření objektu ovládajícího interakci
vtkRenderWindowInteractor interactor =
    vtkRenderWindowInteractor.New();
//nastavení objektu pomocí vytvořeného pozorovatele
interactor.SetInteractorStyle(interactorStyle);
//aplikace stylu interakce na vykreslovací okno
interactor.SetRenderWindow(renderWindow);
```

[7] Zdroj: VTK: vtkRenderWindowInteractor Class Reference. *VTK Documentation* [online]. © 2012 [cit. 2012-05-20]. Dostupné z <http://www.vtk.org/doc/nightly/html/classvtkRenderWindowInteractor.html#details>

Zahájení a obnovení renderování

Po přípravě všech objektů, které jsou součástí renderovacího procesu, můžeme přistoupit k zahájení renderování. Tímto krokem je především zavolání metody Render() objektu vtkRenderWindow. Metodu Render() můžeme volat také u objektu vtkRenderer. V pojetí vtkRenderWindow má však volání této metody obecnější charakter, neboť jím je zahájeno renderování všech rendererů přiřazených do vtkRenderWindow(). Vzhledem k tomu, že jsme v ukázkách použili individuální způsob interakce s daty, je nutné zahájit také tuto interakci zavoláním metody Start() objektu vtkRenderWindowInteractor().

```
/** Renderování **/
//zahájení vykreslování
renderWindow.Render();
//zahájení interakce s uživatelem
interactor.Start();
```

V případě změny vizualizace po zahájení renderování je nutné vizualizační okno přerenderovat a obnovit tak aktuální nastavení. Přerenderování je nutné provést především po následujících změnách:

- přidání/odstranění herce;
- přidání/úprava skalárního pole s požadavkem na jeho zobrazení ve formě barevné škály.

Z hlediska zdrojového kódu probíhá přerenderování vykreslovacího okna stejným způsobem, jako v případě zahájení renderování, tedy zavoláním metody Render() třídy vtkRenderWindow.

3 Control2D, Control3D a jejich implementace v ukázkovém programu Vizualizace

Cílem mého bakalářského projektu je vytvořit dva ovládací prvky, které implementují funkčnost obsáhlé knihovny ActiViz .NET se zaměřením na vizualizaci dat ve dvou a třírozměrné oblasti. Součástí projektu je dále formulářová aplikace Vizualizace, ve které je demonstrováno použití vytvořených ovládacích prvků. Formulářová aplikace také obsahuje třídu GMSHFileParser, pomocí které lze jednoduše zpracovat sítě exportované z freeware programu GMSH.

Vzhled komponent Control2D a Control3D je shodný. Největší část komponenty tvoří vizualizační okno, v němž je zobrazeno těleso. Nad tímto oknem se nachází menu s vybranými ovládacími prvky komponenty. Pod položkou Options lze přepínat povrchový a síťový mód a vyčistit okno, tedy odstranit z něj všechny zobrazené herce. Vedle Options se nachází volba shrink faktoru, tedy koeficientu smrštění buněk sítě.

Obr. 4: Návrh vzhledu komponenty v MS Visual Studion 2010

3.1 Motivace

Důvodem pro vývoj rozšiřující komponenty místo celistvého programu je především znovupoužitelnost takové komponenty v jiných aplikacích. Control2D a Control3D představují pomyslnou programátorskou vrstvu vyšší úrovně nad knihovnou ActiViz .NET. Vznikne-li tedy požadavek na rychlou, jednoduchou a ekonomickou implementaci vizualizační části do aplikace, nemusí vývojář podrobně studovat funkčnost celé knihovny ActiViz .NET, ale vystačí si se znalostí jednoduché API (*Application Programming Interface*) komponent.

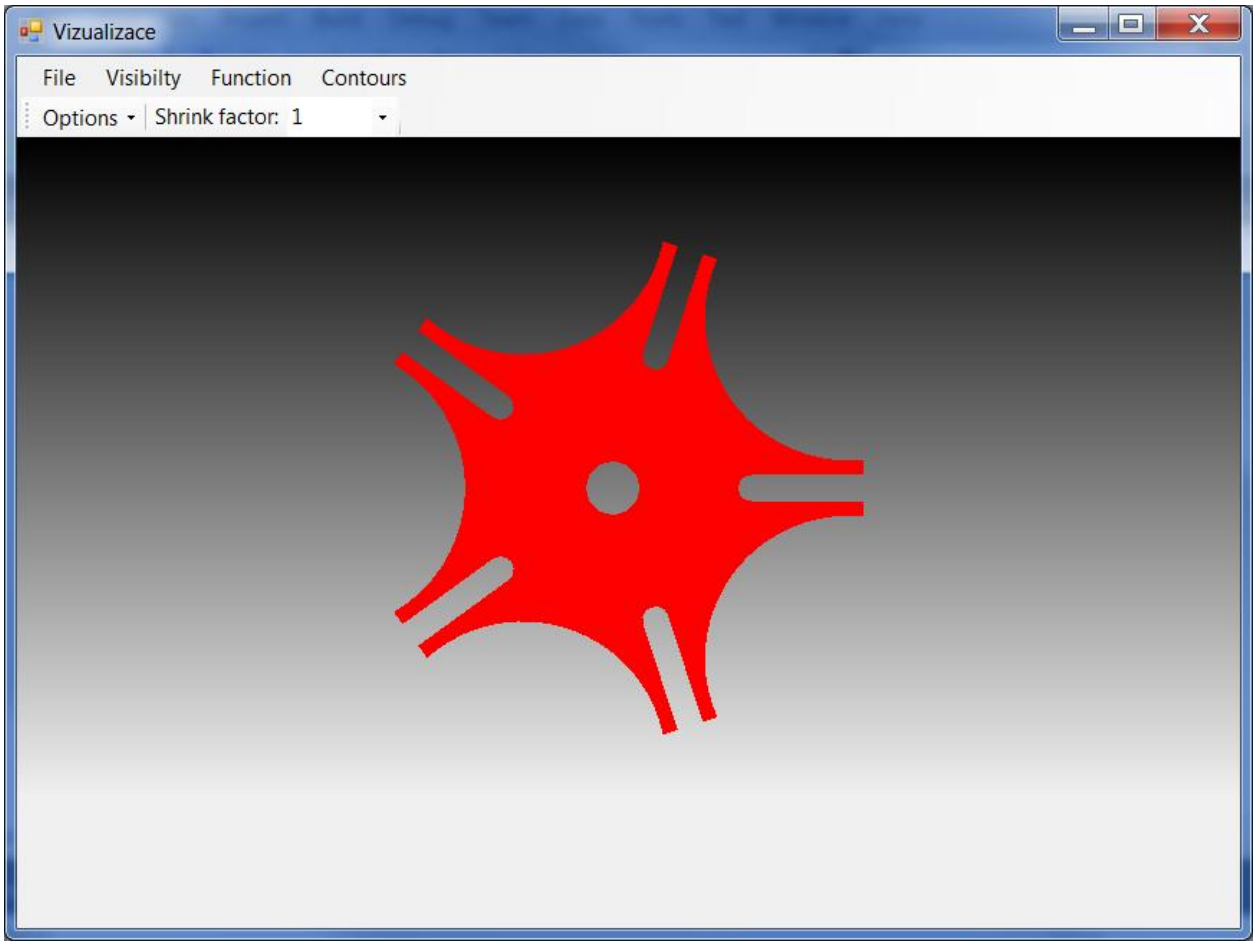
Vytvořené komponenty jsou především vhodné pro programy, které poskytují řešení parciálních diferenciálních rovnic. Jejich nejčastějším použitím tak zřejmě bude vizualizace výstupu metody konečných prvků.

3.2 Rozdíl mezi Control2D a Control3D

Jak již názvy jednotlivých ovládacích prvků napovídají, komponenta Control2D je určena pro zobrazení těles v rovině, zatímco komponenta Control3D vizualizuje objekt v prostoru. Tím však není řečeno, že prostřednictvím Control2D nelze zobrazit prostorové těleso a naopak pomocí Control3D rovinné těleso.

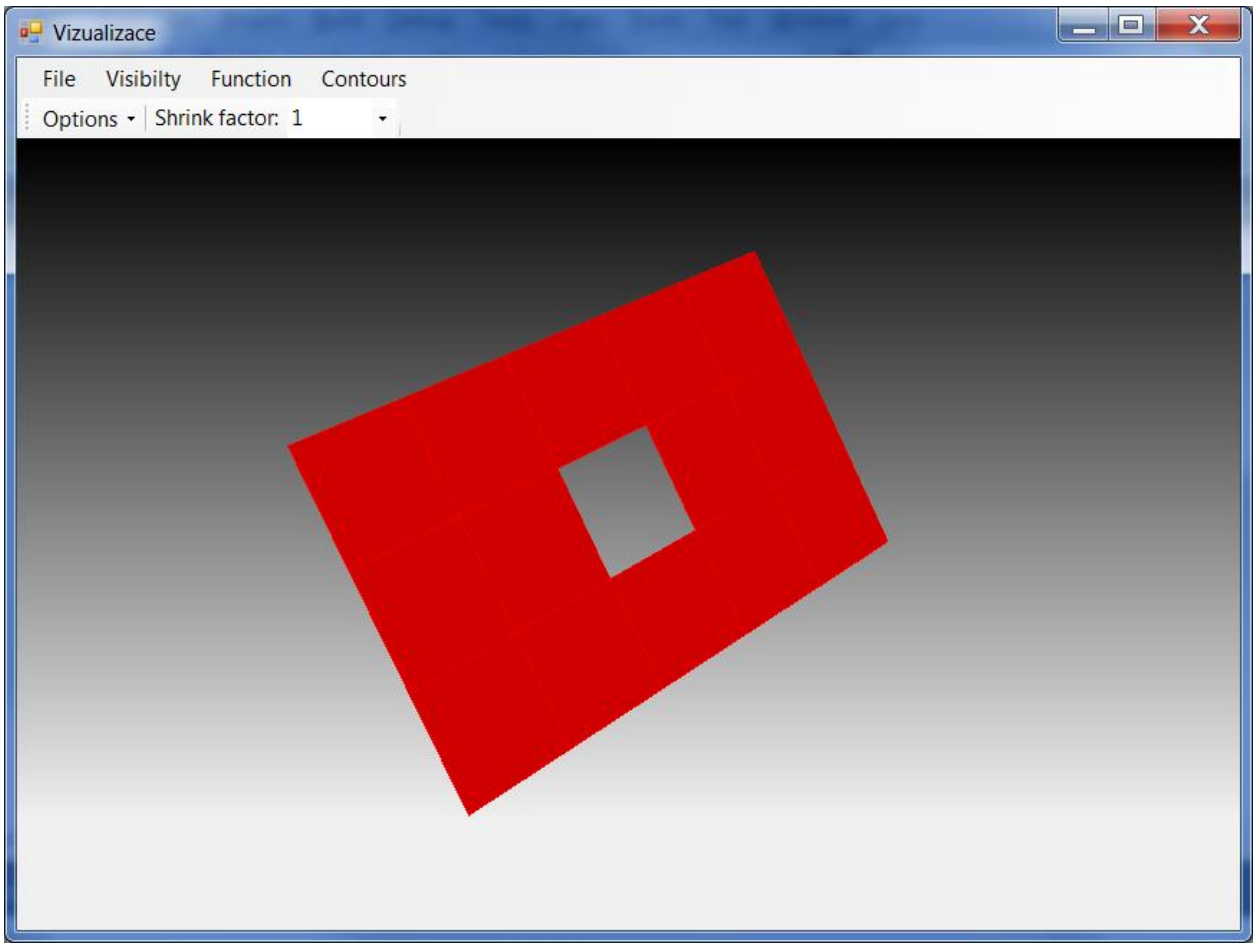
Při načtení sítě prostorového tělesa (tedy sítě, jejíž vrcholy buněk mají nenulové souřadnice *z*) do Control2D jsou v metodách, které plní data do datových objektů, souřadnice *z* ignorovány a jejich hodnota je nastavena na 0. Výstupem je tedy projekce tělesa původně zobrazeného v souřadnicích *x - y - z* do roviny *x - y*. Při tomto zobrazení je však nutné počítat s možným zkreslením sítě.

Jako příklad uveďme krychli, která je při pohledu na vrchní podstavu zobrazena při projekci do roviny jako čtverec. Každá úsečka hrany spodní podstavy krychle je ve zobrazeném čtverci překryta stejně dlouhou úsečkou hrany horní podstavy krychle. Čtverec je tedy fakticky zobrazen dvakrát, přestože naše oko jej vlivem shodného tvaru podstav vnímá pouze jednou. Stejným způsobem by byly překryty i jednotlivé buňky sítě v podstavách a buňky v objemu krychle.



Obr. 5: Zobrazení prostorového tělesa v Control2D

Při zobrazení rovinného tělesa prostřednictvím Control3D nedochází k žádným nežádoucím jevům. Síť je zobrazena nezkresleně, uživatel však má možnost pracovat s objektem jako s prostorovým.



Obr. 6: Zobrazení rovinného tělesa v Control3D

Jak bylo ukázáno, komponenty Control2D a Control3D neomezují počet dimenzí zobrazovaného tělesa, s objektem však zacházejí vždy jako s rovinným (Control2D), resp. prostorovým (Control3D). Nejdůležitějším rozdílem tak jsou různé možnosti interakce s vizualizovanými daty. Zatímco při použití Control2D má uživatel možnost objekt pouze posouvat, přibližovat a oddalovat, při vizualizaci v Control3D je možné objekt také prostorově otáčet.

3.3 Problémy při vývoji

3.3.1 Reference na knihovnu ActiViz .NET

Vývoj ovládacích prvků podléhal několika požadavkům. Předně bylo nutné myslet na možnost nenáročné implementace komponent do programu a jednoduchost celé API. V souladu s tímto požadavkem bylo potřeba zajistit nezávislost projektu na knihovně ActiViz .NET, jejíž referencování není zcela triviální. Reference na knihovnu ActiViz .NET jsou proto poskytnuty pouze v komponentách a při jejich implementaci do aplikace tak stačí uvést pouze referenční zdroj komponent.

V první fázi vývoje jsem tento požadavek podcenil a v demonstračním programu Vizualizace jsem komponentám předával již vytvořené datové soubory VTK, čímž vznikala nechtěná závislost. Navíc byla celá knihovna ActiViz .NET kopírována duplicitně jak do složek komponent, tak do složky samotného programu a celková velikost zkompilované aplikace tak překračovala přípustnou mez.

Tento problém měl naštěstí jednoduché řešení ve formě přesunu algoritmu plnění datových souborů VTK do metod komponent. V aplikaci tedy stačí vytvořit pole bodů a jednotlivých buněk sítě (např. pomocí třídy GMSHFileParser) a ty pak postupně předat skrz inicializační metody komponentám.

3.3.2 Dvourozměrná vizualizace

Knihovna ActiViz .NET je primárně určena pro zobrazování a práci s daty ve třech dimenzích, paradoxně je tak náročnější zobrazit těleso ve 2D (tedy bez možnosti prostorového otáčení). Rovinné zobrazení je v Control2D řešeno použitím objektu vtkInteractorStyleImage, jímž jsou před renderovacím procesem omezeny možnosti interakce uživatele s vizualizovanými daty.

3.3.3 Volba datového souboru

Pro vizualizaci prostorových těles, u kterých je mimo 2D síť obvodového pláště přítomna také 3D objemová síť vnitřního prostoru, jsem v poslední fázi vývoje přidal komponentě Control3D podporu pro buňku čtyřstěn, ze které se objemové sítě ve většině případů skládají. Po renderování v síťovém módu byly čtyřstěny vykresleny správně, nicméně po zapnutí povrchového módu bylo u čtyřstěnů viditelné chybné vykreslení některých stěn. Stěny, které měly být viditelné nebyly vykresleny vůbec, naopak stěny dále od kamery, které být viditelné neměly, renderovací systém vykreslil.

Vyzkoušel jsem několik konfigurací objektu herce, především nastavení vlastnosti BackfaceCulling (vykreslování zadních stěn) a FrontfaceCulling (vykreslování předních stěn), ale žádná z nich problém nevyřešila. Jak jsem nakonec zjistil, celý problém byl způsoben chybně zvoleným datovým souborem. Původně byl totiž u komponenty Control3D použit datový soubor vtkPolyData, který chybně interpretoval orientaci čtyřstěnů. Použitím souboru vtkUnstructuredGrid společně s mapperem vtkDataSetMapper byl problém vyřešen.

3.4 Manuál Control2D a Control3D

V následujícím manuálu jsou popsány postupy při práci s komponentami ve vlastní formulářové aplikaci. Veřejné metody obou komponent Control2D a Control3D jsou pro práci se zobrazeným objektem pojmenovány stejnými názvy. Přesto, že je následující popis omezen na práci s obecnějším Control3D, lze jej aplikovat také na Control2D.

Manuál rovněž předpokládá, že vývoj aplikace probíhá v prostředí Microsoft Visual Studio 2010. V jiných verzích tohoto software se mohou některé postupy, především způsob implementace ovládacích prvků, lišit.

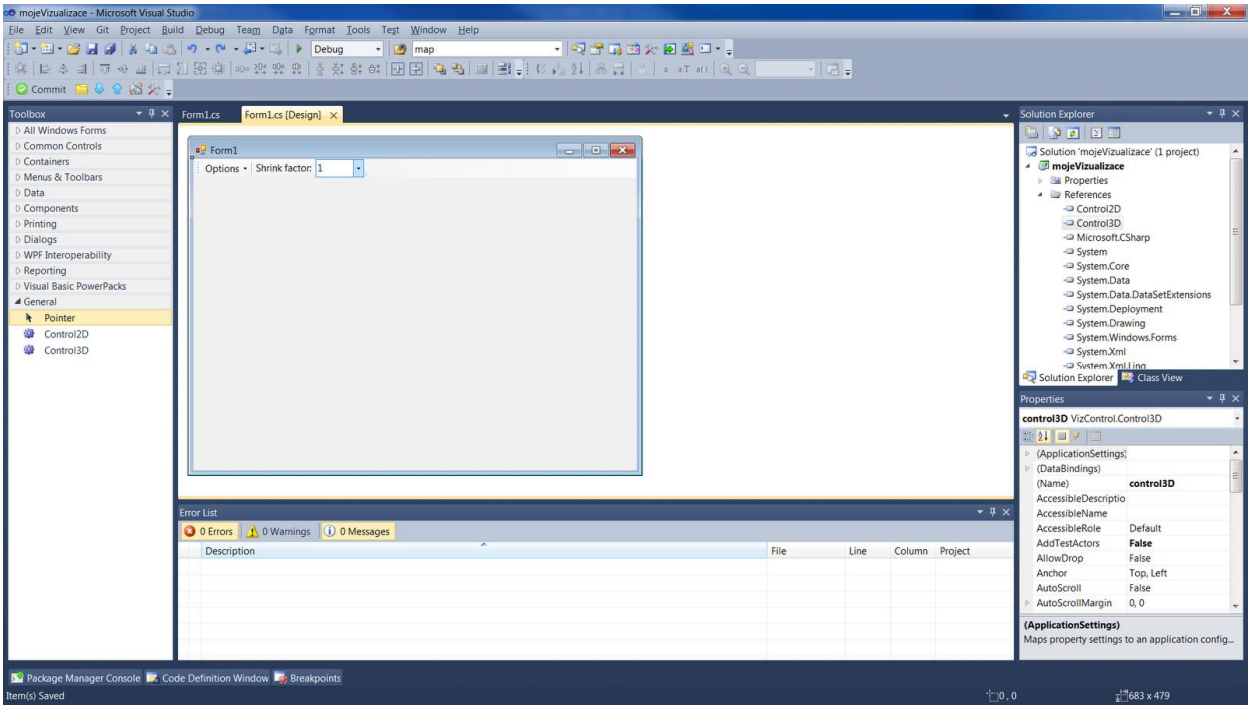
Součástí manuálu jsou také ukázky zdrojového kódu. Popis celého vizualizačního procesu je uspořádán tak, že poskládáním kusů kódu v pořadí, v jakém jsou v textu uvedeny, vznikne funkční aplikace. Plnění a zobrazení dat, vizualizace skalárního pole a zobrazení kontur doporučuji umístit např. do metod spouštěných tlačítkem, tedy ne do konstruktoru formulářové aplikace. V uvedeném příkladu je každý funkční celek spouštěn tlačítkem nacházejícím se pod vizualizačním oknem controlu. Aplikaci, kterou v následujícím textu vytvoříme, je možné nalézt na přiloženém CD pod názvem mojeVizualizace.

3.4.1 Implementace komponenty do formulářové aplikace

Ovládací prvek budeme přidávat do existujícího projektu formulářové aplikace. Popsáno je přidání komponenty Control3D. Control2D je možné přidat stejným postupem, složky a soubory jsou však pojmenovány Control2D.

Prvním krokem implementace je přidání referencí, tedy určení adresy souboru Control3D.dll. Tento soubor se ve složce komponenty Control3D nachází na adrese bin/Release/. Před určením adresy však zkopírujeme celou složku Control3D do složky našeho projektu a adresu na soubor .dll budeme uvádět vždy z této lokální kopie. Ve Visual Studiu v Solution Exploreru (klávesová zkratka Ctrl + W, S) klikneme pravým tlačítkem myši na název projektu a v kontextovém menu vybereme položku Add Reference. V otevřeném dialogovém okně v záložce Browse otevřeme složku projektu a z ní otevřeme Control3D.dll.

Naše formulářová aplikace má nyní přístup k metodám komponenty, ovšem ovládací prvek se zatím nevyskytuje v Toolboxu a není tak možné jej přidat do formuláře. Pro zpřístupnění komponenty proto klikneme pravým tlačítkem myši na Toolbox (klávesová zkratka Ctrl + W, X) a následně vybereme položku Choose Items. Ve spodní části otevřeného dialogového okna klikneme na tlačítko Browse a vybereme soubor Control3D.dll. Pro přidání ovládacího prvku do formuláře přetáhneme v režimu Designer (klávesová zkratka Shift + F7) položku Control3D z Toolboxu na požadované místo ve formuláři. V rámci ukázkových zdrojových kódů je komponenta pojmenována jako control3D.



Obr. 7: Formulářová aplikace s přidanou komponentou Control3D v MS Visual Studio 2010

3.4.2 Podmínkové metody

Při práci s komponentou se často dostaneme do situace, kdy je algoritmus možné provést pouze při splnění určitých podmínek. K rozhodnutí podmínky přitom potřebujeme data, která jsou uložena v privátních vlastnostech komponenty a z prostředí formulářové aplikace k nim tak nemáme přístup.

V komponentách jsou pro tyto situace připraveny dvě veřejné podmínkové metody, které vracejí boolean hodnotu jakožto výsledek dotazované podmínky.

- Metoda **hasData()** ověřuje, zda je v komponentě naplněn hlavní datový soubor sítě. Ověření probíhá zjištěním počtu bodů a buněk v datovém souboru, výsledek metody je true pouze v případě, že ani jedna z těchto hodnot není rovna nule. Metodou je vhodné podmínit algoritmy, které pro svou funkci vyžadují data sítě (např. vizualizace skalárního pole).
- Druhou metodou je **hasValues()**, která zkoumá dostupnost skalárního pole. Touto metodou je vhodné podmínit algoritmy, jejichž funkce je závislá na hodnotách skalárního pole (např. zobrazení kontur).

3.4.3 Struktura dat sítě a jejich předání komponentě

Pro zobrazení sítě je nejprve nutné požadovanou síť definovat pomocí polí bodů a buněk. Vzhledem k objektově orientovanému přístupu jazyka C# jsou pole tvořena

instancemi třídy System.Array, konkrétně se jedná o tzv. zubatá pole (pole polí). Jednotlivé položky polí jsou tvořeny vektory typu double, resp. integer.

Pole bodů

Výchozím polem pro definici sítě je pole bodů. Musí obsahovat souřadnice všech vrcholů každé buňky použité v síti. Každý bod je identifikován svým indexem v poli, indexu se následně využívá při definici buněk. Počet prvků v každé z dimenzí pole se odvíjí od typu použité komponenty. Pole s n body má v případě Control2D $n \times 2$ prvků (2 sloupce, n řádků), v případě Control3D pak $n \times 3$. Souřadnice každého bodu v poli jsou zadány jako čísla typu double.

Definice jednoduchého pole bodů v 3D může vypadat např. takto:

```
double[][] points = new double[4][] {
    new double[] { 0, 0, 0 },
    new double[] { 1.0, 0, 0 },
    new double[] { 0, 1.0, 0 },
    new double[] { 0, 0, 1.0 }
};
```

Pole je z hlediska komponenty interpretováno jako databázová tabulka. Jeho číslování tedy na rozdíl od běžných konvencí začíná od indexu 1. Pod položkou s indexem 1 se tak v uvedeném příkladu nachází bod definující počátek souřadného systému, na pozici 2 je bod na ose *x*, na pozici 3 bod na ose *y* a na pozici 4 bod na ose *z*.

Pole je vhodné si v aplikaci uložit např. do privátní vlastnosti, pořadí bodů a jejich souřadnice jsou totiž stěžejní pro případnou pozdější vizualizaci skalárního pole.

Pole buněk

Množina elementárních geometrických útvarů sítě, tzv. buněk, je definována pomocí dvoudimenzionálních polí, jejichž struktura se liší podle typu buňky. Komponenty momentálně podporují následující typy buněk (v závorce je uveden počet bodů a celkový počet prvků pole o n elementech):

- úsečka (2 body, $n \times 2$ prvků);
- trojúhelník (3 body, $n \times 3$ prvků);
- čtyřstěn (4 body, $n \times 4$ prvků, pouze Control3D).

Každá položka pole buněk je tvořena příslušným počtem celočíselných indexů z pole bodů.

Ukázkové pole s definicí úseček tvořených body z předchozího příkladu může vypadat následovně:

```
int[][] lines = new int[3][] {
    new int[] { 1, 2 },
    new int[] { 1, 3 },
    new int[] { 1, 4 }
};
```

První z uvedených úseček je vymezena body s indexem 1 a 2, tedy počátkem souřadné soustavy a bodem na ose *x*. Druhá úsečka body na pozicích 1 a 3. Třetí úsečka pak body s indexem 1 a 4. Uvedené úsečky tak tvoří normálový osový kříž.

Podobně můžeme definovat pole dvou trojúhelníků ležících v rovině *x* - *y* a v rovině *y* - *z*:

```
int[][] triangles = new int[2][] {
    new int[] { 1, 2, 3 },
    new int[] { 1, 3, 4 }
};
```

Příklad pole, definujícího jeden čtyřstěn vytvořený ze všech čtyř bodů:

```
int[][] tetrahedrons = new int[1][] {
    new int[] { 1, 2, 3, 4 }
};
```

Po získání či naplnění dat předáme jednotlivá pole ovládacímu prvku. Každý typ podporovaných vstupních polí má v komponentě k dispozici intuitivně pojmenovanou metodu, do které se dané pole předá jako parametr. Následující ukázka demonstruje předání výše vytvořených polí points, lines, triangles a tetrahedrons do komponenty Control3D:

```
control3D.setPoints(points);
control3D.setLines(lines);
control3D.setTriangles(triangles);
control3D.setTetras(tetrahedrons);
```

3.4.4 Zobrazení dat

Máme-li vytvořena a naplněna data, můžeme přistoupit k jejich zobrazení, které probíhá ve 3 krocích:

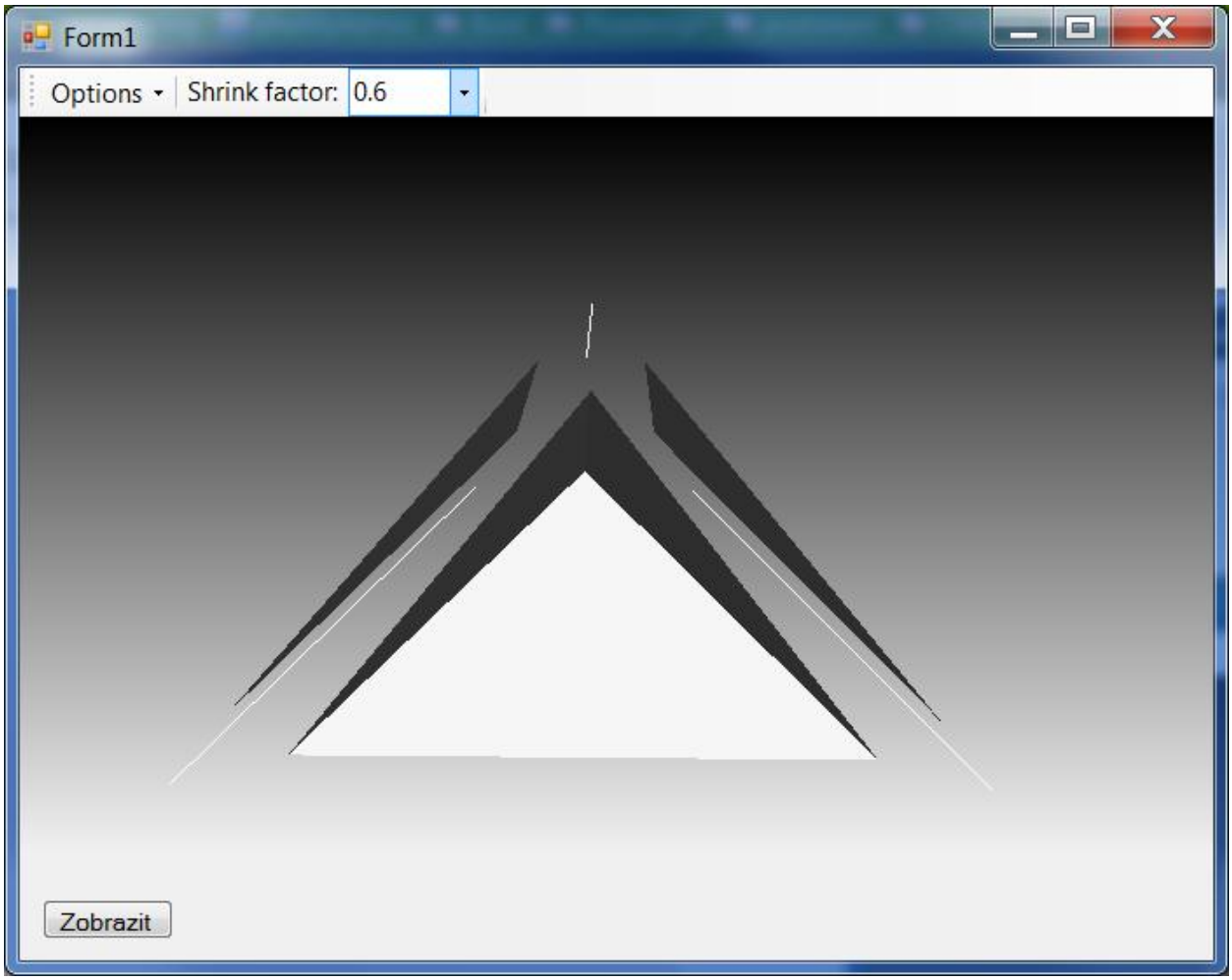
- zmapování dat,
- vytvoření herce a
- renderování.

V prvním kroku je nutné zmapovat naplněný datový soubor. Mapování provedeme zavoláním metody mapData(), která nepřijímá žádný vstupní parametr. Zmapovaná data jsou uložena jako vlastnost aktuální instance komponenty. Následně z dat vytvoříme herce zavoláním metody createDataActor(). Posledním krokem je zavolání metody render(), která způsobí zobrazení dat ve vizualizačním okně ovládacího prvku.

```
control3D.mapData();
control3D.createDataActor();
control3D.render();
```

V této fázi můžeme kompletní projekt sestavit (klávesová zkratka F6) a spustit (klávesová zkratka F5). Pokud byl dodržen popsaný postup, je ve vizualizačním okně ovládacího prvku nyní zobrazen pouze čtyřstěn. Trojúhelníky a úsečky viditelné nejsou, protože splývají se stěnami, resp. hranami čtyřstěnu. K tomu, abychom si ověřili, zda se trojúhelníky a úsečky zobrazily správně, nám poslouží zabudovaný shrink filtr, který nastavíme na hodnotu menší než 1. Zmenšíme tak velikost buněk bez změny polohy jejich těžiště. Mezi jednotlivými buňkami nám tím vzniknou mezery, díky kterým můžeme pozorovat všechny původně splývající útvary.

Můžeme také rovnou vyzkoušet rozdíl mezi povrchovým a síťovým módem, v menu Options stačí kliknout na položku Surface mode (její odškrtnutí zobrazí síť).



Obr. 8: Výstup testovací aplikace

3.4.5 Vizualizace skalárního pole

Důležitou funkčností komponent je možnost vizualizace skalárního pole, tedy proložení sítě funkcí a zobrazení výsledných hodnot pomocí barevné škály přímo na tělese. Z hlediska naší aplikace stačí předat komponentě skalární pole jako jednodimenzionální pole typu double[], ve kterém je zachováno pořadí hodnot z pole bodů. První hodnota skalárního pole tak přísluší k prvnímu bodu, druhá k druhému atd. Skalární pole se s polem bodů musí samozřejmě shodovat také v počtu prvků.

Vizualizace skalárního pole předpokládá předchozí předání dat sítě do komponenty. Před spuštěním algoritmu je proto vhodné tuto skutečnost ověřit podmínkovou metodou hasData().

Pro demonstraci si zvolíme jednoduchou funkci $f(x,y,z)=x+y+z$, počítající hodnoty skalárního pole jako součet hodnot souřadnic bodu. Hodnoty budeme počítat

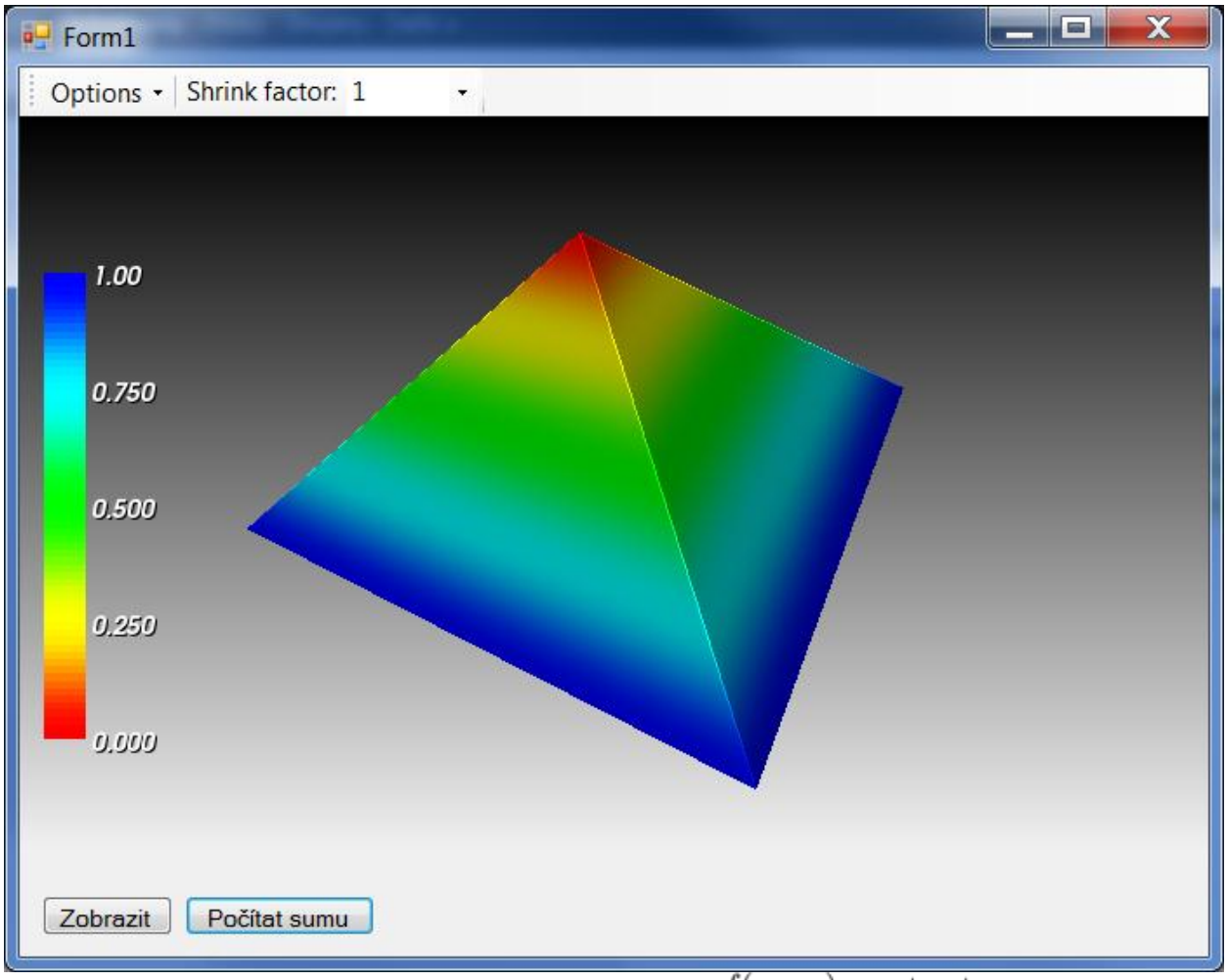
v cyklu for pro každý bod naší sítě a výsledky budeme ukládat do pole v proměnné results:

```
double[] results = new double[points.Length];
for (int i = 0; i < points.Length; i++)
{
    results[i] = points[i][0] + points[i][1] + points[i][2];
}
```

Výsledky výpočtů předáme komponentě jako parametr metody setPointValues(). Metoda přiřadí jednotlivé hodnoty skalárního pole příslušným bodům a podle maximální a minimální hodnoty v poli vygeneruje barevnou škálu, ze které pak každému bodu přiřadí konkrétní barvu. Chceme-li na levé straně vizualizačního okna ovládacího prvku zobrazit také indikátor rozsahu barevné škály, zavoláme metodu addRangeBar(). Vzhledem k tomu, že jsme nyní pozměnili vlastnosti zobrazené sítě, je nutné objekt přerenderovat pomocí metody refresh(). Následuje ukázka popsaného zdrojového kódu:

```
control3D.setPointValues(results);
control3D.addRangeBar();
control3D.refresh();
```

Po sestavení a spuštění aplikace vidíme, že naše těleso změnilo oproti původnímu stavu barvu. Vygenerovaná barevná škála zobrazuje body s nejnižší hodnotou červeně, bodům s nejvyšší hodnotou je přiřazena modrá barva.



Obr. 9: Vizualizace skalárního pole $f(x,y,z)=x+y+z$

3.4.6 Zobrazení vrstevnic skalárního pole

Zobrazení vrstevnic je z hlediska komponenty podmíněno existencí sítě a skalárního pole. Před zobrazením vrstevnic je tedy dobré zkontrolovat dostupnost těchto dat pomocí podmínkových metod hasData() a hasValues().

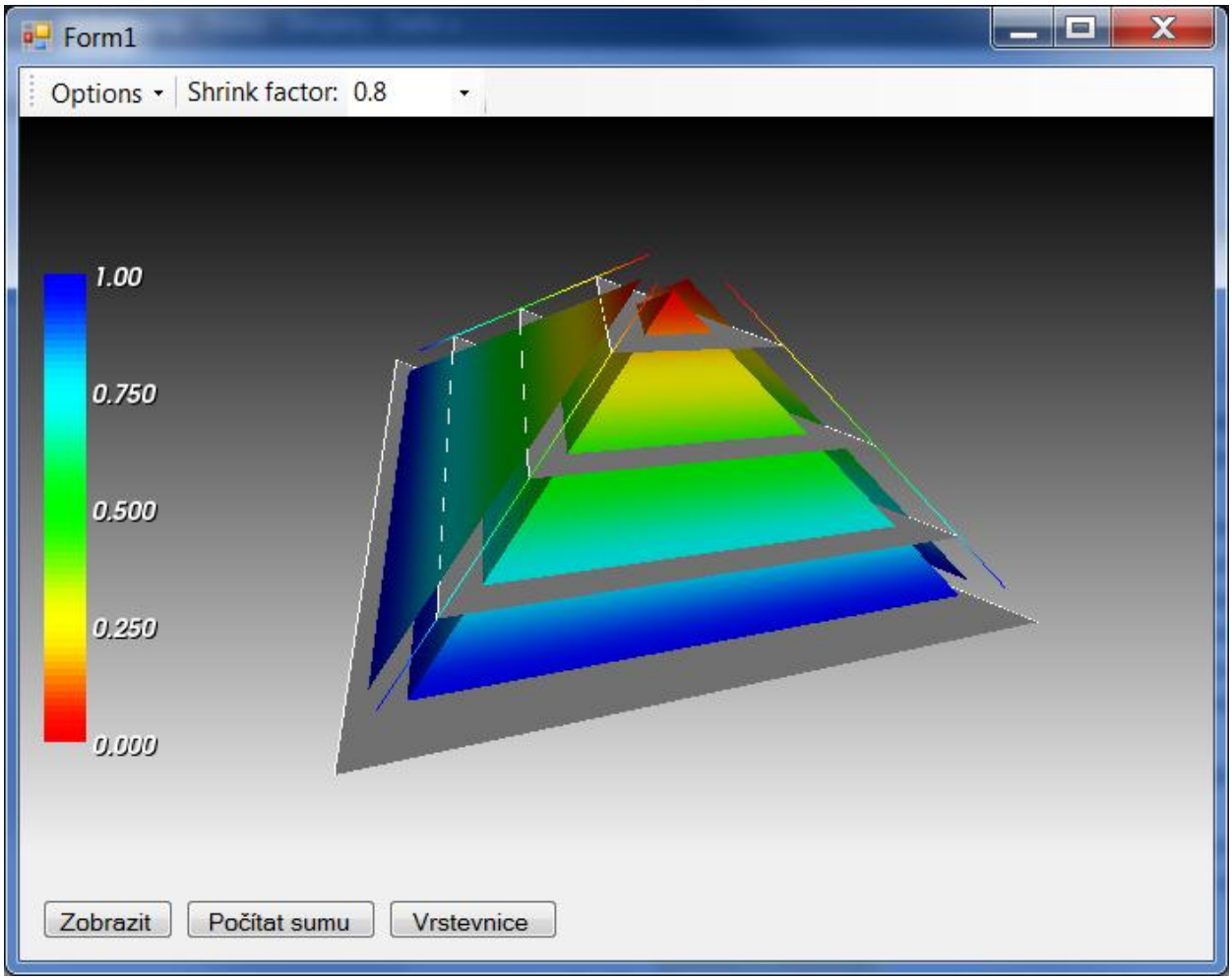
Pro práci s vrstevnicemi je v komponentách připravena přetížená metoda **contours()**, do které může vstupovat jeden, nebo čtyři parametry. První parametr je pro obě varianty stejný. Jedná se o celé číslo definující počet kontur, které budou zobrazeny. V případě druhé varianty následují další tři parametry, jimiž je definována červená, zelená a modrá složka barvy vrstevnic. Každá složka barvy je číslo typu double, oproti klasické definici RGB jsou složky normalizovány a mohou tak nabývat pouze hodnot z intervalu $<0,1>$. Při volání metody s jedním parametrem budou kontury vykresleny výchozí černou barvou.

Po provedení metody contours() je podobně jako v případě vizualizace skalárního pole nutné přerenderovat zobrazenou síť metodou refresh().

Pro demonstraci je uveden zdrojový kód, jehož výsledkem bude zobrazení pěti bílých vrstevnic skalárního pole, které jsme společně s tělesem vytvořili v předchozí části manuálu.

```
control3D.contours(5, 1, 1, 1);
control3D.refresh();
```

V případě prostorového tělesa jsou vrstevnice zobrazeny jako plochy, což je viditelné po smrštění buněk sítě shrink filtrem. Přestože požadovaný počet kontur určený prvním parametrem metody je 5, viditelné jsou pouze 4 vrstevnice. Tato nesourodost je způsobena algoritmem, který při výpočtu intervalu, ve kterém budou kontury zobrazeny, umístí vždy první vrstevnici na začátek barevné škály a poslední zcela na konec. V případě našeho tělesa je tak první kontura tvořena pouze bodem počátku souřadné soustavy, který není vykreslen.



Obr. 10: Zobrazení vrstevnic skalárního pole

3.4.7 Seznam veřejných vlastností

bool ToolBarVisibility

Nastavením na hodnotu false je skryto horní zabudované menu ovládacího prvku, nastavením na true je menu zobrazeno. Výchozí hodnota je true.

3.4.8 Seznam veřejných metod

bool hasData()

Ověření existence objektu, který má být zobrazen. Vrátí true, je-li v komponentě naplněn datový soubor, v opačném případě vrátí false.

bool hasValues()

Ověření existence skalárního pole. Vrátí true, jsou-li v komponentě definovány hodnoty bodů, v opačném případě vrátí false.

void surfaceOn()

Zobrazení vizualizovaného objektu v povrchovém módu.

void surfaceOff()

Zobrazení vizualizovaného objektu v síťovém módu.

void clearWindow()

Odstranění zobrazeného objektu z vizualizačního okna.

void setPoints(double[][] points)

Naplní předané multidimenzionální pole se souřadnicemi bodů do datového souboru. Parametr *points* je pole polí dvou (Control2D), resp. tří (Control3D) souřadnic typu double. V definici buněk je pole indexováno od čísla 1.

void setLines(int[][] lines)

Naplní předané multidimenzionální pole s definicemi úseček do datového souboru. Předpokládá, že datovému souboru byly předány body prostřednictvím metody setPoints(). Parametr *lines* je pole polí dvou celočíselných indexů z pole bodů.

void setTriangles(int[][] triangles)

Naplní předané multidimenzionální pole s definicemi trojúhelníků do datového souboru. Předpokládá, že datovému souboru byly předány body prostřednictvím metody setPoints(). Parametr *triangles* je pole polí tří celočíselných indexů z pole bodů.

void setTetras(int[][] tetras)

Naplní předané multidimenzionální pole s definicemi čtyřstěnů do datového souboru. Předpokládá, že datovému souboru byly předány body prostřednictvím

metody setPoints(). Parametr *tetras* je pole polí čtyř celočíselných indexů bodů z pole bodů.

void mapData()

Zmapuje datový soubor podle aktuálního nastavení na síťový, nebo povrchový model a připraví vstup pro vytvoření herce.

void createDataActor()

Vytvoří herce, tedy síť, jež bude objektem vizualizace. Předpokládá naplněný datový soubor sítě zmapovaný pomocí metody mapData().

void render()

Inicializuje zobrazení herce ve vizualizačním okně a zahájí renderování. Předpokládá naplněný datový soubor sítě zmapovaný pomocí metody mapData() a existenci herce vytvořeného metodou createDataActor().

void refresh()

Obnoví renderování herce po provedení změn.

void setPointValues(double[] values)

Uloží hodnoty předaného skalárního pole jako atributy k jednotlivým bodům v datovém souboru, vygeneruje barevnou škálu a přiřadí bodům barvu. Předpokládá naplněný datový soubor. Parametr *values* je pole hodnot skalárního pole typu double, počet a pořadí položek v poli musí být shodný s počtem a pořadím položek v poli bodů datového souboru.

void disposePointValues()

Odstraní hodnoty předem definovaného skalárního pole a přiřadí bodům výchozí bílou barvu. Předpokládá naplněný datový soubor.

void addRangeBar()

Přidá indikátor rozsahu skalárního pole. Předpokládá naplněný datový soubor a naplněné skalární pole pomocí metody setPointValues().

void removeRangeBar()

Odstraní indikátor rozsahu skalárního pole.

void contours(int count)

Vykreslí předaný počet černých kontur. Předpokládá naplněný datový soubor a naplněné skalární pole pomocí metody setPointValues(). Celočíselný parametr *count* určuje počet kontur, který bude zobrazen.

void contours(int count, double r, double g, double b)

Vykreslí předaný počet kontur. Předpokládá naplněný datový soubor a naplněné skalární pole. Celočíselný parametr *count* určuje počet kontur, který bude zobrazen, parametry *r* (červená), *g* (zelená) a *b* (modrá) typu double jsou normalizované složky barevného modelu RGB.

void removeContours()

Odstraní vykreslené kontury z tělesa. Předpokládá naplněný datový soubor.

void saveAsPNG()

Zobrazí saveFileDialog a uloží do vybraného .png souboru aktuální stav vizualizačního okna včetně všech zobrazených objektů a vlastností sítě. Předpokládá naplněný datový soubor.

3.5 Ukázkový program Vizualizace

Ukázková aplikace Vizualizace obsahuje implementaci obou komponent Control2D i Control3D. Typ komponenty, který je pro vizualizaci použit, rozhoduje uživatel při načítání exportního GMSH souboru. Obě komponenty jsou v aplikaci umístěny na stejném místě a překrývají se, neaktivní komponenta je pak pouze skryta pomocí vlastností Visible.

Vedle ovládacích prvků komponenty rozšiřuje aplikace možnosti interakce uživatele s vizualizovanými daty dalšími ovládacími prvky, které jsou umístěny především v horním menu aplikace. V této kapitole tedy rozebereme možnosti ovládacích prvků aplikace a dopad jejich použití na vizualizovanou síť.

3.5.1 Základní funkce horního menu aplikace

Pro zahájení vizualizace musí uživatel aplikaci předat exportní soubor GMSH, ze kterého jsou načteny data do datového souboru konkrétního ovládacího prvku. Načtení souboru je možné provést kliknutím na položku File v horním menu a následným zvolením mezi Load 3D GSMH file a Load 2D GMSH file. Touto volbou je rozhodnuto, zda bude použit Control3D či Control2D. Vybráním souboru v dialogovém okně je pak zahájeno renderování vybraných dat.

V podmenu položky File se dále nachází možnost Save as PNG, která spustí dialogové okno výběru .png souboru, do kterého je uložena aktuálně zobrazená síť včetně všech svých konfigurací a případně také indikátoru rozsahu barevné škály. Jedná se tedy o printscreen vizualizačního okna použité komponenty.

Při práci s aplikací může uživatel potřebovat využít celý prostor komponenty pouze pro vizualizovanou síť, přijde mu tedy vhod skrytí menu kontrolu. Tato možnost se nachází pod položkou Visibility, kde stačí odškrtnout položku Toolbars.

3.5.2 Připravená skalární pole

Aplikace Vizualizace s sebou přináší několik vytvořených skalárních polí. Aby bylo možné skalární pole vizualizovat, je nutné nejprve načíst data sítě. Následně může uživatel v horním menu aplikace vybrat jednu z nabízených funkcí dvou, tří, případně čtyř neznámých. Při rovinné vizualizaci pomocí Control2D je výběr funkcí vzhledem k absenci třetí souřadnice *z* omezen (viz příloha, obr. 13). Při vizualizaci skalárního pole je ve vizualizačním okně komponenty rovněž zobrazen indikátor barevné škály, ze kterého lze odečíst hodnoty jednotlivých barev.

V případě volby některé z funkcí proměnné *t*, tedy času, se ve spodní části aplikace zobrazí posuvník, skrze který lze hodnotu proměnné *t* ve funkci měnit. Skalární pole se při změně této proměnné dynamicky přepočítává, což se ve vizualizačním okně komponenty projevuje změnou zbarvení sítě a přepočtem hodnot v indikátoru rozsahu barevné škály (viz příloha, obr. 14).

V podmenu položky Functions se pod výpisem funkcí skalárních polí dále nachází položka Clear functions, která vypne vizualizaci skalárního pole a odstraní indikátor rozsahu barevné škály. Zobrazená síť je pak obarvena výchozí bílou barvou.

Funkce dvou neznámých

Tyto základní skalární pole je možné zobrazit jak v režimu rovinné (viz příloha, obr. 13), tak prostorové vizualizace. Skaláry jsou počítány ze souřadnic *x* a *y*, k dispozici jsou následující funkce:

1. $f(x,y) = x + y$
2. $f(x,y) = x - y$
3. $f(x,y) = y - x$
4. $f(x,y) = x^2 + y^2$

Funkce tří neznámých

Skalární pole, jejichž výsledek je počítán ze tří proměnných, jsou dvojího typu.

- Funkce používající proměnné ***x***, ***y*** a ***t*** je možné použít ve dvojrozměrném i třírozměrném režimu, výsledné skaláry lze dynamicky měnit zobrazeným posuvníkem hodnoty *t*.
- Funkce proměnných ***x***, ***y*** a ***z*** je možné zobrazit pouze při vizualizaci prostorové sítě.

Program obsahuje následující skalární pole tří proměnných:

1. $f(t,x,y) = x + y \cdot sin(y \cdot t)$
2. $f(t,x,y) = x + y \cdot cos(x \cdot t)$
3. $f(x,y,z) = x^2 + y^2 + z^2$

Funkce čtyř neznámých

Skalární pole počítané ze čtyř proměnných jsou k dispozici pouze v případě prostorové vizualizace. Jednotlivé skaláry jsou počítány z hodnot *x*, *y*, *z* a *t*, výsledné skalární pole lze dynamicky přepočítat změnou hodnoty *t*. V programu jsou připraveny následující skalární pole čtyř proměnných:

1. $f(t,x,y,z) = x + y \cdot sin(z \cdot t) + z \cdot cos(y \cdot t)$
2. $f(t,x,y,z) = x - cos(y \cdot z \cdot t) + z \cdot sin(x \cdot t)$

3.5.3 Vrstevnice

Byla-li zvolena funkce pro vizualizaci skalárního pole, je možné průběh jejich hodnot na tělese zvýraznit zobrazením vrstevnic. Pro ovládání vrstevnic slouží poslední položka Contours v horním menu aplikace. V podmenu této položky je možné kliknutím na Options v otevřeném dialogovém okně nastavit počet vrstevnic, který bude zobrazen, a dále určit, zda budou vrstevnice vykresleny v síťovém či povrchovém režimu (viz příloha, obr. 16 a 18).

Vrstevnice lze v aplikaci rychle zobrazit také bez předchozího nastavení kliknutím na položku Display v podmenu Contours. V případě, že před tímto rychlým zobrazením nebyly vrstevnice nastaveny, bude zobrazeno 10 vrstevnic v režimu sítě. Pokud již nastavení vrstevnic proběhlo, jsou po jejich rychlém zapnutí vykresleny podle tohoto nastavení. V povrchovém režimu jsou vrstevnice vykresleny černou barvou (viz příloha, obr. 15), v síťovém režimu pak kvůli viditelnosti barvou bílou (viz příloha, obr. 17).

Závěr

Účelem bakalářské práce bylo vytvoření vizualizačních programových komponent za pomoci knihovny ActiViz .NET. Stěžejním požadavkem přitom byla jednoduchost následné implementace a používání ovládacího prvku. Tuto činnost by měl zvládnout také začínající programátor, který se s vizualizací dat setkává poprvé.

V textu práce je nejprve uveden obecný popis prostředků, kterých bylo k vývoji komponent použito, následně je v kapitole o vizualizační knihovně ActiViz .NET tato knihovna popsána detailněji. V poslední kapitole se věnuji vytvořeným komponentám. Součástí 2. a 3. kapitoly jsou ukázky komentovaných zdrojových kódů, které demonstrují způsoby práce s knihovnou, resp. s komponentou na příkladu vizualizace několika jednoduchých buněk sítě. Výsledkem zdrojových kódů obou kapitol je shodný program, u jednoho je však použita pouze knihovna ActiViz .NET, u druhého pak vytvořený ovládací prvek. Zjednodušení procesu vizualizace je tak možné ověřit přímým porovnáním zdrojových kódů obou aplikací.

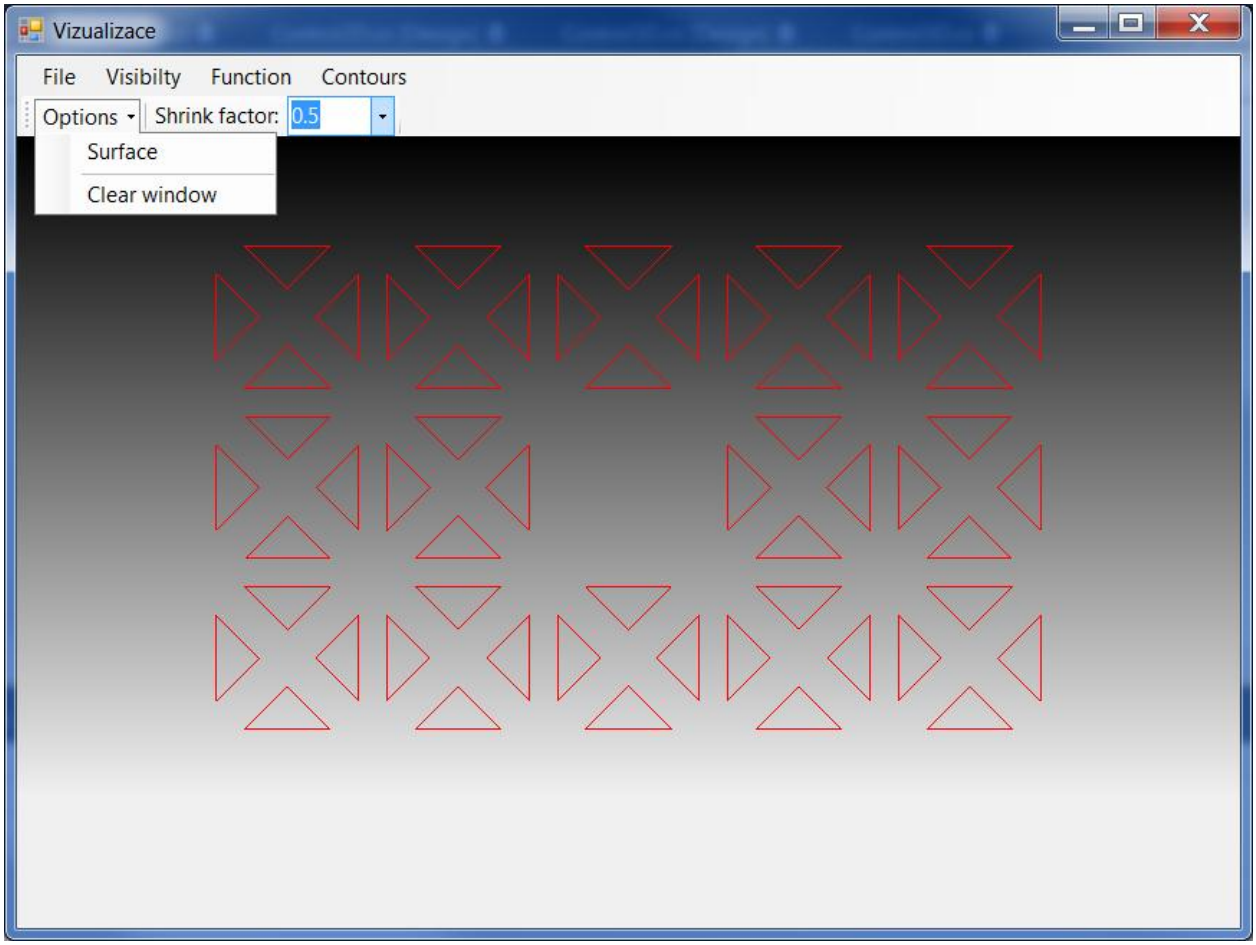
Knihovna ActiViz .NET obsahuje v oblasti vizualizace sítí široké možnosti, vytvořené komponenty je tak do budoucna možné rozšířit o další funkčnost. Bylo by např. vhodné umožnit zobrazení bodů pomocí filtru vtkGlyph a bod tak vykreslovat jako čtverec. Mezi další návrhy na rozšíření patří např. číselné popisy jednotlivých buněk a bodů, možnost zvětšení vizualizačního okna na celou obrazovku (tzv. fullscreen), nebo přesun možností ovládacího prvku do kontextového menu, zobrazovaného po kliknutí pravým tlačítkem myši.

Použité zdroje

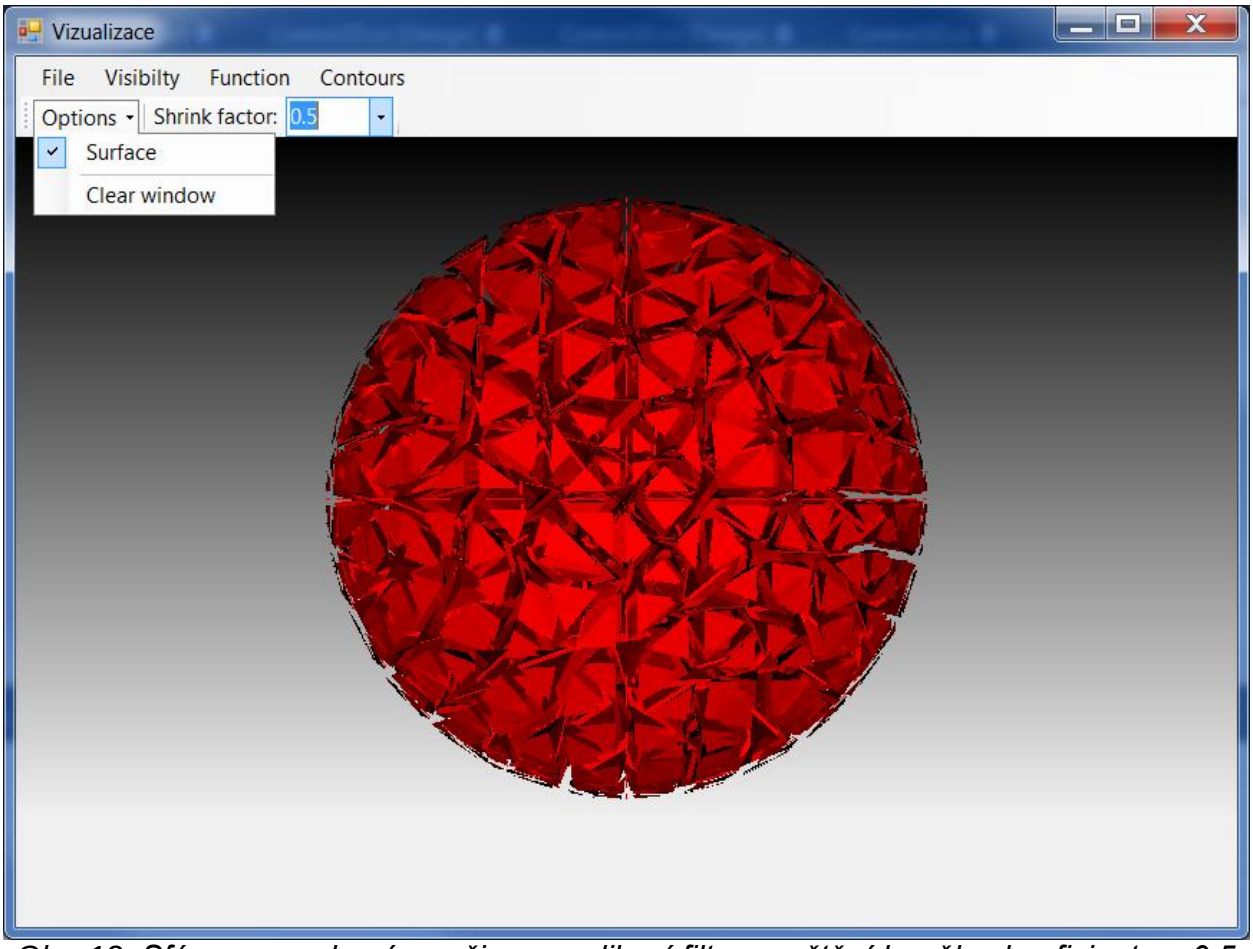
1. GEORGE, Paul-Louis; Pascal JEAN FREY. *Mesh Generation*: *Application to Finite Elements*. 2. vyd. Great Britain: ISTE Ltd., 2008.
2. Kolektiv autorů. *ActiViz .NET User's Guide: Version 5.2.* USA: Kitware, Inc., 2008.
3. Kolektiv autorů. *The VTK User's Guide: Updated for version 4.0*. USA: Kitware, Inc., 2001.
4. MÁLKOVÁ, Martina. *Morfování geometrických objektů v povrchové reprezentaci*. *Bakalářská práce.* Plzeň: Západočeská univerzita v Plzni, 2006.
5. SCHROEDER, Will; Ken MARTIN; Bill LORENSEN. *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics.* 3. vyd. USA: Kitware, Inc., 2002. ISBN 1-930934-07-6
6. SELLS, Chris; Michael WEINHARDT. *Windows Forms 2.0 Programming.* 2. vyd. USA: Addison-Wesley Professional. 2006. ISBN 978-0-321-26796-2
7. About. *VTK - The Visualization Toolkit.* USA: Kitware, Inc. [online]. © 2012 [cit. 2012-05-20]. Dostupné z <http://www.vtk.org/VTK/project/about.html>
8. BELL, John T. VTK Filters Documentation. *University of Illinois at Chicago: Department of Computer Science* [online]. © 2012 [cit. 2012-05-11]. Dostupné z: <http://www.cs.uic.edu/~jbell/CS526/Tutorial/Filters.html>
9. GEUZAINE, Christophe; Jean-François REMACLE. Solver: External Solver Interface. *GMSH 2.5* [online]. © 2010 [cit. 2012-05-20]. Dostupné z: <http://www.geuz.org/gmsh/doc/texinfo/gmsh.html#Solver>
10. Společný jazykový modul runtime (CLR). *MSDN – Možnosti vývoje softwaru pro klientské počítače, web, cloud a telefony*. MICROSOFT [online]. © 2012 [cit. 2012-05-20]. Dostupné z: <http://msdn.microsoft.com/cs-cz/library/8bs2ecf4.aspx>
11. VTK: vtkShrinkFilter Class Reference. *VTK 4.5.0 Documentation* [online]. © 2004-2012 [cit. 2012-05-20]. Dostupné z: <http://davis.lbl.gov/Manuals/VTK-4.5/classvtkShrinkFilter.html#_details>
12. VTK Documentation. *VTK - The Visualization Toolkit* [online]. © 1997-2012 [cit. 2012-05-19]. Dostupné z: <http://www.vtk.org/doc/>

Přílohy

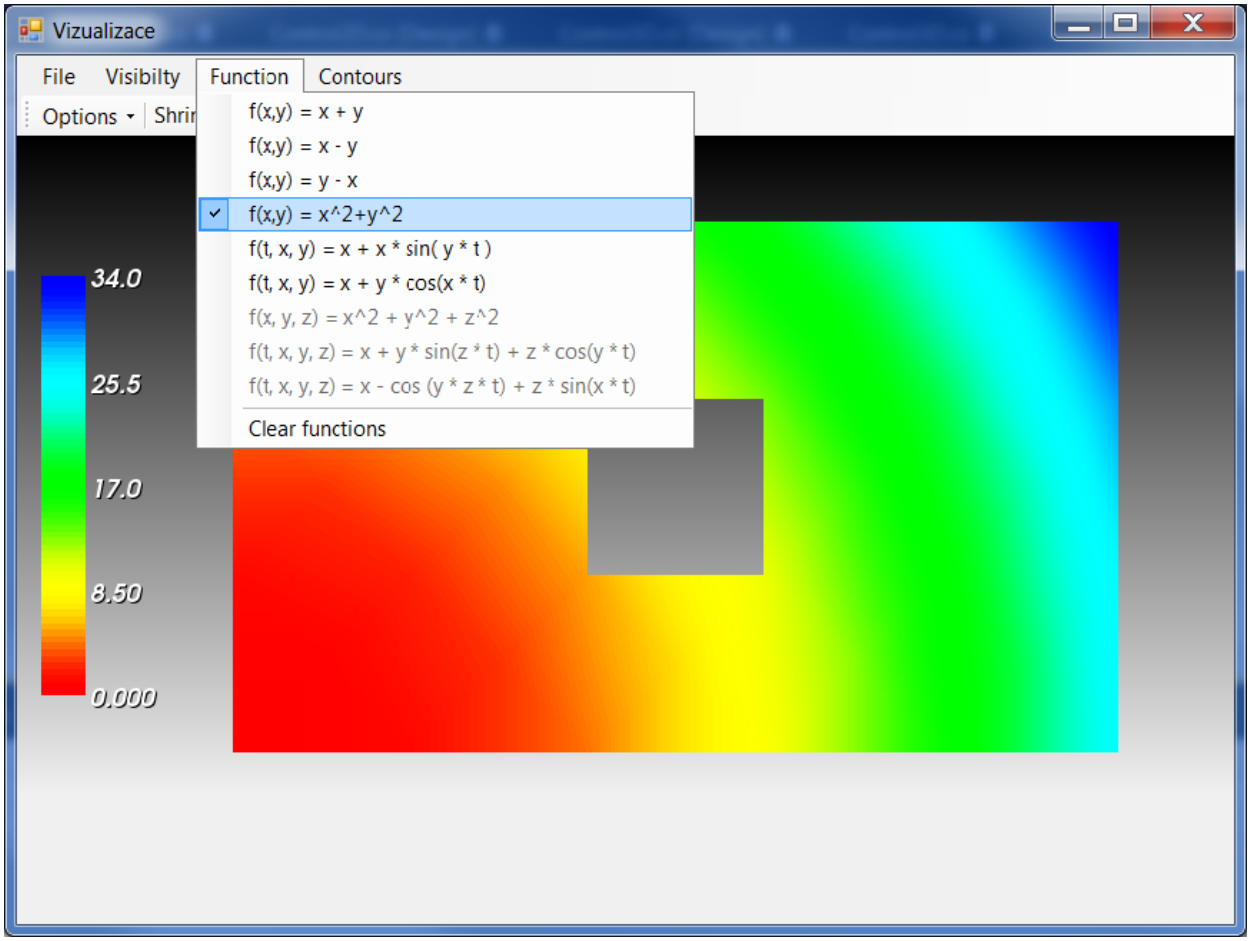
Seznam příloh



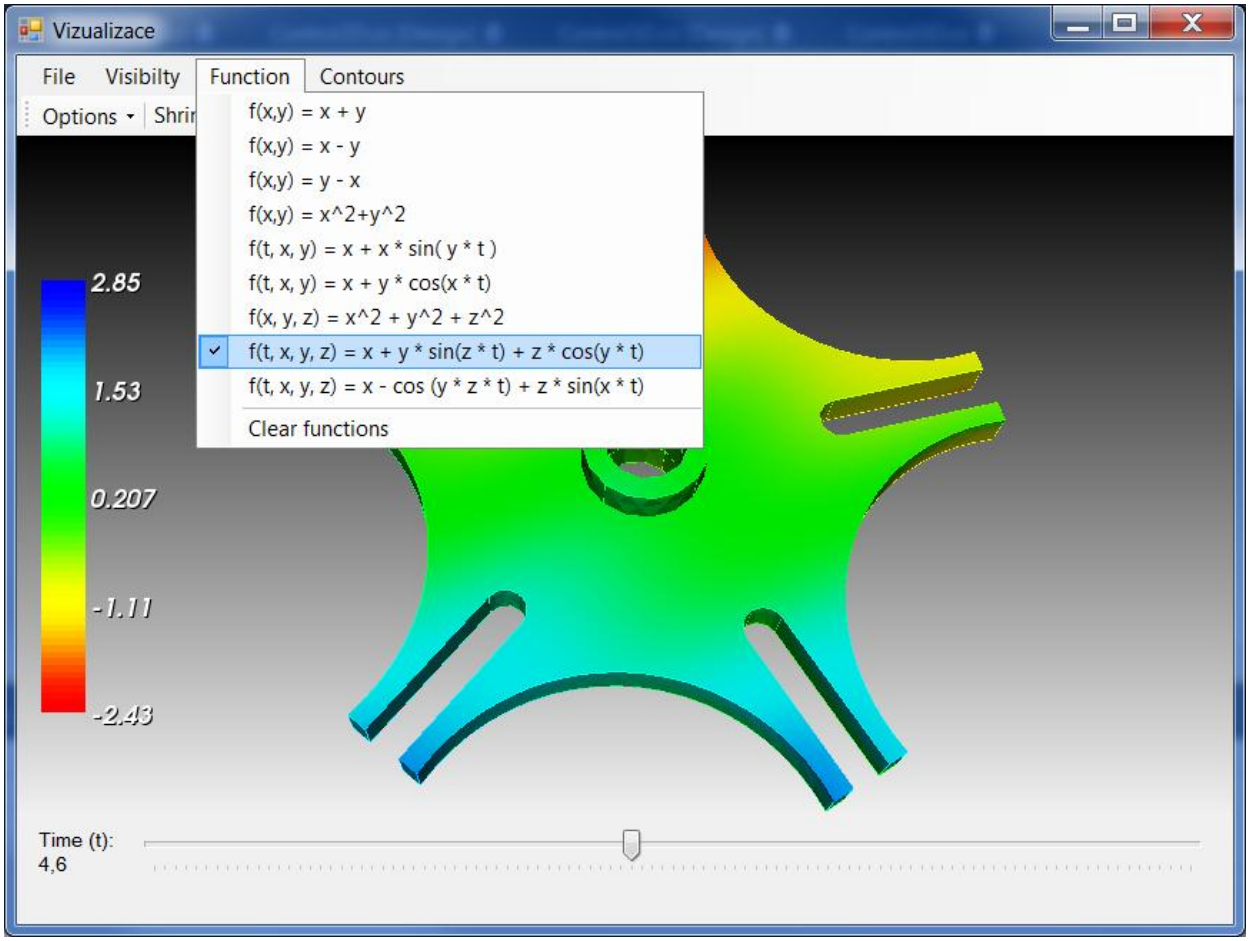
Obr. 11: Rovinné těleso v síťovém režimu s aplikací filtru smrštění buněk s koeficientem 0,5 v Control2D



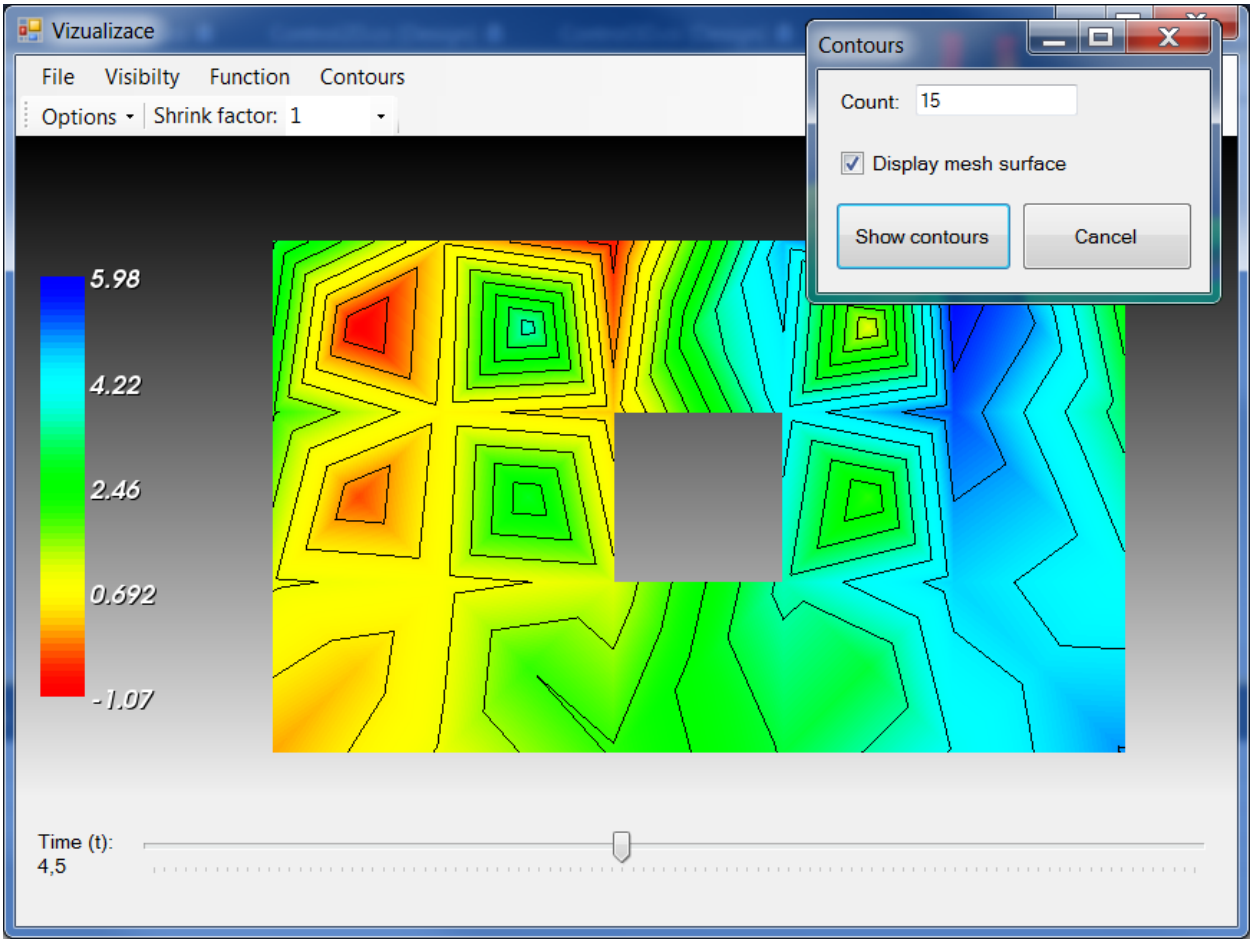
Obr. 12: Sféra v povrchovém režimu s aplikací filtru smrštění buněk s koeficientem 0,5 v Control3D



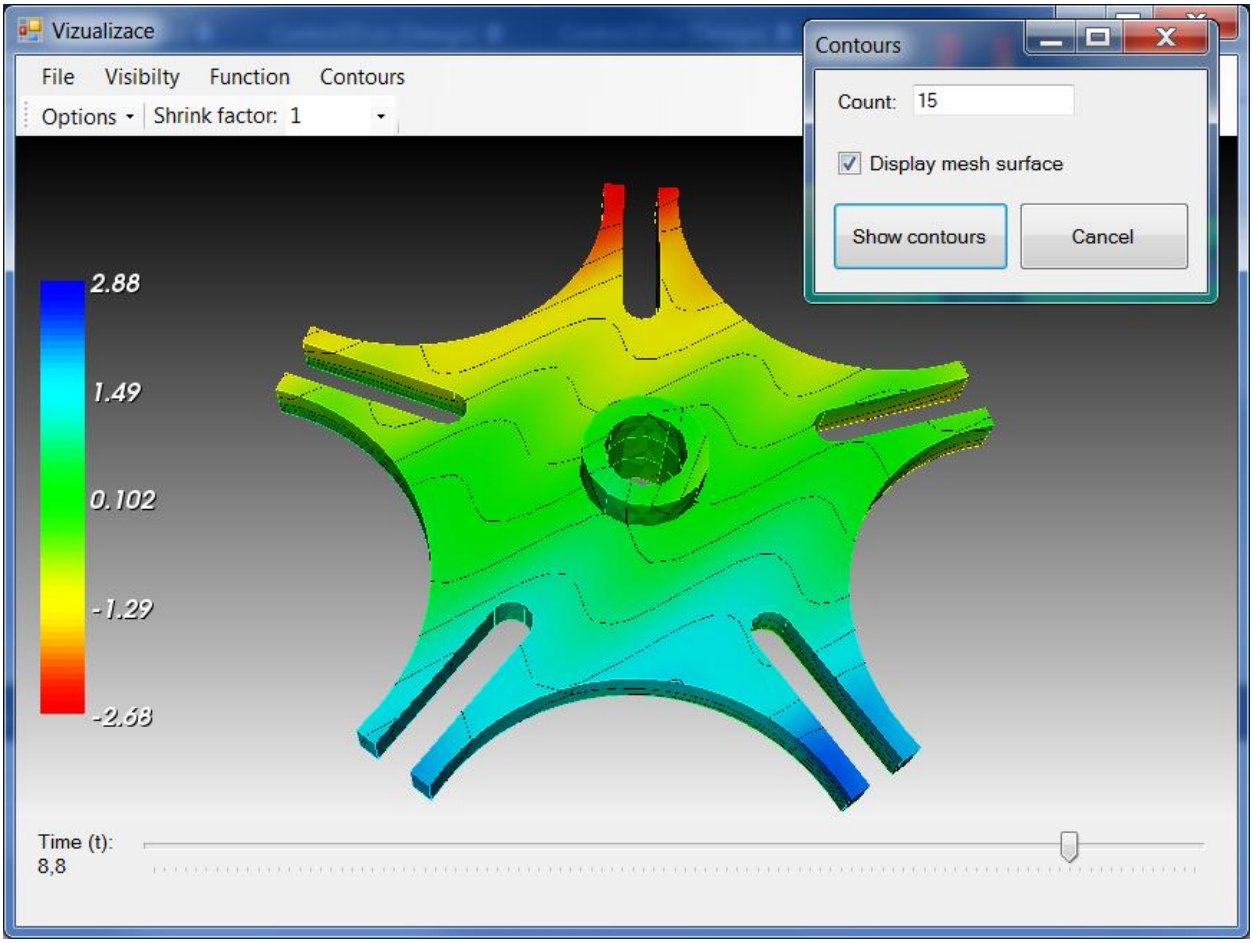
Obr. 13: Vizualizace skalárního pole 2 neznámých v Control2D



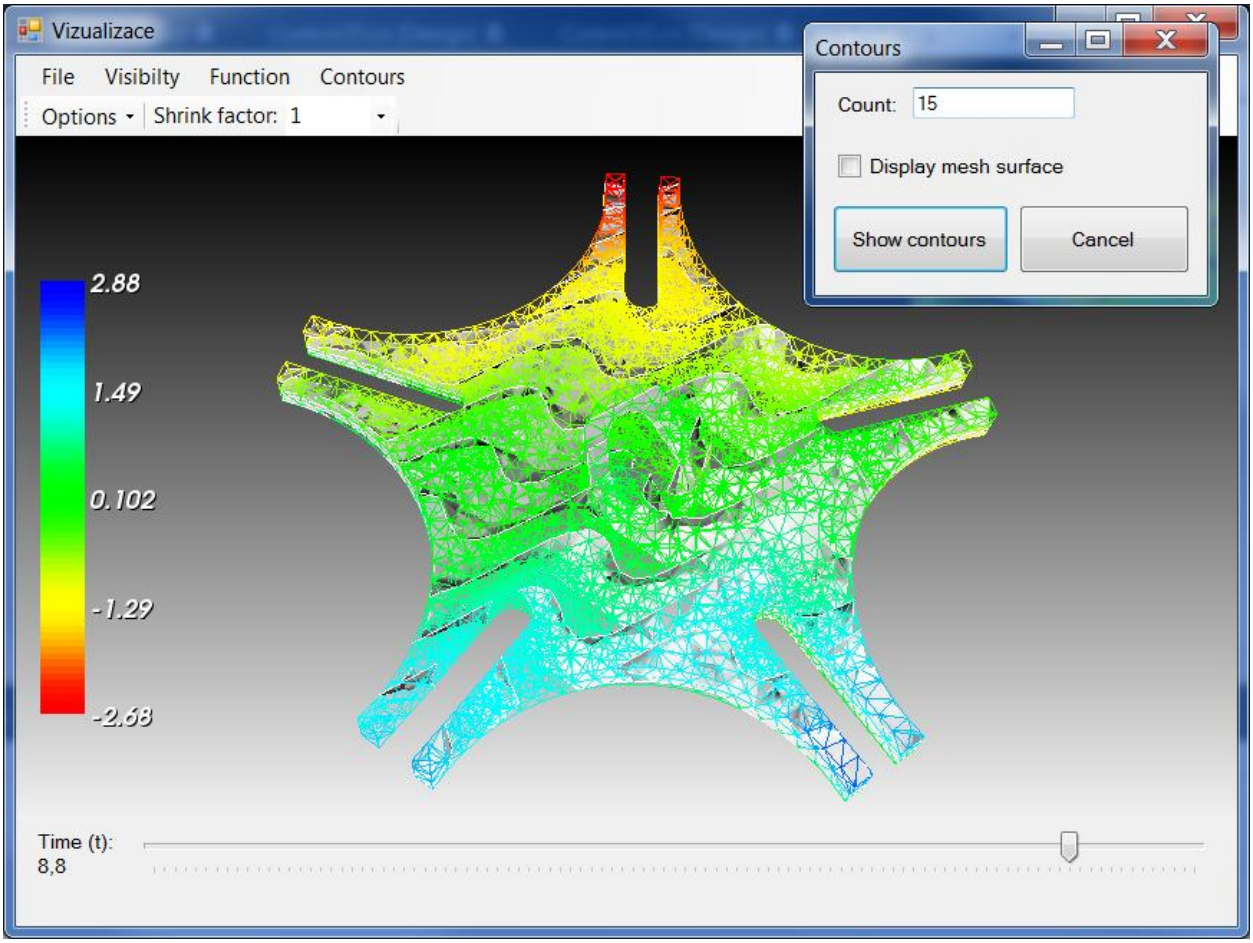
Obr. 14: Vizualizace skalárního pole 4 neznámých v povrchovém režimu v Control3D



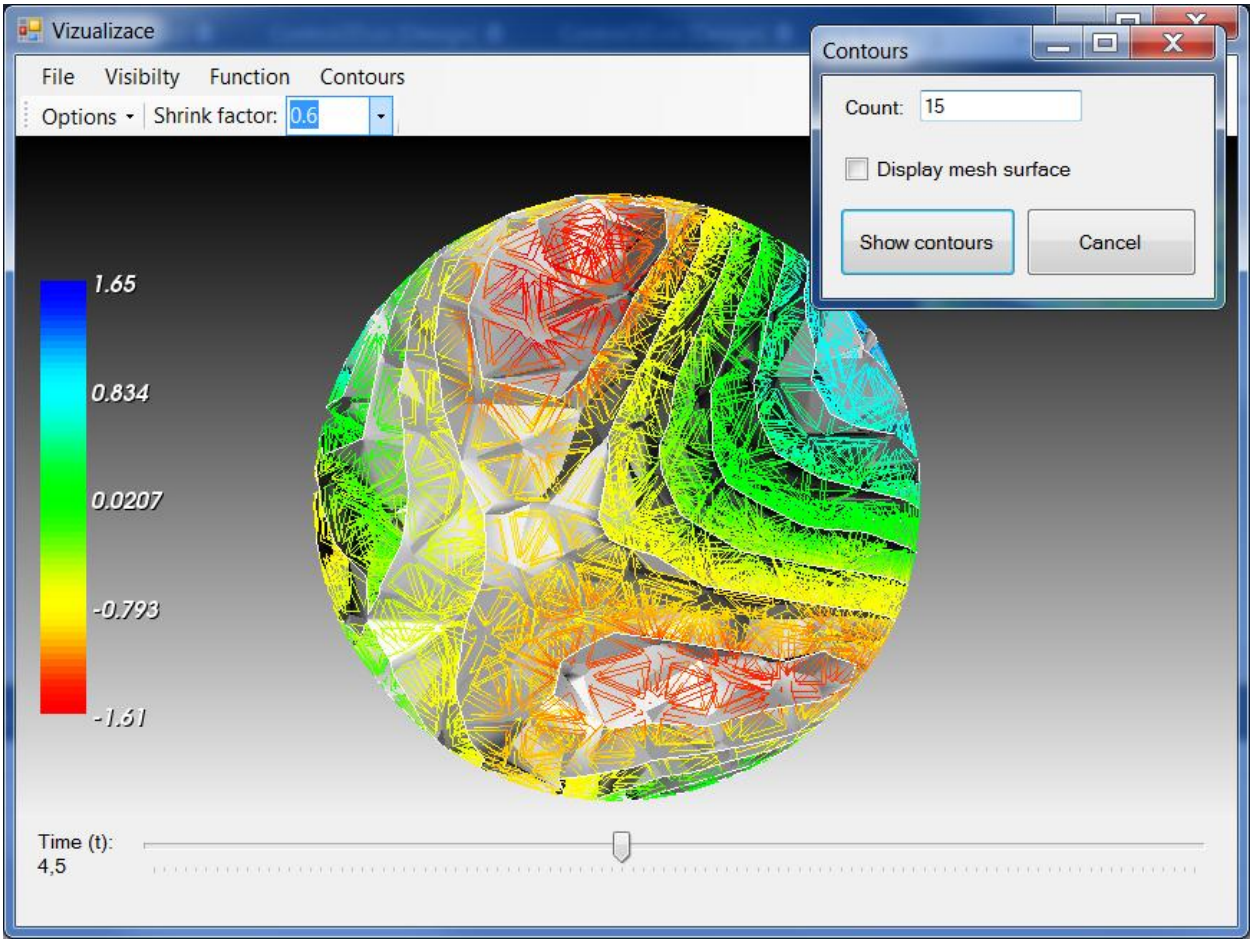
Obr. 15: Vrstevnice skalárního pole 3 neznámých v povrchovém režimu v Control2D



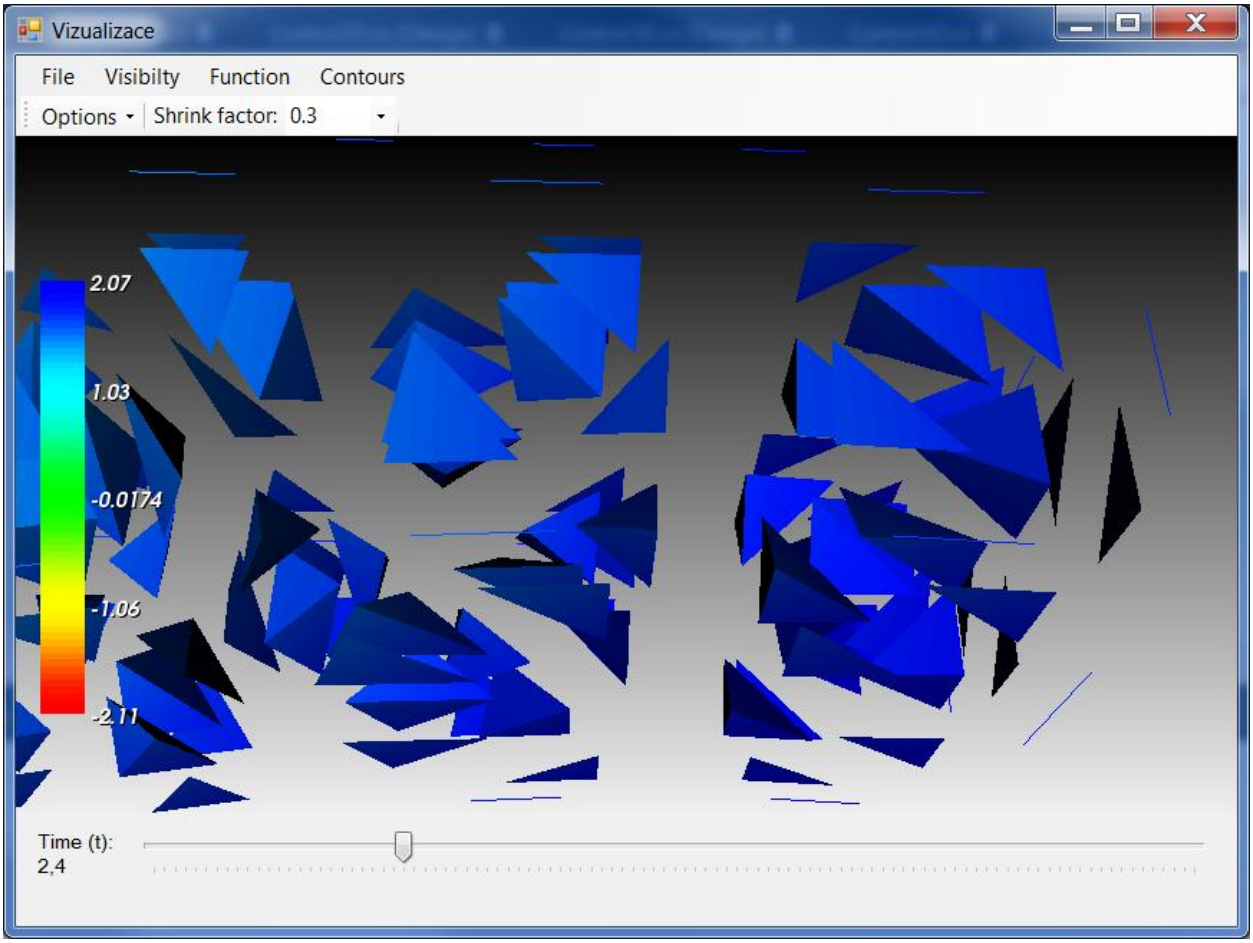
Obr. 16: Vrstevnice skalárního pole 4 neznámých v povrchovém režimu v Control3D



Obr. 17: Vrstevnice skalárního pole 4 neznámých v síťovém režimu v Control3D



Obr. 18: Vrstevnice skalárního pole 4 neznámých v síťovém režimu s aplikací filtru smrštění buněk s koeficientem 0,6 v Control3D



Obr. 19: Detail buněk sítě tělesa v povrchovém režimu s aplikací filtru smrštění buněk s koeficientem 0,3 v Control3D