INTERAKTIVNÍ 2D VEKTOROVÁ GRAFIKA

Animace

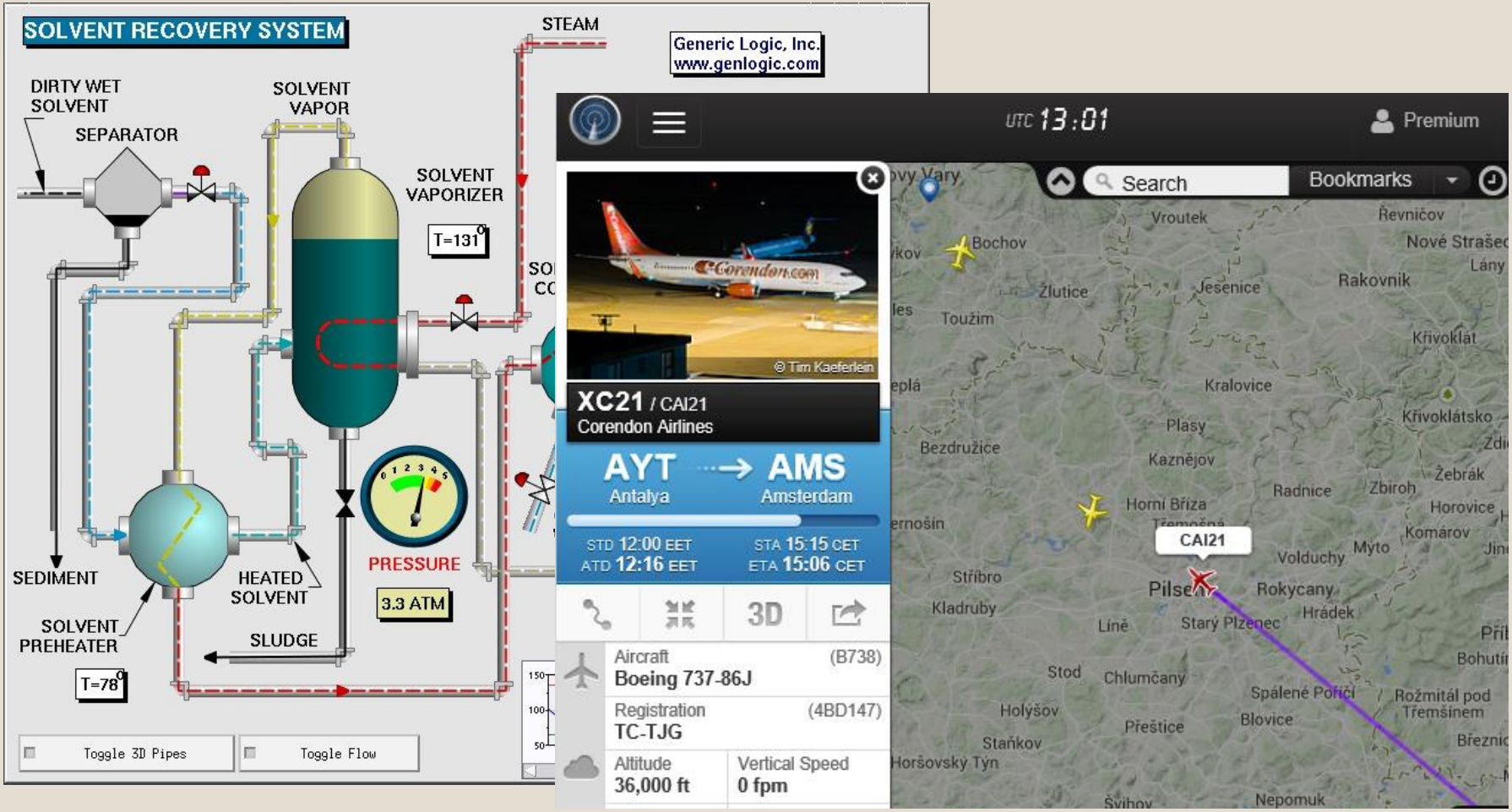
Událostní programování

Aplikační geometrie

ANIMACE VEKTOROVÉ GRAFIKY

- Animace = obraz měnící se v čase
- Význam zejména pro:
  - Snadnější vysvětlení dynamických jevů
  - Upoutání pozornosti (např. marketing)
  - Oživení prezentace
  - Jednoduché hry
  - …

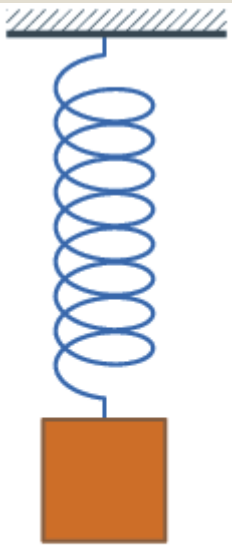
ANIMACE VEKTOROVÉ GRAFIKY



ANIMACE VEKTOROVÉ GRAFIKY

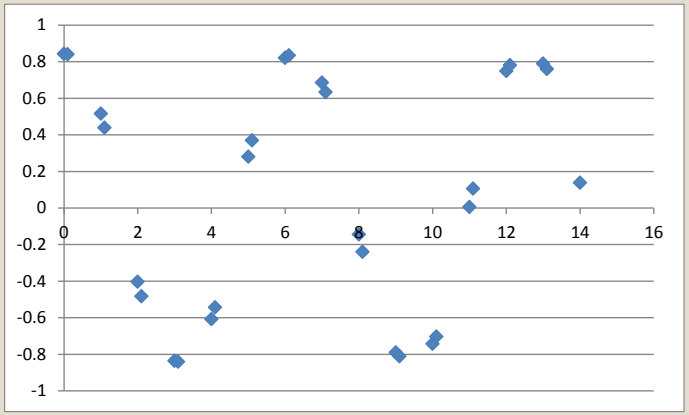
ANIMACE VEKTOROVÉ GRAFIKY

- Animace probíhá v čase t = 0 až T sekund
- Libovolný parametr vektorového elementu může být časově závislý
- Funkce času může mít fyzikální podstatu
  - Např. netlumené kmitání obdélníku na pružině znamená, že jeho y souřadnice se mění podle funkce: $A \cdot \sin(\omega \cdot t + \varphi)$
- Stanovení matematické funkce je často složité případně pro animační účely tato funkce je příliš složitá
- Řešení: aproximace funkce



ANIMACE VEKTOROVÉ GRAFIKY

- Aproximace = výpočet funkčních hodnot funkce nejbližší naší funkci resp. datům
  - Aproximace velmi často po částech
- Pro data užijeme uzlové body v t_i
- V uzlových bodech známe přesnou hodnotu
  - Např. změřeno nebo vhodně zvoleno

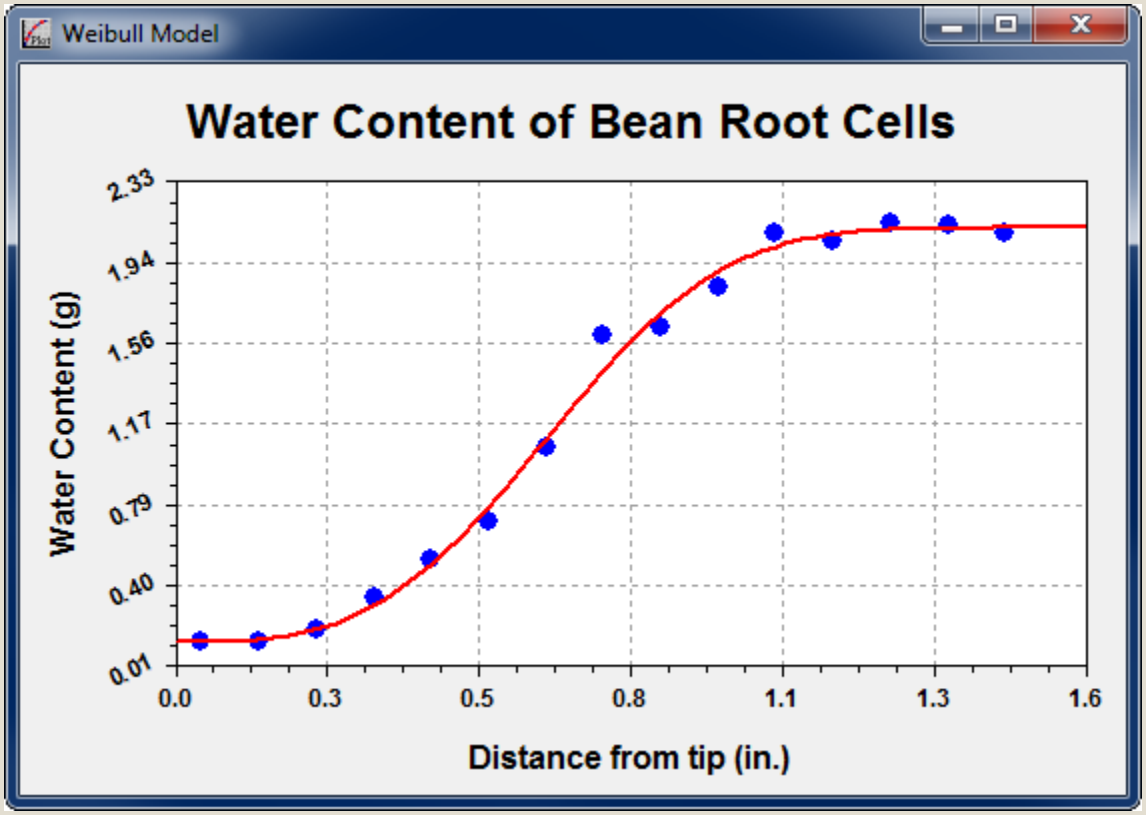


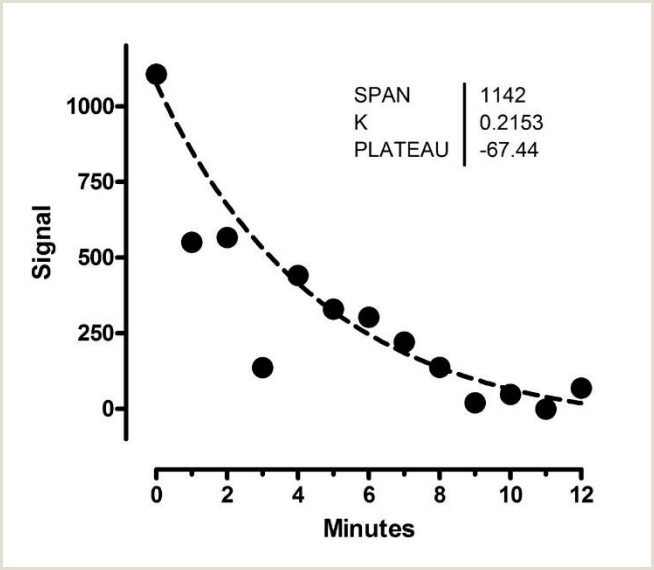
ANIMACE VEKTOROVÉ GRAFIKY

- Hodnotu v obecném čase t aproximací
- Aproximace (obecná)
  - Nemusí v uzlových bodech vrátit zadanou hodnotu
  - Nemá příliš význam pro animaci
  - Vyhlazení závislostí
    - Eliminace nahodilých chyb měření
  - Empirické modely regresního typu
    - Např. metoda nejmenších čtverců

ANIMACE VEKTOROVÉ GRAFIKY

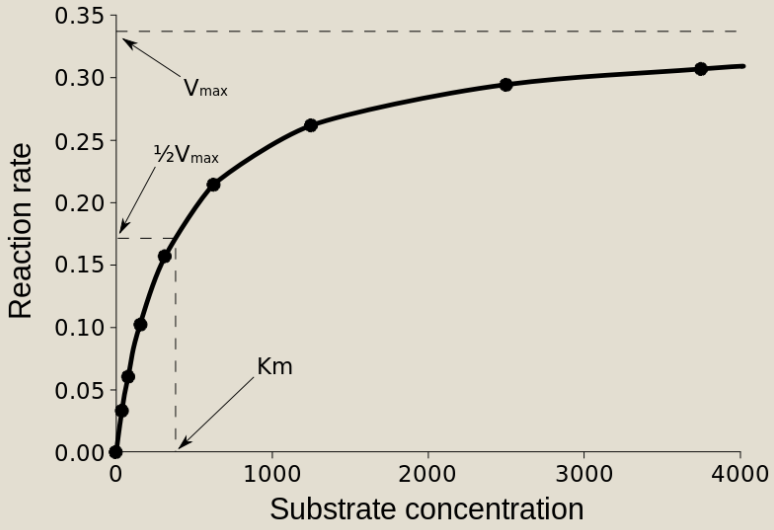
- Aproximace



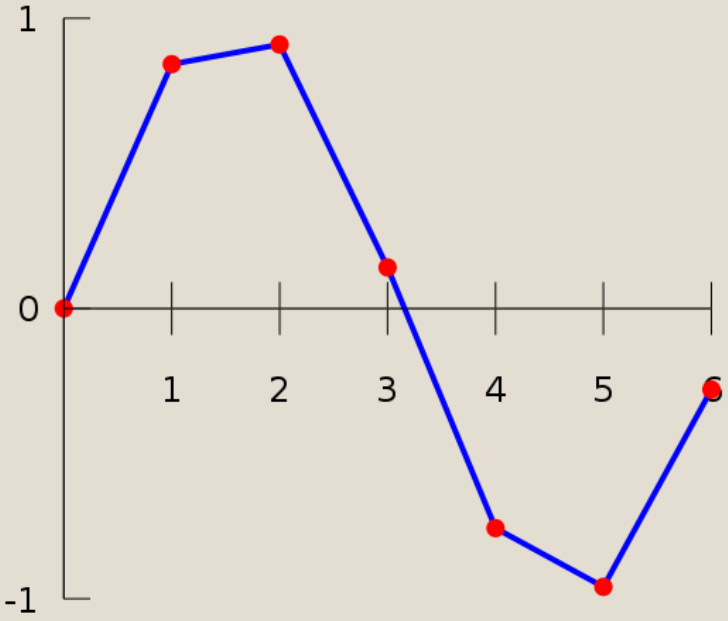


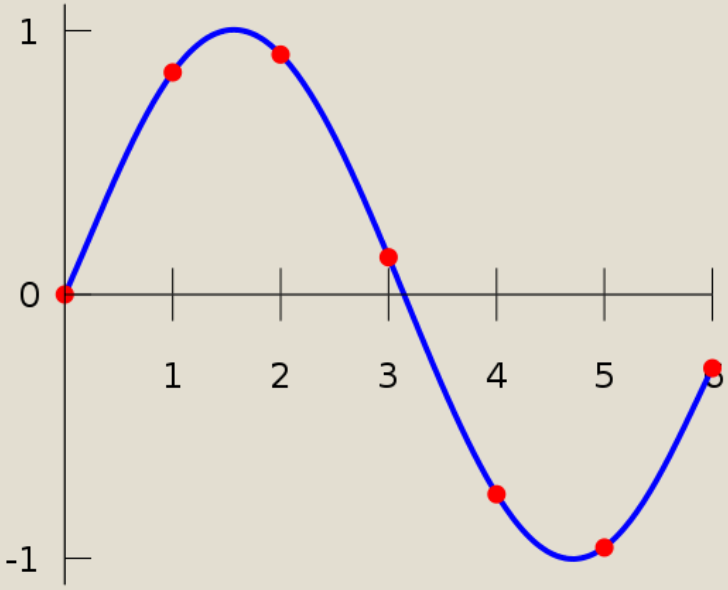
ANIMACE VEKTOROVÉ GRAFIKY

- Interpolace
  - Speciální metoda aproximace funkce
  - V uzlových bodech vrátí zadanou hodnotu
  - Nejčastěji lineární nebo kubická interpolace
    - Lokálně na sousedních uzlech



ANIMACE VEKTOROVÉ GRAFIKY



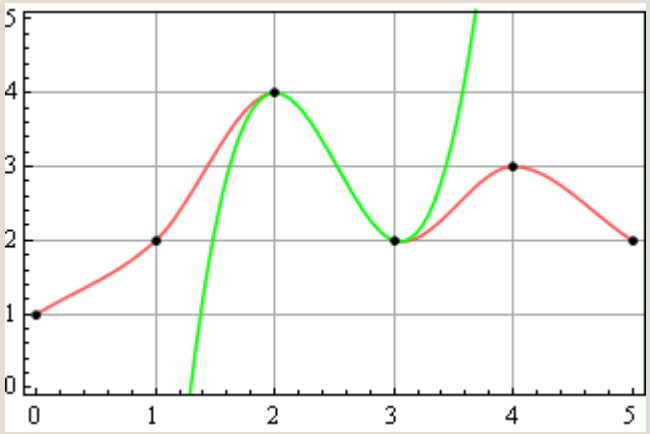


ANIMACE VEKTOROVÉ GRAFIKY

- Uzlové body v časech t_i
  - Jedná se o tzv. klíčové snímky nebo někdy též o časovou událost (leží na časové ose)
  - Někdy též časová událost
- Lineární interpolace
  - Pro stanovení hodnoty parametru v $t \in < t_i, t_{i+1} >$ vyžaduje pouze znalost hodnot v časech t_i, t_{i+1}
  - $h(t) = (1 - t) \cdot h_i + t \cdot h_{i+1}$ pro $t \in < 0,1 >$
  - Lze jen pro jednoduché věci
  - Nelze dost dobře již např. pro pohyb po kružnici

ANIMACE VEKTOROVÉ GRAFIKY

- Kubická interpolace
  - Pro stanovení hodnoty parametru v $t \in < t_i, t_{i+1} >$ vyžadována znalost rychlosti růstu hodnot v časech t_i, t_{i+1}, tj. derivace $h'(t_i), h'(t_{i+1})$
  - $h(t) = a \cdot t^3 + b \cdot t^2 + c \cdot t + d$ pro $t \in < 0,1 >$
    - Konstanty z požadavků na interpolaci



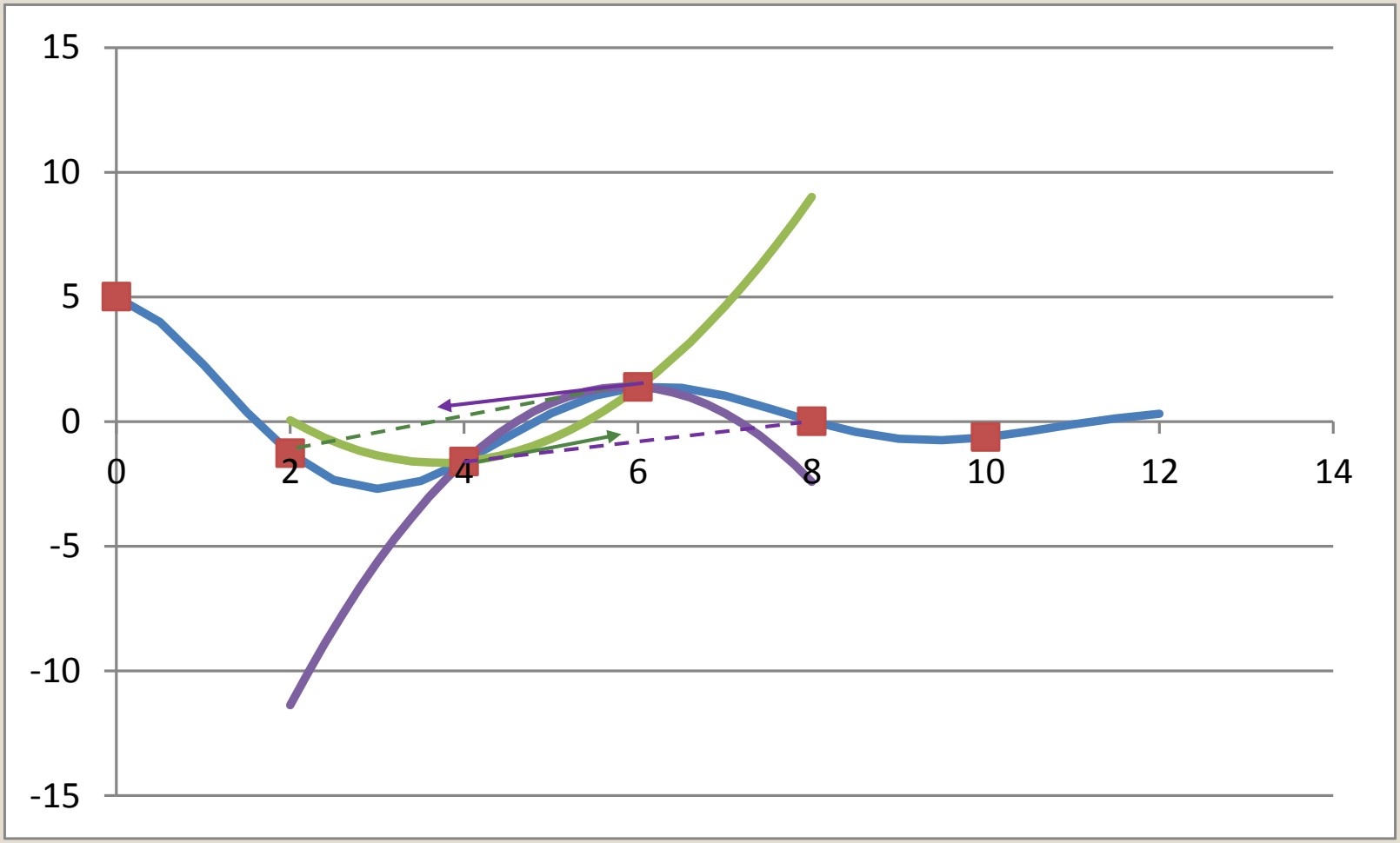
ANIMACE VEKTOROVÉ GRAFIKY

- Rychlost v čase $t = t_i$ lze rovněž odvodit z dat
  - $\frac{h_{i+1}-h_{i-1}}{t_{i+1}-t_{i-1}}$
    - Jednoduché
    - Spíše pro "uniformní" dělení
  - $\frac{h_{i+1}-h_i}{t_{i+1}-t_i} \cdot w + \frac{h_i-h_{i-1}}{t_i-t_{i-1}} \cdot (1-w)$
    - $w = \frac{t_i-t_{i-1}}{t_{i+1}-t_{i-1}}$
    - Vážené jednostranné derivace
    - Složitější, ale lepší

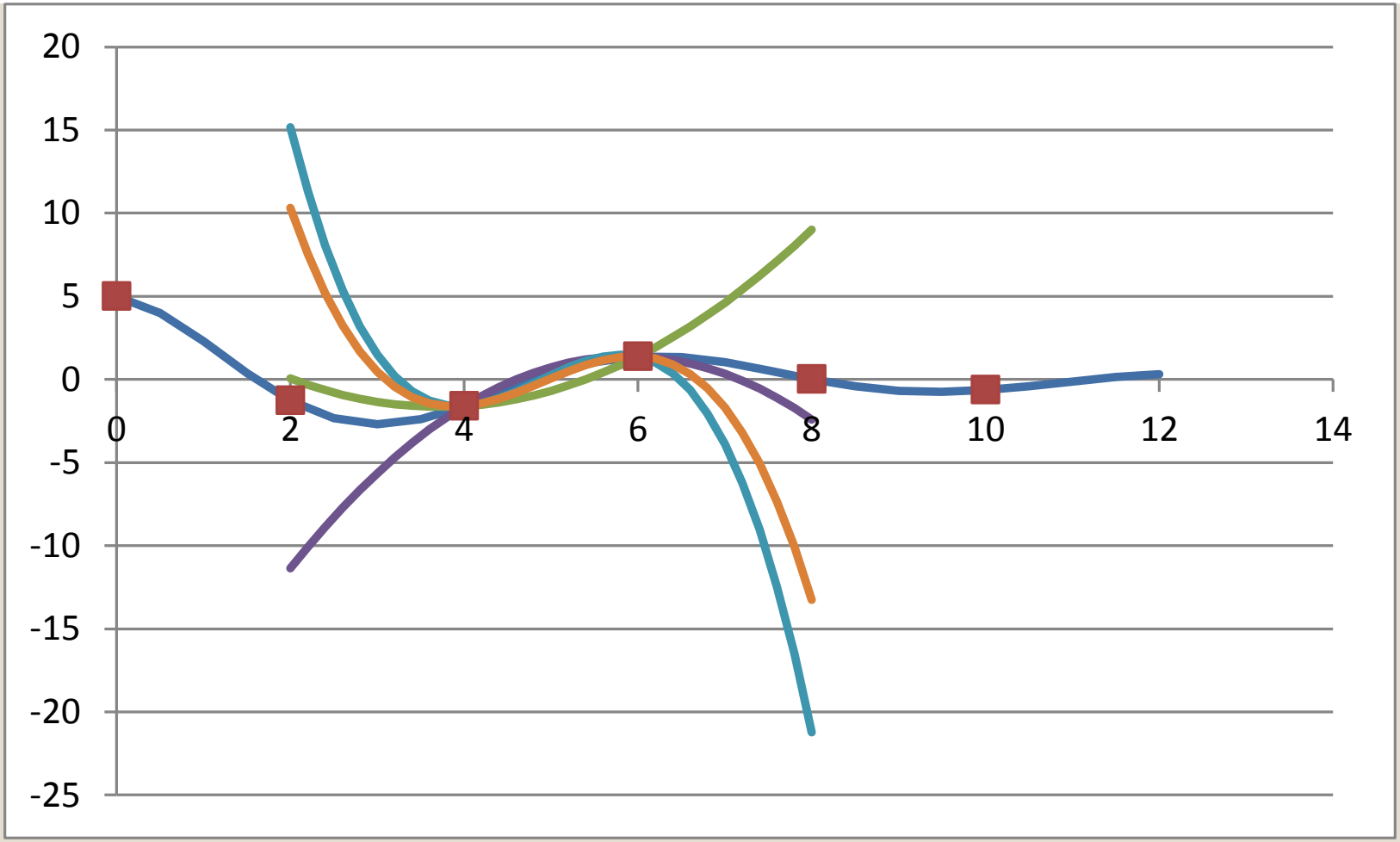
ANIMACE VEKTOROVÉ GRAFIKY

- Interpolace vyššího stupně
  - Obvykle se nepoužívá
    - Nepředvídatelné chování
- Kvadratická interpolace
  - Obvykle se nepoužívá
    - Co je třetím parametrem?

ANIMACE VEKTOROVÉ GRAFIKY



ANIMACE VEKTOROVÉ GRAFIKY



ANIMACE VEKTOROVÉ GRAFIKY

- Mám rozmyšlené klíčové snímky, jakým způsobem se mají hodnoty mezi klíčovými snímky interpolovat, jak animaci udělat?
- Různé možné přístupy:
  - Řešení č. 1 – nechci nic programovat
  - Řešení č. 2 – animace řídí grafická knihovna
  - Řešení č. 3 – kompletně vše si řídím sám

ŘEŠENÍ Č. 1

- Nejjednodušší přístup (pro tvůrce)
- Vše nakreslím v nějakém WYSIWYG vektorovém editoru a uložím do souboru vektorového formátu
- Uživatel pak bude tento soubor zobrazovat v nějaké běžně užívané aplikaci (např. webový prohlížeč, prohlížeč obrázků,…)

ŘEŠENÍ Č. 1

- Problémy:
  - Mnoho vektorových kreslítek animace nepodporuje
  - Většina dostupných vektorových formátů uložení popisu animace nepodporuje
    - Výjimka: SVG a XAML
  - Běžné aplikace mají problém zobrazit animace
    - Např. IE 10 a 11 zobrazí SVG staticky, Mozilla Firefox 4 vůbec nezobrazí XAML
  - Minimální kontrola nad kvalitou vykreslené animace
    - Např. pohyb trvá 5 sekund, ale FPS neovlivním

ŘEŠENÍ Č. 1

- SVG zavádí několik elementů pro animace
  - Uvádějí se jako potomek objektu, který chci animovat
  - `animate` nebo `set` – nastavení skalární hodnoty
    - včetně barvy
  - `animateTransform` – nastavení afinní transformace
  - `animateMotion` – pohyb objektu po křivce

```
<rect …>
  <animateTransform attributeType="xml" attributeName="transform"
  type="rotate" from="0" to="90" begin="0" dur="5s" fill="freeze" />
</rect>
```

ŘEŠENÍ Č. 1

- Společné atributy:
  - `attributeName` – název řízeného atributu (např. stroke)
  - `attributeType` – "xml" nebo "css" určuje, zda řízený atribut je v sadě vlastností objektu nebo jeho stylu
    - není povinný, význam jen v případě kolizí
  - `begin` – čas(y), kdy animace má začít
    - Např. `begin="0s;5s"`
    - Není povinný (výchozí hodnota je 0s)
  - `dur`– doba trvání animace
    - Typicky musí být stanovena
    - Např. `dur="3s"`
  - `end`– čas(y), kdy animace musí skončit
    - Není povinný

ŘEŠENÍ Č. 1

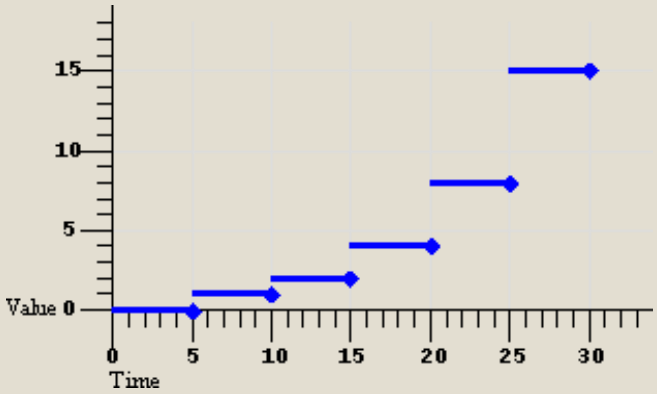
- `repeatCount`– počet opakování animace po zahájení
  - Může být reálné číslo
  - Typicky není používáno
  - Např. `repeatCount="indefinite"`
- repeatDur– celková doba, kterou opakování sežere
  - Alternativa k repeatCount
  - Typicky není používáno
- `fill` – určuje, co se má stát s hodnotou parametru, která byla v průběhu animace měněna, po skončení animace
  - `fill="remove"` – hodnota se nastaví na původní
  - `fill="freeze"` – hodnota se ponechá na poslední
  - Nemusí být specifikováno, výchozí je `remove`

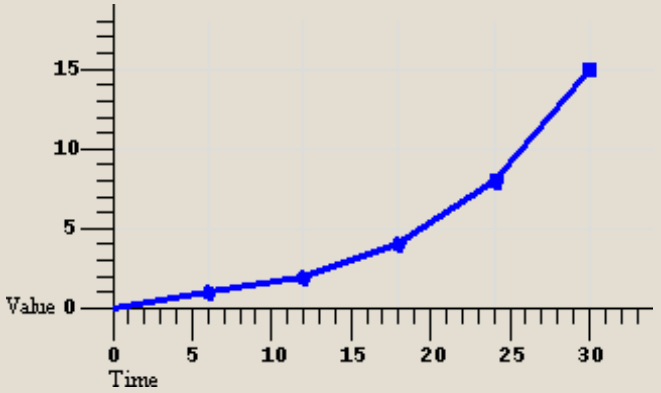
ŘEŠENÍ Č. 1

- SVG má rozsáhlou podporu pro stanovení hodnot parametru a způsobu jeho interpolace
  - Diskrétní, lineární a kubická interpolace
    - Atributy `calcMode, values, from, to`
  - Uniformní i neuniformní rozdělení času
    - Atributy `keyTimes` a keySplines
      - Čas parametrizován na interval 0-1

ŘEŠENÍ Č. 1

```
<animate dur="30s" values="0; 1; 2; 4; 8; 15"
calcMode="discrete" …/>
```

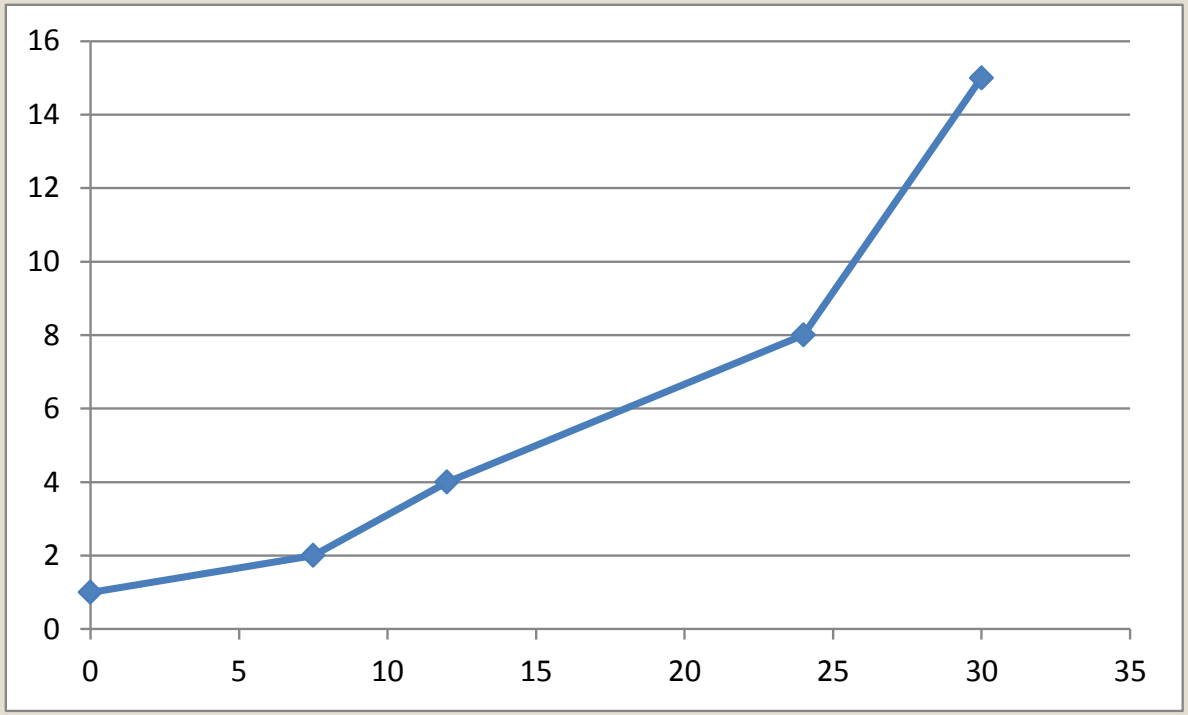





```
<animate dur="30s" values="0; 1; 2; 4; 8; 15"
calcMode="linear" …/>
```

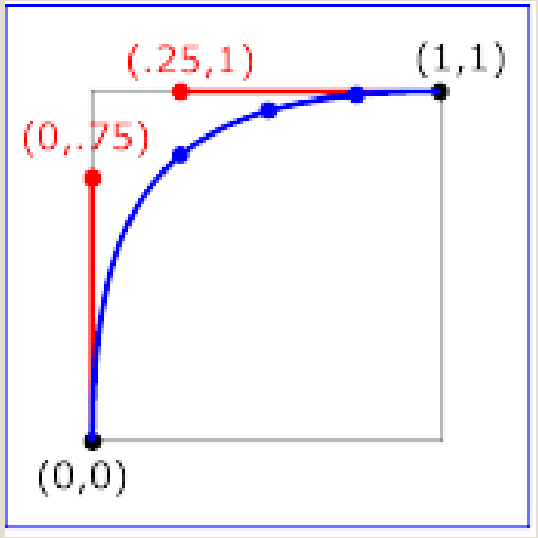
ŘEŠENÍ Č. 1

```
<animate dur="30s" values="0; 1; 2; 4; 8; 15"
keyTimes="0;0.25;0.4;0.7;0.8;1" …/>
```



ŘEŠENÍ Č. 1

```
<animate dur="4s" values="10; 20" keyTimes="0; 1"
     calcMode="spline" keySplines=“0 0.75 0.25 1” />
<animate dur="4s" from="10” to=“20" keyTimes="0; 1"
     calcMode="spline" keySplines=“0 0.75 0.25 1” />
```



ŘEŠENÍ Č. 1

- Element `animateTransform` má atribut `type` pro nastavení, co měním
  - Např. `type="rotate"`
- Element `animateMotion` má atribut `path` pro nastavení cesty pro pohyb objektu
- XAML poskytuje obdobný systém pro animace
- Zásadní rozdíly:
  - Chování řízeno užitým elementem namísto atributů
  - Silná typová kontrola animované hodnoty
    - Odlišný element pro animaci Double a Int
  - Odlišný způsob zahájení animace

ŘEŠENÍ Č. 1

```
<Rectangle Name="MyRectangle" … >
  <Rectangle.Triggers>
      <EventTrigger RoutedEvent="Rectangle.Loaded">
       <BeginStoryboard>
        <Storyboard>
          <DoubleAnimation
            Storyboard.TargetName="MyRectangle"
            Storyboard.TargetProperty="Opacity"
            From="1.0" To="0.0" Duration="0:0:5"
            AutoReverse="True" RepeatBehavior="Forever" />
        </Storyboard>
      </BeginStoryboard>
    </EventTrigger>
  </Rectangle.Triggers>
</Rectangle>
```

ŘEŠENÍ Č. 1

- Jak má animace probíhat říkají elementy s kořenem slova Animation:
  - Např. `<DoubleAnimation From="1.0" To="0.0" Duration="0:0:5" AutoReverse="True" RepeatBehavior="Forever"/>`
- Předpona elementu = datový typ hodnoty, kterou bude animace měnit
  - Např. `Double`, `Int32`, `Color`, …
- Přípona elementu určuje interpolační chování
- Žádná přípona:
  - Uniformní dělení času
  - Lineární interpolace

ŘEŠENÍ Č. 1

- Přípona `UsingKeyFrames`
  - Např. `DoubleAnimationUsingKeyFrames`
  - Vyžaduje specifikování klíčových snímků
    - Může být diskrétní, lineární a kubická interpolace
      - Lze kombinovat
  - Může být neuniformní dělení času

```
<DoubleAnimationUsingKeyFrames Duration="0:0:10">
        <LinearDoubleKeyFrame Value="0" KeyTime="0:0:0" />
        <LinearDoubleKeyFrame Value="350" KeyTime="0:0:2" />
        <LinearDoubleKeyFrame Value="50" KeyTime="0:0:7" />
        <LinearDoubleKeyFrame Value="200" KeyTime="0:0:8" />
</DoubleAnimationUsingKeyFrames>
```

ŘEŠENÍ Č. 1

- Přípona `UsingKeyFrames`
  - Např. `DoubleAnimationUsingKeyFrames`
  - Vyžaduje specifikování klíčových snímků
    - Může být diskrétní, lineární a kubická interpolace
      - Lze kombinovat
  - Může být neuniformní dělení času

```
<DoubleAnimationUsingKeyFrames Duration="0:0:15">
   <DiscreteDoubleKeyFrame Value="500" KeyTime="0:0:7" />
   <LinearDoubleKeyFrame Value="200" KeyTime="0:0:10" />
   <SplineDoubleKeyFrame Value="350" KeyTime="0:0:15"
              KeySpline="0.25,0.5 0.75,1" />
</DoubleAnimationUsingKeyFrames>
```

ŘEŠENÍ Č. 1

- Přípona `UsingPath`
  - Např. `DoubleAnimationUsingPath`
  - Vyžaduje specifikování cesty pro pohyb
    - Předem v resources
- Animace jednotlivých parametrů seskupovány do logických celků, tzv. StoryBoard
  - Např. jedna animace kontroluje změnu X, druhá změnu Y, dohromady je to změna polohy
  - StoryBoard také říká, zda se má animace opakovat
    - ```
      <Storyboard RepeatBehavior="Forever">
      …
      </Storyboard>
      ```
      ```

ŘEŠENÍ Č. 1

- `StoryBoard` musí být spouštěn
  - Element `BeginStoryBoard`
- `BeginStoryBoard` je umístěn v reakci na událost, která má přehrávání odstartovat
  - Objekty jsou zavedeny do paměti
  - Uplynul nějaký čas od něčeho (např. od zavedení)
  - Uživatel kliknul na něco
  - Atribut (např. barva) něčeho se změnil

ŘEŠENÍ Č. 1

```
<Rectangle Name="MyRectangle" … >
  <Rectangle.Triggers>
      <EventTrigger RoutedEvent="Rectangle.Loaded">
       <BeginStoryboard>
        <Storyboard>
          <DoubleAnimation
            Storyboard.TargetName="MyRectangle"
            Storyboard.TargetProperty="Opacity"
            From="1.0" To="0.0" Duration="0:0:5"
            AutoReverse="True" RepeatBehavior="Forever" />
        </Storyboard>
      </BeginStoryboard>
    </EventTrigger>
  </Rectangle.Triggers>
</Rectangle>
```

ŘEŠENÍ Č. 2

- Nakreslím obrázek ve WYSIWIG editoru, uložím do vektorového formátu
- Ve své aplikaci načtu a vektorové objekty rozhýbám prostřednictvím funkcionality mé grafické knihovny
- Problémy:
  - Může být obtížné získat objekty ze souboru
    - Někdy obrázek jen jako celek
    - Bez problémů jen SVG a XAML
  - Grafické knihovny nativně animace nepodporují
    - Výjimka WPF

ŘEŠENÍ Č. 2

- WPF poskytuje stejně jmenující se třídy jako jsou elementy pro animaci v XAML

```
var myDoubleAnimation = new DoubleAnimation();
myDoubleAnimation.From = 1.0; myDoubleAnimation.To = 0.0;
…
myStoryboard = new Storyboard();
myStoryboard.Children.Add(myDoubleAnimation);
…
myStoryboard.Begin(this)
```

ŘEŠENÍ Č. 3

- Animaci si kompletně řídím programově sám
- Objekty mohou být načteny ze souboru
- Univerzální způsob
- Problémy:
  - Není až tak jednoduché
  - Animace nepřenosná bez aplikace

UDÁLOSTNÍ PROGRAMOVÁNÍ

- Moderní OS umožňuje běh více aplikací
- Moderní OS zajišťují správný chod systému
- Aplikace nepřebírá kontrolu nad hardwarem
- Aplikace se svou činností vzájemně ovlivňují
- Aplikace řízena událostmi
  - Obsah okna určen k překreslení
  - Uživatel stiskl klávesu
  - Uživatel pohnul myší
  - Počítač se vypíná
  - …

UDÁLOSTNÍ PROGRAMOVÁNÍ

- Aplikace události od OS přijímá ve smyčce
- „Program neodpovídá“, když aplikace nezvládá reagovat na události
  - Aplikace provádí něco časově náročného (např. čeká na zahájení animace), aniž by události zpracovávala
  - Aplikace uvázla (deadlock)

UDÁLOSTNÍ PROGRAMOVÁNÍ

- Ukázka chybné aplikace (v pseudokódu):

```
startTime = System.getCurrentTimeInMs();
endTime = startTime + Animation.Duration;
curTime = startTime;
while (curTime < endTime) {
  Animation.DoStep(curTime – startTime);
    //Změní hodnotu animovaného parametru dle aktuálního času
  curTime = System.getCurrentTimeInMs();
}
```

- **Proč chybné?**

UDÁLOSTNÍ PROGRAMOVÁNÍ

- Ukázka lepší aplikace (v pseudokódu):

```
startTime = System.getCurrentTimeInMs();
endTime = startTime + Animation.Duration;
curTime = startTime;
while (curTime < endTime) {
  Animation.DoStep(curTime – startTime);
    //Změní hodnotu animovaného parametru dle aktuálního času
  DoEvents(); //Zpracování systémových událostí
  curTime = System.getCurrentTimeInMs();
}
```

UDÁLOSTNÍ PROGRAMOVÁNÍ

- Problémy lepší aplikace:
  - Zpracovávání systémových událostí nemusí být nativně podporováno v standardních knihovnách použitého programovacího jazyka
    - Platformová závislost
  - Vytěžuje hodně procesor
    - Lze vyřešit „uspáním“ aplikace na nějaký čas
- „Uspání“ musí být krátké
  - Lidské oko nezaregistruje „blikání“, pokud k přepínání dochází rychleji než cca 60 Hz

UDÁLOSTNÍ PROGRAMOVÁNÍ

- Řešení pomocí systémového časovače
  - Aplikace zaregistruje u OS svůj časovač
    - Prostřednictvím standartní knihovny
    - Např. třída `java.swing.Timer`
  - Časovač má nastaveno, za jak dlouho dojde k aktivaci
  - Při aktivaci OS zašle systémovou událost aplikaci
  - Jádro aplikace (implementováno v standardní knihovně) zavoláním obslužnou metodu
  - Aplikace v metodě provede jeden krok animace

UDÁLOSTNÍ PROGRAMOVÁNÍ

- Řešení pomocí systémového časovače

```
int delay = 1000; //milliseconds
ActionListener taskPerformer = new ActionListener() {
        public void actionPerformed(ActionEvent evt) {
                //Proveď jeden krok animace
        }
};

Timer myTimer = new Timer(delay, taskPerformer);
myTimer.setRepeats(true);
myTimer.start();
```

UDÁLOSTNÍ PROGRAMOVÁNÍ

- Problémy:
  - Obvykle není garantováno, že se obslužná metoda skutečně zavolá za stanovený čas
    - Zpoždění díky zpracovávání jiných událostí
  - Počet systémových časovačů omezen
- Řešení pomocí vláken
  - Aplikace provádí animaci ve vlákně
    - Programování ve vláknech náplní KIV/PGS
  - Zpoždění minimální
  - Obtížnější

PRÁCE S MYŠÍ A KLÁVESNICÍ

- Manipulace s myší nebo klávesnicí vede k systémovým událostem
  - Aplikace události musí nějak zpracovat
  - Typicky se děje v jádře aplikace
  - Jádro volá uživatelskou obslužnou funkci (metodu)
- Součástí události často parametry, např.:
  - Pozice myši
  - Příznak, která tlačítka myši stisknuta
  - Příznak stisknuté CTRL, ALT, SHIFT klávesy
  - …

PRÁCE S MYŠÍ A KLÁVESNICÍ

- Jednoduchá manipulace vede často k několika systémovým událostem
- Např. levý dvojklik způsobí událost:
  - levé tlačítko myši dole
  - levé tlačítko myši nahoře
  - došlo k levému dvojkliku
  - levé tlačítko myši nahoře

PRÁCE S "UDÁLOSTMI" V JAVĚ

- Registrace obslužné metody se provádí prostřednictvím metod komponenty, která má být událostmi ovlivněna
  - Událost stisknutí myši na okně tlačítka registruji u tlačítka, ne u hlavního okna, apod.
- Základní metody:
  - `addActionListener`
    - Události např. stisknutí tlačítka, …
  - `addKeyListener`
    - Události klávesnice
  - `addMouseListener`
    - Události stisků myši

PRÁCE S "UDÁLOSTMI" V JAVĚ

- `addMouseMoveListener`
  - Událost pohybu myši
- `addMouseWheelListener`
  - Událost kolečka myši
- Parametrem metod je tzv. Listener
  - Instance třídy implementující rozhraní definující hlavičky obslužných metod

PRÁCE S "UDÁLOSTMI" V JAVĚ

```
class MyActionListener implements ActionListener {
  public void actionPerformed(ActionEvent e) {
     //obsluha

  }
}

class MainApp {
…
 MyActionListener actListener = new …
 bttn.addActionListener(actListener);
…
}
```

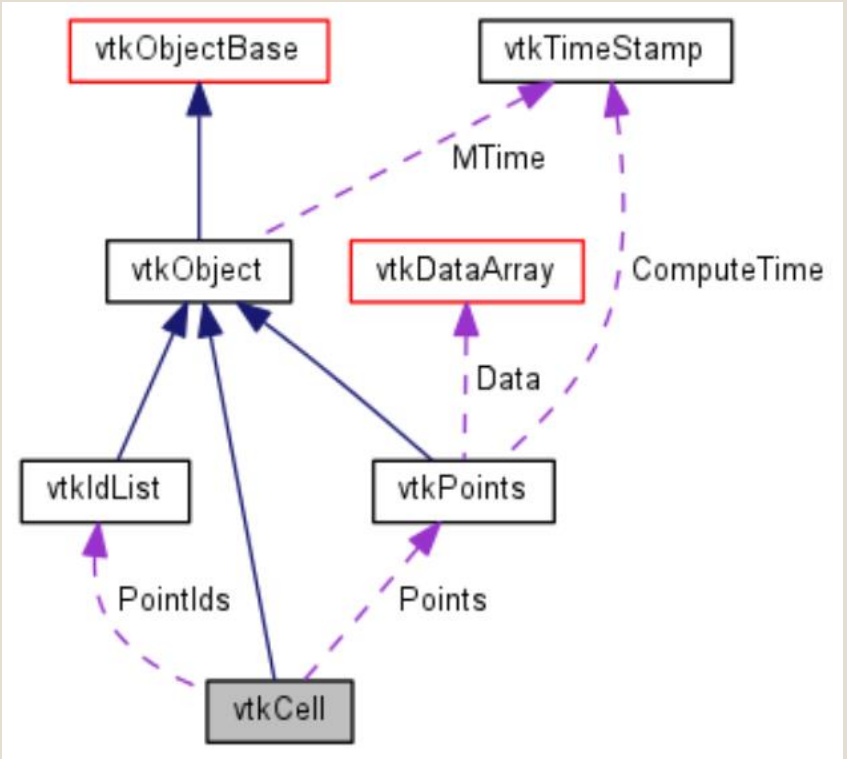
PRÁCE S "UDÁLOSTMI" V JAVĚ

- Od Javy verze X.Y lze používat strukturu anonymních listenerů
  - Není třeba implementovat nové třídy

```
bttn.addActionListener(new ActionListener() { //rozhraní
   public void actionPerformed(ActionEvent e) {
         //obsluha
   }
}
);
```

TROCHU GEOMETRIE

- Klíčové pro popis pozice grafických objektů relativně vůči jiným



TROCHU GEOMETRIE

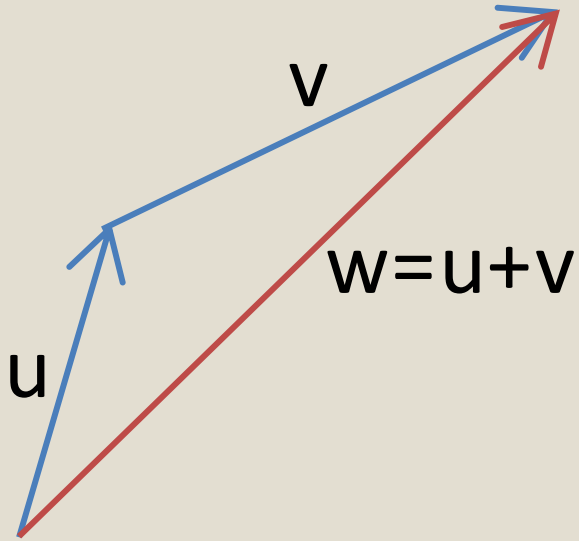
- Využívá se zejména:
  - Vektorový počet
  - Goniometrické funkce
  - Matematický (funkční) popis geometrických tvarů

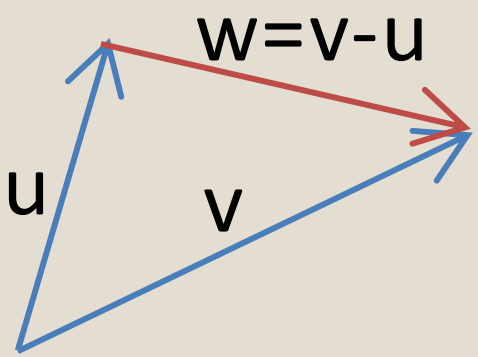
TROCHU GEOMETRIE

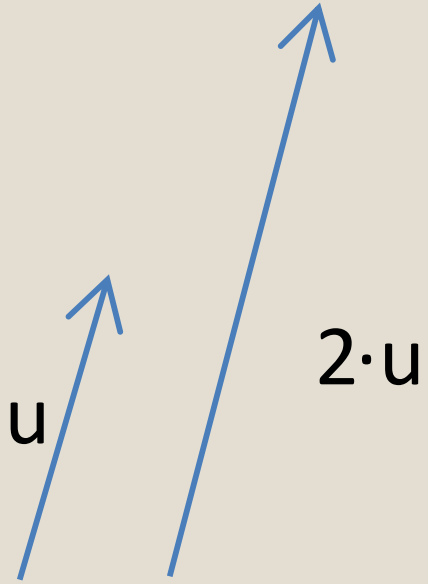
- Vektor z bodu A do bodu B:

$$u(u_x, u_y) = B - A = (b_x - a_x, b_y - a_y)$$

- Sčítání a odčítání vektorů
- Násobení vektoru skalárem







TROCHU GEOMETRIE

- Velikost vektoru:

$$|u| = \sqrt{u_x \cdot u_x + u_y \cdot u_y}$$

- Normalizace vektoru:

$$\hat{u} = \frac{u}{|u|}$$

- Vektor kolmý k vektoru **u**(ux, uy):

$$v(v_x, v_y) = (-u_y, u_x)$$

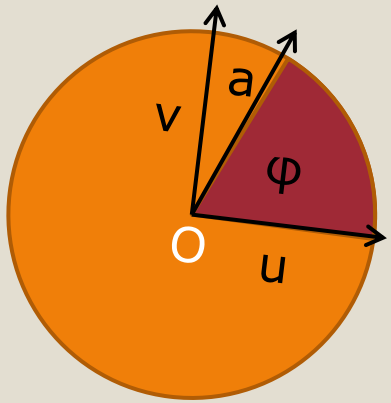
  - Pravotočivý systém
  - **Otázka…**

TROCHU GEOMETRIE

- Vektor a svírající s vektorem u úhel φ:

$$a(a_x, a_y) = (u_x, u_y) \cdot \cos\varphi + (v_x, v_y) \cdot \sin\varphi$$

  - Vektor v je kolmý na u
  - Vektory musí být jednotkové
  - Řešení: $a = (u_x, u_y) \cdot \cos\varphi + (-u_y, u_x) \cdot \sin\varphi$
  - Např. vhodné pro tvorbu pravidelných polygonů nebo hvězd



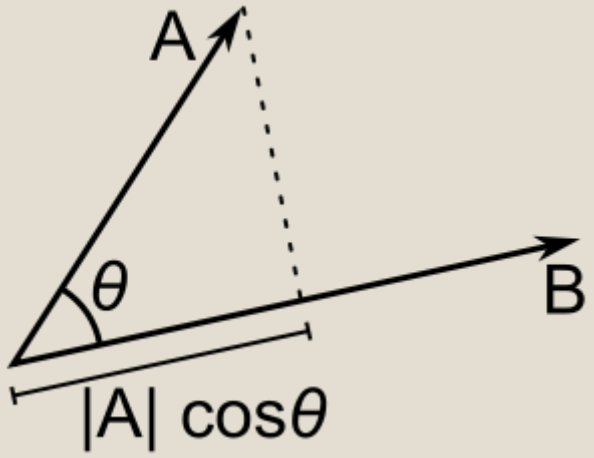
- Parametrizace bodu:

$$P = O + r \cdot \hat{a}$$

  - r je vzdálenost bodu P od O

TROCHU GEOMETRIE

- Skalární součin:
  - $a \cdot b = |a| \cdot |b| \cdot \cos\theta = a_x \cdot b_x + a_y \cdot b_y$
  - Určení úhlu mezi vektory
  - Snažit se vyhnout počítání samotného úhlu!
  - Základ mnoha sofistikovanějších propočtů



TROCHU GEOMETRIE

- Geometrický tvar lze popsat matematicky
  - Různé vyjádření – někdy je vhodnější jedno, jindy jiné

Objekt	Explicitně	Implicitně	Parametricky
Přímka	$y = k \cdot x + q$ resp. $x = k \cdot y + q$	$a \cdot x + b \cdot y + c = 0$ přičemž (a, b) je vektor kolmý	$x = A_x + (B_x - A_x) \cdot t$ $y = A_y + (B_y - A_y) \cdot t$ $t \in (-\infty, \infty)$
Kružnice	$y = \pm\sqrt{r^2 - x^2}$	$x^2 + y^2 = r^2$	$x = r \cdot \cos t$ $y = r \cdot \sin t$ $t \in \langle 0, 2\pi \rangle$

TROCHU GEOMETRIE

- Vzdálenost bodu od přímky
  - Dosadit souřadnice do implicitního tvaru přímky
    - Vektor (a, b) musí být jednotkový
  - Vzdálenost je orientovaná
    - Různé znaménko, pokud bod pod nebo nad přímkou
    - Slouží pro otestování, zda jsem uvnitř či vně objektu
- Např. chci vybrat objekt, který je nejblíže místu, kam uživatel kliknul
  - Označení úzké čáry v grafických kreslítkách
  - Výběr jednoho objektu z okolí (tablety, mobily)
  - Tzv. hit-test

TROCHU GEOMETRIE

- Průsečík objektů
  - V místě průsečíku se x, y obou objektů rovnají
  - Vede na soustavu rovnic o dvou neznámých
  - Výsledkem místo průsečíku
  - Vhodné mít jeden objekt vyjádřen parametricky
    - Přejde na rovnici o 1 neznámé (parametr)
  - Např. vykreslení spojnicového graf se zvýrazněním míst průsečíků dvou řad
  - Např. odrážení "míčku" od stěn okna

TROCHU GEOMETRIE V JAVĚ

- Třídy geometrických objektů (např. Ellipse2D) poskytují metody pro testy:
  - Průsečík s jiným objektem (metoda `intersect`)
  - Zda druhý objekt leží uvnitř jiného (metoda `contains`)
    - Lze využít např. pro zahájení operace drag&drop

KONEC

- Příště: Základní bitmapová grafika