VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMEDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

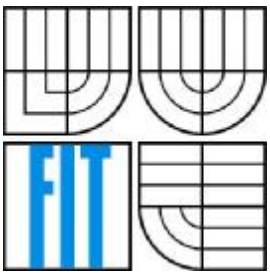
VISUALIZACE CYKLICKÝCH MOTORŮ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE BC. JAN FAJKUS
AUTHOR

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMEDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VISUALIZACE CYKLICKÝCH MOTORŮ

CYCLIC ENGINE VISUALIZATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

BC. JAN FAJKUS

VEDOUCÍ PRÁCE
SUPERVISOR

ING. JAROSLAV PŘIBYL

BRNO 2010

Abstrakt

Diplomová práce se zabývá návrhem a implementací aplikace, která umožňuje navrhovat soukolí složená z ozubených kol. Program je vytvořen ve formě jednoduché počítačové hry. Součástí aplikace je systém pro vykreslování animovaného 3D modelu, který se pohybuje v reálném čase. Jako implementační prostředí byl zvolen jazyk C doplněný o knihovnu OpenGL.

Abstract

This Master's thesis deals with application design and implementation. Application is designed for composition of gears. It is built as a simple computer game. It also contains 3D model renderer, which shows the movement of gears in gearbox. Program is implemented in C language using OpenGL library.

Klíčová slova

Vizualizace, ozubená kola, převodový poměr, přenos síly, OpenGL, počítačová hra

Keywords

Visualisation, gears, gear ratio, power distribution, OpenGL, computer game

Citace

Fajkus Jan: Visualizace cyklických motorů, diplomová práce, Brno, FIT VUT v Brně, 2010

Visualizace cyklických motorů

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením Ing. Jaroslava Přibyla.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jan Fajkus
24.5.2010

Poděkování

Zde bych rád poděkoval vedoucímu práce Ing. Jaroslavu Přibylovi za poskytnuté podkladové materiály a odbornou pomoc během tvorby mé práce.

Obsah

1 Úvod

Stroje se vyskytují všude kolem nás. Potkáváme je na každém kroku, používáme je mnohokrát denně. Jsme na tyto mechanické pomocníky tak zvyklí, že je ani nevnímáme a bereme je jako samozřejmost. Nepřemýšlíme, jakým způsobem to či ono funguje, bereme to jako jasně danou věc. Vnímáme je jako mrtvé studené soustavy z kovů a plastů a neuvědomujeme si krásu precizního zpracování jednotlivých dílů, krásu myšlenky, která dala stroji vzniknout. Nevnímáme krásu mechaniky a to je škoda.

Tento myšlenkový pochod mě, spolu s mou láskou k mechanice a technologiím, vedl k nápadu věnovat se spalovacím motorům v bakalářské práci. S pomocí počítačové vizualizace jsem znázornil princip funkce pístových spalovacích motorů. V diplomové práci plánuji navázat v podobném duchu a to vizualizací převodovek, které jsou nedílnou součástí strojů jak se spalovacím motorem, tak bez něj. Zatímco vizualizace v bakalářské práci byla interaktivní jen velmi omezeně, diplomový projekt bude na interaktivitu klást hlavní důraz. Cílem je vytvořit počítačovou simulaci mechanických převodů v podobě jednoduché počítačové hry. Od tohoto řešení si slibuji větší perspektivitu programu dostat se mezi veřejnost a pomoci lidem pochopit principy převodů ozubenými koly, jakož i zabavit je ve chvílích volna relaxováním.

Ve druhé kapitole se budu věnovat tomu, jak jsem dospěl ke konečnému návrhu oblasti pro simulaci. Zmíním důvody, jaké daly vzniknout postupným variantám a i důvody, které odhalily, proč první dva návrhy nebyly nejvhodnější. Dále se zde čtenář dozví, jak zhruba bude vypadat uživatelské rozhraní programu.

Ve třetí kapitole rozeberu princip funkce celého systému z pohledu návrhu, jednotlivé objekty, jejich vztahy a souvislosti. Nastíním také problémy, které by se mohly vyskytnout a možnosti jak jim předejít, či jak je řešit.

Kapitola čtvrtá je věnovaná hernímu zaměření celé aplikace. Bude uvedeno, jakým způsobem se bude možné pohybovat přes jednotlivé úrovně a také nastíním, jaké úkoly může uživatel očekávat.

Pátá kapitola bude věnována rozboru, jakým bude celá aplikace implementována. Bude zde rozebráno vše, čemu se věnuji již v kapitole druhé, ovšem ne již z pohledu návrhu a analýzy, nýbrž z implementační stránky.

V závěru zhodnotím, jakých výsledků bylo dosaženo, s jakými problémy jsem se při práci potýkal a rovněž se vyjádřím k tomu, nakolik by mohla být moje práce přínosem do budoucna.

2 Volba předmětu vizualizace

V následujících podkapitolách popíšu, jakým způsobem se vyvíjelo moje stanovisko k předmětu simulace v aplikaci. Zmíním postupné návrhy a důvody proč z nich nakonec sešlo. U poslední (a finální) varianty vysvětlím, proč jsem se rozhodl, že právě tato je ideální.

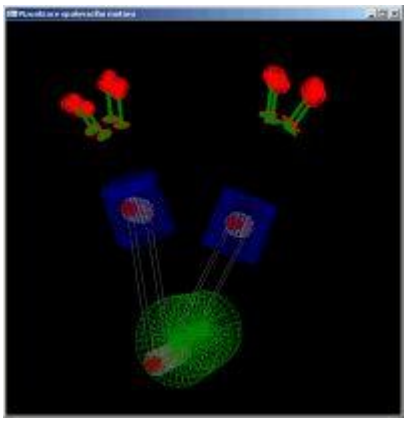
2.1 Skládačka motoru

Původním záměrem bylo vytvořit interaktivní skládačku, kde by se z jednotlivých dílů poskládal celý motor. Jednalo by se v podstatě o specifický způsob zpracování 3D puzzle. Uživatel (hráč) by měl k dispozici pracovní panel se zobrazeným prostorem určeným k sestavení motoru a katalog dílů. Úkolem by bylo správně sestavit jednotlivé díly tak, aby mohl motor pracovat.

Hra by obsahovala několik úrovní, ve kterých by se skládaly různé typy pístových motorů. Kvůli jednoduššímu určení pozice motoru v prostoru by byl vždy nějaký díl předem určený a nehybný. Většinou by se jednalo o setrvačník. K němu by se připojil klikový hřídel, na něj pak ojnice, k nim pomocí ojničních čepů písty. Následovalo by sestavení ventilů a vaček. Rozvod k vačkovým hřídelům (ať už řetězem, řemenem, nebo ozubenými koly – podle typu motoru) by byl pravděpodobně realizován jako jeden díl, protože by nebylo jednoduché vytvořit ovládání takové, aby bylo reálné jej složit z dílů.

Změna dílů v prostoru by byla realizována jako drag-and-drop, přičemž pohyb by byl realizován pouze po dvou osách. Osa, která by svírala nejmenší úhel s osou kamery by byla fixní. Pro pohyb do hloubky by tedy bylo nutné prostor před kamerou natočit. Natáčení prostoru by bylo realizováno klávesnicí a i myší při stisknutém pravém tlačítku. Protože by bylo velmi složité přiložit díly k sobě přesně na pozici, kde se mají dotýkat, bylo by spojování dílů řešeno automaticky při přiblížení na vzdálenost menší než předem daná konstanta. Ta by byla zvolena při testování tak, aby přichytávání (obdobné funkci SNAP v programech typu CAD) zvyšovalo ergonomii ovládání, ale nezpůsobovalo vytváření nechtěných spojení.

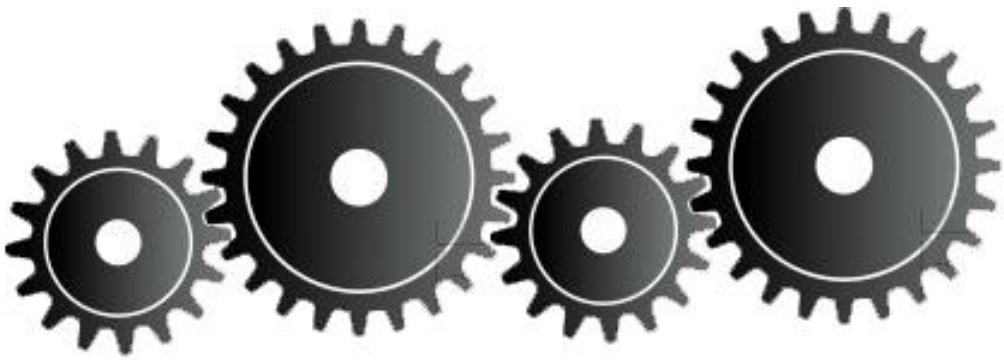
Po dokončení kompozice modelu by uživatel mohl stisknout tlačítko start a tím spustit běh motoru. V případě, že se mu podařilo poskládat motor správně, simulace by se spustila. Zde jsem ale při rozboru řešení došel k závěru, že takováto hra by ani nebyla hrou, protože u motoru existuje pouze jediná možnost, jak jej složit a tedy ani nelze udělat chybu. To bylo důvodem, proč jsem takovéto řešení nakonec zavrhl, ač by pěkně navazovalo na mou bakalářskou práci.



Obrázek 1: Vizualizace cyklického motoru [3]

2.2 Ozubená kola 2D

Dalším konceptem byla aplikace umožňující skládání převodů z ozubených kol v rovině, podobně jako v populární počítačové hře The Incredible Machine od Sierra Entertainment. Jednalo by se o posunování ozubených kol v rovině tak, aby vznikaly převody. Podobně jako u předchozího návrhu by se jednotlivé části (tentokráte ozubená kola) při dosažení určité malé vzdálenosti přichytily. Tím by se mezi nimi vytvořila vazba. Ta by způsobovala, že jakmile by se jedno z ozubených koleček začalo otáčet, vypočítala by se rychlost druhého kolečka na základě převodového poměru a to by se začalo otáčet také. Tím by se dal do pohybu celý již sestavený mechanismus. První komplikací by bylo zabezpečit ošetření smyček, aby se nestalo, že se kolečka navzájem blokují. Druhou komplikací bylo zjištění, že při takovémto návrhu není možné vytvářet složitější převody a zajímavé zadání úkolů pro uživatele.



Obrázek 2: Ozubená kola ve 2D

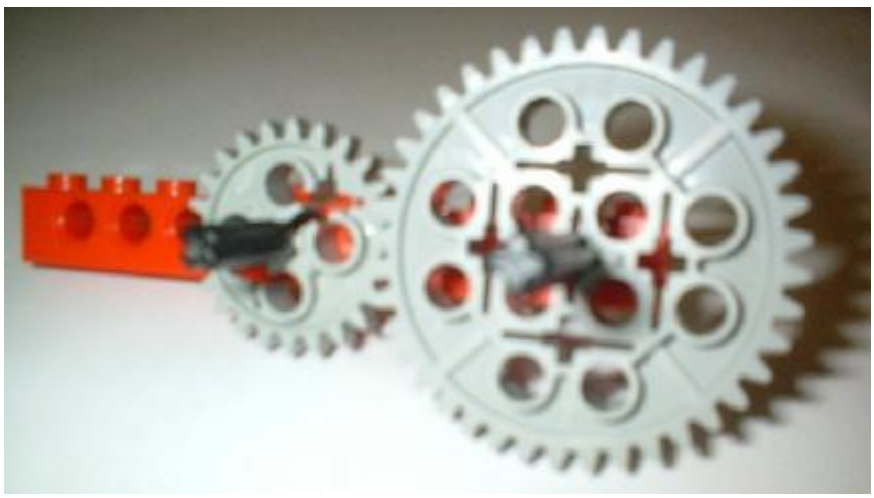
2.3 Ozubená kola 3D

Na základě získaných vědomostí jsem se rozhodl pro následující řešení. V aplikaci bude uživatel sestavovat převody z palety ozubených kol. Místa pro uložení os kol budou předem daná opět s funkcí typu SNAP pro komfortní ovládání. Oproti předchozímu řešení se nebude přichytávat obvod kola k obvodu druhého, ale osa kola k přednastaveným pozicím. Jednotlivé pozice budou od sebe vzdáleny o stejný krok, průměry kol v nabídce budou rovny celočíselnému násobku velikosti tohoto kroku. Tím bude zajištěna jednoduchá kompozice sestavy kol.

Idea použít uniformní vzdálenosti pozic pro osy kol a adekvátně zvolené průměry kol vznikla při realizaci projektu do předmětu Robotika, kde jsme v týmu studentů vytvářeli robotické rameno z legendární stavebnice LEGO Technic od firmy LEGO Group. Odebráním možnosti umístit kola do libovolné pozice v prostoru se ovládání aplikace zjednoduší a více přiblíží skládání stavebnice. To povede k jednoduchému ovládání což je pro aplikaci tohoto typu stěžejní.

Uživatel bude mít k dispozici okno s pracovním prostorem, v němž bude jednotlivé kolečka sestavovat do výsledného převodu. V tom mu bude pomáhat panel nástrojů umožňující se pohybovat napříč vrstvami převodu, umísťovat do pracovního prostoru nová kolečka a osy a opět je odstraňovat. Zde se bude vše kvůli přehlednosti odehrávat ve dvou rozměrech. Třetí rozměr bude suplovaný možností přepínat se mezi vrstvami. To umožní zachovat jednoduchost ovládání aplikace pomocí standardních zařízení (myš, touchpad, trackpoint, trackball atd.).

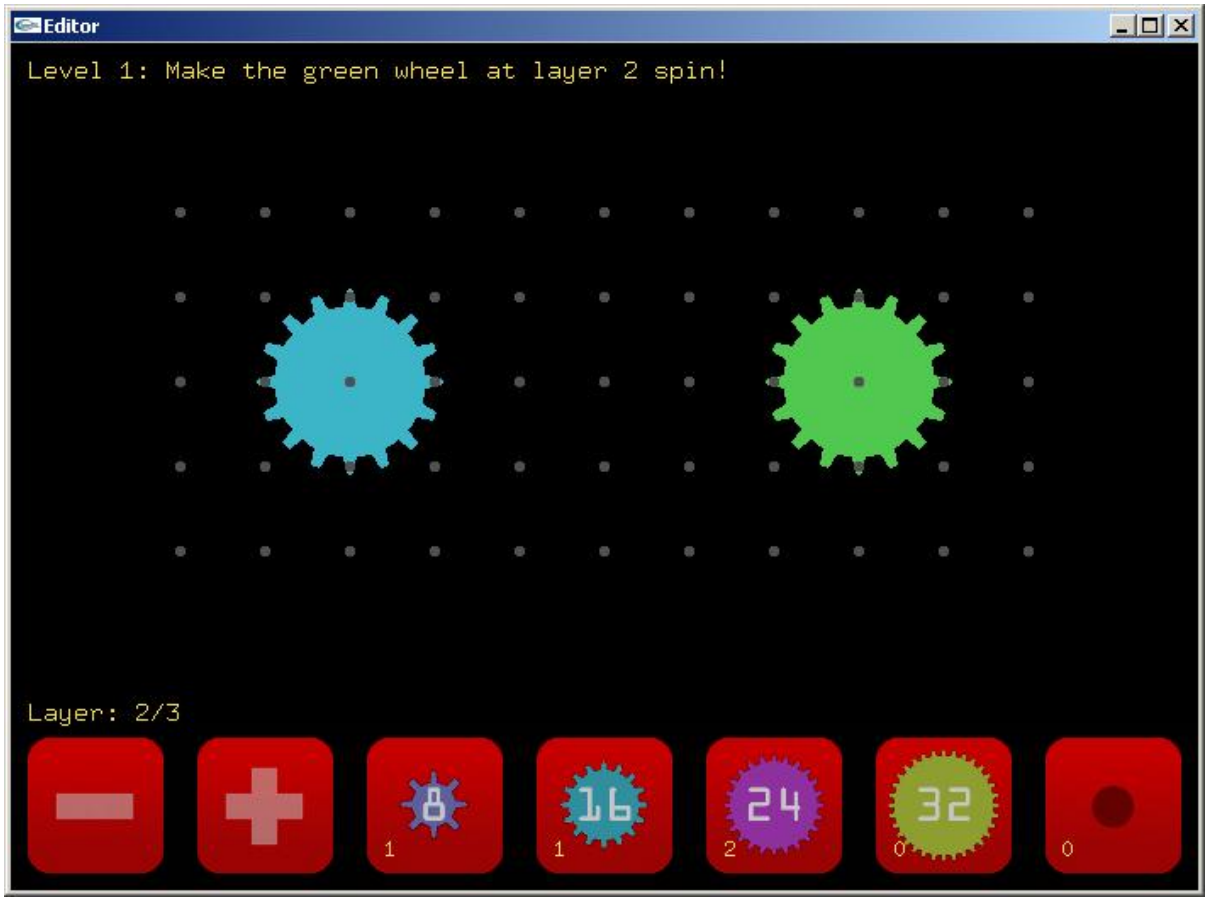
Druhé okno bude zobrazovat 3D vizualizaci vykreslovanou v reálném čase. Uživatel bude mít možnost si 3D model otáčet v prostoru, zoomovat a posouvat s ním po scéně pro možnost jej lépe prozkoumat. Ovládání otáčení bude realizováno obvyklým způsobem. Pomocí drag&drop pravým tlačítkem myši se mění zoom, levým tlačítkem se bude model otáčet a při stisku klávesy shift dojde k posunu v prostoru.



Obrázek 3: Ozubená kola ve 3D, LEGO [7]

2.3.1 Pracovní prostor

Pracovní prostor bude zobrazen v prvním okně aplikace. V něm se bude vykreslovat aktuální stav zvolené vrstvy. Pro zvýšení přehledu uživatele se bude vyobrazovat i obsah vrstev sousedních, rozeznatelný na první pohled od obsahu aktuální vrstvy. Na začátku každé úrovně zde bude umístěno těleso s otvory pro umístění os ozubených kol. Toto těleso bude uzamčeno vůči souřadnicím a nebude možné s ním pohybovat. Některá kola zde budou umístěna už na začátku levelu. Neuzamčenými prvky (ozubenými koly a osami) bude mít uživatel možnost pohybovat stylem drag-and-drop. Pohyb bude realizován pouze v rovině, protože se v tomto zobrazení jedná pouze o dvourozměrný model. Vzhledem k tomu, že myš stejně disponuje pouze dvěma osami, byl by plný pohyb ve třírozměrném prostoru pomocí standardních polohovacích zařízení problematický. Pohyb ve třetí ose tedy nebude možný. Pro pohyb v této ose bude třeba kolo odstranit ze scény a znova jej vložit až po přepnutí mezi vrstvami. Při dostatečném přiblížení středu ozubeného kola k otvoru pro osu se osa automaticky vykreslí a kolo umístí do pracovního prostoru. V případě, že ozubení tohoto kola zapadne do ozubení jiného kola, tyto se spárují. Systém bude pochopitelně hlídat kolize jednotlivých objektů. Po kliknutí na neuzamčené ozubené kolo se toto obarví, aby bylo zřejmé, že je aktivní. Pomocí přesunutí kola mimo samotný pracovní prostor bude možno kolo odebrat z pracovního prostoru a umístit jej do palety dílů. Tím dojde ke zrušení vazeb mezi mazaným kolem a ostatními.



Obrázek 4: Pracovní prostor (dole menu a paleta dílů)

2.3.2 Paleta dílů

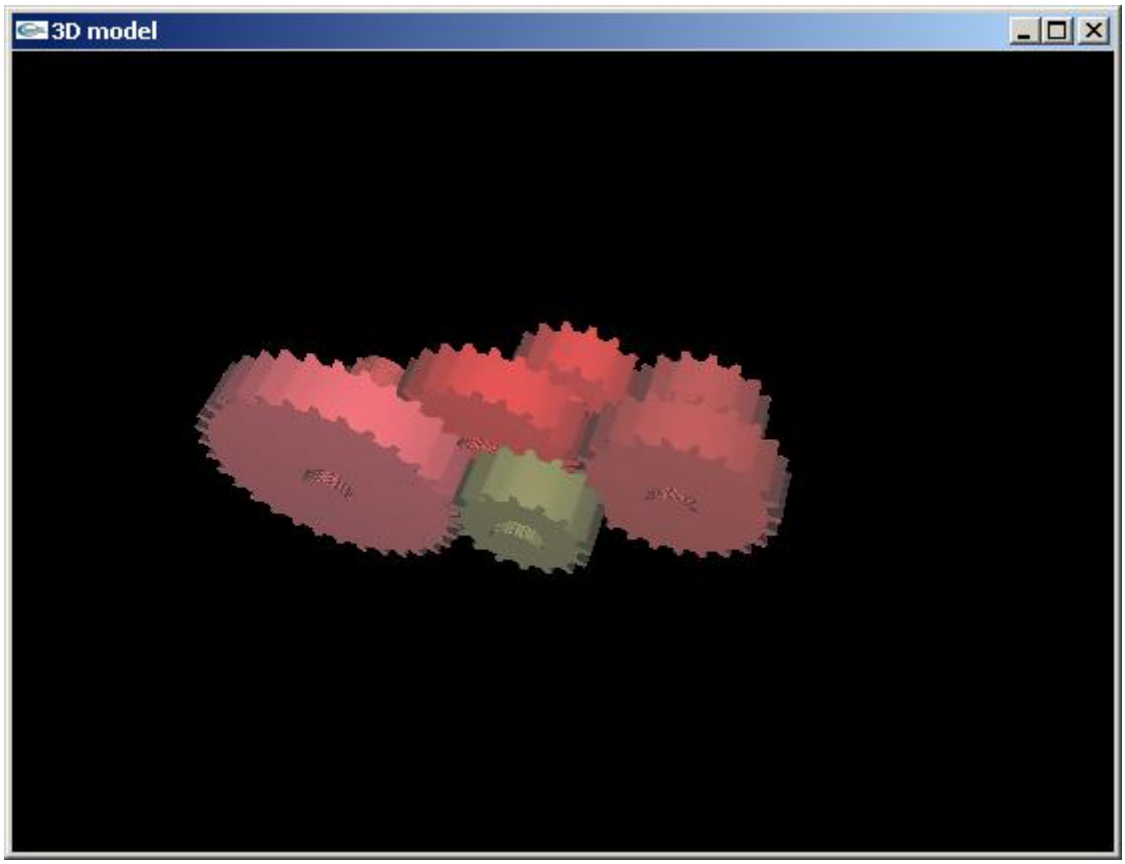
Pod pracovním prostorem bude paleta dílů. U každého dílu bude zobrazen náhled, počet zubů (pokud se nebude jednat o osu) a počet kusů tohoto dílu k dispozici v dané úrovni. Kliknutím na zvolené kolo se automaticky dekrementuje počet těchto kol k dispozici a kolo se objeví v pracovním prostoru (v případě již dosažené nuly nedojde k žádné akci). Po kliknutí na kolo je možné pohybem myši s kolem pohybovat způsobem, jaký byl popsán v předchozí kapitole. Uvolněním stisknutého tlačítka myši se kolo umístí do scény.

Součástí palety dílů jsou také tlačítka pro přepínání aktuální vrstvy. Zobrazená jsou vždy kolečka v aktivní vrstvě. Těmi lze pohybovat, nejsou-li na své pozici zamčené. Dále se zobrazuje obrys koleček v okolních vrstvách z důvodu zlepšení přehlednosti editoru.

Obrázek 5: Paleta dílů a tlačítka pro změnu aktuální vrstvy

2.3.3 Okno s vizualizací

V okně pro vizualizaci se zobrazuje 3D model odpovídající tomu, co je vytvořeno pomocí 2D editoru. Hnané kolečko se otáčí a s ním i ostatní kola připojená převody. Kola se netočí v případě smyčky bránící převodům v pohybu. V případě, že uživatel sestavil soustavu převodů správně, celá soustava se dá do pohybu a uživatel bude informován, že úroveň úspěšně dokončil. Toto platí s jednou výjimkou, o níž se zmíním v kapitole 4.



Obrázek 6: Okno s vizualizací

V tomto okně je možné pomocí myši s animovaným modelem otáčet, zoomovat a nebo posouvat pro lepší možnost prohlédnutí jak celku, tak jednotlivých detailů.

3 Návrh funkce systému

Celý systém se bude skládat z pasivních a aktivních součástí. S pasivními součástmi nebude možné v souřadném systému manipulovat, zatímco s aktivními ano.

Pasivní součásti budou reprezentovány rámem celého převodového ústrojí, zobrazením materiálu s uložením os kol. Tento se bude pro přehlednost zobrazovat pouze v editoru, aby uživatel viděl, do jakého místa může nebo nemůže uložit osu právě přemísťovaného kola. Mezi pasivní součásti lze také zařadit osy kol, které nebudou uživatelsky ovlivnitelné. Vzniknou automaticky připojením středu kola k otvoru pro osu. Zanikají automaticky při zániku kola. Pokud kolo mění svoji pozici, osa se pohybuje také.

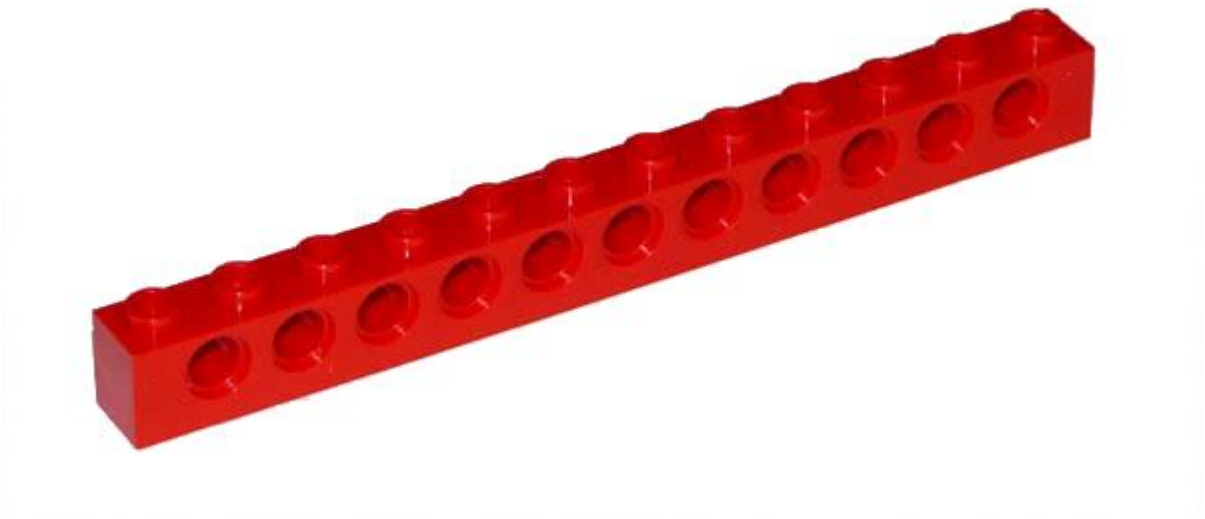
Aktivními součástmi budou ozubená kola různých parametrů. Některé parametry bude možné uživatelsky měnit, jiné ne. Mezi koly budou vznikat binární vazby s parametrem *převodový poměr*. Další vazby budou vznikat mezi aktivními a pasivními prvky. Tím bude zabezpečeno jednoznačné umístění kol v prostoru. Dalším aktivním prvkem je osa bez kola, kterou může uživatel použít pro přenesení točivého momentu přes více vrstev.

3.1 Pasivní součásti

Pasivními součástmi jsou myšleny takové objekty v pracovním prostoru, které uživatel nemůže přímo ovlivňovat, a které ani nezasahují do výpočtů při simulaci.

3.1.1 Rám

Rám bude v aplikaci realizován jako seznam otvorů pro uložení hřídelí. Tím částečně spadá i do aktivních součástí. Pro realizaci výpočtu simulace nebude důležitý, ale bude mít významnou roli pro zachování přehlednosti modelu. V aplikaci bude vyobrazen pouze v 2D editoru aby v modelu nepůsobil zbytečně rušivě. Předem dané možné uložení pro osy kol aplikaci přibližují hře se stavebnicí. Podobně jako u technických stavebnic jsou pevně dány rozestupy mezi možnými uloženími hřídelí a kola odpovídajících průměrů. Tím je dosaženo snadného skládání kol do převodů. Rám je v 2D editoru znázorněn pouze ve formě šedých kroužků, které vyznačují místo pro uložení osy kola. Původní záměr zobrazovat rám i v okně s 3D vizualizací byl nakonec zavrhnut, protože zobrazování čehokoli dalšího kromě koleček snižovalo přehlednost scény.



Obrázek 7: Kostka LEGO Technic

Původně zamýšlené zobrazení rámu převodovky způsobem, který by připomínal kostky oblíbené stavebnice LEGO Technic bylo nakonec zavrhnuto pro zachování přehlednosti simulace. Tím, že jsou vzdálenosti mezi koly jasně vidět ve 2D editoru, není tedy třeba ještě přidávat další prvky do samotné vizualizace.

3.1.2 Motor

Aby uživatel na první pohled poznal, které kolo je poháněné (tudíž vstupní pro točivý moment), měl by u něj být vykreslen motor. Ve většině případů by se jednalo o elektromotor. Z hlediska výpočtu by byl nepodstatný, ale měl zvýšit přehlednost scény.

Obrázek 8: Motor LEGO [1]

Podobně jako u vyobrazení rámu převodovky jsem nakonec došel k závěru, že motor vykreslovaný ve scéně snižoval její přehlednost. Při některých úhlech pohledu na scénu byly

motorem zakryty klíčové části převodů. Odlišení hnacího kola od ostatních se nakonec vyřešilo jednoduše. Vzhledem k tomu, že uživatel má za úkol poskládat převodový mechanismus a v okně simulace se v reálném čase vykreslují změny, stačí nechat hnací kolo otáčet hned od začátku levelu. Tím odpadlo i původně plánované tlačítko „spustit“. Díky této úpravě je celý průběh sestavování převodů atraktivnější, protože uživatel hned vidí, jaký vliv má jeho poslední změna konfigurace systému kol a os na celek. Pro další zvýšení přehlednosti jsou specifická kola vykreslována jinou barvou, než ty ostatní. Konkrétně se jedná o kola s uzamčenou pozicí v souřadném systému modelu. Tato kola jsou uzamčená většinou z důvodu zachování nějaké obtížnosti levelu. Takovým kolem nemůže uživatel pohybovat. Při pokusu o zachycení tohoto kola kurzorem myši v okně editoru kolo pouze změní barvu na červenou, ale svou pozici nezmění. V 3D modelu je pak kolo pro přehlednost vykreslováno červenou barvou pořád. Další možností specifického kola, je kolo takové, které je součástí cíle daného levelu. Cílem může být například rozpohybovat dané kolo na určitou rychlost (v poměru k hnacímu kolu soustavy). Takové kolo je pro jednoduché získání přehledu vykreslováno zelenou barvou a to jak v 2D editoru, tak v 3D vizualizaci.

3.2 Aktivní součásti

Aktivní součásti může uživatel přímo ovlivňovat, v drtivé většině případů je posouvat v prostoru. Z pohledu programu jsou význačné především tím, že se přímo účastní výpočtů při kompozici soustavy a podle prvotního návrhu později i při simulaci. Při implementaci aplikace se podařilo vytvořit takový systém, kdy simulace probíhá v reálném čase s návrhem soustavy a není třeba ji ručně spouštět.

3.2.1 Ozubená kola

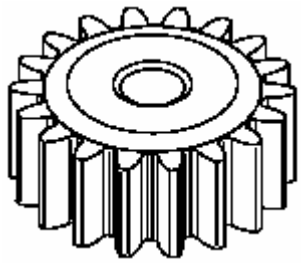
Ozubená kola budou hrát klíčovou roli v celé aplikaci. Budou mít několik parametrů. Z neměnných to bude počet zubů (a jemu odpovídající průměr) a barva pro jasnou identifikaci. Různé velikosti kol mají nastavené různé barvy pro rychlý odhad o jak velké kolo se vlastně jedná.

Z uživatelsky přímo ovlivnitelných jmenujme pozici středu v prostoru a booleanovskou hodnotu *uchyceno k ose*. Pomocí této proměnné bude možné zvolit, zda se dvě kola na stejné ose můžou otáčet nezávisle na sobě a nebo zda jsou osou pevně spojena a tím pádem mají pevně danou stejnou úhlovou rychlost. Tímto spojením automaticky vzniká vazba mezi objekty typu *kolo*. Protože se ukázalo, že neexistuje jednoduchý způsob, jakým tuto vlastnost kola uživatelsky ovládat přímo, byl nalezen systém, jak zachovat jednoduchý a intuitivní způsob ovládání aplikace i v této věci. Kola se stejnou pozicí středu v souřadnicích x a y, jsou spolu svázána (mají stejnou úhlovou rychlost), jen pokud jsou na sousedních hladinách. Z toho vyplývá, že pokud mají být dvě kola na stejné ose, ale nemá mezi nimi být vazba, musí být alespoň ob jednu vrstvu. V opačném případě, tedy v takovém,

kdy kola na stejné ose mají být spolu svázány, ale nejsou v sousedících vrstvách lze využít prvku typu *osa* (více v následující kapitole).

Vlastnosti, které bude mít kolo v závislosti na aplikaci, budou definovány vazbami mezi jednotlivými koly. Ty se budou přímo ovlivňovat ve dvou případech. První případ je takový, že dvěma kolům vzájemně zapadají zuby. V závislosti na převodovém poměru pak bude otáčení jednoho kola způsobovat otáčení druhého a naopak. V tomto případě může být různá úhlová rychlost kol, zatímco obvodová bude shodná. Směr otáčení je v takovémto případě opačný Druhý případ je ten, kdy jsou kola spojena společnou osou. Tato situace může nastat pokud budou tato dvě kola v sousedních vrstvách a nebo pomocí objektu typu *osa*. Při tomto vztahu je u obou kol shodná úhlová rychlost otáčení. Vzhledem k tomu, že v průběhu kompozice stroje nemusí být vždy známo, které kolo z dvojice bude vstupní a které výstupní, nelze tuto relaci realizovat jako jednosměrnou. To způsobovalo velké navýšení složitosti při zpracování dat o prvcích systému. V průběhu implementace se ale tento nedostatek podařilo ošetřit dynamickým generováním vztahů mezi koly. Tím se odstranila nutnost používat neorientované hrany grafu.

Vzhled kol bude daný v aplikaci podle jejich neměnných parametrů. Pro zachování konzistence vzhledu jednotlivých komponentů budou kola zobrazena podobným způsobem jako jsou vyrobena ve stavebnici LEGO.



Obrázek 9: Ozubené kolo [9]

3.2.2 Osy

Při umístění středu kola k otvoru pro osu se kolo ukotví a automaticky se vytvoří jeho osa. Tento odstavec však pojednává o jiném typu osy. Prvek typu *osa* je určený k propojení kol a lze jej využít k přenosu točivého momentu mezi koly vzdálenými o více než jednu vrstvu. V podstatě se jedná o speciální případ ozubeného kola s nula zuby, které se umí navázat na sousední kolo jen pomocí osy, nikoli však vhodnou vzdáleností v rámci jedné vrstvy.

3.3 Vztahy mezi aktivními objekty

Nejjednodušším vztahem mezi dvěma aktivními objekty bude nulový vztah. Takovéto objekty se nebudou navzájem vůbec ovlivňovat. To se týká kol umístěných v jiné rovině bez společné osy, ke které by byly ukotveny a kol ve stejné rovině, jejichž středy jsou navzájem vzdáleny o více, než je součet jejich poloměrů. Menší vzdálenosti mezi středy není možné docílit, protože by tak muselo dojít ke kolizi mezi koly. To bude hlídáno aplikací, která uživateli neumožní takovouto kolizi vůbec způsobit.

Dalším vztahem bude binární relace mezi koly, jejichž osy jsou vzdáleny právě o součet jejich poloměrů a leží ve stejné rovině. S takovýmto převodem se v běžném životě setkáváme velice často. V našem případě (u převodů složených z ozubených kol) se kola otáčejí opačným směrem. V závislosti na poměru počtu jejich zubů může být rozdílná i úhlová rychlost jejich otáčení (absolutní hodnota obvodových rychlostí je shodná). Kolo může být touto relací napojeno na více dalších ozubených kol.

Posledním užitým vztahem mezi dvěma aktivními prvky bude fixace osou. V takovémto vztahu můžou být pouze kola se společnou osou. V případě, že uživatel zvolí, že tato osa má být pevně fixována ke kolům, způsobí tato relace, že obě kola se budou otáčet shodným směrem a se stejnou úhlovou rychlostí.

Kombinací posledních dvou relací lze vytvářet složité systémy převodů, které budou v systému reprezentovány grafem s ohodnocenými hranami. Hodnoty hran udávají převodový poměr.

3.3.1 Detekce kolize otáčení

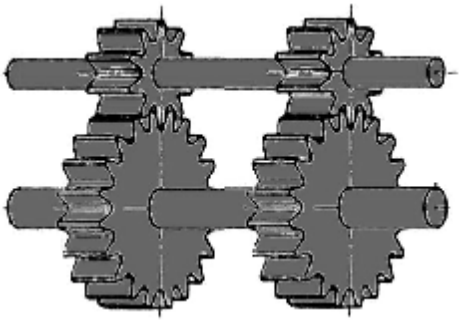
Na první pohled by se mohlo zdát, že dostatečným systémem detekce zablokování systémů převodů by byla detekce kružnic v grafu. V případě zjištění, že graf není stromem by systém vyhodnotil systém převodů jako zaseknutý. Při opačném výsledku by kontrola byla negativní a nic by nebránilo spuštění simulace.

Bohužel algoritmus pro vyhodnocení nemůže být takto jednoduchý. Zatímco v případě negativního výsledku detekce kružnic v grafu můžeme s jistotou říct, že systém zaseknutý není, v případě jejich výskytu musíme provést další analýzu. Ta spočívá ve vyhledání všech kružnic v souvislém grafu obsahujícím nultý uzel (hnací kolečko) a analýze každé z nich následujícím způsobem:

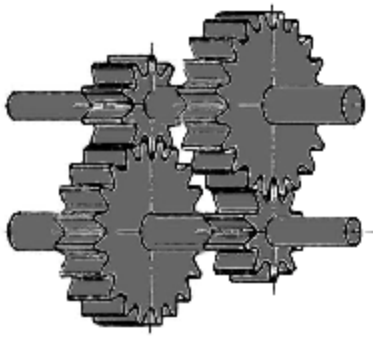
1. U prvního uzlu zvolíme číslo N a to uložíme
2. Hodnotu čísla N uložíme také do proměnné M
3. Postupně procházíme kružnici a číslo M vynásobíme hodnotou hrany. Výsledek uložíme do M.
4. Po dosažení prvního uzlu porovnáme M a N. V případě, že se nerovnají, vyhodnotíme systém jako zaseknutý. V opačném případě pokračujeme další kružnicí.

Pokud se podaří projít celý souvislý graf, můžeme systém převodu prohlásit za provozuschopný a spustit simulaci.

Tato složitá problematika se nakonec podařila efektivně vyřešit pomocí dynamického procházení grafu, a to využitím rekurzivní funkce bez nutnosti ukládat data do uživatelských proměnných.



Obrázek 10: Smyčka nebránící pohybu



Obrázek 11: Smyčka bránící pohybu

3.4 Simulace

Před započetím simulace je třeba vyhodnotit rychlosti otáčení všech zúčastněných kol (včetně směru rotace – dále jen rychlost). To proběhne tak, že se nejdříve nastaví rychlost hnacího kola. Rychlost hnacího kola bude uložena v souboru se zadáním příslušného levelu. Dále algoritmus projde graf a násobením rychlosti otáčení s hodnotou hrany nastaví rychlosti všem ještě nenastaveným kolům. Tímto způsobem projde celý graf. Tím je zároveň zajištěna kontrola smyček bránících pohybu soukolí.

Po inicializaci se začnou kola autonomně pohybovat, budou řízena globálním časem. Globální čas bude velice výhodný pro synchronizaci aktuálního pootočení kol z hlediska zapadání zubů do zubů sousedního kola.

4 Hra

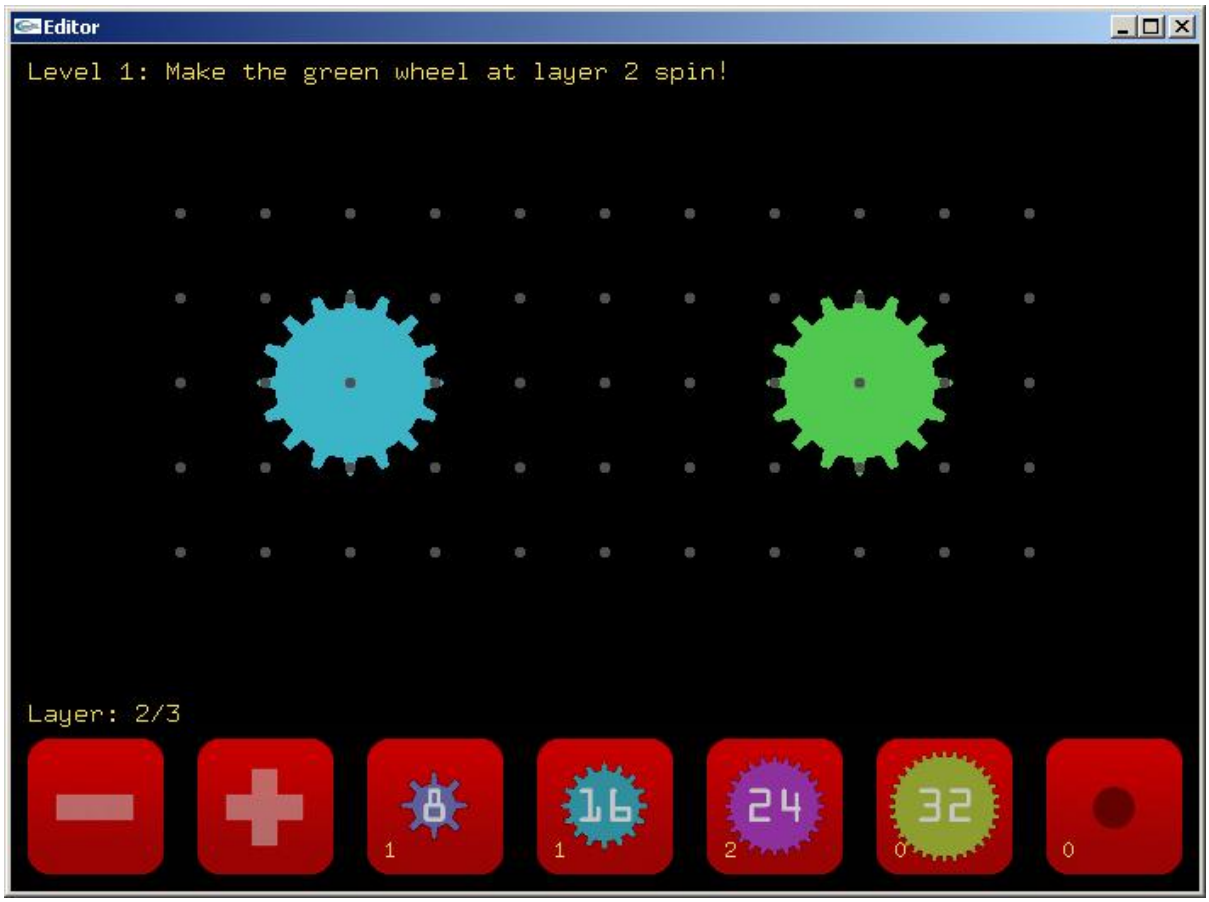
Hra jako taková bude sestávat z několika po sobě následujících úrovní, ve kterých se bude postupně zvyšovat obtížnost úkolů. Pro postoupení do další úrovně musí uživatel úspěšně dokončit stávající. Ve většině úrovní bude mít k dispozici omezený počet ozubených kol, takže předpokladem pro dokončení úrovně bude představivost a schopnost logicky uvažovat. Uživatel nebude mít omezený čas ani počet pokusů.

4.1 Ovládání

Způsob ovládání aplikace je intuitivní a odpovídá zavedeným modelům ovládání, jaké známe z jiných aplikací. Většinou uživatelů upřednostňované ovládání pomocí myši nebo jiného polohovacího zařízení je zde použitelné pro téměř všechny případy požadavků na aplikaci. Použit je i intuitivní způsob ovládání typu drag&drop, se kterým se uživatelé setkávají v různých jiných programech dnes a denně. Ovládání je realizováno zvlášť pro okno s dvourozměrným editorem a zvlášť pro okno s 3D vizualizací.

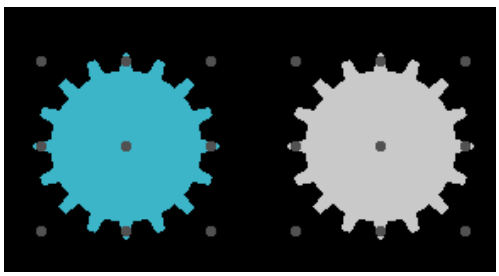
4.1.1 Editor scény

V editoru je možné využívat jak myši (případně jiného polohovacího zařízení), tak klávesnice. Ovládání pouze s použitím klávesnice není možné. V dnešní době není obvyklé, aby počítač neměl připojené žádné polohovací zařízení, takže jsem takovou variantu ani nepředpokládal. Lze říci, že existence myši nebo nějaké její náhrady je jeden z požadavků na použití aplikace.



Obrázek 12: Okno editoru

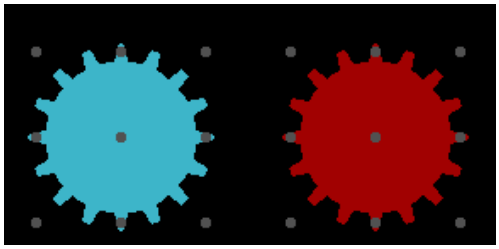
Kola lze po pracovní ploše posouvat pomocí techniky drag&drop. Stačí přesunout kurzor do prostoru nad zvoleným kolem, stisknout levé tlačítko myši a nechat jej stisknuté. Pohybem kurzoru se pak pohybuje i zvolené kolečko. Po uvolnění tlačítka myši se kolečko umístí na požadované souřadnice a provede se výpočet scény pro vizualizaci v druhém okně. Aktivní kolečko se zabarví na světle šedou barvu, aby uživatel hned poznal, s kterým kolem pohybuje.



Obrázek 13: Zvýraznění aktivního kola, kterým je právě pohybováno (pravé kolo)

Některými koly je ale zakázáno pohybovat, mají svoje souřadnice zamknuté proti změně. Taková kola jsou zde kvůli zadání úkolu. Kdyby s nimi bylo možné pohybovat, rapidně by se snížila

obtížnost úkolu nutného pro dokončení levelu. Když se uživatel pokusí pohybovat takovým kolem, kolo nezmění svou barvu na světle šedou jako v předchozím případě, ale na červenou, aby tak upozornilo uživatele, že s ním pohybovat nepůjde. V tomto případě se pozice kola nemění, ani když uživatel pohybuje myší.



Obrázek 14: Zbarvení kola uzamčeného proti posunu při pokusu o jeho posunutí (pravé kolo)

Při posouvání kola se kolo přichytává středem na místa pro tento účel připravená. To kvůli zjednodušení přesného umístění kola. Tato místa jsou znázorněna malými šedými tečkami v prostoru pracovní plochy. Kolo je sice možné umístit i jinak, než středem na jednu z těchto pozic, zbytečně to ale pak ztěžuje navázání na kolo jiným kolem, kdy je pak třeba trefit se s pozicí kola na pixel přesně.

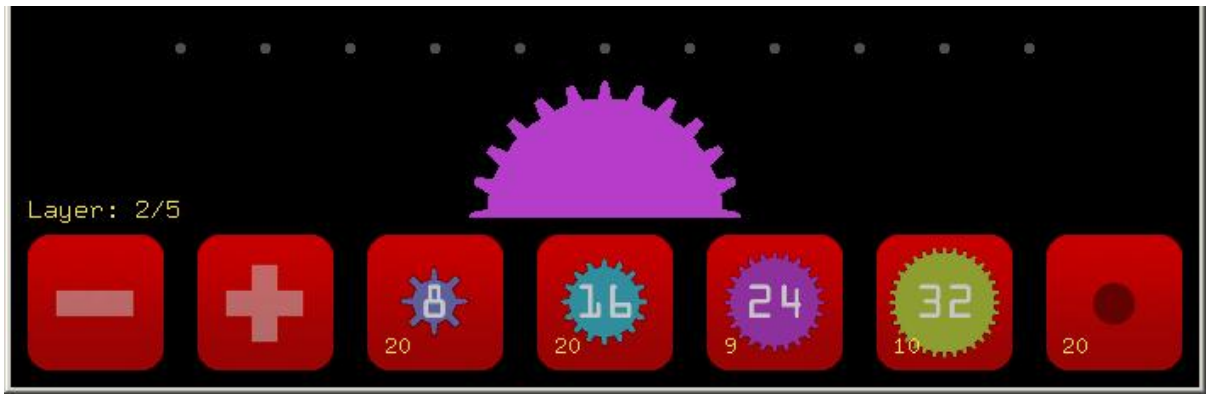
Pokud uživatel při přesouvání kola posune jeho střed pod pracovní plochu, tedy do prostoru menu a uvolní tlačítko myši, dojde k odstranění kola ze scény. Kolo je pak umístěno v paletě nástrojů a připraveno pro vložení pomocí příslušného tlačítka. Tímto způsobem lze i přesouvat kola mezi vrstvami. Stačí kolo přesunout ze scény do palety, přepnout se do požadované vrstvy a kolo přidat.

Přidávat kola a přepínat se mezi vrstvami může uživatel pomocí ovládacího menu umístěného ve spodní části okna s editorem.

Obrázek 15: Menu ve spodní části okna a vyznačení aktivní vrstvy

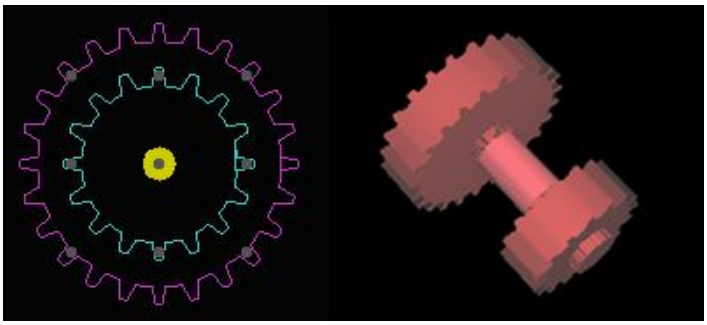
Menu obsahuje sedm tlačítek. Pomocí prvních dvou se symboly mínus a plus lze přecházet mezi vrstvami. Aktuální vrstva je vyznačena v levém dolním rohu pracovní plochy nad tlačítky pro pohyb mezi vrstvami. Číslo před lomítkem určuje aktuální vrstvu, číslo za lomítkem potom celkový počet vrstev. Následující čtyři tlačítka slouží pro přidávání koleček různé velikosti do scény. V editoru má každá velikost kola svou barvu, aby bylo používání editoru jednoduché. Tato barva koresponduje i s barevným provedením koleček na tlačítkách. Zobrazení koleček není proporcionální. Kdyby bylo, tak pokud by se mělo na plochu tlačítka vejít největší kolečko, to nejmenší by téměř nebylo vidět. Je tedy alespoň zachována vzájemná velikost v náznaku. Velké číslo uprostřed kola na

tlačítku sděluje uživateli počet zubů. To se hodí zejména v levelech, kde je úkol sestavit sestavu převodů s přesně daným celkovým převodovým poměrem. Poslední tlačítko přidává do scény osu. Čísla zobrazená v levém dolním rohu tlačítek udávají, kolik má ještě uživatel k dispozici koleček dané velikosti. Při kliknutí na tlačítko pro přidání kola se kolečko (pokud je ještě nějaké k dispozici) vykreslí do aktuální vrstvy a to do místa, kde je střed kola v polovině šířky pracovní plochy a těsně nad hranicí menu. To je výhodou v případě, že uživatel přidá kolo omylem. Posune jej pouze o několik pixelů dolů a kolečko je zpátky v paletě nástrojů.



Obrázek 16: Nově přidané kolo s 24 zuby

V pracovní ploše jsou vykreslena kola (a osy), která se nacházejí na aktivní vrstvě a kterými tedy může uživatel v daném zobrazení manipulovat. Pro zvýšení přehlednosti a jednodušší navazování kol mezi vrstvami se vykresluje i obsah sousedních vrstev ale pouze obrysem. S těmito objekty uživatel manipulovat nemůže, jsou tam vykresleny pouze pro lepší přehlednost. Pro lepší pochopení přidávám dva obrázky.



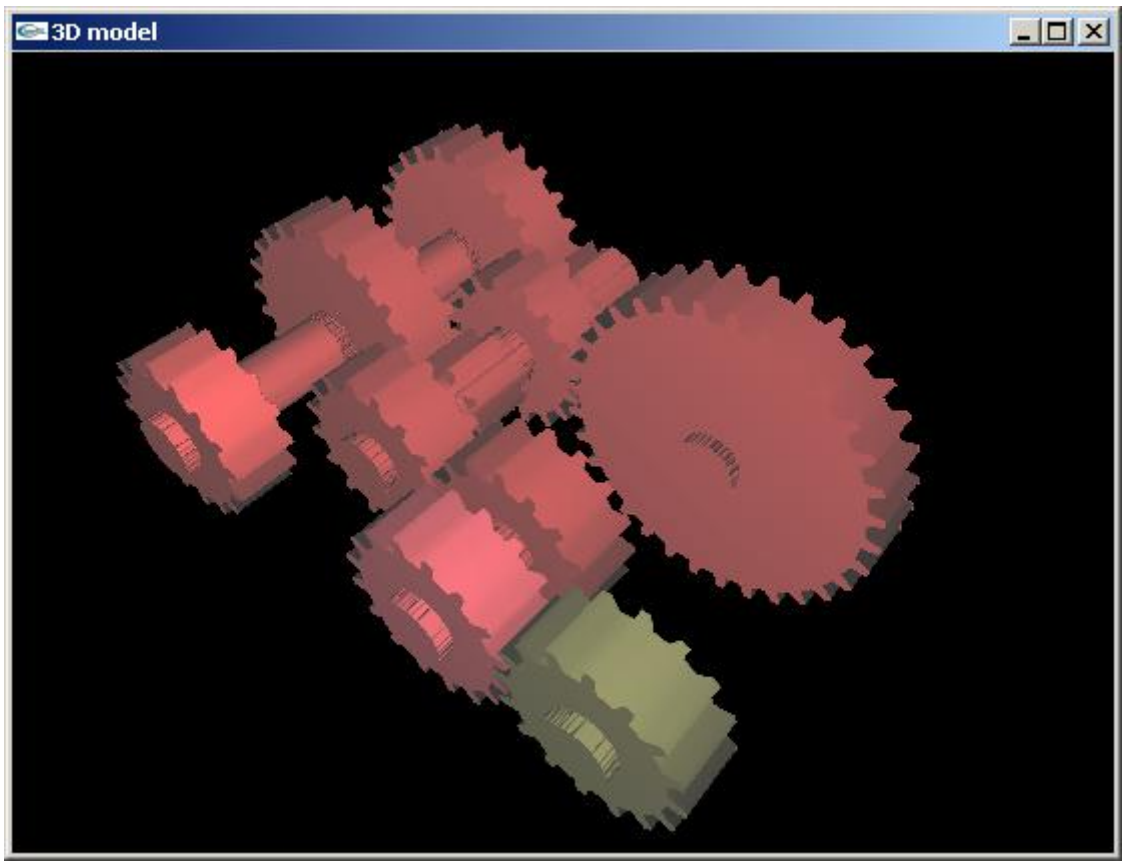
Obrázek 17: Vyobrazení osy v aktivní vrstvě a obrysů kol ve vrstvách vedlejších x porovnání s výsledkem ve 3D

Klávesových zkratek lze využít také. Pomocí kláves A a Y je možné se pohybovat mezi vrstvami. Tlačítkem R dojde k vyresetování nastavení levelu a pomocí klávesy L lze level přeskočit (to je možné pro testovací účely). Ukončení programu lze provést pomocí stisku klávesy Escape, klávesy Q nebo klávesy X.

4.1.2 Okno s 3D vizualizací

Zatímco v okně s editorem může uživatel ovlivňovat rozložení jednotlivých komponentů sestavy převodů, v tomto okně nelze vzájemné vztahy jednotlivých prvků vizualizace nijak ovládat. Toto okno neslouží pro zadávání parametrů programu, ale pro zobrazení výsledku simulace.

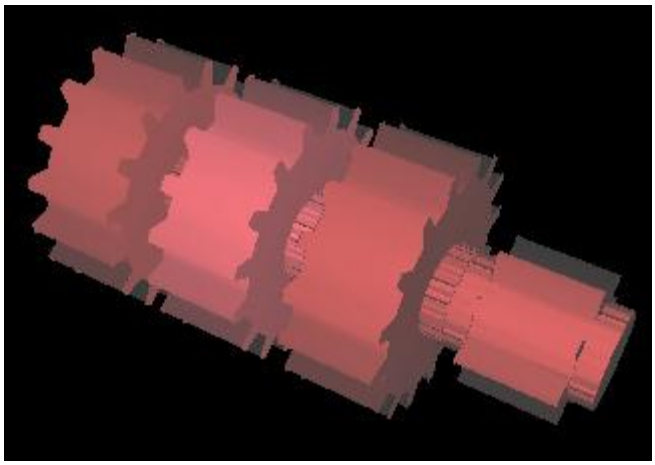
Ovládání tady nespočívá v posunech jednotlivých částí vizualizace, ale k pohybu celého vymodelovaného systému před kamerou. Uživatel může použít obvyklou rotaci modelu pomocí pohybu myši při současně stisknutém tlačítku myši. Tímto způsobem lze modelem otáčet kolem dvou os. Tak lze dosáhnout v podstatě libovolného natočení modelu a ten si tak prohlédnout ze všech stran, takže absence otáčení podle třetí osy není na škodu. Pohybem kurzoru polohovacího zařízení při stisknutém pravém tlačítku je možné model přibližovat ke kameře nebo jej naopak vzdalovat, jde tedy o obvyklou funkci zoom. Při stisknuté klávese shift lze pomocí levého tlačítka myši a pohybu kurzoru s modelem posouvat v osách x a y. To je využitelné při zájmu prohlédnout si nějakou část soustavy převodů v detailu.



Obrázek 18: Okno s 3D modelem

Klávesové zkratky fungují jinak, než při aktivním okně s editorem. Klávesa R v tomto případě obnoví výchozí nastavení kamery. Klávesy Escape, X a Q slouží i zde k vypnutí aplikace. Jiné klávesové zkratky zde nefungují.

Pro ještě lepší přehled o tom, ve které vrstvě se zrovna uživatel nachází je aktivní vrstva zvýrazněna i v okně s 3D modelem. To je realizováno pomocí vykreslení kol v aktivní vrstvě světlejší barvou než u kol ostatních.



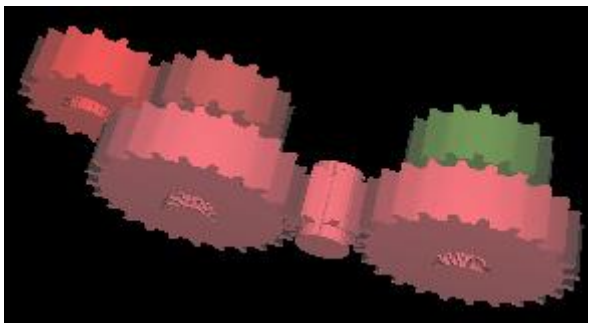
Obrázek 19: Zvýraznění aktivní vrstvy světlejší barvou

4.2 Návrhy úkolů pro úrovně

Při návrhu jednotlivých úrovní jsem se snažil být kreativní, aby hra nepůsobila stereotypně. Protože méně je někdy více a kvantita nenahrazuje kvalitu, vytvořil jsem 4 úrovně (levely). Program je navržen tak, aby nebylo příliš problematické další úrovně vytvořit.

4.2.1 1. level

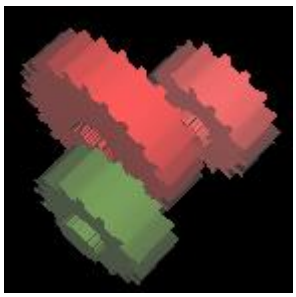
S pomocí omezeného množství koleček je třeba rozpohybovat zelené kolečko. Úkol není složitý, jedná se spíše o tutoriál pro naučení uživatele hru ovládat. Jak je u tohoto typu her obvyklé, možností je více.



Obrázek 20: Jedno z možných řešení 1. levelu

4.2.2 2. level

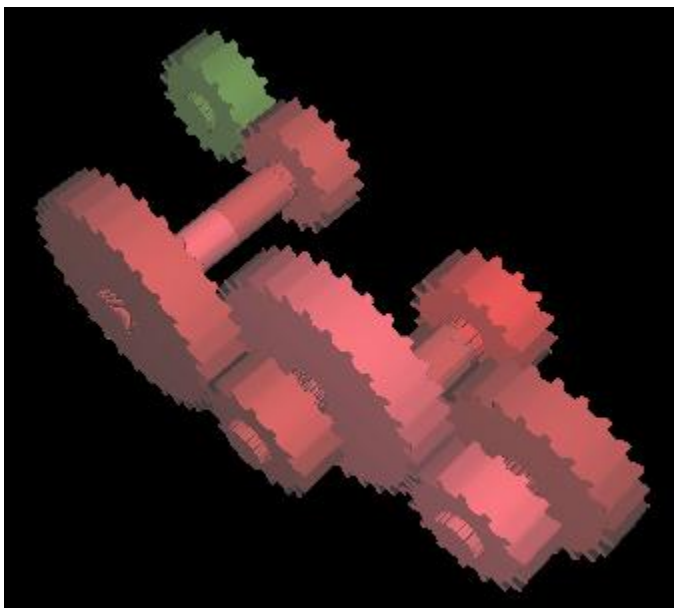
Úkol je velmi podobný tomu z předchozího levelu, jen je o něco ztížen skutečností, že krom hnacího a cílového kola je zde umístěno ještě jedno kolo, které má zamknutou pozici v souřadném systému a nelze s ním tedy uživatelsky pohybovat. To snižuje počet možných řešení a tím zvyšuje nároky na uživatelovu představivost.



Obrázek 21: Výchozí stav 2. levelu

4.2.3 3. level

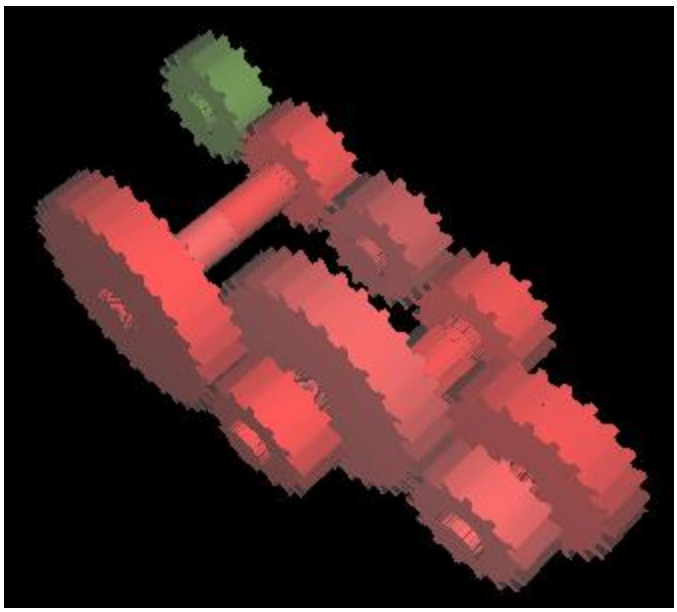
V tomto levelu má uživatel úkol zahrát si na hodináře a vytvořit převod mezi minutovou a hodinovou ručičkou v hodinovém strojku. Cílem je roztočit zelené kolo tak, aby se otáčelo dvanáctkrát pomaleji, než hnací kolo systému. V tomto případě už má uživatel k dispozici prakticky neomezený počet komponentů, takže provedení převodového mechanismu už je zcela na něm. Jsou zde na uživatele kladeny vyšší nároky, musí vědět, jakým způsobem počet zubů kol v převodu ovlivňuje převodový poměr mezi koly a tím pádem výslednou rychlost otáčení.



Obrázek 22: Jedno z nejjednodušších řešení 3. levelu

4.2.4 4. level

Po vytvoření části hodinového stroje nešlo odolat a nenavázat na pověst ze Starých pověstí českých. Podobně jako mistr Hanuš po dokončení pražského orloje a následné zradě od radních mechanismus zničil, je nyní úkolem uživatele vytvořit smyčku blokující pohyb soustavy převodů a tím celý systém zastavit. K dispozici má kola a osy, které v předchozím levelu nepoužil. Při spuštění levelu ovšem došlo k zamknutí všech kol ve svých pozicích. Není tedy možné již existující soustavu převodů modifikovat, nýbrž pouze přidávat další komponenty takovým způsobem, aby k zastavení celku došlo. Tady je opět více možností a jejich počet je závislý na způsobu dokončení předchozího levelu.



Obrázek 23: Dokončení 4. levelu přidáním jednoho kola.

5 Implementace

Tato kapitola pojednává o použitých řešeních při implementaci aplikace. Krom obecných údajů o použitých technologiích se zde pokusím objasnit, jaká řešení byla konkrétně použita pro ten který dílčí problém. Uvedu zde, jak se měnily některé části kódu a co bylo důvodem k těmto změnám. Pro zvýšení informační hodnoty zde také přidávám části zdrojového kódu doplněné o komentář.

5.1 Jazyk

Pro implementaci aplikace byl zvolen jazyk C, doplněný o knihovnu glut, umožňující jednoduché ovládání funkcí OpenGL. Toto rozhodnutí bylo podpořeno mimo jiné tím, že na stejné platformě jsem na konci bakalářského studia vytvářel aplikaci pro bakalářskou práci. Dalšími důvody byla větší propracovanost problematiky na internetu a nedostatek kvalitních zdrojů informací a knihoven pro jiné jazyky.

Další výhodou byl dostatek materiálů jak z tvorby bakalářské práce, tak z absolvovaných předmětů v minulých semestrech. V neposlední řadě jsem značně využíval kódů, které byly k dispozici u tutoriálů na stránkách ceskehry.cz a root.cz.

5.2 Objekty

Objekty simulace byly v průběhu implementace aplikace realizovány různým způsobem, kvůli různým komplikacím se jejich modely několikrát měnily. Nakonec se jejich způsob realizace ustálil na následujících řešeních.

5.2.1 Ozubené kolo

Ozubené kolo, tedy nejdůležitější element aplikace, je implementováno formou struktury.

```
typedef struct _wheel{
      int x;
      int y;
      int l;
      int r;
      colorVector color;
      bool locked;
      bool offset;
      float speed;
      bool powered;
      bool goal;
      bool checked;
      }wheel;
```

Proměnné `x` a `y` určují pozici středu kola v prostoru, v souřadném systému. Souřadnice x a y jsou pro zobrazení ve 2D editoru uvedeny v pixelech, pro 3D zobrazení je potom použitý stejný počet bezrozměrných jednotek. Protože u OpenGL je použito relativního systému vzdáleností, ničemu to nevadí. Vzhledem k tomu, že je posun kol realizován pomocí 2D editoru, kde je s objektem manipulováno pomocí polohovacího zařízení (nejčastěji myši), které vrací polohu kurzoru v celých pixelech, není možné střed kola umístit jinam, než na souřadnice složené z celých čísel. Proto nebylo třeba počítat s necelými čísly a byl použit datový typ integer, který šetří paměť.

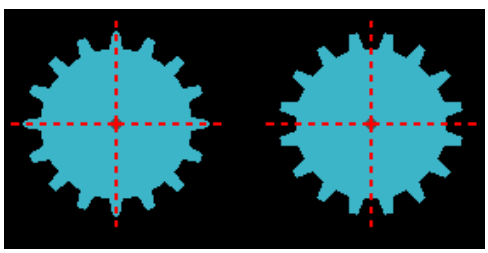
Proměnná `l` určuje, v jaké vrstvě se kolo nachází. To určuje pozici středu kola na ose z, ale na rozdíl od předchozích proměnných není tento údaj absolutní. Hodnota této proměnné dosahuje hodnot `1` až `lmax`, což je maximální počet vrstev v daném levelu. Hodnota se vynásobí s konstantou `layer`, určující tloušťku jedné vrstvy. Tím už vzniká přímá hodnota umístění středu kola na ose z. Speciální případ je hodnota -1, které tato proměnná také může nabývat. To je příznak situace, kdy kolo sice je ve hře, ale není v žádné z vrstev. Kolo je tedy k dispozici v paletě použitelných nástrojů. Toto řešení je elegantní, neboť pouhou změnou hodnoty tohoto parametru lze kola přidávat do scény a opět je odstraňovat. Podobně jednoduchý je také výpočet počtu kol, která jsou ještě k dispozici v paletě – stačí si nechat vrátit počet kol s hodnotou proměnné `l` rovnou -1.

Protože každé ozubené kolo musí mít nějakou velikost, je zde použita proměnná `r`. Ta určuje poloměr kola v pixelech. Bylo by možné určovat velikost kola také počtem zubů a z toho dopočítávat průměr, ale z důvodu jednoduššího vykreslování kol v editoru se dopočítává naopak počet zubů kola z jeho poloměru. Pro úplnost uvádím, že poloměr u ozubeného kola je vzdálenost od středu do poloviny výšky zubu.

Struktura `colorVector` bude zmíněna na konci kapitoly.

Příznak `locked` udává, zda je uživateli povoleno měnit polohu kola. V případě nastavení na hodnotu false je s kolem možno pohybovat libovolně, což znamená i jeho odstranění ze scény a umístění do palety nástrojů. U běžné instance kola je to naprosto v pořádku, ovšem u některých klíčových kol je nutné nastavit hodnotu true.

Proměnná `offset` ovlivňuje výchozí pootočení kola v času nula. Počty zubů jsou záměrně voleny tak, aby se jednalo o násobky čísla čtyři. Při hodnotě této proměnné false se kolo vykreslí tak, že osy x a y prochází vždy středem zubu. Pokud je však nastavena hodnota true, kolo je pootočeno o polovinu vzdálenosti mezi zuby a osy tak procházejí mezerami mezi zuby.



Obrázek 24: U levého kola je proměnná offset nastavena na false, u pravého na true

Algoritmus zajišťuje, aby sousední kola měla nastavenou opačnou hodnotu této proměnné. Výsledkem pak je fakt, že do sebe zuby kol správně zapadají.

Desetinná hodnota `speed` určuje rychlost otáčení kola. Aktuální pootočení kola se počítá vynásobením této hodnoty aktuálním časem simulace. Poté se přičte pootočení o půl zubu, pokud je hodnota proměnné `offset` true. Protože výpočty v plovoucí desetinné čárce nejsou dokonale přesné, je potřeba mít v porovnávacích algoritmech taková rozhodovací pravidla, která s malou přijatelnou odchylkou počítají.

`Powered` je také proměnná typu bool, která určuje, zda se jedná o hnací kolo. Vzhledem k tomu, jak funguje procházení grafu kol, není problém mít hnacích kol více než jedno. V zadáních levelů to však využito není.

Logická hodnota `goal` určuje, zda se jedná o kolo, kterého se týká pravidlo pro úspěšné dokončení levelu. V implementaci má tento parametr vliv jen na barvu vykreslení kola, aby jej mohl uživatel lehce identifikovat. Tento parametr je při zadávání pravidel pro ukončení levelu třeba pohlídat.

Proměnná `checked` není přímým parametrem kola, ale pomocná proměnná, která je využívána funkcí `checkGears()`, která zjišťuje, jakou rychlostí se mají kola točit, nastavuje jim parametr offset a kontroluje, zda nedošlo k zaseknutí převodovky.

Pomocná struktura colorVector vypadá následovně a vzhledem k její jednoduchosti asi není třeba podávat žádné doplňující vysvětlení.

```
typedef struct _color{
      int red;
      int green;
      int blue;
      int alpha;
      } colorVector;
```

5.2.2 Osa – speciální případ kola

Osa je reprezentována úplně stejnou strukturou jako ozubené kolo. Chová se také úplně stejně, jen se díky nulovému poloměru nemůže vázat k jiným kolům. Vazba na kola (a i osy) co jsou na stejných souřadnicích ve vedlejší vrstvě samozřejmě funguje.

5.2.3 Axle - Otvor pro osu

Otvor pro osu je řešen podobným způsobem jako kolo. Jen má méně parametrů:

```
typedef struct _axle{
      int x;
      int y;
      int l;

      }axle;
```

Parametry `x, y` a `l` určují pozici v souřadném systému. Stejně jako u kol je pozice na osách x a y zadána v pixelech a pozice na ose z pomocí relativní hodnoty. Na rozdíl od kol, se tyto elementy ve 3D vizualizaci nevykreslují vůbec. Není totiž třeba, aby se zde zobrazovaly. V 2D editoru jsou výhodné pro orientaci uživatele v tom, do kterých míst může osu kola umístit.

5.3 Kompozice objektů v editoru

V editoru je možné ovlivňovat jednotlivé objekty a modifikovat tím výsledný model. Objekty lze přesouvat po ploše i mezi vrstvami, odstraňovat je z pracovního prostoru a naopak přidávat. Vše pomocí jednoduchého a intuitivního ovládání.

5.3.1 Přidání objektu do scény

Přidání objektu do scény se provádí kliknutím na příslušné tlačítko v paletě nástrojů v dolní části okna s 2D editorem scény. Zavolá se příslušná funkce, která zjistí, zda je ještě kolečko s požadovaným poloměrem k dispozici (pozor, osa je speciální případ kola). To lze zjistit velmi jednoduše. Funkce prochází množinu všech kol a hledá takové, které má požadovaný poloměr a které se nachází ve vrstvě -1. Jakmile je takové nalezeno, je mu nastavena vrstva podle aktuální vrstvy a pozice tak, aby se jeho střed objevil uprostřed v dolní části pracovní plochy, těsně nad spodní hranou povoleného prostoru. To má výhodu takovou, že v případě omylu a kliknutí na nesprávné tlačítko stačí velice krátký pohyb myší pro umístění objektu zpět do palety.

5.3.2 Pohyb s objekty v editoru

Při přesouvání kola nebo osy v editoru je třeba tento objekt nejprve aktivovat. Při stisknutí levého tlačítka myši v oblasti pracovní plochy se zavolá funkce `getwheel()`, která na základě zaslaných parametrů (souřadnice `x` a `y`, aktivní vrstva), projde všechna kola co jsou k dispozici a jakmile najde to, u kterého odpovídá vrstva, zjistí vzdálenost kliknutého bodu od jeho středu pomocí Pythagorovy věty. Tu potom porovná s poloměrem testovaného kola. Pokud je vzdálenost bodu, v kterém se

nacházel kurzor v momentu stisknutí levého tlačítka myši a středu testovaného kola menší než jeho poloměr, funkce označuje toto kolo za aktivní a vrací řízení zpět.

Po dobu stisknutého tlačítka myši se kolo pohybuje zároveň s kurzorem myši, ale jsou zde jistá omezení. Při testování polohy středu kola se přihlíží k jeho vzdálenosti od potenciálních umístění (entit typu `axle`). Při vzdálenosti menší, než je určena konstantou `snap`, dojde k přichycení středu kola (nebo osy) na požadované místo. To umožňuje komfortnější ovládání, kdy uživatel nemusí „lovit“ umístění objektu na pixel přesně.

Zároveň je ověřováno, zda se na aktuálním místě kolo vůbec může nacházet. Je zde testováno, zda již kolo nezasahuje mimo povolený prostor v editoru. Toho je dosaženo zjišťováním vzdálenosti středu kola od hranice okna. To je porovnáváno s poloměrem kola a v případě, kdy je tato vzdálenost menší než poloměr, systém nenechá kolo překreslit na novou pozici. Tento test je prováděn u obou bočních a horního okraje. Dolní okraj tímto způsobem řešen není. V momentě, kdy uživatel umístí kolo do takové pozice, kdy střed kola je vykreslován pod pracovní plochou editoru, tedy kolo vlastně přemístí nad menu, dojde k odstranění kola ze scény. Kolo se tak dostane zpátky do palety nástrojů.
V první části aplikace, tedy při sestavování modelu systémů převodů uživatelem dojde k umístění entity typu kolo v prostoru na místo určené kurzorem myši. Vykreslovat se bude podle aktuální polohy kurzoru. Teprve poté, co uživatel zvolí umístění kola a potvrdí kliknutím, testovací procedura zanalyzuje, zda je zde možné kolo vůbec umístit. Nejprve vyhodnotí vzdálenost středu kola od některé z povolených pozic (projde pole souřadnic s otvory pro osu) a pokud zjistí vzdálenost menší, než jaká bude dána konstantou (vhodná velikost konstanty bude určena během testování), otestuje ostatní kola ve scéně na kolize. Tyto budou zjišťovány následujícím způsobem.

5.3.3 Vzájemná kolize kol

Pro nově přidávané kolo proběhne kontrola kolize následovně:

1. Načti další kolo z ostatních, pokud žádné není, není kolize.
2. Zjisti, zda leží ve stejné rovině, pokud ne, pokračuj dalším kolem od bodu 1.
3. Zjisti, zda vzdálenost středů není menší než součet poloměrů, pokud ne, pokračuj dalším kolem od bodu 1.
4. Kolize detekována.

Tímto způsobem lze jednoduše detekovat kolize s ostatními koly.

5.3.4 Umístění do prostoru

Pokud všechny detekce skončí bezchybně, nově přidané entitě typu *kolo* se nastaví pozice středu v prostoru a vytvoří se binární relace s těsně sousedními koly (pokud nějaké existují). Tyto budou

realizovány grafem s ohodnocenými hranami, kde uzly reprezentují kola a hrany jejich mechanické spojení. Hodnota hrany je rovna převodovému poměru.

5.3.5 Odstranění objektu z pracovní plochy

Pouhým přetažením objektu dolů, tak aby se střed nacházel pod dolní hranicí pracovní plochy, dojde k jeho automatickému přemístění do palety.

5.4 Inicializace a běh

Při spuštění aplikace se nejprve provede funkce `main()`. Ta pak volá další funkce. Nejprve proběhne nastavení OpenGL, pak se inicializuje počáteční stav objektů na scéně a v paletě. V dalších krocích je vytvořeno okno pro editor. K němu jsou poté přiregistrovány funkce, které jsou automaticky volány při události nad oknem.. Podobný způsobem je pak aktivováno druhé okno, použité pro vykreslování 3D modelu. Po úspěšné inicializaci se spustí nekonečná smyčka simulace.

5.4.1 Inicializace

Při inicializaci jsou volány funkce `wheelInit()` a `init3d()`. Funkce `init3d()` realizuje nastavení pro okno s vizualizací. Nastavuje správný stínovací model, složky barev pro materiály, způsob vykreslování, barvu světla, jeho pozici a tak dále.

Zajímavější je funkce `wheelInit()`, která v průběhu implementace realizovala mnoho věcí a měla sama několik desítek řádků kódu. Později se ale, vlivem zapouzdřování některých procesů a zobecňováním kódu, zkrátila na následující stav:

```
void wheelInit(){
       setWheelColors();
       setLevel(1);
}
```

Podíváme se blíže na tyto dvě funkce, které jsou při inicializaci okna pro editor volány.

5.4.1.1 Funkce setWheelColors()

Tato funkce je v podstatě realizací číselníku, který určuje barvu kola podle jeho poloměru. Jednotným obarvením kol stejného poloměru je dosaženo lepší přehlednosti v editoru. Barva kola ovlivňuje pouze jeho reprezentaci v editoru, nikoli v simulaci.

5.4.1.2 Funkce setlevel()

Výsledkem zobecňování původních ručně zadávaných pravidel pro jednotlivé levely je právě tato funkce. Požaduje jediný parametr a to je číslo aktuálního levelu. Tato funkce je umístěna v souboru

levels.cpp a pomocí dalších specializovaných funkcí přímo ovlivňuje kola na scéně, jejich parametry, počet, typ kol v paletě a podobně. Dále tato funkce umí nastavit podmínky, které je třeba splnit, aby byl level uznán jako úspěšně dokončený.

5.4.2 Okno editoru

Okno editoru získá při vytvoření jednoznačný window handler `window1`, pomocí kterého lze na tuto instanci okna odkazovat. Při vytváření má nastaveny výchozí rozměry podle globálních proměnných `windoww` a `windowh`, dále výchozí polohu na obrazovce danou vzdáleností od horní hrany a od levé hrany pomocí globálních proměnných `windowtop` a `windowleft`.

K oknu jsou připojeny funkce, které se volají při událostech. Protože se jedná o velice důležitou část aplikace, rozhodl jsem se jim věnovat samostatné podkapitoly.

5.4.2.1 Funkce onDisplay()

Tato funkce má za úkol vykreslovat kompletně celý obsah okna pro 2D editor. Nejprve se vymaže obsah okna a barva pozadí je nastavena na černou. Následují tři for-cykly, zajišťující vykreslení kol a os.

První smyčka má za úkol vykreslit všechna kola nacházející se v aktuální vrstvě. Jednoduše prochází všechna aktivní kola a jakmile zjistí shodnou hodnotu vrstvy s aktuální vrstvou, volá funkci `drawWheel()`, aby bylo kolo vykresleno.

Kvůli zachování přehledu o situaci mezi vrstvami se do editoru vykreslují i obrazy kol, nacházejících se v sousedních vrstvách. To má za úkol druhá smyčka s použitím for-cyklu v této funkci. Podobně jako předchozí zmíněná, prochází kola. Jakmile zjistí absolutní hodnotu rozdílu mezi vrstvou kola a aktuální, nechá vykreslit jeho obrys zavoláním funkce `drawGhost()`.

Třetí for-cyklus vykresluje místa s možným uložením os kol. Tyto prvky jsou dány strukturou typu axle a vykreslování se provádí pomocí funkce `drawAxle()`. Zjišťování, zda se má objekt vykreslit, probíhá identicky jako u první smyčky.

Po vykreslení všech prvků, které daná scéna na aktivní vrstvě obsahuje (plus obrazy kol na vrstvách sousedících), se zavolá funkce `drawGUI()`, která je zodpovědná za vykreslení celého GUI.

Tím je dokončeno vykreslování veškerého obsahu okna s editorem a již stačí překreslit scénu. Je zde použito double bufferingu pro zabránění blikání obrazu. Pro tyto účely stačí zavolat následující kód:

```
glFlush();    // provedeni a vykresleni zmen
glutSwapBuffers();
```

Protože při změně obsahu okna s editorem může dojít i ke změně obsahu okna s 3D vizualizací, je stejný kód volán i pro druhé okno. Přepnutí aktivního okna je realizováno přes handler okna.

5.4.2.2 Funkce onResize()

Tato funkce není příliš zajímavá nebo komplikovaná, má v podstatě za úkol jen zabezpečit správné vykreslení obsahu okna, pokud uživatel změní jeho velikost. Přenastavuje hodnoty v globálních proměnných `windoww` a `windowh`, které jsou potom následně používány pro ohlídání hranic pracovního prostoru.

5.4.2.3 Funkce onKeyboard()

Aplikaci lze částečně ovládat i klávesnicí. V případě, že je aktivní (má focus) okno s editorem, pak se o obsloužení požadavků zasílaných z klávesnice stará právě tato funkce. Konkrétní seznam ovládacích prvků je zmíněn v kapitole 4.

5.4.2.4 Funkce onMouse()

Pomocí této funkce jsou zpracovávány signály z tlačítek myši. Funkce je velice důležitá, neboť právě zde se pomocí algoritmů rozhoduje, o jakou činnost měl uživatel zájem a co se tedy vlastně má vykonat. Protože OpenGL a winApi mají rozdílný pohled na číslování osy y (tedy zda pixel se souřadnicemi [0,0] má být v levém horním a nebo v levém dolním rohu okna). Jednodušší variantou bylo otočit vnímání pozice myši u winAPI pomocí jednoho jednoduchého řádku kódu.

```
y = windowh-y;
```

Pak již následuje sekvence rozhodovacích větvících příkazů if nebo if-else. Pomocí nich lze rozeznat, zda požadavek poslaný myší (nebo jiným polohovacím zařízením) je charakteru požadavku na GUI, nebo zda jde o úkon související s nějakým kolem nebo osou. Protože krom informace o tom, zda jde o stisk nebo uvolnění tlačítka a o které tlačítko se jedná, máme i informaci o aktuální pozici kurzoru, lze jednoduše určit typ požadavku.

V první fázi je testováno, zda nešlo o kliknutí do menu. Menu se nachází v dolní části okna, takže stačí zkontrolovat, zda jde o stisk tlačítka a jestli se kurzor nachází níže, než je výška menu. Jestli jde o kliknutí a ne uvolnění tlačítka myši je nutné testovat z toho důvodu, že přetažením objektu na úroveň menu a uvolněním tlačítka myši se tento odstraní ze scény a umístí do palety nástrojů. Při zjištění, že skutečně šlo o kliknutí do menu, zavolá se funkce `menuClicked()` s předáním souřadnic kliknutí a funkce předá řízení zpět.

Druhá fáze je zde pro případ, že uživatel právě splnil úkol levelu a chystá se přejít o úroveň výš. V takovou chvíli se v horní části okna zobrazuje textový pokyn, na který má uživatel kliknout. Je tedy třeba otestovat, zda došlo ke splnění úkolu, tedy zjištění, jakou hodnotu vrátí funkce `isLevelCompleted()` , pak kontrola, zda byl kurzor ve správných místech (maximálně 20 pixelů od horní hrany okna).

Když se eliminují speciální případy, lze očekávat, že uživatel kliknul do pracovního prostoru editoru. Testujeme, zda příchozí akce je stisknutí myši. Pokud ano, pomocí funkce `getWheel()` zjistíme, zda je pod kurzorem nějaké kolo a jaké je jeho ID. Zjišťování probíhá na základě předaných souřadnic a aktuální vrstvy. Pokud se uživatel kliknutím „netrefil“ na žádné z kol, vrací funkce hodnotu -1 a nic dalšího se neděje. V případě, že funkce vrátí ID existujícího kola (tj. uživatel opravdu kliknul do prostoru kola), dojde k několika věcem. Protože se právě začíná modifikovat objekt, do globální proměnné `selectedWheel` se nastaví jeho id. Program nejprve zazálohuje aktuální barvu kola. Pak nastaví novou barvu kola. Jaká to bude, záleží na hodnotě proměnné fixed u zvoleného kola. V případě negativní hodnoty, se barva nastaví na světle šedou, aby uživatel viděl aktuální kolo zvýrazněné. V případě pravdivé hodnoty se barva nastaví na červenou – upozornění uživateli, že manipulovat s tímto kolem není možné.

Nakonec se obslouží požadavek zaslaný uvolněním tlačítka myši. V případě, že bylo nějaké kolo aktivní, tedy proměnná `selectedWheel` neobsahuje hodnotu -1 ale ID kola, obnoví se tomuto kolu původní barva, protože právě přestává být aktivním.

Dále se testuje, zda nedošlo k uvolnění tlačítka myši v momentě, kdy byl kurzor níž, než je horní hranice menu. V případě, že ano, pak šlo o odstranění kola z pracovního prostoru. Aktivnímu kolu se nastaví hodnota vrstvy na -1, čímž automaticky mizí do palety.

Protože evidentně došlo ke změně stavu v modelu, následuje kompletní kontrola a přepočítání vztahů mezi koly. V první fázi se všem kolům nastaví příznak `checked` na false. Následuje nastavení nulové rychlosti všem kolům s výjimkou těch hnaných. Nakonec se spustí rekurzivní funkce `checkGears()` pro všechna hnací kola.

Před opuštěním funkce je ještě proměnná `selectedWheel` nastavena na hodnotu -1, protože po uvolnění tlačítka myši není žádné kolo vybráno.

5.4.2.5 Funkce onMouseMotion()

Zatímco předchozí funkce zabezpečovala stav tlačítek myši, tato funkce realizuje změny, které jsou potřeba provádět při pohybu myši. Podobně jako u předchozí funkce se nejprve otočí osa y pro winAPI.

Nejprve je zjištěno, zda aktivní kolo není zamčené vůči posunu. V případě, že ano, ke změnám nedochází. Pokud tato situace nenastane, program si vypočítá požadované nové souřadnice středu kola, s kterým je právě pohybováno. Není možné přímo změnit pozici kola, protože by se vynechala kontrola, zda se na požadovaných souřadnicích vůbec může kolo nacházet. Dále by nefungovalo přichytávání kol na předchystaná místa.

V první fázi se prochází pole s prvky typu axle a v případě detekce malé vzdálenosti (nastaveno konstantou), se požadované nové souřadnice středu kola přenastaví na souřadnice prvku axle. Ani zde ještě nedojde k fyzické změně umístění středu kola, protože stále nejsou ošetřeny všechny stavy, ke kterým by mohlo dojít.

V druhé fázi probíhá kontrola, zda v novém umístění nebude docházet ke kolizi s ostatními koly. K tomu slouží funkce `isCollision()`, která zjišťuje, zda ke kolizi dochází a na základě toho vrací true nebo false. Tato funkce funguje tak, že zjišťuje u všech dvojic kol ve vrstvě, jakou mají vzdálenost středů od sebe a tu porovnává se součtem poloměrů. Vzdálenost mezi středy nikdy nesmí být menší. V případě detekce kolize souřadnice kola nemění a jen zavolá překreslení obrazovky.

Třetí fáze kontroluje, zda se kolo nedostalo svou částí mimo povolený prostor. Jednoduchá kontrola spočívá v porovnávání vzdálenosti středu kola od hranice okna s poloměrem. V případě, že dojde ke kolizi, změna pozice středu kola se nekoná.

Pokud však k žádné kolizi nedojde, souřadnice středu kola jsou posunuty na požadované souřadnice. Dále dojde k vynulování pozice myši a překreslení okna.

5.4.3 Okno vizualizace

Okno vizualizace má window handler `window2`, pomocí kterého se na něj lze odkazovat. Parametry velikosti a umístění na obrazovce jsou zde určeny globálníma proměnnýma `window2w`, `window2h`, `window2top` a `window2left`. Stejně jako první okno, jsou zde připojeny funkce pro obsluhu příchozích signálů. Ty budou přiblíženy v následujících podkapitolách.

5.4.3.1 Funkce onDisplay3d()

Funkce, která se stará o vykreslování obsahu okna s 3D vizualizací. Na začátku se vynuluje jednotková matice transformací, dále se nastaví barva materiálu, povolí se stínování, nastaví pozice světla a podobně. Pro jednodušší výpočet normálových vektorů se povolí jejich automatická normalizace.

V dalším kroku se provede transformace souřadnic pomocí následujícího kódu.

```
glTranslatef(0.0, 0.0f, -800.0f);         /* posun modelu dale od kamery */
glTranslatef(xcamera, -ycamera, znew);    /* posun modelu podle pohybu mysi
*/
glRotatef(ynew, 1.0, 0.0, 0.0);           /* rotace objektu podle pohybu
kurzoru mysi */
glRotatef(xnew, 0.0, 1.0, 0.0);           /* rotace objektu podle pohybu
kurzoru mysi */
```

První transformace zajišťuje posunutí celého objektu simulace dále od kamery tak, aby se celý vešel do zorného úhlu. Při výchozím nastavení bez zásahu uživatele je toto jediná transformace, která se provede, protože následující parametry transformací mají jako výchozí hodnotu nastavenou nulu. To lze obnovit stisknutím tlačítka R při aktivním okně s 3D modelem. Následuje posun o uživatelsky zadané hodnoty. Ty vznikají při zoomování scény pomocí pravého tlačítka myši a nebo při posunu modelu v prostoru tažením myši se současně stisknutou klávesou shift. Poslední dvě rotace jsou

natočení celé scény podle nastavení od uživatele. Uživatel scénou otáčí pohybem kurzoru myši v okně s vizualizací při stisknutém levém tlačítku myši.

Následuje část kódu, která zařizuje vykreslování všech prvků 3D scény. K tomuto účelu využívá specializované funkce, které budou podrobněji popsány v kapitole 5.7.

```
glPushMatrix();
 glTranslatef(-windoww/2.0,-windowh/2.0,0.0f);
 for(int i=0; i<wheelCount; i++){
       if(wheels[i].l > 0){
             glPushMatrix();
             glTranslatef(wheels[i].x,wheels[i].y,layer*wheels[i].l);
             drawWheel(i);
             glPopMatrix();
             }
       }
 glPopMatrix();
```

Tento kód zabezpečuje vykreslování objektů do prostoru. Prvním úkonem je uložení aktuální transformační matice na zásobník. Na konci běhu této části kódu je tento stav opět obnoven. Hned po uložení aktuálního nastavení transformační matice na zásobník se provede posun systému o polovinu šířky okna na ose x v záporném smyslu a posun o polovinu výšky okna na ose y v záporném směru. Touto transformací je docíleno toho, že přímo proti kameře se nenachází bod [0,0] jak tomu před transformací bylo, ale bod, který je ve středu okna editoru. Tím je zajištěno, že model je umístěný uprostřed okna pro vizualizaci a ne jen v levé horní části. Kromě toho se tím posunul i střed otáčení modelu z bodu [0,0], tedy jeho levého dolního rohu, na prostředek. Tím už je prostředí připraveno k vykreslení všech kol.

Pomocí for-cyklu je procházeno jedno kolo po druhém a v případě zjištění kladné hodnoty v parametru layer (tedy kolo je umístěno ve scéně) je provedena transformace souřadného systému tak, aby místo pro vykreslení středu kola bylo v počátečních souřadnicích. Pak je zavolána funkce `drawWheel()`, která zajišťuje vykreslení 3D modelu kola. O tom více v kapitole 5.7. Před transformací je aktuální stav transformační matice umístěn na zásobník a po zavolání vykreslovací funkce je opět obnoven. Tím se značně zjednodušuje hlídání souřadnic.

Na konci funkce je voláno překreslení okna s využitím double bufferingu.

5.4.3.2 Funkce onReshape3d()

Při změně velikosti okna s 3D vizualizací se zavolá tato funkce. Rozměry tohoto okna nejsou pro výpočty příliš důležité, změna globálních proměnných však přesto probíhá. Hlavním úkolem funkce je při změně velikosti okna přepočítat perspektivu a FOV (field of view).

5.4.3.3 Funkce onMouseClick3d()

Nejprve se zjistí, o jakou činnost má vlastně uživatel zájem. Podobně jako u odpovídající funkce u okna s editorem se zde pomocí větvení zjišťuje o jaký stav tlačítek se vlastně jedná.

Nejprve proběhne test, zda není stisknuto tlačítko shift a zároveň nedošlo ke stisknutí levého tlačítka myši. Pokud tomu tak je, uživatel aktivoval posun modelu v prostoru a funkce nastaví příslušný parametr, aby následující funkce zpracovávající pohyb myši měla pokyn, jaké parametry z pohybu myši číst.

Druhé testování probíhá na stisk pravého tlačítka, které aktivuje zoomování. V případě pozitivní kontroly, jsou parametry pro změnu pohledu nastaveny na zoom.

Třetí pak kontroluje stisk levého tlačítka bez shiftu. Tak je aktivována rotace modelu kolem středu. Opět se zde nastaví odpovídající příznak.

U každé z předchozích tří větví je pak i testován příznak na uvolnění daného tlačítka myši. V případě detekce uvolnění tlačítka dochází k nulování příznaku pro funkci `onMouseMotion3d()` na takzvaný idle stav. Tím je zabezpečeno, že pohybem kurzoru při aktivním okně s vizualizací bez stisknutí tlačítka na myši ke změnám ve vyobrazení modelu nedochází.

5.4.3.4 Funkce onMouseMotion3d()

Na základě příznaku nastaveného předchozí funkcí myš přepočítává pohyb myši na požadované nové hodnoty parametrů, které se pak použijí pro transformaci matice při vykreslování modelu.

5.4.3.5 Funkce onKeyboard3d()

Funkce má podobný úkol jako její ekvivalent pro okno s editorem. Změna spočívá v tom, že při stisku klávesy R při aktivním okně s vizualizací dojde k vynulování uživatelských proměnných pro transformaci souřadnic. Tím se kamera nastaví do výchozí pozice.

5.5 Implementace kol

Ozubená kola a vztahy mezi nimi tvoří hlavní část aplikace. Proto jsem se rozhodl věnovat implementaci kol vlastní kapitolu. Ta bude obsahovat několik podkapitol podle tematického zaměření.

Struktura typu `wheel` reprezentuje jedno kolo. Pole těchto struktur `wheels[]` je v podstatě zásobníkem jednotlivých kol. Při inicializaci se nastaví maximální velikost tohoto pole podle konstanty `wheelCountMax`. Ta určuje také maximální možný počet kol ve hře. Proměnná `wheelCount` určuje aktuální počet kol ve hře a při inicializaci má hodnotu nula.

5.5.1 Vytvoření nového kola

Pro vytvoření nového kola je potřeba zavolat funkci `createWheel()`, která jedno kolo s požadovaným poloměrem vytvoří a přidá jej do palety nástrojů. Zároveň dojde k inkrementaci proměnné `wheelCount`. Kolo má po vytvoření nastaveny výchozí parametry, barvu má takovou, jaká odpovídá této velikosti kola.

5.5.2 Modifikace existujícího kola

Při vytváření levelů je potřeba některé parametry některých kol zadat explicitně. K tomu slouží funkce `setWheel()`, pomocí které lze kolu nastavit vše. Od pozice, vrstvy, přes poloměr, informaci zda bude uzamčené na pozici, až po rychlost a příznak, že se jedná o kolo hnací. Podobně lze nastavit kolu barvu pomocí funkce `setWheelColor()`. V obou případech se jako identifikátor používá ID kola.

5.5.3 Odstranění všech kol

Pro odstranění všech kol stačí nastavit proměnnou `wheelCount` na nulu. O to se stará funkce `removeWheels()`. Pro opětovné přidání kola do systému je nutné využít funkci `createWheel()`, která nastavuje parametry kola na výchozí hodnoty, takže není potřeba parametry kol nulovat. Je to podobné mazání souboru z pevného disku, kde stačí pouze zahodit ukazatel na místo, kde byl soubor uložen, ale není nutné fyzicky mazat jeho data.

5.5.4 Zamknutí všech kol ve scéně proti pohybu

Pomocí funkce `fixAllWheels()` je možné všechna kola, která se nacházejí ve scéně, zamknout proti posuvu. To je realizováno nastavením jejich parametru `fixed` na hodnotu true. Této funkce je využito na začátku čtvrtého levelu, kdy součástí zadání je stav systému na konci levelu předchozího.

5.5.5 Zjištění počtu kol v paletě

Jednoduchým průchodem všech kol ve hře a porovnáním jejich poloměrů s požadovanou hodnotou a kontrolou, zda se nacházejí na vrstvě -1, lze sečíst počet kol o daném poloměru. Takového postupu využívá funkce `inStack()`, která na vstupu požaduje poloměr kol, které má v paletě sečíst. Tato funkce se využívá při vykreslování GUI ke zobrazení počtu kol k dispozici.

5.5.6 Přesun kola z palety do scény

Použitím funkce `addWheel()`, volané s parametrem požadovaného poloměru, lze přidat kolo z palety do scény. Volání této funkce je přiřazeno k tlačítkům v GUI. Postup přidání kola je

jednoduchý. Smyčkou for-cyklu je procházeno pole všech kol ve hře. Pokud je nalezeno kolo, jehož poloměr odpovídá požadovanému a navíc se nachází ve vrstvě -1 (tedy v paletě nástrojů), je jeho příznak vrstvy nastaven na aktuální vrstvu a pozice středu je umístěna do poloviny šířky okna, těsně nad menu.

5.6 Simulace

Hlavním úkolem návrhu i implementace celé aplikace bylo navrhnout systém, který bude schopen počítat vzájemné ovlivňování kol v celé soustavě převodů tak, aby se vykreslená scéna chovala stejně, jako kdyby byla reálná. Bylo potřeba vytvořit mechanismus schopný hlídat vztahy mezi jednotlivými koly, hlídat převodový poměr a podle toho dopočítávat rychlost otáčení. Model je potřeba modifikovat, když se změní umístění kola, které na to má vliv. Protože se zde mohou vyskytovat kruhy, není možné pro modelování vztahů použít strom. Ve smyčkách je ještě třeba hlídat to, jestli nedošlo k sestavení takového převodu, který blokuje sám sebe. Realizace pomocí grafu s ohodnocenými uzly se zdála být výhodnou, nicméně při implementaci jsem narazil na několik zásadních problémů.

5.6.1 Vztah mezi koly

Dvojice kol mezi sebou může mít tři druhy vztahu. Prvním z nich je jakási nulová relace. Tato kola se navzájem žádným způsobem neovlivňují, nijak se nedotýkají.

Druhým typem vztahu je varianta, kdy jsou ve stejné vrstvě a vzdálenost jejich středů je rovna součtu jejich poloměrů. V takovou chvíli zapadají zuby těchto kol do sebe a podle poloměrů, respektive počtu zubů, lze vypočíst převodový poměr. V tomto případě bude převodový poměr vždy záporné číslo, protože dvě ozubená kola spojená tímto způsobem se otáčí vždy opačným směrem. Při výpočtu převodového poměru záleží na pořadí kol, protože převodový poměr prvního kola k druhému je převrácenou hodnotou převodového poměru kola druhého k prvnímu. Tato čísla se rovnají pouze pokud obě kola mají stejný poloměr (to znamená i stejný počet zubů). V takovém případě je převodový poměr roven číslu -1.

Při modelování vztahů mezi koly pomocí nějaké reprezentace grafu vzniká problém. Hrany mezi uzly (odpovídající převodovým poměrům mezi koly) nemohou být neorientované a musí být ohodnocené. Při každé změně v modelu převodovky je pak nutné překontrolovat související relace mezi koly a zjistit, zda všechny vytvořené hrany stále existují, zda žádným nově vzniklým relacím nechybí odpovídající hrana v grafu, zda počet uzlů odpovídá počtu kol v modelu a jestli jsou všechny hodnoty u hran v pořádku.

V tomto vztahu kol se nemění obvodová rychlost, takže všechna propojená kola na jedné vrstvě mají shodnou obvodovou rychlost. To je ale pro účely modelování nevýhodný údaj, takže všechny výpočty stejně probíhají podle úhlové rychlosti.

Třetím typem vztahu mezi koly je takový, kdy kola leží na jedné ose, kterou jsou obě pevně spojena. V takovémto případě se úhlová rychlost obou kol rovná. Při realizaci takovéhoto vztahu pomocí grafu nevyvstává problém s orientací hrany grafu, protože v obou směrech je hodnota převodového poměru rovna jedné. V praxi to v programu znamená to, že pokud kola mají shodné souřadnice svého středu a nacházejí se na vrstvách přímo sousedních, mají společnou osu a tedy i stejnou úhlovou rychlost.

5.6.2 Výpočet vztahů

Pro zjištění vztahu mezi dvěma koly se využívá dvou funkcí. Jedna zjišťuje, zda se kola dotýkají, druhá, zda nemají společnou osu. Obě funkce vrací hodnotu bool.

Funkce `isConnected()` přijímá jako argumenty dvě ID od kol, které má zkontrolovat. Zjistí, zda kola jsou ve stejné vrstvě. Pokud ne, vrací false. Spočítá vzdálenost mezi středy obou kol a tu porovná se součtem jejich poloměrů. Pokud se tyto hodnoty rovnají, funkce vrací hodnotu true. Funkce pouze zjišťuje, zda se kola dotýkají (tj vzájemně zapadají zuby), ale převodový poměr nepočítá.

Podobně pracuje funkce `isOnAxle()`, která rovněž zjišťuje vztah mezi dvěma koly, s jejichž ID v parametrech je volána. Nejdříve porovná souřadnice středů obou kol. Pokud nejsou shodné, vrací false. Jinak funkce zjistí, zda se kola nachází na sousedních vrstvách a v případě, že ano, vrací true. Podobně jako u předchozí funkce, ani zde taky není řešen převodový poměr. Tady by ale ani být řešen nemusel, protože kola spojena touto formou mají převodový poměr roven jedné a shodnou úhlovou rychlost.

5.6.3 Model grafu

Při každém uvolnění tlačítka myši - pokud bylo nějaké kolo aktivní - probíhá kontrola systému převodů. Tato kontrola se spouští ve funkci `onMouse()`. Před samotným spuštěním kontroly se všem kolům ve hře nastaví rychlost otáčení na nulu. Výjimku mají jen kola hnací, u těch je rychlost zachována. Dále se u všech kol nastaví parametr `checked`, sloužící právě pro účely modelování grafu, na hodnotu false. Následuje for-cyklus procházející všechna kola ve hře. Jakmile zjistí, že se jedná o hnací kolo, spustí funkci `checkGears()` s ID dotyčného kola ve vstupním parametru volání. Po tomto volání nenásleduje `return` nebo `break`, takže funkce `checkGears()` se zavolá na všechna hnací kola, co jsou v systému. To činí systém dostatečně univerzálním. Ač v žádném levelu není více hnacích kol než jedno, v případě vytvoření nového levelu s více hnacími koly nevznikne pro algoritmus zpracování vztahů žádný problém.

5.6.4 Funkce checkGears()

Tato funkce je stěžejní částí algoritmu celé aplikace. Po dlouhém přemýšlení a několika pokusech jak vytvořit univerzální algoritmus pro modelování vztahů mezi koly, ale také nastavení jejich rychlostí, počátečních pootočení a kontrole, zda nedošlo v soustavě převodů k vytvoření blokující smyčky, se nakonec vyplatilo použít rekurzivní řešení.

Funkce přijímá jediný parametr typu integer – ID kola určeného ke kontrole. Postup ve funkci vysvětlím v následujícím textu.

První, co funkce při spuštění udělá, je nastavení hodnoty `checked` u právě kontrolovaného kola na true. Tím se zamezí případnému zacyklení. K tomu více dále v textu této kapitoly. Hned při spuštění funkce také následuje inicializace proměnné `ratio` typu float na hodnotu 1.0. Touto proměnnou je reprezentován převodový poměr.

Následuje for-cyklus, který je volán pro každé kolo, které je ve hře. V něm se zjišťuje, zda některé z kol ve hře není propojeno s právě kontrolovaným kolem. Testují se tedy všechna kola postupně.

Kolo, které je aktuálně na řadě ve for-cyklu, se otestuje na dotek s kolem testovaným pomocí funkce `isConnected()`. Pokud funkce vrátí hodnotu true, ozubená kola do sebe zapadají. Funkce nastaví kolu opačnou hodnotu parametru offset, než má testované kolo. Tím je zajištěno správné pootočení kol a správné zapadání zubů.

Pokud má testované kolo nenulovou rychlost, provede se výpočet převodového poměru. Tato pojistka je zde kvůli zamezení potencionálního dělení nulou. Výpočet převodového poměru probíhá jednoduše podělením poloměru aktuálního kola podle počítací proměnné ve for-cyklu poloměrem testovaného kola a vynásobením mínus jedničkou. V tomto případě je převodový poměr záporný. Nyní se vypočítá požadovaná rychlost. V případě, že aktuální kolo již bylo zkontrolované (tedy má proměnnou `checked` hodnoty true), porovná se jeho rychlost s rychlostí dopočítanou. V případě, že se tyto rychlosti liší, byla detekována smyčka blokující pohyb soustavy převodů a animace se zataví. V tomto případě však nešlo použít pouze obyčejné porovnávání dvou hodnot, neboť výpočty v plovoucí desetinné čárce nejsou vždy naprosto přesné. Použití datového typu integer se pro tyto účely nehodí, neboť je nutné používat i převrácené hodnoty celých čísel. Tohle v průběhu implementace způsobovalo velké problémy, kdy se animace zastavovala bez zjevné příčiny. Problém se vyřešil jednoduše přidáním tolerance do porovnávání rychlostí.

Následuje nastavení rychlosti aktuálního kola na hodnotu vzniklou vynásobením rychlosti testovaného kola dříve spočítaným převodovým poměrem. Nakonec se na aktuální kolo, u kterého byl zjištěn dotek s testovaným kolem, zavolá funkce `checkGears()`. To neplatí, pokud má aktuální kolo hodnotu `checked` rovnu true. Tím se zajišťuje zabezpečení rekurzivního procházení před zacyklením.

Další for-cyklus prochází všechna kola ve hře a kontroluje je na společnou osu s testovaným kolem. K tomu se využívá funkce `isOnAxle()`. Tato funkce se zavolá s dvěma parametry: ID aktuálního kola a ID testovaného kola. V případě návratové hodnoty true se zjistí, zda má testované kolo nenulovou rychlost, pokud ano, zjistí se hodnota proměnné u aktuálního kola. Pokud je tato hodnota true, provede se test, zda má aktuální kolo stejnou rychlost jako testované kolo. Opět je zde potřeba použít nějakou toleranci, protože rychlosti si často neodpovídají úplně přesně. V případě, že se rychlosti liší, je animace zastavena. Pak se nastaví rychlost aktuálního kola na shodnou s rychlostí testovaného kola, pokud není aktuální kolo hnacím kolem. Nakonec se zjistí, zda aktuální kolo již bylo kontrolováno (hodnota parametru `checked`) a pokud ne, zavolá se na něj funkce `checkGears()`.

Takovéto rekurzivní řešení je výhodné především v tom, že není třeba implementovat složitou datovou strukturu reprezentující graf vztahů mezi jednotlivými koly. Rekurzivní řešení pracuje samo a efektivně.

5.7 Vykreslování objektů

Vykreslování objektů probíhá na základě dat získaných výpočty simulace.

Kola se během vývoje vykreslovala pouze jako válce s malou výškou oproti poloměru. Výška byla u všech stejná, poloměr rozdílný. Pro jednoduché rozpoznání směru a rychlosti otáčení byly jejich podstavy rozděleny dvěma kolmými průměry a vybarveny šachovnicovým způsobem.

Později byly doplněny funkce, které na základě parametrů (barvy a vzhledu) vykreslují realistická ozubená kola. Pro vykreslení ozubeného kola je použito opakované volání funkce, která na základě parametrů vykreslí výseč kola. Otáčením funkce kolem osy kola tak vnikne kolo celé.

5.7.1 Vykreslení kola ve 3D

O vykreslení kola ve trojrozměrné podobě se stará funkce `drawWheel()`. Ta je pomocí for-cyklu volána z funkce `onDisplay3d()`. Protože před každým voláním proběhne nastavení souřadného systému tak, aby se kolo vykreslilo středem do počátku souřadného systému, není třeba řešit žádné koordináty. Funkce očekává na vstupu jeden parametr a to ID kola, které má být vykresleno. To bude ve výsledku vykresleno s příslušným poloměrem a z toho vyplývajícím počtem zubů na počátečních souřadnicích. Osa kola leží na ose z.

Funkce nejdříve spočítá počet zubů kola na základě zjištěného poloměru. Převodní poměr funguje tak, že kolo o poloměru 100 má 32 zubů. Každý zub je vymodelován z osmi podčástí. Z toho vyplývá, že kolo se skládá z 8 krát počet zubů částí. Tato hodnota je uložena v proměnné s názvem max. Rozdělení zubu na 8 částí je výsledkem bádání, jak vykreslovat zub tak, aby působil co nejvíce

realisticky. V první části z osmi je náběhová hrana, další dvě části jsou horní ploška zubu. Ve třetí části je sestupná hrana a poslední čtyři části tvoří plošku mezi dvěma zuby. Počet těchto částí se postupně měnil a až u počtu osmi vypadaly zuby kola realisticky. Na ilustraci je znázorněno, jak je zub vlastně realizován. Jedná se o jeden kompletní zub i s navazující mezerou rozvinutý do roviny.

Obrázek 25: Ilustrace osmi částí zubu kola

Dále se nastaví difúzní složka materiálu, podle toho, o jaké kolo se jedná. Jinou barvu než standardní má kolo, které je uzamčené proti pohybu (`fixed`) , jinou barvu má kolo, které souvisí se splněním úkolu levelu (`goal`). Dále se ještě mění barva těch kol, která jsou v právě aktivní vrstvě. To z důvodu lepší orientace uživatele.

Na základě hodnoty proměnné `offset` u vykreslovaného kola se nastaví, zda se kolo bude vykreslovat nahoru zubem a nebo mezerou mezi zuby. Tím je zařízeno pěkné zapadání zubů do sebe u dvojice kol, kde kola mají opačnou hodnotu v proměnné `offset`.

Následuje výpočet aktuálního pootočení kola podle času. Proměnná `position` (pootočení kola ve stupních) se spočítá jako součin globální proměnné `time` a rychlosti příslušného kola. Dojde k pootočení souřadného systému kolem osy z o požadovaný počet zubů. Není třeba ukládat stav jednotkové matice, protože při volání funkce `drawWheel()` se matice vrací do původního stavu.

Následuje část kódu, která za pomoci for-cyklu vykreslí jednu stranu ozubeného kola. Hodnota `offset` nabývá hodnot 2 nebo -2. Rozdíl mezi těmito čísly je 4, což je polovina součástí z kterých je vykreslený zub.

```
for(int i=offset; i<=max+offset; i+=8){...}
```

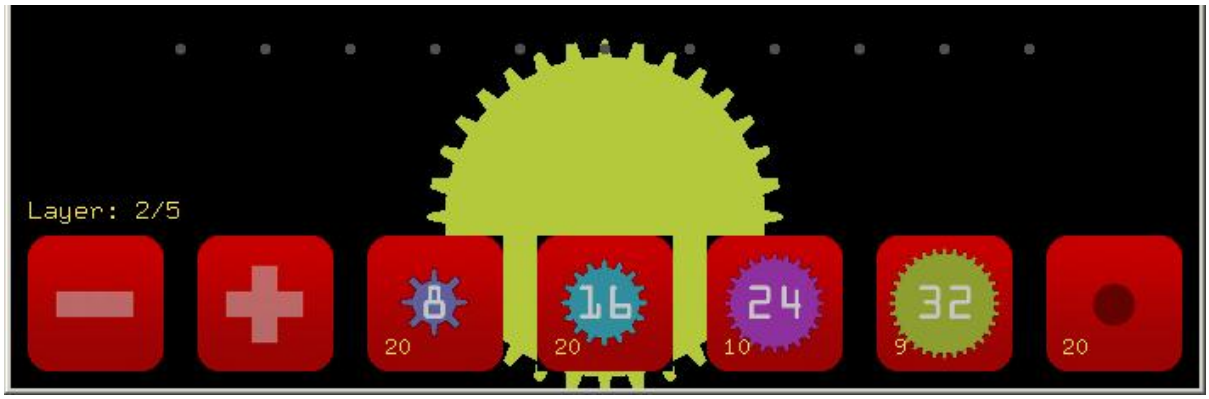
Počítací proměnná i se inkrementuje vždy o 8 a v kódu se vykresluje vždy 8 trojúhelníků po sobě. Přesné souřadnice se počítají pomocí goniometrických funkcí. Podobným způsobem se vykresluje i styčná plocha ozubeného kola a osa kola. U všech vertexů je dopočítáván normálový vektor, aby stínování fungovalo bez problémů.

5.8 GUI

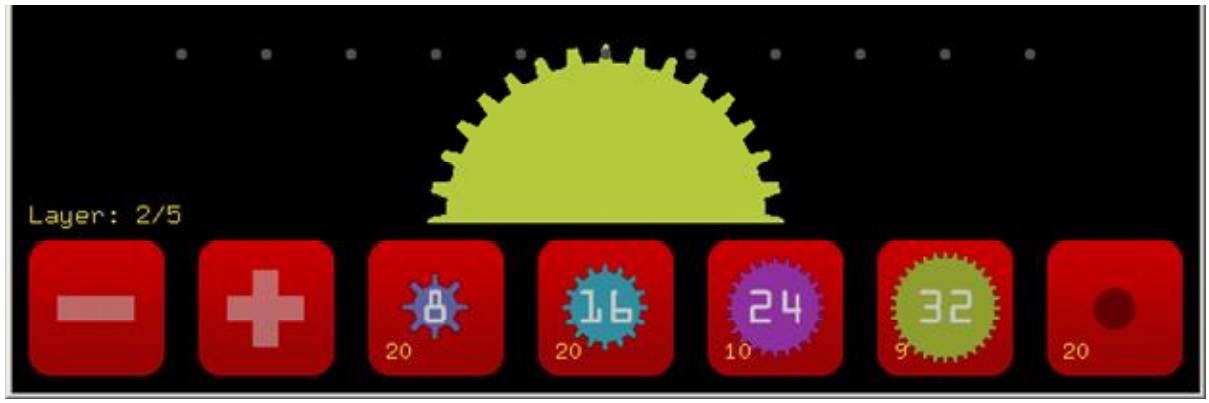
Pro vykreslení GUI je použito několik funkcí, které jsou umístěny ve zvláštním souboru. Samotné vykreslení GUI je zcela autonomní a spouští se zavoláním hlavní funkce. Pro přehlednost ale nejprve

popíši podpůrné části kódu. Pro texturování je využito funkce `LoadTextureFromBMP()` , kterou jsem převzal z cvičení předmětu PGR. Funkce načte soubor ve formátu BMP a zpracuje jej pro použití v OpenGL. Tato funkce je volána pokaždé, kdy je potřeba před vykreslením otexturovaného objektu nahrát novou texturu. To také umožňuje použít pouze jednu proměnnou pro uchování textury. Před překreslením se stejně obsah proměnné vždy změní. Původní obava ze zbytečného zpomalování běhu programu tím, že při každém vykreslování GUI bude nutné několikrát otevírat soubor s texturou z disku, se nakonec nepotvrdila, takže nebylo třeba ukládání textur nějak zvláště optimalizovat.

Funkce `drawRectangle()` vykresluje obdélník na určených souřadnicích. Ten je pak potáhnut aktuálně nastavenou texturou. Funkce je nejprve volána s ještě nenastavenou texturou pro vykreslení černého pozadí menu. Ačkoli se toto pozadí na první pohled nijak neprojeví, po přidání nového kola do scény zabraňuje jeho prosvítání mezi ovládacími tlačítky.



Obrázek 26: Kolo prosvítající mezi tlačítky

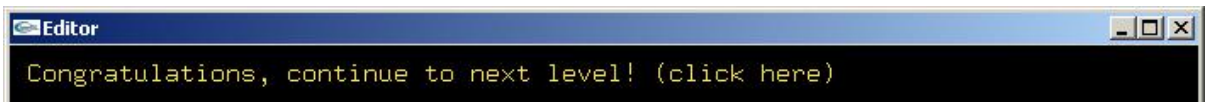


Obrázek 27: Menu s vykresleným pozadím

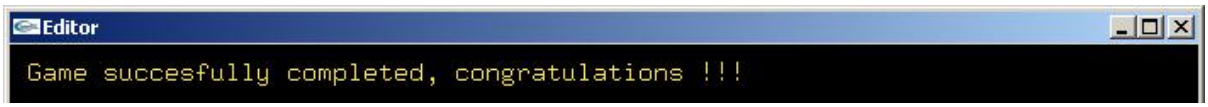
Následně je tato funkce volána ještě sedmkrát pro vykreslení ovládacích tlačítek. Před každým voláním je načtena příslušná textura. Textury byly vytvořeny pomocí Photoshopu z obrázků vzniklých snímáním obsahu obrazovky za běhu aplikace. Čísla v levém dolním rohu tlačítek nejsou součástí textury, jedná se o aktuální počet kol dané velikosti, která jsou k dispozici. Číslo udávající počet zubů kola naopak součástí textury je.

Další podpůrná funkce `printStringUsingRasterFont()` je, podobně jako funkce pro načítání textury, převzata z podkladů na cvičení z předmětu PGR. Funkce umožňuje vypisovat text do okna aplikace. Pomocí toho se uživateli v levém dolním rohu pracovní plochy zobrazuje, v jaké vrstvě se aktuálně nachází a kolik vrstev má vůbec k dispozici. V horní části okna si může uživatel přečíst, v jakém se nachází levelu a jaký je jeho aktuální úkol, případně informaci, že level (nebo i celou hru) zdárně dokončil. Čísla v levém dolním rohu tlačítek jsou rovněž realizována touto funkcí. Tímto způsobem je uživatel informován, kolik jakých kol má ještě k dispozici v paletě nástrojů.

Obrázek 28: Zobrazení čísla levelu a aktuálního úkolu



Obrázek 29: Informace, že uživatel dokončil level



Obrázek 30: Informace, že uživatel úspěšně dokončil hru

Hlavní funkcí je pro celé GUI funkce `DrawGUI()`. Ta je volána z funkce `onDisplay()`, tedy při každém překreslení okna. Funkce přijímá dva argumenty a to šířku okna a výšku menu. Podle toho je vykreslováno pozadí menu. Nejprve pomocí podpůrných funkcí vykreslí pozadí menu a následně ovládací tlačítka. Pak následuje generování textů pro výpis. U zobrazování počtu kol k dispozici v paletě nástrojů je použit jednoduchý kód.

```
itoa (inStack(25),msg,10);
printStringUsingRasterFont(msg,GLUT_BITMAP_8_BY_13,220,20,1.0,0.9,0.2);
```

Zde je konkrétně výpis stavu volných kol pro kola s poloměrem 25, tedy s osmi zuby. Funkce `inStack()` vrací hodnotu typu integer takovou, jaký je zbývající počet kol k dispozici v paletě nástrojů. Pomocí funkce `itoa()` je následně hodnota převedena na text. Ten je následně pomocí funkce printStringUsingRasterFont() vypsán na požadované místo. Parametry této funkce jsou postupně:

- `msg` – pole typu char obsahující zprávu k vypsání
- `GLUT_BITMAP_8_BY_13` – nastavuje bitmapový font se znaky ve velikosti 8 pixelů na šířku, 13 pixelů na výšku
- `220` – souřadnice na ose x

- `20` – souřadnice na ose y
- `1.0,0.9,0.2` – barva fontu ve formátu RGB zadaná datovým typem float, kde hodnoty smějí být v rozsahu 0.0 – 1.0

Jen o velice málo složitějším postupem je vygenerován řetězec pro výpis aktuální vrstvy. Je zde použito spojování řetězců. K získání textu pro zobrazení aktuálního levelu a současného úkolu je využito funkce `levelMessage()`. Ta vrací aktuální text. Výpis je pak analogický jako u předchozího případu.

Poslední funkcí, která se týká GUI, ačkoli se nejedná o vykreslovací funkci, je funkce `menuClicked()`. Ta je volána se dvěma parametry `x` a `y`, jenž jsou typu integer a předávají funkci souřadnice místa, kam bylo kliknuto. Funkce vyhodnotí, zda a na které tlačítko v menu bylo kliknuto a následně obslouží uživatelův požadavek.

6 Závěr

V závěru své práce bych chtěl shrnout, jaké poznatky jsem během tvorby diplomového projektu získal, které aspekty při návrhu a implementaci byly nejkomplikovanější a jakým způsobem jsem se s nimi vypořádal. Dále zhodnotím, zda jsem splnil všechny cíle stanovené v úvodu a nastíním, jaké budoucí využití by mohl projekt dále nabízet.

Vytvořit interaktivní hru se nakonec podařilo, analýza vzájemných vztahů v soukolí funguje a 3D vizualizace se vykresluje bez problémů. Přesto však tento úkol nebyl tak snadný, problémů se vyskytlo několik, především ve fázi návrhu a později při implementaci. Nakonec se ale podařilo všechny nástrahy zdárně překonat. Problematické situace nakonec daly vzniknout novým originálním řešením, se kterými jsem předtím nepočítal, ale které se nakonec ukázala jako vhodná.

Hlavním problémem bylo navrhnout vhodný způsob reprezentace vztahů mezi jednotlivými koly a algoritmy na zpracovávání těchto informací. Z plánu vytvořit úplný model grafu s orientovanými a ohodnocenými hranami pomocí datových struktur nakonec sešlo. Tento model byl příliš komplikovaný a údržba grafu v případě modifikace situace v návrhu soustavy kol byla téměř nadlidským úkolem. Naštěstí pak přišel nápad řešit vztahy mezi jednotlivými koly dynamicky po každé provedené změně bez nutnosti uchovávat informace o předchozím stavu. Tím bylo dosaženo jednoduchého způsobu výpočtu rychlosti a pootočení jednotlivých kol pomocí jednoduché rekurzivní funkce, která pro svůj běh požaduje pouze příznak zpracování u každého kola. Funkce zároveň zajišťuje kontrolu, zda se aktuálně existující soukolí vůbec otáčet může.

Jednoduché nebylo ani vyřešit problém se špatně počítanými normálovými vektory, které jsou klíčové pro správné zobrazení stínování na objektu vizualizace. Stínování je velice důležitým prvkem, protože bez něj nepůsobí model plasticky. Problém s nesprávným vykreslováním povrchu tedy nešlo obejít pouhým odstraněním stínování z aplikace, ale bylo nutné problém identifikovat a vyřešit. Přes dlouhé hledání chyby ve výpočtu normálových vektorů se jí stále nedařilo nalézt. Nakonec bylo zjištěno, že jedna funkce knihovny OpenGL se chová jinak, než bylo očekáváno. Pouhou změnou funkce použité pro nastavení normálových vektorů se vše opravilo a systém funguje na základě původních výpočtů, ve kterých byla původně chyba mylně očekávána. Ukázalo se, že výpočty byly od začátku řešeny správným způsobem.

Přes všechny komplikace se nakonec podařilo aplikaci vytvořit tak, aby splňovala veškeré cíle, které jsem si stanovil. Jednoduchá počítačová hra obsahující systém simulace soustavy ozubených kol se ovládá intuitivně a umožňuje uživateli získat lepší představu o funkci převodovky realizované ozubenými koly nebo jen relaxovat u plnění jednoduchých úkolů. Aplikace by se mohla stát inspirací pro budoucí plnohodnotnou počítačovou hru. Při tvorbě takovéto hry by mohly být použity zdrojové kódy této aplikace.

Při tvorbě aplikace jsem se naučil nové technologie, všechny nástrahy jsem nakonec zdárně překonal a všechny stanovené cíle dokázal splnit. Největší odměnou je ale pro mě vědomí, že je reálné na tuto aplikaci v budoucnu navázat, a to vytvořením plnohodnotné počítačové hry, která by díky mezeře na trhu mohla mít naději na úspěch. Zejména jsem si ale osvojil nové dovednosti a vědomosti, které s největší pravděpodobností využiji při plánovaném doktorském studiu.

Literatura

[1] *Active robots* [online]. [cit. 2010-01-06]. Dostupné z WWW: <http://www.active-robots.com/>.

[2] *CZ NeHe OpenGL - Obsah NeHe OpenGL tutoriálů* [online]. 2008 [cit. 2010-01-06]. Dostupné z WWW: <http://nehe.ceske-hry.cz/tut_obsah.php>.

[3] Fajkus Jan: *Znázornění práce cyklických motorů*. Brno, bakalářská práce, FIT VUT v Brně, 2007

[4] Herout, Pavel, *Učebnice jazyka C.*1. díl / 6. vyd. České Budějovice: Kopp, 2009. 271, viii s.; 20 cm. ISBN 978-80-7232-383-8 (brož.)

[5] Herout, Pavel, *Učebnice jazyka C.*2. díl / 4. vyd. České Budějovice: Kopp, 2008. 1 sv. (různé stránkování). ISBN 978-80-7232-367-8 (brož.)

[6] Munshi, Aaftab. *The OpenGL ES 2.0 programming guide.* Upper Saddle River: Addison-Wesley, c2009. xxxi, 417 s., ISBN 978-0-321-50279-7 (brož.)

[7] *Sandaig Primary School* [online]. [cit. 2010-01-06]. Dostupný z WWW: <http://www.sandaigprimary.co.uk/>.

[8] *Seriál Grafická knihovna OpenGL* [online]. 2010 [cit. 2010-01-06]. Dostupné z WWW: <http://www.root.cz/serialy/graficka-knihovna-opengl/>.

[9] Wikipedia, the free encyclopedia [online]. [cit. 2010-01-06]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Gear>.

Seznam obrázků