ZÁKLADY VIZUALIZACE VĚDECKÝCH DAT

Skalární pole

Vektorová pole

Barevné sekvence

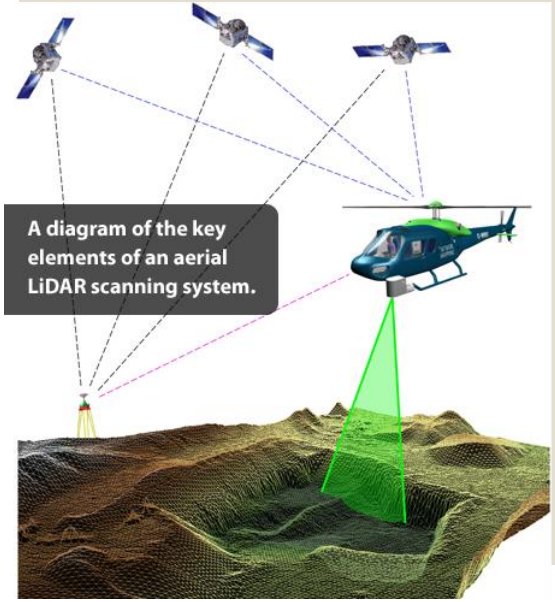
Glyfy

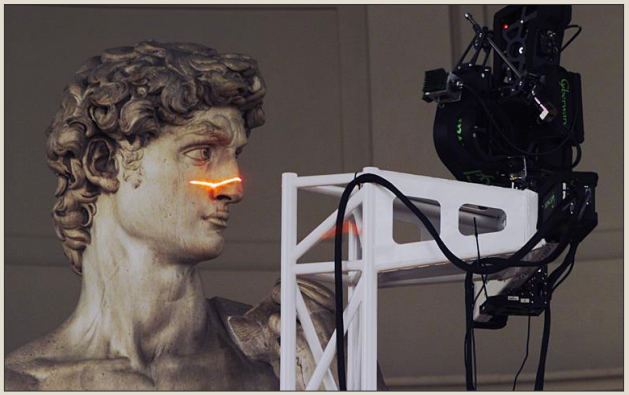
Streamlines

Vizualizační nástroje

VĚDECKÁ DATA

- Pocházejí z fyzikálních měření a simulací
- Několik málo příkladů:
  - LIDAR – skenování zemského povrchu
  - 3D skenování historických památek (resp. továren v USA)
  - Lékařská vyšetření počítačovou tomografií (CT, MRI, …)
  - Ultrazvuková vyšetření (např. "sono" jater)





VĚDECKÁ DATA

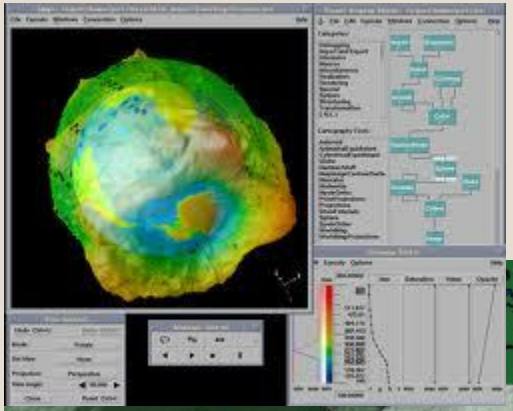
- Měření směru proudění vzduchu v aerodynamickém tunelu
- Měření teploty vzduchu v různých místech
- Simulace proudění krve v cévách
- Simulace zemětřesení
- Simulace zatížení kostí během pohybu
- Simulace vyhladovění nádorových buněk
- Simulace štěpení proteinů

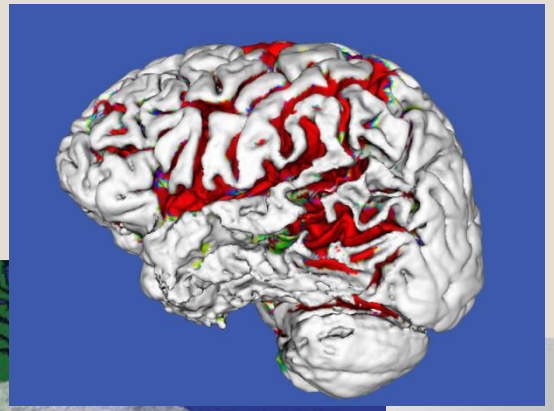
VIZUALIZACE VĚDECKÝCH DAT

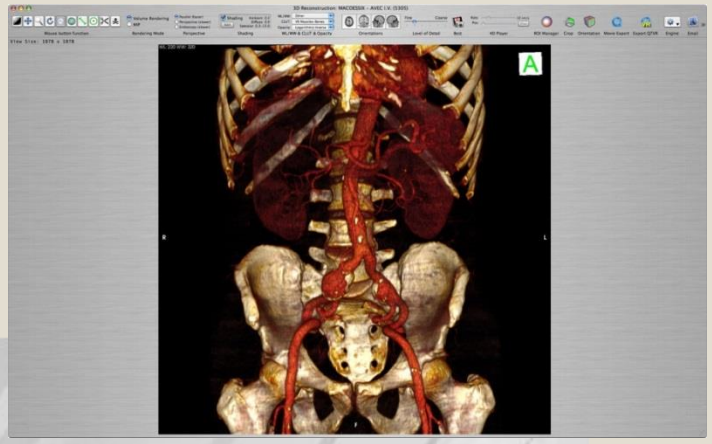
- Umožňuje snadnější pochopení problému
- Používána zejména v oblastech:
  - Aplikovaná matematika
  - Geofyzika
  - Chemie a biologie
    - Bioinformatika
  - Medicínský výzkum
    - Biomedical visualization
    - Biomechanika

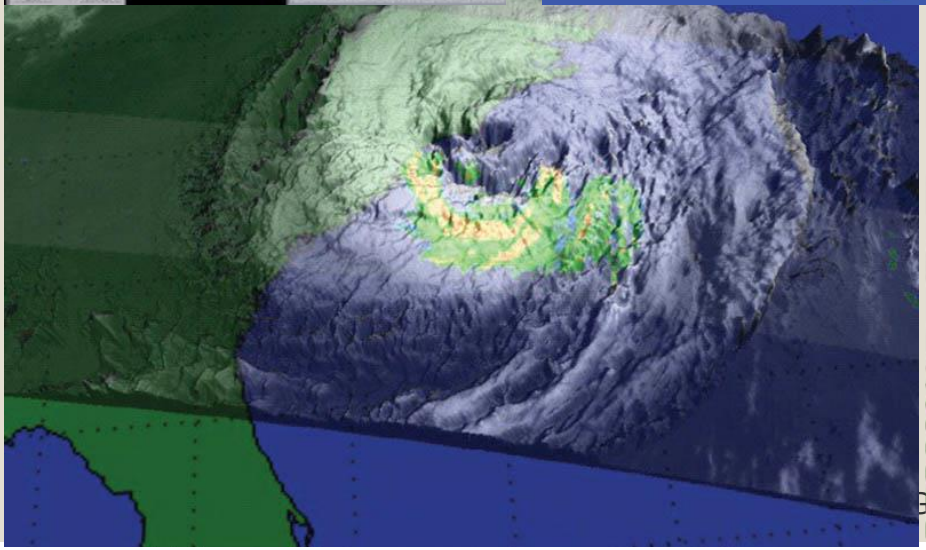
VIZUALIZACE VĚDECKÝCH DAT

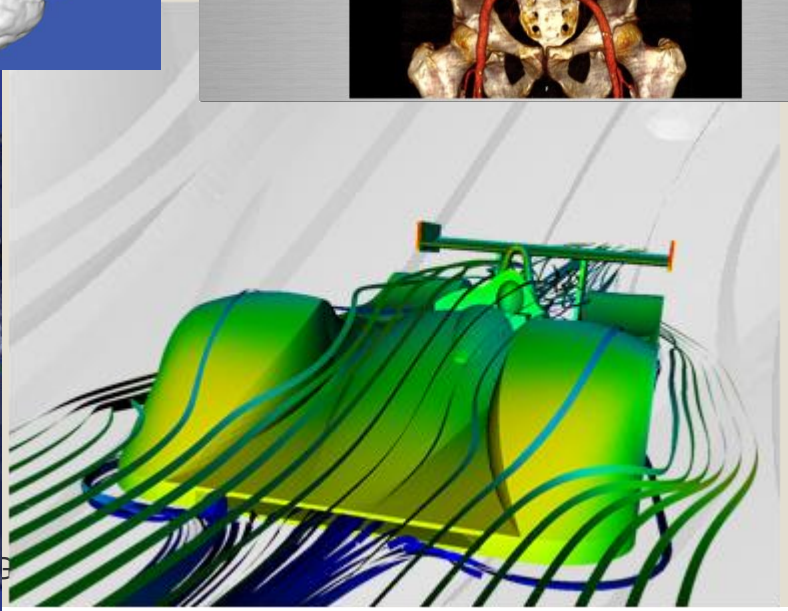
- Několik málo výsledných ukázek











VIZUALIZACE VĚDECKÝCH DAT

- Několik málo výsledných ukázek od nás

CHARAKTERISTIKA DAT

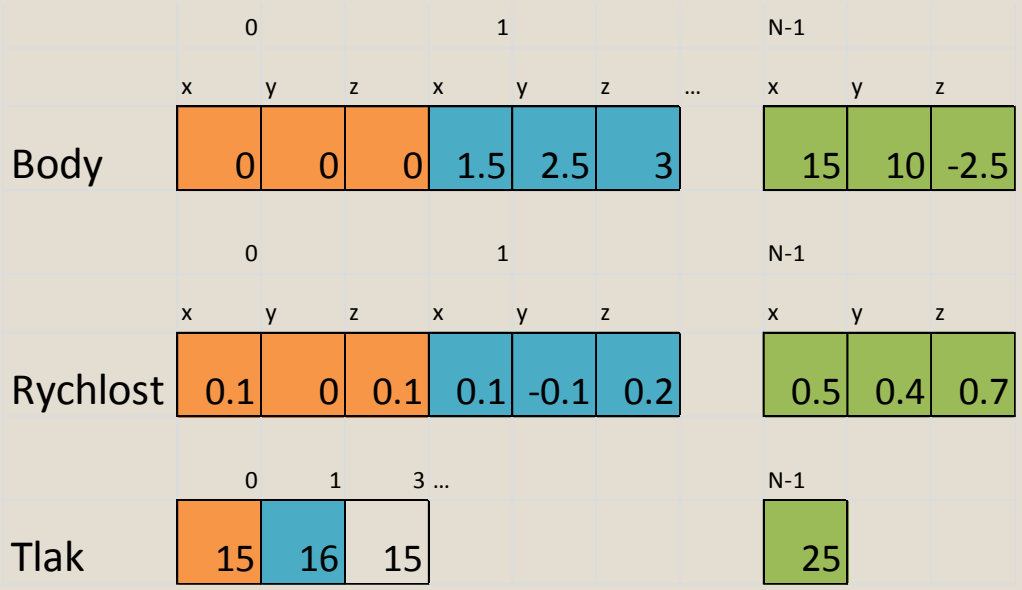
- VD = {P, PD, C, CD}
- P = uspořádaná množina bodů v E^d
- Dimenze bodů v praxi nejčastěji:
  - 1D – čas nebo parametrická vzdálenost od počátku
    - Např. senzory vlhkosti v nějaké místnosti
  - 2D – x, y
    - Pozor: body v Kartézském prostoru nemusí ležet na rovině
    - Např. zeměpisná šířka a výška
  - 3D – x, y, z nebo x, y, čas
    - Např. skenování 3D objektů (MS Kinect, …), objemová data
  - 4D – x, y, z a čas
    - Např. oskenovaný tlukot srdce

CHARAKTERISTIKA DAT

- PD = množina hodnot přiřazených k bodům
  - Každému bodu může být přiřazeno více hodnot
- Přiřazenou hodnotou je:
  - Skalár = reálná veličina
    - Např. teplota, tlak, elektrické napětí, výška, …
  - Vektor (nejčastěji ve 3D)
    - Např. rychlost, zrychlení, síla, momenty síly, gradient, …
  - Tensor = matice reálných čísel
    - Např. mechanické napětí (stress) – matice 3x3, deformace (strain) – matice 3x3
    - viz KIV/GSVD

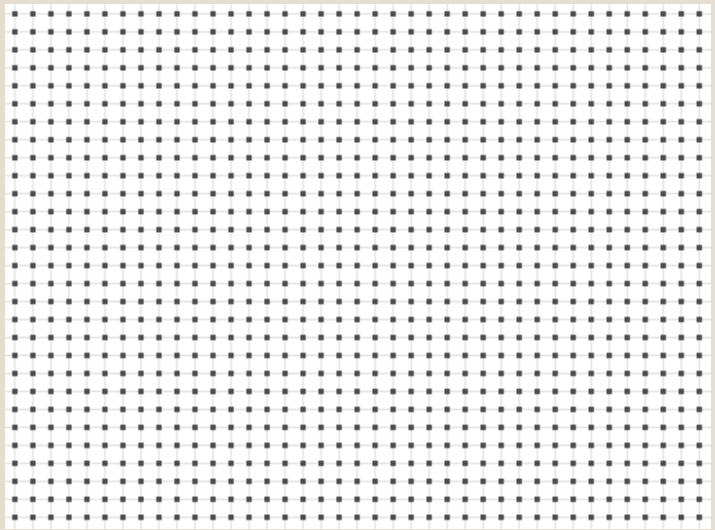
CHARAKTERISTIKA DAT

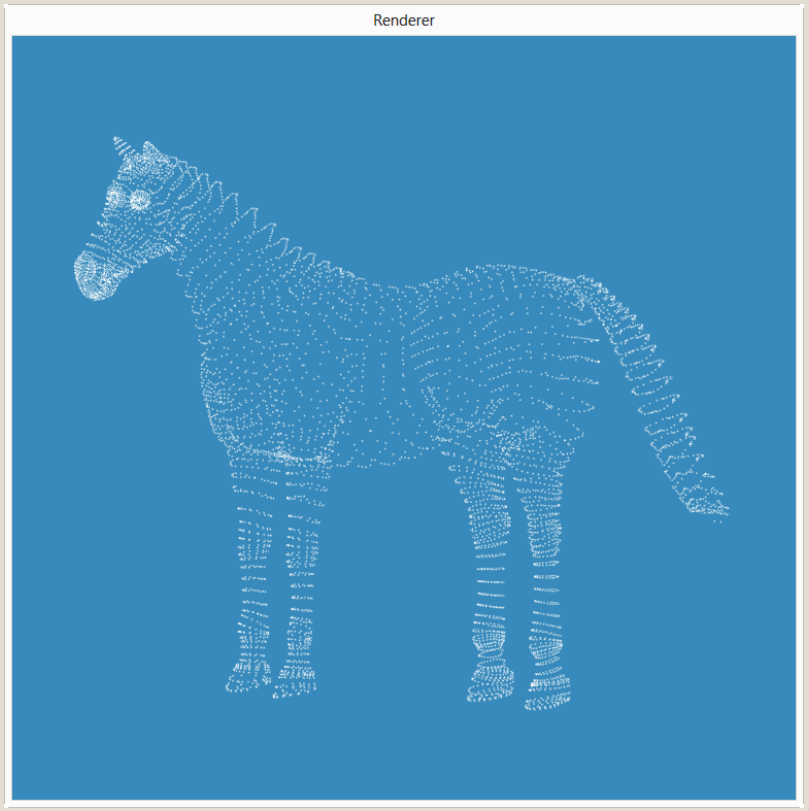
- Hodnoty ukládány v tzv. polích (např. skalární pole)
- Informace o bodech ukládána odděleně
  - Umožňuje mít v paměti informaci o bodech jen jednou pro libovolný počet polí (skalárních, vektorových, tensorových)
  - Každý bod jednoznačně identifikován svým indexem
    - Index do paměti pole



CHARAKTERISTIKA DAT

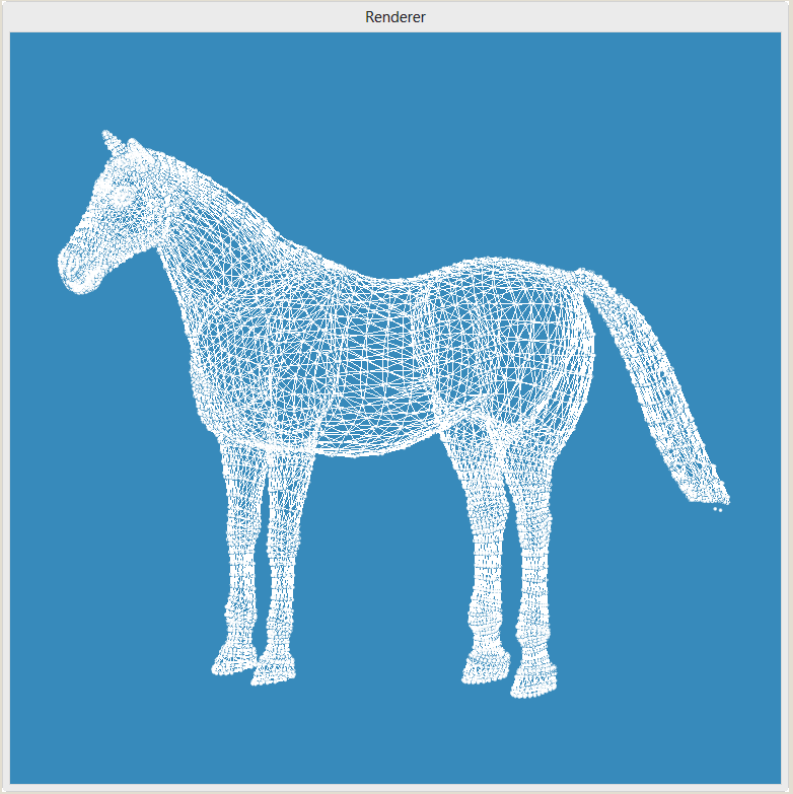
- Body mohou být rozmístěny:
  - Nestrukturovaně
  - Strukturovaně
    - Mřížky





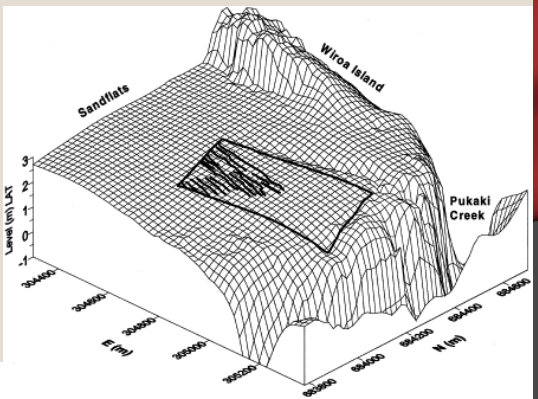
CHARAKTERISTIKA DAT

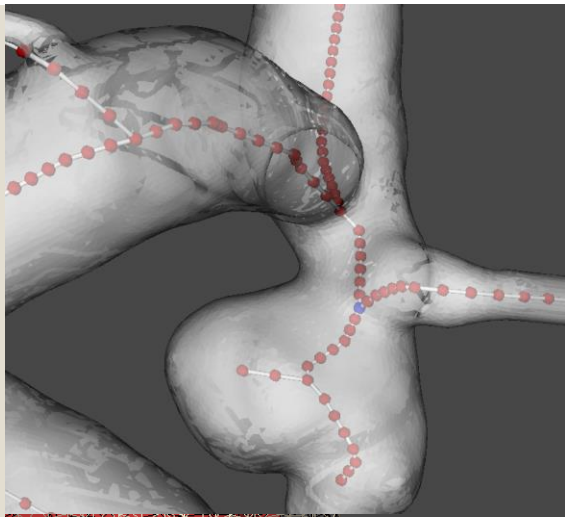
- Body mohou být propojeny hranami pro vyjádření jejich sousednosti
  - Pokud informace o sousednosti bodů chybí, je často nezbytné ji automaticky vypočítat
    - např. Delaunay triangulace, …
- Propojením vzniká:
  - Neorientovaný graf
  - C = Množina buněk

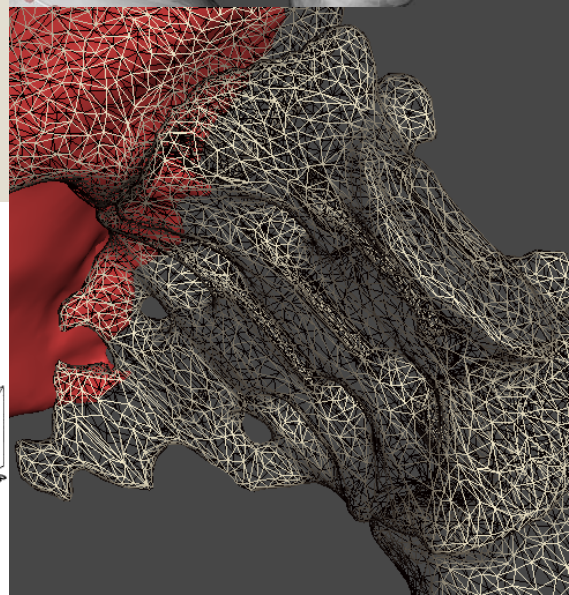


CHARAKTERISTIKA DAT

- Buňka = geometrický útvar
  - Vrcholy = podmnožina P
  - Žádný bod z P není uvnitř buňky
- Nejčastější tvary buněk pro nestrukturované rozložení bodů:
  - Skeleton, např. kostra cévy
    - Úsečka
  - Povrchová data, např. povrch svalu
    - Trojúhelník
    - Čtyřúhelník
      - Nemusí být na rovině

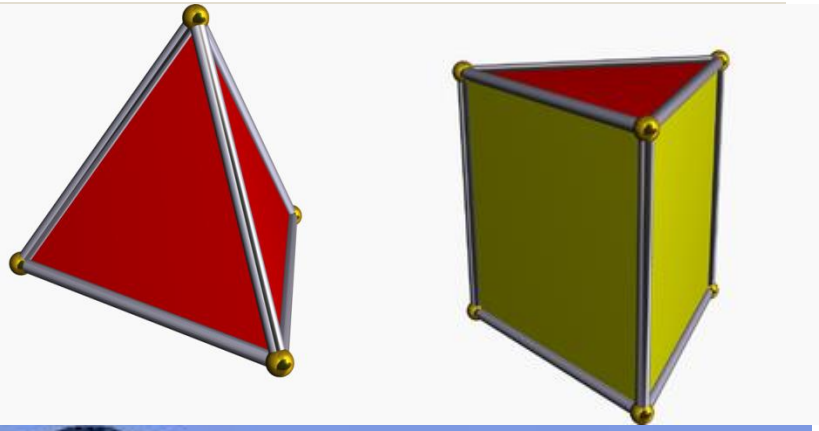


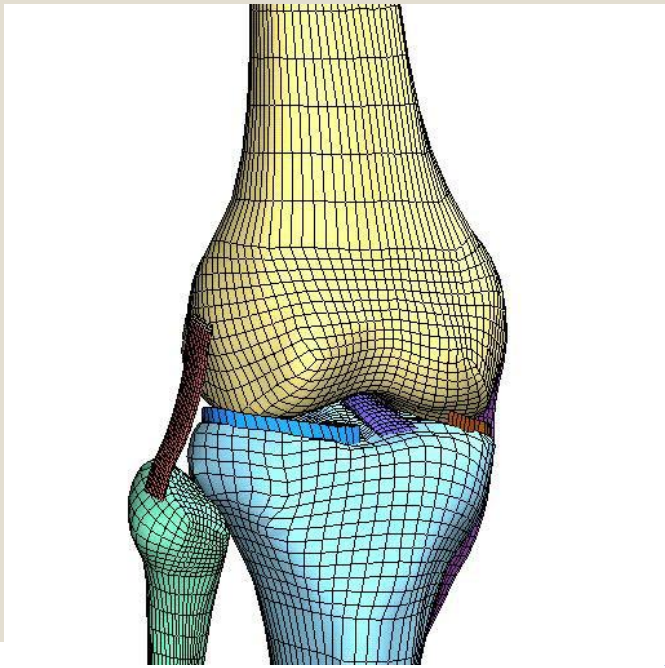


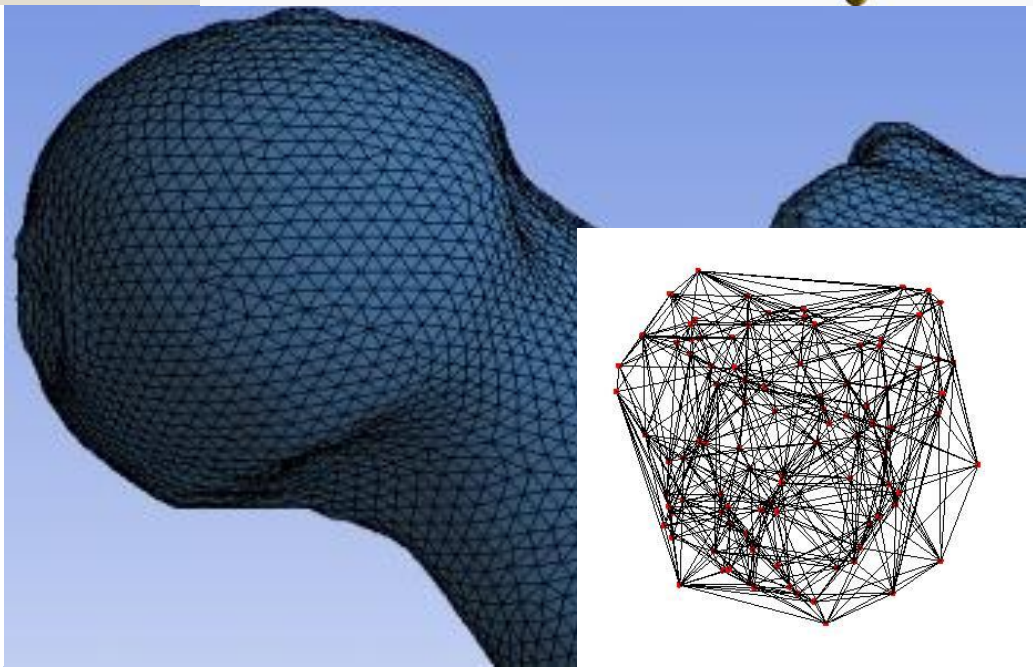


CHARAKTERISTIKA DAT

- Objemová data (body nejen na povrchu, ale také uvnitř)
  - Tetrahedron (čtyřstěn)
  - Hexahedron
  - Wedge

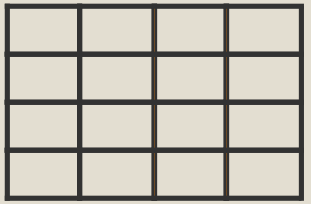


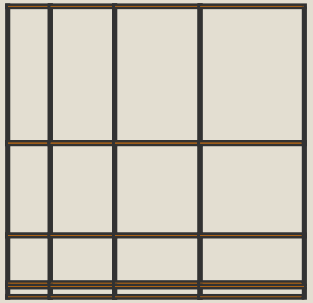




CHARAKTERISTIKA DAT

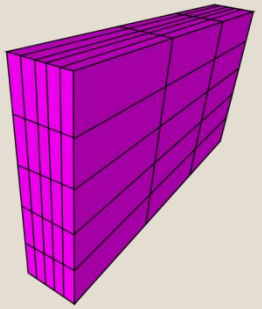
- Nejčastější tvary buněk pro strukturované rozložení bodů (body na mřížce):
  - Povrchová data, např. terén
    - Trojúhelník
    - Čtverec
      - Jedná se o pravidelnou mřížku
    - Obdélník
      - Jedná se o pravidelnou nebo pravoúhlá mřížka
  - Objemová data, např. CT data
    - Krychle
      - Jedná se o pravidelnou mřížku
    - Kvádr
      - Jedná se o pravidelnou nebo pravoúhlá mřížka



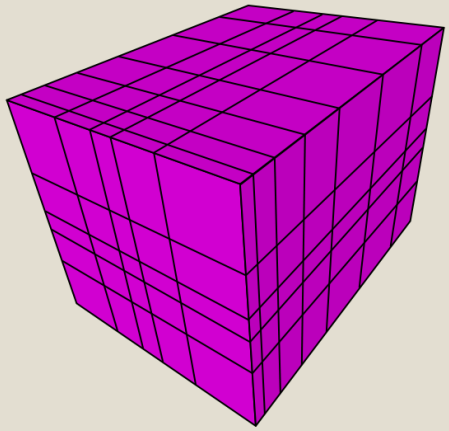


CHARAKTERISTIKA DAT

- Pravidelná mřížka (regular grid)
  - Postačuje uložit:
    - Souřadnice prvního bodu
      - Často lze ignorovat , pokud absolutní poloha není nedůležitá
    - Vzdálenost mezi body na jednotlivých osách
  - Polohu ostatních bodů lze vypočítat



- Pravoúhlá mřížka (rectilinear grid)
  - Postačuje uložit:
    - Souřadnice na jednotlivých osách
      - Např. pro objemová data ve 3D se jedná o 3 pole
  - Polohu ostatních bodů lze vypočítat



CHARAKTERISTIKA DAT

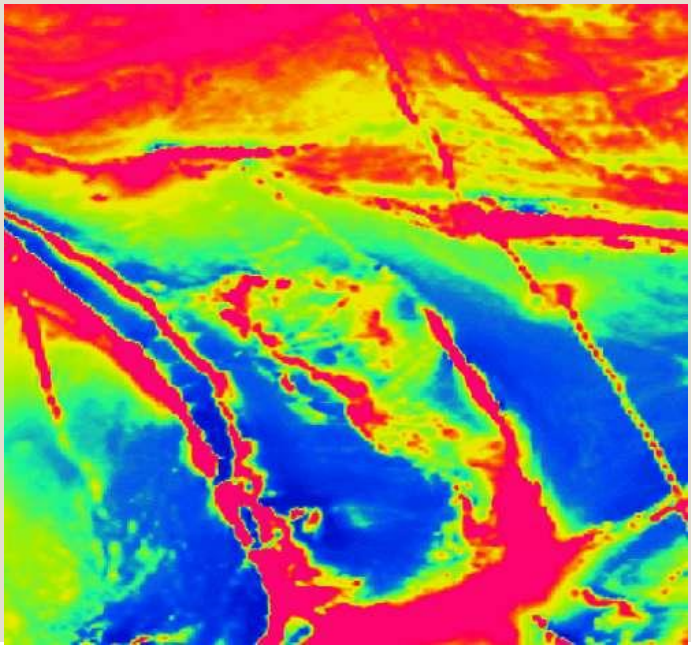
- Význam buněk:
  - Zjednodušuje odhadnutí hodnoty (např. teploty, tlaku) v obecném bodě $X \notin P$
    - Interpolace z hodnot ve vrcholech buňky
    - KIV/UPG: budeme uvažovat jen pravoúhlé a pravidelné mřížky
    - KIV/GSVD, KIV/ZPG, KIV/VD: obecná vědecká data
- CD =množina hodnot přiřazených k buňkám
  - Každé buňce může být přiřazeno více hodnot
  - Analogie k PD
- Kromě množiny bodů P ostatní množiny (PD, C, CD) mohou být prázdné
  - Obvykle CD není k dispozici

VIZUALIZAČNÍ PŘÍSTUPY

- Různé pro různé dimenze bodů
  - 2D (x, y) a 3D (x, y, z)
  - Jiné dimenze viz KIV/VD
- Různé pro různý typ hodnot
  - Skalární pole
  - Vektorová pole
  - Tenzorová pole viz KIV/GSVD
- Neexistuje univerzální přístup
- Nejvhodnější přístup vizualizace je závislý aplikaci

2D SKALÁRNÍ POLE: BAREVNÁ MAPA

- Předpoklad: pravidelná mřížka
  - O(x, y) = pozice počátku
  - Δx = vzdálenost mezi body na ose x
  - Δy = vzdálenost mezi body na ose y
  - m = počet bodů v ose x
  - n= počet bodů v ose y
- Data převedena na obrázek
  - Šířka = m
  - Výška = n
  - Barva pixelu I(i, j) = stanovena dle hodnoty přiřazené bodu (i, j)



2D SKALÁRNÍ POLE: BAREVNÁ MAPA

- Je-li (0, 0) okna v levém horním rohu (většina grafických knihoven), je nutné provést inverzi řádek, jinak by zobrazuji data vzhůru nohama
- Ukázka v pseudokódu:

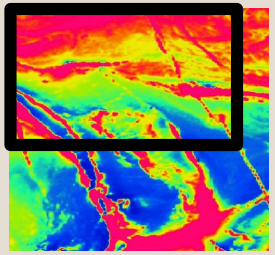
```
BufferedImage myImage = new BufferedImage(
     m, n, BufferedImage. TYPE_3BYTE_BGR);

for (int y = 0; y < n; y++) {
  for (int x = 0; x < m; x++) {
    myImage.setRGB(x, n – y - 1,
     vratBarvu(Data[y*m + x]));
  }
}
```

2D SKALÁRNÍ POLE: BAREVNÁ MAPA

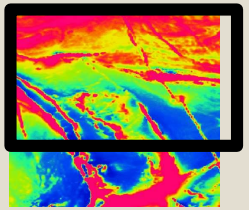
- Obrázek zobrazen roztažen do oblasti v okně:

```
g2.drawImage(myImage, 0, 0,
             Δx*m, Δy*n, null);
```



  - Mapuje 1 fyzickou jednotku v datech na 1 px
- Lze provést změnu měřítka (zoom):

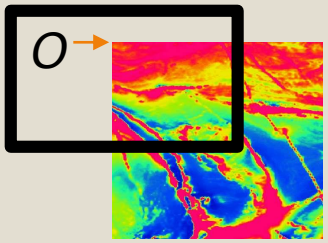
```
g2.drawImage(myImage, 0, 0,
             Δx*m*unitsToPx,
             Δy*n*unitsToPx, null);
```



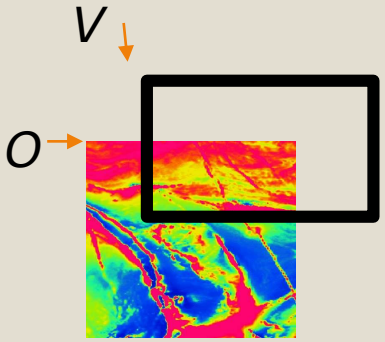
2D SKALÁRNÍ POLE: BAREVNÁ MAPA

- Využití počátku dat O(x, y):

```
g2.drawImage(myImage, Ox*unitsToPx,
     Oy*unitsToPx, Δx*m*unitsToPx,
     Δy*n*unitsToPx, null);
```



- Pohyblivé okno:
  - Levý horní rohu obrazovky (0, 0) odpovídá V(x, y) reálného světa



2D SKALÁRNÍ POLE: BAREVNÁ MAPA

- Předpoklady:
  - Barva reprezentována v RGB
  - Rozsah barevné složky je 0..1
  - Rozsah skalárních hodnot je 0..1
    - Lze provést jednoduché mapování min $\rightarrow$ 0, max $\rightarrow$ 1
- Pro daný bod/pixel a hodnotu h se stanoví:
  - $r = f_r(h)$, $g = f_g(h)$, $b = f_b(h)$
  - Funkce f_r, f_g a f_b jsou tzv. **přenosové funkce**
  - Mohou, ale nemusí být stejné
  - Stejné funkce = obrázek ve škále šedé
  - Různé funkce = obrázek barevný

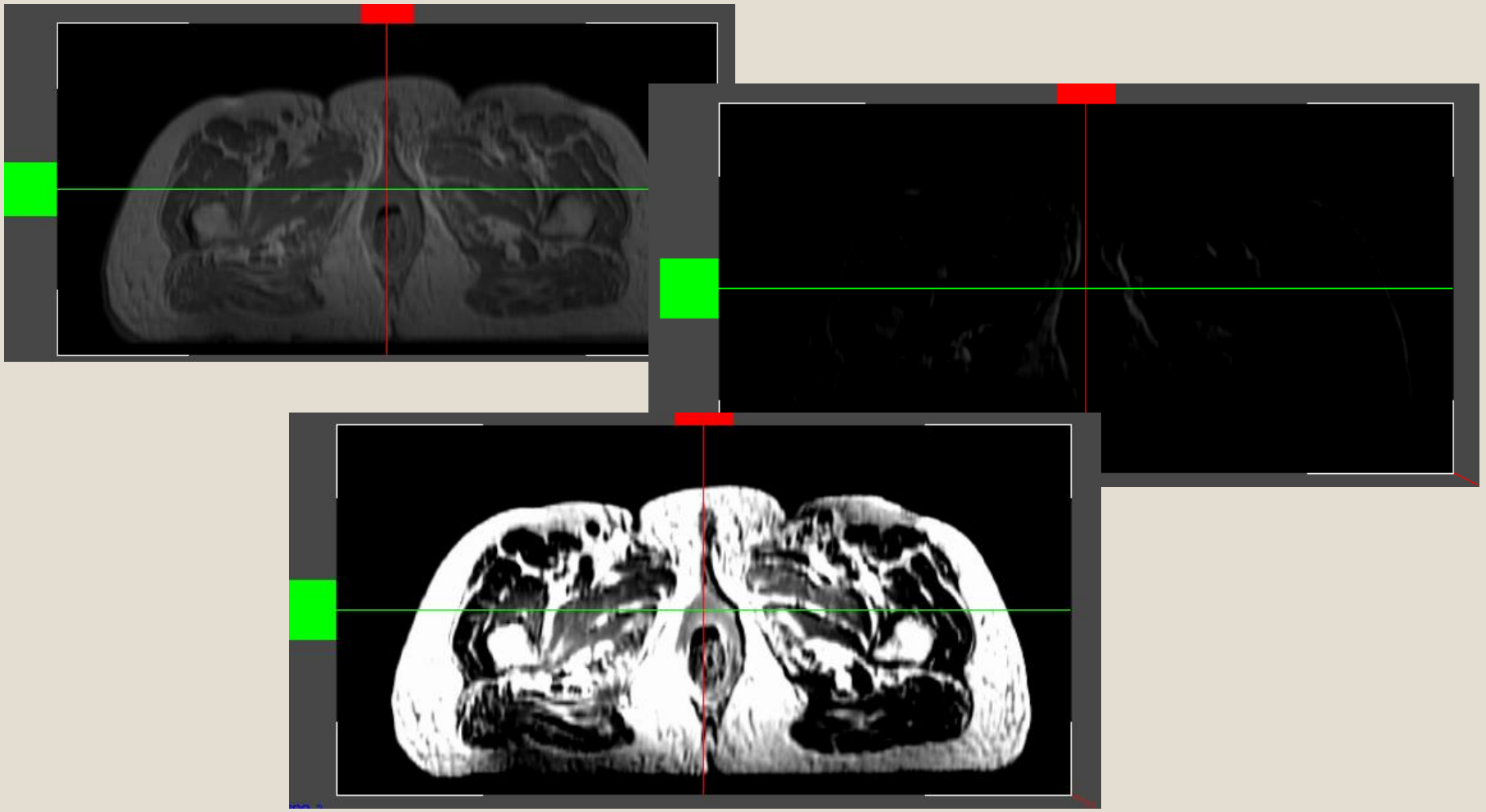
2D SKALÁRNÍ POLE: BAREVNÁ MAPA

- Přenosová funkce je často složená funkce
  - $f_r(h) = \varphi_r\big(\psi(h)\big), f_g(h) = \varphi_g\big(\psi(h)\big),\ f_b(h) = \varphi_b\big(\psi(h)\big)$

    nebo

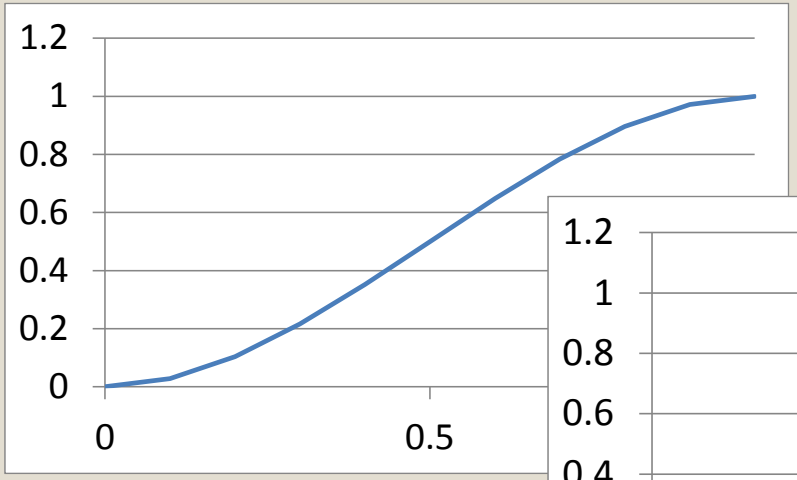
  - $f_r(h) = \psi\big(\varphi_r(h)\big), f_g(h) = \psi\left(\varphi_g(h)\right),\ f_b(h) = \psi\big(\varphi_b(h)\big)$
- Přenosová funkce ψ řídí výsledný kontrast
  - ψ(h) > h $\rightarrow$ zesvětlení místa
  - ψ(h) < h $\rightarrow$ ztmavení místa
  - Slouží k potlačení šumu v nějaké oblasti
    - Např. namísto černo-černého šumu budu mít jen černou
    - Vede k zvýšení kontrastu

2D SKALÁRNÍ POLE: BAREVNÁ MAPA

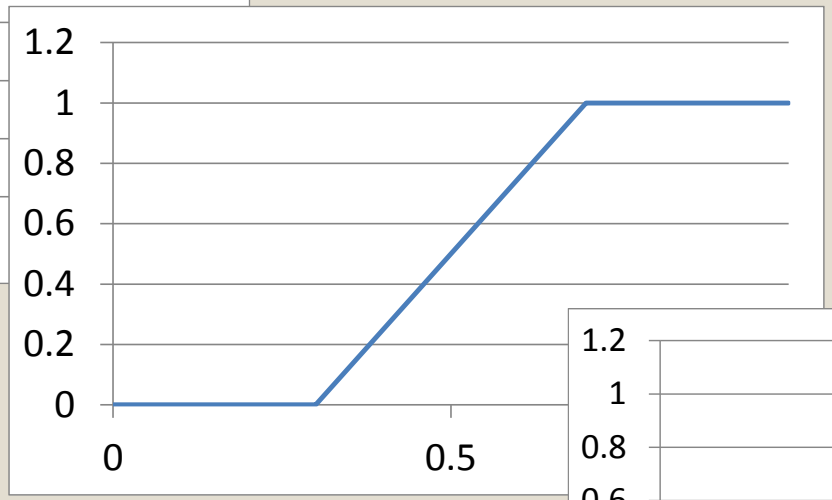


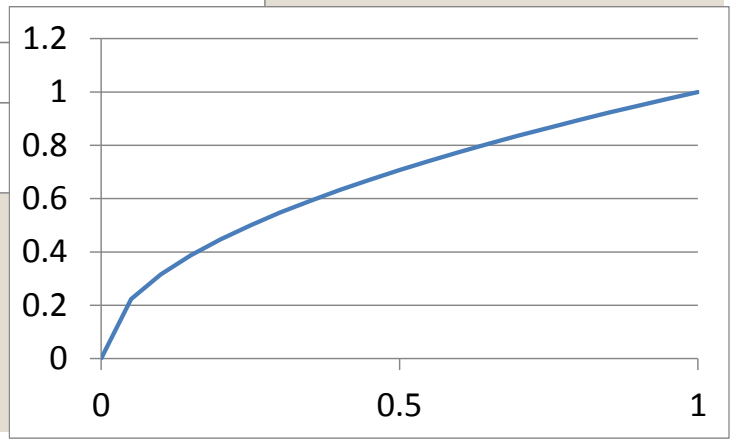
2D SKALÁRNÍ POLE: BAREVNÁ MAPA

- Nejčastější tvar přenosové funkce ψ je "S" nebo "log"



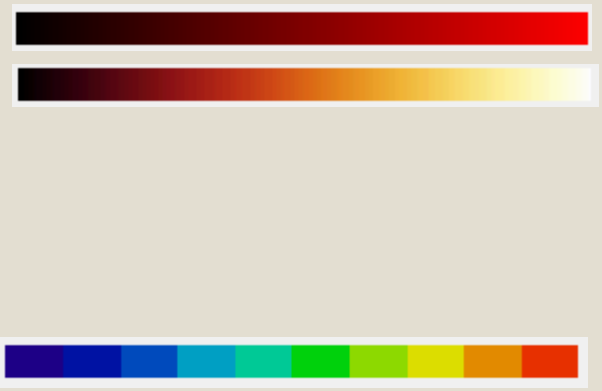
$3 \cdot h^2 - 2 \cdot h^3$

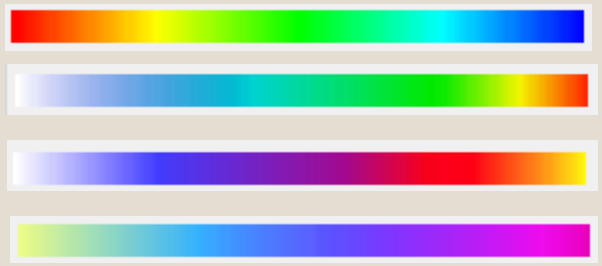




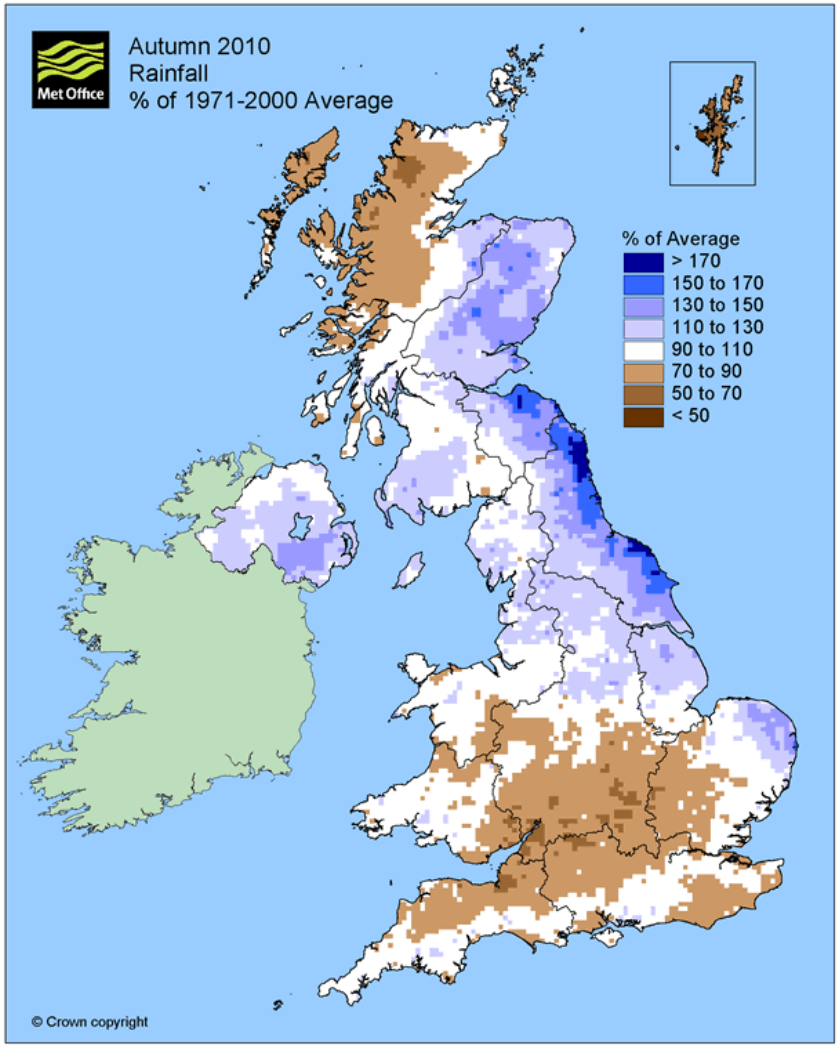
2D SKALÁRNÍ POLE: BAREVNÁ MAPA

- Přenosová funkce $\varphi_{r,g,b}$ slouží k obarvení
  - Volba barevné škály je typicky aplikačně závislá
  - Často se jedná po částech lomenou funkci
    - Pro klíčové hodnoty se stanoví RGB barva ostatní interpolací v barevném prostoru RGB nebo častěji v HSV
    - Lineární interpolace nebo interpolace nejbližším sousedem





2D SKALÁRNÍ POLE: BAREVNÁ MAPA



2D SKALÁRNÍ POLE: BAREVNÁ MAPA

- Naivní implementace – nejbližší soused
  - ```
    if (h < H_THR1) return Color.RED;
    if (h < H_THR2) return Color.GREEN;
    return Color.BLUE;
    ```
- Lepší implementace – nejbližší soused
  - ```
    class BarevnaMapa {
       public double h_thr;   //prahová hodnota
       public Color h_clr;    //přiřazená barva
    }
    …
    BarevnaMapa[] bm = new …  //naplnění (např. ze souboru)
    ```
  - ```
    for (int i = 0; i < bm.length; i++) {
       if (h < bm[i].h_thr) return bm[i].h_clr;
    }
    return bm[bm.length-1].h_clr;
    ```
    ```

2D SKALÁRNÍ POLE: BAREVNÁ MAPA

- Implementace – lineární interpolace

```
int low = 0;
while (bm[low].h_thr < h) low++;

int hi = low + 1;
double t = (h - bm[low].h_thr) /
   (bm[hi].h_thr - bm[low].h_thr);

return new Color(
   (1-t)*bm[low].h_clr.getRed() +
   t*bm[hi].h_clr.getRed(),

   (1-t)*bm[low].h_clr.getGreen() +
   t*bm[hi].h_clr.getGreen(),

   (1-t)*bm[low].h_clr.getBlue() +
   t*bm[hi].h_clr.getBlue());
```

2D SKALÁRNÍ POLE: BAREVNÁ MAPA

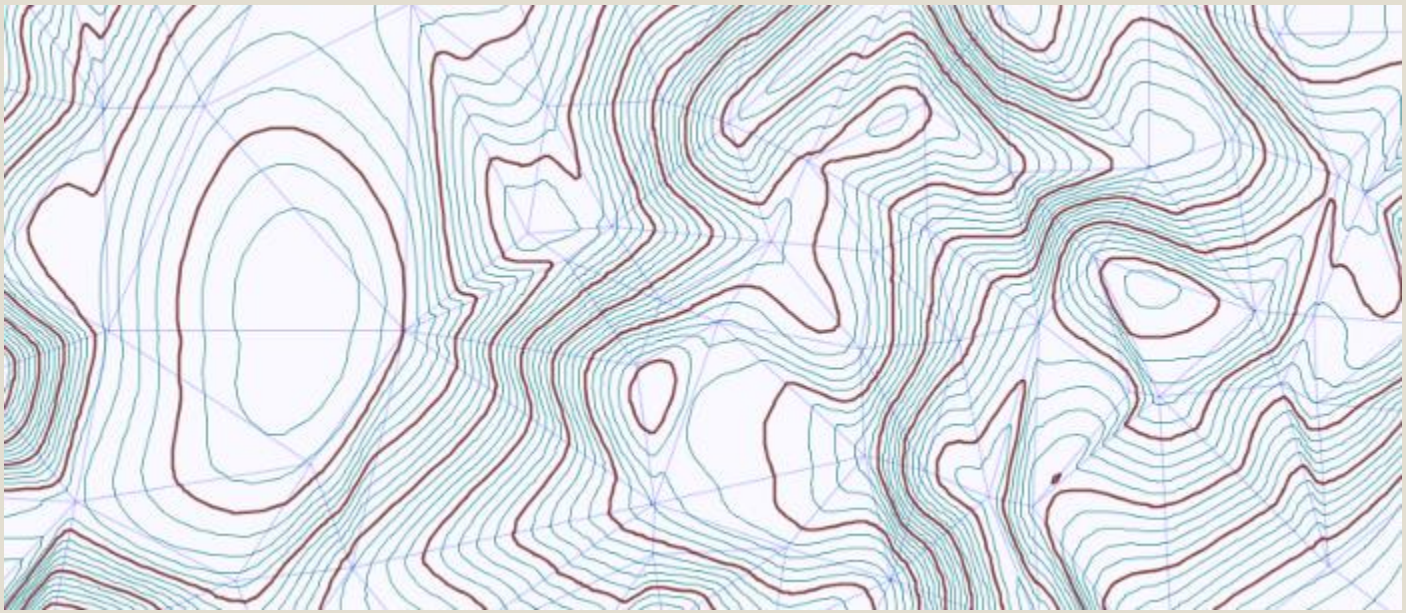
- Často počet různých hodnot omezený (např. 256)
  - Případně lze vstupní hodnoty omezit přenosovou funkcí ψ
- Hodnoty funkce vhodné spočítat v předzpracování a uložit do tabulky (tzv. lookup table)
- Pro mapování hodnoty na barvu pak se použije:
  - `Color[] lut = …`
  - `Color = lut[h];`
- Maximální časová efektivita za cenu malé paměti
  - Např. pro 256 hodnot = 768 B

2D SKALÁRNÍ POLE: BAREVNÁ MAPA

- Java poskytuje třídy `ByteLookupTable` a `ShortLookupTable` + operátor `LookupOp`

2D SKALÁRNÍ POLE: KONTURY

- Kontura, iso-čára nebo také vrstevnice = spojnice míst se stejnou hodnotou
  - Typicky neprochází body z P
  - viz KIV/ZPG



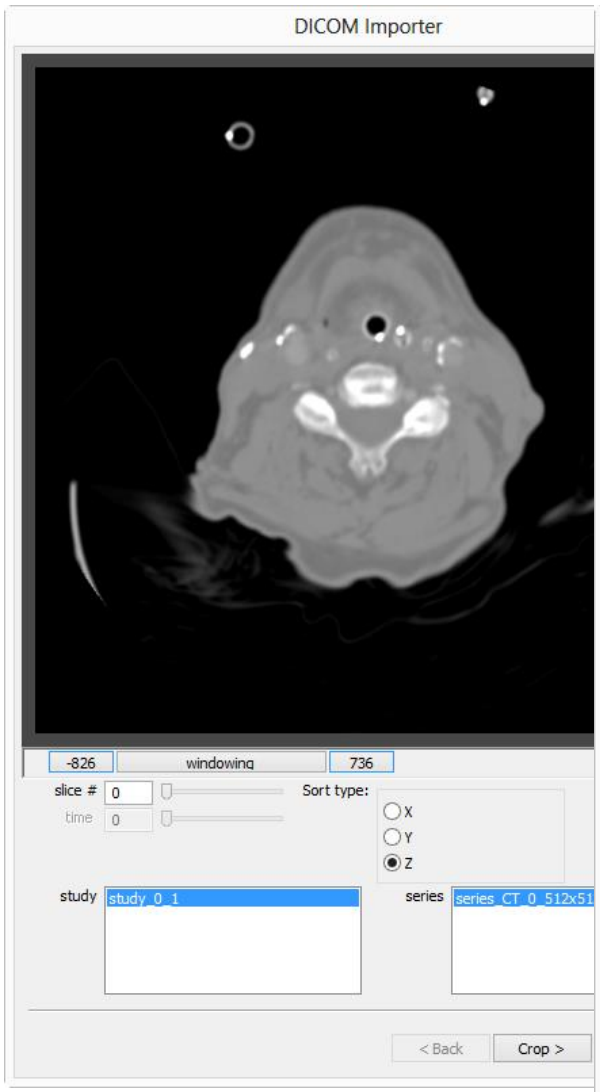
3D SKALÁRNÍ POLE: BAREVNÁ MAPA

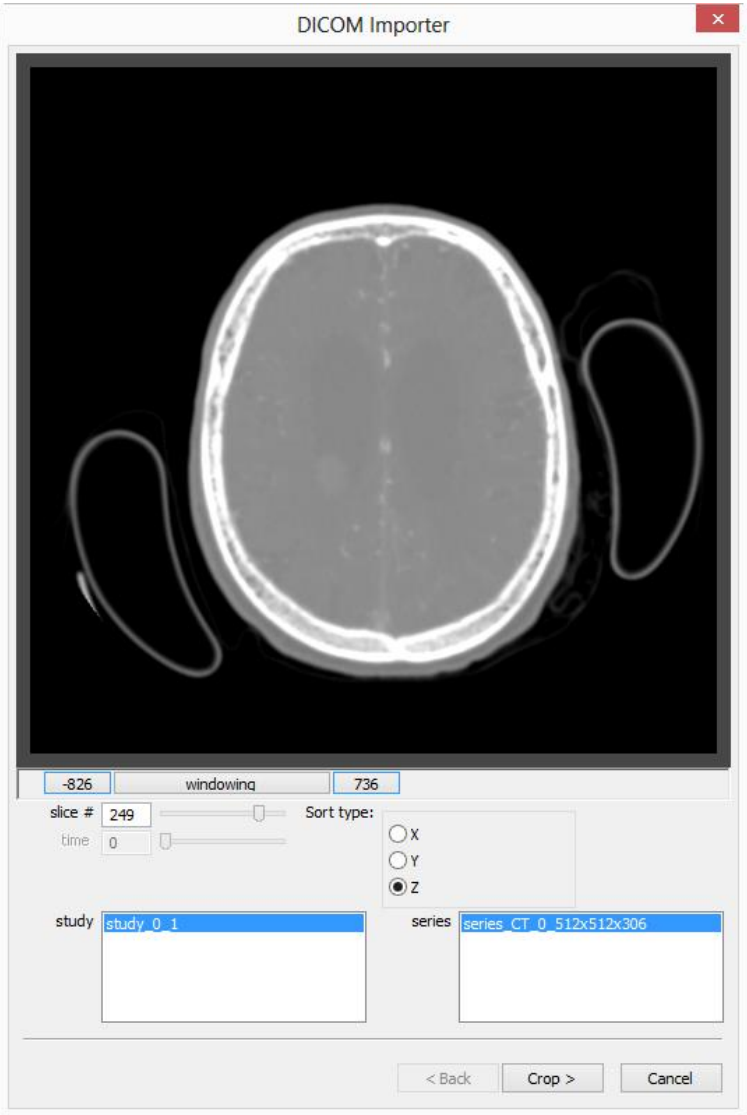
- Předpoklad: pravidelná mřížka
  - O(x, y, z) = pozice počátku
  - Δx = vzdálenost mezi body na ose x
  - Δy = vzdálenost mezi body na ose y
  - Δz = vzdálenost mezi body na ose z
  - n_x= počet bodů v ose x
  - n_y= počet bodů v ose y
  - n_z= počet bodů v ose z
- Skalární pole v datech ukládáno výhradně po řádcích jako 3D matice (x pak y a pak z)
  - Lze na něj nahlížet jako na uspořádanou množinu 2D skalárních polí → lze vizualizovat po "vrstvách"

3D SKALÁRNÍ POLE: BAREVNÁ MAPA

- Předpoklad: pravidelná mřížka
  - O(x, y, z) = pozice počátku
  - Δx = vzdálenost mezi body na ose x
  - Δy = vzdálenost mezi body na ose y
  - Δz = vzdálenost mezi body na ose z
  - n_x= počet bodů v ose x
  - n_y= počet bodů v ose y
  - n_z= počet bodů v ose z
- Skalární pole v datech ukládáno výhradně po řádcích jako 3D matice (x pak y a pak z)
  - Lze na něj nahlížet jako na uspořádanou množinu 2D skalárních polí → lze vizualizovat po "vrstvách"

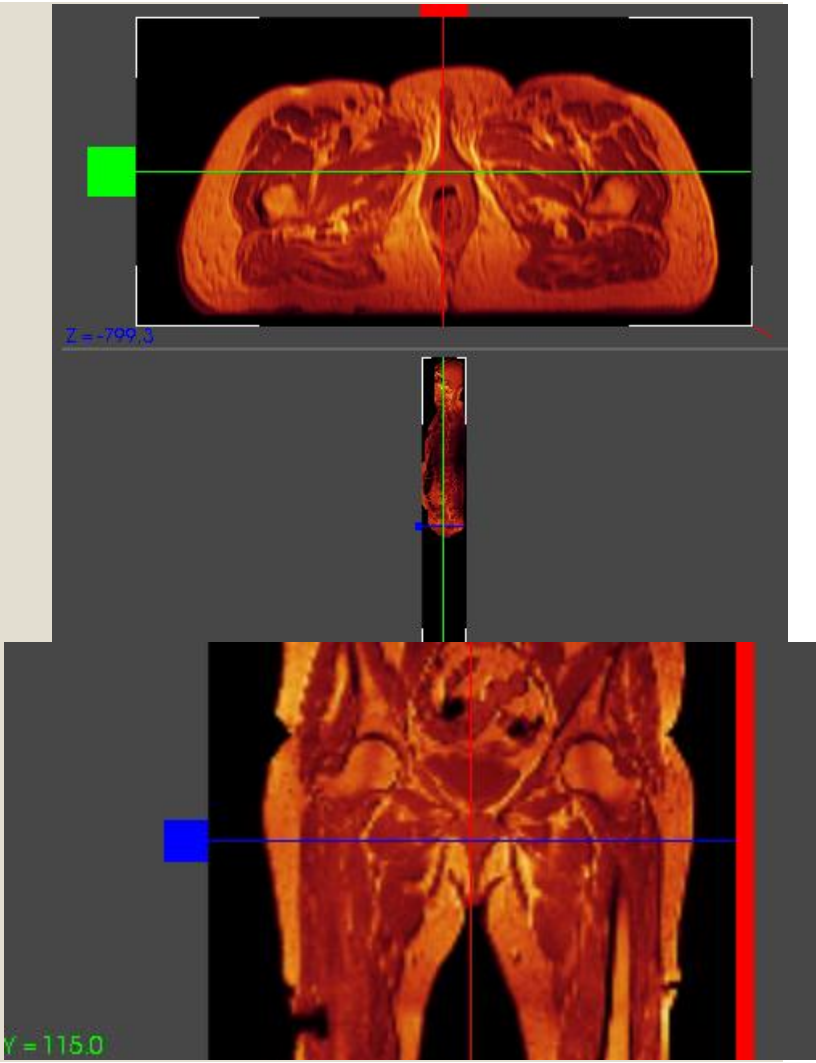
3D SKALÁRNÍ POLE: BAREVNÁ MAPA





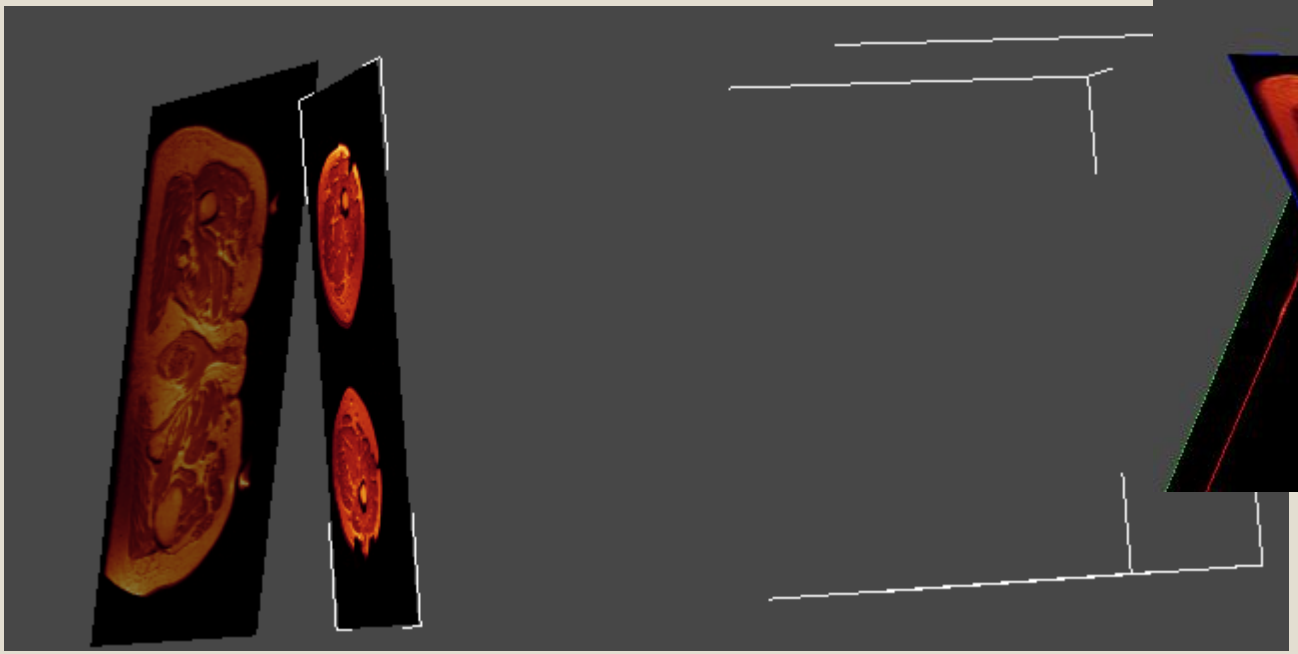
3D SKALÁRNÍ POLE: BAREVNÁ MAPA

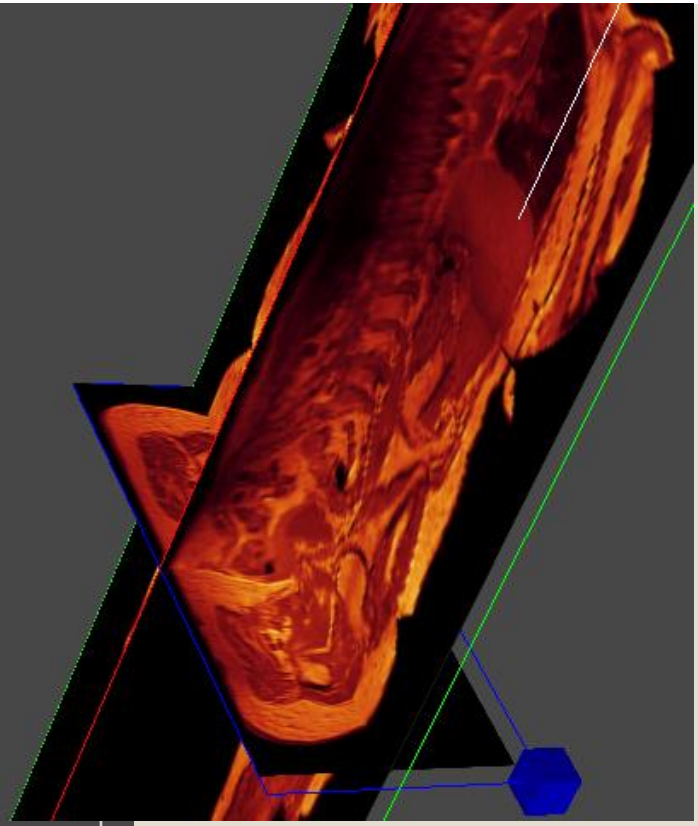
- Data lze rovněž zobrazit po řezech na ose x nebo y
  - Obtížnější extrahovat hodnoty
- Vhodné uživateli nechat možnost, aby si vybral, který řez chce vidět
- Vhodné umožnit uživateli obrázek přiblížit
  - Zejména, pokud se obrázek zobrazuje na menší plochu, než je jeho nativní rozlišení



3D SKALÁRNÍ POLE: BAREVNÁ MAPA

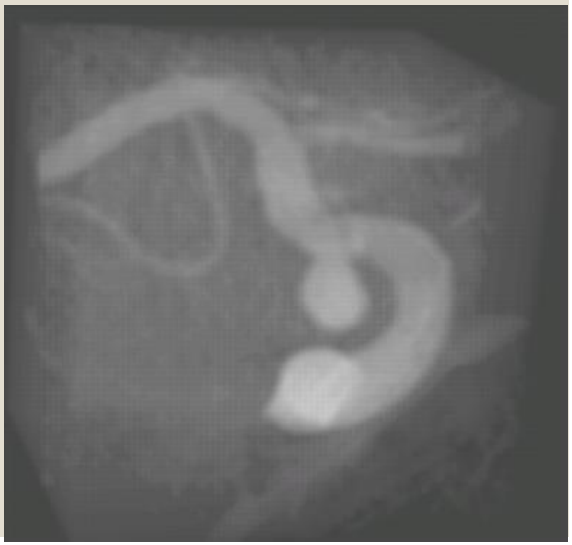
- Řezy lze rovněž zobrazit ve 3D
  - Náročnější – viz základy 3D grafiky
- Řezy nemusí být kolmé na nějakou z os x, y nebo z





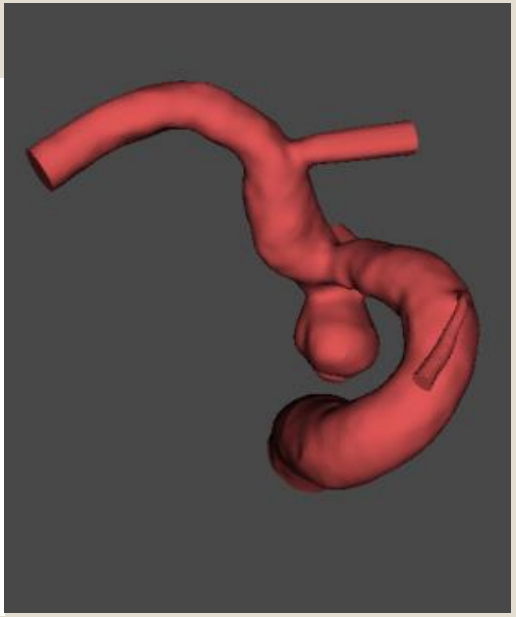
3D SKALÁRNÍ POLE: BAREVNÁ MAPA

- Problémy s řezy:
  - Uživatel musí si 3D model zkompletovat v hlavě
- Možné řešení = Direct Volume Rendering (DVR)
  - Zjednodušeně: obraz stanoven kompozicí více řezů
    - Nemusí být ve směru jedné z os x, y, z
  - Různé kompoziční přístupy
  - Nejjednodušší = Maximum Intensity Projection (MIP)
    - Z = MAX(X, Y)
    - Vhodné např. pro CT
      - Kosti mají vyšší hodnoty
  - Více viz KIV/ZPG



3D SKALÁRNÍ POLE: ISOPLOCHA

- Obdoba vrstevnic z 2D
- Uživatel typicky musí stanovit iso-hodnotu
- Plocha procházející danou iso-hodnotou je vytvořena
  - Nejčastěji se jedná o trojúhelníkovou síť
  - Různé přístupy – viz KIV/ZPG
    - Např. Marching Cubes
      - Základní algoritmus
- Plocha zobrazena jako 3D objekt (ve 3D)
  - viz Základy 3D grafiky
  - Může být poloprůhledná

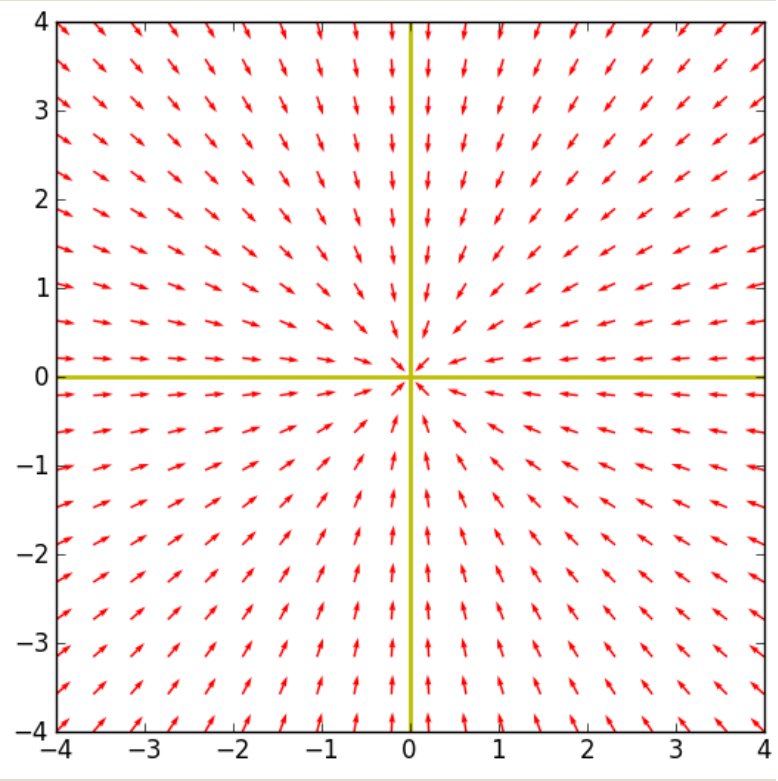


VEKTOROVÁ POLE: BAREVNÁ MAPA

- Vektorové pole převedeno na skalární
- Skalární pole zobrazeno již známými technikami
- Možnosti převodu:
  - Vektor na velikost vektoru (magnituda)
    - Např. rychlost $v(x, y, z) \rightarrow$ velikost rychlosti $\|v\| = \sqrt{{v_x}^2 + {v_y}^2 + {v_z}^2}$
  - Pouze vybraná složka vektoru
    - Např. rychlost $v(x, y, z) \rightarrow$ velikost rychlosti v ose y

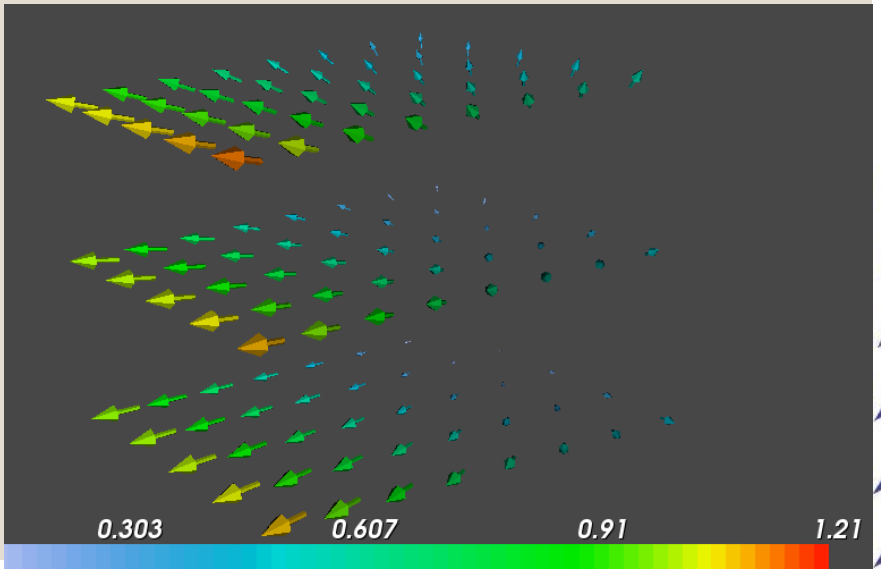
VEKTOROVÁ POLE: GLYFY

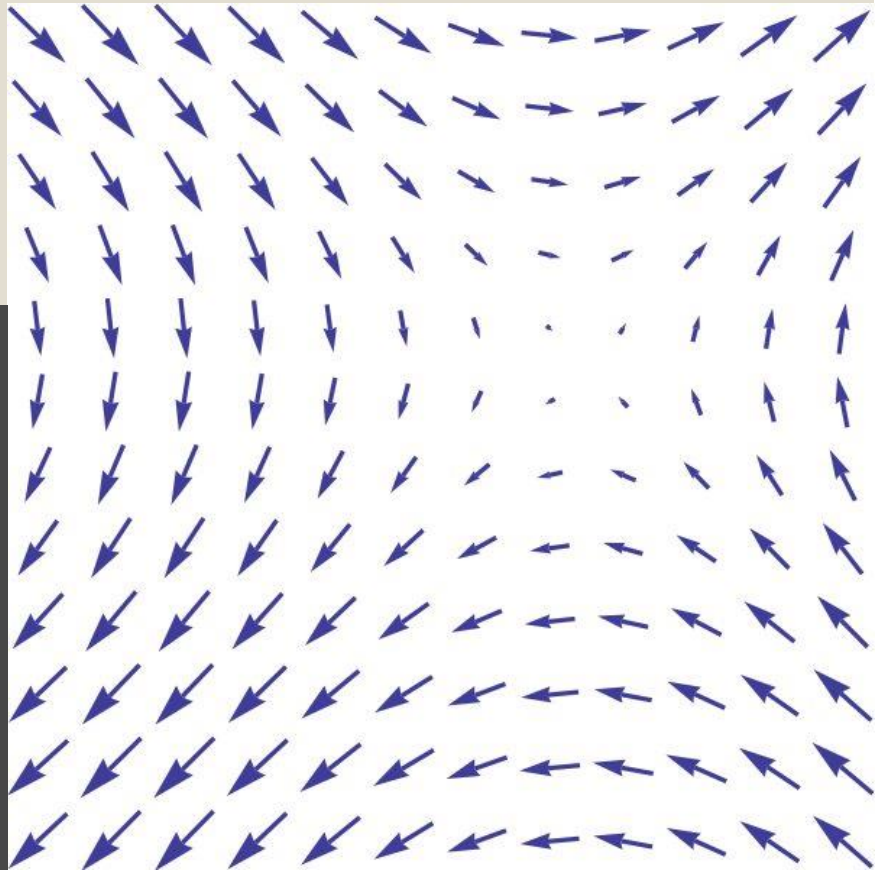
- V bodech zobrazen glyf (objekt)
  - Typ glyfu, jeho velikost, barva, a orientace závisí na velikosti a směru vektoru
- Glyfem často šipka
  - Nejjednodušší případ = šipka má konstantní velikost a je vykreslena jednou barvou



VEKTOROVÁ POLE: GLYFY

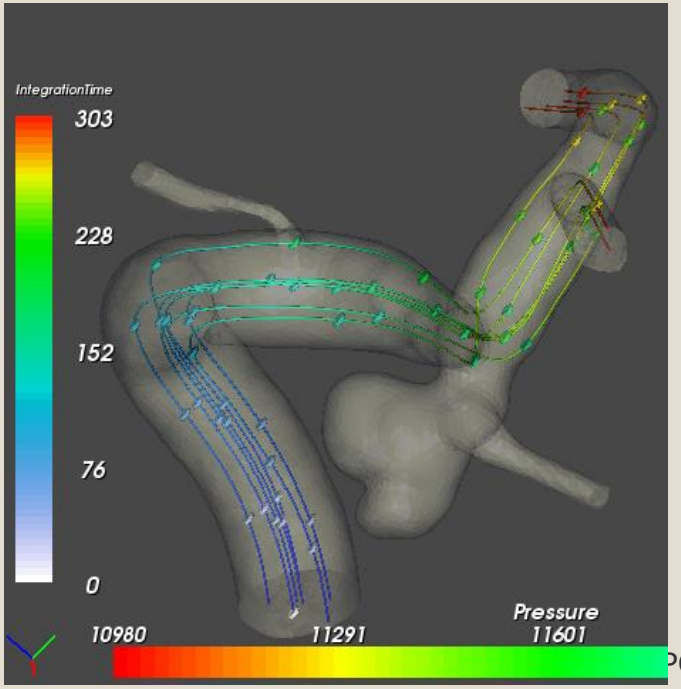
- Složitější (ale častější) = velikost šipky je funkcí velikosti vektoru
  - Funkce nemusí být lineární
    - Vhodné zejména pro pole s velkou variabilitou velikostí
- Alternativa: barva odpovídá velikosti vektoru

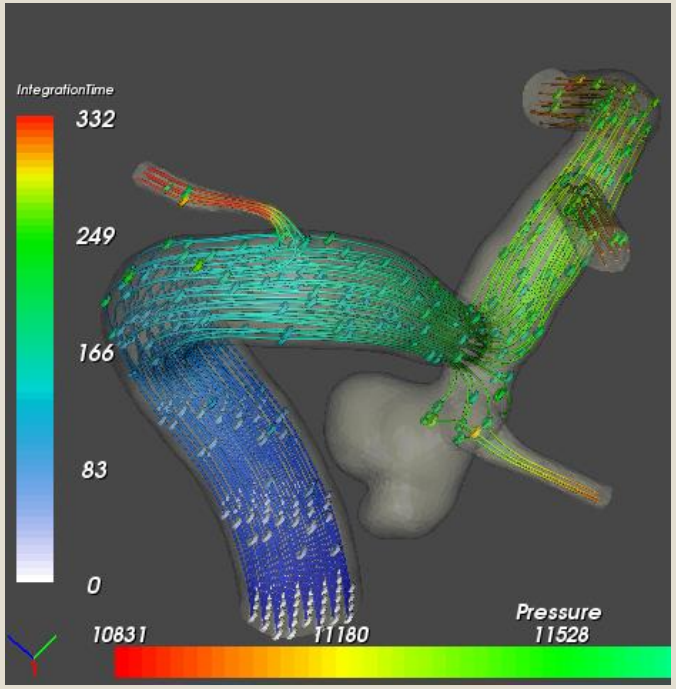




VEKTOROVÁ POLE: STREAMLINES

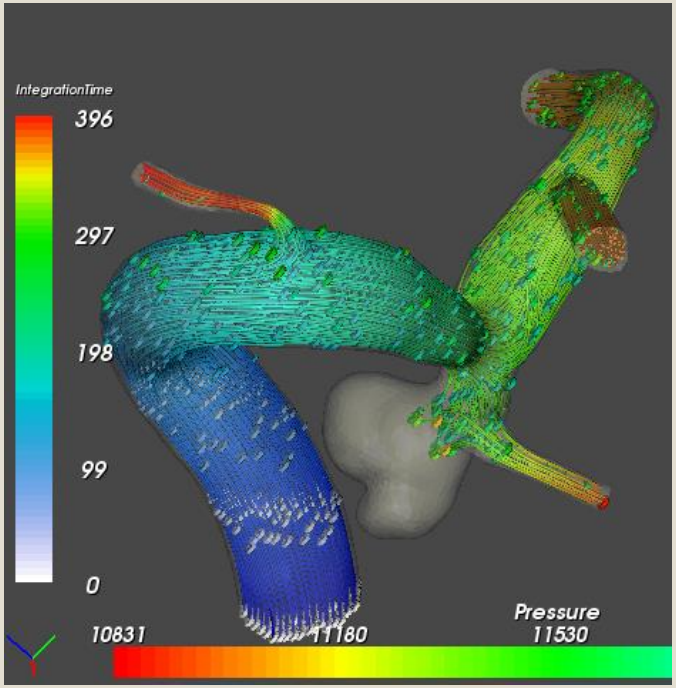
- Křivky trajektorií částic ve vektorovém poli
  - Pole nesmí být časově proměnlivé
- Nutno specifikovat startovní body
  - Málo bodů = nedostatečně vystihuje chování

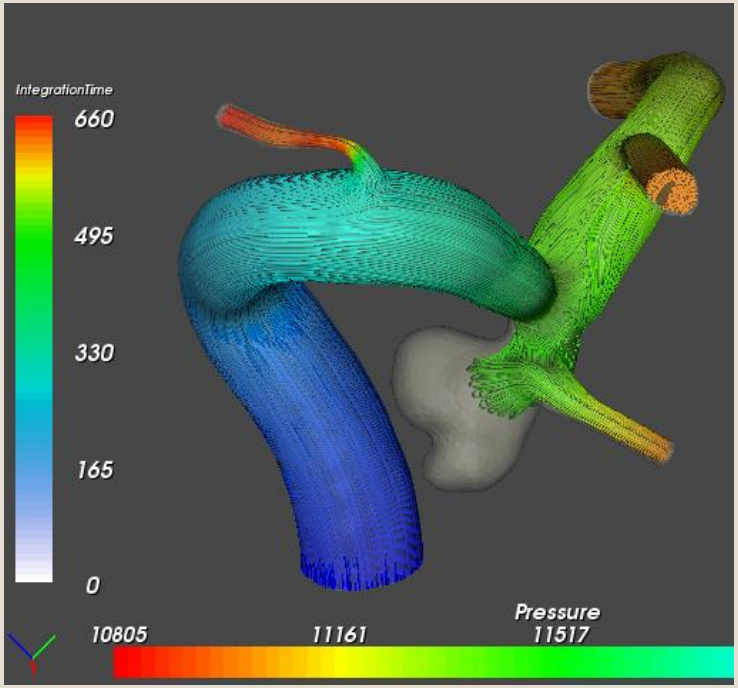




VEKTOROVÁ POLE: STREAMLINES

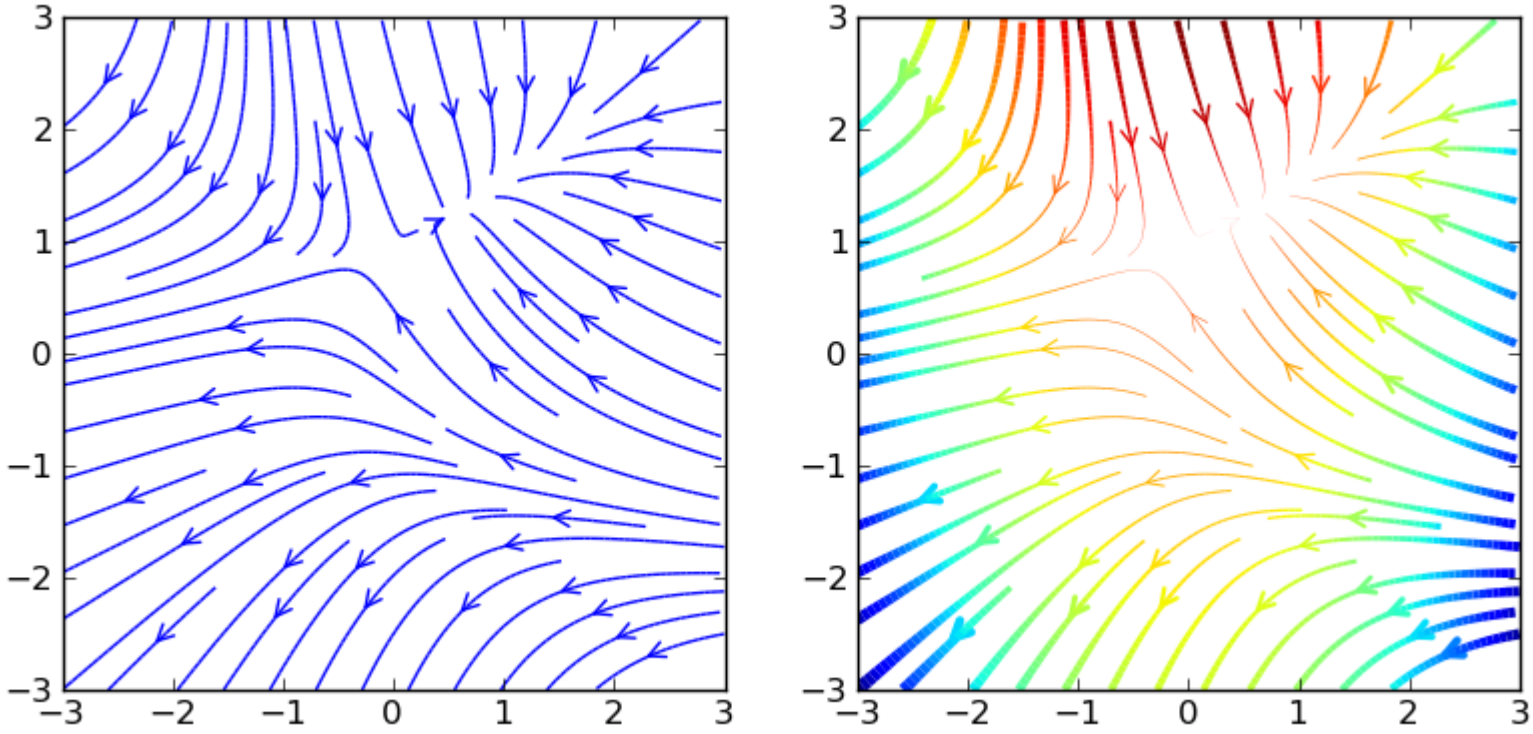
- Mnoho bodů = vysoké časové i paměťové nároky, navíc může být nepřehledné pro uživatele





VEKTOROVÁ POLE: STREAMLINES

- Výpočet streamlines
  - Algoritmus postupuje ze startovacího bodu a hledá, kam se z tohoto bodu dostane částice unášena polem
    - Náročný výpočet, vyžaduje znalost buněk – více viz KIV/VD



VEKTOROVÁ POLE: STREAMLINES

- Možnosti zobrazení streamlines
  - Staticky = vykreslení lomených čar (trajektorií)
  - Dynamicky = vykreslení částic pohybujících se po „streamlines“ v závislosti na čase
    - Částice reprezentována glyfem
    - **DEMO**

VIZUALIZAČNÍ NÁSTROJE

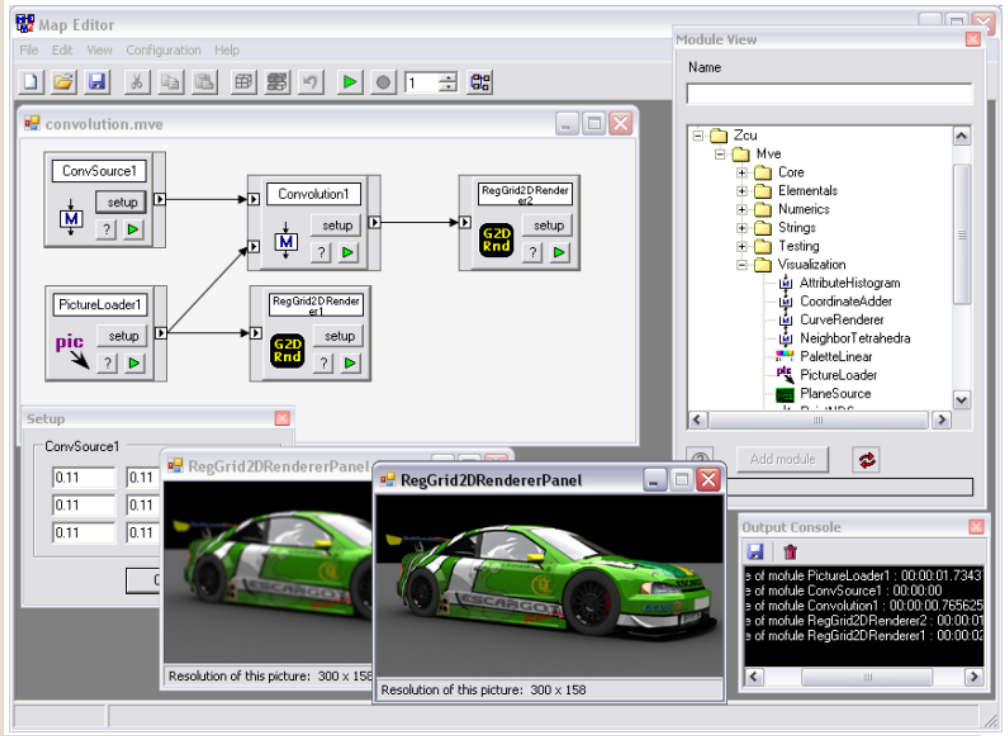
- Vizualizace vědeckých dat aplikačně závislá
- Nástroje musí být použitelné lajkami
- Vizualizační nástroje mají společnou myšlenku:
  - Nástroj obsahuje sadu předdefinovaných modulů
  - Modul má na vstupu data, která zpracuje podle svého nastavení a pošle na výstup
  - Uživatel vybere moduly, které potřebuje, a propojí je do výsledné aplikace
    - V některých případech bez napsání jediné řádky kódu

VIZUALIZAČNÍ NÁSTROJE

- Dostupné vizualizační nástroje
  - Komerční
    - např. ANSYS, Matlab, 3D Doctor, TrueGrid
  - Volně dostupné
    - např. MVE2, MAF, ParaView, OsiriX, OpenDX (dřívější IBM Data Explorer), 3D Slicer, MayaVi Data Explorer

VIZUALIZAČNÍ NÁSTROJE: MVE2

- Vytvořeno na ZČU pro .NET
  - http://graphics.zcu.cz/Projects/MVE-2
- Uživatel "programuje" vizuálně poskládáním dostupných modulů
- Počet modulů omezen



VIZUALIZAČNÍ NÁSTROJE: MAF

- MAF = Multimod Application Framework
  - http://www.openmaf.org
- C++ API pro rapidní vývoj vizualizačních aplikací
  - crossplatform (Windows, Linux)
- Založeno na VTK (Kitware Inc.)
- Integruje další specializační knihovny
  - wxWidget, ITK, Crypto++, XerxesC, Curl
- Lze snadno poskládat kompletní aplikaci
  - Současná práce s různorodými daty

VIZUALIZAČNÍ NÁSTROJE: MAF

OrthoSlice

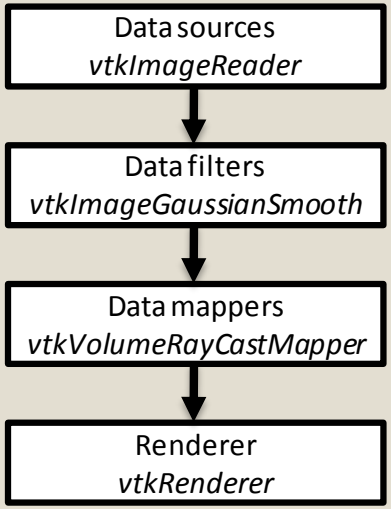
Surface

DRR

RX-CT

VIZUALIZAČNÍ NÁSTROJE: VTK

- http://www.vtk.org/
- Open source C++ knihovna tříd
- Existuje wrapper pro Javu
- Klíčové prvky:
  - vtkPanel = Java swing GUI panel pro vykreslení a manipulaci
    - Základní interakce automaticky k dispozici
  - vtkRenderer = "grafický kontext" pro vtkPanel
    - Instanci lze získat metodou getRenderer() z vtkPanel
    - Obsahuje informaci, odkud se na scénu díváme
  - vtkActor = říká rendereru, jak geometrii jemu předanou zobrazit (drátěný model, poloprůhledný model, …)



VIZUALIZAČNÍ NÁSTROJE: VTK

- vtkMapper = převádí data do podoby připravené k vizualizaci (např. provádí mapování hodnot na barvy)
  - Např. vtkVolumeRayCastMapper
    - Provádí kompozici řezů dle zadaného operátoru (např. MIP)
- vtkSource = zdroj dat
  - Např. vtkImageReader
    - Načítá objemová data ze souboru na disku
    - Data musí být uložena v raw formátu

VIZUALIZAČNÍ NÁSTROJE: VTK

- Hlavní třída musí obsahovat kód:
  - Není v žádné metodě, ale přímo v těle třídy

```
// ----------------------------------------------------------------
// Load VTK library and print which library was not properly loaded
static {
   if (!vtkNativeLibrary.LoadAllNativeLibraries()) {
      for (vtkNativeLibrary lib : vtkNativeLibrary.values()) {
        if (!lib.IsLoaded()) {
          System.out.println(lib.GetLibraryName() + " not loaded");
        }
      }
   }
   vtkNativeLibrary.DisableOutputWindow(null);
}
```

VIZUALIZAČNÍ NÁSTROJE: VTK

```
// ------------------------------------------------------------------
public SimpleVTK() {
    super(new BorderLayout());

    // build VTK Pipeline
    vtkConeSource cone = new vtkConeSource();
    cone.SetResolution(8);

    vtkPolyDataMapper coneMapper = new vtkPolyDataMapper();
    coneMapper.SetInputConnection(cone.GetOutputPort());

    vtkActor coneActor = new vtkActor();
    coneActor.SetMapper(coneMapper);

    renWin = new vtkPanel();
    renWin.GetRenderer().AddActor(coneActor);

    // Add Java UI components
    exitButton = new JButton("Exit");
    exitButton.addActionListener(this);

    add(renWin, BorderLayout.CENTER);
    add(exitButton, BorderLayout.SOUTH);
}

/** An ActionListener that listens to the button. */
public void actionPerformed(ActionEvent e) {
    if (e.getSource().equals(exitButton)) {
        System.exit(0);
    }
}
```

```
public static void main(String s[]) {
        JFrame frame = new JFrame("SimpleVTK");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().setLayout(new BorderLayout());
        frame.getContentPane().add(new SimpleVTK(), BorderLayout.CENTER);
        frame.setSize(400, 400);
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
}
```

KONEC

- Příště: Základy 3D grafiky