VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

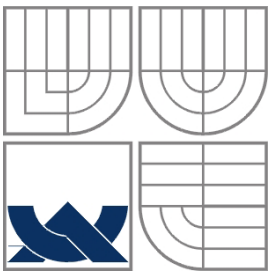
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

# VIZUALIZACE A UŽIVATELSKÉ ROZHRANÍ PRO ŘÍDICÍ SYSTÉM DIVADELNÍHO JEVIŠTĚ

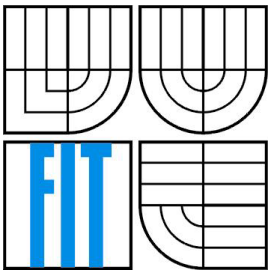
DIPLOMOVÁ PRÁCE
MASTER‘S THESIS

AUTOR PRÁCE Bc. Lukáš Kobza
AUTHOR

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

# VIZUALIZACE A UŽIVATELSKÉ ROZHRANÍ PRO ŘÍDICÍ SYSTÉM DIVADELNÍHO JEVIŠTĚ

VISUALIZATION AND USER INTERFACE FOR THEATRE STAGE CONTROL SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. Lukáš Kobza

VEDOUCÍ PRÁCE
SUPERVISOR

prof. Dr. Ing. Pavel Zemčík

BRNO 2013

# Abstrakt

Tato práce se zabývá problematikou modelování a 3D vizualizace. Rovněž obsahuje přehled technického zařízení divadelního jeviště a jeho řízení, s důrazem na interakci s obsluhou a uživatelské rozhraní. Hlavním předmětem pak je vyšetření možností využití technologie 3D vizualizace v řízení jevištní techniky a následně návrh a implementace aplikace pro 3D vizualizaci jeviště za účelem zvýšení přehlednosti a bezpečnosti obsluhy divadelního řídicího systému.



# Abstract

This thesis deals with questions of modelling and 3D visualization. Also, it involves an overview of technical equipment on a theatre stage and control systems of this machinery with accent on user interface and all the interaction with staff. Afterwards, the main topic is the investigation of 3D visualization utilization technology in the field of theatre stage control systems and then the proposal and implementation of the theatre stage 3D visualization application follows in order to increase a clearness and safety of operation with the theatre control system.



# Klíčová slova

Model, modelování, 3D vizualizace, vizualizace, detekce kolizí, obalové těleso, octree, divadlo, jeviště, řídicí systém, řízení, uživatelské rozhraní

# Keywords

Model, modelling, 3D visualization, visualization, collision detection, bounding volume, octree, theatre, stage, control system, control, user interface


# Citace

Kobza Lukáš: Vizualizace a uživatelské rozhraní pro řídicí systém divadelního jeviště, diplomová práce, Brno, FIT VUT v Brně, 2013

# Vizualizace a uživatelské rozhraní pro řídicí systém divadelního jeviště

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením prof. Dr. Ing. Pavla Zemčíka
Další informace mi poskytli Bc. Jaromír Boček, Mgr. Zbyněk Mikša a Ing. Michal Drlík
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Lukáš Kobza
22. května 2013

## Poděkování

Rád bych zde poděkoval panu prof. Dr. Ing. Pavlu Zemčíkovi za jeho vedení práce a poskytnuté konzultace. Za další cenné informace patří poděkování panu Bc. Jaromíru Bočkovi, Mgr. Zbyňku Mikšovi a Ing. Michalu Drlíkovi

# Obsah

# 1 Úvod

Člověk se již odpradávna snaží si nějak ulehčit svojí práci. S postupujícím věkem přicházely stále dokonalejší vynálezy a efektivnější zdroje energie. Mnoho vynálezů v průběhu historie lidstva přineslo jistou revoluci. V moderní historii pak jednu z nejzásadnějších revolucí způsobil vynález počítače, s jehož přičiněním se mohl svět vydat na dlouhou cestu vývoje počítačového řízení a automatizace. V současnosti se již za velikým kusem této cesty můžeme ohlédnout a prohlásit, že automatizace již zdaleka není doménou pouze průmyslu, nýbrž nás provází a podílí se na zajišťování našeho pohodlí takřka neustále.

Žijeme v době, kdy je počítač schopen člověka nahradit při vykonávání zdlouhavých, rutinních ale často i poměrně komplikovaných činností. Počítačovým řídicím systémům bývá mnohdy svěřována obsluha takových systémů, kde i malá chyba či odchylka může způsobit havárii s velmi vážnými následky. Za příklad lze postavit automatizované přístroje ve zdravotnictví, systémy řídící dopravu, řídicí počítač na palubě letadla a mnohé další. Na návrh řídicích algoritmů běžících na těchto počítačích a odpovídajících za bezpečně vykonanou práci jsou jistě kladeny velmi vysoké nároky. Nakonec však lze prohlásit, že počítač pod vládou člověka je s to zajišťovat lecjakou činnost.

Nakonec i pozornému divákovi, sedícímu v hledišti divadla, zpravidla dojde, že veškerý ten pohyb na jevišti, dotvářející scénář odvíjejícího se divadelního děje musí jistě být podpořen složitou aparaturou jevištního řídicího systému. Moderní technika a automatizace již pochopitelně pronikla i do prostorů divadelních scén. Od počátku k divadlu patří kulisy a rekvizity dokreslující herecké výkony umělců a s těmito je nutně potřeba manipulovat, pomalu je spouštět z výšky či znenadání vysunout z podlahy jeviště, otáčet s nimi apod. Nezřídka je navíc zapotřebí takto manipulovat i se samotnými herci. I v divadle bylo dříve užíváno mechanických pomůcek, kladek a rumpálů. Avšak postupem času byla klika nahrazena motorem, později řízeným počítačem a došlo tak k postupnému zavádění jevištní automatizace a řídicích systémů.

Návrh řídicího systému divadelního jeviště zdaleka není triviální záležitostí, neboť současná velká divadla disponují velkým množstvím pohonů (obvykle v řádu nemálo desítek kusů) a protože se zde mnohdy manipuluje s těžkými předměty ba dokonce s lidmi, patří jevištní technika k oněm potenciálně nebezpečným oblastem automatizace. Je zřejmé že vývoj směřuje k novým technologiím, které by napomohly vyšší bezpečnosti nebo přinejmenším zlepšením přehledu nad děním ve scéně.

Právě s tímto je spojeno téma této práce. Popisuje význam uživatelského rozhraní divadelního ŘS a šetří perspektivy 3D modelování, tedy prostorového zobrazení scény divadla za účelem zvýšení přehlednosti a bezpečnosti při obsluze ŘS. Problematikou jevištní technologie a jejího řízení jsem se zabýval již ve své bakalářské práci a tímto dílem na svojí minulou činnost volně navazuji, tentokrát ve spolupráci s firmou DriveControl s.r.o., zabývající se vývojem a distribucí divadelních ŘS.

Po úvodu následuje kapitola věnovaná modelování. Dávám si zde za úkol uvést čtenáře především do problematiky 3D modelování a seznámit jej s dostupnými nástroji. V první podkapitole definuji poněkud obecněji pojem model a následně popisuji modelování a počítačovou simulaci. Postupně se již dostávám k prostorovému modelování a stručnému popisu geometrických transformací, 3D zobrazování a také některých nástrojů pro 3D modelování. Poslední část kapitoly pak popisuje metody a postupy používané při detekci kolizí v prostoru. Kapitola 3 nabízí přehled divadelní techniky a diskutuje aspekty divadelního řídicího systému včetně problematiky bezpečnosti. Druhá půle této kapitoly je pak zaměřena na interakci divadelního ŘS s jeho obsluhou, na jeho uživatelské rozhraní a zabývá se inovativními přístupy – zejména přínosem 3D modelování a rovněž uvádí příklady z praxe. Kapitola 4 analyzuje současný stav problematiky, formuluje motivaci k vývoji aplikace pro 3D vizualizaci divadelního jeviště a specifikuje návrh její budoucí podoby. Kapitola 5 pak popisuje samotnou realizaci 3D vizualizační aplikace.

# 2 Modelování a počítačová 3D grafika

Následující text poskytuje shrnutí současného stavu v problematiky relevantní k této práci. Shrnutí sestává ze dvou kapitol, které pokrývají klíčová témata. První z důležitých oblastí je modelování, které je spolu s dalšími okolnostmi předmětem této kapitoly. Modelování je velice obsáhlá disciplína, která se ve svém obecném pojetí dotýká široké řady oborů a výklad samého pojmu model je nutné hledat i ve zdrojích filosofie. Jeho uvedení je obsaženo v první podkapitole. Počínaje podkapitolou druhou se ovšem text již oprošťuje od širších souvislostí a zaměřuje se na popis modelování v kontextu počítačové vědy a protože v praxi patrně nejrozšířenější je modelování využívané pro simulaci a rovněž je toto relevantní k zaměření práce, je zde zařazeno uvedení do problematiky modelování a simulace. A konečně se kapitola věnuje prostorovému modelování a počítačové grafice, její podstatě, principům, používaným technikám a dostupným nástrojům. Poslední část kapitoly je věnována detekci kolizí v prostoru, což je problematika taktéž úzce spjata se 3D grafikou.

## 2.1 Model

*Model* může v současné době být chápán různým způsobem, jeho interpretace se může lišit v závislosti na autorovi, záměru, vědního či technického oboru, apod. Historie pojmu byla údajně započata v dávné minulosti, kdy ve stavitelství reprezentoval vize budoucího díla. Později, od dob renesance a nejvíce pak v průběhu 20. století a nedávné minulosti, byl tento pojem rozšířen do teorie poznání a tímto do základů vědních disciplín.

Model v dnešním pohledu lze ve své podstatě chápat jako jazykový útvar, který slouží coby komunikace schopný záznam rozpracovanosti dané skutečnosti. Vyjadřuje to, co už vytvořeno bylo a to, čehož vytvoření je v očekávání. Toto platí na polích výzkumných i tvůrčích, jednak pro vědce, kteří poznávají reálný svět a formulují nějaký svůj model a poznatky či předpoklady o něm a v přenesení to platí i pro myslitele nebo tvůrce prezentující své tvůrčí představy, pocity, prožitky a postoje. Jaromír Křemen v [1] píše: *„Model tak slouží nejen tvůrci či vědci, jako reflexe jeho konání a předmět ověřování správnosti jeho invence, hypotézy, apod., ale i jako produkt k diskusi, obdivu, poučení, tedy k určité komunikaci přinejmenším jednosměrné“*. Model, ať už se na onen pojem díváme jakkoliv, je něčím, co reprezentuje část reality anebo fantazie a především reprezentuje vlastnosti této reality (fantazie) zpravidla ty, které jsou v kontextu daného účelu klíčové. Je zřejmé, že vhodný formální popis těchto vlastností je rovněž klíčový. Jak je již zmíněno výše, model chápeme jako jazykový útvar schopný komunikace. Zřejmě tedy existuje jednak *jazyk modelu* a potom jiné jazyky, kterými se komunikuje o modelu („nad modelem“). Jazyk modelu je nazýván jako *objektový jazyk* [1]. Ten se během cesty poznání, tedy nabývání znalostí o modelu a tvorby vyskytuje prvotně v lidském vědomí v jakési vágní předpodobě, jež psychologové označují jako *pocitový* nebo *obrazový* (*představový*) jazyk a poté v podobě komunikovatelné, z hlediska lidského vědomí vnější, která již tvoří prezentovatelný model.

Popsáno formálněji se modelem zpravidla rozumí nějaká entita *M*, která pro konkrétní účely představuje jinou entitu *O*, kterou chápeme jako zdrojovou a lze ji rovněž označit coby předlohu, případně originál [1]. Obvykle entita *O* představuje konkrétní část reálného světa a *M* je pak souborem znalostí o ní získaných. *M* lze nazvat *znalostním* či *kognitivním modelem* oné části reality. Znalostní model dané skutečnosti rozumíme jako soubor relevantních znalostí o ní, které jsou zaznamenané ve vhodném objektovém jazyce. Tento jazyk na sebe může brát nepřeberné množství formulací. Pokud jsou znalosti získané o modelu formulované např. v jazyku geometrie, hovoříme

často o geometrickém modelu. Jiné modely jsou popsány matematickým formálním zápisem a pak hovoříme o modelech matematických, je-li užito formulí logiky, tak o logických, apod. [1].

Jedna z takových formulací může být i pomocí prostorové geometrie, kde je model reprezentován množinou prostorových entit či primitiv. Problematice takovéto formulace se věnuji v kapitole 2.3.

## 2.2 Modelování a simulace

Tato podkapitola zkoumá modelování v současném pohledu, coby výzkumnou metodu, která nalézá uplatnění v široké paletě oborů, jako je fyzika, genetika, geologie, biologie, ekonomie a mnohé další.

Jak je popsáno v [2], důvody vytváření modelů mohou být rozličné. Jsou to jednak ekonomické, kdy cena experimentů prováděných na modelu a cena modelu samotného může být nepoměrně nižší než cena experimentů prováděných na originálu. Jednak důvody časové, jelikož modelování umožňuje významné urychlení nebo naopak zpomalení experimentů. Např. počítačový model rostlinné výroby umožňuje průběh celé simulace urychlit a podat výsledky během několika dnů či hodin oproti experimentům se skutečnými rostlinami, které by trvaly v řádu měsíců. Naproti tomu možnost zpomalení bývá vítána v případě simulací v oblasti chemie, kde umožňuje přehledné zaznamenání jednotlivých fází průběhu experimentu. Další důvody mohou být ryze principiální. Může například nastat situace, kdy je originální systém absolutně nedostupný pro přímé zkoumání. Jsou to případy výzkumu systémů dostupných v minulosti, ale nikoli dnes (př. výzkum fyziologie pravěkých organismů) nebo systémů které dosud nevznikly (experimenty spojené s vývojem vesmírné sondy) nebo různých teoretických hypotéz a jevů. Důvodem může nakonec být pouze potřeba vizualizovat nějaký reálný děj za účelem zvýšení přehlednosti – reálný děj není dostatečně dobře pzorovatelný, a tak počítačová vizualizace může poskytnout jeho přehlednější zobrazení a možnost snadnějšího pozorování.

*„Jak již sám název napovídá, je základním obsahem výzkumné metody modelování jednak vytváření modelů různého charakteru, ověřování správnosti vytvořených modelů a provádění experimentů s modely, jejichž výsledky lze s větší či menší dávkou přesnosti a přiblížení vztahovat na předmět našeho výzkumu. Součástí metody modelování je samozřejmě i aplikace získaných výsledků vztažených na zkoumaný objekt.“* [2] Jinými slovy lze říci, že modelování bývá dnes úzce spjato se *simulací*.

V souladu definicemi uvedenými v [3], je třeba zavést pojem *systém*. Systém představuje soubor elementárních částí, prvků systému, které mezi sebou mají určité vazby, propojení prvků. Systémem tedy lze nazvat i onu část reálného světa, která je předmětem našeho výzkumu. *Modelování* pak je cílevědomou činností, během níž uplatňujeme znalosti o modelovaném systému a na jejich základě vytváříme jiný systém, tedy model zkoumaného systému. Konečně *simulace* představuje získávání nových znalostí o zkoumaném systému prostřednictvím experimentů prováděných s jeho modelem.

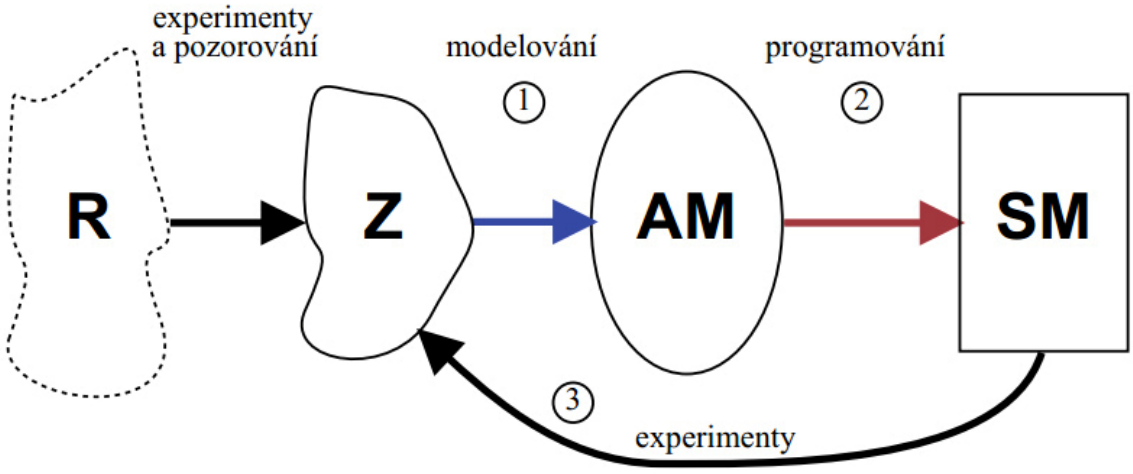
Během modelování v podstatě nahrazujeme zkoumaný systém jiným, novým systémem takovým, který bude lépe vyhovovat našim cílům, tedy bude vhodnější pro typ experimentů, které hodláme provádět a umožní nám tak potřebné znalosti získat snadněji a pohodlněji než zkoumaný systém samotný. Z toho důvodu není vztah mezi modelem a jeho předlohou dán jenom jejich obecnými vlastnostmi, ale i účelem, k němuž je model předurčen.

Máme-li nějaký systém, který hodláme zkoumat a předpokládáme, že chováme určitou množinu znalostí o něm, lze přistoupit k procesu modelování (a simulace), který sestává ze tří základních kroků. Předně je to vytvoření abstraktního modelu, zadruhé pak vytvoření modelu simulačního. Třetí etapou je samotná simulace, tedy experimentování se simulačním modelem.

Budováním *abstraktního modelu*se rozumí takový popis systému, který abstrahuje od faktů a vlastností nerelevantních k účelu modelu a cíli našeho výzkumu [4]. Těžištěm tvorby je určení vhodných složek takového modelu. Identifikovat, které vlastnosti mají stěžejní vliv na efektivnost systému a případně, zda daná vlastnost bude v rámci modelu součástí systému nebo jeho okolí. Abstraktní model může mít podobu matematických rovnic, schémat, skic, apod.

Na základě abstraktního modelu je vybudován *simulační model*, čímž se, zjednodušeně řečeno, obvykle rozumí abstraktní model zapsaný v některém z vhodných programovacích jazyků [4]. Zatímco mezi originálním systémem a abstraktním modelem platí homomorfní vztah, mezi abstraktním modelem a simulačním modelem je požadován vztah izomorfní. To znamená, že struktura těchto modelů a chování jejich prvků by mělo být ekvivalentní. Pro implementaci simulačního modelu lze využít některého ze speciálních simulačních programovacích jazyků (např. Simula, Simscript, Modsim, …), které svými prostředky usnadňují práci, nicméně lze v zásadě použít i „běžného“ programovacího jazyka (např. C++, C#, Java, …) nebo simulační knihovny (např. knihovna Simlib pro C++). Podle způsobu, jakým je abstraktní model reprezentován modelem simulačním lze rozlišit dvě základní skupiny simulačních modelů (systémů), k nim příslušejících programovacích jazyků a potažmo samotných simulací. Jedná se o diskrétní systémy a spojité systémy, případně kombinované v závislosti na tom, jak přistupují k reprezentaci času.

Mezi spojité systémy patří systémy takové, při jejichž modelování je vhodné změny vstupních, výstupních a stavových proměnných reprezentovat spojitými funkcemi; například v teorii řízení modelování elektrických obvodů, či netechnických oborech jako biologie či ekologie. Popis dynamiky spojitých systémů typicky bývá popsán diferenciálními rovnicemi. Naproti tomu diskrétní systémy jsou systémy takové, kde ke změnám stavu dochází skokem, na základě nějaké události za předpokladu, že událost trvá nulovou dobu a mezi jednotlivými událostmi se stav systému nemění. Diskrétní systémy mohou být buďto stochastické (typickým příkladem jsou systémy hromadné obsluhy) nebo deterministické.




**Obrázek 2.1 Ilustrace postupu při modelování a iterativního procesu simulace [3]**

Po „mezikroku“ *verifikace simulačního modelu*, tedy ověření jeho korespondence s modelem abstraktním, nastává třetí krok celého procesu modelování a tím je simulace [4]. V této fázi již existuje vytvořený simulační model a lze přistoupit k experimentování s ním na základě vstupních veličin a parametrů. Tento proces zpravidla probíhá ve více simulačních bězích. Těch je nutné provést dostatečný počet k tomu, aby získaná informace o zkoumaném systému byla dostatečná, případně abychom nalezli optimální vstupní parametry, za kterých systém vykazuje požadované chování.

Se simulací je rovněž spojen proces *ověřování validity modelu*, kdy průběžně konfrontujeme nově získané znalosti se znalostmi, které jsou o systému známé. Tímto dokazujeme, že skutečně pracujeme s modelem adekvátním předloze. Avšak absolutní určení přesnosti modelu je problematické, proto pracujeme pouze s relativní *hladinou spolehlivosti*, která je pak směrodatná pro určení správnosti získaných výsledků. Pakliže chování modelu není v souladu s předpokladem založeném na znalostech o originálu, je třeba model modifikovat, a to s přihlédnutím k nově získaným znalostem. Po modifikaci přistoupíme opět k simulaci a celé tyto simulační běhy opakujeme tak dlouho, dokud výsledky konání nejsou upokojivé.

## 2.3 Prostorové modelování, 3D zobrazení

Předmětem této práce je ale zejména modelování ve významu prostorovém. Závěrem kapitoly 2.1 stojí, že každý model je popsán vhodným objektovým jazykem a jedním z takových je jazyk prostorové geometrie. *Prostorové*, tedy *3D modelování* je pak vlastně modelováním, které je postavené na jazyku prostorové geometrie, pracujícím s prostorovými entitami. Je to jeden z možných způsobů, jak popsat model nějaké skutečnosti, nějaké předlohy a protože je to způsob velmi názorný, který přehledně a jednoznačně popisuje svojí předlohu, stala se přirozeně tato metoda notně populární a v současnosti hojně užívaná. Uplatnění nalézá nejen jako statický model nějakého skutečného či smyšleného objektu, ale je možné například vytvářet animace a rovněž mohou prostorové modely posloužit coby názorná ilustrace nějakého experimentu při chemických, biologických či jiných simulacích.

### *Prostorové modelování*

Prostorové entity a vůbec celé prostorové modelování může být děleno podle přístupu k jejich reprezentaci v prostoru, kde základní dělení je na objemové modely a povrchové modely [5]:

- *Povrchové modely* – sestávají zpravidla z ploch nebo prostorových křivek. Tyto útvary vykazují nulový objem a jsou definovány matematickým vzorcem. Už v roce 1959 P. de Casteljau u firmy Citroen používal matematický model křivek a ploch a P. Béziere u firmy Renault vedl vývoj programového systému UNIDURF pro návrh křivek a ploch. Existuje několik typů křivek a z nich odvozených ploch. Základní dělení je na interpolační (Hermitovské kubiky) a aproximační (Beziérovy křivky a kubiky, B-spline křivky nebo neuniformní racionální B-spline – NURBS křivky).
- *Objemové modely* – pracují s objemem, též je lze nazývat *tělesy*; dá se říci, že jsou obdobou skutečných hmotných předmětů, které zaujímají určitý objem v prostoru. Na objemový model se lze dívat jako na množinu bodů, splňující určitá kriteria. Lze říci, že se jedná o sjednocení dvou disjunktních množin – množiny *vnitřních bodů* a množiny *hraničních bodů*, kde vnitřní bod sousedí pouze s body vnitřními a body hraničními, naproti tomu hraniční bod sousedí s alespoň jedním bodem vnitřním, hraničním a bodem vnějším nepatřícím do žádné z uvedených dvou množin. Toto z objemových modelů vylučuje křivky a plochy, avšak tyto neobjemové entity bývají využívány pro popis množiny hraničních bodů. K reprezentaci a modelování objemových modelů bývají využívány různé metody, založené například na hraniční reprezentaci těles, konstruktivní geometrii (CSG), nebo využívající deformačních a transformačních operací.

Jakýmsi třetím, dodatečným přístupem je *procedurální modelování*, což je soubor technik postavených na specifických algoritmech, například *L-systémy* nebo *fraktální geometrie*

pro modelování krajin, kamenů, stromů nebo *systémy částic* pro modelování kouře, explozí či dalších nejen přírodních jevů. Všechny zmíněné přístupy zahrnuji široké množství metod a matematických modelů sloužící k jejich tvorbě a práci s nimi, avšak jejich podrobný popis není k této práci zcela relevantní a proto ho neuvádím. Tyto metody jsou podrobně popsány v různých výukových materiálech a související literatuře, například v [5].

## *Geometrické transformace*

Geometrické transformace patří k nejčastěji aplikovaným transformacím nejen v rámci prostorového modelování ale v počítačové grafice vůbec [6]. Základními transformacemi jsou *translace*, *rotace* a *změna měřítka*. Aplikace geometrické transformace spočívá ve vynásobení transformovaného bodu *transformační maticí* [7]. Translací se rozumí posun daného bodu o požadovaný vektor. Pro homogenní systém souřadnic pak platí, že máme-li bod

$$\mathrm{P} = [\mathrm{x}, \mathrm{y}, 1] \tag{1}$$

a máme-li dosáhnout translace bodu *P* o vektor

$$d = (d_x, d_y), \tag{2}$$

pak aplikací transformační matice

$$D = \begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{pmatrix} \tag{3}$$

na bod *P* dostaneme

$$\mathrm{P}' = \mathrm{DP} = \begin{pmatrix} 1 & 0 & \mathrm{d_x} \\ 0 & 1 & \mathrm{d_y} \\ 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} \mathrm{x} \\ \mathrm{y} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathrm{x} + \mathrm{d_x} \\ \mathrm{y} + \mathrm{d_y} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathrm{x}' \\ \mathrm{y}' \\ 1 \end{bmatrix}. \tag{4}$$

Aplikace transformačních matic na bod v prostoru pak funguje analogicky, tedy pro homogenní souřadný systém mají transformační matice pro základní operace následující podobu [7]. Pro translaci o vektor $d = (d_X, d_\mathrm{Y}, d_\mathrm{Z})$ slouží matice *T* (5), pro změnu měřítka pak matice *S* (6), kde hodnoty $s_X$, $s_Y$ a $s_Z$ představují koeficienty pro změnu měřítka ve směru dané osy a pro rotaci kolem osy *x* systému souřadnic o úhel $\theta$ je to pak matice $R_X$ (7). Podobně pro rotaci kolem osy *y* slouží matice $R_Y$ (8) a kolem osy *z* matice $R_Z$ (9).

$$\mathrm{T} = \begin{pmatrix} 1 & 0 & 0 & \mathrm{d_x} \\ 0 & 1 & 0 & \mathrm{d_y} \\ 0 & 0 & 1 & \mathrm{d_z} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5}$$

$$S = \begin{pmatrix} s_X & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_Z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6}$$

$$R_X = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\vartheta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{7}$$

$$R_Y = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{8}$$

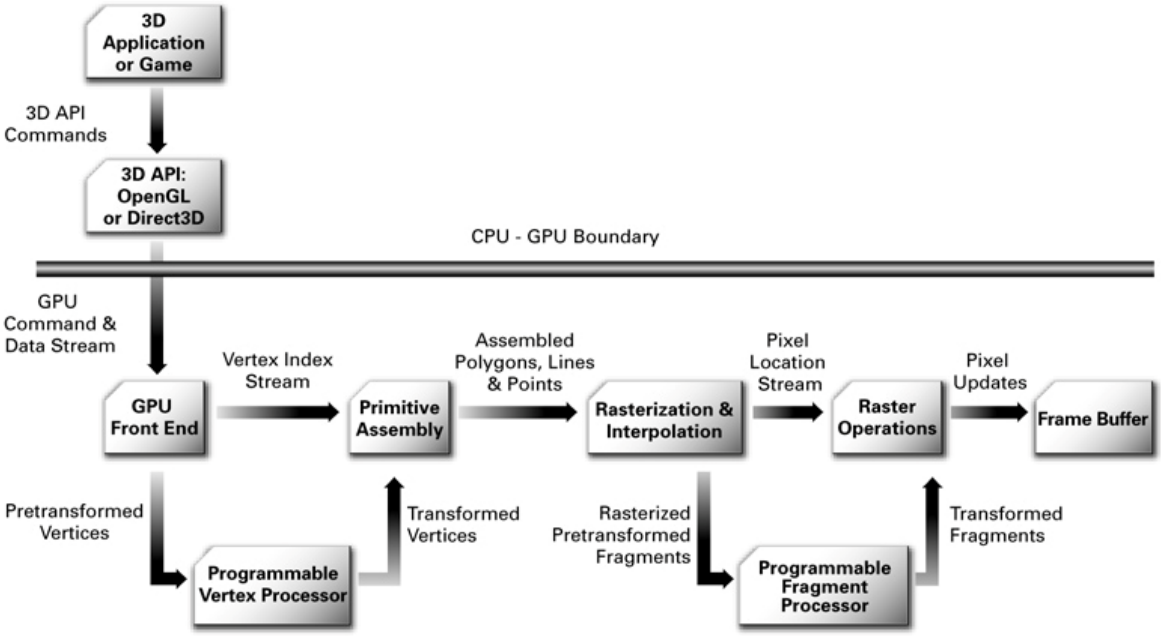
$$R_Z = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{9}$$

Aplikování více transformací na stejný bod je možné, avšak je důležité zachovat správné pořadí aplikovaných transformací [6]. Rovněž lze sloučit více transformačních matic do jedné a provést pak pouze jedinou transformaci aplikací výsledné matice. Tohoto sloučení dosáhneme postupným vynásobením všech dílčích transformačních matic, avšak opět zde záleží na pořadí při násobení.

## *Zobrazení 3D scén*

Abychom s prostorovým modelem mohli nějakým způsobem manipulovat nebo ho jen prohlížet, je zapotřebí jeho zobrazení. Jakkoliv to zní logicky, problematika *3D zobrazení* (*rendering*) [5] není zdaleka triviální. Základní myšlenkou je vlastně převod trojrozměrné informace na dvojrozměrnou tak, aby bylo možné ji zobrazit pomocí standardního zobrazovacího zařízení, které v dnešní době bývá zpravidla dvojrozměrné a obvykle jím rozumíme displej či monitor.

Pro komplikovanější modely může být tento proces výpočetně velmi náročný, zejména při použití realistických osvětlovacích modelů a metod stínování. Zobrazení 3D grafiky proto dnes běžně bývá hardwarově akcelerováno a celý proces zajišťuje tzv. vykreslovací řetězec. Obrázek 2.2 naznačuje, že vykreslování sestává jednak z výpočtů spojených s operacemi s vrcholy, které zajišťuje programovatelný vertex procesor a jednak z výpočtů procesu rasterizace, tedy převodu trojrozměrné reprezentace na dvojrozměrnou, následované rastrovými operacemi, zpracování textur, apod. Rastrové operace má na starosti programovatelný fragment procesor. Výsledný 2D snímek je nakonec uložen do frame bufferu a předán na výstup zobrazovacímu zařízení.




**Obrázek 2.2 Vykreslovací řetězec [14]**

Pokud uvažujeme jakých postupů užít při zobrazování 3D modelu, je třeba si předně ujasnit jeho účel. Dokonalé světelné modely a stínování mají místo tam, kde je realistických výsledků třeba, ať už při statických záběrech modelované scény nebo při animaci či dynamickém zobrazování počítačových modelů ve filmu nebo v počítačových hrách. Ovšem například u zobrazení 3D modelů a animovaných simulací v průmyslu či vědě je rozhodující názornost, přehlednost a před perfektním vizuálním vjemem má přednost spíše přesnost a názornost modelu.

## 2.4 Nástroje pro 3D modelování

Tvorba a manipulace s 3D grafikou a modely může mít více podob. V první řadě jsou to nízkoúrovňové knihovny jako je *OpenGL* nebo *direct3D* od Microsoft implementované jako API pro některý programovací jazyk. Programátor je pak schopný pomocí dostupných funkcí a tříd ovládat vykreslovací stavový stroj a pracovat s prostorovými objekty. Tyto nástroje (konkrétně např. OpenGL), jsou však postaveny přímo nad vykreslovacím řetězcem a díky své nízké úrovni abstrakce je jejich využívání poměrně pracné a pro rozsáhlejší projekty nepříliš efektivní. Proto v takových případech je vhodnější sáhnout po API vyšší úrovně, jako je *OpenSceneGraph* nebo *Open Inventor*. Nástroje této kategorie jsou na postaveny nízkoúrovňových API, avšak abstrahují od elementárních operací a přinášejí tak mnohem vyšší efektivitu při vývoji. Prostor a v něm vytvářená scéna bývá popsána *grafem scény*, jehož uzly představují jednotlivé objekty ve scéně. Díky tomuto je zajištěna hierarchická reprezentace modelu umožňující velmi efektivní přístup k jeho dílčím objektům.

Nejvyšší úrovní pak je široká paleta 3D grafických editorů, *CAD systémů* (computer-aided design – počítačem podporované projektování) a modelovacích nástrojů, lišících se zpracováním a robustností v závislosti na jejich účelu. Pro animaci, film nebo nejrůznější designové studie může posloužit například *Blender* nebo *3D Studio Max*, pro strojírenskou konstrukci nebo projektování ve stavebnictví pak *Autodesk Inventor* či *SolidWorks*. Svou jednoduchostí a názorností se vyznačuje nástroj *SketchUp*, v současnosti vyvíjený firmou *Trimble Navigation*.

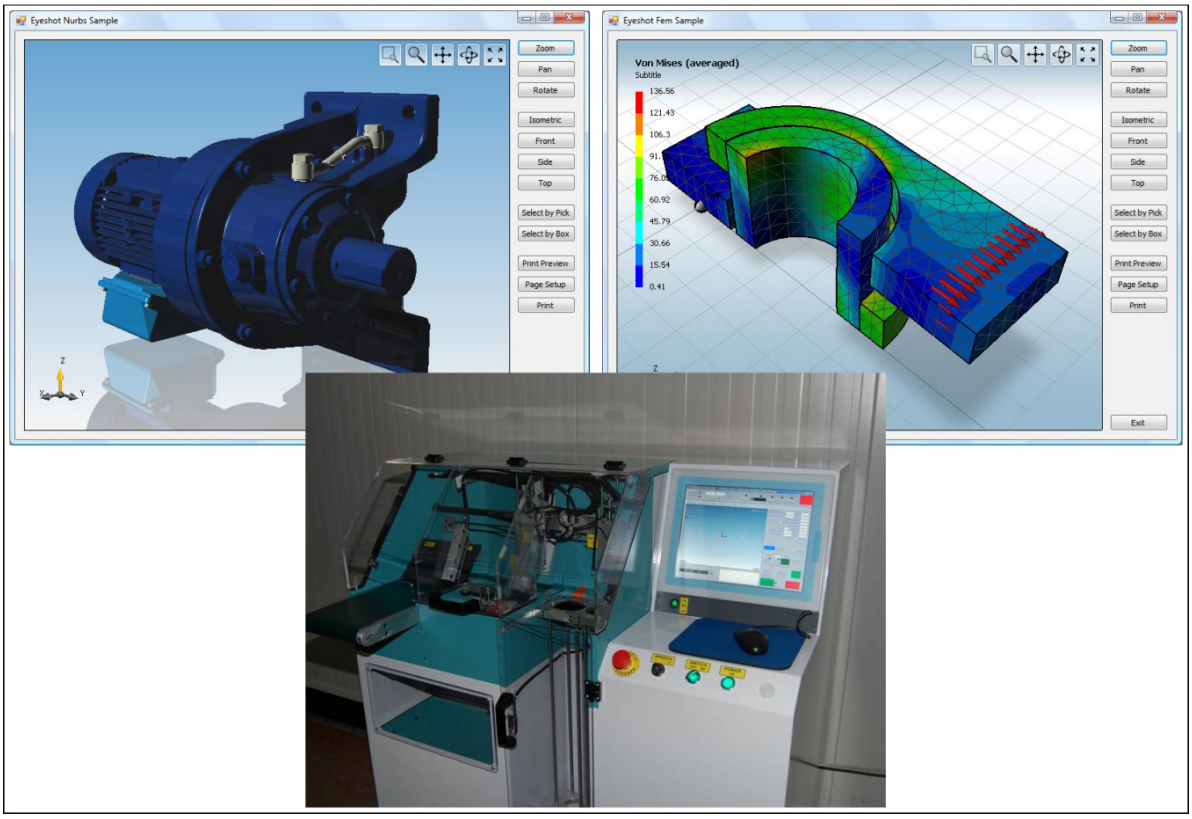
Pro mnohé účely může být zajímavá možnost kombinovat možnosti prostředí grafických editorů nebo v nich vytvořené modely s některým API vyšší i nižší úrovně – například enginy počítačových her představují výkonné stroje operující na tomto pomezí.

Dalším příkladem, který je tak trochu na pomezí, může být počítačová simulace. Simulační model je implementován programovacím jazykem (ať už s využitím speciálních nástrojů – knihoven nebo simulačních jazyků – nebo bez nich) a pro poskytnutí názornější ilustrace je simulační model doplněn o model prostorový, který simulaci dynamicky dokresluje. K tomuto lze buďto opět zvolit vhodné API pro vykreslování 3D grafiky a nebo využít některého z 3D grafických editorů, které poskytují své API pro vývoj pluginů či jinak umožňující propojení s externím softwarem a tímto programovatelně manipulovat svým prostředím a objekty ve scéně. Na tomto místě stručně uvedu několik nástrojů, které takovými prostředky disponují.

### *Cadwork LexoCAD*

Lexocad je CAD systém z dílny firmy Cadwork Informatik [11]. Jedná se o jednoduchý nástroj pro široké spektrum použití, podobně jako SketchUp. Disponuje integrovaným interpretem skriptovacího jazyka Python a umožňuje tak programově pracovat s parametry modelů ve scéně. Buď lze využít vestavěné konzole jazyka Python a nebo zasílat příkazy na komunikační rozhraní programu založeném na TCP/IP komunikačním protokolu.

### *DevDept Eyeshot*




**Obrázek 2.3 Demonstrace užití nástroje Eyeshot, vlevo: 3D model motoru, vpravo 3D vizualizace metody konečných prvků, dole: řízení CNC umožňující 3D zobrazení [12]**

Eyeshot je nástroj od italské firmy DevDept Software [12]. Firma byla založena roku 2005 a věnuje se nástrojům pro 3D modelování a vizualizaci určené vývojářům užívajícím technologie Microsoft.

Eyeshot je vysokoúrovňové API pro Visual Basic a C#, umožňující zobrazování 3D grafiky a manipulaci s modelem.

Podporuje povrchové i prostorové modelování a standardní možnosti, jako ortogonální i perspektivní zobrazení, práci s vrstvami, realistické stínování, mapování textur, antialiasing a další. Například pro účely fyzikální simulace může být užitečná podpora vizualizace výpočtů metodou konečných prvků. Rovněž umožňuje import/export jiných CAD formátů (včetně DWG).

Nástroj poskytuje velmi vysokou úroveň abstrakce a pro vývojáře tak se tak nabízí jako efektivní nástroj pro poměrně široké spektrum oblastí užití.

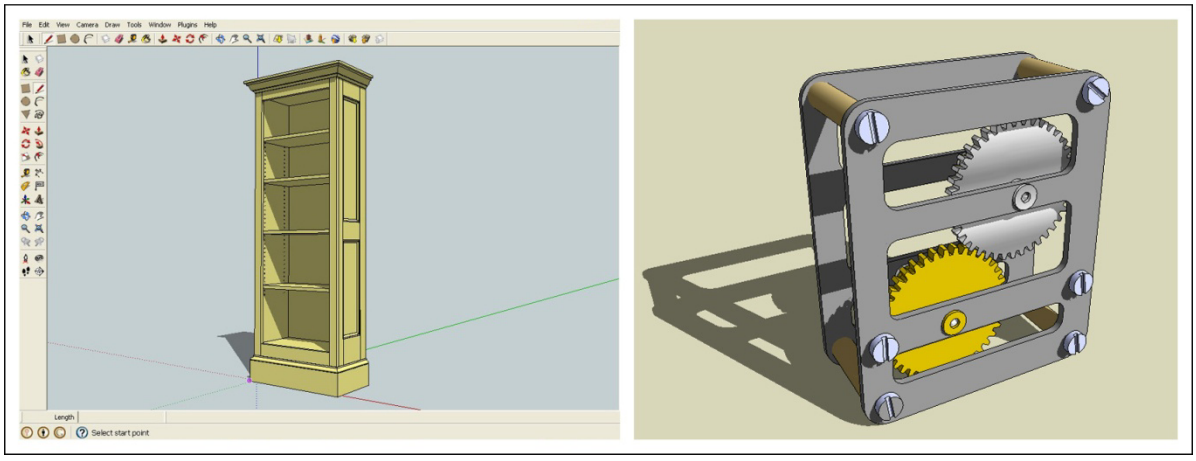
### *Trimble SketchUp*

SketchUp je CAD nástroj pro tvorbu 3D modelů, původně vyvinutý firmou *@Last Software* roku 1999, navržený pro potřeby architektů, stavebních i strojních inženýrů, ale i filmové tvůrce či vývojáře počítačových her [6]. Roku 2006 byla společnost koupena firmou *Google* a vývoj projektu byl dále směrován pro účely *Google Earth*, k modelování krajinných prvků, převážně staveb. Google nástroj postupně zdokonaloval ve verzích 6 až 8, doplnil jej o *3D Warehouse* (volně dostupnou databázi 3D modelů, do které může kdokoli přispět svým výtvorem, nebo si jiné stáhnout) nebo rozšířil *Ruby API*. V červnu 2012 byl projekt SketchUp koupen firmou Trimble Navigation, kterou je nadále vyvíjen.

SketchUp se vyznačuje svojí intuitivností, tedy jednak uživatel-začátečník se s nástrojem poměrně snadno dovede naučit pracovat a především je schopný rychle a efektivně tvořit. Předností tohoto nástroje není dokonalá „strojírenská“ přesnost a robustnost, ale svým účelem slouží spíše pro jednoduché modelování a návrhy, což plyne i z jeho názvu. Dobře se tak hodí pro jednoduché „skicování“ ve strojírenství nebo architektuře stejně jako pro modelování budov a krajinných prvků pro aplikaci Google Earth, ale také jako nástroj pro počítačovou 3D simulaci.

V prvním odstavci jsem zmínil Ruby API, což je poměrně silný nástroj aplikace SketchUp. Jedná se o API implementované pro skriptovací jazyk Ruby a umožňuje implementovat nové pluginy a tím zprostředkovat propojení 3D scény v prostředí SketchUpu s jiným externím programem [9]. API je zdokumentováno na domovských stránkách aplikace, kde je dostupných i několik tutoriálů.

K programu SketchUp je rovněž dostupný jednoduchý SDK pro C++ zprostředkovávající vstupní a výstupní operace s modelovými soubory ve formátu SketchUp.



**Obrázek 2.4 vlevo prostředí nástroje SketchUp [10], vpravo model vytvořený nástrojem SketchUp [8]**

## 2.5 Geometrické postupy při detekci kolizí v prostoru

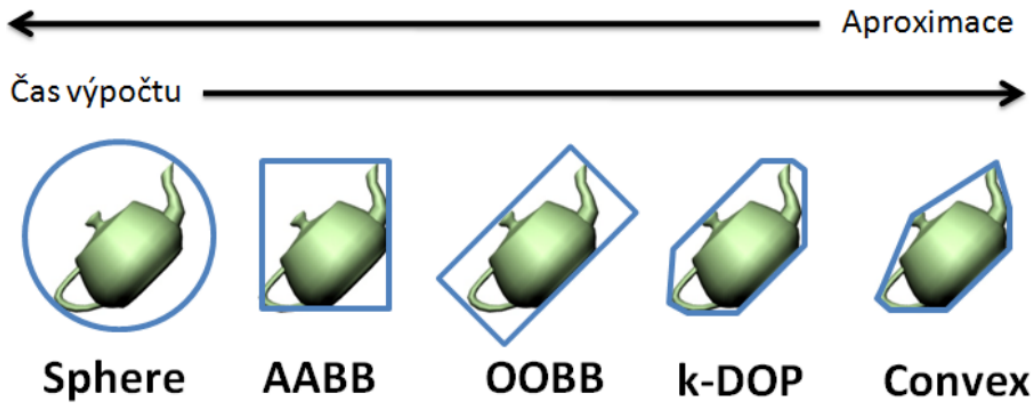
Ve velmi široké škále výpočetních disciplín, kde se potýkáme s trojrozměrným prostorem a manipulací s jemu příslušnými objekty, bývá zapotřebí řešit problém detekce kolizí mezi těmito objekty. Tento problém je zásadní například při plánování pohybu robota, vyhodnocování crash testů automobilů či v nejrůznějších oblastech automatizace. V neposlední řadě se však s řešením problému detekce kolizí potýkáme v počítačové 3D grafice – v CAD/CAM systémech, ve virtuální realitě, počítačových hrách a různých prostorových vizualizacích a simulacích.

V této kapitole jsou popsány metody, jaké lze aplikovat při detekci kolizí ve 3D grafice. Hranice modelu bývají reprezentovány sítí polygonů, nejčastěji trojúhelníků. Máme-li tedy dva objekty vymezené sítí trojúhelníků, jeden ze základních přístupů bude testovat trojúhelníky na kolizi jeden po druhém. Při každém testu jsou hledány průsečíky hran jednoho z trojúhelníků a roviny, ve které leží druhý trojúhelník, za pomoci standardních postupů analytické geometrie. Jedná se o metodu jistě velmi přesnou, neboť jsou brány v úvahu nejmenší detaily testovaných objektů, avšak zejména při dynamických simulacích a vizualizacích tento přístup sám o sobě není dost dobře použitelný pro jeho značnou výpočetní i paměťovou náročnost, zvláště vezmeme-li v úvahu, že složitější modely bývají reprezentovány tisíci až miliony trojúhelníky [13].

Situace tedy volá po metodě, která by celý výpočet vhodně urychlila a zefektivnila. Zaprvé lze výpočet urychlit, pokud budeme abstrahovat od detailů objektu zjednodušíme jej na co možná nejjednodušší útvar. Tento princip v praxi využívá metoda tzv. *prostorových obálek* (angl. *bounding volumes*), do kterých je daný objekt „vsazen“ a test na kolizi se tak výrazně urychlí. A za druhé můžeme dosáhnout dalšího urychlení, pokud zvýšíme efektivitu prohledávání 3D prostoru při hledání kolizních objektů.

### Princip prostorových obálek

Prostorová obálka je 3D objekt co nejjednoduššího tvaru, který v sobě zapouzdřuje nějaký jiný 3D objekt, typicky komplikovaného tvaru. Obálka tedy abstrahuje od detailů – aproximuje tvar daného objektu, přičemž tento v ní musí být obsažený celý. Během testu na kolize je pak počítáno pouze s obálkou, nikoliv se samotným objektem – pokud obálka není v kolizní situaci, pak jistě ani objekt uvnitř ní. Tímto se vyhneme náročnému výpočtu „trojúhelník po trojúhelníku“.




**Obrázek 2.5 Znázornění několika základních typů obálek a míry abstrakce [15]**

Zpravidla se ale nelze vyhnout tomu, že v obálce bude obsažen i nějaký prostor navíc – pokud tvar objektu přímo neodpovídá tvaru zvolené obálky. Nastává tedy nutnost správně zvolit optimální typ prostorové obálky – ten by měl vyplývat z hrubého tvaru objektu, měl by být jeho co nejvěrnější aproximací – ovšem v rámci zachování co nejvyšší míry abstrakce. Na Obrázku 2.5 je znázorněno několik typů obálek (zde pouze dvojrozměrně, schematicky). Je patrné, že například obálka typu *OOBB* aproximuje objekt čajové konvice značně věrněji, než obálka tvaru koule. Ještě přesnější bude *minimální konvexní obálka*, avšak již za cenu náročnějších výpočtů. Následuje stručný popis některých základních typů obálek [16]:

- *Koule (bounding sphere)* – je koule obsahující celý objekt, reprezentovaná středem a poloměrem. Výhodou je velmi nenáročný test na kolizi mezi dvěma koulemi. Stačí spočítat vzdálenost obou středů a pokud je větší, než součet obou poloměrů, ke kolizi nedochází. Navíc test zůstává stejně jednoduchý, i když se obálky pohybují podle jakkoli složité trajektorie. Nevýhodou je jejich ve většině případů až příliš vysoká míra abstrakce.
- *Válec (bounding cylinder)* – je válec obsahující celý objekt, reprezentovaný středem, poloměrem a výškou. Tuto obálku lze s výhodou použít v případě objektů, které mohou rotovat pouze podle jedné osy – často to bývá osa vertikální, v takovém případě hovoříme o vertikálně zarovnaném válci. Mezi dvěma vertikálně zarovnanými válci nastává kolize, pokud jsou v kolizi jejich vertikální a zároveň i horizontální průměty. I obálka typu válce poskytuje nenáročný test na kolizi, bývá využívána např. pro stojící lidské postavy.
- *Osově zarovnaný kvádr (AABB – axis-aligned bounding box)* – je kvádr obsahující celý objekt, orientovaný podle os pravoúhlého souřadného systému. Pro jeho reprezentaci postačí dva body – v jeho protilehlých rozích, což stále zachovává výhodu jednoduchého testu na kolizi. Problém ale může nastat, pokud zapouzdřený objekt bude rotovat, tehdy může docházet k výraznému snížení přesnosti aproximace.
- *Objektově zarovnaný kvádr (OOBB – object oriented bounding box)* – je kvádr obsahující celý objekt, avšak orientovaný v závislosti na něm. Toto oproti AABB přináší rezistenci proti transformaci rotace, kdy přesnost aproximace obálky zůstává zachována. Avšak cenou za to je zde již vyšší náročnost výpočtů při testování na kolize. Obálky typu kvádr, AABB i OOBB, poskytují ve většině případů přesnější aproximaci než např. koule, která pokrývá více specifickou oblast užití. Kupříkladu použijeme-li obálku typu AABB pro automobil, uvidíme evidentní výhodu oproti užití obálky typu koule, která by zahrnula i část silnice.
- *Diskrétně orientovaný polytop (DOP – discrete oriented polytope)* – je nadřazený pojem zahrnující polygon (v případě užití ve 2D) a *polyhedron* (v případě 3D), jež zahrnuje prostorová tělesa s libovolným počtem plochých stěn. DOP je konstruován tak, že určitý počet rovin zvolených v prostoru tak, že zapouzdřují daný objekt je posunováno (přibližováno k objektu), dokud nedojde ke kolizi s objektem. Obvykle bývá obálka značena jako k-DOP, kde *k* je počet rovin, ze kterých je DOP konstruován. V praxi se užívá množina rovin tvořící například kvádr se zkosenými horními hranami (10-DOP) nebo i spodními (14-DOP). 6-DOP vlastně představuje kvádr, bounding box, avšak libovolně orientovaný.

- *Konvexní obálka (convex hull)* – představuje minimální konvexní obálku objektu. Zapouzdřuje-li objekt s konečným počtem vrcholů, jedná se o polytop. Konvexní obálka představuje typ s nejpřesnější aproximací, avšak s nejnáročnějším testem kolize.

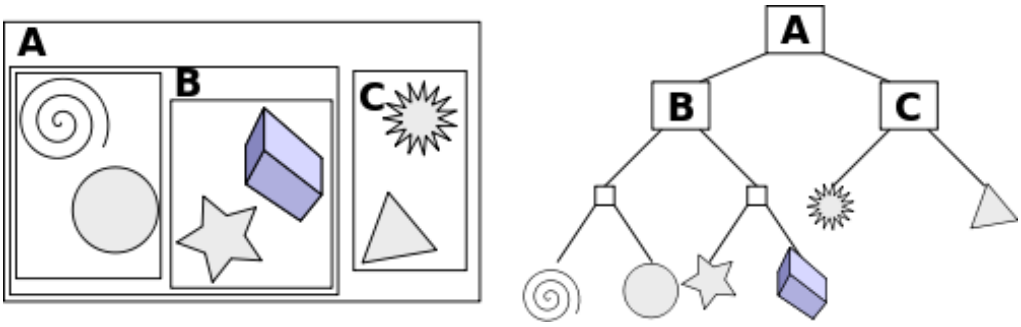
Ačkoliv tedy existuje celá řada typů obálek a lze definovat mnohé další, nejběžněji využívaným typem v praxi zůstává výpočetně nenáročný osově orientovaný kvádr, neboť přesnost aproximace, kterou poskytuje, je často dostatečná a pokud není, tak právě typ AABB je velmi vhodný pro skládání do složitějších hierarchií obálek. Jak už jsem zmínil výše, koule nechává příliš málo prostoru pro variabilitu pokud jde o těsnější přilnutí k zapouzdřenému objektu, a tak množina jeho využití zůstává poněkud užší [17]. Avšak nikoliv prázdná, stejně jako mohou v některých speciálních případech nalézt uplatnění i komplexnější typy, jako třeba k-DOP. Volba vhodného tvaru obálky je tedy v každém konkrétním případě předmětem pro důkladné studium daného problému.

# Metody hierarchického dělení prostoru

Již samotné použití principu obálek slibuje značné urychlení výpočtu, avšak stále nedostatečné pro větší množství objektů a zejména pro dynamické scény. Je třeba si uvědomit fakt, že málo kdy je pravděpodobné, že kolize může nastat s kterýmkoli objektem ve scéně. Tato myšlenka je zásadní, neboť prohledáváním pouze části prostoru se časová náročnost citelně sníží. S výhodou je pro tyto účely využíváno hierarchických struktur na principu stromů, a to v různých variantách. Přiblížím zde metodu *hierarchie obálek* (dále také *BVH – Bounding Volume Hierarchy*) a metodu rozkladu prostoru na *oktalový strom* (dále také z angl. *Octree*). Dále lze využít třeba R-stromy, k-d stromy apod.

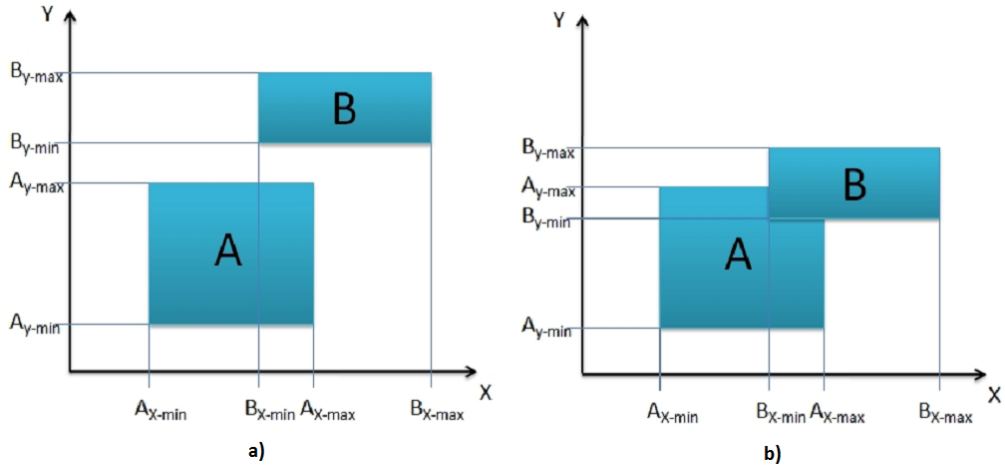
## *Hierarchie obálek*

BVH je stromová hierarchická struktura postavená nad obálkami objektů ve scéně. V nejtypičtějším a nejjednodušším pojetí se jedná o binární strom, kde jsou v listech obsaženy samotné objekty a nadřazenými uzly jsou seskupovány do dvojic a postupně do početnějších skupin a kořen stromu pak obsahuje všechny objekty ve scéně tak, jak je vidět na Obrázku 2.6.



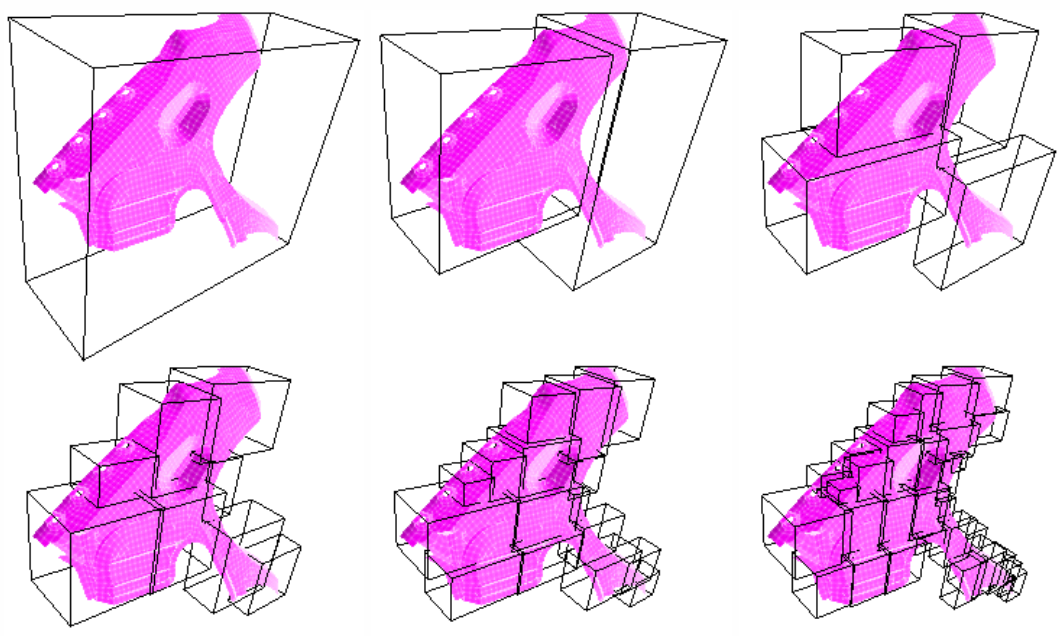

**Obrázek 2.6 Příklad jednoduché hierarchie obálek [18]**

Ačkoliv uzly BVH by mohly mít i více potomků, varianta binárního stromu je nejpoužívanější pro svojí jednoduchou reprezentaci, stavbu a implementaci. Případ od případu se ale může lišit počet objektů obsažený v jednom listu a hloubka stromu ale i metoda výstavby stromu – odspoda nahoru nebo shora dolu. Tato rozhodnutí závisí na předpokládaném celkovém počtu objektů ve scéně, jejich rozmístění a charakteru i na dalších požadavcích pro konkrétní užití dané BVH.




**Obrázek 2.7 a) Podmínka *A-max* ≥ *B-min* neplatí pro osu *y*, obálky *A* a *B* tedy nejsou v kolizi b) Podmínka *A-max* ≥ *B-min* platí pro osy *x* i *y*, obálky *A* a *B* tedy jsou v kolizi [15]**

Obecně však platí že kriteriem pro výstavbu stromu a zejména pak rozhodovacím kriteriem při procházení stromu jsou souřadnice udávající umístění obálek obsažených v jednotlivých uzlech. Z toho vyplývá, že pro správnou funkci by hierarchická struktura stromu měla logicky odpovídat skutečnému rozmístění obálek v prostoru. Jinými slovy, obálky, jež jsou potomky stejného uzlu, by měly být umístěny blízko sebe, ideálně v přímém sousedství.



**Obrázek 2.8 Příklad hierarchického rozkladu složitého objektu na obálky typu AABB v 6ti úrovních [19]**

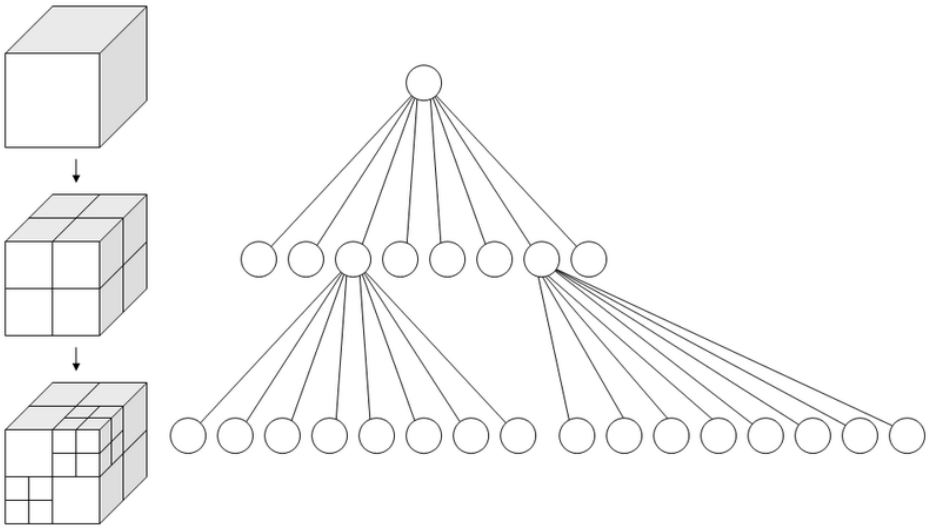
Vyhledávání, tedy průchod BVH pak funguje poměrně prostě, v závislosti na použitém typu obálky. Princip při užití typu AABB, pro jednoduchost převedený na dvojrozměrný systém, demonstruje Obrázek 2.7. Porovnáváme-li objekty *A* a *B*, tak stačí, aby podmínka *A-max* ≥ *B-min* nebyla platná pro jedinou z os souřadného systému a lze prohlásit, že obálky nejsou v kolizní situaci. Pokud je podmínka platná pro všechny osy, obálky jsou v kolizi [15].

V případě, že scéna obsahuje komplikovanější objekty, není nutné, aby celý objekt byl vždy obsažen v jediném listu [19]. V takovém případě může výstavba BVH pro jediný objekt vypadat např. jako na Obrázku 2.8, kde vidíme rozklad objektu na hierarchii obálek typu AABB v šesti úrovních. Všechny listy tohoto stromu pak obsahují odkazy na tentýž objekt. S výhodou tak lze odbourat větší část prázdného prostoru v obálce kolem objektu, který by vzniknul při užití prosté jednoúrovňové obálky pro inkriminovaný objekt (na Obrázku 2.8 vlevo nahoře) a docílit tak citelně vyšší míry přesnosti aproximace.

### *Octree*

Octree představuje hierarchickou dekompozici prostoru na osově orientované kvádry (tzv. oktanty) [20]. Celek je ve všech třech osách souřadného systému půlen a tím rozdělen na osm dílčích celků, které jsou stejným způsobem rekurzivně dále děleny tak dlouho, dokud není dosaženo požadované úrovně. Vzniklé dílčí celky jsou pak provázány stromovou strukturou, kde každý uzel má právě osm potomků, tedy oktalovým stromem. Tento princip zajišťuje dělení prostoru na celky, které se nepřekrývají, na odpovídající si úrovni mají shodnou velikost a jsou uspořádány ve vhodné struktuře, což z něho dělá velmi vhodnou metodu pro uchování libovolných informací vztažených ke konkrétní části prostoru. Například může odkazovat na objekty, případně jejich obálky, které se v dané části nacházejí, což umožňuje efektivně uchovávat informace o rozmístění objektů ve scéně s podporou rychlého vyhledávání a to je pro detekci kolizí zásadní. Dělení prostoru ve dvou úrovních a odpovídající octree je znázorněn na Obrázku 2.9.

Při hledání bodu v prostoru dle daných souřadnic se pak postupuje jednoduše tak, že v rámci každého uzlu se provede porovnání souřadnic hledaného bodu se souřadnicemi všech potomků aktuálního uzlu a rozhodne se, kterou cestou bude prohledávání pokračovat. Alternativou je vyhledávání poli v rámci rodičovského uzlu, uchovávajícím souřadnice všech jeho potomků, což omezí cyklické přistupování k potomkům.



**Obrázek 2.9 Vlevo prostor rozdělený ve dvou úrovních a vpravo odpovídající octree [20]**

Další variantou octree je pak například tzv. *linear octree* [21], který je namísto skutečné stromové struktury reprezentován pouze lineárním polem, které obsahuje pouze listové uzly. V tomto případě tedy není možné přistoupit k uzlům uvnitř stromu. Tento přístup šetří paměťové nároky, avšak za cenu pomalejšího prohledávání. Tento přístup tedy nalézá uplatnění pouze ve specifických případech, kde je rozhodující pouze znalost uzlů a kde jsou kritické paměťové nároky.

# 3 Jevištní technika a její řízení

Tato kapitola obsahuje přehled jevištní techniky, která je ve větší či menší míře standardem v současných divadlech prakticky po celém světě. Nutno však podotknout, že tato problematika zahrnuje široké spektrum disciplín, nejen z oblasti elektrotechniky, strojírenství, automatizace a řízení, které jsou navíc vázány na specifické požadavky prostředí divadla. Jedná se tedy pouze o stručný přehled klíčových aspektů. Popis jeviště vychází z klasického, tzv. kukátkového uspořádání divadla, tedy z konceptu, který známe z nejtypičtějších dnešních divadel. Jeviště je zde od hlediště výrazně odděleno, obvykle portálem a oponou. Postupný vývoj tohoto konceptu sahá až do 16. století a nejvíce se o něj zasadilo divadelnictví v zemích, které vždy byly velmocemi v oboru, Francie a Itálie.

Jevištní technika pojímá širokou paletu zařízení jak strojního, tak elektrotechnického charakteru. Může se jednat o zvukovou aparaturu nebo osvětlovací techniku, točny, hydraulické stoly či závěsné mechanismy. V zásadě lze ale říci, že některé zařízení lze označit jako pohyblivé a některé jako nepohyblivé, přičemž ta nepohyblivá (světla, reproduktory) mohou být umístěna buď na statické konstrukci či na stěně a nebo na některém pohyblivém zařízení. Ačkoliv divadelní řídicí systémy mohou spravovat i osvětlení a ozvučení scény, v dalším textu je uvažováno řízení pouze pohyblivých komponent.

## 3.1 Přehled mechanického zařízení jeviště

Pro pohyblivé komponenty mechanizace jeviště se v praxi užívá základní rozdělení na horní a dolní mechanizaci podle toho, ve kterém prostoru jeviště je umístěna [22]. Dolní mechanizace umožňuje měnit uspořádání podlahy jeviště, a její strojovna se z většiny nachází dole pod podlahou. Výjimku mohou tvořit například baletní vozy, které se pohybují po podlaze a pohonnou jednotku mají integrovanou v sobě.

Horní mechanizace, jak už název napovídá, má své místo v prostoru nad podlahou jeviště a její strojovna bývá umístěna často poměrně vysoko, těsně pod střechou budovy v prostoru nazývaném provaziště. Většinou se jedná o různé tahy, tedy závěsná zařízení, určená pro zavěšení kulis, zmiňovaných světel nebo celých osvětlovacích baterií, ba dokonce samotných herců.

### *Horní mechanizace*

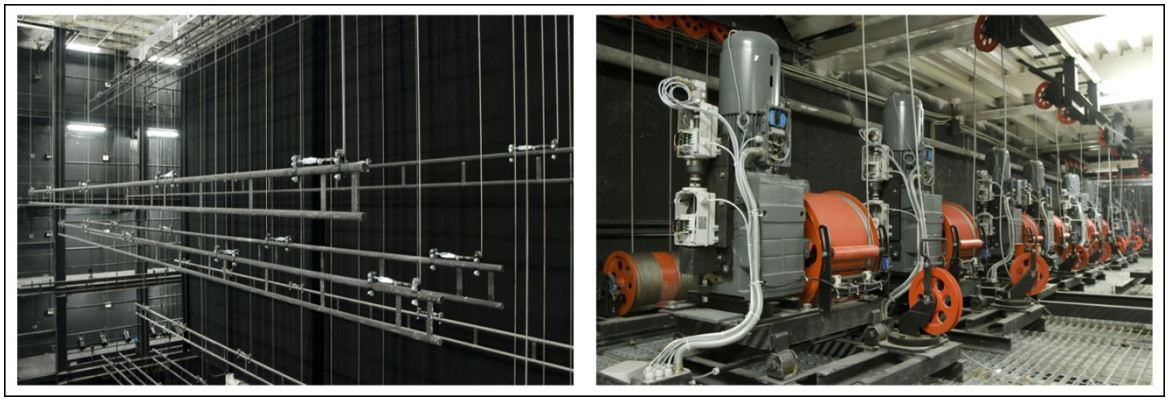
Je tedy umístěna v provazišti nahoře nad jevištěm a typicky se skládá z většího počtu tahů, tedy pohonů s navijáky a lany, na kterých mohou být zavěšeny kulisy a jakákoliv potřebná dekorace. V praxi se setkáme se dvěma základními typy tahů [22]:

- *Prospektové tahy* – jsou zakončeny tahovou tyčí, jejíž délka je rovna šířce hrací plochy jeviště. Na tahovou tyč jsou pomocí dvou či více lanových závěsů zavěšovány dekorace. Případně lze zátěž připevnit pevně do kleštin na tahové tyči, což je vhodnější například pro osvětlovací baterie.
- *Bodové tahy* – jedná se pouze o jeden lanový závěs, na jehož konci je upínací mechanizmus pro bodové zavěšení dekorace.

Prospektové tahy jsou připevněny napevno k roštu provaziště s pravidelnou roztečí tak, aby pokrývaly pokud možno celou plochu jeviště od portálu až dozadu po horizont. Bodové tahy mohou být i mobilní. Rychlost pohybu tahů bývá obvykle od 0,5 m/s do 1 m/s, někdy dokonce až 2 m/s [23]. Nosnost tahů bývá poměrně vysoká, často více než jednu tunu. Mechanismy tahů tedy

musejí být konstruovány s ohledem na značný výkon. A jelikož mnohdy bývá zapotřebí dosti jemných posunů, je zřejmé, že nároky na spolehlivost zde budou vysoké.

Pohon tahů je zajištěn zpravidla motorově. V současné době je hojně využíváno asynchronního elektromotoru poháněnjícího buben pro navíjení závěsného lana [24]. Rychlost motoru je regulovaná pomocí frekvenčního měniče. Toto řešení se v posledních letech ukazuje být výhodné i díky přijatelné ceně. Lze se též setkat s hydraulickým motorem (lineárním nebo rotačním), dříve i s motory stejnosměrnými. Součástí každého tahu jsou i bezpečnostní prvky jako brzda (bývá zdvojená) a senzory v mezních polohách (rovněž mohou být i zdvojené). Stále může být ale užitečný i ruční pohon. A to ať při nějaké speciální potřebě přesného nebo rytmického spuštění předmětu třeba podle hudby, tak třeba v případě náhlého problému či nehody. Motorové tahy proto obsluze též nabízejí možnost ručního pohonu. Méně často, spíše na menších scénách, se dokonce i dnes můžeme setkat s ryze ručními pohony.



**Obrázek 3.1 Horní mechanizace: vlevo prospektové tahy, vpravo pohonné jednotky v provazišti [25]**

## *Dolní mechanizace*

Dolní mechanizace se nachází v dolních partiích celého prostoru. Podlaha jeviště totiž zpravidla nebývá pevná a jednolitá, nýbrž se skládá z více samostatných dílů, jejichž pohybem a nastavením lze její uspořádání nastavit podle požadavků dané scény. Mezi nejběžnější zařízení spadající pod dolní mechanizaci můžeme zařadit například tyto [22]:

- *Jevištní stoly* – realizují vertikální pohyb ve spodní oblasti jeviště, jsou schopny vyvýšit se nad úroveň podlahy, rozšiřují variabilitu uspořádání objektů na jevišti nebo mohou tvořit reliéf podlahy jeviště.
- *Propadla* – také realizují vertikální pohyb, ale zajišťují spíše spojení mezi prostorem pod jevištěm a úrovní hrací plochy, umožňují vyjetí herce či jiného objektu na hrací plochu a zpět pod jeviště.
- *Točny* – realizují otáčivý scénický pro nejrůznější scénografické potřeby.
- *Jevištní vozy* – realizují horizontální pohyb po jevišti. Bývají umístěny obvykle na bočních jevištích a pomocí lan mohou pohybovat s objekty na scéně.

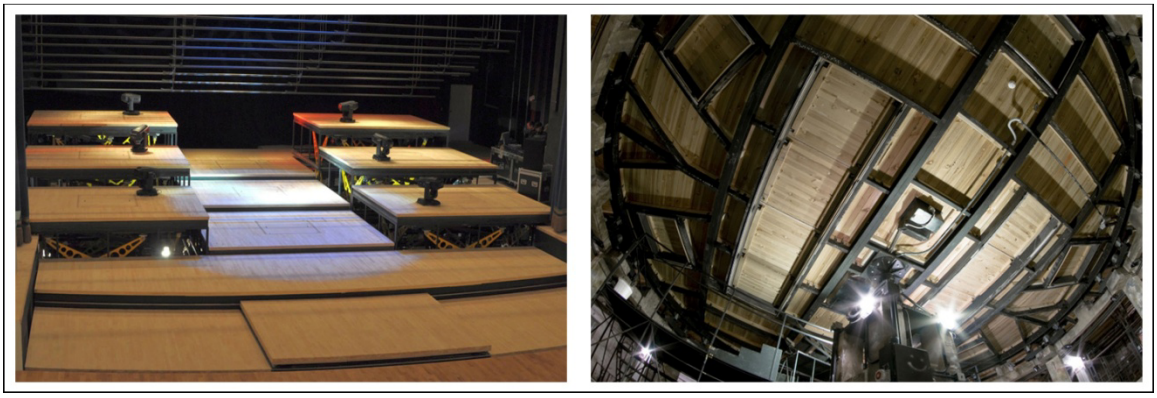
Jevištní zdvižné stoly bývají poháněny hydraulicky nebo pohony se šroubovými či řetězovými mechanismy (například Serapid). Ruční pohon zde kvůli někdy až několikatunové hmotnosti samotných stolů není dost dobře možný. Rychlost pohybu stolů bývá nízká, obvykle maximálně několik dm/s, což vyplývá z jejich účelu.

K pohonu propadel bývá podobně jako u tahů horní mechanizace použito asynchronních elektromotorů a k jejich regulaci frekvenčního měniče, přičemž může podporován i neregulovaný

režim pohybu. Aktuální poloha zařízení je snímána inkrementálním snímačem polohy. Součástí zařízení musí nezbytně být bezpečnostní prvky, senzory hlídající, aby zařízení při pohybu nepřekročilo povolené meze, a jiné.

Točny mohou být buďto jednoduché, čili kruhové, anebo složené, tedy prstenec a kruh uvnitř něho. Rozměry bývají rozličné v závislosti na celkové velikosti jeviště. K pohonu slouží obvykle též asynchronní elektromotory. Lze se setkat i s případem, kdy točna uvnitř obsahuje zdvižné stoly (např. v Janáčkově divadle v Brně).

Jevištní vozy bývají, v případě menších scén, poháněny ručně, ovšem ve velkých divadlech jsou obvyklejší rovněž motorové. Veškeré popsané prvky jevištní mechanizace mohou být používány jako scénické prvky, tedy sloužící k přestavbě scény odehrávající se zpravidla mimo pohled diváka (za oponou, o přestávce, a nebo je alespoň během této činnosti divákova pozornost umě odvedena osvětlením či dějem v popředí), ale i jako inscenační, kdy se jejich činnost přímo stává součástí děje (např. herec pomocí otočné scény pomalu prochází různými prostředími hry, apod.) [23]. Na jevišti se ještě nachází opona a někdy i další pomocné prvky, jako posuvné stěny, portály, horizonty a další pomocné mechanismy.



**Obrázek 3.2 Dolní mechanizace: vlevo podlaha jeviště složená ze zdvižných stolů, vpravo mechanismus točny [25]**

## 3.2 Řídicí systémy a řízení jevištní techniky

Jedna možnost je ovládat divadelní techniky manuálně a druhá možnost je spustit sestavený program a nechat tak ovládat scénu zcela automaticky. Protože inscenační požadavky divadel zahrnují obě možnosti, divadelní ŘS zpravidla umožňují tyto režimy dynamicky kombinovat. Pro manuální řízení v základním pojetí slouží sestava ovládacích prvků sestávající z tlačítek, pák či potenciometrů, jejichž prostřednictvím obsluha může přímo spouštět a zastavovat jednotlivé pohony nebo měnit jejich rychlost či směr. Často se také objevují možnosti seskupování pohonů do skupin za účelem jejich koordinovaného pohybu. Nutno však zmínit, že v dnešní době tento princip ovládání bývá instalován spíše jakožto záložní systém. Primárně je užíváno pohonů disponujících čidly snímajícími jejich polohu a tím umožňujících budování automatizovaných systémů řízení, řídicích systémů. Zmíněné hardwarové prvky bývají pak doplňovány nebo zcela nahrazovány grafickými displeji, často dotykovými.

Napřed zařazuji podkapitolu, věnující se nejprve obecněji řídicím systémům a poté přibližující aspekty řídicích systémů divadelních, neboť je toto vhodným úvodem pro pochopení principů a požadavků na UI. Interakce s obsluhou je pak předmětem kapitoly 3.3, kde se věnuji samotnému *uživatelskému rozhraní* (dále též *UI*) divadelních řídicích systémů, grafickému UI a nakonec 3D modelování a simulaci využitelnému na poli řízení jevištní techniky.

## *Řídicí systémy obecněji*

V automatizaci se dnes setkáváme zpravidla se řízením, které pracuje s určitou zpětnou vazbou, což typicky představuje data poskytovaná čidly a senzory. Na základě vstupních dat pak systém zajistí adekvátní reakci. V rámci těchto systémů existuje několik přístupů k řízení [26]:

- *On-off řídicí systémy* – je řízení se zpětnou vazbou založené na nějaké jednoduché podmínce, například řízení termostatu, který sepne, pokud teplota spadne pod určitou mez. Tyto systémy bývají levné a efektivní, pro jednoduché systémy zcela dostatečné.
- *logické řídicí systémy* – získávají informace o aktuální situaci na zařízení pomocí nejrůznějších snímačů nebo čidel a prostřednictvím svých logických obvodů na tyto příchozí binární data vhodným způsobem reagují. Zpravidla se vyznačují poměrně jednoduchým návrhem a dokážou zastat velmi komplexní operace; bývají užívány například pro výtahy nebo myčky nádobí.
- *Lineární řídicí systémy* – je řízení se zpětnou vazbou produkující řídicí signál, který je lineárně závislý na nějaké proměnné (obecně i na více proměnných), pomocí které lze parametrizovat řídicí proces. Tento přístup je v praxi poměrně rozšířený, může být implementován např. na principu PID regulátoru.
- *Fuzzy řízení* – využívá empirických znalostí o řízeném procesu, tzv. báze znalostí. Probíhá ve třech fázích, kdy nejprve, během *fuzzyfikace* je báze znalostí převedena na fuzzy data, poté během *inference*, ústřední fáze, probíhá mechanismus využívající rozhodovací pravidla a mění vstupní množinu na výstupní. *Defuzzyfikace* nakonec výstupní množinu převede na hotová výstupní data [27].

Se řízením bývá úzce spjat proces regulace, konstruován zejména proto, aby řízení procesu nevyžadovalo nepřetržitou pozornost a ruční zásahy operátora. Jako příklad v praxi rozšířeného přístupu zmíním PID regulátor. PID regulátory patří mezi spojité regulátory. Název vychází ze tří složek, které se na regulaci podílejí – proporcionální, integrační a derivační [28]. PID regulátor pak kombinuje chování těchto tří složek. Vstupem každé z nich je regulační odchylka a výstupem je akční veličina. Proporcionální složka pak je prostý zesilovač, kdy je vstupní signál násoben konstantním činitelem, regulační odchylka je tedy akční veličině přímo úměrná. Integrační složka je taková složka, kdy akční veličina je přímo úměrná integrálu regulační odchylky. Konečně v případě derivační složky je akční veličina přímo úměrná derivaci regulační odchylky. PID regulátor si pak lze nakonec představit jako součet proporcionální, integrační a derivační složky [29]. Vyskytují se i redukované varianty PID regulátoru – P regulátor, PD regulátor a PI regulátor. Takovéto vyřazení některé ze složek může být nezbytné kvůli stabilitě systému, zjednodušení implementaci v případech kdy je to vhodné, atp.

Dalším hlediskem, rozdělujícím přístupy k implementaci ŘS je jeho struktura a zejména úroveň decentralizace řízení. Z tohoto pohledu lze ŘS rozdělit na tři základní skupiny [30, 31]:

- *Centralizované řízení* – realizuje řídicí proces běžící pouze na jedné centrální výpočetní jednotce. Dnes se tohoto přístupu využívá pouze při malých aplikacích, nebo tam, kde nejsou kladeny příliš vysoké nároky na spolehlivost.
- *Distribuované řízení* – realizuje systém, jehož dílčí procesy jsou distribuovány na více výpočetních jednotek. Tato skutečnost přináší jednak vyšší efektivitu, neboť zátěž řídicího procesu je rozložena mezi více výkonných jednotek a rovněž s sebou nese vyšší spolehlivost.

- *Hierarchické distribuované řízení* – Jednovrstvé distribuované systémy s několika výpočetními jednotkami na jedné úrovní propojených sběrnicí jsou vhodné pouze pro malé až střední systémy. Pro rozsáhlejší systémy je třeba užít vícevrstvého hierarchického distribuovaného řízení. Zde probíhá řízení v hierarchickém systému vrstev, kde nižší vrstvě (například úroveň senzorů a akčních členů) je nadřazena vrstva vyšší (například úroveň místního bezprostředního řízení) a ta je zase např. prostřednictvím Ethernetu propojena s nějakou další nadřazenou vrstvou řízení.

Hardwarové řešení výpočetních jednotek může být u jednodušších systémů postaveno na jednoduchých jednoúčelových logických obvodech nebo mikrokontrolérech. Rozsáhlejší systémy bývají postaveny na technologii programovatelných automatů (PLC – Programmable Logic Controler), případně na nějakém typu průmyslových počítačů.

## *Metody řízení v divadlech*

I divadelní řídicí systémy zacházejí se zpětnou vazbou, s daty získanými prostřednictvím akčních členů. V případě jeviště jsou nejzákladnějšími akčními členy čidla snímající polohu, konkrétně tzv. inkrementální rotační čidla (Incremental Rotary Coder – IRC) a absolutní rotační čidla (ARC) [23], přičemž ARC je schopné si zapamatovat polohu pohonu i v případě vypnutého stavu. Toto je nasnadě, neboť stěžejními předměty řízení na divadelním jevišti jsou regulace a sledování polohy pohonů všech výše zmíněných typů, definice softwarových mezí pro pohony a automaticky řízená jízda v rámci těchto mezí, společná jízda pohonů v synchronním režimu, apod. Tedy, prakticky veškeré operace s pohony vyžadují informaci o jejich poloze.

Regulované řízení pohonů je klíčové – nedostatečná přesnost může snadno způsobit poškození zejména hmotnějších kulis při jejich manipulaci či přinejmenším špatný dojem u diváků. Dojezd na cílovou pozici musí být přesný a navíc je třeba zajistit pozvolný rozjezd a zastavení. Cílem regulačního algoritmu je zajistit takovouto optimální trajektorii, což není jednoduchý úkol a zpravidla při vývoji takového algoritmu kromě analytických metod založených na fyzikální analýze celé soustavy hrají nikoliv bezvýznamnou roli i empiricky nabyté znalosti a z nich odvozené regulační parametry. Zpravidla bývá v praxi užíváno PID regulátorů (viz výše).

Řízení jeviště může být navrženo jako centralizované nebo distribuované. V praxi se lze setkat s oběma přístupy. Pokud ovšem chceme hodnotit, který je vhodnější, je nejprve nutné si uvědomit strukturu celého systému – lze detekovat pět úrovní řízení [23]:

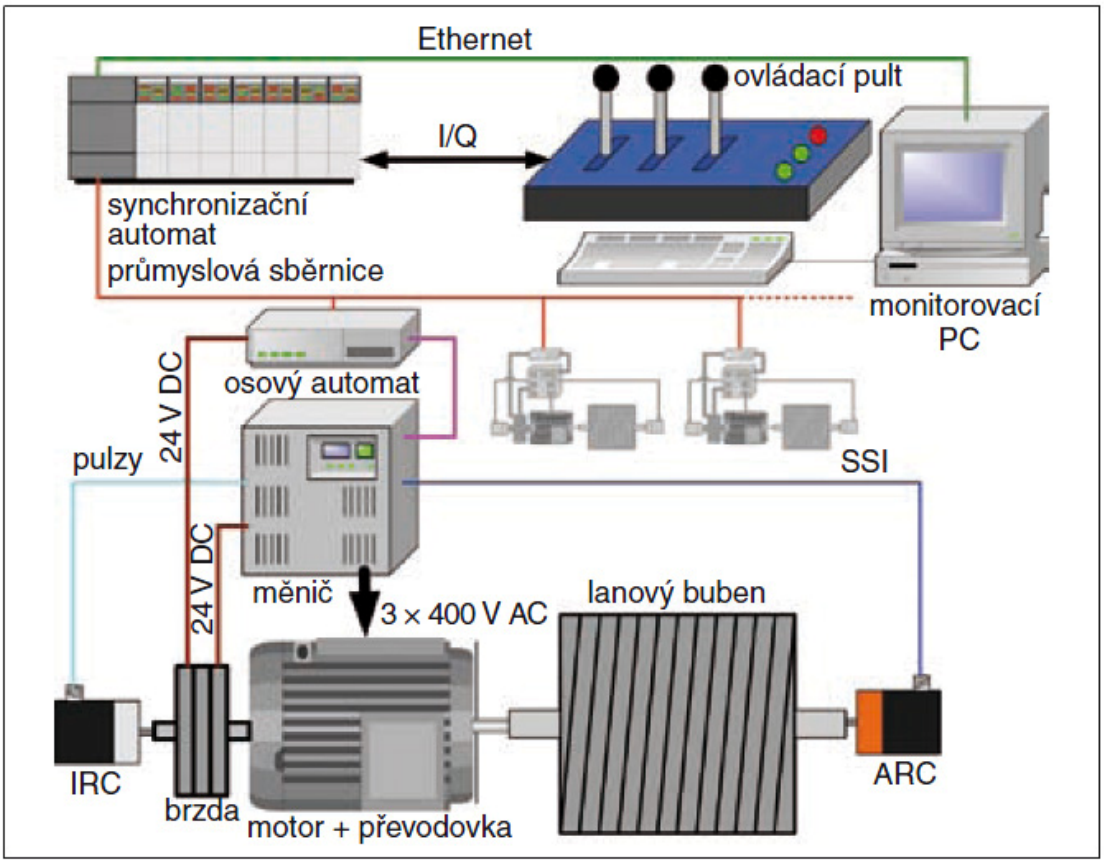
1. *Vlastní pohon* – zahrnuje hardwarové zařízení pohonu a jeho příslušenství (snímání a regulaci rychlosti, bezpečnostní koncové spínače, ovládání brzd, apod.)
2. *Řízení jednoho pohonu* – disponuje čidlem pro snímání polohy a regulátor polohy pro daný pohon. Regulátor musí být schopen řídit jízdu v požadovaném směru, požadovanou rychlostí a v požadovaných mezích. Disponuje i potřebnou diagnostikou, jako detekce poruch nebo hlídání regulační odchylky. Řídicí prvek této úrovně bývá označován jako *osový automat (případně osový regulátor)*.
3. *Řízení skupiny pohonů* – má za úkol zejména synchronizaci seskupených pohonů. Na tomto místě je tedy implementován koordinační algoritmus, který řídí podřízené pohony prostřednictvím jejich osových automatů. Pro tuto úroveň existuje označení *synchronizační automat*.
4. *Řízení představení* – zahrnuje správu předdefinovaných jízd pohonů a proměn scény. Řídí celé skupiny pohonů, udává parametry jízdy.

5. *Uživatelské rozhraní* – představuje nejvyšší úroveň, interakci s obsluhou, která zde má možnost zasáhnout do řídicího procesu, nastavovat parametry pro automatické řízení na všech nižších úrovních a koordinovat jeho běh.

*Centralizované řízení jeviště* [23] staví na výkonné centrální výpočetní jednotce, počítači, na kterém je implementováno řízení na úrovních 2 až 4, popřípadě i 5 (je-li UI implementováno např. prostřednictvím dedikovaného ovládacího pultu – i několika – jedná se stále o centralizovaný řídicí systém). Úroveň 1 pak často bývá zajišťována frekvenčním měničem.

Toto řešení vykazuje výhody i nevýhody. Fakt, že je vše zajištěno jedinou výpočetní jednotkou, přináší dostupnost všech potřebných dat a signálů a eliminuje tak problém meziúrovňové komunikace a komplikace s přenosem dat. Takové uspořádání má velmi praktický přínos ve chvílích oprav a údržby, neboť veškerá technika se nachází na jednom místě. Na vrub centralizovanému řízení můžeme připsat nutnost velmi rozsáhlé kabeláže pro komunikaci s pohony, ačkoliv se začíná objevovat tendence k bezdrátové komunikaci, která by tento problém eliminovala. Evidentní je ovšem pravděpodobnost vzájemné závislosti pohonů, kdy výpadek jednoho může snadno způsobit pád ostatních a obtížněji realizovatelná rozšiřitelnost (např. instalace dalších pohonů).

*Distribuované řízení jeviště* přináší způsob, jak od sebe jednotlivé úrovně řízení fyzicky, hardwarově oddělit, přičemž na druhou stranu není nezbytně nutné každou zcela izolovat. Je vhodné najít optimální míru distribuce. Například úrovně 1 a 2 (pohon a k němu příslušející osový regulátor) bývají často situovány společně na jednom místě. 3. úroveň pak může představovat více fyzicky oddělených, vzájemně nezávislých synchronizačních automatů, komunikujících pouze s osovými regulátory „svých“ pohonů a s nadřazeným uzlem vyšší úrovně. Úrovně 4 a 5 už pak může být vhodné spojit do jediné jednotky obsahující uživatelské rozhraní. Ilustraci jednoho z možných řešení vidíme na Obrázku 3.3.




**Obrázek 3.3 Jedno z možných řešení distribuovaného řízení tahu horní mechanizace. [23]**

Tento přístup oproti centralizovanému vykazuje přehlednější komunikační síť. Výpadek zde ovšem může znamenat pád komunikace mezi celými dvěma úrovněmi a je více než vhodné zajistit dostatečná opatření, například pomocí redundance. Redundance navíc obecně umožňuje realizovat určitou míru odolnosti vůči chybám. Rozhodující předností jsou pak možnosti rekonfigurace a rozšiřitelnosti systému. Distribuované řízení divadel tedy často bývá pojato coby vícevrstvé hierarchické distribuované řízení [23].

Kromě spolehlivé regulace správného návrhu architektury řídicího systému, hraje klíčovou roli rovněž uživatelské rozhraní, jehož kvalitní návrh je zásadní pro efektivitu i bezpečnost celého systému. Návrh UI musí být přizpůsoben tak, aby co nejvíce usnadňoval splnění požadavků které jsou kladeny na obsluhu řídicího systému při různých situacích, které mohou během manipulace nastat (více o UI divadelního řídicího systému v kapitole 3.3).

### *Bezpečnostní požadavky na divadelní řídicích systémů*

V automatizaci, a to zejména tam, kde figurují zařízení větších rozměrů a vyšší hmotnosti, jsou na bezpečnost přirozeně kladeny vysoké nároky. Jako příklad poslouží automatizace v průmyslu. Ať už u těžkého těžebního průmyslu, tak třeba u montážní linky ve výrobní hale – všude hrozí vysoké riziko ujmy na zdraví. Z toho důvodu je každý takový automatizovaný provoz vybaven celou řadou jak vizuálních, tak zvukových varovných mechanismů. Obsluhující pracovníci nejenže jsou patřičně proškoleni, navíc jsou před jakýmkoli nebezpečím vždy důsledně varováni [24].

Pokud se nyní přeneseme do prostředí divadelního jeviště, nemůže nám ujít že kulisy a nejrůznější další vybavení, se kterým se zde manipuluje, si se svými rozměry a hmotností nic nezadá se zařízením výrobní linky v továrně. Ovšem divadlo je stánkem umění. Herec stojící na jevišti uprostřed svého výstupu je soustředěn na svůj výkon a nemá v tu chvíli příliš prostoru uvažovat nad faktem, že neustále kolem něho, ba dokonce zavěšené vysoko nad jeho hlavou čeká potenciální nebezpečí. Je rovněž jasné, že zvuková varovná znamení zde ze zřejmých důvodů nejsou použitelná a vizuální jen v omezené míře (nesmí být viditelná pro diváka). Hlavní odpovědnost nejenom za bezpečí osob, ale i materiálu pohybujícího se na divadelní scéně, nesou osoby, které obsluhují jevištní techniku. K tomu, aby mohli úspěšně dostát své odpovědnosti, by jim měl pomáhat důmyslně navržený řídicí systém [23].

Součástí procesu návrhu takového systému tedy bývá důkladná analýza rizik, která šetří všechny potenciálně nebezpečné situace, možné chyby a havárie. Na základě jejích výsledků jsou implementována patřičná opatření. Chyby a potenciální nebezpečí havárie mohou souviset už se samotným hardwarem. Některým těmto nebezpečím lze předejít instalací bezpečnostních prvků jako snímačů mezních poloh, zdvojených brzd, záchranných stop tlačítek, nastavením frekvenčního měniče tak, aby v případě přetížení tahu motor okamžitě zastavil, atp. Pokud již dojde k některé z těchto chyb, je nutné pomocí vhodných opatření co nejrychleji dosáhnout bezpečného stavu (například zastavení činnosti zařízení). Přesné směrnice pro tato opatření specifikují příslušné normy (mezi ty základní patří zejména ČSN EN 954-1, ČSN EN 61511 a ČSN EN 61508). Dosažená úroveň bezpečnosti divadelního řídicího systému dle daných norem je stvrzována certifikací SIL (Safety Integrity Level), v Evropě udělovanou organizací TÜV. V současné době je ve vyspělých zemích standardem úroveň SIL3.

## 3.3 Uživatelské rozhraní divadelního řídicího systému a možnosti interakce s obsluhou

Uživatelské rozhraní obecněji vyjadřuje velmi široký pojem, zahrnující prostředky, které realizují interakci mezi nějakým systémem, typicky strojem a člověkem, tedy obsluhou stroje. Mohou to být prostředky jak softwarové, tak hardwarové nebo jejich kombinace. Na tomto místě je tedy systémem myšlen celý řídicí systém divadelního jeviště a za UI jsou považovány veškeré SW i HW prostředky podílející se na interakci s obsluhující osobou.

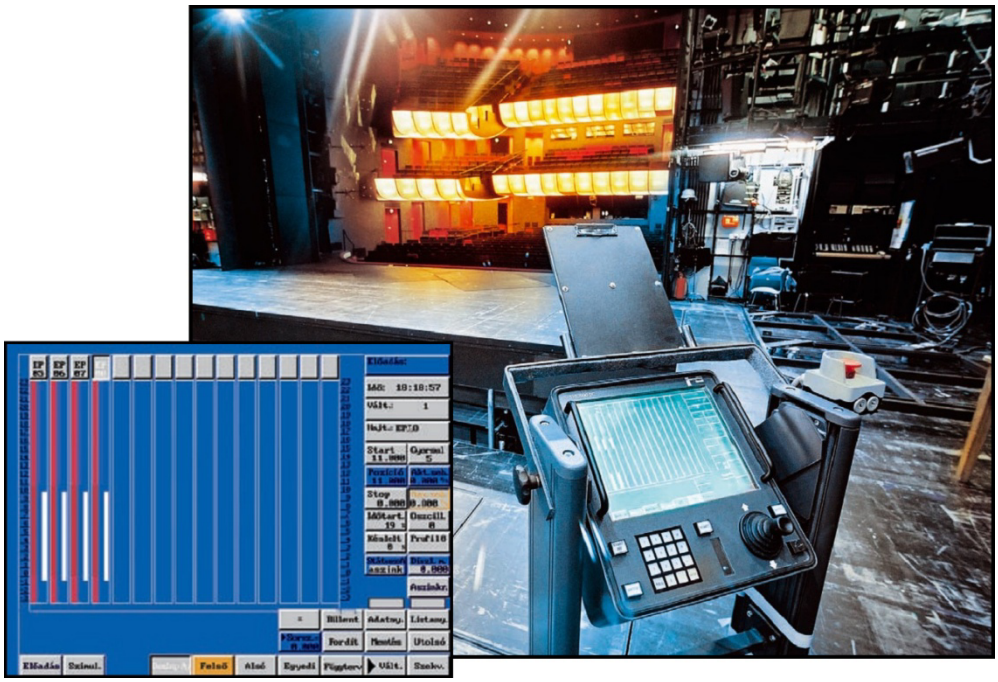
Návrh takového UI musí vycházet z požadavků které staví na technických parametrech zařízení, na situacích, které mohou na jevišti nastat a na praktických zkušenostech. Lze říci, že práci obsluhy divadelního řídicího systému lze shrnout do tří okruhů činností, při kterých dochází k manipulaci s jevištní technikou [23]:

- *Definice představení* – představuje „off-line“ manipulaci se systémem, kdy obsluha na základě požadavků režie či poznatků ze zkoušek představení prostřednictvím UI definuje jednotlivé jízdy pohonů nebo proměny celé scény a jejich následnost.
- *Vlastní představení* – obsluha během probíhajícího představení spouští jednotlivé předdefinované proměny scény, přičemž má možnost reagovat na nečekané situace na jevišti a např. zpomalit některou skupinu pohonů, je-li to náhle nutné.
- *Přestavba jeviště* – probíhá během sestavování scény, zpravidla za účasti jevištních techniků („*kulisáků*“). Obsluha tak musí být schopna pružně reagovat na jejich požadavky.

V prvním případě obsluhu při práci příliš netlačí stres z právě probíhající hry, ani neustále nové požadavky jevištních techniků. Ovšem ve druhém a zejména pak v posledním případě je zásadním kriteriem kvalitního návrhu UI snadný přístup k často užívaným funkcím pro základní manipulaci s pohony, jako je volba pohonu požadovaného pro jízdu a volba režimu jízdy, uvedení do pohybu a zastavení. Rovněž je třeba brát na zřetel fakt, že pozornost obsluhujícího pracovníka (též strojníka) bývá velmi vytížena, neboť musí dávat pozor na obslužné prvky uživatelského rozhraní, které má před sebou a zároveň úzkostlivě vnímat dění na jevišti, případně ještě přijímat další požadavky od jevištních techniků, režiséra či dalších lidí na scéně. Forma obslužných prvků a jejich rozvržení jsou proto klíčovými aspekty.

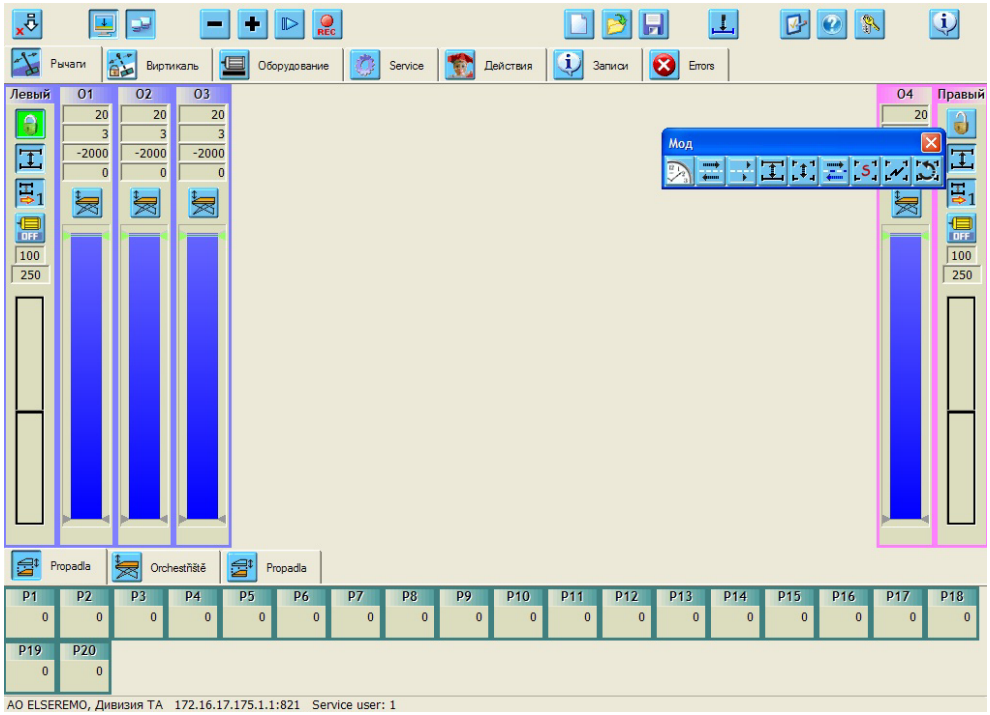
Standardním hardwarovým obslužným prvkem je v současnosti ovládací panel, často strohého avšak účelného průmyslového vzezření. Obvykle bývá osazen displejem a mezi dalšími ovládacími prvky dominují páky či podobné mechanické prostředky umožňující uvést pohon do pohybu a pohodlně a přitom bezpečně kontrolovat průběh jízdy až po zastavení. Ovládací panely mohou rovněž disponovat pomocnými funkčními tlačítky, popřípadě numerickou klávesnicí nebo touchpadem. Důležitým prvkem je bezpečnostní tlačítko *Totalstop* [33], po jehož stisku dojde k okamžitému zastavení jízdy všech pohonů a odstavení systému z provozu. Rovněž bývá přítomna určitá forma bezpečnostního zámku, mechanického nebo elektronického (např. autentizace SW klíčem pomocí USB flash nosiče), z důvodu zabezpečení proti manipulaci s řídicím systémem nepovolanou osobou [33].

Pro ilustraci na Obrázku 3.4 vidíme ovládací panel výrobce *Bosh Rexroth* s dotykovým displejem, num. klávesnicí a tlačítkem Totalstop a detail GUI. Jde o méně robustní panel poskytující určitou mobilitu, může tak být v rámci jeviště přemístěn tam, kde je ho právě třeba. Rozumné je pak řešení UI s více panely – alespoň jedním „hlavním“ statickým a v závislosti na velikosti dané scény až několika dalšími mobilními panely rozmístěnými na strategických místech jeviště.



**Obrázek 3.4 Ovládací panel firmy Bosh Rexroth s detailem grafického uživatelského rozhraní [32]**

Displej zobrazuje grafické uživatelské rozhraní (též GUI), které má za úkol obsluze sdělovat jednak potřebná diagnostická data o řídicím systému především pak komplexní ale zároveň přehlednou a výstižnou informaci o aktuální situaci a stavech pohonů na scéně. Displeje instalované na ovládacích panelech bývají dnes již standardně dotykové, a tak ke sdělovací funkci displeje se přidává i možnost ovládání prvků řídicího systému jeho prostřednictvím. Ne vždy je ale toto šťastné řešení. Při nevhodném uspořádání GUI totiž může být řízení prostřednictvím dotykového displeje problematické z hlediska bezpečnosti a rychlosti volby funkcí. Důraz na kvalitní uspořádání GUI je tedy dalším zásadním faktorem při návrhu.




**Obrázek 3.5 Grafické uživatelské rozhraní řídicího systému firmy Elseremo [33]**

Na Obrázku 3.5 je GUI k řídicímu systému moravského výrobce *Elseremo*, kde je patrné grafické znázornění drah čtyřech aktivních pohonů přiřazeným ke dvěma pákám ovládacího panelu. Jejich poloha je naznačena graficky znázorněným pohyblivým „sloupcem“, což je řešení velmi efektivní (za předpokladu, že je zachována jeho střídmost – není přehlceno grafikou) a v různých variacích typické i u jiných výrobců (viz také Obrázek 3.4).

GUI kromě samotného řízení jízdy pohonů a zobrazování jejich polohy a stavu umožňuje i jejich seskupování a specifikaci synchronní jízdy a rovněž definici průběhu představení, tedy nastavení následnosti jednotlivých jízd pohonů a celých proměn scény. Také zahrnuje další širokou paletu funkcí – samozřejmě nejrůznější oblasti nastavení celého systému, diagnostiku pohonů a jejich management. Tyto součásti již ale vynechávám z bližšího popisu a příští podkapitolou se zaměřuji na 3D vizualizaci divadelní scény [33].

### *3D modelování a vizualizace v oblasti divadelních řídicích systémů*

Ačkoliv lze předem vytvořit následnost akcí, nelze však jízdy pohonů definovat v čase a nechat představení řídit zcela automaticky. Herci jsou také jen lidé, načasování výstupů nikdy není stoprocentně spolehlivé a průběh hry je ovlivňován řadou dalších vlivů včetně samotného diváka. Ačkoliv je zde inspicient, člověk udávající pokyny k „dalšímu kroku“ představení, stále z velké části zůstává na odpovědnosti obsluhy, aby byl pro zahájení jízdy zvolen ten správný okamžik. [23]

Schematické 2D animované zobrazení aktivních pohonů, byť se zobrazenou informací o přesné poloze a s přehledným děleným do skupin, stále není dostatečné pro poskytnutí úplného pojmu o situaci ve scéně. Ačkoliv obsluhující strojník je již díky svým zkušenostem schopen si situaci do značné míry představit, nemusí tato míra být dostatečná. Pro efektivní a bezpečnou manipulaci se zařízením je nevyhnutelné realizovat nějaký další způsob zobrazení scény poskytující obsluze komplexnější reálnou informaci o konfiguraci objektů na jevišti. Tyto požadavky je jistě možné splnit zavedením technologie 3D modelování do oblasti divadelních řídicích systémů.

Pokud vytvoříme prostorový model jeviště, kde budou obsaženy pohony horní i dolní mechanizace a pokud bude navíc možné zahrnout i kulisy a objekty, které jsou pomocí pohonů manipulovány a umožníme-li takovýto model zobrazit na displeji ovládacího panelu, s výhodou ho pak lze propojit s komunikačním rozhraním řídicího systému tím vytvořit animovanou vizualizaci pohybujících se pohonů, včetně manipulovaných břemen, v reálném čase. Pro obsluhu již tak nebude nezbytně nutné udržovat si nepřetržitý výhled do všech prostorů jeviště (ačkoliv přímý výhled pochopitelně počítačovou vizualizací zcela nahradit nelze).

Patrně ale ještě větší přínos 3D vizualizace slibuje ve fázi definice představení, kdy lze nanečisto simulovat jednotlivé dojezdy tahů a nebo přehrát celé obrazy z chystané hry ještě před jejím začátkem a bez nutnosti manipulace se skutečným zařízením (mezitím se může na jevišti fyzicky připravovat zcela jiné představení).

### *Příklady z praxe*

Výhody pro divadlo, plynoucí z 3D vizualizace a simulace provozu jeviště jsou tedy evidentní. Na tomto místě bych uvedl pro ilustraci několik příkladů z praxe, neboť tato technologie je v současnosti na světových divadelních scénách poměrně rozšířená.

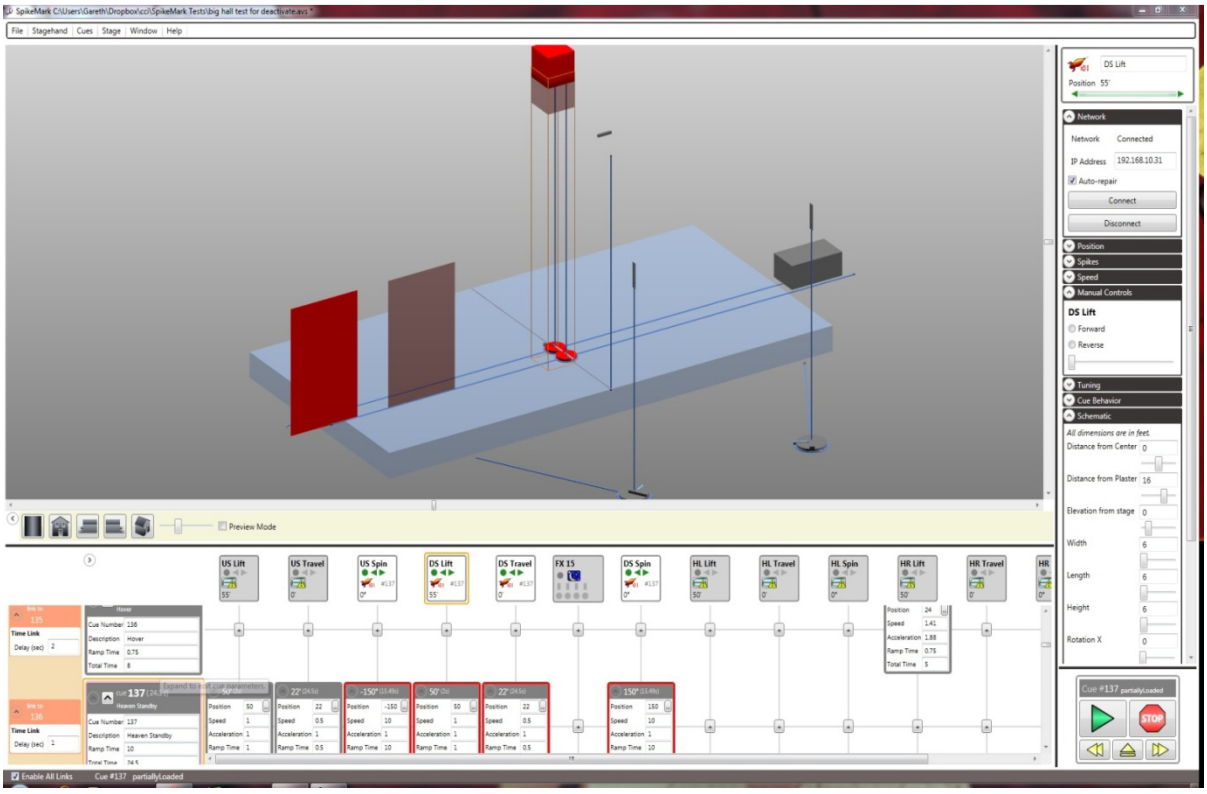
Například britská společnost Stage Technologies jako součást softwarového prostředí eChameleon [34], které instaluje na své ovládací panely, nabízí i 3D vizualizátor scény. Uživatel tak má možnost přepínat mezi tabulkovým přehledem pohonů a 3D pohledem na scénu a nebo mít před sebou obě zobrazení najednou, pohodlněji například prostřednictvím stacionárního panelu Acrobat G6, který disponuje dvěma monitory. Kromě toho je k dispozici nástroj Sculptor Animation Toolkit, který umožňuje předdefinovat uspořádání scény. Obsahuje Add-on pro 3D Studio Max. Toolkit

umožňuje komponovat obraz na jevišti z jednotlivých scénických elementů, simulovat jízdu, vytvářet animace. Rovněž je možné importovat modely z 3D Studia Max. Třetím firmou nabízeným nástrojem je Visual Creator, za jehož pomoci lze definovat pohyb například jevištního vozu. V integraci s výše zmiňovaným Toolkitem dokáže obsluha namodelovat požadovanou trajektorii ve 3D. Na Obrázku 3.6 vidíme ovládací panel Acrobat G6 umožňující dvojí zobrazení a nástroj Skulptor Animation Toolkit, rovněž ve spojení s Visual Creatorem.

**Obrázek 3.6 UI výrobce Stage Technologies - vlevo ovládací panel Acrobat G6 a vpravo nástroj Sculptor Animation Toolkit [34]**

Dalším příkladem je *Creative Conners* [35], americký výrobce jevištní automatizační techniky působící téměř výhradně v USA. Jejich řešení řídicího systému stojí za zmínku, neboť disponuje velmi kompaktním uživatelským rozhraním. Proces řízení je distribuován na tzv. *Stagehands* – počítače obsluhující vždy právě jeden pohon. Stagehands jsou propojeny Ethernetem mezi sebou a rovněž s počítačem (jedním nebo více) na kterém je instalován *Spikemark* – obslužný software s GUI. Toto prostředí přímo integruje prostorové zobrazení scény s aplikací pro obsluhu pohonů. Při definování nového pohonu ve scéně je tento zobrazen jako nový 3D objekt v modelu scény a s využitím 3D editace lze specifikovat jeho rozměry a umístění v prostoru spolu s nastavením jeho dynamických vlastností jako je rychlost a dosah pohybu. Během představení je pak dění na scéně

znázorněno 3D animací příslušných objektů. Obsluha tedy vidí jak nabídku pohonů včetně jejich stavů a provozních informací, tak i jejich prostorové zobrazení, a to zároveň na jedné obrazovce (viz Obrázek 3.7). K provozu aplikace Spikemark je často užíván obyčejný notebook. Navíc však výrobce dodává i malý ovládací pult *Showstopper* s jednoduchým tlačítkovým (nikoli pákovým) ovládáním jízdy s pohony a červeným tlačítkem totalstop.




**Obrázek 3.7 Screenshot prostředí Spikemark amerického výrobce Creative Conners s integrovaným 3D zobrazením [35]**

Jistě lze najít celou řadu dalších příkladů, avšak jejich ráz je často principielně podobný. Mohu zmínit nizozemského výrobce *Trekwerk*, který pro 3D zobrazení využívá nástroje SkatchUp (viz kapitola 2.4), dále rovněž prostorové zobrazení scény nabízí německá *BBH* a mnozí další.

# 4 Návrh aplikace pro 3D vizualizaci divadelního jeviště

Tato kapitola vyhodnocuje současný stav diskutované problematiky. Nejprve shrnuje důležité aspekty uživatelského rozhraní divadelních řídicích systémů, analyzuje inovativní přístupy k interakci s obsluhou a postupně se dostává k motivaci pro vývoj aplikace pro 3D vizualizaci jeviště a popisuje požadavky na jeho funkcionalitu. Následuje shrnutí dostupných metod 3D modelování a motivace pro volbu nástroje. Zbytek kapitoly pak obsahuje vlastní návrh aplikace.

Ze třetí kapitoly, která shrnuje problematiku řízení jevištní techniky vyplývá, že jde o poměrně komplikovanou úlohu vyžadující adekvátní technologické řešení, zejména s ohledem na bezpečnost herců, rekvizit i zařízení samotného. Dalším kritériem hned po bezpečnosti je samozřejmě funkčnost, jež implikuje nutnost pohodlného a efektního uživatelského rozhraní. Dispozice systému by jistě měly obsluhujícímu strojníkovi umožnit obratně plnit požadavky režiséra, scénografů i jevištních techniků, přičemž obsluze by k tomuto mělo co nejvíce napomáhat optimálně navržené UI. Konkrétní požadavky na UI jsou podrobněji popsány v předešlé kapitole, stejně jako význam dalších technologií přinášejících vyšší sdělovací úroveň a míru interaktivity. Na uvedených praktických příkladech můžeme vidět, že velmi často využívanou technologií je 3D zobrazení scény, které bývá v rámci uživatelských rozhraní poměrně často implementováno jakožto doplněk samotného GUI. Motivace pro užití této technologie založená na poznatcích a zkušenostech z praxe je tedy evidentní.

Náplní této práce je především návrh a realizace 3D grafické vizualizační aplikace divadelního jeviště. Druhou motivaci pro tento záměr představuje firma DriveControl s.r.o., která vývoj aplikace pro 3D vizualizaci iniciuje a má na jejím praktickém využití zájem.

## 4.1 Požadavky na aplikaci

Navrhovaná aplikace pro 3D vizualizaci (dále také VA – vizualizační aplikace) má pracovat v přímé návaznosti na řídicí systém vyvíjený společností DriveControl a obslužnou aplikaci řídicího systému (dále také OA), přičemž typicky poběží na jiném počítači a s řídicím systémem bude komunikovat prostřednictvím komunikačního rozhraní postaveném na komunikačním protokolu ADS (specifikováno níže). VA by měla mít formu samostatné aplikace s integrovaným oknem pro 3D grafické zobrazení scény. Měla by obsahovat nástroje pro vložení nového modelu reprezentujícího reálný pohon, jeho přesné umístění ve scéně a nastavení jeho rozměrů včetně vymezení prostoru, do kterého zasahuje jízdní dráha pohonu. Nadefinovaný model pohonu tedy bude po jeho specifikaci nutné spárovat s odpovídajícím reálným pohonem, na jehož pohyb bude reagovat a vizualizovat jej. Informace o poloze, požadovaném směru a rychlosti jízdy bude aplikace přijímat prostřednictvím zmíněného komunikačního rozhraní.

Aplikace by měla umožnit vkládat do prostoru scény objekty reprezentující nejen samotné pohony, ale rovněž kulisy a rekvizity, čímž se vzniklá scéna více přiblíží aktuální reálné situaci na jevišti. VA bude na základě informací o současném uspořádání prostoru a požadavcích k jízdě přijímaných od obslužné aplikace řídicího systému schopna detekovat hrozící kolize a včasným varováním tak pomoci předejít havárii.

Požadavek na další funkcionalitu pramení z faktu, že na scéně se nachází velké množství pohonů a to jak v rámci horní tak i dolní mechanizace a právě zejména mezi pohony horní a dolní mechanizace může dojít ke vzájemné kolizi. Zvláště pak, budeme-li uvažovat i kulisy či rekvizity, které jsou na pohonech zavěšeny, respektive položeny. Například vysoká kulisa stěny domu, která je

postavena na podlaze jeviště složené z hydraulických stolů, navíc vyzdvižených do výše, pak zasahuje do značného prostoru jeviště a jistě může snadno zkřížit dráhu prospektovým tahům spouštěným shora. Vizualizační aplikace by tedy měla disponovat detektorem kolizí, přičemž tento by měl být koncipován coby sekundární redundantní detektor kolizí k detektoru primárnímu, který je již implementován v samotném řídicím systému. Tato duplicita slibuje zvýšení celkové spolehlivosti řídicího systému a zároveň je jednou z podmínek pro splnění předpisů certifikace *SIL3* (*Safety Integrity Level 3* – mezinárodně uznávaný standard pro bezpečnost nejenom v oblasti jevištní techniky [36]).

## 4.2 Implementační prostředí a nástroje

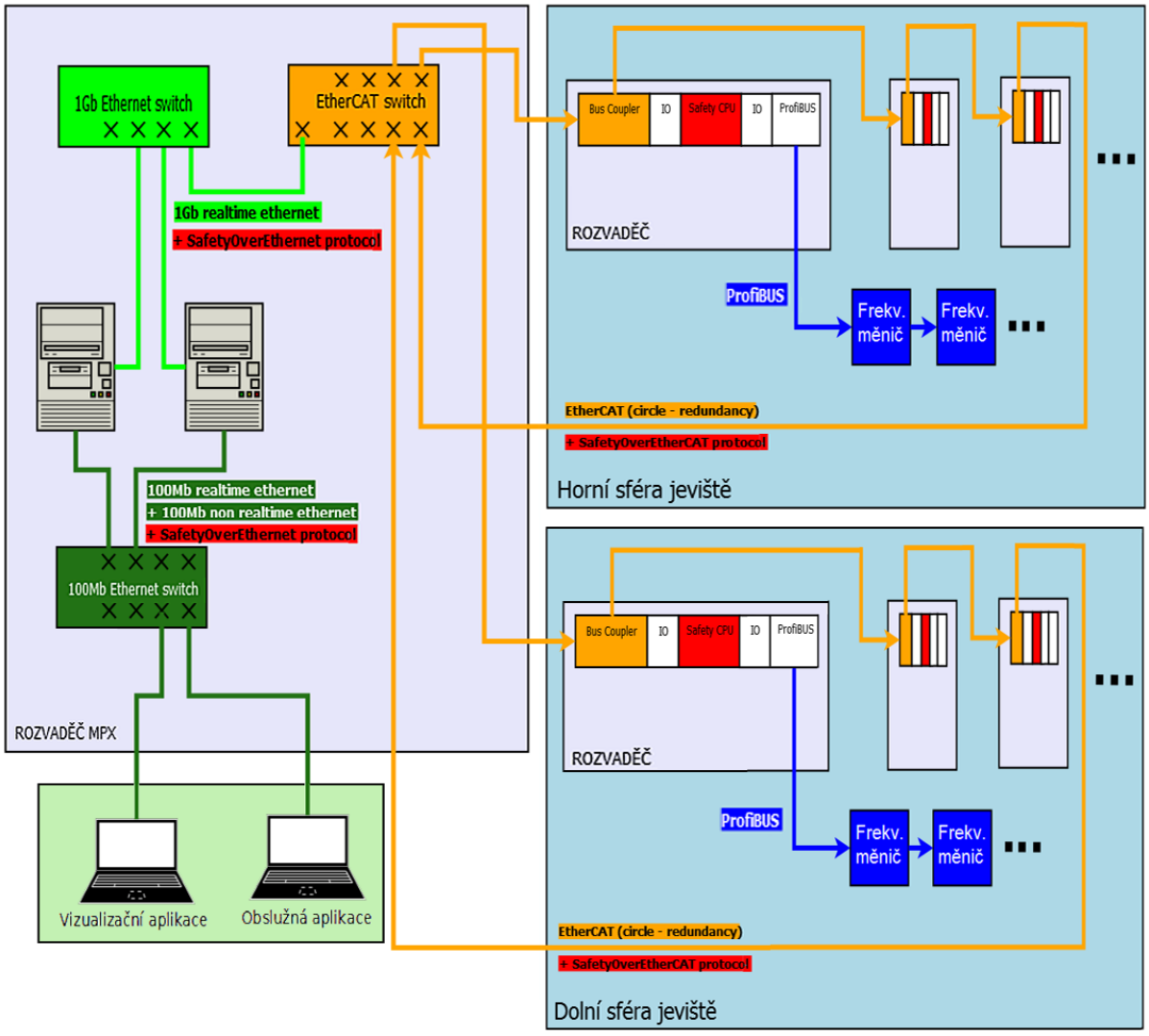
Nejprve je třeba zvolit vhodný nástroj, který by umožnil 3D zobrazení modelu jeviště a poskytl dostatečnou míru manipulovatelnosti s objekty ve scéně. Výsledná aplikace však musí obsahovat i grafické prostředí umožňující obsluze definovat nové pohony a editovat jejich vlastnosti. Nabízejí se tedy v zásadě dva možné přístupy, a to buďto zvolit některý 3D grafický editor či lépe CAD systém, který disponuje rozhraním pro komunikaci s externí aplikací a umožňuje jí manipulovat s objekty ve scéně. Onu externí aplikaci by pak již bylo možné vytvořit s využitím libovolné knihovny pro implementaci GUI v programovacím jazyce takovém, který by svými prostředky umožňoval navázat spojení s komunikačním rozhraním daného CAD systému. Druhou možností je využití některého, nejlépe vysokoúrovňového API pro práci s 3D grafikou a celé zobrazení prostorové scény tak integrovat přímo s GUI prostředím vizualizační aplikace.

Při volbě proti sobě stojí dva kandidáti, z nichž každý zastupuje jeden ze zmíněných přístupů (oba viz také 2.4). Lexocad je jednoduchý, avšak plně postačující CAD systém disponující v Pythonu implementovaným skriptem, jež slouží jako komunikační rozhraní s externí aplikací. Proti němu nástroj Eyeshot coby vysokoúrovňové API napsané pro jazyk C# (nebo alternativně Visual Basic). Pro účely této práce si jsou oba nástroje co do využitelnosti prakticky rovnocenné, neboť aplikace poběží na samostatném počítači a ani v jednom případě by neměl být problém pokrýt požadavky, které jsou na aplikaci kladené. Pro zadávající firmu je však schůdnější DevDept Eyeshot a tak z toho důvodu jsem se rozhodl využít tohoto nástroje. Implementačním jazykem je tedy C#, který umožňuje realizaci GUI například v prostředí MS Visual Studio.

## 4.3 Návrh aplikace a její začlenění do řídicího systému

Řídicí systém, ke kterému bude aplikace připojena, je postaven na PLC s využitím systému *TwinCAT* od německé společnosti *Beckhoff* [37]. Schéma celého systému přibližuje Obrázek 4.1. Pohony na jevišti jsou primárně děleny na dva zcela oddělené okruhy – na horní a dolní sféru mechanizace. Každý z těchto okruhů sestává z rozvaděčů, zapojených redundantně v kruhu sběrnicí *EtherCAT*. Každý z rozvaděčů disponuje PLC se vstupně-výstupními moduly a přes sběrnici *ProfiBUS* jsou k němu připojeny frekvenční měniče jednotlivých pohonů. Samotný TwinCAT řídicí systém běží na redundantně zdvojeném centralizovaném řídicím počítači, s nímž jsou za pomoci *real-time Ethernetu* propojeny okruhy rozvaděčů. Pomocí Ethernetu jsou připojeny i ovládací pulty (případně přenosné počítače), na nichž běží jednak obslužná aplikace a jednak 3D vizualizační aplikace. Komunikace na aplikační vrstvě probíhá podle protokolu ADS běžícím nad TCP/IP. Real-time Ethernet komunikace ovládacího pultu slouží k přenosu signálů od hardwarových tlačítek

a ovládacích pák pultu. Přenos signálu od bezpečnostního tlačítka Totalstop je zajištěn *Safety over Ethernet* protokolem.




**Obrázek 4.1 Schéma zapojení celého řídicího systému a začlenění Vizualizační aplikace**

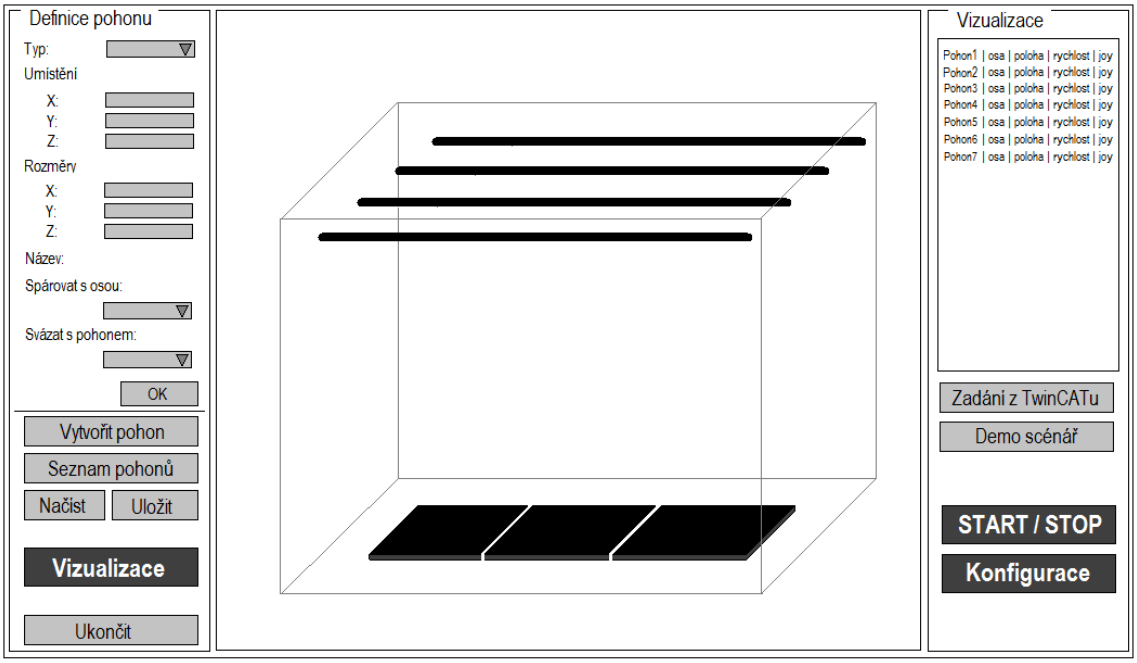
Komunikační rozhraní (jak pro obslužnou aplikaci, tak pro aplikaci vizualizační) má formu sady DLL knihoven, které implementují aplikační komunikační protokol ADS. VA prostřednictvím nástrojů příslušné DLL knihovny naváže spojení s řídicím systémem a bude moci přijímat zadání k vizualizaci, tedy datové struktury obsahující aktuální pozice, rychlosti a další data o dostupných pohonech.

Vizualizační aplikace bude pracovat ve dvou režimech, a to v režimu konfiguračním a režimu vizualizace. Konfigurační režim bude určen pro výstavbu modelu scény – vkládání nových pohonů a břemen a jejich editaci nebo pro definici rozměrů jeviště. Druhý režim pak pro vizualizaci jízdy pohonů, tedy grafické zobrazování jejich aktuální polohy v prostoru.

Po spuštění bude aplikace uvedena do konfiguračního režimu. Prvním krokem bude založení nového modelu, případně načtení některého dříve vytvořeného ze souboru. Následně bude možné konfigurovat scénu. Každý vkládaný pohon bude specifikován svými rozměry, umístěním ve scéně a zvoleným názvem. Dále bude umožněno jej spárovat s reálnou osou z nabídnutého seznamu všech dostupných os, připojených k řídicímu systému. Tento seznam si aplikace bude pravidelně aktualizovat na základě dat, která budou přijímána od řídicího systému prostřednictvím

komunikačního rozhraní. Dále bude umožněno tzv. svazování pohonů pro případy, kdy bude zapotřebí vsadit například zdvižný stůl do točny. V takovém případě bude vždy jeden z pohonů závislý na druhém – rodičovském pohonu. Zde tedy, pokud bude točna uvedena do pohybu, s ní svázaný zdvižný stůl se vůči točně bude chovat jako břemeno a „otáčet se s ní“.

Kdykoli v režimu konfigurace bude možné model scény uložit do souboru na lokální disk. Jako formát souboru jsem zvolil XML, jež se pro tyto účely jeví jako velmi vhodný a rovněž C# .NET disponuje kvalitní implementací vstupně-výstupních operací s XML soubory.




**Obrázek 4.2 Návrh GUI pro vizualizační aplikaci**

Následně bude umožněno aplikaci přepnout do režimu vizualizace. V závislosti na definovaných vazbách s reálnými pohony, budou modelu v pravidelných časových intervalech doručována nová zadání obsahující aktuální pozici a rychlost pohybu pro každý spárovaný pohon. Po doručení zadání, bude provedena vizualizace – dojde k aktualizaci parametrů jednotlivých pohonů ve scéně a překreslení grafických entit, kterými jsou reprezentovány. Informace o aktuální poloze, rychlosti, joysticku ovládacího panelu, pomocí kterého bylo zadání pro daný pohon vygenerováno a rovněž i o přiřazené reálné ose bude zároveň zobrazována i vhodnou textovou formou.

Konkrétná návrh GUI pro vizualizační aplikace naznačuje Obrázek 4.2. Panel v levé části okna bude určen pro režim konfigurace a pravý panel pro režim vizualizace, přičemž typicky bude uživateli zobrazen vždy jenom jeden z nich, podle režimu. V režimu konfigurace bude možné zobrazit seznam již vytvořených pohonů, který by měl zobrazovat i název pohonu a jemu přiřazenou osu. Seznam pohonů na panelu vizualizace obsahuje zobrazení textových informací, doplňujících grafickou vizualizaci. Ta je umístěna v dominujícím středovém prostoru okna. Samotné grafické zobrazení bude sestávat z prostorových grafických entit, reprezentujících jednotlivé pohony. Prostor jeviště bude schematicky vymezen úsečkami.

V režimu vizualizace bude v pohotovosti detektor kolizí. Pokud jízda některého z pohonů bude směřovat ke kolizi s jiným objektem ve scéně, detektor kolizí hrozící havárii rozpozná, specifikuje a vhodným způsobem upozorní obsluhu. Detektor kolizí bude volán i v režimu konfigurace, aby při vkládání pohonu nedošlo ke kolizi s jiným.

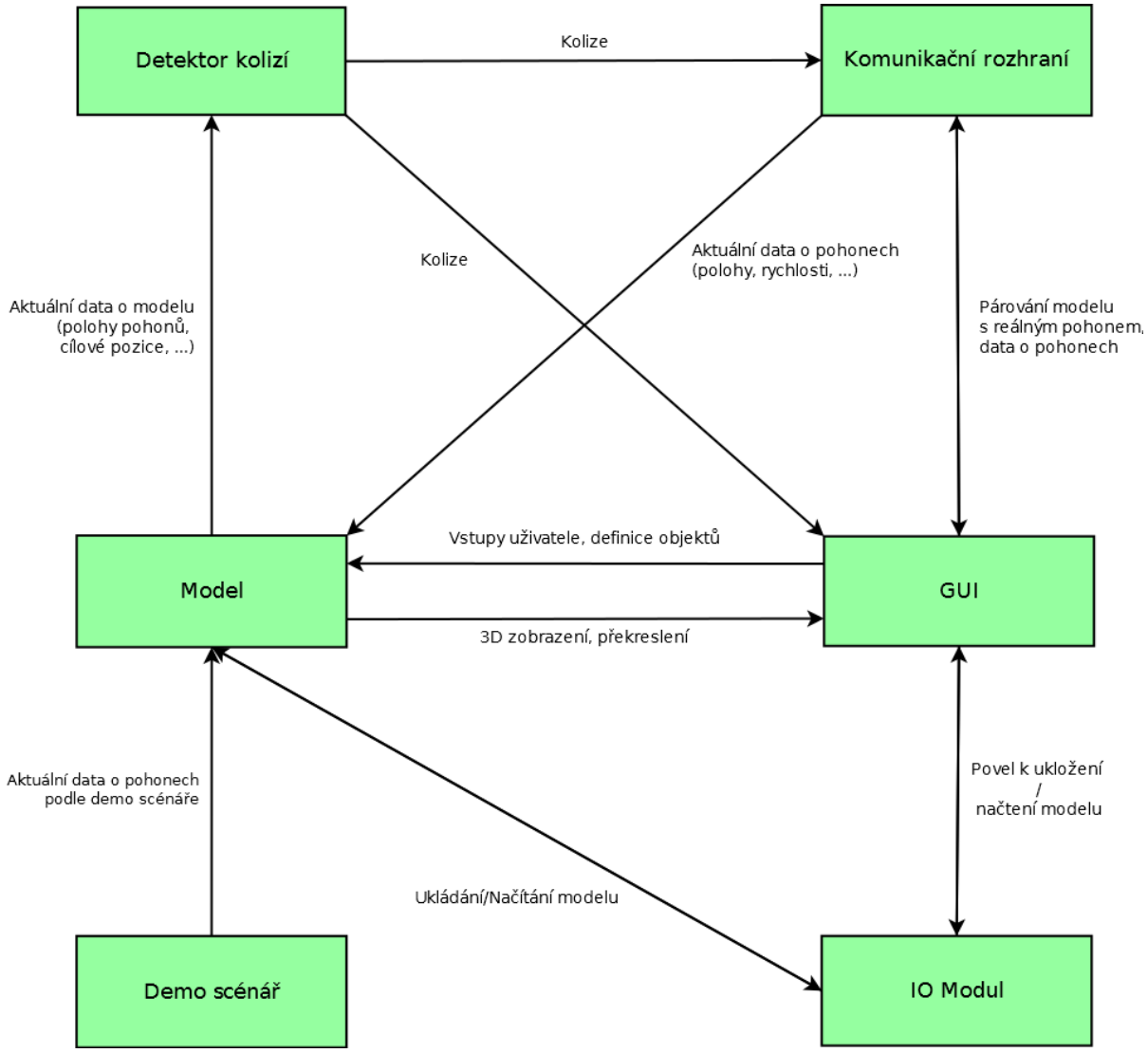
Navržené řešení by mělo splňovat požadavky zadání. Jeho výhodou je způsobilost pro pohodlné vkládání a editaci pohonů s přesnou definicí jejich rozměrů a umístění, včetně přehledného grafického zobrazení scény. Nevýhodou by pak mohlo být pouze jednoduché textové zobrazení přesných aktuálních dat o poloze a rychlosti pohonů, avšak i toto řešení plní svůj účel upřesňujícího doplňkového zobrazení ke grafické vizualizaci.

# 5 Realizace

Tato kapitola popisuje realizaci aplikace pro 3D vizualizaci divadelního jeviště, jejíž návrh specifikuje předcházející kapitola a rozebírá zvolené implementační postupy. Své řešení jsem implementoval v prostředí MS Visual Studio 2012, v programovacím jazyce C#, jako aplikaci určenou pro běh pod operačním systémem MS Windows. Nejprve je specifikována vnitřní struktura aplikace – přehled jejích stěžejních komponent a další podkapitoly pak postihují detailnější popis jednotlivých součástí a tříd, kterými jsou implementovány.

## 5.1 Popis a vnitřní struktura aplikace

Aplikace sestává z několika vnitřních komponent. V samotném centru dění stojí model, udržující si pojem o objektech ve scéně. Jsou v něm zapouzdřeny jednak instance reprezentujících grafických entit, ale i další abstraktní data o objektech. Model je vsazen do grafického uživatelského rozhraní aplikace, které přijímá vstupy uživatele a zobrazuje výstup, typicky v podobě 3D zobrazení grafických entit reprezentujících model.

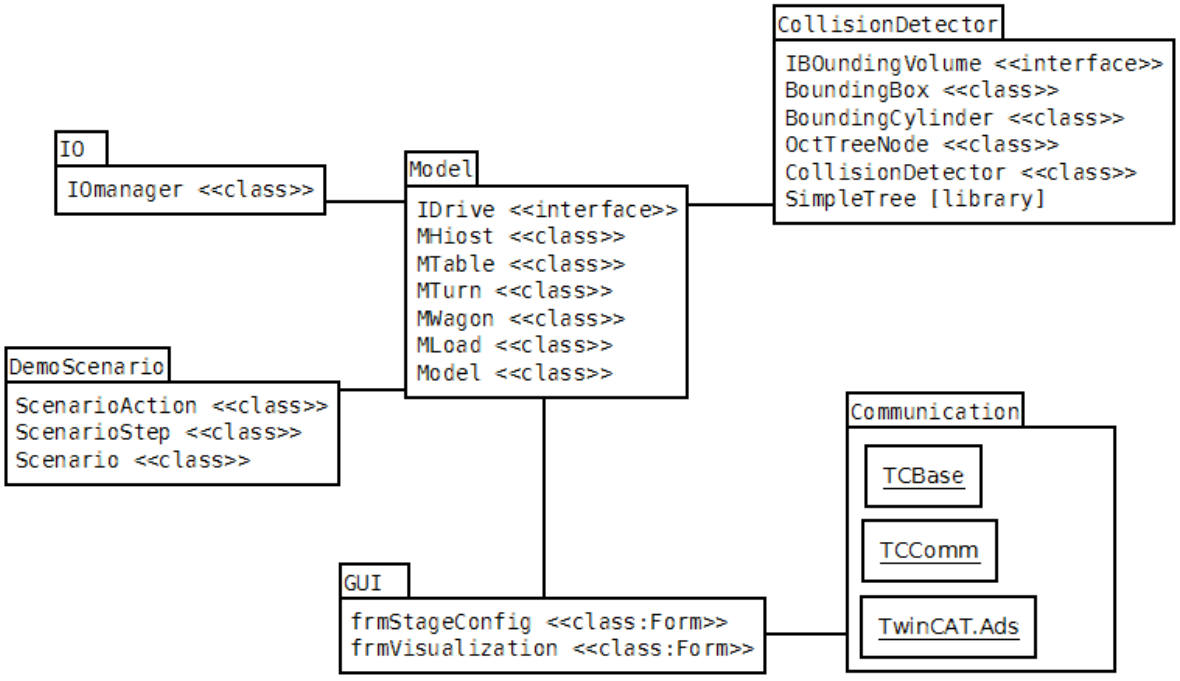



**Obrázek 5.1 Schéma vnitřní struktury vizualizační aplikace**

Obě tyto komponenty úzce spolupracují se třetí komponentou, komunikačním rozhraním, které zajišťuje příchozí a odchozí komunikaci s řídicím systémem. Zadání pro vizualizaci je možné přijímat buďto od řídicího systému anebo z demo scénáře, což je komponenta generující jednoduchý scénář, jež by měl sloužit především pro demonstrační účely. Za další samostatnou součást lze označit modul pro vstupní a výstupní operace, tedy ukládání modelu do souboru a jeho opětovné načítání. Poslední komponentou je pak detektor kolizí. Ten je úzce provázán s modelem – reaguje na každou změnu kteréhokoli objektu ve scéně a prověřuje míru potenciálního nebezpečí kolize s kterýmkoli jiným objektem v prostoru jeviště. Rovněž ošetřuje potenciální kolizní situace které mohou nastat během vkládání nebo editace pohonů.

Aplikace ve svém grafickém prostředí zobrazuje seznam dostupných reálných pohonů. Dále obsahuje ovládací prvky pro definici a specifikaci modelů objektů ve scéně. Pomocí vhodných nástrojů je pak možné každý nadefinovaný model pohonu spárovat s reálným pohonem z nabídky, případně definovat a k pohonu přiřadit další objekt, typicky model nějakého břemene. GUI pak dominuje prostor, ve kterém je 3D model zobrazován. Během vizualizace je informace o aktuální poloze a rychlosti pohonů pro úplnost zobrazována i formou textu. Aplikace rovněž obsahuje nástroje pro nastavení volitelných parametrů, např. pro detektor kolizí (viz níže).

Obrázek 5.1 zobrazuje vnitřní strukturu aplikace, sestávající ze šesti základních komponent a prezentuje vzájemné relace mezi nimi. Tomuto přibližně odpovídá skladba programu. Tedy každá ze zmíněných komponent je implementována skupinou tříd, pro přehlednost organizovaných do oddělených balíků (složek) tak, jak specifikuje diagram na Obrázku 5.2. Jakousi výjimku tvoří komponenta komunikačního rozhraní, jež je reprezentována DLL knihovnami TCBase, TCCom, a TwinCAT.Ads, dodanými firmou DriveControl. Podrobný UML diagram tříd je k dispozici pro nahlédnutí v Příloze A. V následujících třech podkapitolách blíže popisuji důležité aspekty implementace včetně uživatelského rozhraní.



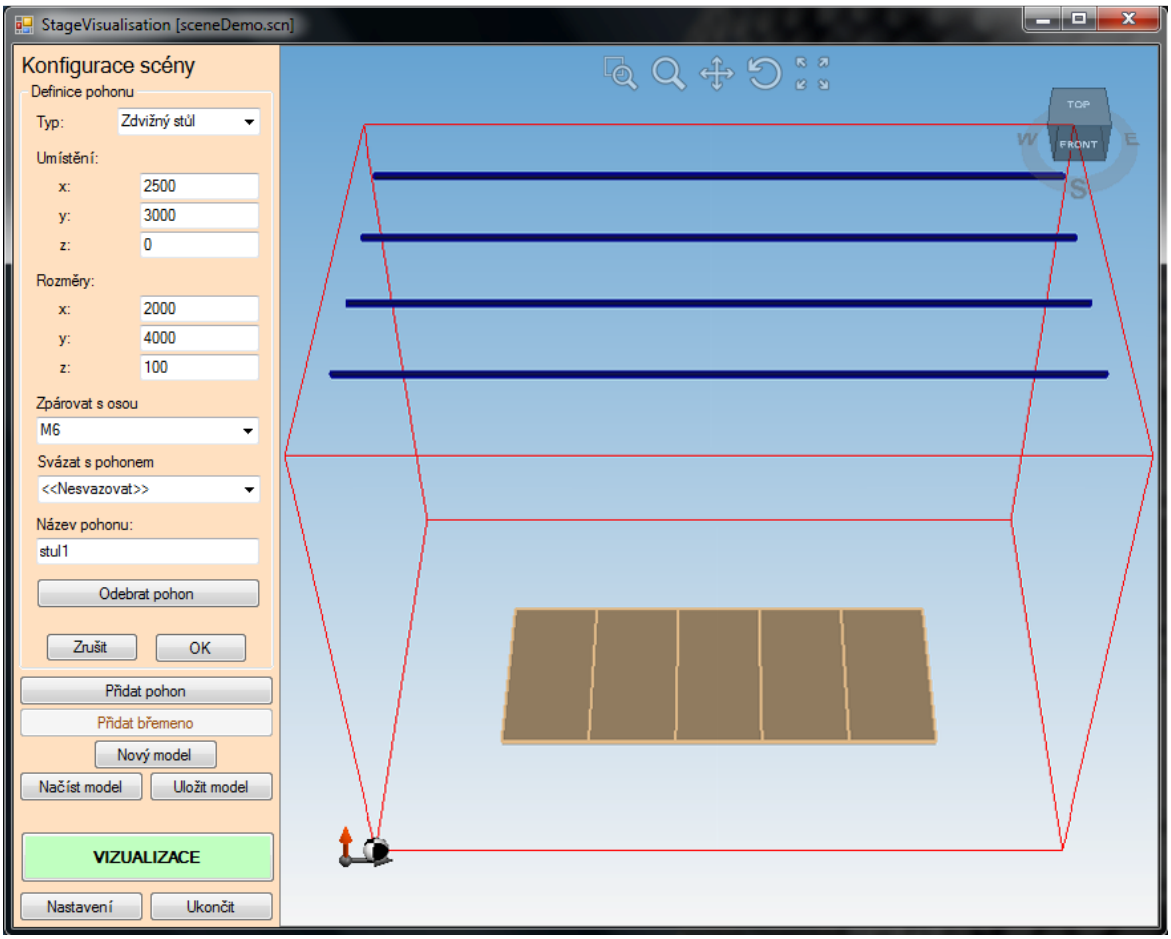

**Obrázek 5.2 Struktura dílčích komponent aplikace a tříd, resp. komunikačních DLL knihoven, které je implementují**

## 5.2 Uživatelské prostředí

Uživatelské prostředí vizualizační aplikace má formu GUI. Téměř veškerá interakce mezi uživatelem a aplikací se odehrává prostřednictvím hlavního okna *3D vizualizace jeviště*. Další pomocné okno *Nastavení jeviště* slouží pro definici rozměrů jeviště a též umožňuje otevřít model ze souboru. Po spuštění aplikace je zobrazeno zmíněné pomocné okno, aby uživatel mohl zahájit obsluhu buďto vytvořením nové scény anebo načtením připraveného modelu. Po tomto prvním kroku je otevřeno hlavní okno.

Běh aplikace se dělí na režim konfigurace a režim vizualizace. Po spuštění je aplikace uvedena do režimu konfigurace, který je určen pro tvorbu a editaci modelu scény. Oknu dominuje prostor komponenty pro 3D zobrazení a po jeho levé straně má své místo panel *Konfigurace scény* (viz Obrázek 5.3). Jeho prostřednictvím je umožněno načtení modelu ze souboru nebo uložení rozpracované scény. Opětovné nastavení rozměrů jeviště je možné, avšak pouze přes tlačítko *Nový model* a znamená tedy vytvoření nového prázdného modelu, kdy model stávající bude zahozen – uživatel je na tuto skutečnost upozorněn dialogovým oknem.

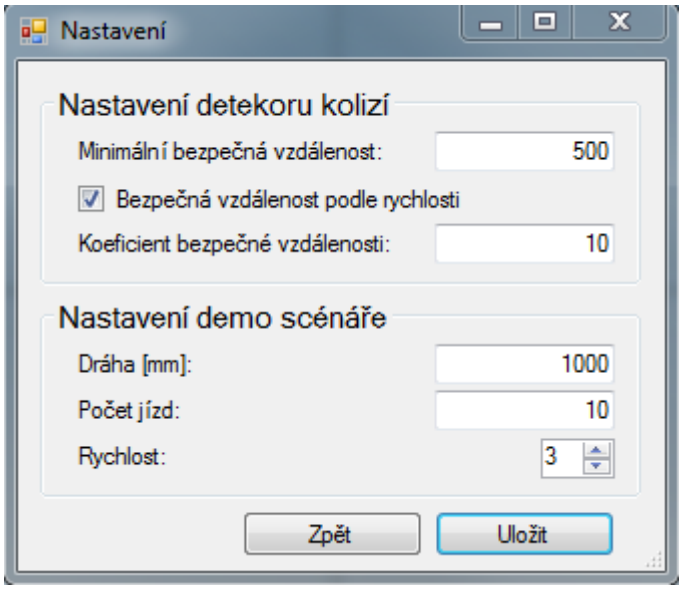
Formulář pro definici pohonu obsahuje potřebné nástroje k zadání všech údajů – název a typ pohonu, jeho rozměry a umístění v prostoru. Je zde rovněž nabídka dostupných reálných os, se kterými je možné pohon spárovat. Poslední z ComboBoxů pak slouží k výběru nadřazeného pohonu, se kterým má být pohon svázán. Panel konfigurace s formulářem definice pohonu je zobrazen na Obrázku 5.3 vlevo.




**Obrázek 5.3 Vlevo panel Konfigurace scény, vpravo celé hlavní okno s panelem Vizualizace**

Po vložení nového pohonu je na místě formuláře v horní oblasti konfiguračního panelu zobrazen seznam vytvořených pohonů. Seznam má stromovou strukturu pro přehledné zobrazení břemen svázaných s pohony (u každého pohonu je v této fázi pouze pro ilustraci implicitně zobrazeno jedno břemeno). Za každým pohonem je navíc v závorce zobrazen název osy, se kterou je spárován (je-li s některou spárován).

Když je scéna nadefinována, příslušným tlačítkem lze aplikaci přepnout do režimu vizualizace. Namísto konfiguračního panelu je zobrazen panel Vizualizace se seznamem pohonů v horní části. Pod ním jsou umístěna tlačítka pro přepínání mezi zadáním od řídicho systému nebo podle demo scénáře a pro spuštění nebo zastavení vizualizace. Po odstartování vizualizace zmíněným tlačítkem se spustí časovač `tmrVisRun` a je zahájena vizualizace podle zadání ze zvoleného zdroje (viz níže). Zobrazovací komponenta SingleViewportLayout zajišťuje grafické zobrazení vizualizace. Výhodou je, že již implicitně disponuje standardní sadou nástrojů pro manipulaci s pohledem: rotaci, změnu měřítka, posun a zarovnání a vycentrování měřítka na celý pohled. V seznamu pohonů na panelu Vizualizace jsou pak textově zobrazena přesná data o aktuální poloze a rychlosti vizualizovaných pohonů, včetně reálných os, se kterými jsou spárovány a označení joysticku, na který je pohon přiřazen na ovládacím panelu (resp. v GUI obslužné aplikace – viz Obrázek 5.6). Náhled GUI v režimu vizualizace je vidět např. na Obrázku 5.9. Po přepnutí zpět do režimu konfigurace, je vizualizace zastavena a modely resetovány do nulových poloh.




**Obrázek 5.4 Nastavení uživatelských parametrů**

Aplikace umožňuje uživatelsky parametrizovat svojí činnost. Pro detektor kolizí je možné definovat výpočet minimální bezpečné vzdálenosti, po jejíž překročení je během vizualizace detekována kolize. Dále je možné parametrizovat generovaný demo scénář: lze zvolit délku dráhy, kterou budou pohony během jízdy vykonávat, počet jízd a rychlost.

Parametry jako ADS adresa pro navázání komunikace s ŘS, počet dostupných reálných os a nebo požadovanou hloubku octree stromu detektoru kolizí není možné editovat za běhu aplikace. Tyto lze nastavit v souboru init.xml, umístěném v adresáři aplikace, ze kterého jsou data po startu načtena.

## 5.3 Model a vizualizace

Pro 3D grafické zobrazení modelu scény jsem použil prostředky nástroje Devdept Eyeshot. Pro 3D zobrazení slouží GUI komponenta SingleViewportLayout, která realizuje zobrazení s jedním pohledem (Eyeshot podporuje i zobrazení se třemi nebo čtyřmi pohledy) a zahrnuje pohodlné nástroje pro základní manipulaci s pohledem.

Grafické znázornění jeviště jsem zvolil pouze schematicky a jednoduše formou přímek. Pro reprezentaci pohonů se je velmi vhodná entita typu Mesh, která představuje jakýsi obecný prostorový objekt. Díky jejímu polymorfismu je možné z ní vytvořit více typů prostorových primitiv. Pro reprezentaci tahů jsem zvolil formu horizontálně orientovaného válce souměrného podle osy *x* souřadného systému. Tahy jsou umísťovány implicitně u stropu jeviště. Zdvižné stoly jsou reprezentovány nízkými kvádry umístěných při podlaze a konečně točna je tvaru vertikálně orientovaného válce vsazeného do podlahy jeviště.

Avšak reprezentace modelu nespočívá pouze v reprezentaci grafické. Celou jeho implementaci popisuje následující text, spolu s popisem principů vizualizace a komponent s modelem a vizualizací úzce spjatých, konkrétně demo scénáře a operací vstupu a výstupu.

### *Model*

Model jako celek sestává z modelů jednotlivých objektů ve scéně. Nese jak instance grafických entit, které je reprezentují, tak další abstraktní data. Každý z možných objektů má svá specifika. Existují 4 typy pohonů – tah (implementován jako tah prospektový), zdvižný stůl, točna a jevištní vozík. Jednotlivé typy se liší typickou trajektorií a vůbec principem pohybu, tvarem i svým implicitním umístěním ve scéně. Nelze tak definovat univerzální třídu, která by reprezentovala všechny typy pohonů. Implementoval jsem tedy rozhraní `IDrive`, jež reprezentuje obecný pohon. Obsahuje deklarace metod pro potřebné operace s pohony, implementovaných již konkrétní třídou reprezentující daný typ pohonu.

Dalšími objekty, které se mohou ve scéně objevit jsou břemena. Břemeno je uvažováno jako objekt, který je svázán s některým pohonem, přičemž se nerozlišuje více typů břemen. Břemeno je reprezentováno svojí vlastní třídou.

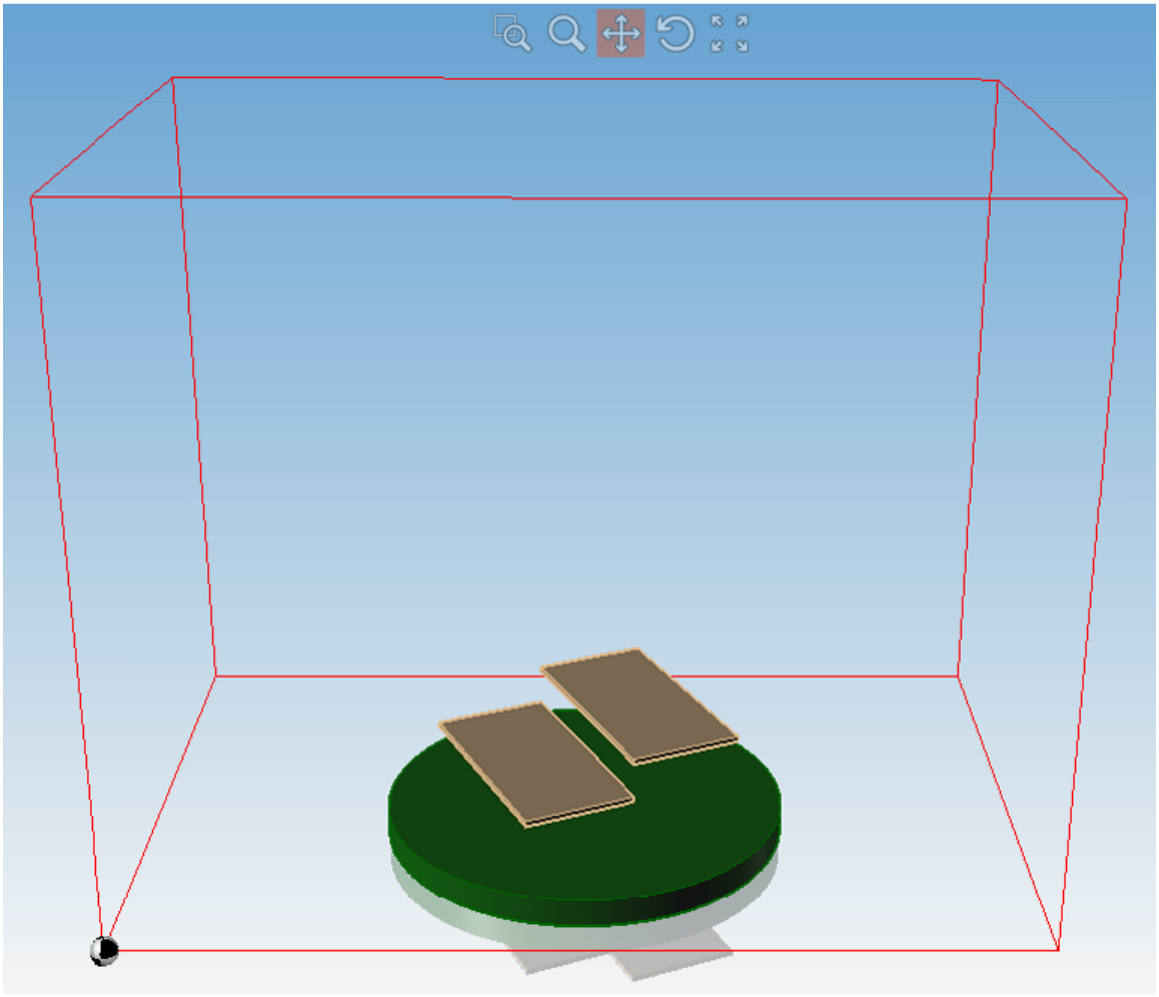
Model jako celek je implementován třídou `Model`, které je předána instance vykreslovací komponenty `SingleViewportLayout` (s již nadefinovanými vrstvami pro jeviště, břemena a různé typy pohonů), aby z této i ostatních dílčích tříd modelu bylo možné přímo vkládat a editovat grafické entity. Grafická reprezentace jeviště je pojata jako co nejjednodušší a má formu 12ti přímek vymezujících jeho prostor ve tvaru kvádru. Metoda `stageRedraw` má pak na starosti jeho překreslení v příslušné vrstvě podle aktuálně nastavených rozměrů (reprezentace modelu jeviště viz Obrázek 5.5).

Třída `Model` je implementovaná jako seznam všech pohonů ve scéně, tedy jako seznam nad obecným typem pohonu a rovněž si udržuje i seznam všech existujících břemen (id břemene a nadřazeného pohonu) a udržuje si i instanci detektoru kolizí. Vložení nového pohonu spočívá ve vytvoření instance příslušné třídy, dle typu pohonu a její připojení k seznamu. Zároveň dojde i ke vložení odkazu na pohon do detektoru kolizí se současnou kontrolou na případné kolize během vkládání – je-li detekována kolize, vytvořený pohon bude zahozen a metoda vrátí odpovídající chybový kód. Vytvoření reprezentující grafické entity a její umístění v prostoru je již v režii metody `driveCreate` příslušného objektu daného pohonu.

Při editaci pohonu je pak vždy vytvářena jeho nová instance podle nových parametrů a odkaz v detektoru kolizí je odebrán a nahrazen odkazem na novou instanci (tedy včetně znovu provedení testu na kolize v novém umístění).

Zmíněná metoda `driveCreate` pomocí entity `Mesh` vytvoří prostorové primitivum odpovídající typu pohonu, zajistí jeho správnou orientaci a přesun na požadované umístění v prostoru a vloží jej mezi entity ve vykreslovací komponentě `SingleViewportLayout`.

Základní prostorové transformace jako rotace, změna měřítka nebo translace jsou v Eyeshot již implementovány, a tak stačí například zavolat metodu Rotate (Double angle, Vector3D axis) nebo Translate(Vector3D vector) entity Mesh a potřebné maticové operace jsou provedeny na pozadí (pochopitelně je nutné dodržet správné pořadí při volání transformací). V případě potřeby sofistikovanějších transformací Eyeshot umožňuje i aplikaci transformací podle volitelně definované matice. K tomu slouží metoda `TransformBy(Transformation xform)`, kde `Transformation` je typ pro transformační matici velikosti 4x4.



**Obrázek 5.5 Model jeviště s točnou a do ní vsazenými dvěma zdvižnými stoly**

V modelu je implementovaná i funkcionalita svazování pohonů, která bývá v praxi, zvláště ve větších divadlech, často zapotřebí. Jedná se o případ, kdy je např. zdvižný stůl vsazen do točny a je tedy potřeba, aby se otáčel s ní (viz Obrázek 5.5). Pokud má být pohon svázán s jiným, je toto třeba zahrnout do procesu jeho definice. Objekt každého pohonu má poznačen odkaz na svůj nadřazený pohon (není-li svázán s žádným pohonem, je hodnota proměnné nastavena na -1). Zároveň si uchovává seznam všech svých podřízených pohonů. Má-li být tedy vkládaný pohon *A* svázán s pohonem *B*, je u pohonu *A* nastaven odkaz na nadřazený pohon jako ID pohonu *B* a do seznamu podřízených pohonů pohonu *B* je vložen celý objekt pohonu *A*. Vložení kopie celého objektu je nutné z důvodu vizualizace svázaných pohonů (viz níže).

### *Vstup a výstup*

Aplikace umožňuje ukládání modelu do souboru a jeho opětovné načítání. Proces načítání modelu ze souboru znamená, že na základě dat přečtených ze souboru bude probíhat vkládání pohonů do scény. Logicky nejjednodušší přístup je toto provádět stejným způsobem jako při vkládání pohonu nadefinovaného v GUI, tedy cyklické volání té samé metody (`driveAdd`). Pro každý pohon je tedy zapotřebí ukládat shodnou množinu parametrů, jako je zadávána při definici pohonu pomocí GUI: typ pohonu, rozměry, umístění, název, ID případného rodičovského pohonu a ID *osy* (tedy reálného pohonu, připojeného k ŘS), se kterou je pohon spárován (viz níže). Tuto množinu je potřeba navíc sjednotit s parametry, které jsou generovány automaticky a s parametry, které jsou proměnlivé během vizualizace, tedy s ID pohonu a aktuální pozicí. Kromě dat o pohonech jsou do souboru ukládány i rozměry jeviště.

Jedná se tedy o značné množství povětšinou číselných dat, které evidentně vykazují potenciál pro hierarchické strukturování souboru. Pro ukládání takovýchto strukturovaných dat jsem poměrně rozhodně zvolil formát XML. Jazyk C# navíc pro manipulaci s XML soubory poskytuje kvalitní nástroje.

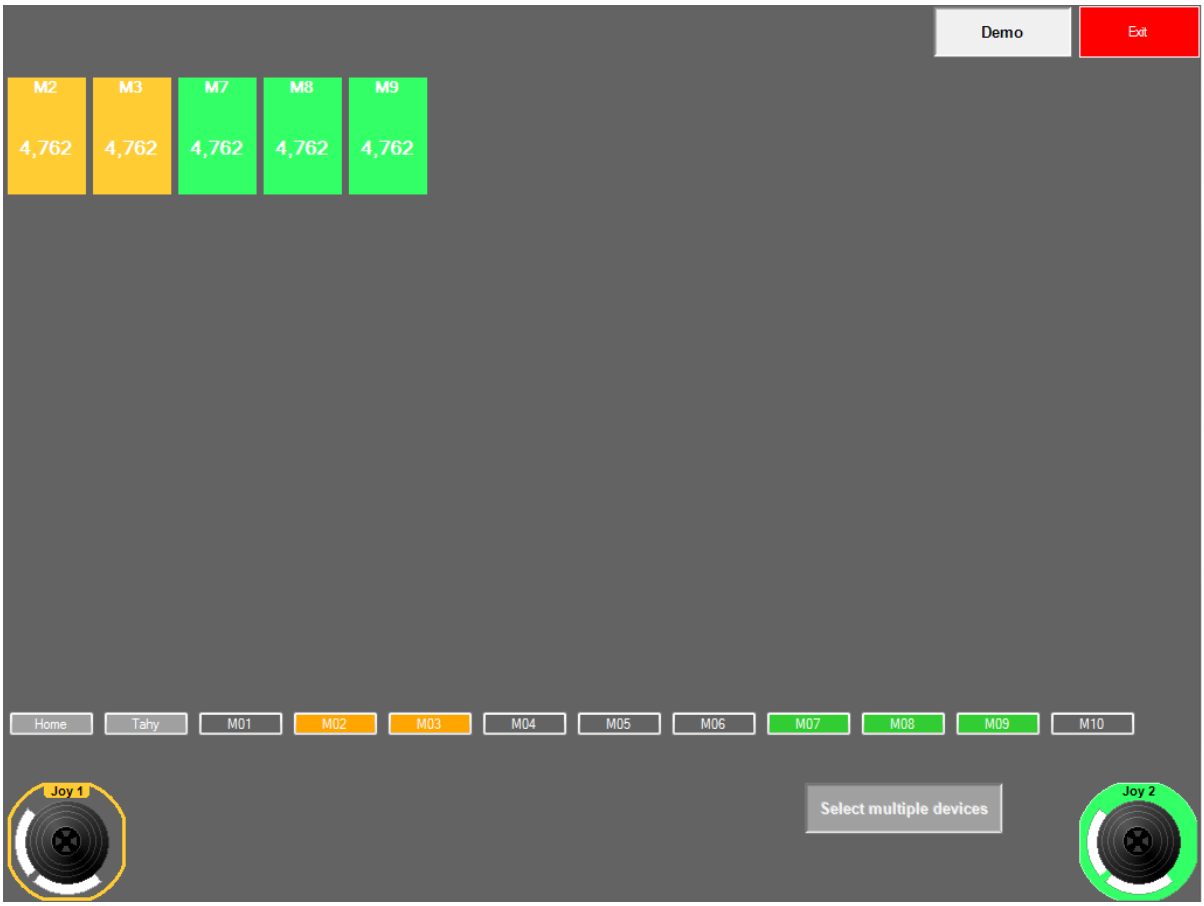
Metoda `modelLoad` třídy `IOmanager` po načtení souboru vytváří novou instanci třídy `Model` a jak jsem již předeslal v odstavci výše, cyklicky volá metodu `driveAdd` a postupně vloží všechny načtené modely do seznamu. Standardním způsobem jsou rovněž vkládány odkazy na pohony do detektoru kolizí a jsou vytvořena provázání mezi pohony, které jsou vzájemně svázané. Provádí se zde i kontrola rozměrů a umístění pohonů vzhledem k jevišti, stejná jako při ošetření GUI formuláře pro definici nového pohonu pro případ, kdyby byl soubor nepatřičným způsobem neuváženě editován (i detekce kolizí je během vkládání pohonů do Detektoru kolizí opět implicitně prováděna). Do souboru je ukládána i aktuální pozice pohonů, takže model je po načtení uveden do své původní stavu.

### *Vizualizace*

Je-li nastaven příjem zadání pro vizualizaci z TwinCATu, je zadání generováno řídicím systémem, typicky prostřednictvím ovládacího pultu nebo obslužné aplikace (viz Obrázek 5.6). Jeho příjem má na starosti objekt třídy TcLink komunikačního rozhraní, který udržuje spojení s řídicím systémem. Je mu předána rutinní obslužná metoda `link1_OnStartSending`, kde cyklicky přijímá aktuální zadání od řídicího systému a předává jej do seznamu zadání, který tak obsahuje stále aktuální souhrnné zadání pro vizualizaci všech dostupných os.

Tohoto seznamu je využíváno již v režimu konfigurace, coby seznamu dostupných reálných os. Ve formuláři GUI pro definici nového pohonu je na základě tohoto seznamu naplněn příslušný ComboBox pro výběr reálné osy, se kterou má být nový pohon spárován.

Je-li pak aplikace přepnuta do režimu vizualizace, spustí se časovač `tmrVisRun` který v pravidelných intervalech voláním metody `axesTaskExecute` iniciuje update a překreslení modelu podle aktuálního zadání – řídí chod vizualizace. Zmíněná metoda hledá pohony, pro které je k dispozici zadání a pouze tyto pohony jsou poté aktualizovány a překresleny. Pohony jsou vyhledávány podle ID osy, se kterou jsou spárovány. Tedy pokud pohon není spárován s žádnou osou, není s ním více ztrácen čas. Odpovídajícím pohonům je aktualizována aktuální pozice, aktuální rychlost a informace, na kterém z joysticků ovládacího panelu (případně obslužné aplikace) je osa přiřazena. Samotné překreslení pohonu je však voláno s podmínkou, že pohon musí být podle posledního přijatého zadání v pohybu, tzn. aktuální rychlost je nenulová (jízda zpět je indikována zápornou hodnotou aktuální rychlosti).




**Obrázek 5.6 GUI obslužné aplikace řídicího systému zobrazující dostupné osy; na levou páku jsou přiřazeny dvě a na pravou tři osy**

Proces překreslení pohonu zajišťuje metoda `driveRedraw`, deklarovaná v rozhraní `IDrive`. Každá z tříd tuto metodu implementuje různě, v závislosti na typu pohonu. Obecně je společným faktorem výpočet offsetu, jako rozdílu aktuální a poslední zaznamenané polohy. Na základě tohoto offsetu je sestaven vektor posunu a volána transformace translace příslušné grafické entity v případě tahu nebo zdvižného stolu. V případě točny offset představuje úhel otočení kolem vertikální osy točny. Zde je třeba provést všechny tři dílčí transformace samostatně (pomoci metod zmíněných výše), tedy translace do počátku souřadného systému, rotace kolem osy z a translace zpět na původní pozici.

Překreslování svázaných pohonů je implementováno pouze pro třídy točny a zdvižného stolu, tedy pro patrně nejběžnější kombinaci využití této funkcionality. Překreslení podřízeného pohonu má na starosti metoda `driveFollowParent`. Pokud má pohon nějaké podřízené pohony, jsou tyto obsažené v příslušném seznamu podřízených pohonů, jež si pohon uchovává. Po překreslení nadřazeného pohonu je procházen tento seznam a pro každý z podřízených pohonů je volána metoda `driveFollowParent`, se stejnými parametry pro transformaci, podle kterých byla provedena transformace nadřazeného pohonu. Je-li tedy nadřazeným pohonem točna, je předán úhel otočení (offset), vektor posunutí do počátku souřadnic a osa rotace, na základě čehož je na `Mesh` podřízeného pohonu aplikována odpovídající transformace. Metoda `driveFollowParent` je přetížená – pokud je nadřazeným pohonem točna, je pro transformaci zapotřebí jiná skladba parametrů než v případě např. zdvižného stolu.

Během překreslování pohonu detektor kolizí rovněž aktualizuje data o pozici překreslovaného pohonu a zároveň otestuje, zda na základě vykonaného zadání nedošlo ke kolizi s jiným objektem ve scéně – pokud ano, uživatel je o této skutečnosti informován chybovým hlášením.

### *Demo scénář*

Demo scénář je slouží coby alternativa k zadání od řídicího systému pro demonstrační účely nebo jednoduše pro případy, kdy řídicí systém není dostupný. Je tedy koncipován jako komponenta, která podle daného algoritmu generuje zadání pro vizualizaci bez nutnosti komunikace s řídicím systémem. Zde je však nutné podotknout, že tato funkcionalita je považována pouze jako doplňková, případně pomocná (např. během dalšího vývoje aplikace) a z pohledu koncového uživatele aplikace ji není přikládán velký význam. Proto není umožněna vyšší míra nastavení či například výběr z více variant.

Implementovaný algoritmus generuje zadání pro jednoduchý periodický synchronizovaný pohyb připomínající „vlnobití“ kdy ve stejný okamžik všechny pohony v sudém pořadí jsou uvedeny do pohybu jedním směrem a ty v lichém pořadí směrem opačným. Po dosažení požadované polohy se rozjedou opačným směrem opět proti sobě. Cílová poloha a počet „vln“ (tedy počet změn směru) jsou nastavitelnými parametry. Zadání je generováno vždy pro všechny pohony ve scéně. Mezi vizualizací přijímající zadání generované demo scénářem a příchozím od řídicího systému je přepínáno příslušným tlačítkem GUI přímo v režimu vizualizace.

Generování scénáře zajišťuje metoda `generateDemo` ve třídě demo scénáře. Celý scénář se skládá z kroků (vždy jízda jedním směrem) a každý krok z dílčích akcí (elementárních úseků jízdy pohonu o délce závislé na zvolené rychlosti scénáře – délka tohoto úseku představuje krok mezi jednotlivými zadáními). V rámci zachování obecnosti a umožnění případného dalšího rozšíření na více variant demo scénáře jsem krok i dílčí akce scénáře implementoval jako samostatné třídy.

Vizualizace podle demo scénáře probíhá velmi podobně jako v případě zadání přijímaného od ŘS. Ovšem v tomto případě obslužnou rutinní metodu komunikačního rozhraní nahrazuje časovač `tmrVisAxisUpdate`, který v pravidelných intervalech aktualizuje seznam zadání pro jednotlivé osy – volá metodu `getAxisToClient` třídy demo scénáře, která vrací zadání (pro všechny osy) pro následující akci právě probíhajícího kroku scénáře. Je-li zadání v seznamu aktualizováno, pak už časovač, obdobně jako při zadání od ŘS, volá metodu `axesTaskExecute`, která zajistí překreslení modelu.

## 5.4 Detektor kolizí

Detektor kolizí nalézá dvě různá uplatnění. Jednak v režimu konfigurace, při vkládání nového pohonu do scény, kdy má za úkol odhalit, pokud by se nový pohon ocitl v kolizní situaci s jiným pohonem ve scéně. Ve svém druhém uplatnění, v režimu vizualizace, má pak za úkol odhalit případy, kdy situace vyplývající z přijatého zadání je situací kolizní nebo ke kolizní situaci směřuje.

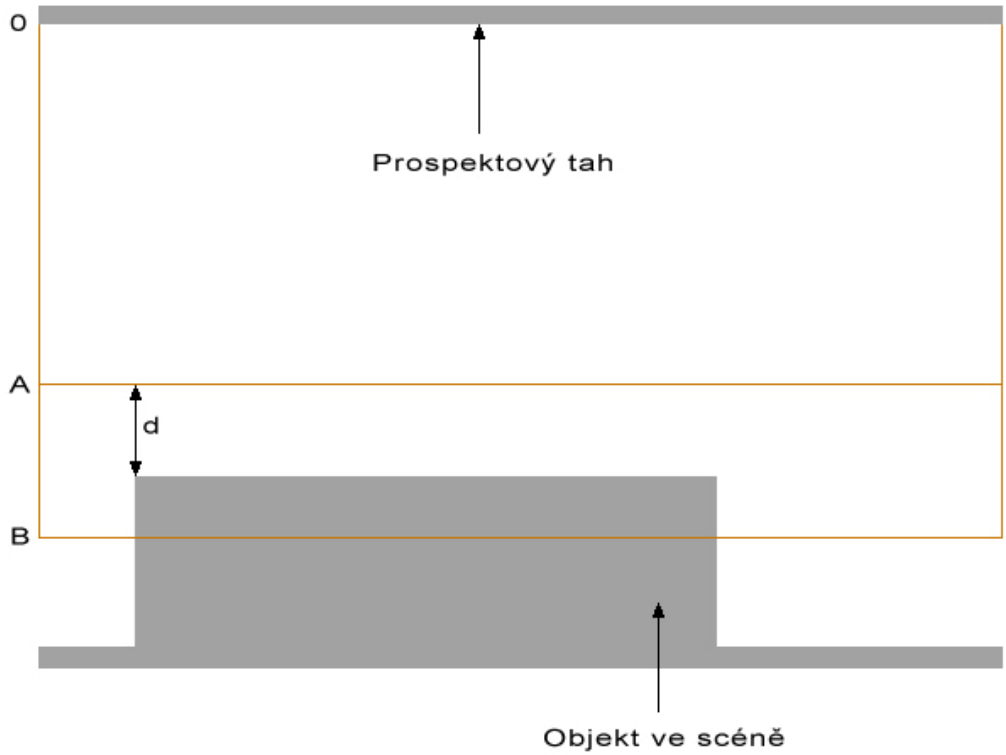
Pokud jde o detekci kolizí v režimu vizualizace, je tedy zapotřebí hrozící kolizi odhalit dříve, než k ní fyzicky dojde. Obrázek 5.7 schematicky znázorňuje možnou kolizní situaci. Nachází-li se prospektový tah v poloze *0* a zadání vizualizace udává novou aktuální polohu v pozici B, tedy v pozici přímo zasahující do jiného objektu ve scéně, bude detekována kolizní situace. Pokud příchozí zadání udává novou aktuální polohu tahu v pozici *A*, tedy v pozici blízké jinému objektu ve scéně, bude detekována hrozící kolizní situace v takovém případě, kdy pro vzdálenost *d* mezi požadovanou aktuální polohou *A* a jiným objektem ve scéně platí:

$$d \leq D, \tag{10}$$

kde *D* je povolená bezpečná vzdálenost mezi objekty ve scéně a je závislá na rychlosti jízdy:

$$D = k \cdot v \tag{11}$$

Zde *v* je rychlost jízdy a koeficient *k* je uživatelsky nastavitelný činitel určující bezpečnou vzdálenost. Podmínka (10) je vyhodnocována rovněž vzhledem k aktuální poloze tahu a pokud je v platnosti, je ihned detekována kolize. V případě, že se nějaký objekt nachází v kolizní situaci nebo k ní směřuje, je na tuto skutečnost uživatel vhodným způsobem upozorněn.




**Obrázek 5.7 Schéma kolizní situace**

Ačkoliv se je zde manipulováno se spíše jednoduchými, schematickými modely reálných objektů, v rámci zobecnění je pro samotnou detekci kolizí využito principu obálek. Obálka v prostoru zapouzdřuje samotný model a tím jej zjednodušuje a přináší tak vyšší efektivitu při výpočtu kolize, ale i při vyhledávání, neboť to probíhá v rámci těchto obálek, nikoli samotných (teoreticky i složitějších) modelů. Pro tento účel je užito obálek ve formě vertikálně orientovaných válců pro točny, respektive osově orientovaných kvádrů (AABB – axis aligned bounding box, viz kapitolu 2.5) pro ostatní typy pohonů.

U modelů reprezentujících pohony se nepředpokládá přílišná komplikovanost. A tak jejich obalová tělesa není třeba rozkládat do rekurzivní hierarchie obálek. Avšak pro efektivní vyhledávání kolidujících objektů v prostoru jsem zvolil použití struktury vyhledávacího stromu typu Octree, který rozkládá celý prostor scény na kvádry - oktanty. Každý listový oktant pak obsahuje seznam obálek (zapouzdřujících vždy konkrétní objekt ve scéně), které do něho zasahují. Vyhledávání je pak založeno na porovnávání souřadnic testované obálky se souřadnicemi oktantů v každém uzlu a na postupném zanořování až k listovému oktantu, kde se provede test na kolizi s každou z obálek, jež do oktantu zasahují. Jednotlivé obálky tedy mohou zasahovat do více oktantů.

### *Struktura a výstavba octree*

Detektor kolizí tedy využívá principu obálek a pro efektivní prohledávání scény slouží stromová struktura octree, do níž je celý prostor jeviště dělen. Bylo třeba se rozhodnout jaký typ obálek zvolit. Není triviálním úkolem najít obalové těleso takové, aby bylo dostatečně univerzální a bylo použitelné pro všechny typy pohonů s uspokojivě přesnou mírou aproximace a zároveň si zachovalo dostatečnou jednoduchost a potenciál pro rychlost výpočtů. Implementoval jsem tedy dva typy obálek. Pro točnu je ideální obálkou vertikálně orientovaný válec a pro všechny ostatní typy pohonů jsem implementoval osově zarovnaný kvádr s tím, že pro snadné vkládání do stromové struktury je třeba všechny typy obálek nějak sjednotit. Sestavil jsem tedy rozhraní `IBoundingVolume` a dále třídy reprezentující jednotlivé typy obálek, které toto rozhraní implementují.

Jako základ pro octree jsem použil implementaci datové struktury obecného stromu *SimpleTree* (viz [38]). Tato implementace sestává ze 4 tříd, které jsou přiloženy ke zdrojovým souborům aplikace jako balík *SimpleTree*. Tato implementace obecného stromu skrývá potenciál pro realizaci prakticky libovolného typu stromu. Rozvinul jsem jí do své implementace stromové struktury octree. Základní třída detektoru kolizí pak dědí ze třídy `SimpleTree` reprezentující obecný strom nad typem reprezentujícím uzel octree.

Z obecného stromu a naimplementovaných obálek sestavuji oktalový strom. Ten rekurzivně dělí prostor na oktanty, do nichž je třeba uložit obálky zapouzdřující pohony tak, aby při testování na kolize bylo možné je co nejrychleji vyhledávat. Nejprve je inicializován, tedy sestaven, prázdný strom, což je úkol pro rekurzivní metodu `octreeInitiate`, která postupně rozděluje prostor a buduje strom tak dlouho, dokud není dosažena požadovaná hloubka stromu daná uživatelsky nastavitelným parametrem.

Obálka má definované rozměry a umístění v prostoru. Každý list stromu si udržuje seznam obálek, které do jím vymezeného oktantu zasahují. Třída `OctreeNode` tedy implementuje seznam nad typem obecné obálky. Dále obsahuje specifikaci oktantu, který je uzlem vymezen a je dán souřadnicemi dvou diagonálně protilehlých rohů oktantu. V případě uzlu, který není listem, je seznam obálek vždy prázdný.

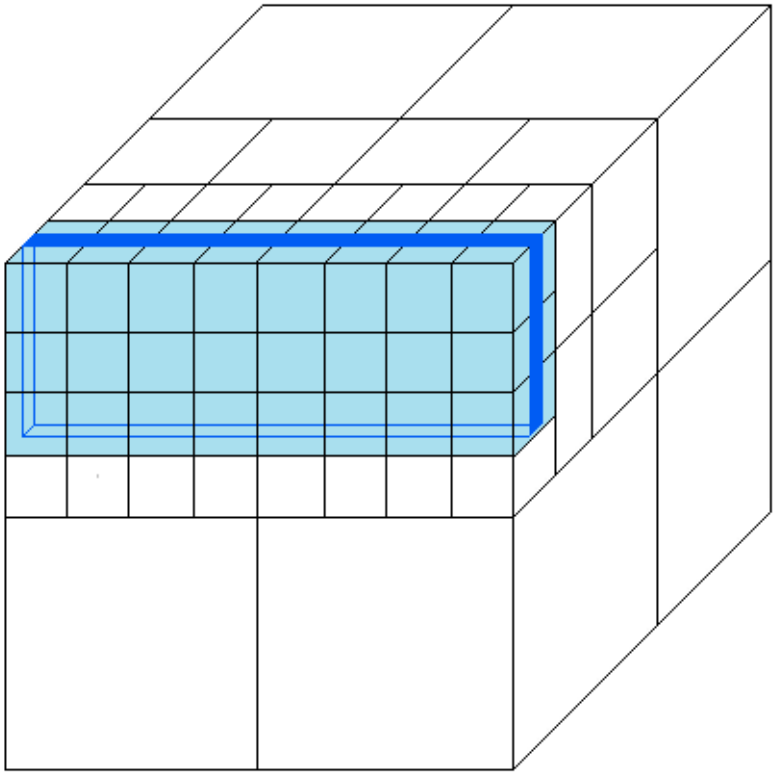
### *Detekce kolizí při vkládání pohonu*

Při vkládání nového pohonu je třeba na něho vložit odkaz do octree stromu detektoru kolizí a zároveň provést test na případné kolize s již existujícími pohony. Obě tyto funkce zajišťuje rekurzivní metoda detektoru kolizí `octreeDriveInsert`.

Metoda bere jako parametr kořen stromu, vkládaný (a testovaný) pohon, hloubku stromu a parametr indikující, zda je pohon svázán s jiným pohonem (buď obsahuje ID nadřazeného pohonu nebo je roven -1). Algoritmus implementuje prohledávání stromu do hloubky, neboť primárním cílem není projít všechny listy, nýbrž najít první listový oktant, obsahující obálku s potenciálně kolidujícím pohonem. Pokud je takový oktant nalezen, není již třeba v prohledávání dále pokračovat a je rovnou vráceno ID kolidujícího pohonu. Pokud do oktantu v listovém uzlu nezasahuje žádná obálka nebo zasahující obálky nejsou s vkládaným pohonem v kolizní situaci, je pro pohon vytvořena nová obálka odpovídajícího typu (vertikálně orientovaný válec pro točnu nebo osově orientovaný kvádr v ostatních případech) a je přidána do seznamu obálek v daném listovém oktantu. Metoda `driveResideInOktant` při průchodu stromem během vyhledávání testuje, ve kterém z potomků uzlu vkládaný pohon leží a tedy do kterého oktantu je třeba se dále zanořit.

Test, zdali je pohon s danou obálkou v kolizní situaci, je realizován pomocí polymorfní metody `collisionWithDrive` rozhraní pro obecnou obálku, jejíž implementace se tedy od typu obálky liší. V každém případě však metoda bere v úvahu parametr, který indikuje, zda jsou pohony svázané.

Může totiž nastat situace, kdy vkládaný pohon bude do svého nadřazeného pohonu přímo vsazen. Za jiných okolností by algoritmus pro takový případ detekoval kolizi – na však v případě svázaných pohonů. Rovněž je testován i opačný případ vazby, tedy zda se testovaný pohon nenachází v seznamu podřízených pohonů vkládaného pohonu.



**Obrázek 5.8 Obálka typu AABB (osově orientovaný kvádr) vložená do struktury octree o hloubce 3**

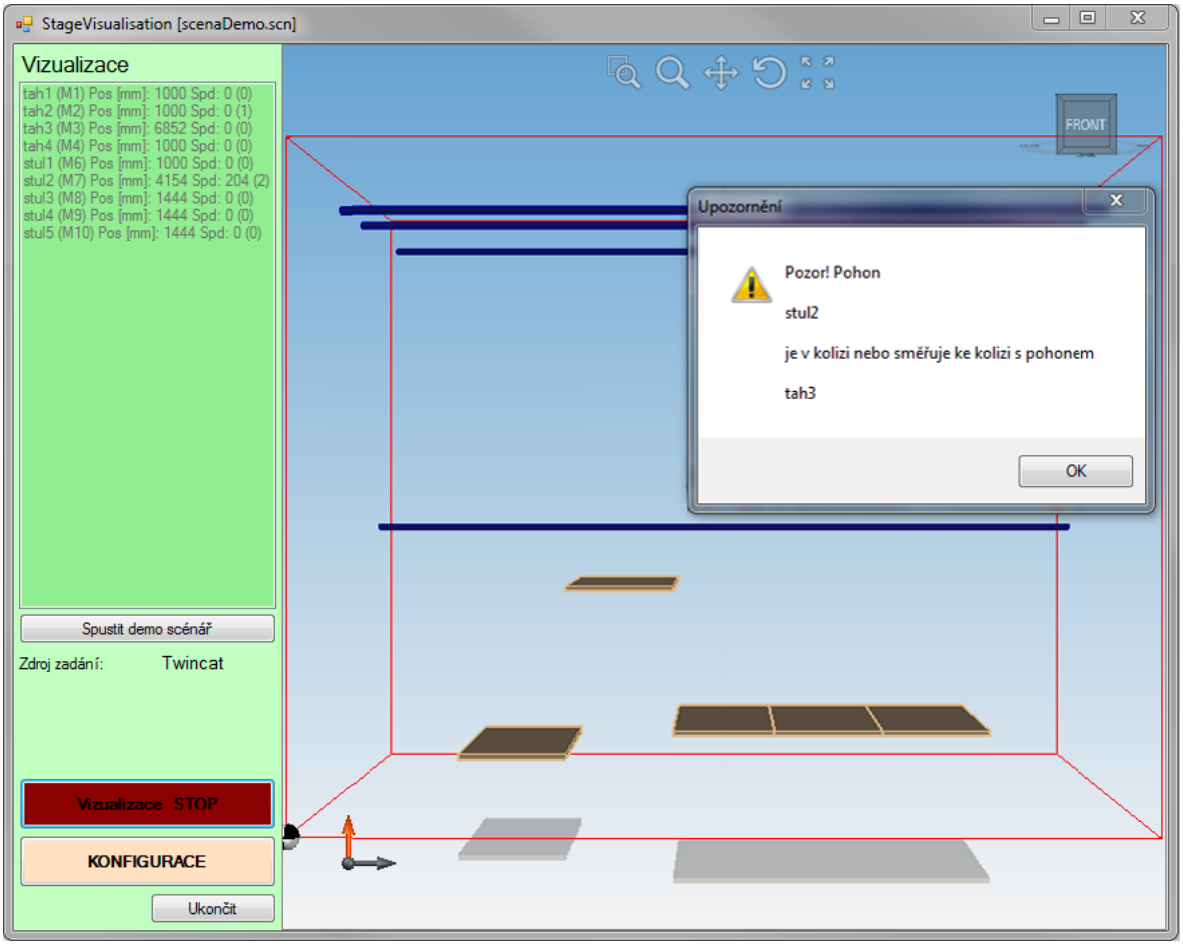
Na Obrázku 5.8 je vidět obálka typu AABB (zapouzdřující například spuštěný tah) zobrazená tmavě modrou barvou. Obálka zasahuje do oktantů zvýrazněných světle modrou barvou. Seznamy obálek ve všech příslušných listových uzlech stromu budou obsahovat právě jednu obálku, a to typu AABB. Při prohledávání stromu je navíc prostor rozkládán pouze tak, jak je naznačeno na obrázku – do ostatních uzlů není nutné aby se algoritmus zanořoval, neboť do nich žádný pohon nezasahuje.

### *Detekce kolizí v režimu vizualizace*

Detekce kolizí v režimu vizualizace znamená detekci kolizí mezi pohony ve chvíli, kdy mohou být v pohybu. Je nutné brát v úvahu jejich aktuální polohu, což vyžaduje průběžnou aktualizaci dat ve vyhledávacím stromu detektoru kolizí. Na druhou stranu, často nejsou v pohybu všechny pohony ve stejném okamžiku, a tudíž není nutné aktualizovat vždy celý model. V každé iteraci procesu vizualizace je tato operace volána pouze pro pohony, jejichž aktuální rychlost je nenulová (tedy není ani kladná ani záporná – pohon není v pohybu ani jedním směrem).

A protože pohony mohou být v pohybu, je třeba kolizi detekovat s předstihem. Jak jsem již zmínil výše, při detekci je tedy brána v úvahu i minimální bezpečná vzdálenost, kterou si pohon musí udržovat od nejbližšího pohonu, ke kterému se blíží, přičemž tato vzdálenost může být navíc závislá na rychlosti pohybu – s rostoucí rychlostí přímo úměrně (podle nastaveného koeficientu) roste

minimální bezpečná vzdálenost. Obrázek 5.9 zobrazuje situaci, kde jeden ze zdvižných stolů je v pohybu směrem vzhůru a blíží se ke spuštěnému prospektovému tahu. Jeho rychlost je přibližně *20 cm/s*, z čehož při nastaveném koeficientu $k = 10$ způsobí, že je hrozící kolizní situace detekována s předstihem dvou metrů před překážkou.

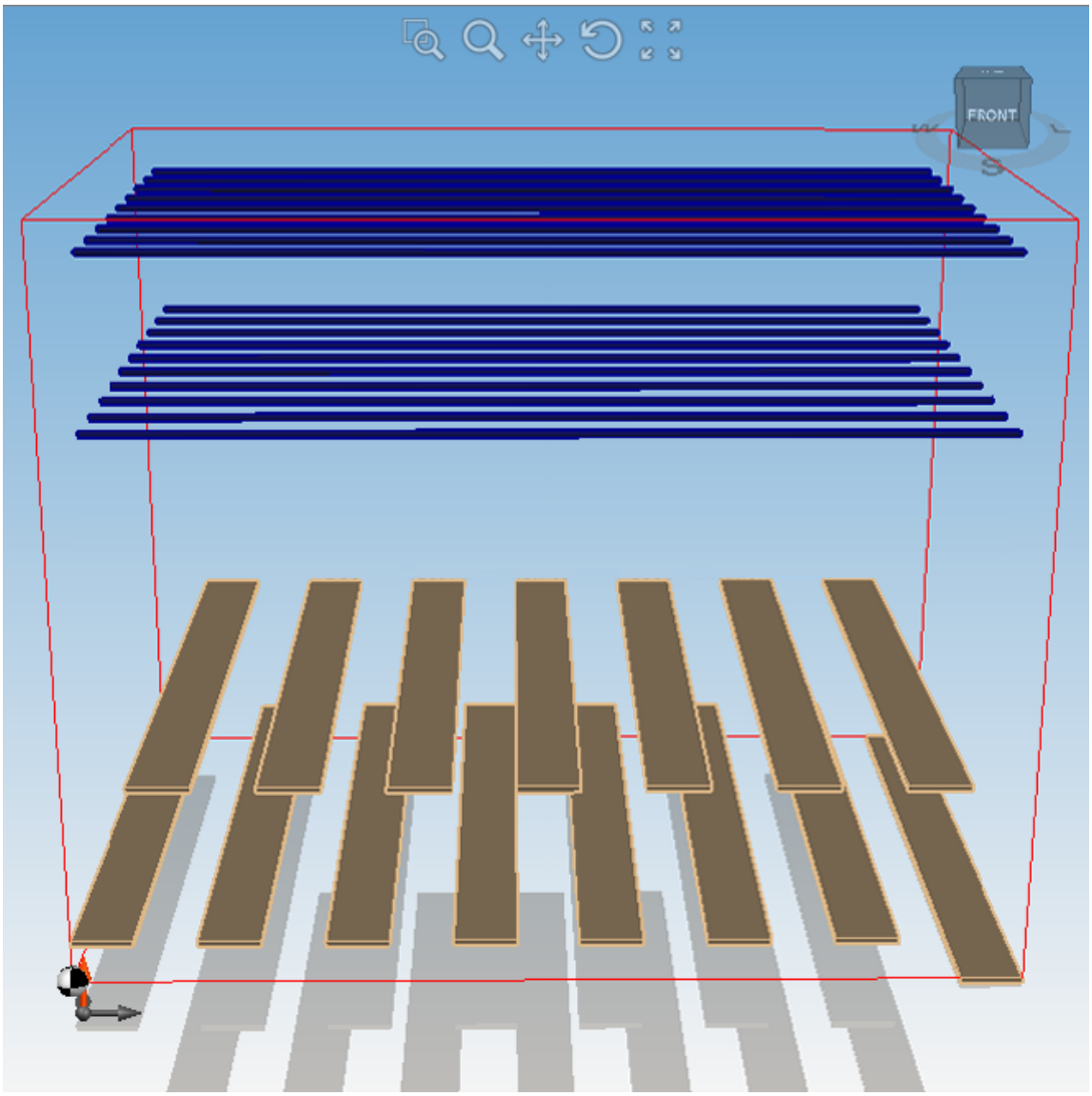



**Obrázek 5.9 Detekovaná kolize během vizualizace - zdvižný stůl směřuje ke kolizi s prospektovým tahem**

Update vyhledávacího stromu zajišťuje metoda `octreeDriveUpdate` a podobně jako při definici pohonu je během vkládání obálek do octree rovnou prováděn test na kolize s ostatními pohony, tak i tato metoda zároveň testuje, zda se daný pohon během vizualizace nedostal do kolizní situace nebo zda k takové situaci nesměřuje. Metoda pracuje na obdobném principu jako metoda pro vkládání s tím, že během průchodu stromem je do rozměrů pohonu již zahrnuta jeho aktuální poloha a navíc i minimální bezpečná vzdálenost. Algoritmus se tedy zanoří to všech oktantů takových, do kterých pohon zasahuje svojí aktuální polohou a do kterých by zasahoval po uražení minimální bezpečné vzdálenosti. Pokud v listovém oktantu nalezne obálku se stejným pohonem, jsou editovány rozměry jeho obálky (se započítanou aktuální polohou, nikoliv minimální bezpečnou vzdáleností). Nalezne-li v oktantu obálku jiného pohonu, je proveden test na kolizi. Je-li detekována kolize, algoritmus končí svojí činnost a prostřednictvím GUI informuje uživatele, v opačném případě je do oktantu přidána obálka s testovaným pohonem).

## 5.5 Testování a zhodnocení výsledků

Výsledná aplikace je schopna graficky vizualizovat zadání příchozí od řídicího systému a rovněž i zadání generované komponentou demo scénáře. Během vizualizace dochází v pravidelných intervalech v překreslování reprezentujících grafických entit pro všechny pohony, pro které je v daný moment dostupné zadání. Překreslování mimo jiné zahrnuje výpočty geometrických transformací, ale i update dat ve struktuře octree vyhledávacího stromu detektoru kolizí. Toto může být v případě více překreslovaných pohonů časově poměrně náročné. Ačkoliv velká divadla mohou disponovat i více pohony, v praxi bývá ve stejném okamžiku v pohybu nejvýše 10 až 20 pohonů (velmi často však méně než 10). Provedl jsem tedy zátěžové testy pro 10, 19 a pro nadsazených 34 pohonů, a to ve 3 sériích – pro různé hloubky octree. Již octree o hloubce 2 dělí prostor na dostatečný počet dílů ($8^2$), avšak při testech jsem volil i vyšší hloubky 3 a 5. Testovací scéna s 34 pohony je pro ilustraci k vidění na Obrázku 5.10, videozáznamy některých testů jsou pak k dispozici na přiloženém CD (viz Přílohu B).




**Obrázek 5.10 Zátěžový test - běžící demo scénář pro 19 tahů a 15 zdvižných stolů**

Všechny provedené testy jsem spouštěl na notebooku o konfiguraci *2GB* operační paměti a procesoru *Intel Core2 Duo U7600* s taktovací frekvenci *1,2GHz*, pod *32b* operačním systémem *Windows 7*. Komunikace s řídicím systémem probíhala v rámci jednoho počítače, na kterém tedy kromě testované vizualizační aplikace běžel i systém *TwinCAT* a obslužná aplikace. Výsledky provedených testů shrnuje Tabulka 5.1.

| Test č. | Hloubka octree | Pohonů celkem (tahů + stolů) | Zadání | Dosažená frekvence překreslování [př./s] |
|---|---|---|---|---|
| 1.1 | 2 | 10 (5 + 5) | ŘS | 35 |
| 1.2 | 2 | 10 (5 + 5) | Demo scénář | 35 |
| 1.3 | 2 | 19 (19 + 0) | Demo scénář | 25 |
| 1.4 | 2 | 34 (19 + 15) | Demo scénář | 15 |
| 2.1 | 3 | 10 (5 + 5) | ŘS | 30 |
| 2.2 | 3 | 10 (5 + 5) | Demo scénář | 30 |
| 2.3 | 3 | 19 (19 + 0) | Demo scénář | 20 |
| 2.4 | 3 | 34 (19 + 15) | Demo scénář | 12 |
| 3.1 | 5 | 10 (5 + 5) | ŘS | 25 |
| 3.2 | 5 | 10 (5 + 5) | Demo scénář | 25 |
| 3.3 | 5 | 19 (19 + 0) | Demo scénář | 20 |
| 3.4 | 5 | 34 (19 + 15) | Demo scénář | 10 |

**Tabulka 5.1 Výsledky zátěžových testů**

Z výsledků testování vyplývá, že se zvyšujícím se počtem vizualizovaných pohonů dosažená frekvence překreslování klesá, a to od téměř plynulé až po hodnotu 10 v případě 34 pohonů, při hloubce vyhledávacího stromu rovné 5. Dosažené výsledky testů poukazují na prostor pro další zvyšování efektivity překreslování modelu, avšak celkově lze výsledky hodnotit jako uspokojivé s přihlédnutím k faktu, že v praxi bude vizualizační aplikace běžet na samostatném a výkonnějším počítači, na kterém lze očekávat výrazné zvýšení frekvence překreslování. Navíc, jak jsem již zmínil výše, typicky nebývají uváděny v pohyb všechny pohony ve stejný okamžik.

Implementovaná vizualizační aplikace představuje funkční nástroj schopný plnit svojí roli v řídicím systému divadelního jeviště, avšak pro jeho plnou využitelnost a nasazení v praxi bude nutné realizovat ještě další rozšíření a funkcionalitu. Jedná se například o podporu vkládání a vizualizaci břemen a také dalších typů pohonů, ale také o podporu komplikovanějších tvarů jednotlivých objektů a celkový směr ke komplexnějšímu modelu scény a tím co nejvěrnější vizualizaci děje na jevišti. Další prostor pro rozšíření se otevírá na poli interaktivity s uživatelem. Jako prakticky využitelný se jeví prostorový editor předdefinovaných pohybů a celých scénářů, včetně definování složitých trajektorií jevištních vozů nebo vícebodových závěsných „létacích zařízení“.

# 6 Závěr

Cílem této práce bylo nastudovat a shrnout dostupnou literaturu, relevantní k problematice 3D modelování a k problematice divadelních řídicích systémů, jejich uživatelských rozhraní a o možnostech využití 3D vizualizace v této oblasti. Poté vytvořit návrh aplikace pro 3D vizualizaci divadelního jeviště včetně jejího uživatelského rozhraní a tento návrh implementovat.

Vytyčený cíl byl splněn. V textu práce je popsáno modelování od obecné roviny tohoto pojmu až po prostorové modelování a počítačovou grafiku, včetně stručného přehledu vybraných vhodných nástrojů. Rovněž je zde popsána technika divadelního jeviště a problematika jejího řízení, s důrazem na interakci řídícího systému s obsluhou, jeho uživatelské rozhraní. V neposlední řadě jsou diskutovány možnosti integrace 3D vizualizace s divadelními řídicími systémy, včetně uvedení příkladů z praxe.

Ze shrnutí literatury vyplývá motivace pro návrh vizualizační aplikace. V textu práce jsou nastíněny požadavky na aplikaci, její koncepce a začlenění do celku řídicího systému, následované popisem samotné realizace aplikace a jejích dílčích komponent. Návrh staví nejen na literatuře, ale i na konzultacích s odpovědnými osobami ve firmě DriveControl s.r.o., na jejichž požadavky i připomínky byl brán zřetel. Výhody a nevýhody navrženého řešení byly rovněž diskutovány. Pro implementaci 3D zobrazení byl zvolen nástroj DevDept Eyeshot ve spojení s programovacím jazykem C#.

Výsledkem práce je aplikace, která je schopna plnit svojí roli v rámci divadelního řídicího systému, prostřednictvím komunikačního rozhraní přijímat zadání pro vizualizaci pohonů horní i dolní mechanizace jeviště a toto zadání graficky vizualizovat. Aplikace rovněž plní i roli sekundárního detektoru kolizí mezi objekty pohybujícími se na scéně. Bylo rovněž provedeno testování, které prokázalo, že při vhodném nastavení je možné i za běhu detektoru kolizí plynule vizualizovat současnou jízdu 10 až 19 pohonů, což je počet, který je v praxi obvykle postačující.

Výsledky této práce slibují technický přínos v oblasti interaktivity při řízení divadelního jeviště a implementovaná vizualizační aplikace je způsobilá pro praktické využití v rámci divadelního řídicího systému. Mě samotnému práce poskytla prohloubení zkušeností v práci s 3D grafikou a rozšíření přehledu o divadelní technice a jejím řízení.

Do budoucna se počítá s realizací dalších rozšíření aplikace. Zejména jde o podporou širší palety nástrojů pro komplexní a detailní návrh scény a její editaci ve 3D a celkové posílení interaktivity.

# Literatura


[1] Křemen, J.: Modely a systémy, Knižnice České matice technické LANNA; svazek 1, Academia, Praha, 2007, ISBN 978-80-200-1477-1

[2] Malík, M.: Počítačová simulace, Skripta pro posluchače matematicko-fyzikální fakulty Univerzity Karlovy, Univerzita Karlova, Praha, 1989

[3] Peringer, P.: Modelování a Simulace IMS, studijní opora, Fakulta Informačních technologií VUT, Brno, 2008

[4] Rábová, Z., Zendulka, J., Češka, M., Peringer, P., Janoušek, V.: Modelování a Simulace, Skripta, Fakulta elektrotechnická VUT, Brno, 1992

[5] Žára, J., Beneš, B., Sochor, J., Felkel, P.: Moderní počítačová grafika, Computer press, Brno, 2004, ISBN 80-251-0454-0

[6] *Geometrie/Afinní transformace souřadnic* [online], Wikiknihy, 2012 [cit. 2013-05-02], dostupné na URL: <http://cs.wikibooks.org/wiki/Geometrie/Afinn%C3%AD_transformace_sou%C5%99adnic>

[7] Strachota, P.: Geometrické transformace pomocí matic, FJFI ČVUT, Praha, 2010

[8] *SketchUp* [online], Wikipedia, 2012 [cit. 2012-12-12], dostupné na URL: <http://cs.wikipedia.org/wiki/SketchUp>

[9] Chopra, A., Town, L.: Google SketchUp, Wiley, 2007, ISBN 978-0-470-17565-1

[10] Branch, J.: *Will I ever become a SketchUp pro?* [online], 2011, dostupné na URL: <http://www.woodfever.net/2011/02/will-i-ever-become-sketchup-pro.html>

[11] Melo, D.: Řídicí systém divadelní scény s grafickým rozhraním, bakalářská práce. Fakulta informačních technologií VUT, Brno, 2010.

[12] *DevDept* [online]. Webová prezentace firmy [cit. 2012-12-25] dostupné na URL: <https://www.devdept.com/>

[13] Lin, M. C., Gottschalk, S.: Collision detection between geometric models: a survey, University of North Carolina.

[14] *Lecture: Graphics pipeline and animation* [online], [cit. 2012-12-10], RMIT University, Australia, 2012, dostupné na URL: <http://goanna.cs.rmit.edu.au/~gl/teaching/Interactive3D/2012/lecture2.html>

[15] Krystl, F.: Detekce kolizí pro simulaci dynamických jevů v 3D scénách, diplomová práce, Fakulta elektrotechnická, České vysoké učení technické v Praze, Praha, 2010.

[16] *Bounding volume* [online], Wikipedia, 2012 [cit. 2013-01-06], dostupné na URL: <http://en.wikipedia.org/wiki/Bounding_volume>

[17] Haverkort, H. J.: Introduction to bounding volume hierarchies, výňatek z disertační práce, Utrecht University, 2004.

[18] *Bounding volume hierarchy* [online], Wikipedia, 2013 [cit. 2013-04-07], dostupné na URL: <http://en.wikipedia.org/wiki/Bounding_volume_hierarchy>

[19] Frish, N.: Procedures for Assisting the Workflow of Car Crash Simulations, shrnutí disertační práce, Institute for Computer Science, Stuttgart University, dostupné na URL: <http://www.vis.uni-stuttgart.de/~frisch/h/diss.htm>

[20] *Octree* [online], Wikipedia, 2013 [cit. 2013-04-07], dostupné na URL: <http://en.wikipedia.org/wiki/Octree>

[21] Knoll, A.: A Survey of Octree Volume Rendering Methods, Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, Utah.

[22] Bezděk, P.: Jevištní technologické zařízení – divadla. Typizační směrnice všeobecná. Ministerstvo kultury ČSR, Praha, 1987.

[23] Jančík, J.: Moderní řídicí systémy jevištní techniky, Magazín AUTOMA 3/2009. FCC Public s.r.o., Praha, ČR, 2009, dostupné na URL: <http://www.odbornecasopisy.cz/index.php?id_document=38732>

[24] Ševcovic, J.: Řídicí systém divadelní scény s grafickým rozhraním, diplomová práce, Fakulta informačních technologií, Vysoké učení technické v Brně, Brno, 2009.

[25] *Elseremo a.s.* [online]. Webová prezentace firmy. 2008 [cit. 2010-04-08]. Dostupné na URL: <http://www.elseremo-stage.com/cz/>

[26] *Control system* [online], Wikipedia, 2012 [cit. 2012-12-08], dostupné na URL: <http://en.wikipedia.org/wiki/Control_system>

[27] Modrlák, O.: Fuzzy řízení a regulace. Teorie automatického řízení II, Studijní materiál, Technická univerzita v Liberci, 2002

[28] *PID regulátor* [online], Wikipedia, 2012 [cit. 2012-12-08], dostupné na URL: <http://cs.wikipedia.org/wiki/PID_regulátor>

[29] *Co znamená PID*, Magazín AUTOMA 3/2003. FCC Public s.r.o., Praha, ČR, 2003, dostupné na URL: <http://www.odbornecasopisy.cz/index.php?id_document=28768>

[30] Otčenášek, M. Distribuované řídící systémy a jejich využití v praxi, Diplomová práce, Fakulta Elektrotechniky a komunikačních technologií VUT, Brno, 2008

[31] Bradáč, Z.: Hierarchické decentralizované systémy řízení, zkrácená verze habilitační práce, Fakulta elektrotechniky a komunikačních technologií, Brno, 2008, ISBN 978-80-214-3577-3

[32] *Rexroth Drive and Control Technology*, Prezentace firmy, Bosh Rexroth Group [cit. 2012-12-12]

[33] *Příručka k divadelnímu řídicímu systému*. Elseremo a.s. [cit. 2012-12-12].

[34] *Stage Technologies Ltd.* [online]. Webová prezentace firmy [cit. 2012-12-13] dostupné na URL: <http://www.stagetech.com/products>

[35] *Creative Conners, inc.* [online]. Webová prezentace firmy [cit. 2012-12-25] dostupné na URL: <http://creativeconners.com/>

[36] Mitchell, M. A.: SIL Made Simple, Cameron Flow Control.

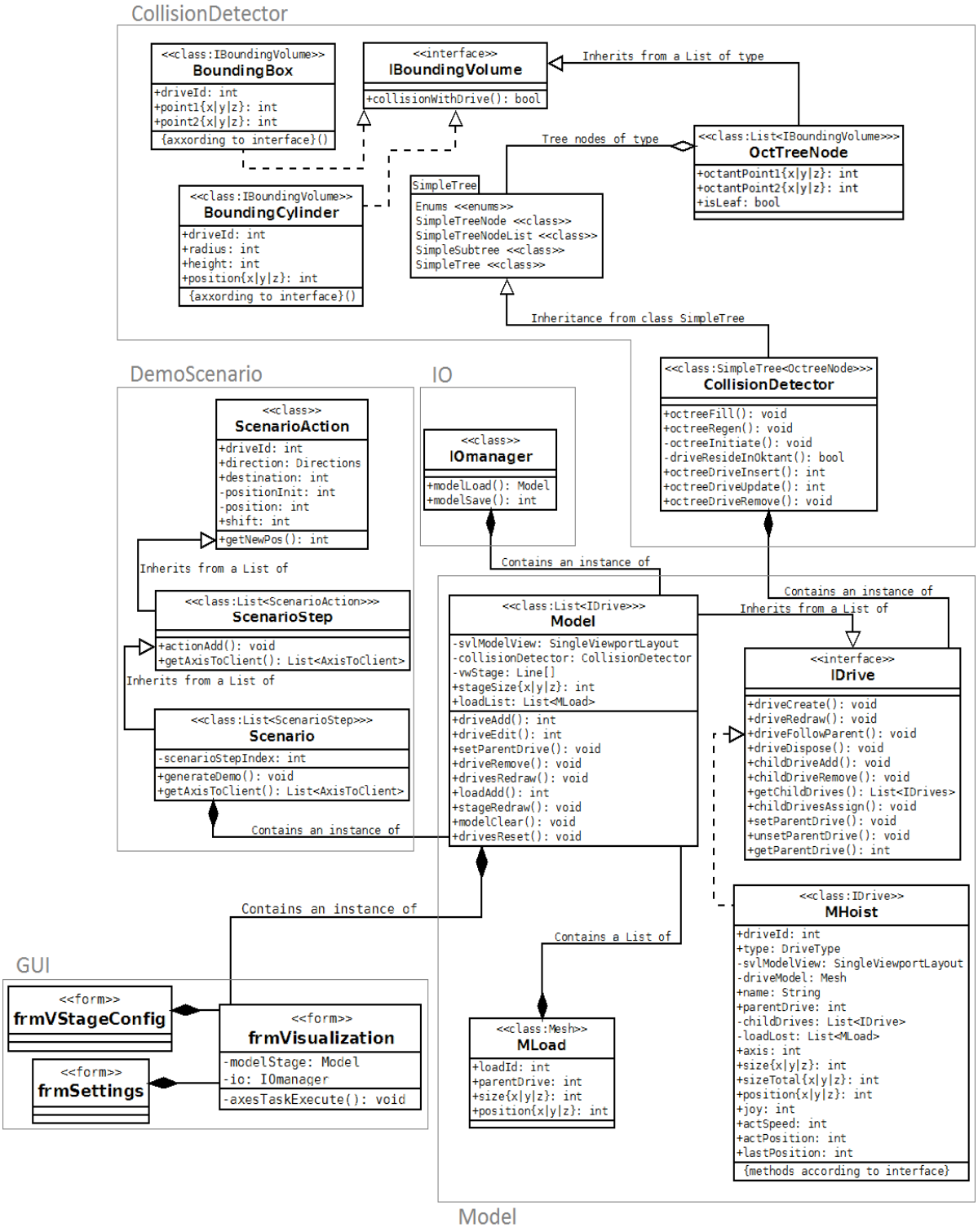
[37] *Beckhoff.* [online]. Webová prezentace firmy [cit. 2013-04-05] dostupné na URL: <http://www.beckhoff.com/english.asp?twincat/default.htm>

[38] *Vanderboom, D.: Tree<T>: Implementing a Non-Binary Tree in C#.* [online]. Popis implementace [cit. 2013-04-14], 2008, dostupné na URL: <http://dvanderboom.wordpress.com/2008/03/15/treet-implementing-a-non-binary-tree-in-c/>

# Seznam příloh

# Příloha A – UML model aplikace

*Diagram poskytuje přehled implementovaných tříd, seskupených do pěti komponent aplikace a jejich nejdůležitějších atributů a metod. Třídy reprezentující pohony jsou pro příklad zastoupeny pouze třídou* `MHoist`*, která reprezentuje tah. Znaky „+“ před metodami a atributy značí specifikátor přístupu* `public`*, znaky „-“ pak* `private`*.*




**UML model struktury tříd vizualizační aplikace**

# Příloha B – obsah přiloženého CD

*Následující přehled reprezentuje strukturu adresářů na přiloženém CD a zobrazuje nejdůležitější soubory, včetně jejich stručného komentáře. Podrobnější popis je pak k dispozici v souboru Readme.*

|_ [Root]
- |_ [Aplikace]
  - |_ [Podpurna data a SW]
    - |_ [Demo - pult]
      - *(demonstrační verze obslužné aplikace ŘS)*
      - |_ BDTDesign.exe
        *(Spustitelný soubor aplikace obslužného pultu)*
    - |_ [Demo - twincat]
      - |_ DriveControlTemplate.pro
        *(Demonstrační zdrojový projekt pro PLC ŘS)*
      - |_ DriveControlTemplateDemo.tsm
        *(TwinCAT System Manager)*
    - |_ [Eyeshot]
      - |_ EyeshotProfessional60232.exe
        *(Nástroj DevDept Eyeshot)*
  - |_ [Pripravene modely]
    - *(Složka s předchystanými modely scén)*
  - |_ [Vizualizacni aplikace - release]
    - *(Složka se zkompilovanou aplikací)*
    - |_ StageVisualisation.exe
      *(Spustitelný soubor Vizualizační aplikace)*
    - |_ init.XML
      *(inicializační XML soubor aplikace)*
  - |_ [Vizualizacni aplikace - zdrojovy projekt]
    *(Zdrojové soubory aplikace)*
- |_ [Demonstracni video]
  - |_ DemoVideoDP.wmv
    *(video demonstrující běh vizualizační aplikace)*
- |_ [Technicka zprava]
  - |_ [Zdrojovy format]
    - |_ TechnickaZprava_DIP.doc
      (*Zdrojový soubor technické zprávy)*
  - |_ TechnickaZprava_DIP.pdf
    *(Technická zpráva diplomové práce ve formátu PDF)*
- |_Readme
  *(Soubor obsahující podrobný obsah CD a instrukce k instalaci a spuštění aplikace)*