ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

Katedra řídící techniky

Bakalářská práce

Prostorové zobrazení automatu

leden 2007

Autor práce: **Jiří Zavrtálek**

Vedoucí práce: **Ing. Richard Šusta, Ph.D.**

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze, dne 19.1.2007

Jiří Zavrtálek

České vysoké učení technické v Praze, fakulta elektrotechnická

Katedra řídicí techniky Školní rok:2005/2006

Zadání bakalářské práce

Student: Jiří Zavrtálek

Obor: Kybernetika a měření

Název tématu: Prostorové zobrazení automatu

Zásady pro vypracování:

1. Prostudujte si předchozí diplomovou prací Planar.
2. Navrhněte program pro prostorové kreslení stavových automatů pomocí OpenGL.
3. Program realizujte v jazyce C# pod VisualStudio. 2005.

Seznam odborné literatury: Dodá vedoucí práce

Vedoucí bakalářské práce: Ing. Richard Šusta, Ph.D.
Datum zadání bakalářské práce: zimní semestr 2005/06

Termín odevzdání bakalářské práce: 30. 6. 2006

Prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

L.S.

Prof. Ing. Zbyněk Škvor, CSc.
děkan

V Praze, dne 1. 2. 2006

Anotace

Bakalářská práce se zabývá prostorovým zobrazením orientovaných grafů. Ty jsou totiž vhodným nástrojem pro popis sekvenčních automatů. Hlavní náplní práce byl vývoj aplikace v jazyce C#, která by s využitím grafického rozhraní OpenGL toto zobrazení umožňovala. Práce částečně vychází z diplomové práce ing. Haščáka.Ta řešila obdobnou problematiku ovšem s důrazem na planární zobrazení grafu a algoritmy k jeho vytvoření..

Důležitými výstupem této práce je za prvé nástin postupu, kterým lze tvořit aplikace s podporou OpenGL v jazyce C#, který standardně podporu pro toto rozhraní nenabízí. Za druhé je to samotná aplikace umožňující interaktivní tvorbu, úpravu a zobrazení orientovaných grafů v prostoru.

Annotation

The bachelor thesis considers threedimensional rendering of oriented graphs. These are namely suitable tool for describing sequence automaton. Main contents of work involved creation of an application in C# that would allow this with a support of OpenGL interface. Work goes out from a diploma thesis by ing. Haščák that dealt with planar embedding of oriented graph.

Main outputs of the work are at first summary of ways to involve OpenGL to a C# application, secondly the application itself that allows interactive design, editing and display of an oriented graph in 3D space.

Úvod

Cílem této bakalářské práce bylo vytvoření aplikace k prostorovému zobrazení automatů s využitím jazyka C#.

Vhodným prostředkem interpretace stavového automatu je jistě orientovaný graf, definovaný množinami vrcholů a hran, podobně jako automat mající množiny stavů a přechodů mezi nimi. Lze ho tedy popsat jako prostorové zobrazení orientovaného grafu.

K produkci trojrozměrného obrazu se využilo grafické rozhraní OpenGL. Jako inspiraci pro svou tvorbu jsem získal kopii programu Planar, který vytvořil ing. Radek Haščák v rámci své diplomové práce v roce 2002, a to v jazyce C++. Zabýval se zejména zobrazením rovinných grafů.

Hlavní úkoly řešené v této práci byly zhruba následující:

- prostudovat možnosti použití knihovny OpenGL v jazyce C#;
- prostudovat základy teorie grafů a navrhnout datové struktury vhodné pro zpracování, vizualizaci a uložení grafu;
- vyvinout aplikaci pro vizualizaci orientovaného grafu v trojrozměrném prostředí, která nabídne některé možnosti k jeho zpracování, ale zejména umožní jeho interaktivní návrh a úpravy.

Výše uvedený seznam úkolů lze současně považovat za osnovu, podle které jsem k řešení problému přistupoval. V následujícím textu čtenáře seznámím s konkrétním řešení zadaného problému.

1. Grafické rozhraní OpenGL

Jelikož pro zobrazení 3D objektů používám v programu právě tuto knihovnu, je na místě čtenáře případně neznalého tohoto standardu v krátkosti seznámit s jeho možnostmi. Navíc zpřístupnění jeho funkcí v jazyce C# není triviální, a proto v závěru kapitoly vyjmenuji několik možností jak toho dosáhnout a zkonkretizuji svůj přístup k řešení tohoto problému.

1.1 Základní seznámení

1.1.1 Co je OpenGL

OpenGL (Open Graphic Library) byla vytvořena na počátku devadesátých let minulého století firmou SGI. Jejím předchůdcem byla knihovna IRIS téže firmy. Jedná se o knihovnu na platformě nezávislou, umožňující velmi efektivně pracovat s trojrozměrnou grafikou. Knihovna se během již téměř 15 let své existence stala široce uznávaným standardem v oblasti grafickým aplikací, virtuální reality, inženýrských aplikacích typu (CAD).

1.1.2 Proč se používá

Jedná se vlastně o rozhraní API (Application Programming Interface) mezi programem a grafickým hardware počítače. Knihovna je tvořena více jak stovkou funkcí, zahrnujících zejména příkazy pro tvorbu jednotlivých objektů na scéně, specifikaci jejich tvaru a povrchu, nastavení světelných podmínek scény, volbu perspektivy a mnoho dalších. Díky nim mohou být úseky programů řešící právě zobrazení trojrozměrné scény poměrně krátké a přehledné.

Delší nespornou výhodou je fakt, že již dlouhou dobu drtivá většina grafických karet obsahuje podporu hardwarové akcelerace funkcí knihovny OpenGL, takže množství výpočtů spojených vykreslením a trojrozměrné scény je realizováno grafickou kartou, což výrazně šetří výpočetní čas procesoru.

1.1.3 Základní principy a součásti

Pomocí knihovny OpenGL je možné vykreslovat obrazce složené ze základních prvků, které nazýváme **primitiva**. Mezi ně patří bod, úsečka, trojúhelník, čtyřúhelník, bitmapa (rastrový obrázek). Na vrcholy těchto primitiv mohou pak být aplikovány geometrické transformace, jako jsou rotace, posuny, změny měřítka a perspektivy. Na tyto primitiva lze dále aplikovat osvětlení a textury. Tyto operace jsou přímo podporovány grafickým hardware.

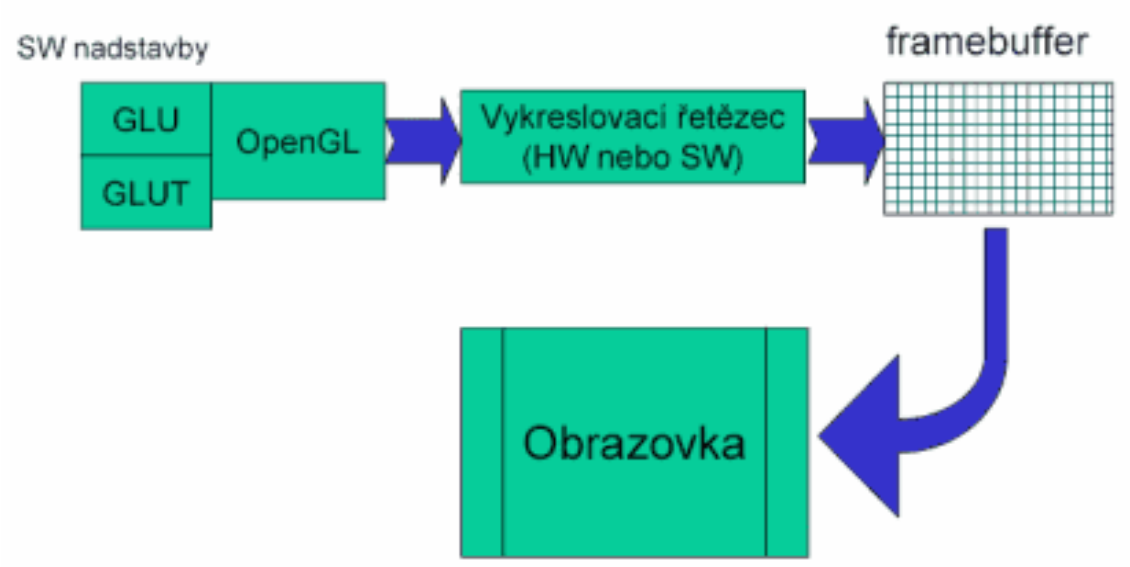
Pokročilejší funkce výrazně rozšiřující možnosti rozhraní lze nalézt v softwarových nadstavbách, tedy rozšiřujících knihovnách. Nejpoužívanější z nich je GLU (OpenGL Utilities), kde nalezneme například funkce pro vykreslení kvadrik. O dalších rozšířeních a nadstavbách se lze dočíst např. v [4].

1.1.4 Rendering

K anglickému termínu rendering, v přímém překladu znamenajícím ztvárnění, spodobnění, neexistuje český ekvivalent. V následujícím textu ho budu používat jako technický termín.

Postup, kterým probíhá rendering v OpenGL, nastiňuje následující obrázek 1.1. Vysvětlíme si ho na jednoduchém příkladu.

Mějme scénu tvořenou jednou osvětlenou kouli. Tu v programu vytvoříme jednoduchým použitím funkce z nadstavbové knihovny GLU. Koule se převede na množinu primitiv, zřejmě trojúhelníků, provede se výpočet vrcholů a normálový vektorů potřebných při výpočtu osvětlení. Tento převod výpočetně zatíží procesor. Dále se provedou výpočty spojené s transformacemi souřadnic a výpočty osvětlení jednotlivých bodů atd., čili zkrátka zobrazení 3D scény na 2D rastrový obrázek, který se uloží do tzv. framebufferu. Tyto výpočty již zpravidla obstarává grafický subsystém počítače.



Obrázek 1.1 : Jak probíhá rendering

OpenGL se z pohledu programátora chová jako stavový stroj. Obsahuje množství stavových proměnných, ke kterým programátor přistupuje ne přímo, ale pomocí určených funkcí. Jejich hodnota se nemění, dokud ji programátor opět explicitně nezmění. Tato filozofie umožňuje realizovat architekturu Klient – Server. Uživatelský program (Klient) zasílá své požadavky OpenGL (Server). Výpočet tak může být prováděn na jiném počítači než zobrazení.

1.2 OpenGL v C#

V této kapitole se zmíním o několika způsobech, jak implementovat a umožnit použití knihovny OpenGL v jazyce C#. Podrobněji se poté zmíním o řešení, které jsem upřednostnil pro použití v programu GrafMan3D.

1.2.1 Možnosti implementace

Jelikož jazyk C# potažmo platforma .NET Framework nebyly od počátku vývojáři obdařeni podporou tohoto grafického rozhraní, vzniklo zejména na počátku tohoto desetiletí několik nezávislých projektů, které se snažily tento problém odstranit. Přístup tvůrců těchto řešení k danému problému byl často poměrně odlišný a samozřejmě stěží lze pomyslet o nějaké přenositelnosti kódu obstarávajícího vykreslování pomocí OpenGL mezi programy využívajících odlišných řešení. Je tedy na zvážení každého programátora, které jedno z nabízených řešení použije.

1.2.2 Seznam několika nejpoužívanějších knihoven

Níže uvedená řešení lze považovat za poměrně prověřená, u některých již byl vývoj ukončen s poznámkou, že již není co zlepšovat.

Tao Framework

Jedná se o skutečně robustní sadu tříd pro připojení řady nativních knihoven. Kromě OpenGL zpřístupňuje například Cg, DevIL, FreeGLUT, ODE, OpenAL. Domovská stránka projektu je k nalezení na http://www.taoframework.com.

CsGL

Další poměrně populární knihovna. Bližší informace o ní lze získat na stránkách projektu http://csgl.sourceforge.net.

SharpGL

Další knihovna umožňující využití OpenGL. Jejím specifikem je fakt, že obsahuje řadu zjednodušení, zejména pro práci s Windows formuláři. Nevýhodou je, že napsaný kód pak již samozřejmě není lehce přenositelný na jinou platformu.

Colin P. Fahey's C# OpenGL Wrapper

Nakonec knihovna, kterou jsem se rozhodl použít ve svém programu. Přesvědčila mě zejména jednoduchost až strohost tohoto řešení spojená se snadným použitím. Prakticky se nejedná o knihovnu, nýbrž o sadu šesti tříd, které se připojí k projektu a tím zpřístupní či vlepí (wrap) funkce rozhraní OpenGL.. Aktuální verze je dostupná na http://www.colinfahey.com.

Pozn : Informace o tomto wrapperu se týkají verze vydané v roce 2005. Na internetu se dostupná novější verze, u které jsou některé aspekty implementace pozměněny.

1.2.3 Colin P. Fahey's C# OpenGL Wrapper

Instalace probíhá přidáním souboru tříd do vlastní aplikace. Jedná se o následující soubory:

wglav5_gl.cs: Soubor obsahuje definici třídy GL. Zpřístupňuje všechny funkce a konstanty hlavičkového souboru gl.h, který je součástí systému Windows.

wglav5_glu.cs: Soubor obsahuje definici třídy GLU. Analogicky s předchozí třídou zpřístupňuje funkce a konstanty obsažené v glu.h.

wglav5_wgl.cs: Soubor obsahuje definici třídy WGL, která koresponduje s nativní knihovnou WGL obsaženou ve Windows.

wglav5_gdi.cs, wglav5_user.cs: Třídy GDI a USER obsahují důležité funkce pro vykreslení OpenGL obsahu do okna Windows.

wglav5_demo.cs: Definuje třídu DEMO obsahuje kód příkladu demonstrující animaci a použití fontů. Ačkoli není nedílnou součástí balíku tříd, mnoho ponechal jsem ji ve svém programu, neboť obsahuje množství metod realizujících zejména inicializaci, které jsem použil i ve své třídě obstarávající vykreslení 3D scény .

Pro úspěšné použití wrapperu je nutné překladači povolit použití nechráněného kódu, tzv. pointrů, které jsou v nativních knihovnách OpenGL hojně využívány.

Ve svém kódu intenzivně využívám téměř výhradně funkcí třídy GL, tedy funkce základní knihovny. Pro vykreslení kvadrik, které používám pro tvorbu objektů reprezentujících vrcholy a hrany, volám funkce třídy GLU, které pomocí nichž lze vytvořit i tyto složité mnohapolygonové objekty.

2. Teorie grafů

Jelikož se tato práce zabývá prostorovým zobrazení grafů, a to konkrétně grafů orientovaných, je záhodno uvést určité množství teorie o grafech alespoň v objemu, jež je nutný k vysvětlení pojmů a vlastností grafů, které využívá aplikace GrafMan3D.

2.1 Základní pojmy

Pro sepsání této kapitoly jsem čerpal z [1]. Sem také odkazuji případné zájemce o hlubší proniknutí do problematiky grafů.

2.1.1 Definice grafu

Orientovaný graf, vrcholy a hrany. *Orientovaným grafem* nazveme trojici G = {V,E,ε}, kde V je konečná neprázdná *množina vrcholů* a E je konečná *množina hran*.

Zobrazení $\varepsilon : E \rightarrow V^2$, které nazýváme *vztahem incidence*, přiřazuje každé hraně uspořádanou dvojici vrcholů (x,y) z množiny V. První z této dvojice je počátečním vrcholem hrany, druhý se nazývá koncovým vrcholem hrany.

Vrcholy a hrany. Nechť hrana *e* spojuje vrcholy *x,y*. Pak řekneme, že vrcholy *x* a *y* jsou incidentní s hranou *e*, a rovněž hrana *e* je incidentní s vrcholy *x* a *y*. Hrany, jež mají shodný počáteční a koncový vrchol nazýváme *rovnoběžné* nebo též *násobné*. Hranu, jejíž počáteční a koncový vrchol jsou shodné nazveme *smyčka*. Vrchol, který není incidentní s žádnou hranou, nazveme *izolovaným vrcholem*. *Stupněm vrcholu* nazveme číslo, které odpovídá počtu hran incidentních s tímto vrchol. Jelikož je to přesněji součet počtu hran vycházejících z vrcholu a počtu hran do něj vstupujících, započítávají se smyčky do součtu dvakrát.

Poznámka. Ve své práci se zabývám zejména orientovanými grafy. Některé z dále uvedených pojmů mají své definice pro grafy orientované i pro grafy neorientované. Omezím se z úsporných důvodů na definice pro grafy orientované.

2.1.2 Sled, tah, cesta

Sled. Posloupnost $v_0, e_1, v_1, e_2, v_2, e_3, \ldots v_{k-1}, e_k, v_k$, kde v_i jsou vrcholy a e_i hrany grafu, nazveme *orientovaným sledem*, jestliže jednotlivé sousední prvky posloupnosti jsou vzájemně incidentní a platí, že vždy vrchol bezprostředně předcházející hraně je jejím počátečním vrcholem a vrchol bezprostředně následující hranu je jejím vrcholem koncovým. *Triviální sled* je takový, který obsahuje jediný vrchol a žádnou hranu. Sled, v němž se neopakuje žádná hrana, označujeme jako *tah*. Sled, v němž se neopakuje žádný vrchol, označujeme jako *cesta*.

Sled nazveme *uzavřený*, pokud v něm splývají počáteční a koncový vrchol, tedy platí $v_0 = v_k$. Termín uzavřenosti pak lze přenést i na pojmy tah a cesta. Uzavřenou orientovanou cestu označujeme jako *cyklus*.

Dostupnost. Vrchol *y* je orientovaně *dostupný* z vrcholu *x*, jestliže existuje cesta vedoucí z vrcholu *x* do vrcholu *y*. Definujme následující množiny:

$D^+(x)$ množina vrcholů orientovaně dostupných z vrcholu x.

$D^-(x)$ množina vrcholů, ze kterých je orientovaně dostupný vrcholu x.

Definujeme *matici dostupnosti* D tak, že položíme $d_{ij} = 1$, právě když vrchol v_j je dostupný z vrcholu v_i, jinak položíme $d_{ij} = 0$.

2.1.3 Speciální typy grafů

Prostý graf. *Prostý graf* je takový, jehož hrany mají násobnost nejvýše jedna. Tento graf tedy neobsahuje rovnoběžné hrany. *Multigraf* naopak takové hrany obsahovat může. V této práci se zabývám pouze grafy prostými.

Podgraf. Graf G' je *podgrafem* grafu G, když vznikne z grafu G vynecháním nějakých vrcholů a hran. G' ovšem musí zůstat grafem, a tedy s každou hranou musí obsahovat oba její vrcholy.

Kořenový strom. *Kořenový strom* je orientovaný graf,v němž existuje význačný vrchol r (zvaný kořen), který je pro všechny své incidenčním hrany pouze vrcholem počátečním (tedy do něj žádná hrana nevede). Do všech ostatních vrcholů vede jen jedna hrana a jsou dostupné z kořene.

Acyklický graf. *Acyklický graf* je orientovaný graf, který neobsahuje žádný cyklus.

2.1.4 Silná souvislost a související pojmy

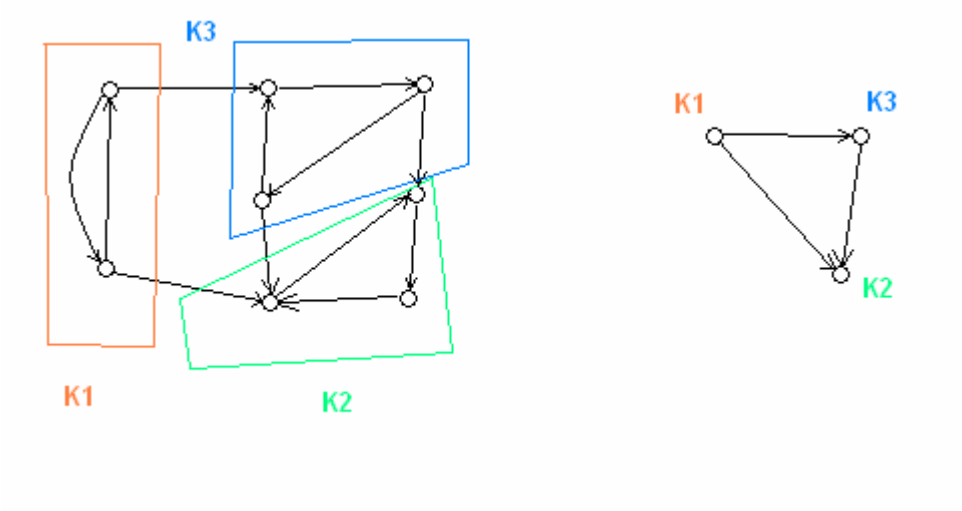
Silná souvislost. Orientovaný graf nazveme *silně souvislým*, právě když existuje orientovaná cesta mezi každou dvojicí jeho vrcholů. *Silně souvislou komponentou* grafu G nazveme každý podgraf G` grafu G, který je silně souvislý a je maximální, tedy není kromě sebe sama není podgrafem žádného rozsáhlejšího silně souvislého podgrafu grafu G.

Důležité: Každý vrchol grafu leží právě v jedné komponentě souvislosti grafu.

Kondenzace. *Kondenzace* grafu G je takový prostý orientovaný graf G_K bez smyček, pro který platí:

1. Množinu vrcholů grafu G_K tvoří všechny silně souvislé komponenty grafu G

2. Vrcholy K_1, K_2 grafu G_K (tedy silně souvislé komponenty grafu G) jsou spojeny hranou, jestliže v grafu G existuje hrana spojující nějaký vrchol z komponenty K_1 s nějakým vrcholem komponenty K_2.

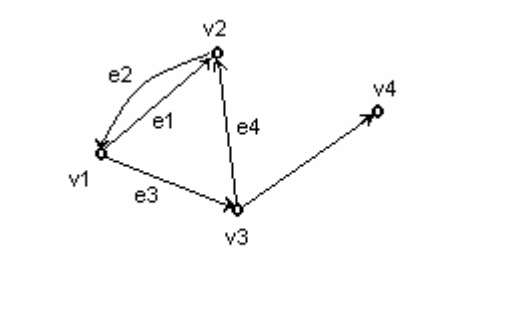


Obr 2.1 Orientovaný graf s vyznačením silných komponent (vlevo)
a jeho kondenzace (vpravo)

2.2 Základní způsoby popisu a zadávání grafu

Pro mnohé je zřejmě nejsrozumitelnějším popisem grafu obrázek, tedy alespoň těch méně rozsáhlých. Je ale jasné ,že tato grafická reprezentace není vhodná pro zadání a zpracování na počítači a často svádí k jednostrannému pohledu.

Uvedu zde jeden příklad menšího grafu, jenž dále využiji při objasňování dalších metod popisu.



Obr. 2.2 : Příklad orientovaného grafu

Matice sousednosti. Mějme orientovaný graf *G* s počtem vrcholů *n*. Zvolíme-li nějaké pevné pořadí vrcholů $v_1, v_2, ..., v_n$, můžeme vytvořit matici sousednosti M_G^+ řádu n, příslušející grafu *G* , a to předpisem

$$m_{ij}^+ = m(v_i,v_j)$$

Symbolem *m(x,y)* rozumíme násobnost orientované hrany *(x,y)*. Čísla v této matici tedy představují počet hran vedoucích z vrcholu daného indexem řádku do vrcholu daného indexem sloupce matice.

Příklad**:** Matice sousednosti pro graf na obrázku 2.2:

$$M_G^+ = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Matice incidence. Mějme orientovaný graf G, který obsahuje n vrcholů,, m hran a je bez smyček. Zvolíme pevné pořadí vrcholů a hran $v_1, v_2, ...,v_n$ a $e_1, e_2 , ...,e_m$. Můžeme nyní grafu G přiřadit matici incidence (též incidenční matici) B_G typu (n,m) danou předpisem

$b_{ij} = 1$, jestliže v_i je počátečním vrcholem hrany e_j,
$b_{ij} = -1$, jestliže v_i je koncovým vrcholem hrany e_j,
$b_{ij} = 0$, jestliže v_i není incidentní s hranou e_j.

Jelikož každá hrana má jeden počáteční a jeden koncový vrchol, obsahuje každý sloupec incidenční matice pravě jednu hodnotu -1 a jednu hodnotu 1.

Příklad: Matice incidence pro graf na obrázku 2.2:

$$B_G = \begin{pmatrix} 1 & -1 & 1 & 0 & 0 \\ -1 & 1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 1 & 1 \\ 0 & 0 & 0 & 0 & -1 \end{pmatrix}$$

Seznamy vrcholů a hran. Množina vrcholů je dána prostým výčtem prvků. Množina hran je popsána seznamem trojic, které jsou tvořeny jménem hrany a jmény jejího počátečního a koncového vrcholu. V případě, že nezáleží na pojmenování hran, lze použit jen seznam dvojic vrcholů.

Příklad**:** Popis grafu na obrázku 2.2 seznamem vrcholů a hran:

Vrcholy : v1, v2, v3, v4.
Hrany : (e1, v1, v2), (e4, v3, v2),
(e2, v2, v1), (e5, v3, v4),
(e3, v1, v3).

Seznamy vrcholů a seznamy okolí vrcholů. Jedna se o variantu předchozího způsobu. Pro každý vrchol v je uveden seznam dvojic tvořených názvem hrany, pro kterou je vrchol v počátečním vrcholem, a koncovým vrcholem této hrany.

Příklad: Popis grafu na obrázku 2.2 seznamem vrcholů a jejich okolí:

v1: (e1, v2), (e3, v3);
v2: (e2, v1);
v3: (e4, v2), (e5, v4);
v4: prázdný seznam

2.3 Aplikace jednotlivých metod popisu při počítačovém zpracování

2.3.1 Maticový popis

Maticový popis je poměrně přehledný a matematicky elegantní, ale má jisté závažné nedostatky, a proto je zřídkakdy využíván při počítačovém zpracování. Počet hodnot v matici sousednosti roste kvadraticky s počtem vrcholů a pokud graf obsahuje malý počet hran, tak je matice velmi řídká (většina prvků je nulových). Na druhou stranu může být tento popis vhodný u grafů, které se blíží grafům úplným (tedy těm, ve kterých je každý vrchol spojen se všemi ostatními). Stejně neúsporná je i matice incidenční, která v každém sloupci obsahuje jen 2 nenulové prvky.

Paměťové nároky matic pro popis grafu můžeme snížit, když si uvědomíme, že matice sousednosti prostého grafu obsahuje pouze hodnoty 1 a 0. Stejně tak matici incidence lze rozdělit na dvě matice (vstupní a výstupní), které budou rovněž dvouhodnotové. S výhodou lze následně použít binární popis. V tomto případě ovšem za tuto úsporu zaplatíme zvýšenými výpočetními nároky vzniklými převodem z binárního tvaru.

2.3.2 Popis využívající seznamy

Popisy grafu využívající seznamy jsou ve většině případů pro programové zpracování nejvýhodnější. Zde se využívá zejména statických datových struktur typu pole a dynamických struktur spojový seznam. K vyložení obecných principů uložení nám tyto struktury postačí. Moderní programovací jazyky jako C# nicméně mají implementovány i pokročilejší datové struktury jako jsou kolekce, které jsou využitelné pro zpracování grafu. Mezi ně patří v jazyce C# například struktury *Hashtable* nebo *Arraylist*.

Seznam hran lze jednoduše realizovat využitím dvou jednorozměrných polí. Jeden index obou polí reprezentuje jednu hranu. V prvním poli (nazvěme jej PPV jako “pole počátečních vrcholů“) jsou uloženy indexy počátečních vrcholů jednotlivých hran. Ve druhém (nazvaném PKV) jsou analogicky uloženy indexy koncových vrcholů. Pro urychlení přístupu ke všem hranám daného vrcholu je výhodné zavést třetí pole (nazvěme PDH), které bude obsahovat index další hrany, které vychází ze stejného vrcholu jako hrana daná indexem pole. Pokud taková hrana neexistuje, obsahuje například číslo -1.

Příklad: Datová realizace seznamu hran pro graf na obrázku 2.2

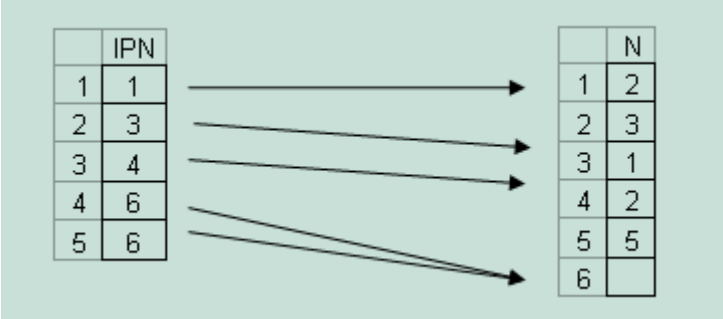
	PPV
1	1
2	2
3	1
4	4
5	4

	PKV
1	2
2	1
3	3
4	2
5	5

	PDH
1	3
2	-1
3	-1
4	5
5	-1

Seznam následníků pro svou realizaci využívá dvou polí. V prvním poli (nazveme IPN jako “index prvního následníka“) jehož indexy představují jednotlivé vrcholy jsou uloženy ukazatele místa ve druhém poli (označíme N jako “následníci”), kde začíná seznam následníků daného vrcholu.Pro lepší pochopení opět uvádím konkrétní realizaci pro vzorový graf.

Příklad: Datová realizace seznamu následníku pro graf na obrázku 2.2



Index konce seznamu následníků pro určitý vrchol se pozná podle indexu prvního následníka dalšího vrcholu. Tento způsob uložení je velmi úsporný a výhodný v případech, kdy potřebujeme rychle vyhledat následníky daného vrcholu. Pro hledání předchůdců i pro některé další úlohy zpracování grafů však může být dost nepraktický.

Jinou možností realizace seznamů následníků je využití spojových seznamů. První pole IPN je v podstatě zachováno. Jeho prvky jsou v tomto případě výchozími body spojových seznamů hran vycházejících z daného vrcholu. Výhodou je snazší manipulace s daty při úpravách grafu.

2.4 Reprezentace grafu v programu

Program se zabývá zejména zobrazením grafu v prostoru. S ohledem na to je vytvořena i datová struktura jeho popisu. Pro uchování informací o grafu je využito zejména datových kolekcí. V krátkosti se tedy nejdříve zmíním o hlavních vlastnostech těchto tříd.

2.4.1 Datové kolekce v jazyce C#

Datové kolekce jsou v jazyce C# často používanými třídami pro ukládání dat. Jedná o tzv. beztypové kolekce v prostoru jmen `System.Collections`. Přívlastkem beztypové je myšlen fakt, že uložená data jsou instance tříd odvozených od typu `System.Object`. Lze do nich tedy uložit téměř vše,ale při zpětném vyjmutí získáme pouze odkaz na typ `Object`. Je tedy důležité držet si přehled o tom, s jakými daty pracujeme. Při vyjímání je nutné provést explicitní konverzi na správný tvar. Výhodou kolekcí je fakt, že obsahují řadu předdefinovaných metod např. pro třídění, vyhledávání, vytváření kopii, které činí práci s nimi poměrně intuitivní a výsledný kód přehlednější.

Zmíním se nyní podrobněji o dvou kolekcích, které využívám ve svém programu.

`ArrayList` je asi nejpoužívanější sekvenční kolekcí. Jedná se de facto o dynamicky alokované pole, se kterým můžeme zacházet jako se seznamem nebo jako s polem (tedy pomocí indexů). Pokud vytváříme instanci této kolekce konstruktorem bez parametrů, alokuje se paměť pro 16 prvků. Pokud je tento překročen, je poněkud neefektivním způsobem tato kapacita dvojnásobena a stejný postup se opakuje. Proto je lepší, pokud známe předem přibližně potřebný počet prvků, volat konstruktor s parametrem určující počáteční rozsah alokované paměti.

`Hashtable` je výhodná pokud potřebujeme asociativní kolekci., do níž budeme ukládat dvojici (*klíč,hodnota*). K prvkům této kolekce pak přistupujeme pomocí indexeru, kde jako index nám poslouží *klíč*.

Více o možnostech a použití datových kolekcí v lze najít např. v [3].

2.4.2 Vlastní reprezentace grafu

Zopakujme, že graf tvoří trojice G=(V, E , ε). Uveďme, jak je realizován popis jednotlivých množin v programu.

Reprezentace vrcholů. Prvky množiny V vrcholů grafu popisují v programu 2 entity. První z nich je třída **`Uzel_coor`**. Ta obsahuje 3 atributy typu `float` popisující souřadnice vrcholu v prostoru. Druhou je pak třída **`Uzel_desc`** , která obsahuje další informace popisující daný uzel. Například zde najdeme atributy typu `string` pro uložení názvu a popisu uzlu, atribut typu `bool` určující, zda je uzel stabilní (tedy jestli je incidentní s hranou typu smyčka). Důvodem pro použití dvou entit místo jedné je fakt, že v programu lze provádět prostorové transformace, které mění souřadnice vrcholů, zatímco ostatní informace o vrcholu zůstávají stejné. Díky tomu je úspornější ukládat souřadnice zvlášť.

Reprezentace hran. Popis prvků množiny E hran grafu má v programu na starosti třída **`Hrana`**. Obsahuje dvojici atributů typu `string`, ve kterých je uloženy identifikátory počátečního a koncového vrcholu hrany. Kromě nich obsahuje tato třída celočíselný atribut definující typ hrany. Tento atribut obsahuje informaci o tom, zda se jedná o jedinou hranu incidentní s svými vrcholy, nebo zda se grafu vyskytuje i hrana opačná. Třetí možnost je ta, že je daná hrana smyčkou. Informace o typu hrany je důležitá při vykreslování grafu a zjišťuje se při načítání grafu.. Pokud by program tyto informace o hraně při vykreslovaní neměl, musel by je pokaždé znovu zjišťovat, což by zbytečně zpomalovalo rendering scény.

Reprezentace incidenčního zobrazení a celého grafu. Veškeré informace o daném grafu v sobě má uloženy třída **`Graf`**. Obsahuje několik dynamických datových kolekcí typu `ArrayList` a typu `Hashtable`.

`Hashtable` **`u_table`**:

Obsahuje seznam dvojic kde klíč představuje proměnná typu `string` určující název vrcholu a přiřazenou hodnotou tomuto klíči je instance třídy `Uzel_desc`, ve které je popis vrcholu.

`Hashtable` **`u_coord`**

Tato datová kolekce je jako předchozím případě indexována názvem vrcholu, k němuž je přiřazenou hodnotou instance třídy `Uzel_coor`, kde jsou uloženy souřadnice daného vrcholu v prostoru.

`ArrayList` **`h_Blist`**`,ArrayList` **`h_Elist`**

Dvojice těchto kolekcí zajišťuje de facto principielní základ popisu grafu, neboť přiřazují číslu (indexu) hrany její incidenční vrcholy. Přesněji kolekce `h_Blist` obsahuje názvy počátečních vrcholů jednotlivých hran a analogicky kolekce `h_Blist` názvy vrcholů koncových.

`ArrayList` **`h_Dlist`**

Kolekce `h_Dlist` je seznamem, přiřazujícím indexu hrany instanci třídy Hrana. Ta, jak jsme dříve uvedli, obsahuje dvojici názvů incidenčních vrcholů této hrany a navíc její typ.

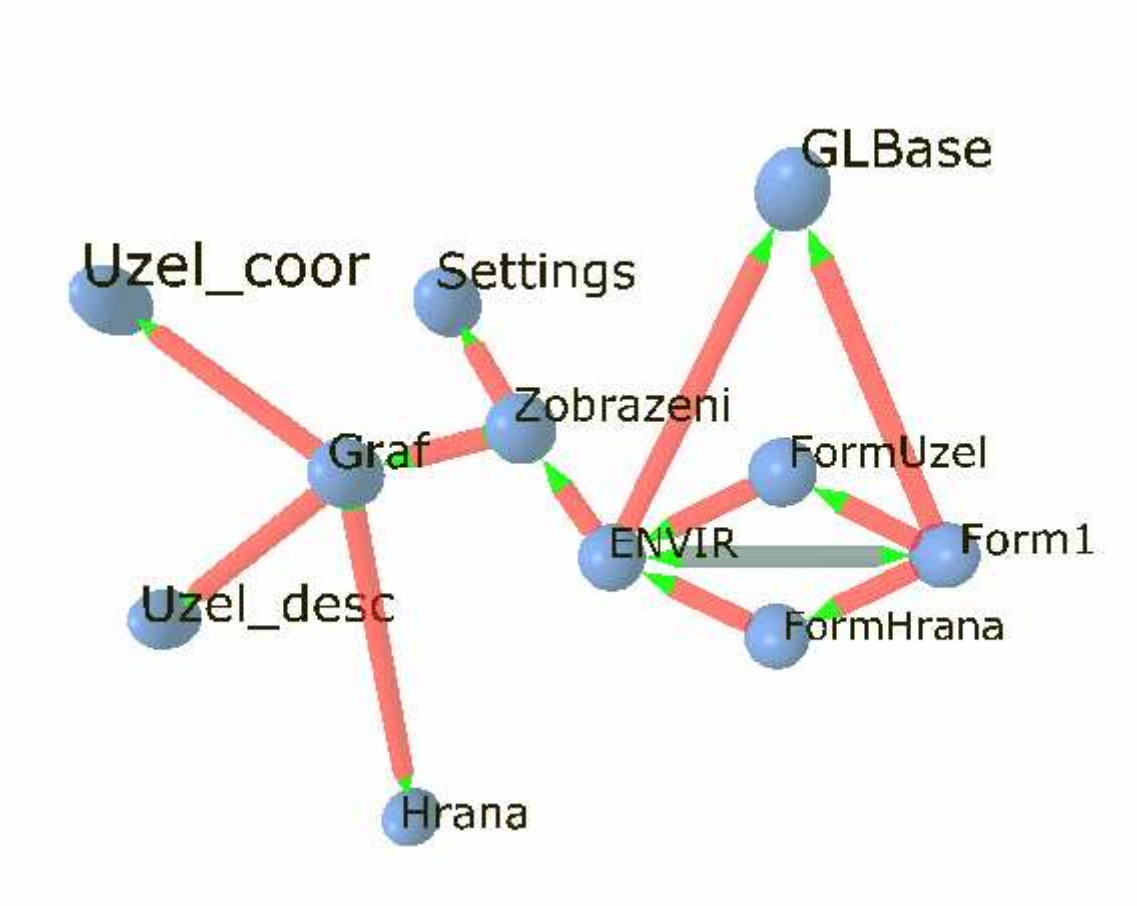
Dalo by se říct, že informace obsažená v kolekci `h_Dlist` je do určité míry redundantní s informací v předchozích dvou kolekcí. Nicméně existence všech těchto kolekcí najednou má svůj význam, neboť umožňuje v určitých případech rychlejší vyhledávání a lepší čitelnost kódu.

3. Aplikace GrafMan3D

V této kapitole se již budu věnovat popisu konkrétních aspektů hlavního produktu své činnosti související s tímto pojednáním, a to aplikaci pro prostorové zobrazení orientovaných grafů. Tuto aplikaci jsem nazval GrafMan3D. V jednotlivých oddílech této kapitoly se zastavím a rozeberu zejména strukturu kódu programu, tedy jeho rozdělení do tříd a popis jejich funkce. Dále se zaměřím na výčet vlastních možnosti aplikace a na její vstupy a výstupy. V závěru kapitoly se pak budu věnovat možnostem grafického rozhraní programu a jeho ovládání.

3.1 Struktura kódu programu a postup vývoje

Zdrojový kód celé aplikace je rozdělen do řady souborů. Ty většinou obsahují definici 1 stejnojmenné třídy. V několika případech, kdy spolu jednotlivé třídy úzce souvisejí a nejsou co do počtu řádků nijak rozměrné, jsem se rozhodl sloučit definice více tříd do jednoho souboru. Třídy v této aplikace lze zhruba rozdělit do několika kategorii podle základní funkce, kterou v programu a jeho struktuře vykonávají. Následující graf (vytvořen v samotné aplikaci) zhruba nastiňuje vzájemné vazby nejdůležitějších tříd. Jejich funkci osvětlíme v dalším textu.



Obrázek 3.1: Hierarchie programu – obraz vzájemných vazeb nejdůležitějších tříd

3.1.1 Rozčlenění tříd do kategorii podle funkce v programu

První kategorii jsou třídy zajišťující spojení s rozhraním OpenGL.. Patří sem třídy patřící OpenGL wrapperu, které připojují funkce 3D grafického rozhraní. O těch již byla zmínka v oddíle 1.2.3.

Další kategorii jsou třídy realizující grafický výstup. Sem patří samozřejmě třída `Form1`, která obstarává obsluhu hlavního okna běžící aplikace, a jí vlastněné instance tříd `FormUzel` a `FormHrana`. Kromě nich do této kategorie spadá třída `GLBase`, která využívá členů tříd wrapperu k vykreslení vlastní programem definované 3D scény.

Do třetí kategorie spadají třídy, jejichž primárním určením je reprezentovat datový popis nějakého objektu, se kterým v programu pracujeme. Jelikož se jedná program zobrazující grafy, budou zřejmě mezi takové objekty patřit právě graf a jeho součásti. Patří sem tedy třídy `Graf`, `Hrana`, `Uzel_desc`, `Uzel_coor`. Aplikace zvládne uchovat a vyvolávat více grafů a náhledů na ně. Proto sem patří i třída `Sets`, která má za úkol uchovat jistá nastavení k danému náhledu.

Třída `Zobrazeni pak` zastřešuje soubor veškerých dat souvisejících s daným náhledem.

Poslední kategorii budou zastupovat statická třída `ENVIR` obsahující globální data přístupná z každého místa programu a třída `Operations` obsahující funkce pro souřadnicové transformace čili algoritmy pro rozmístění vrcholů.

3.1.2 Postup vývoje programu

Pořadí v jakém jsem uvedl tyto kategorie není náhodné nýbrž zhruba odpovídá postupu ,který jsem zvolil při vývoji. V první fázi jsem řešil zobrazení 3D scény na plochu okna. Jakmile jsem byl schopen zobrazit reprezentaci vrcholu a hrany zabýval jsem se řešením datového modelu grafu tak,aby byl vhodný pro zobrazení a zpracování. Nakonec jsem řešil implementaci interaktivních uprav tohoto zobrazení.

3.2 Popis funkce jednotlivých tříd

Nyní se zaměřím na popis klíčových tříd. Z důvodů úspornosti a přehlednosti vypíši jen hlavičky jednotlivých členských metod, u vybraných případně naznačím důležité pasáže kódu. .Pro lepší přehlednost také vynechám modifikátory přístupu, které nehrají zásadní význam.

Třídy `Hrana, Uzel_desc, Uzel_coor`

```
class Hrana
{
     string sU1,sU2
     int typ

     Hrana()
     void Reset()
     void CopyTo(Hrana h)
}
```

```
class Uzel_desc
{
     int index
     string nazev
     string popis
     bool stable
          int komponenta

     Uzel_desc()
     void Reset(0
     void CopyTo(Uzel_desc u)
}

class Uzel_coor
{
     float x,y,z

     Uzel_coor()
     void Reset()
     void CopyTo(Uzel_coor u)
}
```

Jedná se o jednoduché třídy pro popis hrany resp. vrcholu grafu, jak jsme je zmínil v oddíle 2.4.2. Zde je snad zajímavá jen metoda `CopyTo,` která ve všech třech případech obstarává kopii dat do jiné instance téže třídy. Smysl této metody ještě zmíním při popisu třídy Graf, která je těmto třem třídám hierarchicky nadřazená.

Třída `Graf`

```
class Graf
{
     int pocetUzlu
     int tez_X;
     int tez_Y;
     int tez_Z;


     Hashtable u_table
     Hashtable u_coord
     ArrayList h_Blist
     ArrayList h_Elist
     ArrayList h_Dlist

     Graf()
     void Reset()
     Graf  Clone()
     void vypis()
     ArrayList GetNasled(string u)
     int GetHranaIndex(string uzel1, string uzel2)
```

```
        void AddHrana(Hrana hrana)
        void AddUzel(Uzel_coor coor,Uzel_desc desc)
        float GetMinVzdal()
        void LoadFile(System.IO.StreamReader sr)
        void SaveFile(System.IO.StreamWriter sw)

        void CopyTo(Hashtable Nu_table,Hashtable Nu_coord)
        void CopyTo(ArrayList Nh_Dlist, ArrayList Nh_Blist,
        ArrayList Nh_Elist)

    }
```

O dynamických datových kolekcích jako atributech této třídy již byla řeč v oddíle 2.4.1. Metoda `Clone` složí ke kopii obsahu celého datového obsahu dané instance třídy na jiné místo v paměti. V této metodě jsou využity pomocné metody `u_CopyTo` a `h_CopyTo`, které provádí tvrdou kopii dat z kolekcí uzlů a hran. Ty využívají obdobné metody pro každý svůj člen. Nikde se proto nepřekopíruje jen odkaz na hodnotu, vždy a pro každý prvek jen hodnota sama.

Metoda GetNasled vrací seznam následníků daného uzlu. Metoda GetMinVzdal slouží k určení minimální vzdálenosti uzlů v grafu, což je potřebné pro normalizaci zobrazení daného grafu. Atributy tez_X .. Z slouží k uchování těžiště, které používám jako střed rotace grafu. Tyto hodnoty jsou upravovány automaticky při každém přidání vrcholu nebo úpravě jeho souřadnic.

Třída `Sets`

```
class Sets
{
        float cam_tx, cam_ty, cam_ty
        float cam_alfa, cam_beta, cam_gama
        float obj_tx, obj_ty, obj_tz
        float uzelR
        float hranaR

        Sets()
        Reset()
        Sets Clone()
}
```

Třída slouží zejména k uchovaní nastavení pohledu na daný graf v aktuálním náhledu. Atributy uzelR a hranaR určují rozměry objektů symbolizujících vrchol (koule) a hrany (útvar složený z jednoho či dvou jehlanů a válce o stejné základně).

Třída `Zobrazeni`

```
class Zobrazeni
{
     Graf graf
     Sets settings
     string AktUzel
     string NextUzel;
     int AktHrana;
     string fmode;

     Zobrazeni()
     Zobrazeni Clone()
     void Reset()
     void Normalize()
     void SetOriginal()
     int Focus(string s)
}
```

Metodou, jejíž funkce nemusí být jasná podle názvu, je metoda `Normalize`, která upravuje rozměry prvků grafu podle jejich rozmístění v prostoru a metoda `SetOriginal`, která do daného zobrazí ukládá graf naposledy vložený ze souboru. Metodu `Focus` využívá vykreslovací rutina pro zjištění bodu, do kterého má posunout střed souřadnic.

Tato třída tedy jistým způsobem zastřešuje všechny předešlé třídy. Zjednodušeně lze říci, že instance třídy obsahuje pro zobrazení daného grafu v daném náhledu všechna data, kterými se mohou jednotlivé náhledy lišit. Uchovává se tu aktuální stav zobrazení, jako např. aktivní vrchol a hrana. Tyto informace jsou pak použity při vykreslení pro volbu barvy objektu. Jediná data, která zde nenajdeme jsou globální nastavení pro všechny náhledy. Ty najdeme ve třídě ENVIR.

Třída `ENVIR`

```
class ENVIR
{
     static int Akt_nahled
     static bool zobraz_popis;
     static bool zobraz_info;
     static int b_schema;

     static Graf Orig_Graf
     static Zobrazeni[] Z_list

     static ENVIR()
     void Reset()
```

```
        static void Load()
        static void LoadInit()
        static void ZapsatNahled(int i)
        static Zobrazeni GetAktZobr()
        static void KeyApply(int kv, string kd)

    }
```

Tato třída je statická, neboť je myšlena jako úložiště globálních dat a nastavení, na která se odkazují zejména metody třídy `Form1`. Obsahuje i jisté obslužné metody zejména pro práci s celými instancemi třídy `Zobrazeni.` Aspekty jejího použití jsou v jistých případech z hlediska striktně objektového pohledu možná trochu ideologicky nesprávné, nicméně plní svůj účel .

Nejdůležitějšími členy jsou zde Z_list, což je pole instancí třídy `Zobrazeni`, kde jsou data pro jednotlivé náhledy. Číslo uložené v atributu `Akt_náhled` označuje index tohoto pole, který slouží pro aktuální zobrazení, tedy s posunutím o jedničku odpovídá v běžící aplikaci číslu aktuálního náhledu.

3.3 Vytvoření 3D scény - třída `GLBase`

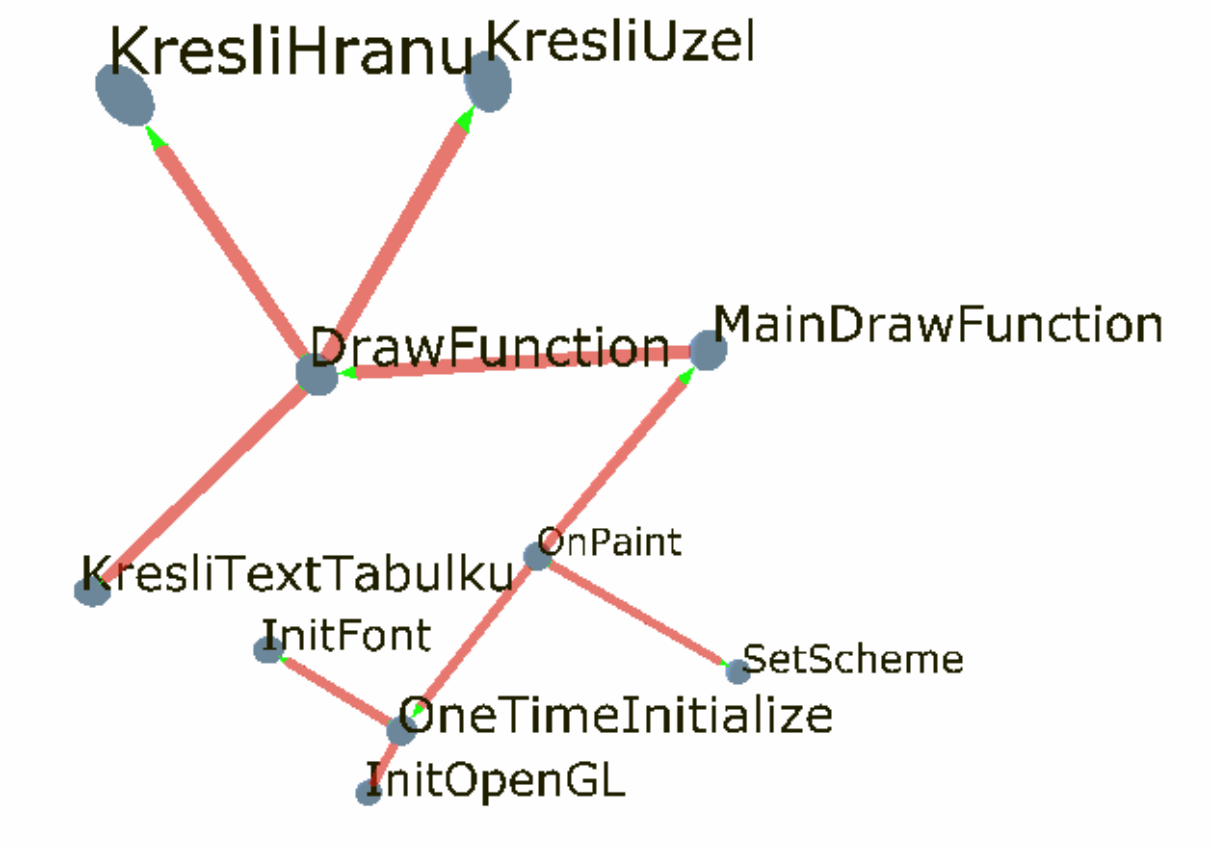
Tato třída je stěžejní částí programu neboť právě metody této třídy provádějí vykreslení 3D scény s prostorově zobrazeným grafem.

Při tvorbě této třídy jsem vycházel z třídy `DEMO`, obsahující demonstrační přiklad použití wrapperu. Učinil jsem tak z důvodů, že obsahuje kromě konkrétních vykreslovacích rutin i množství důležitého inicializačního kódu, který sám autor wrapperu doporučuje při experimentování ponechat.

Postupně jsem z kódu třídy eliminoval nerelevantní metody a ponechal jen ty důležité pro prvotní nastavení prostředí. K nim jsem potom přidal vlastní vykreslovací metody zajišťující veškeré vykreslení potřebných elementů 3D scény.

3.3.1 Hierarchická struktura metod třídy

K pochopení významu jednotlivých dílčích metod třídy by měl pomoci pohled na následující graf (vytvořený pomocí samotné aplikace), který hierarchickou strukturu třídy. Hrana mezi uzly zde označuje, že metoda volá metodu jinou.



Obrázek 3.2: Znázornění hierarchické struktury metod třídy `GLBase`

3.3.2 Postup vykreslení 3D scény ve třídě `GLBase`

Vstupním místem vykreslovacího procesu je metoda `OnPaint`. Ta při prvním spuštění iniciuje spuštění metody `OneTimeInicialize`. Ta zastřešuje cely proces počátečního nastavení prostředí OpenGL a vytvoření fontů. `OnPaint` dále volá metodu `SetScheme`, kterou jsem vytvořil za účelem přehledného nastavení všech barev a materiálů vyskytujících se ve vykreslované scéně v závislosti na zvoleném barevném schématu. Konečně poslední rutinou volanou z metody `OnPaint` je `MainDrawFunction`, kde začíná samotný vykreslovací proces.

`MainDrawFunction` nastavuje stavové proměnné vykreslovacího stroje (vyhlazování, osvětlení, projekci) a předá řízení metodě `DrawFunction`, která již volá rutiny pro zobrazení jednotlivých objektů na scéně. Konkrétně to jsou metody `KresliTextTabulku` a `KresliGraf`.

3.3.3 Umístění a rozměry objektů

Vrcholy a hrany nejsou umísťovány v prostoru přímo na souřadnice jim příslušející. Všechny jsou posunuty tak, aby se v průsečíku souřadnic nacházel bod, kolem kterého pak můžeme graf na obrazovce rotovat..Tímto bodem je těžiště grafu, pokud je program v režimu sledování grafu jako celku. Pokud je uživatelem nastaveno sledování aktivního vrcholu, pak jsou souřadnice relevantního bodu dány souřadnicemi tohoto vrcholu.

Rozměry objektů jsou programem dynamicky měněny. Hodnotou od, které jsou odvozeny je nejmenší vzdálenost mezi dvěma vrcholy grafu. Tím pak lze zajistit, že každá hrana v grafu bude viditelná, a dva vrcholy se nebudou překrývat. Jelikož jsou souřadnice zaokrouhlovány na celá čísla, je vhodné pro zajištění správné funkce při zadávání grafu volit určitou minimální vzdálenost vrcholů. Doporučenou hodnotou je zhruba 10.

3.3.4 Vykreslení vrcholů grafu

Vykreslení grafu v metodě `KresliGraf` probíhá následujícím způsobem. Pro aktuálně nastavený graf pro každý prvek kolekcí `u_coord` a `u_table` aktuálního grafu se předají jejich odkazy metodě `KresliUzel`. Ta nejdříve přepočítá souřadnice a zvolí materiál podle toho, zda se jedná o aktivní vrchol. Na souřadnice umístí kouli pomocí funkce *gluSphere*. Pokud se jedná o vrchol stabilní, umístí sem prstenec představující smyčku. Pro ní se zvolí odlišný materiál podle toho, zda odpovídá uživatelem nastavené aktivní hraně. Na odvozené souřadnice dále umístí text názvu uzlu. Nápis je rovněž polygonovým objektem a je třeba dosáhnout toho, aby byl natočen čelem k pozorovateli. Tento problém jsem vyřešil tak, že jsem pro transformační matici MODELVIEW použil úhly opačné , než v matici PROJECTION. Více o transformačních maticích OpenGL viz [4].

3.3.5 Vykreslení hran grafu

V souvislosti s problémem vykreslení hran používám k identifikaci jejich typu číselné značení. Typ 0 odpovídá smyčce. Ty se vykreslují v rámci rutiny `KresliUzel`. Typ 1 odpovídá takové hraně, ke které v grafu neexistuje hrana opačná (čili s prohozenými incidenčními vrcholy). Typ 2 odpovídá hranám, ke kterým v grafu existuje hrana opačná. Jelikož obě opačné hrany znázorňuje jeden zobrazovaný objekt, je jedna z nich značena číslem 2 a druhá z nich číslem -2.

V metodě `KresliGraf` se prochází všechny prvky kolekce `h_Dlist` aktuálně nastaveného grafu, které popisují hrany typu 1 nebo 2. Jejich odkazy jsou předávány metodě `KresliHranu`. Zde probíhá přepočet souřadnic incidentních vrcholů, směrových úhlů hrany.a vzdálenosti vrcholů (tedy délky hrany). Vlastní objekt znázorňující hranu je tvořen válcem a dvěma jehlany. V případě hrany typu 1 je výška jehlanu u počátečního vrcholu nulová. Vykreslení jehlanů i válce obstarává funkce *gluCylinder* knihovny GLU. Materiál válce je volen podle typu hrany a podle toho, zda je hrana totožná s uživatelem nastavenou hranou aktivní.

3.4 Možnosti aplikace

Náplní této podkapitoly by měl být výčet dosavadních možností, které program nabízí. Dosavadních proto, že na jeho vylepšování mám v plánu pokračovat i po odevzdání této práce. Jelikož je jeho primárním cílem zobrazení grafu v prostoru, je důležité zmínit, jaké jsou způsoby vstupu grafu do aplikace,jakými způsoby jej lze v rámci aplikace upravit, čí způsob uložení do souboru.

3.4.1 Vstupy a výstupy

Jedním způsobem vstupu grafu do aplikace je načtení ze souboru. Program očekává následující formát textového souboru:

- první řádek obsahující řetězec „**[Graf]**“,
- sekce popisující jednotlivé entity (vrcholy resp. hrany) začínají řádky s řetězci „**[Uzel]**“ resp. „**[Hrana]**“ – dále je budu nazývat počáteční tagy,
- sekce končí koncovými tagy „**[/Uzel]**“ a „**[/Hrana]**“,
- definice atributů jednotlivých prvků grafu mají podobu řádku ve tvaru: „<název atributu>=<hodnota atributu>“,
- v sekci popisu vrcholu načítací algoritmus akceptuje atributy „**nazev**“, „**popis**“, „**x**“, „**y**“, „**z**“,

- v sekci popisu hrany jsou akceptovány atributy „**pv**“ a „**kv**“, za kterými by měly být uvedeny názvy incidenčních vrcholů hrany ,
- načítání končí tagem „**[/Graf]** “.

Dalším způsobem vstupu je přímé vytvoření grafu v programu. Grafické rozhraní programu nabízí funkce pro vytvoření a editaci hran a vrcholů velmi intuitivní a rychlou cestou.

Výstupy, které program nabízí jsou zobrazení grafu na obrazovce nebo jeho uložení do textového souboru, který obsahuje seznam vrcholů s jejich souřadnicemi a seznam hran. Jeho struktura odpovídá výše uvedenému formátu vstupního souboru.

3.4.2 Výčet schopností aplikace

Mezi důležité funkce programu kromě přidání nových uzlů a hran do prázdného grafu nebo grafu načteného ze souboru patří možnosti interaktivní editace. Uživatel má možnost vybrat vrchol ze seznamu vrcholů zobrazeného grafu, který budeme nazývat aktivní. Ten se zobrazí zvýrazněně a zpřístupní se možnost editace jeho atributů. Aktivní vrchol lze rovněž smazat, načež aplikace sama zajistí smazaní jeho incidenčních hran. Po výběru aktivního vrcholu se automaticky aktualizuje seznam následníků tohoto vrcholů., čímž se hrana incidenční s aktivním vrcholem a jeho následníkem aktivuje a zvýrazní. Takto aktivovanou hran lze smazat nebo uskutečnit přechod jí symbolizovaný a přejít do nového aktivního stavu, jinak řečeno aktivovat další vrchol.

Kromě přímé editace jednotlivých prvků grafu lze na graf jako celek aplikovat několik transformačních algoritmů. V současné době jsou k dispozici algoritmus pro rozmístění vrcholů na kružnici, plánovaný algoritmus pro kondenzaci grafu ještě není implementován.

Aplikace obsahuje podporu pro pohodlné natáčení a posouvání grafu v náhledu z pomocí pohybu myši. Lze také přepnout do módu kdy lze myší přesouvat jednotlivé vrcholy.

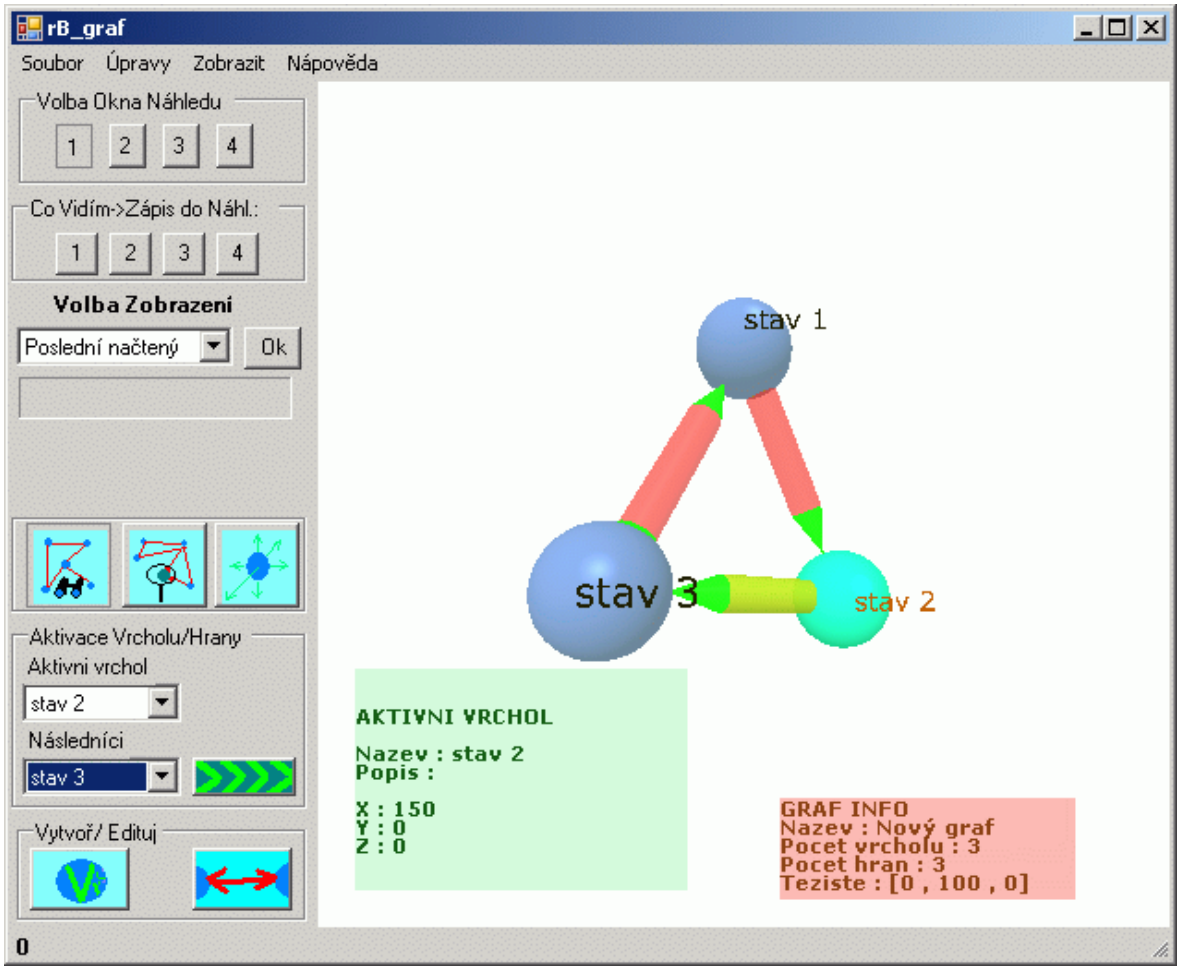
Pro výstupní zobrazení jsou k dispozici dvě barevná schémata. Schéma s černou podkladovou barvou je kontrastnější a výhodné pro prezentaci na počítači, zatímco schéma využívající bílý podklad je vhodné pro tisk.

3.5 Uživatelské rozhraní aplikace

Grafické rozhraní aplikace je poměrně jednoduché a přehledné. Jeho primární součástí je hlavní okno, z něhož lze vyvolat několik vedlejších dialogů. Popišme si nejdřív komponenty hlavního okna.

3.5.1 Hlavní okno

Následující obrázek zachycuje hlavní okno aplikace.

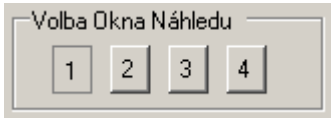


Obrázek 3.3: Ukázka hlavního okna aplikace se zobrazením jednoduchého automatu

Formulář aplikace se skládá ze tří hlavních částí. V horní části je standardní hlavní menu s položkami “Soubor“, “Úpravy“, “Zobrazit“ a “Nápověda“. V levé části formuláře je umístěn panel obsahující řadu ovládacích prvků. V pravé části je pak panel, na který se vykresluje 3D scéna a informativní tabulky.

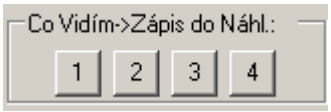
Hlavní menu obsahuje v nabídce “Soubor“ klasickou sadu funkcí pro načtení grafu ze souboru, jeho uložení do souboru a položku pro ukončení aplikace. V dalších dvou nabídkách “Üpravy“ a “Zobrazit“ lze nalézt například funkce pro smazání grafu v aktuálním náhledu, nastavení kamery do počáteční pozice, přepínače zobrazení popisu vrcholů a informačních tabulek. V neposlední řadě zde můžeme přepínat mezi barevnými schématy 3D zobrazení.

P**anel** v levé časti formuláře obsahuje hlavní ovládací prvky pro manipulaci s grafy. Pomocí první sady tlačítek (obr 3.4) může uživatel přepínat tzv. náhledy. Nejedná se vlastně o nic jiného než o čtyři nezávislé pracovní plochy, díky nimž lze pracovat s více grafy najednou.



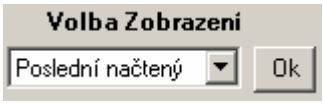
obr. 3.4 – Ovládací prvek pro přepínání pracovních ploch

Kopírovat obsahy náhledů mezi sebou umožňuje druhá sada tlačítek. Pokud je například aktivní první náhled a uživatel stiskne tlačítko pro zápis do 3.náhledu, zapíše se do něj obsah 1. náhledu a navíc se cílový náhled aktivuje.



obr. 3.5 – Ovládací prvek pro kopírování obsahu náhledu

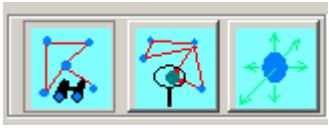
Roletová nabídka na obr. 3.6 umožňuje zobrazit poslední ze souboru načtený graf a zpřístupňuje funkce změny rozvržení grafu v prostoru. Mám v plánu sem zahrnout například vytvoření kondenzace grafu. V aktuální verzi programu však funkce této nabídky nejsou plně implementovány.



obr 3.6 – Roletová nabídka s některými pokročilými funkcemi

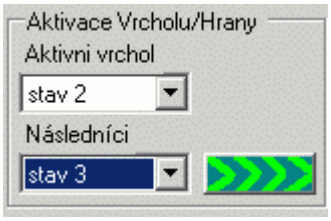
Dostáváme se k přepínacímu panel s grafickými tlačítky. Vyobrazení na nich snad obstojně naznačují jejich funkci, ale pro jistotu je v krátkosti objasním. Tlačítko vlevo zapíná zobrazovací mód ,kdy se kamera snaží sledovat graf jako celek. Pokud v tomto módu budeme stiskem pravého tlačítka myši a současně jejím pohybem graf rotovat, bude středem

této rotace bod v prostoru spočtený jako těžiště zobrazeného grafu. Druhé tlačítko se naopak snaží sledovat uživatelem vybraný aktivní vrchol. Třetí tlačítko slouží pro interaktivní přemisťování vrcholu pomocí myši.



obr 3.7 – Prvek pro přepínaní módu zobrazení a funkce myši

Blok nazvaný “Aktivace Vrcholu/Hrany“ obstarává velmi důležitou funkci programu. Slouží k výběru aktivního vrcholu. Po uskutečnění tohoto výběru se tento vrchol zobrazí odlišnou barvou a ve spodní roletové nabídce se vygeneruje seznam jeho následníků. Pokud uživatel některý z nich vybere, dojde k výběru aktivní hrany, která se ve 3D zobrazení odliší od ostatních jinou barvou. Zeleným tlačítkem lze danou hranu “odpálit“, neboli uskutečnit přechod automatu z jednoho stavu do jiného. Lze tak primitivním, leč názorným způsobem chování automatu simulovat.



obr 3.8 – Ovladač primitivní simulace chování stavového automatu

Následující oddíl obsahuje dvě tlačítka, jejichž grafický popis má evokovat dvojici základních prvků grafu, čili vrchol a hranu. Jen pomocí těchto dvou tlačítek a dialogových oken jimi aktivovaných, lze velmi rychle a intuitivně vytvořit graf a vhodně jej rozvrhnout v prostoru. Chování tlačítek ovlivňuje fakt, zda je vrchol nebo hrana aktivován. Pokud není, slouží tlačítka pro přidání objektu. Pokud objekt aktivován je, slouží k jeho editaci či vymazání.

obr. 3.9 – Tlačítka pro editaci grafu

Pravý panel, do něhož je vykreslována 3D scéna je rovněž interaktivní. Současným přidržením tlačítka myši a jejím pohybem po panelu lze vrchol nebo celý graf rotovat či posouvat. O tom jak bude scéna reagovat rozhoduje volba módu zobrazení na panelu viz obr. 3.7. K ovládání náhledu je možné použít i některé klávesy. Jejich seznam obsahuje Nápověda k programu.

Závěr

Jako podklad pro svou práci jsem získal program Planar vytvořený jako součást diplomové práce ing. Haščákem. Ten byl naprogramován s cílem prezentovat algoritmy pro rovinné zobrazení grafů. Autor tento program napsal v jazyce C++.

Před prací na tvorbě programu jsem si dal za cíl, vytvořit aplikaci, která nebude v žádném aspektu kopírovat funkci programu Planar, nýbrž ho bude svými schopnostmi spíše doplňovat. Při tvorbě kódu a grafických prvků jsem tedy spoléhal jen na svou fantasii, programátorské zkušenosti a podporu odborné literatury. Nicméně je nutno říci, že například při tvorbě formátu pro uložení grafu do souboru jsem se řešením ing. Haščaka inspiroval.

V průběhu prací na vývoji aplikace, která by splňovala požadavky kladené na ni v úvodu textu, jsem řešil několik dílčích problémů.

V rozsahu nutném pro pochopení problému jsem prostudoval základy teorie grafů a navrhnul vhodnou datovou strukturu pro uchování a zpracování grafu v programu.

Dále jsem prozkoumal možnosti implementace podpory rozhraní OpenGL do jazyka C#. Funkce tohoto rozhraní jsem následně použil pro návrh a zobrazení objektů, které by vhodně zpodobňovaly prvky orientovaného grafu.

Spojením těchto dvou výše uvedených výsledků jsem získal schopnost vykreslení orientovaného grafu v prostoru, což je hlavní funkce vytvořené aplikace.

V závěrečné fázi jsem za pomoci intenzivního využití možností vizuálního návrhu v prostředí Microsoft Visual Studio .NET pracoval na rozšíření schopností aplikace tak, aby konečným výstupem mého snažení byla skutečně použitelná aplikace pro návrh a zobrazení a úpravy prostorových orientovaných grafů.

Na vývoji této aplikace bych rád pokračoval i nadále. Minimálně mnohé jeho součásti bych také rád využil při vývoji aplikace nové, která by měla za cíl modelování Petřino sítí opět za pomoci rozhraní OpenGL.

Použitá literatura

[1] Demel J.: Grafy a jejich aplikace, Academia, 2002

[2] Virius M.: Od C++ k C#, Kopp, 2002

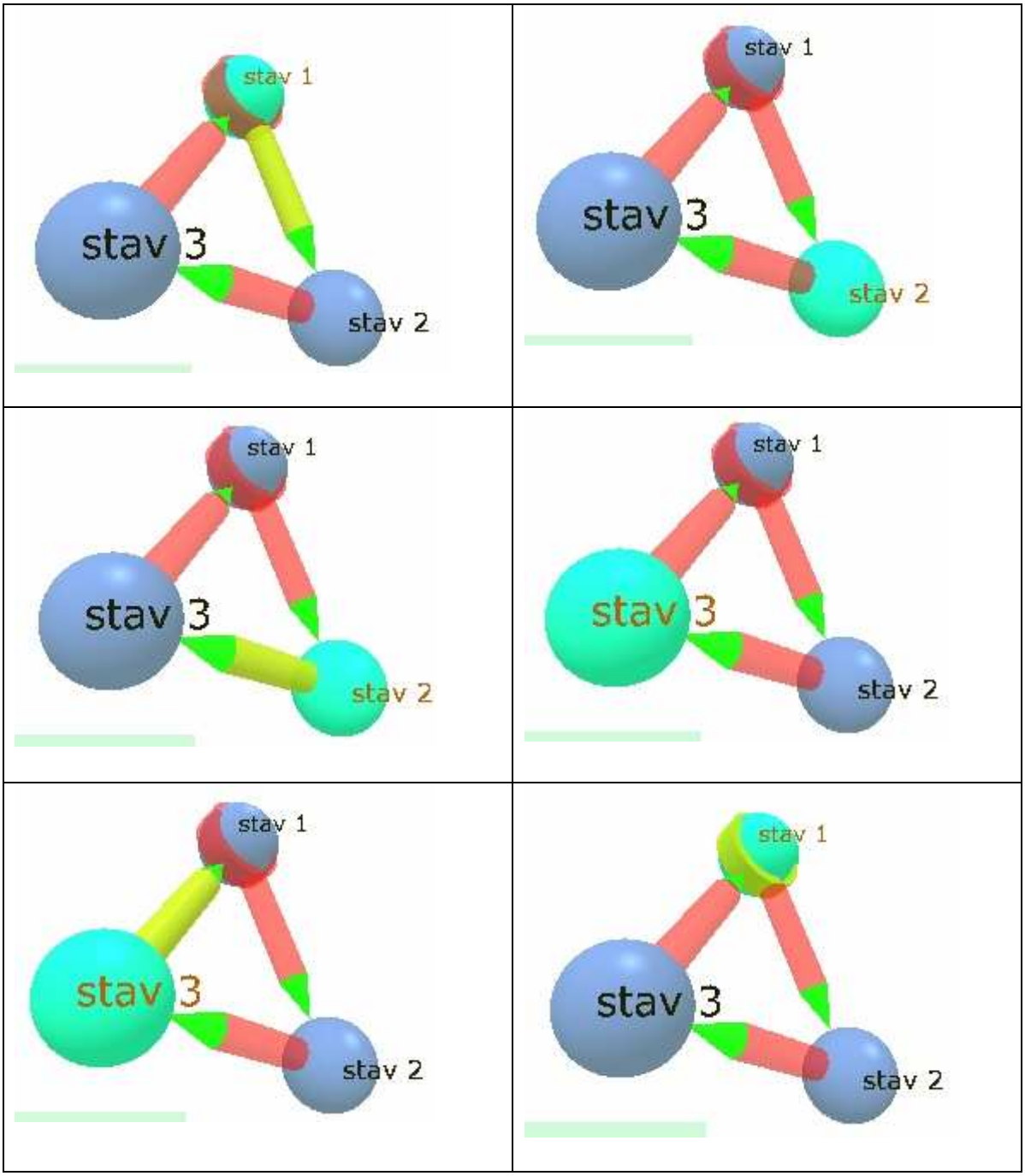
[3] Virius M.: C# Hotová řešení, Computer Press, 2006

[4] Silicon Graphics, Inc: OpenGL Programming Guide, Addison-Wesley Publishing Company, 1997 – Elektronická verze

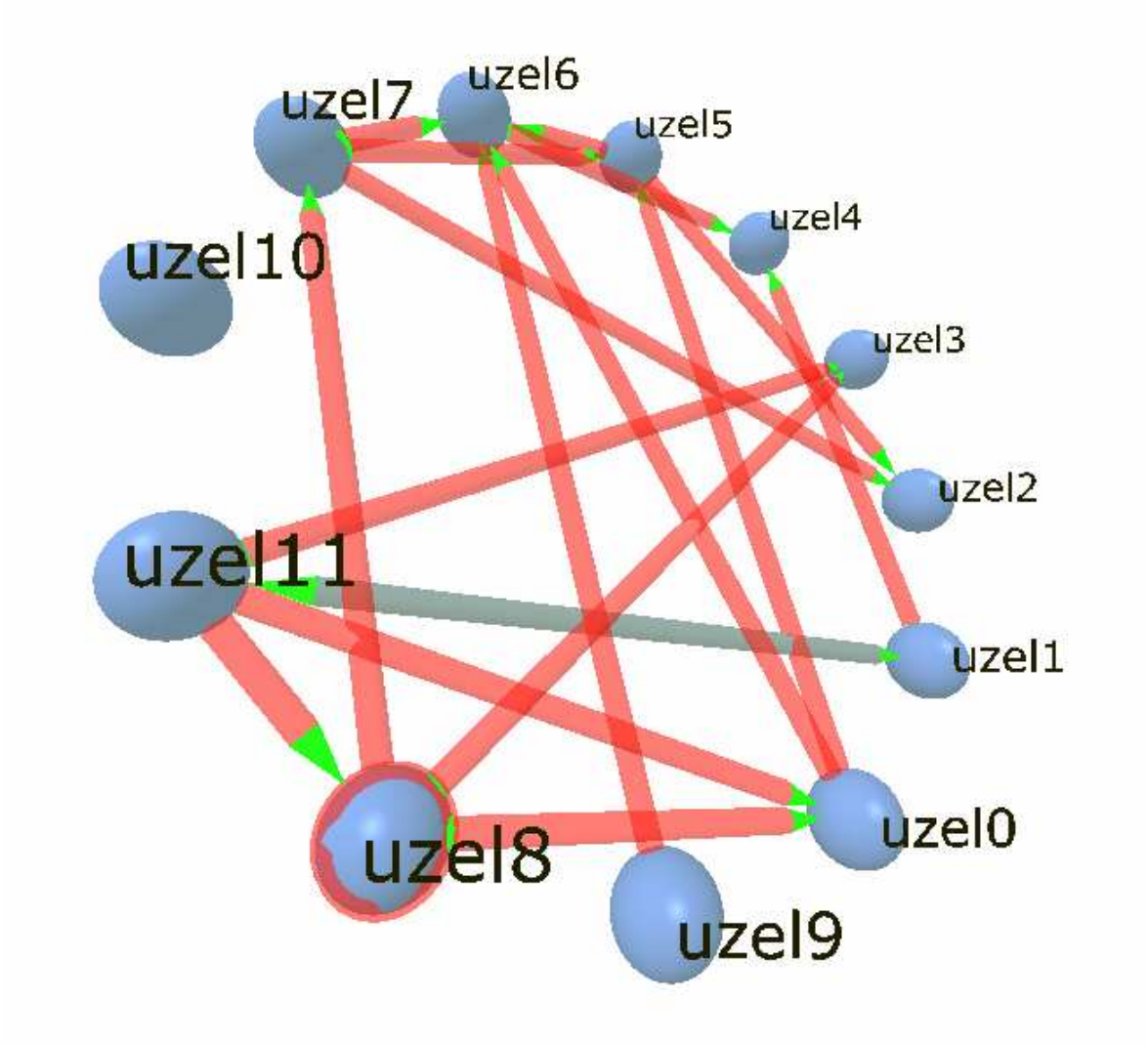
[5] Haščák R.: Zobrazení orientovaného grafu, Diplomová práce ČVUT FEL, 2002

[6] Tišnovský P. : Grafická knihovna OpenGL, seriál článků na www.root.cz, 2002-2004

Příloha: Ukázky grafů vytvořených pomocí programu



Obrázek 1 : Vývoj stavů jednoduchého automatu s jedním stabilním uzlem



Obrázek 2 : Náhodně generovaný graf s rozmístěním vrcholů do kružnice