VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY
INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

NÁSTROJ PRO IMPORT A ANIMACI VEKTOROVÉ GRAFIKY V JAZYCE MATLAB

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
Daniel Šustáček

VEDOUCÍ PRÁCE
SUPERVISOR
doc. Ing. Robert Grepl, Ph.D.

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav mechaniky těles, mechatroniky a biomechaniky
Student:	**Daniel Šustáček**
Studijní program:	Aplikované vědy v inženýrství
Studijní obor:	Mechatronika
Vedoucí práce:	**doc. Ing. Robert Grepl, Ph.D.**
Akademický rok:	2020/21

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:

Nástroj pro import a animaci vektorové grafiky v jazyce MATLAB

Stručná charakteristika problematiky úkolu:

Tato práce se bude zabývat otázkou importu vektorového grafického formátu SVG do podoby vhodné pro vykreslení v jazyce MATLAB.
Výstupem bude sada funkcí a doporučení, která umožní uživatelsky snadné využití složitých vektorových obrázků pro vizualizaci. Součástí práce je podpora animace těchto objektů, typickým příkladem je pohyb schémat mechanismů. Důležitou částí práce je vypracování souboru ukázkových příkladů.

Cíle bakalářské práce:

1) Prozkoumejte dostupné nástroje pro práci s XML soubory a import/export vektorových grafických formátů do prostředí MATLAB.
2) Pro vybraný vektorový formát (pravděpodobně SVG) vytvořte nástroj pro import, zpracování a vykreslení. Zaměřte se na všechny hlavní objekty SVG (čáry, kružnice, obdélníky, křivky, texty a jejich vlastnosti, bitmapy). Ověřte možnosti a vlastnosti SW pro práci s vektorovou grafikou (Inkscape, Corel, AI) vzhledem k exportovanému SVG a vypracujte doporučení pro tvorbu grafických objektů.
3) Vytvořte nástroj pro animaci objektů v rovině, zahrnující definici vazeb mezi objekty (vychází z group ve formátu SVG), kinematiku a např. změnu měřítka.
4) Vlastnosti a možnosti vytvořených knihoven demonstrujte na sadě příkladů z oblasti mechaniky těles (statika, kinematika).

Seznam doporučené literatury:

DUŠEK, F.: Matlab a Simulink, skriptum ČVUT, 1998.

KRATOCHVÍL, C.: Mechanika těles - dynamika, skriptum FSI VUT v Brně, 1995.

GREPL, R.: Modelování mechatronických systémů v Matlab/SimMechanics, BEN, 2007.

VALÁŠEK, M.: Mechatronika, Vydavatelství ČVUT 1995.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2020/21

V Brně, dne

L. S.

prof. Ing. Jindřich Petruška, CSc. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

ABSTRAKT

Tato práce se zabývá návrhem nástroje pro import a animaci vektorové grafiky SVG v jazyce MATLAB. První část práce se zabývá způsobem importu SVG formátu do prostředí MATLAB a následnými principy vykreslení jednotlivých elementů SVG. Druhá část objasňuje vytvoření nástroje pro animaci pohybu mechanismů v rovině definovaných ve formátu SVG.

ABSTRACT

This thesis deals with the design of a tool for the import and animation of vector graphics SVG in MATLAB. The first part of the thesis deals with the means of importing the SVG format into the MATLAB environment and the subsequent principles of rendering individual SVG elements. The second part explains the design of a tool for animating the movement of mechanisms in the plane defined in the SVG format.

KLÍČOVÁ SLOVA

SVG, XML, MATLAB, vektorová grafika, kinematika mechanismů, animace objektů v rovině, Inkscape

KEYWORDS

SVG, XML, MATLAB, vector graphics, kinematics of mechanisms, animation of objects in the plane, Inkscape

BIBLIOGRAFICKÁ CITACE

ŠUSTÁČEK, Daniel. *Nástroj pro import a animaci vektorové grafiky v jazyce MATLAB*. Brno, 2021. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/129703. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav mechaniky těles, mechatroniky a biomechaniky. Vedoucí práce Robert Grepl.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že jsem bakalářkou práci na téma Nástroj pro import a animaci vektorové grafiky v jazyce MATLAB vypracoval samostatně s využitím citovaných podkladů uvedených v seznamu zdrojů.

Daniel Šustáček

PODĚKOVÁNÍ

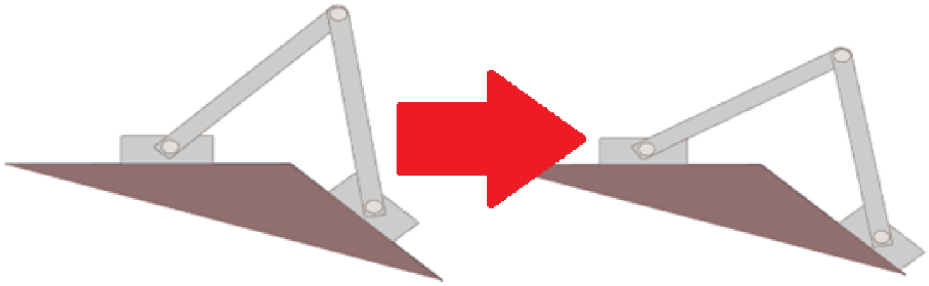
Tímto bych chtěl poděkovat svému vedoucímu doc. Ing. Robertu Greplovi, Ph.D. za přínosné vedení a správné nasměrování při psaní mojí práce. Také chci poděkovat své rodině a přátelům za podporu při studiu

OBSAH

1. ÚVOD

1.1. SVG A ANIMACE

Ve strojírenství se na spoustě míst setkáváme s nejrůznějšími mechanismy a systémy, které se pohybují. Díky programům, jako je právě MATLAB, dokážeme dnes během několika vteřin přesně vypočítat a simulovat pohyb jednotlivých součástí těchto mechanismů. Výsledné hodnoty jsou nesmírnou pomocí při řešení různých problémů. Pokud však chceme tento pohyb systému přehledně znázornit, je nejlepším způsobem systém načrtnout a pomocí softwaru vytvořit animaci tohoto obrázku.



Obrázek 1.1: Příklad mechanismu

MATLAB sám o sobě nenabízí kvalitní a rychlý způsob, jak takovou animaci realizovat. Pokud chceme animovat graficky komplikovaný a přehlednější mechanismus (například mechanismus na obr 1.1), nemáme k tomu úkonu dostupné nástroje.

Tento nedostatek se bude snažit vyřešit tato práce. Pokusí se vytvořit nástroj, který nám umožní provést libovolnou animaci z jakéhokoli obrázku. K navržení takového nástroje bude potřeba vyřešit hned několik problémů. Nejprve musíme umožnit MATLABu vykreslit složitý obrázek, který zvolí uživatel tohoto nástroje. Následně je třeba umožnit uživateli jednoduchým způsobem definovat, jak se má tento obrázek pohybovat, a na závěr celou tuto animaci vykreslit.

K vytvoření obrázku bude použit libovolný obrázek v SVG formátu. Finálním výsledkem by tedy měl být nástroj, který umožní vykreslit v MATLABu libovolný obrázek ve formátu SVG, jenž bude rozpohybován animací podle způsobu zvoleného uživatelem.

Tento nástroj by mohl být využíván k vizualizaci výsledků ve spoustě odvětví, a nejen ve strojírenství, například v oblasti statiky, kinematiky nebo pružnosti a pevnosti.

1.2. CÍLE ŘEŠENÍ

Cílem práce je vytvořit kód v prostředí MATLAB, který bude splňovat požadovanou funkcionalitu. Kód by měl být obsluhován sadou jednoduše pochopitelných funkcí. Cílem práce tedy je:

1. Najít a prozkoumat funkcionalitu nástrojů, které jsou dostupné pro práci se SVG formátem – rozebráno v kapitole 2.4.
2. Navrhnout nástroj, který dokáže importovat SVG soubor do MATLABu a vykreslit jej – kapitola 3.3.
3. Navrhnout nástroj, který umožní uživateli animovat pohyb mechanismu nakresleného ve formátu SVG – kapitola 3.4.
4. Demonstrace funkcionality nástroje na sadě příkladů – kapitoly 3.4.5–3.4.7 a 3.5.

2. REŠERŠE

Jelikož je zásadním faktorem jednoduchost obsluhy nástroje a co nejpřesnější vykreslení daného obrázku, bylo zadání upraveno na specifikování softwaru pro vytváření animovatelných mechanismů na softwaru Inkscape, viz 2.3.5.

2.1. XML STRUKTURA

XML (Extensible Markup Language) je značkovací jazyk obecně používaný napříč celým internetem, v oblasti výpočetní techniky a informační vědy. První verze tohoto jazyka byla vytvořena již v roce 1997. Jazyk byl v průběhu let několikrát aktualizován až do páté edice verze 1.0 vydané v roce 2008. Účelem vytvoření tohoto jazyka bylo poskytnutí jednoduchého způsobu předávání informací mezi vzájemně nezávislými systémy. Dnes je používán prakticky všude na internetu jakožto univerzální způsob předávání informací. Bez nadsázky je tento jazyk jedním ze zásadních stavebních bloků internetu a výpočetní techniky, tak jak je známe. Informace o tomto jazyce můžeme čerpat z oficiálních stránek W3C [1].

2.1.1. ZNAČKOVACÍ JAZYK XML A JEHO SYNTAX

K porozumění jazyku XML je vhodné správné pochopení pojmu značkovací jazyk. Definovat jej můžeme takto:

„A markup language is a computer language that uses tags to define elements within a document. It is human-readable, meaning markup files contain standard words, rather than typical programming syntax."

(Zdroj: TechTerms The Computer Dictionary [2])

Největší výhodou formátu XML je, že značky používané k označkování informací jsou nepřednastavené, to znamená, že uživatel může sám zvolit, jak bude struktura vypadat. Tuto vlastnost je možné demonstrovat na zapsání jednoduché informace v tomto jazyce (Obrázek 2.1). Z příkladu si lze i vytvořit základní představu o syntaxi tohoto jazyka. Příklad a více informací můžeme dohledat na stránkách W3schools v sekci *xml syntax* [3].

```
<note>
  <date>2015-09-01</date>
  <hour>08:30</hour>
  <to>Tove</to>
  <from>Jani</from>
  <body>Don't forget me this weekend!</body>
</note>
```

Obrázek 2.1: Příklad zápisu v XML

Jak lze poznat z uvedeného příkladu, označení informace v jazyce XML spočívá v uzavření dané informace mezi začátek značení <specifikovaná_značka> a ukončení značení <specifikovaná_značka>. Do těchto značek můžeme uzavírat další značky. Element může být prázdný, v takovém případě jej značíme takzvanou samouzavírací značkou <samouzavírací_značka />. Po přečtení příkladu asi každý člověk pochopí, co je významem této informace. Díky XML jazyku může být tato informace efektivně zpracována i počítačovým systémem.

2.1.2. VYUŽITÍ XML

Posledním zásadním poznatkem o XML je skutečnost, že tato struktura sama o sobě nemá žádnou výpočetní funkci, v XML tedy není nijak nastaveno, jak s danou informací zacházet. Pokud chceme, aby software informaci určitým způsobem zpracoval či zobrazil, potřebujeme ještě další software, nebo sadu pravidel, která specifikuje, jak zacházet s informacemi s příslušnou značkou.

Proto bylo na principu XML vytvořeno několik dalších jazyků, které již očekávají specifické značení a dokážou konkrétní informaci dál zpracovat. Nejrozšířenější z „nástaveb“ XML je protokol HTML, sloužící k zobrazování internetových stránek. Dalším využitím XML je třeba jazyk SVG, který informace v těchto značkách dokáže graficky zobrazit.

2.1.3. DEFINOVÁNÍ POJMŮ XML

Syntax samotného XML formátu není nijak složitá záležitost, pochopení by tedy nemělo být problematické. Definujeme si následující pojmy, které jsou nezbytné k porozumění funkcionality XML jazyka, viz w3school [3].

Element

Jedná se o základní bloky XML jazyka, jehož rozsah je značen pomocí *tagů* (značek). Rozsah je vymezen od *start tagu* po *end tag*. Do elementů jsou informace vkládány pomocí textu, atributů, nebo pomocí vkládání dalších elementů. Název značek používaný k ohraničení elementu odpovídá i názvu samotného elementu. První element v XML struktuře se nazývá *root*. Pro demonstraci použijeme příklad z w3school [4].

```
<bookstore>
  <book category="children">
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title>Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

Obrázek 2.2: Příklad použití elementů (Zdroj: w3school [4])

Uvedený příklad je vymezen hlavním (root) elementem <bookstore>, jenž obsahuje dva další elementy <book>. Element <book> obsahuje atribut (category="children") a další elementy <title>, <author>, <year>, a <price>, jež obsahují informace v podobě textu.

Tag
Tag, v češtině značka, je používán pro značení začátku a konce elementu. V XML existují dva typy tagu – *start tag* a *end tag*. *Start tag* se značí pomocí <NázevTagu> a slouží k určení počátku elementu. *End tag* se značí pomocí </ NázevTagu> a slouží k ukončení elementu.

Attribute
Attribute česky označujeme termínem atribut. Atributy jsou vkládány do značky start tagu. Každý atribut má název a svoji hodnotu. Ta musí být vždy uvedena v uvozovkách. V následujícím příkladu můžeme vidět použití atributu k definici polohy obdélníku.

```
1   <rect
2       id="rect845"
3       width="572.93628"
4       height="406.65866"
5       x="72.646111"
6       y="-148.85098" />
```

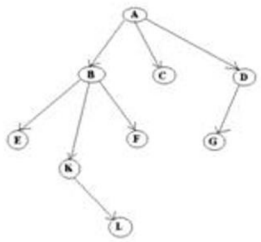
Obrázek 2.3: Příklad použití atributů

2.1.4. XML JAKO STROM STRUKTURA

Jak bylo uvedeno, XML jazyk nám umožňuje označit data do takové podoby, v níž jsou nezávislé programy schopny tato data jednoduše reprezentovat. Způsob, jak toho XML jazyk dosahuje, je jeden ze základních principů používaných napříč informačními vědami. Jedná se o takzvaný *Tree Data Structure* neboli česky o strukturu stromu.

„A tree is a collection of nodes connected by directed (or undirected) edges. A tree is a nonlinear data structure, compared to arrays, linked lists, stacks and queues which are linear data structures. A tree can be empty with no nodes or a tree is a structure consisting of one node called the root and zero or one or more subtrees. A tree has following general properties:

One node is distinguished as a root; Every node (exclude a root) is connected by a directed edge from exactly one other node;A direction is: parent -> children



Obrázek 2.4: Tree structure, zdroj viz [5]

A is a parent of B, C, D, B is called a child of A. On the other hand, B is a parent of E, F, KIn the above picture, the root has 3 subtrees.

Each node can have arbitrary number of children. Nodes with no children are called leaves, or external nodes. In the above picture, C, E, F, L, G are leaves. Nodes, which are not leaves, are called internal nodes. Internal nodes have at least one child. Nodes with the same parent are called siblings. In the picture, B, C, D are called siblings."

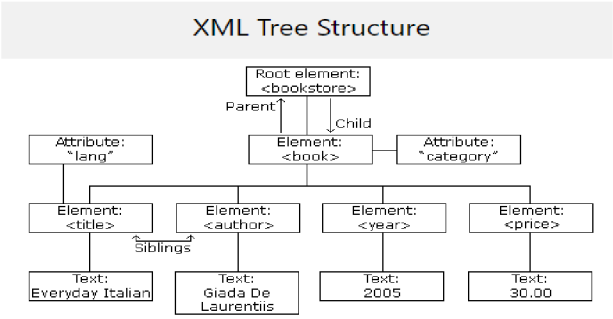
(Zdroj: An Online Textbook for 15-111 Intermediate and Advanced Programming [5])

Definujeme si několik pojmů, jež budou napříč textem práce používány. Pro přehlednost uvedeme jejich český a anglický název. České názvy jsou přebrány z článku *Stromové datové struktury* na webu itnetwork.cz [6].

Node (Uzel)
Jedná se o stavební bloky stromu. Jejich spojováním získáváme požadovanou strukturu stromu.

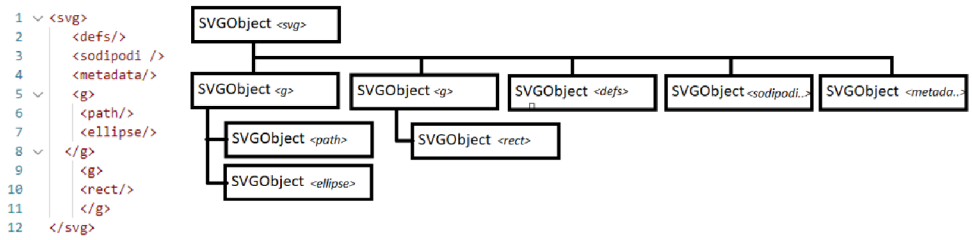
Root (Kořen)
První uzel stromu nazýváme kořen. Nemá předcházející uzly a zároveň na něj navazují (jsou k němu připojeny) všechny ostatní uzly.

Siblings, Parent a Child
Uzly mají mezi sebou určité vztahy. Jsou-li dva uzly spojeny, pak ten, který je ve struktuře výše (blíže ke kořenu stromu), se nazývá *Parent* (předcházející uzel), druhý uzel, který je ve struktuře níž (dál od kořene), se nazývá *Child* (následující uzel). Uzly, jež mají stejný předcházející uzel, nazýváme *Siblings* (sourozenci). Uvedené vztahy mužeme demonstrovat na XML struktuře příkladem z w3school [7].



Obrázek 2.5: Diagram stromu v XML (Zdroj: w3school [7])

V XML je struktury stromu docíleno vkládáním jednotlivých elementů do elementů jiných. Naznačení této struktury můžeme ukázat na následujícím příkladu.

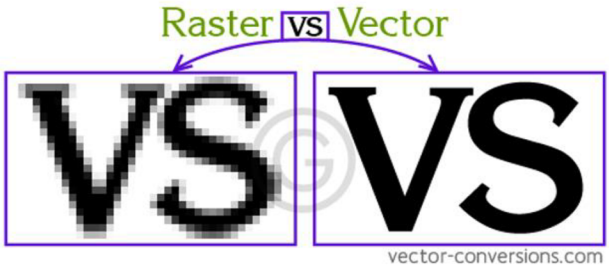


Obrázek 2.6: Ukázka vytvoření stromu z XML struktury

2.2. VEKTOROVÁ GRAFIKA

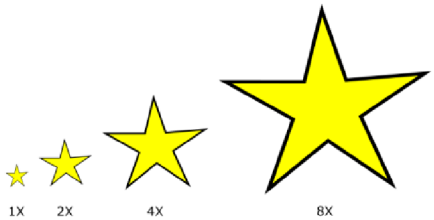
Vektorová grafika je typ počítačové grafiky, která příslušný obrázek ukládá jakožto matematický popis jednotlivých částí, z nichž se obrázek skládá. Obrázek vektorové grafiky je v podstatě soubor souřadnic spojených přímkami, nebo křivkami. Dále je určeno, zda a jak jsou obrazce vytvořené těmito křivkami vyplněny.

Klasicky je vektorová grafika srovnávaná s grafikou rastrovou. Ta funguje na principu pixelů, kde je celý obrázek rozložen do mřížky čtverečků, které mají specifikovanou barvu (viz Vektorová a rastrová grafika na PC [8], str.7–8).



Obrázek 2.7: Srovnání rastrové a vektorové grafiky (Zdroj: vector-conversion.com [19])

Při srovnávání těchto dvou typů grafiky zjistíme, že vektorová grafika má jednu hlavní výhodu, a to takzvané „*scalování*“ neboli zvětšování obrázku. Jelikož je vektorový obrázek tvořen pouze zápisem souřadnic, je možné jej neomezeně zvětšovat bez ztráty kvality. Pokud bychom chtěli zvětšovat rastrový obrázek, můžeme se dostat do velikosti, kde budou na obrázku viditelné jeho jednotlivé pixely, a tím pádem již nebude vhodný pro chtěné použití.



Obrázek 2.8: Scalování ve vektorové grafice

Právě díky této možnosti zvětšování se vektorová grafika stala normou pro vytváření obrázku napříč internetem. V průběhu historie internetu bylo vytvořeno nespočet formátů vektorové grafiky, jako například .AI, .EPS, .SVG nebo DRW. K vytváření této grafiky je logicky dostupné velké množství softwarových editorů k vytváření vektorových obrázků.

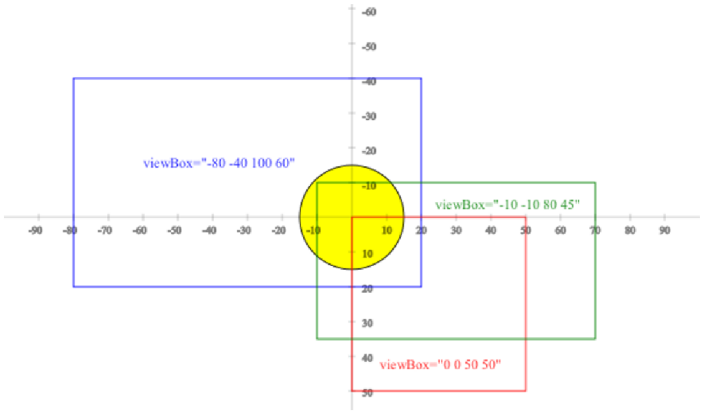
2.3. SVG FORMÁT

SVG (Scalable Vector Graphics) je formát 2D vektorové grafiky založený na XML jazyce. Vznikl jakožto formát používaný k vytváření grafického designu internetových stránek. Vývoj tohoto formátu započal v roce 1999 a jeho nejnovější verze SVG 1.1 (Second Edition) byla publikována v roce 2011. SVG formát umožňuje vytvořit jakýkoli grafický obrázek, který je pomocí jeho vlastností možno různě uskupovat, stylovat nebo transformovat do požadované podoby. Zobrazení obrázků SVG je velice lehké, protože se dnes jedná o hlavní formát webového designu, většina dostupných webových prohlížečů je schopna obrázky uložené v něm zobrazit. Jeho principy a syntax jsou jednoduché a pro člověka čitelné, je tedy možno SVG obrázky vytvářet i prostým zápisem v textovém editoru. Jelikož je SVG formát uchováván v XML struktuře, je snadné části SVG obrázku indexovat a algoritmicky zpracovávat. Stejně jako u XML i u SVG můžeme využívat oficiální dokumentaci W3C [9].

Jelikož je tento formát navržený k designu webových stránek, je lehké z obrázků v něm uložených vytvořit animace. K samotnému rozpohybování můžeme využít jeho vlastnost transformace, jíž se bude zabývat kapitola 2.3.4. Jak již bylo zmíněno, obrázek je uložen v textovém souboru XML, což jej činí snadno exportovatelným do MATLABu. Z těchto dvou důvodů je uvedený soubor zvolen jakožto formát pro vytváření animací v navrhovaném nástroji.

2.3.1. SOUŘADNICOVÝ SYSTÉM

Prvním krokem k pochopení funkcionality SVG formátu je porozumění souřadnému systému, ve kterém jsou obrázky uloženy. Obrázky v něm uložené jsou pouhé souřadnice geometrických křivek. Vykreslení obrázku je docíleno zobrazením těchto křivek v kartézském systému. Problémem však je, že software, který má tento obrázek zobrazit, neví, kam umístit střed tohoto systému. SVG toto řeší pomocí atributu „*viewbox*“ (co je atribut, je popsáno v kapitole 2.3.5).



Obrázek 2.9: ViewBox (Zdroj: Fizz [20]

Atribut „viewbox“ je umístěný v každém souboru SVG, jsou v něm definovány 4 souřadnice relativně k souřadnému systému, jež říkají, jak velký obrázek vlastně je a které části obrázku mají být vykresleny (pokud se nějaká část obrázku nachází mimo „*viewbox*“, nemá být vykreslena). Na obrázku 2.9 je znázorněno, jak může být definovaný „*viewbox*“ a jaká část obrázku bude zobrazena.

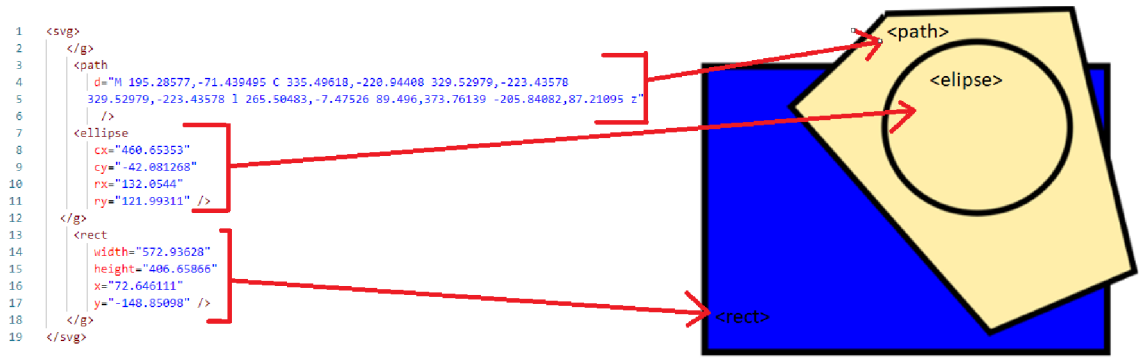
2.3.2. SVG ELEMENTY

Pokud otevřeme SVG soubor v libovolném textovém editoru, uvidíme zápis ve struktuře XML (tento zápis budeme referovat jakožto zdrojový kód SVG). Z kapitoly 2.1 víme, že tato struktura se člení do elementů. Každý z těchto elementů uchovává určitou informaci o obrázku. Na rozdíl od XML v SVG již značky těchto elementů nemohou být libovolné. Pro SVG jsou definované značky, jež říkají, jak má informace v těchto elementech upravit obrázek. Pokud by se v souboru vyskytla značka, která není SVG standardy rozpoznána, byla by ignorována a obrázek se vykreslí, jako kdyby zde příslušný element s touto značkou nebyl.

Definované elementy SVG můžeme pro naše účely rozdělit do dvou kategorií: ty, které reprezentují jednotlivé části obrázku a budou při zobrazení obrázku vykresleny (např. *rect*, *line* apod.), a elementy, jež samy vykreslené nejsou, pouze obsahují dodatečné informace o obrázku.

Elementy v první kategorii jsou vlastně stavební bloky obrázku ve formátu SVG, při zobrazení je jeden po druhém vykreslen, čímž se objeví celý obrázek. Elementy se vykreslují

v pořadí, v němž jsou zapsány, elementy zapsané v pořadí A, B budou vykresleny tak, že B bude nad A. Vybrané elementy, s nimiž nástroj bude pracovat, budou popsány dále.

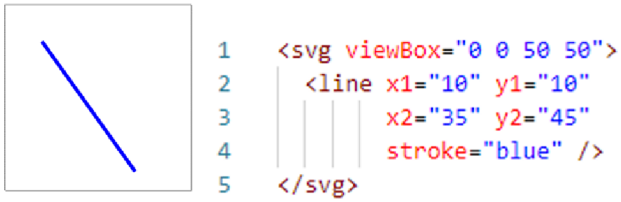


Obrázek 2.10: Reprezentace elementů na vykresleném obrázku

Elementům jsou přiřazovány atributy, jež blíže specifikují, jak má daný element vypadat. Například kruhu jsou přiřazeny atributy *cx* a *cy*, které obsahují souřadnice jeho středu a atribut *r*, který obsahuje jeho poloměr. Některé atributy lze přiřadit pouze k jednomu specifickému elementu (tyto atributy většinou musejí být specifikovány, aby se element vůbec vykreslil), zatímco jiné lze přiřadit k libovolnému elementu (ty většinou k úspěšnému vykreslení potřeba nejsou). Atributy, které jsou specifické pro daný element, budou popsány v této kapitole ve výčtu elementů. Ty, které lze přiřadit ke všem elementům, budou popsány v kapitole 2.3.4.

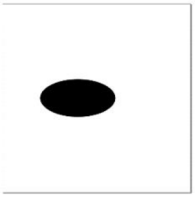
Line

Element <line> je prostá přímka, je specifikovaná atributy jejího počátku *x1, y1* a jejího konce x*2, y2*.



Obrázek 2.11: Element <line>

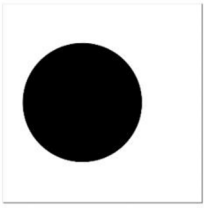
Ellipse
Element <ellipse> reprezentuje elipsu. Je definována pomocí středu elipsy a dvou poloměrů; atributy *cx, cy* (souřadnice středu) a *rx, ry* (poloměry). Samotný element umožňuje vykreslit elipsu pouze ve dvou základních orientacích. Pro natočení elipsy jsou využity další prostředky (viz kapitola 2.3.5).



```
<svg viewBox="0 0 50 50">
  <ellipse cx="20" cy="25"
           rx="10" ry="5" />
</svg>
```

Obrázek 2.12: Element <ellipse>

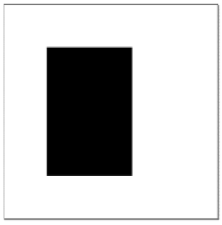
Circle
Element popisující kruh, jelikož je kruh v podstatě specifický případ elipsy, kdy jsou oba jeho poloměry stejné, má stejné atributy pro určení středu a jeden atribut *r* pro určení poloměru.



```
<svg viewBox="0 0 50 50">
  <circle cx="20" cy="25"
          r="15"/>
</svg>
```

Obrázek 2.13: Element <circle>

Rect
Element <rect> představuje obdélník. Pro jeho platné vykreslení musí být zadán jeho počátek (souřadnice levého horního rohu) a jeho výška a šířka – pomocí atributů *x,y* a *width a height*.



```
<svg viewBox="0 0 50 50">
  <rect x="10" y="10"
        width="20"
        height="30"/>
</svg>
```

Obrázek 2.14: Element <rect>

G
Tento element je pouze kontejner používaný pro seskupování dalších elementů. Možnosti jeho využití a bližší popis najdeme v kapitole 2.3.3.

Path

Path je nejobsáhlejší element SVG. Většina obrázků, jež budou v SVG vytvořeny nebo které jsou běžně dostupné na internetu, vzniká převážně pomocí tohoto elementu. Právě on dokáže reprezentovat prakticky jakoukoli křivku nebo útvar. Dosáhne toho tím, že v sobě obsahuje pouze jeden atribut *d,* atribut instrukcí, podle kterých je daná křivka vykreslena. Instrukce bývají typu: vykreslit přímku z bodu A do bodu B, z bodu B vykreslit část kruhu o poměru x do bodu C. Celkový popis tohoto elementu najdeme zde v dokumentaci [10]. V rámci tohoto elementu můžeme vykreslovat takzvané Bézierovy křivky. Ty jsou definovány jako:

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-1} t^i P_i \tag{2.1}$$

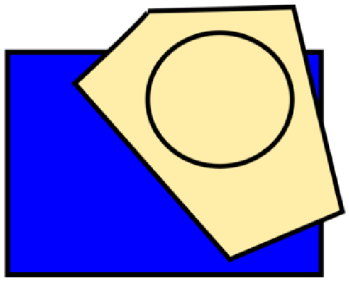
Právě díky nim lze pomocí elementu <path> reprezentovat prakticky jakýkoli obrazec.

2.3.3. VRSTVY A POŘADÍ VYKRESLOVÁNÍ V SVG

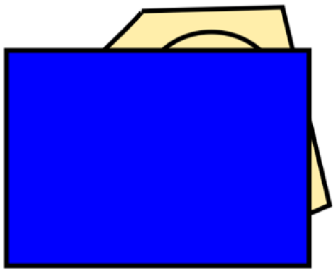
V předchozí kapitole jsme již představili element <g>, jenž obsahuje další elementy (viz obr. 2.14). Elementy, jež sám obsahuje, uskupuje do skupiny, která může být celkově upravována. Této skupině se ve vektorové grafice říká vrstva. Například pokud chceme tři elementy posunout doprava, můžeme je vložit do elementu <g> a jedním příkazem všechny tři posunout. Jak SVG docílí posunutí elementů, se dozvíme v kapitole 2.3.6. Prakticky tedy tento element umožňuje rychlejší editaci více elementů a seskupování elementů do skupin.

Druhou vlastností, která nemusí být hned patrná, je úprava pořadí vykreslování vrstev. Pokud některé elementy umístíme do vrstvy a přemístíme ji v XML pod ostatní elementy, docílíme odlišného překrytí částí obrázků (viz obr. 2.15).

```svg
<svg
   id="svg8">
   <g id="layer1">
    <rect
       style="fill:#0000ff;stroke:#000000;stroke-width:9.35497"
       id="rect845"
       width="572.93628" height="406.65866"
       x="72.646111" y="-148.85098" />
   </g>
   <g
     id="layer2">
    <path
       style="fill:#ffeeaa;stroke:#000000;stroke-width:8.6714;"
       d="M 195.28577,-91.439495 C 335.49618,-220.94408
          329.52979,-223.43578 329.52979,-223.43578 l 265.50483,
          -7.47526 89.496,373.76139 -205.84082,87.21095 z"
       id="path838" />
    <ellipse
       style="fill:#ffeeaa;stroke:#000000;stroke-width:8.6714"
       id="path840"
       cx="460.65353" cy="-62.081268"
       rx="132.0544" ry="121.99311" />
   </g>
</svg>
```



```svg
<svg
   id="svg8">
   <g
     id="layer2"
     style="fill:#ffeeaa;stroke:#000000;stroke-width:8.6714;">
    <path
       style="fill:#ffeeaa;stroke:#000000;stroke-width:8.6714;"
       d="M 195.28577,-91.439495 C 335.49618,-220.94408
          329.52979,-223.43578 329.52979,-223.43578 l 265.50483,
          -7.47526 89.496,373.76139 -205.84082,87.21095 z"
       id="path838" />
    <ellipse
       id="path840"
       cx="460.65353" cy="-62.081268"
       rx="132.0544" ry="121.99311" />
   </g>
   <g id="layer1">
    <rect
       style="fill:#0000ff;stroke:#000000;stroke-width:9.35497"
       id="rect845"
       width="572.93628" height="406.65866"
       x="72.646111" y="-148.85098" />
   </g>
</svg>
```



Obrázek 2.15: Změna pořadí pomocí elementu <g>

2.3.4. SVG ATRIBUTY

Již v kapitole 2.3.2 bylo nastíněno, že právě pomocí atributů jsou k elementům přiřazovány další informace, podle kterých se mění to, jak finální obrázek vypadá (například říkají, jakou má mít grafický útvar barvu). Každý element má specifické atributy, jež mu lze přiřadit. Existují ale i atributy, které lze přiřadit elementu libovolnému, je jich velké množství a jejich kompletní seznam nalezneme v dokumentaci viz [11]. Ty, které jsou pro rozsah práce nejdůležitější, budou popsány zde.

Fill

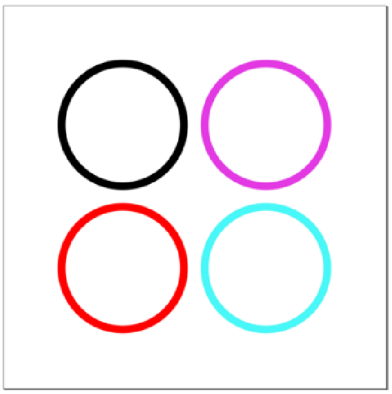
Atribut *fill* ovlivňuje barvu výplně grafických útvarů reprezentovanými elementy. Jeho hodnota je buď název podporované barvy (např. *black*, *blue*, *red* apod.), nebo hodnota RGB barvy v HEX podobě (#000000 - #ffffff), viz [12].

```
<svg viewBox="0 0 50 50">
  <circle fill="blue"
          cx="16.6" cy="16.6"
          r="8"/>
  <circle fill="#e3c41b"
          cx="16.6" cy="33.3"
          r="8"/>
  <circle fill="red"
          cx="33.3" cy="16.6"
          r="8"/>
  <circle fill="e339e3"
          cx="33.3" cy="33.3"
          r="8"/>
</svg>
```

Obrázek 2.16: Atribut fill

Stroke

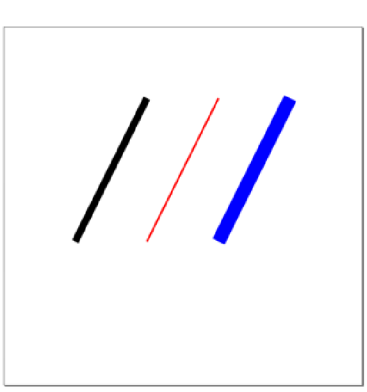
Atribut *stroke* označuje barvu ohraničení grafického útvaru (samotná čára spojující grafické souřadnice). Podobně jako u atributu *fill* je jeho hodnota buď specifikovaná barva, nebo RGB hodnota barvy.



```
<svg viewBox="0 0 50 50">
 <g fill="white">
  <circle stroke="black"
          cx="15.6" cy="15.6"
          r="8"/>
  <circle stroke="red"
          cx="15.6" cy="34.3"
          r="8"/>
  <circle stroke="#e339e3"
          cx="34.3" cy="15.6"
          r="8"/>
  <circle stroke="#48f7f7"
          cx="34.3" cy="34.3"
          r="8"/>
 </g>
</svg>
```

Obrázek 2.17: Atribut stroke

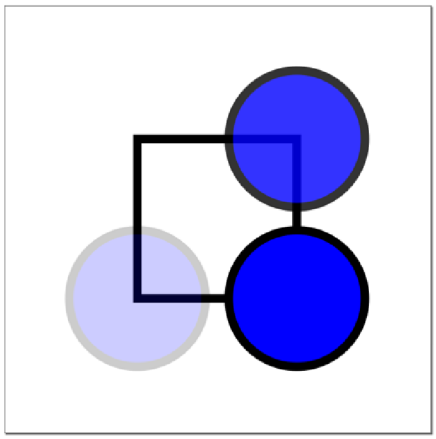
Stroke-width
Stroke-width je atribut, který určuje tloušťku ohraničení. Jeho hodnota je číslo šířky ohraničení s příslušnou jednotkou.



```
<svg viewBox="0 0 50 50">
<line   stroke-width="1"
        x1="20" y1="10"
        x2="10" y2="30"
        stroke="black" />
<line   stroke-width="0.2pt"
        x1="30" y1="10"
        x2="20" y2="30"
        stroke="red" />
<line   stroke-width="0.5mm"
        x1="40" y1="10"
        x2="30" y2="30"
        stroke="blue" />
</svg>
```

Obrázek 2.18: Atribut stroke-width

Opacity
Opacity určuje průhlednost jednotlivých elementů. Jeho hodnota je číslo v rozmezí 0–1, kde 0 reprezentuje úplnou průhlednost (element nebude zobrazen) a 1 náleží žádné průhlednosti (element je zobrazen, jako by zde tento atribut nebyl).



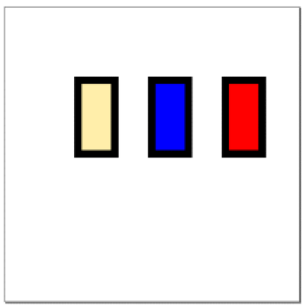
```
<svg viewBox="0 0 50 50">

 <rect fill="none"
       stroke="black"
       x="15.6" y="15.6"
       width="18.7"
       height="18.7"/>
 <g fill="blue"
    stroke="black" >
   <circle opacity="0"
           cx="15.6" cy="15.6"
           r="8"/>
   <circle opacity="0.2"
           cx="15.6" cy="34.3"
           r="8"/>
   <circle opacity="0.8"
           cx="34.3" cy="15.6"
           r="8"/>
   <circle opacity="1"
           cx="34.3" cy="34.3"
           r="8"/>
 </g>
</svg>
```

Obrázek 2.19: Atribut opacity

Transform
Tento atribut umožňuje upravovat pozici grafických souřadnic. Jeho hodnotu představuje jedna z definovaných transformačních funkcí (*rotate*, *translate*, *matrix*…) a její parametry. Tyto transformační funkce umožňují změnu polohy souřadnic grafických souřadnic (například posunutí nebo natočení). Všechny funkce můžeme dohledat v dokumentaci viz [13]. Tyto funkce používají definované matice, aplikované na souřadnice grafických útvarů (viz kapitola 2.3.6).

Style
Atribut *style* umožňuje upravovat grafické elementy pomocí CSS deklarací. Tyto deklarace jsou většinou stejné jako samotné atributy SVG. Jeho hodnotou je seznam atributů s jejich hodnotami ve formátu „atribut1: hodnota_atributu1; atribut2: hodnota_atributu2; …;“.



```
<svg viewBox="0 0 40 40">
  <rect style="fill:#ffeeaa;stroke:#000000;"
        x="10" y="10"
        width="5"
        height="10"/>
  <rect style="red:blue;stroke:black;"
        x="20" y="10"
        width="5"
        height="10"/>
  <rect style="fill:red;stroke:#000000;"
        x="30" y="10"
        width="5"
        height="10"/>
</svg>
```

Obrázek 2.20: Atribut style

2.3.5. SVG EDITORY

Programů k editaci a tvorbě obrázků ve formátu SVG existuje velké množství, jejich kvalita, funkčnost a cena licence se liší. Při vytváření nástroje pro import SVG obrázku byla vyzkoušena práce s editory **Vecteezy Editor**, **Gravit Designer**, **Adobe Illustrator Draw**, **CorelDRAW** a **Inkscape**.

Většina z těchto editorů nabízí možnosti, které se dají využít k vytváření animovatelných mechanismů. Problém však pramení ze způsobu, jakým ukládají zdrojový kód obrázku. Každý z editorů ukládá zdrojový kód SVG v poněkud odlišné podobě. Jelikož úkolem této práce je zpracovávat tento kód pro prostředí MATLAB, je nepraktické řešit velké množství odlišností ve zpracovávaných obrázcích. Z tohoto důvodu byl vybrán editor Inkscape jakožto podporovaný editor pro tvorbu obrázků v tomto nástroji.

Inkscape byl vybrán především kvůli jeho free verzi, která umožňuje přístup k velkému počtu nástrojů, a pro jeho jednoduchý způsob práce s vrstvami. Navrhovaný nástroj bude stále umožňovat vykreslování a animaci obrázků vytvořených v jiných editorech, pouze bude tento úkon pro uživatele složitější a zvýší se pravděpodobnost, že vykreslení proběhne špatně.

2.3.6. TRANSFORMACE SVG

Jelikož je obrázek ve formátu SVG v podstatě pouze soubor bodů v kartézském systému, reprezentující souřadnice křivek, můžeme na ně aplikovat všechny matematické principy, které fungují pro úpravu těchto bodů. Jedním z nejužitečnějších principů je transformace pomocí matic. Nejprve definujeme, co maticové transformace jsou (definujeme pro 2D souřadný systém), viz Interactive Linear Argebra – Matrix Transformations [14].

Nechť množina n souřadnic jakýchkoli bodů je zapsaná v matici r:

$$r = \begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \end{bmatrix} \tag{2.2}$$

Nechť matici transformace A definujeme jakožto matici o rozměrech 2×2:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \tag{2.3}$$

Pokud aplikujeme maticová násobení matice A na množinu souřadnic r, získáme:

$$Ar = r' \tag{2.4}$$

Kde r je:

$$r' = \begin{bmatrix} x_1' & x_2' & \dots & x_n' \\ y_1' & y_2' & \dots & y_n' \end{bmatrix} \tag{2.5}$$

Aplikací této matice jsme transformovali souřadnice ve vektoru r do nové podoby r'. Můžeme tedy uvažovat o násobení maticí jako o funkci $f(r)$, která přetvoří vstupní souřadnice r do nových souřadnic r'. To, jak budou souřadnice transformovány, plně závisí na tom, jakou matici A zvolíme.

Pomocí těchto transformací (a volbou vhodné matice) můžeme docílit změny obrázku (potažmo jeho souřadnic) do prakticky jakékoli podoby. Nejvíce užitečné a nejpoužívanější jsou však dvě transformace, a to translace a rotace.

Translace

Definujeme matici translace A jako:

$$A = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

Samotná aplikace transformace je tedy:

$$\begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} d_x + x \\ d_y + y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \tag{2.7}$$

Pokud tuto matici aplikujeme na souřadnice *r*, získáme nové souřadnice *r'*, které odpovídají souřadnicím posunutí ve směru *v*.

Rotace
Definujeme matici rotace *A* jako:

$$A = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & 1\cos\theta \end{bmatrix} \tag{2.8}$$

Aplikací této matice získáme nové souřadnice bodů r', což jsou souřadnice otočené okolo středu souřadného systému o úhel θ ve směru hodinových ručiček.

$$\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & 1\cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix} \tag{2.9}$$

Vhodné je ještě zmínit, že kombinací translace a rotace můžeme souřadnice otočit okolo libovolného bodu.

2.4. EXISTUJÍCÍ NÁSTROJE PRO PRÁCI SE SVG V MATLABU

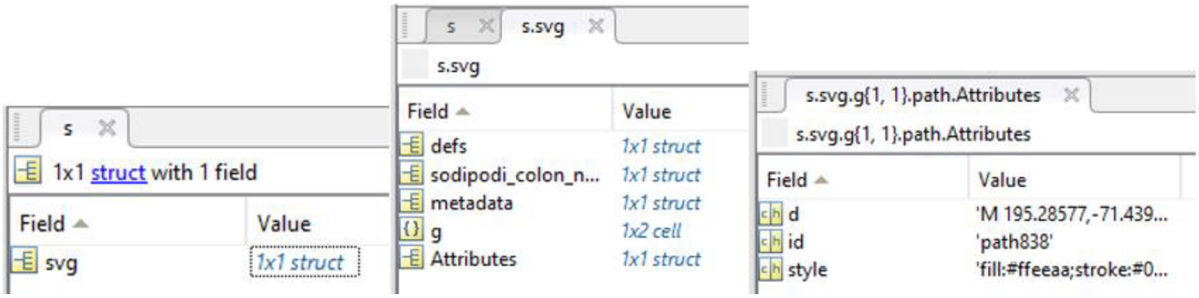
Pro práci se SVG formátem jako takovým neexistuje žádný nástroj, který by umožnil v MATLABU tento formát vykreslit, nebo nějak analyzovat. Oproti tomu pro práci s formátem XML můžeme najít nástroje, jež fungují relativně dobře.

MATLAB pro práci s XML formátem nabízí funkci *xmlread*, která převede XML soubor do objektu DOM. Ten umožňuje nahlížet na XML strukturu jako na strom a dále jej zpracovávat.

2.4.1. XML TO STRUCT

Jedním z nástrojů volně dostupných na *MathWorks: File Exchange* [15] je funkce vytvořená pro MATLAB *xml2struct()*. Tato funkce umožňuje dostat formát XML do podoby, v níž se v něm budeme moci v MATLABu orientovat. Jak plyne z názvu, tato funkce ze souboru ve formátu XML vytvoří v MATLABu datovou strukturu *structure*, která bude obsahovat všechny elementy, jež zpracovávaný soubor obsahuje. Elementy jsou reprezentovány dalšími

strukturami. Atributy jednotlivých elementů jsou poté převedeny na jednotlivá pole struktury (vytvořenou strukturu v MATLABu můžeme vidět na obrázku 2.21).



Obrázek 2.21: Zobrazení struktury vytvořené funkcí xml2struct v MATLABu

Tento způsob převedení XML formátu je velice užitečný, pokud však chceme zpracovávat formát SVG, je kvůli navigaci ve vytvořené struktuře tato funkce nevhodná. Ke zpracování SVG souboru však můžeme aplikovat podobné koncepty, které jsou použity v této funkci.

Po analýze zdrojového kódu této funkce zjistíme, že k převedení XML formátu do MATLABu je použito rekurzivní procházení zpracovávaného souboru a k samotnému získání informací o jednotlivých elementech souboru jsou využity v MATLABU implementované funkce pro práci s DOM soubory.

2.4.2. OBJEKT DOM V MATLABU

MATLAB umožňuje pracovat s knihovnami programovacího jazyka Java. Díky tomu můžeme pracovat v prostředí MATLAB s objektem DOM.

„Document Object Model (DOM) is a standard tree structure, where each node contains one of the components from an XML structure. Element nodes and text nodes are the two most common types of nodes. With DOM functions we can create nodes, remove nodes, change their contents, and traverse the node hierarchy.“

(Zdroj: ZetCode.com Java Dom [16])

Každý DOM objekt reprezentuje jeden uzel, tento uzel obsahuje všechny atributy reprezentované XML struktury a svoje následující uzly.

Pracovat s XML souborem jako se stromem je pro práci s ním v MATLABu velice užitečné. Použitím funkcí, jež jsou pro tento objekt definovány, můžeme jednoduše do MATLABu importovat všechny informace, které nás v souvislosti se zpracovávaným souborem zajímají. Představíme několik funkcí, jež mohou být využity k importování XML, potažmo SVG do MATLABu. Jejich bližší využití můžeme vidět v blogu XML in MATLAB od Michaela Katze [17].

getDocumentElement
Pokud tuto funkci aplikujeme na soubor XML, získáme objekt ve formátu DOM (reprezentovaný kořenovým uzlem této struktury), se kterým můžeme dále pracovat.

getNodeName
Jedná se o funkci, která vrátí jméno uzlu příslušného DOM objektu.

getChildNodes
Voláním funkce *getChildNodes()* s parametrem příslušného uzlu získáme seznam všech uzlů, jež jsou uzly následujícími ve struktuře stromu.

getAttributes
Touto funkcí získáme seznam všech atributů příslušného uzlu.

2.5. DEFINICE POHYBU POTŘEBNÁ K VYTVOŘENÍ ANIMACE

K navržení nástroje, který z příslušného obrázku vytvoří animaci, musíme nejprve definovat několik pojmů z oblasti kinematiky. Jelikož je pro nástroj důležité pouze vykreslování pohybu obrázku, bude se tato kapitola zabývat pouze kinematikou potřebnou k vykreslení pohybu (neboli popisující polohu). Rovnice, jež řeší rychlost a zrychlení, není třeba definovat.

Jelikož hovoříme o pohybu, musíme vždy stanovit, k jakému tělesu je pohyb vztahován (neboli kde se nachází střed vztahovaného souřadného systému). Pokud toto stanovíme, můžeme definovat polohový vektor a zadání pohybu (viz Kinematika [18], str. 6).

$$r = \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.10}$$

Tento vektor stanovuje polohu specifického bodu od zvoleného souřadného systému. Při pohybu se tento vektor mění, tím pádem je vektor *r* proměnnou času neboli:

$$r = r(t) \tag{2.11}$$

Pokud je nám známa tato časová funkce, hovoříme o zadaném pohybu.

Při animování mechanismů nás budou zajímat dva typy pohybu, a to translační přímočarý a rotační okolo středu souřadného systému. V těchto dvou případech je těleso určitým způsobem zavázáno a vykonává specifický pohyb.

V případě přímočarého translačního pohybu víme, že se těleso bude pohybovat po předem specifikované přímce, tím pádem můžeme pohyb zadat pomocí vzdálenosti posunutí d od výchozího bodu neboli:

$$r = r(d) \tag{2.12}$$

V případě rotačního pohybu okolo souřadného systému víme, že se velkost vektoru r nemění, mění se pouze úhel natočení θ, neboli:

$$r = r(\theta) \tag{2.13}$$

„Tuhé těleso je definováno jako soustava hmotných bodů, jejichž vzájemná poloha se při pohybu tělesa nemění.“

*(*Zdroj: Kinematika [18], *str. 27)*

Při animování se vnitřní struktura jednotlivých obrázků nemění, to znamená, že se budeme zabývat výhradně tuhými tělesy.

Další potřebné definice jsou definice mechanismů a jejich součástí (viz Kinematika [18], str. 102).

„Mechanismus je mechanické zařízení, které slouží k transformaci pohybu, nebo přenosu síly. Zařízení je tvořeno soustavou vzájemně pohyblivých těles, z nichž je vzhledem k ostatním jedno nepohyblivé. Toto nepohyblivé těleso nazýváme rám. Tělesa nazýváme členy mechanismu. Členy mechanismu jsou spojeny v kinematických dvojicích.“

„Kinematický řetězec vznikne spojením více těles pomocí kinematických dvojic.“

„Kinematická dvojice je spojení těles. Plochy, křivky nebo body, které se spolu stýkají a tvoří tak kinematickou dvojici, jsou prvky této dvojice.“

„Uzavřený kinematický řetězec je řetězec, který vznikne tak, že každý člen řetězce je připojen nejméně dvěma kinematickými dvojicemi.“

Kinematické dvojice neboli vazby můžeme popisovat ve 2D jakožto rotační, posuvné nebo vetknutí.

2.6. DEFINICE POJMŮ OOP

V této kapitole definujeme pojmy z oblasti objektově orientovaného programování (OOP) v MATLABu. OOP je princip zápisu kódu, který spoléhá na vytváření takzvaných objektů. Ty obsahují různá data, jež jsou pomocí těchto objektů upravována. Zmíněný styl psaní kódu je v oblasti programování velice populární, proto byl vybrán jakožto způsob vytvoření navrhovaného nástroje v této práci. Pro ujasnění definujeme několik pojmů z této oblasti, jež se v textu práce objevují.

Třída
Třída představuje jakýsi předpis, podle něhož následně mohou být vytvořeny požadované objekty. Jsou v ní definované proměnné uchovávající data a funkce, pomocí nichž jsou tato data zpracovávána.

Zapouzdření
Základním principem OOP je takzvané zapouzdření. Spočívá v definici proměnné nebo funkce v rámci specifické třídy. Takové proměnné a funkce budou existovat pouze v objektech vytvořených z těchto tříd.

Objekt
Objekt je datový typ, který je vytvořen pomocí třídou předem stanoveného předpisu. O objektech můžeme uvažovat jakožto o samostatných celcích, které mezi sebou neinteragují. Aby mezi sebou mohly interagovat, musíme jim to umožnit implementací určitých funkcí (metod).

Metoda a funkce
Obecně jsou všechny funkce, které jsou definovány pro specifickou třídu (jsou v ní zapouzdřeny), označovány jakožto metody. V prostředí MATLAB jsou však metody v rámci tříd definovány jako funkce. Rozdíl mezi těmito dvěma pojmy je tedy v MATLABu o něco méně zřejmý. V textu práce budeme ale funkcím, které jsou definovány pro určitou třídu, říkat metoda.

3. POSTUP A VÝSLEDKY ŘEŠENÍ

3.1. ANALÝZA PROBLÉMU

Cílem práce je návrh funkčního nástroje k importu a animování obrázku formátu SVG. Tvorba nástroje bude probíhat v prostředí MATLAB. Pomocí objektově orientovaného programování bude v MATLABu vytvořeno několik tříd, které budou uživatelem používány k vytváření jím zvolených animací.

V obsahu této práce bude použit pseudokód vytvořený v prostředí MATLAB. Nejedná se o funkční kód použitý ve finální verzi nástroje. Je z něj vyjmuto mnoho částí, které jsou potřeba k upravování datových typů a vytváření různých proměnných, jež jsou nutné k úplné funkčnosti kódu. Tyto ukázky kódu slouží k demonstraci určitého principu, který je zásadní k pochopení funkcionality nástroje. Funkční implementace je možno dohledat v příloze práce.

V práci budou podrobně rozebrány čtyři dílčí zásadní problémy, které bylo potřeba vyřešit při vývoji nástroje.

Parsování struktury do stromu v MATLABu – aby se dala data v MATLABu zpracovávat, musejí být nejprve nějak převedena do jeho prostředí. Je tedy třeba vytvořit algoritmus, který v MATLABu vytvoří data ve struktuře stromu. Dále je problematika rozvinuta v kapitole 3.3.1.

Vytvoření grafických dat pro MATLAB – aby bylo možné daný obrázek vykreslit v MATLABu, musejí být data z formátu SVG přetvořena do podoby, kterou bude MATLAB schopen pomocí příkazu *plot* vykreslit (dále rozvinuto v kapitole 3.3.2).

Převedení do struktury, v níž bude MATLAB schopen vykreslit a animovat celý obrázek – aby byla práce s daty pro následné animování pro uživatele jednoduchá a přehledná, musejí se data o grafických částech obrázku uskupit do přehledné podoby, vysvětlené v kapitole 3.3.3.

Animace obrázku – kapitola 3.4 se bude zabývat problematikou animování v MATLABu.

3.2. KONVENCE, FORMÁT A ARCHITEKTURA PROGRAMU

Při psaní kódu je vhodné dodržovat předem zvolené konvence, které umožní někomu, kdo by chtěl kód dále rozvíjet nebo jej pouze pochopit, se v něm lépe orientovat. Tyto zvolené

konvence, společně se základním představením, jak bude nástroj vypadat, budou představeny v nesledující kapitole.
Uvedeme nejdůležitější konvence, jež jsou v kódu (a i v textu práce) dodržovány:

Proměnná
Název proměnné je zapsán v tzv. *camelFormat* neboli jejich první písmeno je malé a následující slova jsou uvedená bez mezery s velkým písmenem.

Proměnné reprezentující SVG pojem
V rámci importu SVG obrázku do MATLABu jsou v kódu specifikované proměnné, které reprezentují určitou informaci (většinou atribut) z formátu SVG. Nesou stejný název, jako měly v samotném zdrojovém kódu SVG. Některé proměnné v SVG obsahují znaky, které není možné v MATLABu přiřadit proměnným ("-","." apod.). Tyto znaky jsou vynechány.

Metody a funkce
Metody a funkce značíme stejně jako proměnné, tedy *camelFormátem* (první písmeno malé, další velké). Od proměnných jsou v kódu odlišovány tak, že mají argument v závorkách. V textu práce budeme pro odlišení proměnných od funkcí za funkce uvádět prázdné závorky. (např. *plot()*)

Struktury
MATLAB umožňuje vytvářet několik datových typů, jež obsahují data uložená v nějaké struktuře. Používané jsou *struct* a *cell*. Tyto struktury značíme prvním velkým písmenem a každým dalším písmenem v názvu velkým bez mezer.

MATLAB umožňuje zapisovat data do dvou typů vektorů (*row* a *column*).

$$row\ vector = [x_1 \quad x_2 \quad \dots \quad x_n], colum\ vector = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix} \tag{3.1}$$

Aby se předcházelo případným nesrovnalostem a chybám v zápisu kódu, je těmto dvěma typům vektorů přidělen jiný způsob použití.

Row vektor
Jedná se o vektor obsahující číselné hodnoty. Jako příklad můžeme uvést vektor souřadnic, vektor poloh objektu a podobně.

Column vektor
Column vektor je seznam obsahující nečíselné informace. Příkladem může být seznam elementů SVG v daném souboru, seznam instrukcí a podobně.

Při vývoji kódu budou jeho jednotlivé funkce rozděleny do dvou tříd. Představíme je, nastíníme jejich funkcionalitu a účel jejich implementace.

SVGObject
Hlavní třída nástroje, *SVGObject*, má dvě hlavní funkcionality. Jeho první funkcí je poskytnout uživateli možnost obsluhovat nástroj a tím spustit i všechny implementované algoritmy. Druhou funkcí je vytvořit objekty této třídy. Ty budou sloužit jakožto stavební bloky, z nichž bude vytvořena struktura stromu SVG již v MATLABu. Každý vytvořený objekt této třídy bude reprezentovat jeden uzel SVG stromu, v objektu tedy budou uloženy všechny informace, které má daný uzel obsahovat.

Nejpodstatnější proměnnou těchto objektů je seznam *ChildNodes*, jedná se o proměnnou typu *Cell*. Obsahuje všechny uzly, jež jsou potomky uzlu, který je daným objektem reprezentovaný. Skládáním těchto bloků tedy vytvoříme požadovanou strukturu stromu (viz obr. 3.1).

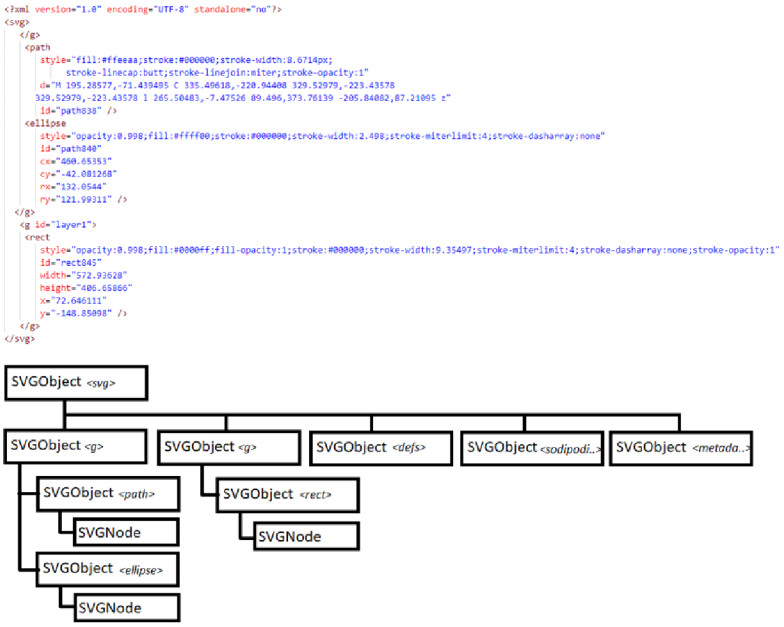
Druhou významnou proměnnou je *SVGNode*. Jak napovídá název, objekt je stejnojmenné třídy. Tato proměnná je specifická pro grafické uzly, pro ostatní uzly zůstává prázdná.

Zdrojový kód tohoto objektu nalezneme v Příloze 1.

SVGNode
Druhá třída vytvořená pro nástroj je *SVGNode*. Objekty této třídy budou vytvořeny pro specifické uzly, obsahující grafickou informaci (které uzly jsou grafické, je rozebráno v kapitole 2.3.2). Tento objekt obsahuje metodu *plot(),* pomocí které bude možno vykreslit daný grafický uzel.

Na následujícím příkladu můžeme vidět, jak nástroj převede SVG soubor do stromu pomocí uvedených dvou tříd.

Zdrojový kód tohoto objektu nalezneme v Příloze 2.



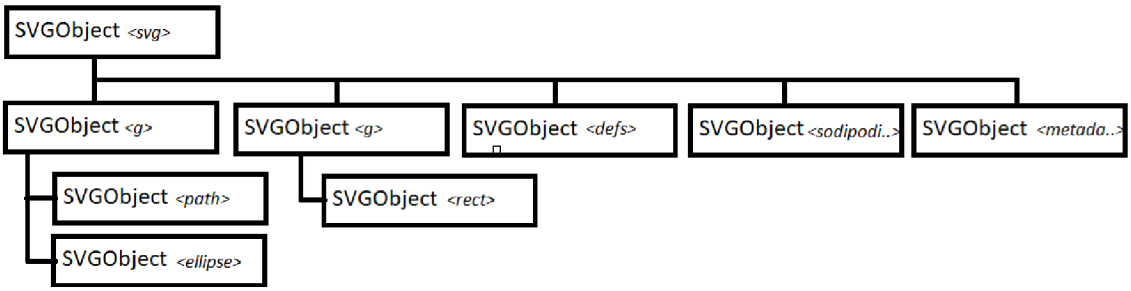
Obrázek 3.1: Reprezentace SVG souboru v MATLABu

3.3. NÁVRH NÁSTROJE PRO IMPORT SVG

Tato kapitola se bude zabývat podstatnými částmi ve vývoji nástroje pro import SVG obrázků do MATLABu. Tento nástroj vezme jakožto uživatelský vstup soubor SVG, uloží jej do prostředí MATLAB a umožní uživateli zobrazit jej pomocí grafických nástrojů MATLAB (funkce *plot()*).

3.3.1. PARSOVÁNÍ STRUKTURY

Prvním krokem k úspěšnému vytvoření chtěného nástroje je převést data o SVG obrázku do MATLABu a vytvořit z nich strukturu, s níž bude možno dále pracovat. Cílem této konverze je vytvořit ze struktury XML v SVG obrázku strom, který bude tvořen v MATLABU definovanými objekty, každý objekt bude reprezentovat jeden element (neboli uzel stromu) SVG souboru. Usnadníme tak další zpracovávání dat a jejich následné vykreslení. Tento strom budou tvořit námi definované objekty *SVGObject*, přičemž každému uzlu v XML struktuře bude přiřazen vlastní objekt. Tyto objekty budou uloženy do objektů předcházejících ve struktuře stromu, čímž se vytvoří struktura stromu v MATLABu.



Obrázek 3.2: Struktura stromu v MATLABu vytvořená objekt SVGOnject

Navrhneme tedy algoritmus, který vezme soubor SVG a vytvoří požadovaný strom, inspirujeme se funkcí *xml2struct*() (viz kapitola 2.4.1) a k analýze využijeme princip rekurzivní funkce. Algoritmus vytvoříme implementováním dvou metod pro třídu *SVGObject*. První metodou je konstruktor objektu (*Příloha 1: SVGObject řádek 513*), který slouží jakožto začátek rekurzivní analýzy XML struktury, a zároveň je zamýšleno, aby sloužil jako funkce, která uživateli umožní nástroj spouštět.

```
function obj = SVGObject(file,createBodiesStrut,attributes,printable)
    % reads the svg to dom object
    % function gets the file name or directly the Node
    %and reads the XML structur
    if ischar(file)%if file name use xml read to get first node
        xDoc=xmlread(file);
        Node=xDoc.getDocumentElement;
    else %if Node use it imidietly
        Node=file;
    end
    obj.printable=printable;
    %start scanning the Nodes
    scanNode(obj,Node);
    %Creates 'bodes' struct
    if createBodiesStrut
       obj.createBodiesStrut(obj);
    end
end
```

Obrázek 3.3: Konstruktor objektu SVGObject

Druhou implementovanou metodou je *scanNode()* (*Příloha 1: SVGObject řádek 38*), jež zpracuje každý jednotlivý uzel.

```
function scanNode(obj,Node)
   saveNode(obj,Node); %calles function to safe the node as SVGNode object to this SVGObject
   Childrens=Node.getChildNodes;
   n = Childrens.getLength;
      for i = 0:(n-1) %!indexing from java start at 0! goes thru all childs
            name=char(toString(Childrens.item(i).getChildNodes.getNodeName));
            if~isempty(newname) %we care only about nodes with name
                %reculsive start of new subobject
                %paste the node and the attributes that it inhirite
                obj.ChildNodes.(name)=SVGObject(Childrens.item(i).getChildNodes,false,obj.Attributes,obj.printable);
            end
      end
  end
end
```

Obrázek 3.4: Metoda scanNode()

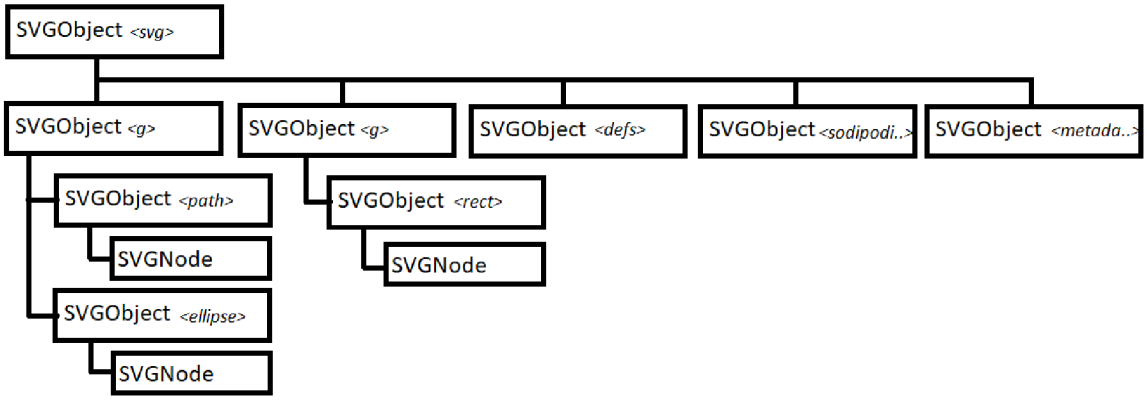
K získání informací o uzlech jsou použity funkce k práci s objektem DOM. Tato funkce zároveň získá přístup ke všem následujícím uzlům zpracovávaného uzlu. Pro každý následující uzel je znovu volán konstruktor objektu, čímž probíhá rekurze. Pokud uzel nemá žádné následující uzly, rekurze je ukončena.

Pro samotné uložení dat do MATLABu je v třídě *SVGObject* definována další metoda, *saveNode()*, která uloží všechny potřebné informace do příslušných objektů. Tato metoda uloží do proměnné *ChildNodes* (proměnná objektu *SVGObject*) všechny objekty, které jsou objekty následujícími ve struktuře stromu.

3.3.2. IMPLEMENTACE VYKRESLENÍ SVG PRVKŮ V MATLABU

V kapitole 2.3.2 bylo vysvětleno, že některé elementy SVG souboru jsou grafické. Vykreslení celého obrázku docílíme postupným vykreslením všech těchto grafických elementů. Zatím není řešeno, jak tyto elementy správně vykreslit dohromady (tím se zabývá následující kapitola), ale pouze jednotlivá vykreslení každého z nich.

Dalším krokem je tedy vytvořit způsob, jak tyto elementy zobrazit. Toho docílíme vytvořením objektů *SVGNode* pro každý grafický uzel stromu. V minulé kapitole jsme ukázali, jak nástroj dokáže vytvořit strukturu stromu z objektů *SVGObject*, ve třídě *SVGObject* definujeme proměnnou *NodeObject*, která bude obsahovat objekt *SVGNode* v případě, že je daný uzel stromu grafický.



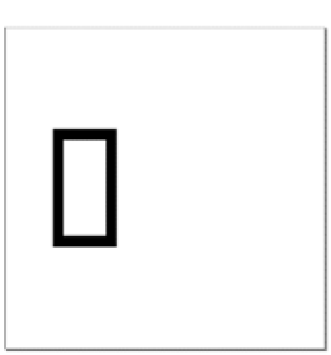
Obrázek 3.5: Reprezentace SVG souboru v MATLABu

Objekt *SVGNode* bude obsahovat všechny informace k úspěšnému vykreslení daného elementu v MATLABu. Abychom mohli vykreslit příslušný grafický útvar, musí třída *SVGNode* obsahovat algoritmus, který pro tyto elementy vytvoří vektor souřadnic, jež budou následně pomocí příkazu *plot()* v MATLABu zobrazeny. Konstruktor tohoto objektu (*Příloha 2: SVGNode řádek 70*) funguje jakožto stavový automat, který na základě typu elementu spustí příslušnou část algoritmu pro vytvoření souřadnic.

```
function obj = SVGNode(attributecell,name)
 scanAttributes(obj,attributecell);
 %based on node type set coordinates proper way
 switch name
       case "path"
         %do stuff to create coordinates
       case "line"
         %do stuff to create coordinates
       case "circle"
         %do stuff to create coordinates
       case "ellipse"
         %do stuff to create coordinates
        case "rect"
        %do stuff to create coordinates
 end
 %apply tranform matrix
 %after creation of coordiantes needs to be transformed be
 %attribute transform
  %patch
  obj.patchX=temp(1,:);
  obj.patchY=temp(2,:);
  %line
  temp=obj.M*[obj.coordiantesX;obj.coordiantesY;ones(size(obj.coordiantesX))];
  obj.coordiantesX=temp(1,:);
  obj.coordiantesY=temp(2,:);
end
```

Obrázek 3.6: Konstruktor objektu SVGNode

Pro lepší pochopení ukážeme příklad, jak jsou tyto souřadnice vytvořeny pro element <rect> (viz obrázek 3.7).



```
1   <svg viewBox="0 0 30 30">
2     <rect fill = "none"
3           stroke="black"
4           x="5" y="10"
5           width="5"
6           height="10"/>
7   </svg>
```

Obrázek 3.7: Příklad elementu <rect>

Z popisu tohoto elementu vidíme, že jej reprezentuje obdélník, jehož levý horní roh se nachází na souřadnicích $x = 5$, $y = 10$, jeho výška je 10 a šířka 5. Na základě těchto informací budou algoritmem vytvořeny vektory souřadnic x = [5, 10, 10, 5], y = [10, 10, 20, 20], které

po vykreslení vytvoří obrázek obdélníku v MATLABu. Všechny implementované algoritmy pro vykreslení jednotlivých elementů nalezneme v příloze ve zdrojovém kódu nástroje (*Příloha 2: SVGNode*).

Pro třídu *SVGNode* implementujeme metodu *plot() (Příloha 2: SVGNode řádek 600) řádek 38*, tato metoda již vykreslí daný element v MATLABu. Jelikož SVG elementy obsahují jak ohraničení objektu čárou, tak barevnou výplň, je toto vykreslení uskutečněno voláním funkce *line()* pro zobrazení ohraničení a funkce *fill()* pro výplň.

```
function handels=plot(obj)
    if ~isempty(obj.stroke)
    LINES=line(obj.coordiantesX,obj.coordiantesY);
    set(LINES,'LineWidth',obj.stroke_dash_width);
    else
    LINES=[];
    end
    PATCH=fill(obj.patchX,obj.patchY,'r');
    set(PATCH,'LineStyle','none') %patches dont have lines
    set(PATCH,'FaceAlpha',obj.fillAlpha);
    if nargout>0
      handels=[PATCH;LINES];
    end
end
```

Obrázek 3.8: Metoda plot() objektu SVGNode

Jak bylo uvedeno v kapitole 2.3.4., SVG soubor obsahuje velké množství atributů, které obrázek nějak upravují, ne všechny jsou v tomto nástroji implementované. Ve zdrojovém kódu nástroje vidíme, které atributy jsou ignorovány a které jsou dále zpracovány. Všechny důležité atributy jsou však implementovány, to by mělo zajistit co nejpřesnější vykreslení.

3.3.3. VYTVOŘENÍ FINÁLNÍ STRUKTURY OBRÁZKU ULOŽENÉHO V MATLABU

Program tedy dokáže vykreslit jakýkoli grafický element, který je definovaný v souboru SVG. Abychom však vykreslili obrázek jakožto celek, je ještě potřebné, aby byla v nástroji implementována funkce, umožňující uživateli bez porozumění vnitřní struktury objektů tento obrázek zobrazit.

Navrhneme tedy metodu *createBodiesStruct() (Příloha 1: SVGObject řádek 197),* která projde strom uložený v MATLABu, nalezne všechny grafické uzly (elementy) a vytvoří novou strukturu *Bodies*, jež bude obsahovat již kompletní informace o obrázku. Tato metoda bude volána v konstruktoru objektu *SVGObject* jakožto poslední krok konverze obrázku do

MATLABu. Struktura již bude brát v úvahu i následující animování a seskupí objekty *SVGNode* tak, aby bylo následné animování co nejsnadnější.

svg.Bodies

svg.Bodies

Fields	id	nodesObject	joints	parent	point	pointPath	path	currentPos
1	1	2x1 cell	1x2 struct	[]	[]	[]	[]	[]
2	2	2x1 cell	1x1 struct	[]	[143.6915,3...	595x2 double	1x600 double	15.1571
3	3	3x1 cell	[]	[]	[159.3986,1...	595x2 double	1x600 double	-16.3798
4	4	1x1 cell	[]	1	[159.6208,1...	595x2 double	1x600 double	24.7913

Obrázek 3.9: Struktura Bodies v objektu SVGObject

Tato struktura rozdělí obrázek do několika těles a ke každému z nich přiřadí ty objekty SVGNode, ze kterých se dané těleso bude skládat (podle čeho jsou objekty rozděleny, bude vysvětleno v kapitole 3.4.2). Struktura obsahuje dvě pole, která jsou pro vykreslení obrázku důležitá. Jsou jimi pole ‚*id*', jež identifikuje příslušný objekt, a pole ‚*nodesObject*', ve kterém jsou uloženy objekty SVGNode, z nichž se skládá příslušné těleso. Ostatní pole jsou důležitá pro animování a jejich účel bude objasněn později.

Samotné vykreslení obrázku bude umožněno metodou *plot() (Příloha 1: SVGObject řádek 543)* implementovanou pro objekt *SVGObject*, kterou může volit uživatel. Tato metoda jednoduše vykreslí tělesa definovaná ve struktuře Bodies ve správném pořadí voláním funkce plot (definované pro *SVGNode*). Aby velikost obrázku byla korektní, je ještě potřeba nastavit správné velikosti jeho ohraničení, toho je docíleno pomocí nastavení velikosti figure, do kterého bude vykreslení probíhat na velikosti viewBoxu definovaného v root elementu příslušného SVG souboru (kapitola 2.3.1).

```
function plot(obj)
    for k=1:length(obj.Bodies)
        if isempty(obj.bodyOrder)
        obj.bodyOrder=1;
        end
        Nodes=obj.Bodies(obj.bodyOrder(k)).nodesObject;
        for i=1:length(obj.Bodies(obj.bodyOrder(k)).nodesObject)
           Nodes{i}.plot;
        end
    end
end
```

Obrázek 3.10: Metoda plot() objektu SVGObject

3.4. NÁVRH NÁSTROJE PRO ANIMOVÁNÍ

V této kapitole bude vysvětleno, jak funguje nástroj pro animování importovaných obrázků. Na rozdíl od samotného importu obrázku bude tento nástroj pracovat s rozmanitějším uživatelským vstupem. Nejprve objasníme, jak pracuje algoritmus, který vytvoří animaci specifikovanou uživatelem, načež ukážeme, jak připravit a definovat uživatelský vstup.

3.4.1. PRINCIP ANIMOVÁNÍ

Animace se skládá z postupně vykreslených snímků. Nástroj bude vykreslovat jednotlivé snímky implementovanou metodou *plotFrame() ()(Příloha 1: SVGObject řádek 349)*:

```
function plotFrame(obj)
    for k=1:length(obj.Bodies)
        Nodes=obj.Bodies(obj.bodyOrder(k)).nodesObject;
        for i=1:length(obj.Bodies(obj.bodyOrder(k)).nodesObject)
           Nodes{i}.plot;
        end
    end
end
```

Obrázek 3.11: Metoda plotFrame()

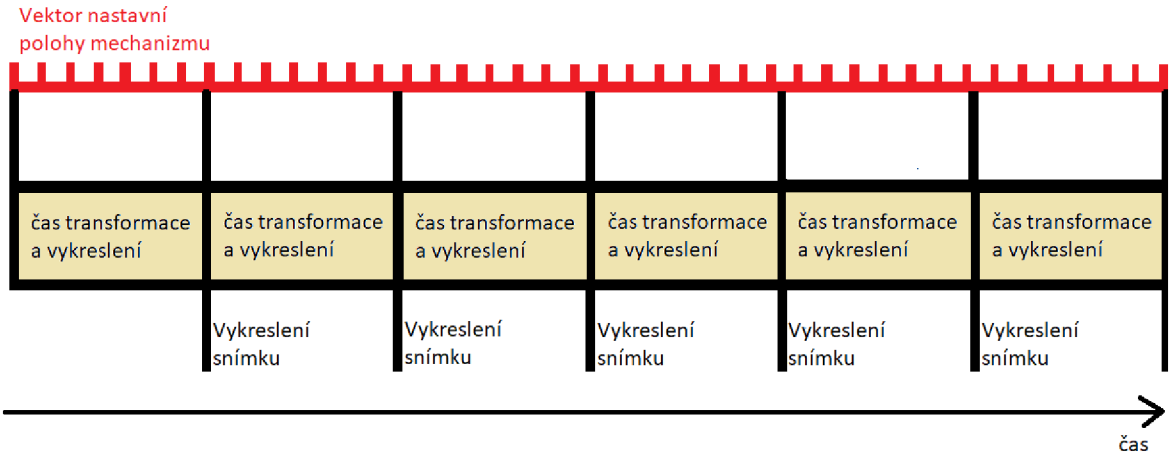
Tato funkce bude opakovaně volána a pokaždé vykreslí obrázek v jiné poloze, čímž bude docíleno zobrazení plynulého pohybu obrázku. Funkce *plotFrame()* sama o sobě pouze zobrazí obrázek definovaný ve struktuře Bodies (viz 3.4.1). Abychom tedy docílili požadované animace, musíme před každým vykreslením obrázek transformovat do polohy, která odpovídá příslušnému časovému okamžiku.

Animace bude spuštěna voláním while loopu (součást funkce *startAnimation()*). Před jeho voláním spustíme timer pomocí funkce *tic*. V tomto loopu bude volána funkce *toc*, jež vrátí aktuální čas. Pro tento čas najdeme příslušnou pozici, ve které se má v tento časový okamžik obrázek nacházet. Pomocí funkce *moveObject()() (Příloha 1: SVGObject řádek 564)* přetransformujeme obrázek do podoby, která odpovídá tomu, jak má obrázek vypadat ve zvoleném časovém okamžiku. Nový přetransformovaný obrázek vykreslíme funkcí *plotFrame()*. Jak funguje funkce *moveObject()*, bude objasněno v kapitole 3.4.3.

```
tic
xtoc=toc;
while(xtoc<obj.time(end))
     position=find(abs(obj.time-xtoc)==min(abs(obj.time-xtoc)));
     if(position~=lastPosition)
         obj.moveObjects(xtoc,xtoc,position)
         obj.plotFrame()
         pause(0.000001)
         lastPosition=position;
     end
     xtoc=toc;
end
```

Obrázek 3.12: While loop volající jednotlivé snímky animace

Jelikož před každým vykreslením snímku probíhá transformace mechanismu do požadované podoby, každé další vykreslení proběhne až po určité době, odpovídající času transformace a vykreslení snímku. Tento čas se nedá předpovídat. Proto aby animace probíhala ve správné podobě, před každým vykreslením proběhne vyhledání správné polohy. To můžeme prezentovat následujícím obrázkem.



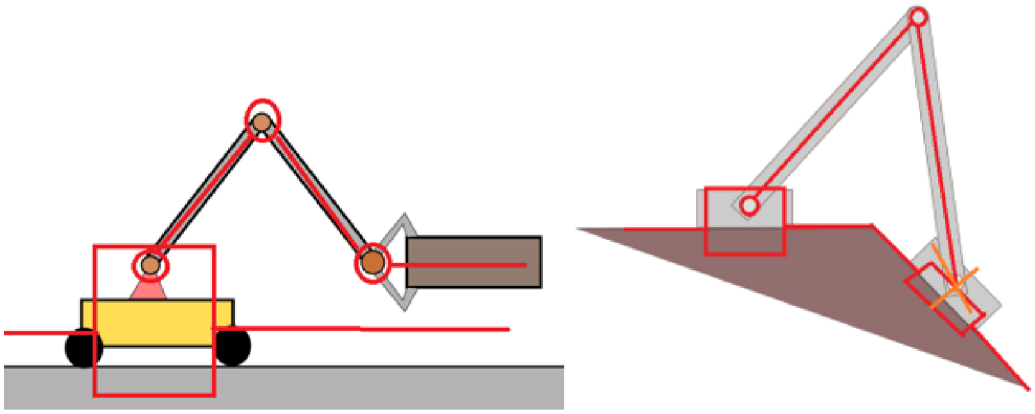
Obrázek 3.13: Diagram vyhledávání správné polohy mechanismu při animování

Program pracuje s vektorem nastavení mechanismu. Při každém vykreslení program najde správnou polohu a začne ji transformovat a vykreslovat. Po dokončení vykreslení vyhledá novou pozici, jež začne být vykreslována. Z tohoto plyne omezení frameratu animací v závislosti na časové náročnosti transformace a vykreslení daných snímků.

3.4.2. ANIMOVATELNÉ MECHANISMY

Aby nástroj dokázal příslušné mechanismy animovat, musí být při jejich definici dodržena jejich správná struktura. V závislosti na způsobu, jakým nástroj transformuje celý mechanismus při vytváření pohybu, je možné vytvářet animace pouze pro mechanismy, které neobsahují žádný uzavřený řetězec kinematických dvojic. Proto při vytváření obrázku bude muset uživatel dbát na to, aby mechanismus tvořil správným způsobem.

Pokud tedy definujeme mechanismus způsobem, že nevznikne žádná uzavřená smyčka, vytvoří se z jednotlivých těles jakýsi strom. V něm bude první těleso tělesem základním a na něj budou pomocí vazeb připevněna tělesa další. Na každé těleso (včetně základního) můžeme připevnit libovolný počet dalších těles. Můžeme tedy v tomto stromu definovat vztah mezi tělesy každé kinematické dvojice jakožto tělesa předcházejícího a tělesa následujícího (*parent* x *child*). Z toho plyne, že těleso, které je ve struktuře blíže tělesu základnímu, je těleso předcházející a těleso připevněné přímo na toto těleso je tělesem následujícím.



Obrázek 3.14: Naznačení vytvořených řetězců z animovatelných obrázků

V praxi však bude uživatel chtít v mnohých případech animovat i mechanismy, které obsahují uzavřenou smyčku. V tomto případě musí uživatel v mechanismu přerušit některé vazby, aby byla smyčka odstraněna.

Takto definovaný mechanismus má počet stupňů volnosti roven počtu těles (bez základního tělesa). Abychom mohli definovat polohu tohoto mechanismu, musíme specifikovat nastavení úhlu a posunutí v jednotlivých vazbách. Jak má uživatel nastavit tyto polohy vazeb, je vysvětleno v kapitole 3.4.6.

3.4.3. TRANSFORMACE OBRÁZKU PŘI POHYBU

V kapitole 3.4.1 jsme objasnili, jak nástroj vytváří animaci pomocí postupného zobrazení polohy animovaného obrázku. K vytvoření pohybu je potřeba tento obrázek – před vykreslením každého snímku – přetransformovat do podoby, která bude odpovídat poloze obrázku v daný časový okamžik. Toto přetransformování umožňuje metoda *moveObject()* definovaná pro třídu *SVGObject*.

```
function moveObjects(obj,~,~,positionInQuee)

  for i=1:length(obj.Bodies) %goes thru all bodies and does their movement
  if ~isempty(obj.Bodies(i).joints)%if has any joints moves each child body

       for k=1:length(obj.Bodies(i).joints)%goes thru all the jointst and does their movement
          switch obj.Bodies(i).joints(k).type

              case "rot"
               ID=obj.Bodies(i).joints(k).child;
               x=obj.Bodies(i).joints(k).x;
               y=obj.Bodies(i).joints(k).y;
               angel=obj.Bodies(ID).path(positionInQuee)-obj.Bodies(ID).currentPos; %find proper angel to rotate
               obj.rotateAroundPoint(ID,x,y,angel);

              case "shift"
               Xposition=diff(obj.Bodies(i).joints(k).x);
               Yposition=diff(obj.Bodies(i).joints(k).y);
               V=[Xposition,Yposition];
               unitV = V/norm(V);
               ID=obj.Bodies(i).joints(k).child;
               shiftBy=obj.Bodies(ID).path(positionInQuee)-obj.Bodies(ID).currentPos;
               obj.shifts(ID,unitV(1)*shiftBy,unitV(2)*shiftBy);
          end
       end
  end
  end
end
```

Obrázek 3.15: Metoda moveObjects()

Tato metoda potřebuje jakožto argument hodnotu času, do kterého chceme animovaný obrázek přetvořit. Metoda najde hodnotu nastavení jednotlivých vazeb v určeném časovém okamžiku a pro každou vazbu zavolá funkci *rotateAroundPoint()* (pro rotační vazbu) nebo funkci *shifts()* (pro posuvnou vazbu).

Funkce *rotateAroundPoint() (Příloha 1: SVGObject řádek 433)* a *shifts() (Příloha 1: SVGObject řádek 463)* používají maticové transformace (viz 2.3.6) k tomu, aby přesunuly souřadnice všech těles, která jsou ve struktuře pod touto vazbou do příslušné pozice.

3.4.4. UŽIVATELSKÝ VSTUP

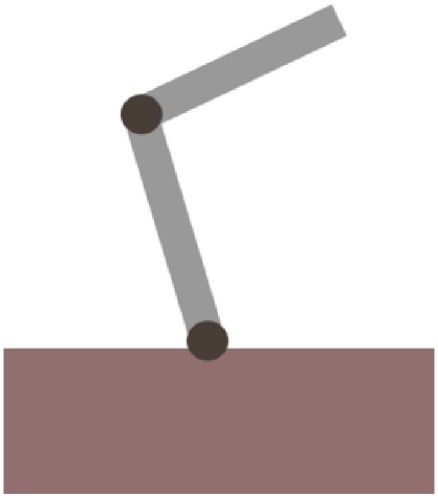
Aby nástroj dokázal z obrázku ve formátu SVG vytvořit chtěnou animaci, musí uživatel poskytnout obrázek ve správné podobě. Musí také k tomuto obrázku doplnit informace o poloze vazeb a definici pohybu jednotlivých těles.

Tělesa

Každý mechanismus se skládá z těles. Úkolem tohoto nástroje je animovat mechanismus, který je reprezentovaný obrázkem ve formátu SVG. Aby bylo možno docílit správné animace, musí být tento obrázek rozdělen do částí, které budou odpovídat jednotlivým tělesům, v každé z těchto částí bude celé těleso popsáno. Jak víme z kapitoly 2.3.2, obrázek ve formátu SVG se skládá z grafických elementů. Logicky tedy každé těleso budeme chtít reprezentovat určitými grafickými elementy. Pro příklad uvedeme jednoduchý manipulátor.

```
<svg>
  <g teles="1">
    <rect
       style="fill:#916f6f;stroke:none;stroke-width:4.13545"
       width="211.28868" height="52.916672"
       x="0.75595242" y="197.68155" />
  </g>
  <g teles="2">
    <rect
       style="opacity:0.998;fill:#999999;stroke:none;stroke-width:5.6896"
       width="10.423512" height="67.619186"
       x="29.684265" y="144.94902"
       transform="rotate(-16.628382)" />
    <ellipse
       style="opacity:0.998;fill:#483e37;stroke:none;stroke-width:4.31357"
       cx="94.680237" cy="195.44084"
       rx="6.214016" ry="5.9467468" />
  </g>
  <g teles="3">
    <rect
       style="opacity:0.998;fill:#999999;stroke:none;stroke-width:5.6896"
       width="10.423512" height="67.619186"
       x="142.2406" y="-78.314743"
       transform="rotate(64.594701)" />
    <ellipse
       style="opacity:0.998;fill:#483e37;stroke:none;stroke-width:4.31358"
       cx="74.969101" cy="127.75484"
       rx="6.214016" ry="5.9467468" />
  </g>
</svg>

```



Obrázek 3.16: Příklad rozdělení SVG obrázku do vrstev těles

Uvedený manipulátor sestává ze základního tělesa a dvou dalších těles. Naše reprezentace tohoto manipulátoru v SVG se skládá z několika grafických elementů, které vidíme ve zdrojovém kódu tohoto obrázku. Jednotlivé elementy tedy chceme uskupit do skupin, jež budou reprezentovat jednotlivá tělesa.

Abychom nástroji řekli, ke kterému tělesu, které grafické elementy patří, využijeme SVG element <g> neboli rozdělíme obrázek do vrstev tak, aby každá z nich správně reprezentovala konkrétní těleso v mechanismu. Tyto vrstvy musíme příslušně označit, tak aby algoritmus v nástroji dokázal grafické elementy správně roztřídit. Tohoto označení docílíme přiřazením atributu ‘těleso‘ k jednotlivým elementům <g>. Tento atribut bude obsahovat číselnou hodnotu *1*, *2*, *3*… *n*, která specifikuje, o jaké těleso se jedná.

Algoritmus v objektu *SVGObjekt* bude při analýze XML struktury vyhledávat takto označené vrstvy a přiřadí k elementům v nich příslušné ID, které umožní následné rozdělení těles.

Jelikož je nepraktické definovat tělesa pomocí editace zdrojového kódu, využijeme k rozdělení obrázku SVG do vrstev existující software k editaci SVG obrázků. Tyto SVG editory nám umožňují rozdělovat obrázek do vrstev a přiřazovat k nim jejich označení. Každý editor ale označuje vrstvy jiným atributem. (V příkladu jsme jednotlivé vrstvy pojmenovali přiřazením atributu ‘těleso‘. Editory nám sice umožňují pojmenovat jednotlivé vrstvy, ale reprezentace názvu v SVG kódu je odlišná, např. pro Inkscape se tento atribut jmenuje ‚*inkscape:label*‘.) Jelikož jsme jakožto podporovaný editor pro tento nástroj zvolili Inkscape (viz 2.3.5), algoritmus sám rozpozná pouze tělesa, jež jsou definována v něm.

Vazby
Dalším krokem po definici jednotlivých těles je definování polohy příslušných vazeb. Ty spojují vždy dvě tělesa do kinematické dvojice. Jelikož musí být mechanismus definován v podobě stromu, každá vazba obsahuje informaci o tom, které z těles příslušné kinematické dvojice je ve stromu tělesem předcházejícím a které tělesem následujícím. K možnosti animace jakéhokoli 2D mechanismu podporuje nástroj rotační a posuvné vazby.

Vazby lze v nástroji zadávat funkcí *setJoint() (Příloha 1: SVGObject řádek 594)*. Tato funkce vytvoří specifikovanou vazbu mezi dvěma tělesy.

```
svg.setJoint(1,2,"rot",5,5)
```

Obrázek 3.17: Vytvoření rotační vazby mezi tělesy 1 a 2 v bodě [5, 5]

Rotační vazba
Rotační vazba je v nástroji značená typem „*rot*“. Definuje ji jeden bod se souřadnicemi *x*, *y*, který specifikuje bod, okolo kterého se bude těleso ve struktuře následně pohybovat.

Posuvná vazba
Posuvná vazba je v nástroji značená typem „*shift*“. K její definici jsou potřeba dva body, jež určují směrový vektor. Ve směru tohoto vektoru se těleso následující bude posouvat vůči tělesu předcházejícímu.

Vetknutí
Vetknutí není v nástroji specifikované, lze ho však docílit definicí vazby rotační nebo posuvné a následným definováním nulového pohybu. To znamená, že tělesa v této kinematické dvojici se vůči sobě nebudou pohybovat neboli jsou vůči sobě vetknuté.

Tyto vazby jsou definovány relativně k předcházejícímu tělesu příslušné kinematické dvojice. To znamená, že pokud se předcházející těleso bude samo pohybovat, bude se pohybovat i pozice příslušné vazby.

Fields	type	parent	child	x	y
1	"shift"	1	4	[11.0517,78....	[122.3994,123.1485]
2	"rot"	1	2	215	135
3					

Obrázek 3.18: Uložení vazeb v prostředí MATLAB

Jak tyto vazby sepcifikovat pomocí softwaru Incscape, je vysvětleno v kapitole 3.4.6.

Definice časového rozsahu animace
Aby algoritmus pro animování popsaný v podkapitole 3.4.1. fungoval správně, je potřeba nástroji říct, jaký je rozsah animace. Toho docílíme definováním vektoru časových okamžiků neboli vytvořením vektoru obsahujcího hodnoty času v sekundách. Vektor musí obsahovat

po sobě jdoucí řadu čísel, přičemž jeho poslední hodnota označuje celkovou dobu animace. Počet členů ve vektoru a jejich rozpětí určuje maximální framerate animace.

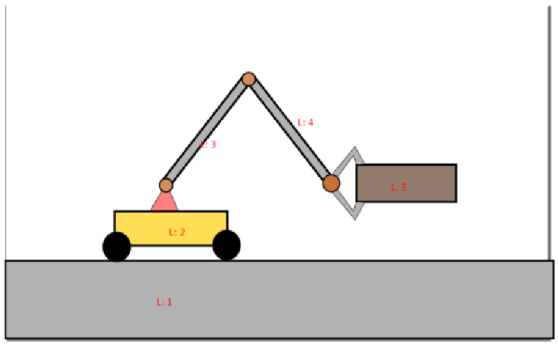
Definice pohybu
Poslední informací, kterou musí uživatel nástroji poskytnout, je definice polohy jednotlivých těles. Nejlepším způsobem je definovat danou polohu jakožto nastavení jednotlivých vazeb. Uživatel tedy musí poskytnout hodnoty úhlů (pro rotační vazby) a posunutí (pro vazby posuvné) v každém časovém okamžiku animace. Řešení spočívá v definování vektorů s těmito hodnotami.

3.4.5. DEFINICE TĚLES POMOCÍ INKSCAPE

V minulé kapitole jsme ukázali, jak rozdělit SVG obrázek do částí, tak aby příslušné grafické elementy náležely chtěným tělesům pomocí editace zdrojového kódu. V této kapitole ukážeme, jak obrázek připravit v softwaru pro editaci SVG obrázků Inkscape.

Pro rozdělení těles využijeme možnost programu Inkscape pro práci s vrstvami. Abychom rozdělili obrázek do těles, musíme vytvořit pro každé těleso vlastní vrstvu. Tyto vrstvy pojmenujeme způsobem L: 1, L: 2, … a tak dále. Podrobněji je vše vysvětleno v návodu *(Příloha 4: Návod k použití nástroje)*.

Do každé z definovaných vrstev umístíme příslušné těleso (viz obr. 3.16).



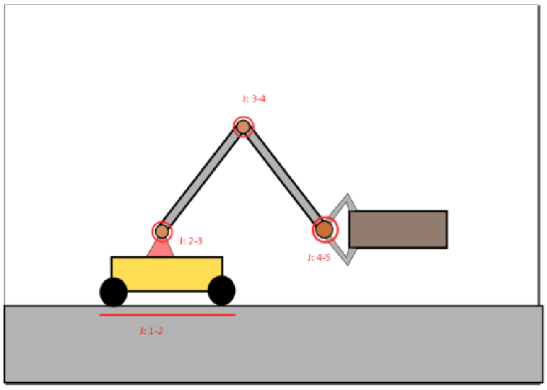
Obrázek 3.19: Demonstrace rozdělení obrázku mechanismu do vrstev

Algortimus při analýze SVG obrázku rozdělí grafické elementy do definovaných těles. Budou uloženy ve struktuře *Bodies*.

3.4.6. DEFINICE VAZEB POMOCÍ INKSCAPE

Jak jsme již objasnili, každá vazba obsahuje informaci, mezi kterými dvěma tělesy se nachází, které z těchto dvou těles je ve struktuře následující, které je předcházející a souřadnice vazeb relativně definované k předcházejícímu tělesu. V kapitole 3.4.2 jsme ukázali, jak definovat vazbu pomocí implementované funkce. Tato kapitola objasní, jak definovat typ vazby pomocí softwaru Inkscape.

Podobně jako při definici těles vložíme v Inkscape novou vrstvu pro každou vazbu. Tyto vrstvy pojmenujeme stylem J: 1-2, J:2-3, J: 2-4. Následně do těchto vrstev vložíme specifickou vazbu. Rotační vazbu definujeme jakožto střed kruhu a posuvnou vazbu jako přímku, která určuje směr posuvu této vazby. Do každé vazby umístíme pouze jeden z elementů, jinak při zpracovávání dojde k chybě.



Obrázek 3.20: Demonstrace vytvoření vazeb

Vrstvy, které jsou pojmenovány tímto stylem (J:1-2…) nebudou vykresleny v MATLABu, slouží pouze pro definování vazeb. Algoritmus při zpracovávání obrázku tyto vrstvy identifikuje a místo jejich vykreslení bude volat funkce *setJoint()* s příslušnými argumenty.

3.4.7. DEFINOVÁNÍ POHYBU A ČASOVÉHO ROZSAHU ANIMACE V MATLABU

Poslední krok k úspěšnému animování spočívá v definici časového rozsahu animace a pohybu jednotlivých těles. Pro jejich definici je v nástroji implementovaná funkce *setTime()* a *setMovement() (Příloha 1: SVGObject řádek 568).*

Funkce *setTime()* požaduje jediný argument, a to vektor časového rozsahu.

```
time = [0 2 4 6];
svg.setTime(time)
```

Obrázek 3.21: Specifikace časového rozsahu pro snímky v čase 0 2 4 a 6 vteřin

Tento vektor specifikuje každý časový okamžik animace. Ke každému z těchto okamžiků musíme později definovat i nastavení všech vazeb.

Do funkce *setMovement()* chceme vložit dva vstupy, a to *id* objektu, jehož pohyb budeme definovat, a samotný vektor polohy. Jelikož mechanismus musí být definovaný ve struktuře stromu, znamená to, že každé těleso je do struktury připevněno jednou vazbou. Pokud zadáváme pohyb tělesa, znamená to, že zadáváme hodnotu nastavení vazby, kterou je těleso

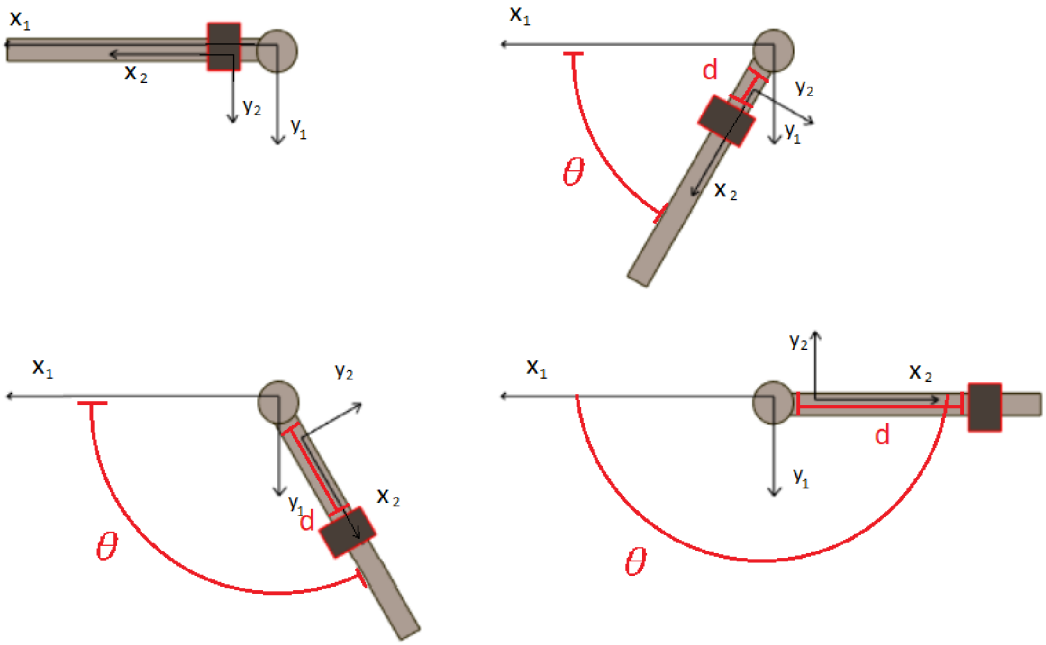
připevněno. Těleso může být připevněno rotační, nebo posuvnou vazbou. Pro tělesa připevněná rotační vazbou vkládáme jakožto argument polohu ve stupních natočení a pro posuvnou v jednotkách, jež byly použity při vytváření obrázku. Vektor pohybu musí mít stejný počet členů jako vektor časového rozsahu.

```
svg.setMovement(2,[0 -60 -120 -180]);
svg.setMovement(3,[0 10 30 50])
```

Obrázek 3.22: Specifikace vektorů pohybu pro tělesa 2 a 3

Pro příklad ukážeme animaci smýkadla. Tato animace se bude skládat pouze ze 4 snímků (pro názornost).

```
time = [0 2 4 6];
svg.setTime(time)
svg.setMovement(2,[0 -60 -120 -180]);
svg.setMovement(3,[0 10 30 50])
```



Obrázek 3.23: Demonstrace animace vytvořené použitím funkcí setTime() a setMovement()

3.4.8. FUNKCE POUŽÍVANÉ UŽIVATELEM

V minulých kapitolách byla představena podstata navrhovaných nástrojů. Tato kapitola shrne všechny funkce, které uživatel nástroje může používat, a definuje jejich očekávaný vstup.

svg = SVGObject(*filename*)
Zahájení konverze obrázku do prostředí MATLAB – tato funkce očekává jeden vstupní argument, jímž je samotný soubor ve formátu SVG, který budeme chtít vykreslit/animovat. Soubor, který je vykreslovaný, musí být v PATH cestě MATLABu, pokud je toto splněno, vložíme do této funkce název nástroje s koncovkou ve formátu string. Výstup z této funkce je objekt SVGObject, jenž reprezentuje daný SVG soubor v MATLABU.

```
FileName = 'smykadlo.svg';
svg=SVGObject(FileName);
```

Obrázek 3.24: požití konstruktoru SVGObject() k vytvořen objektu SVG

plot(*svg*)
Jedná se o funkci, která zobrazí SVG obrázek v MATLABU. Je definována pro objekt *SVGObjekt.* Tato funkce vykreslí do nového okna *figure* daný obrázek.

setTime(*svg,time*)
Tato funkce nastaví časové rozmezí animace. Její očekávaný vstup je vektor *time*, jenž má obsahově číselnou hodnotu času pro všechny animovatelné snímky ve vteřinách. Hodnota na posledním místě reprezentuje celkový čas animace a jeho délka reprezentuje maximální počet snímků animace.

```
time=linspace(0,5,600); %vektor 5 sekund s 600 famy
svg.setTime(time)
```

Obrázek 3.25: Přiřazení vekotu času metodou setTime()

setMovement(*svg,n ,path*)
Tato funkce definuje pohyb jednotlivého tělesa označeného s ID n. Očekává vstup čísla *n*, jenž představuje id tělesa, pro které zadáváme jeho pohyb, a dále vektor *path*. Tento vektor musí mít stejný rozměr jako vektor času *time* (nastavený funkcí *setTime()*). Jeho hodnoty odpovídají nastavením vazby, jíž je příslušné těleso *n* připevněno do struktury. Pro rotační vazbu je tato hodnota ve stupních a pro posuvnou je ve stejných jednotkách, které byly použity pro vytvoření obrázku.

```
path2=linspace(0,-180,600);
svg.setMovement(2,path2)
path3=linspace(0,50,600);
svg.setMovement(3,path3)
```

Obrázek 3.26: Přiřazení vektorů pohybu metodou setMovement()

startAnimation(*svg*)
Po správném nastavení vektoru času a vektorů pohybu nám tato funkce umožňuje spustit animaci.

3.4.9. VYTVOŘENÍ PŘÍKLADU ANIMACE

Tato kapitola naznačí, jak vytvořit mechanismus, který lze pomocí navrženého nástroje animovat. Přesnější popis tohoto vytvoření s podrobným výkladem jednotlivých kroků nalezneme v návodu (*Příloha 4: Návod k použití nástroje*). Uvedeme si seznam kroků, které je potřeba učinit.

1. **Vytvoření obrázku mechanismu v editoru Inkscape**
   Uživatel musí pomocí editoru Inkscape nakreslit obrázek mechanismu (za dodržování stanovených konvencí), to znamená každé těleso uložit do nově označené vrstvy.

2. **Definování vazeb v editoru Inkcape**
   Druhým krokem je definovat vazby tělesa pomocí vkládání vrstev do obrázku, s příslušným definováním těles vzniká i struktura mechanismu, musí se tedy brát zřetel na to, aby definovaný mechanismus neobsahoval uzavřený řetězec těles.

3. **Uložení vytvořeného obrázku do PATH cest MATLABu**
   Připravený obrázek mechanismu správně uložíme, tak aby k němu měl MATLAB přístup.

4. **Zadání časového rozsahu a polohových vektorů v MATLABu**
   Použitím vytvořených nástrojů zadáme časový vektor a vektory nastavení poloh jednotlivých vazeb.

Následně můžeme animaci vykreslit.

Pro demonstraci funkcionality nástroje bylo vytvořeno několik ukázkových příkladů, které nalezneme v Příloze 3. Tyto ukázkové příklady lze spustit pomocí souboru „prikladyRUN“.

4. Závěr

Práce se zabývala popisem vývoje nástroje pro import a animaci SVG obrázků v prostředí MATLAB. Popsala funkčnost formátu SVG (potažmo XML) ke zobrazování obrázků vektorové grafiky, představila, jak se v obrázku chovají jednotlivé elementy tohoto formátu. Definovány byly i základní kinematické principy, které jsou nezbytné k animaci mechanismů.

V praktické části byly navrženy dva nástroje (pro import a vykreslení SVG obrázku a pro jejich animaci) v programovacím jazyce MATLAB. Objasnění funkcionality těchto nástrojů byly věnovány kapitoly 3.3 a 3.4. V neposlední řadě byly vytvořeny ukázkové příklady, na kterých jsou demonstrovány funkcionality zmíněných nástrojů.

V nástroji pro import a vykreslení SVG obrázků byly implementovány funkce a algoritmy, schopné vykreslit většinu hlavních elementů tohoto souboru. Jelikož bylo zadání upraveno způsobem, který se soustředí více na obrázky vytvořené v programu Inkscape, byly optimalizovány převážně elementy formátu SVG, jež se nacházejí v obrázcích vytvořených tímto programem. Nástroj tedy dokáže vykreslit v prostředí MATLAB většinu obrázků, které mohou být v programu Inkscape vytvořeny. Celkově je nástroj schopen vykreslit obrázky v uspokojivé podobě v případě jakéhokoli obrázku (nevytvořeného přímo v programu Inkscape).

Pro nástroj k animaci mechanismů reprezentovaných SVG obrázkem bylo navrženo několik funkcí, za pomoci, kterých je uživatel schopen je jednoduše obsluhovat. Jelikož vytvoření pohyblivých mechanismů není triviální činností, byl zpracován návod, jak navržený nástroj pro tento úkon používat. Při dodržení postupů uvedených v tomto návodu by měl být uživatel schopen bez větších překážek vytvořit libovolnou animaci.

4.1. DALŠÍ PRÁCE

Jelikož v rámci SVG formátu existuje velké množství způsobů zápisu informací, příslušných elementů a atributů, nejsou v nástroji implementovány všechny. Je možné pokračovat v práci na algoritmech, které převádějí jednotlivé elementy do prostředí MATLAB, tak aby vykreslení SVG obrázků bylo přesnější.

Jak je vysvětleno v kapitole 3.4.1, počet snímků, a tedy plynulost animace závisí na rychlosti transformace těchto obrázků. V případě velkých obrázků jsou animace vykreslovány neplynule s nízkým frameratem. Lze pokračovat v optimalizaci animací, aby bylo možno vytvořit animace i z větších obrázků.

5. SEZNAM POUŽITÝCH ZDROJŮ

[1] TIM, Bray et al. Extensible Markup Language: W3C Recommendation. *W3C* [online]. 2008, 26 November [cit. 2021-5-13]. Dostupné z: https://www.w3.org/TR/xml/

[2] CHRISTENSSON, P. Markup Language. *TechTerms The Computer Dictionary* [online]. 2011, 1 June [cit. 2021-5-13]. Dostupné z: https://techterms.com/definition/markup_language

[3] W3C, XML Syntax Rules. *W3schools* [online]. [cit. 2021-5-13]. Dostupné z: https://www.w3schools.com/xml/xml_syntax.asp

[4] W3C, XML Elements. *W3schools* [online]. [cit. 2021-5-13]. Dostupné z: https://www.w3schools.com/xml/xml_syntax.asp

[5] GUNAWARDENA, Ananda. Tree Data Structure. *An Online Textbook for 15-111 Intermediate and Advanced Programming* [online]. 2007, 22 October [cit. 2021-5-13]. Dostupné z: https://www.cs.cmu.edu/~clo/www/CMU/DataStructures/Lessons/lesson4_1.htm

[6] MICHÁLEK, Ondřej. Stromové datové struktury. *ITnetwor.cz* [online]. [cit. 2021-5-13]. Dostupné z: https://www.itnetwork.cz/navrh/algoritmy/algoritmy-datove-struktury/stromove-datove-struktury

[7] W3C, XML Tree. *W3schools* [online]. [cit. 2021-5-13]. Dostupné z: https://www.w3schools.com/xml/xml_tree.asp

[8] KOUTNÁ, Marcela a Jana KAPOUNOVÁ. *Vektorová a rastrová grafika na PC*. Orlová: Obchodní akademie Orlová, 2007. ISBN 978-80-87113-18-9.

[9] DAHLSTRÖM, Erik et al. Scalable Vector Graphics: W3C Recommendation. *W3C* [online]. 16 August [cit. 2021-5-13]. Dostupné z: https://www.w3.org/TR/SVG11/

[10] BELLAMY-ROYDS, Amelia et al Paths. *SVG Working Group document repository* [online]. 2018, 7 August [cit. 2021-5-13]. Dostupné z: https://svgwg.org/svg2-draft/paths.html

[11] BELLAMY-ROYDS, Amelia et al Appendix G: Attribute Index. *SVG Working Group document repository* [online]. 2018, 7 August [cit. 2021-5-13]. Dostupné z: https://svgwg.org/svg2-draft/attindex.html

[12] ROKYTA, M. *RGB color coding* [online]. 1997 [cit. 2021-5-13]. Dostupné z: https://www2.karlin.mff.cuni.cz/~rokyta/leisure/rgb.htm

[13] BELLAMY-ROYDS, Amelia et al Coordinate Systems, Transformations and Units. *SVG Working Group document repository* [online]. 2018, 7 August [cit. 2021-5-13]. Dostupné z: https://svgwg.org/svg2-draft/coords.html#TransformProperty

[14] MARGALIT, Dan a Joseph RABINOFF. Matrix Transformations. *Interactive Linear Algebra* [online]. 2019, 3 June [cit. 2021-5-13]. Dostupné z: https://textbooks.math.gatech.edu/ila/matrix-transformations.html

[15] FALKENA, Wouter. Xml2struct. *MathWorks: File exchange* [online]. 2012, 15 May [cit. 2021-5-13]. Dostupné z: https://www.mathworks.com/matlabcentral/fileexchange/28518-xml2struct

[16] BODNAR, Jan. Java DOM. *ZetCode* [online]. 2020, 6 July [cit. 2021-5-13]. Dostupné z: https://zetcode.com/java/dom/

[17] KATZ, Michael. Using XML in MATLAB. *MathWorks: Blogs* [online]. 2010, 28 June [cit. 2021-5-13]. Dostupné z: https://blogs.mathworks.com/community/2010/06/28/using-xml-in-matlab/

[18] PŘIKRYL, Karel. *Kinematika*. 3. Brno: Akademické nakladatelství CERM, 2003. ISBN 80-214-2412-5.

[19] VECTOR-CONVERSION.COM. Raster vs Vector. Vector Conversion [online]. [cit. 2021-5-19]. Dostupné z: http://vector-conversions.com/vectorizing/raster_vs_vector.html

[20] FIZZ. Perplexed by SVG viewBox, width, height, etc. Stack Overflow [online]. 2014, 1 August [cit. 2021-5-19]. Dostupné z: https://stackoverflow.com/questions/14553392/perplexed-by-svg-viewbox-width-height-etc

6. SEZNAM PŘÍLOH

Příloha 1: Zdrojový kód SVGObject

Příloha 2: Zdrojový kód SVGNode

Příloha 3: Nástroj pro import a animaci obrázků ve formátu SVG s příklady

Příloha 4: Návod k použití nástroje