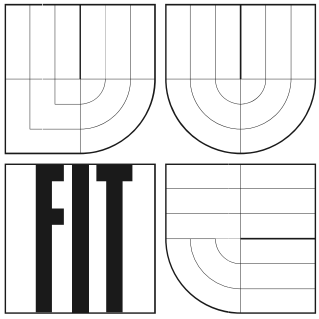
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



Simulace pohybu auta

Ročníkový projekt

2005 Jiří Pochop

Simulace pohybu auta

Odevzdáno na Fakultě informačních technologií Vysokého učení technického v Brně, dne 10. května 2005.

Prohlášení

Prohlašuji, že jsem tento ročníkový projekt vypracoval samostatně pod vedením pana Ing. Jana Pečivy.
Uvedl jsem všechny prameny, ze kterých jsem čerpal.

……
Jiří Pochop
10.května 2005

Abstrakt

Cílem této práce je vytvořit simulaci pohybu auta ve 3D scéně, ovládaného uživatelem klávesnicí nebo ovládané automaticky. Automatické řízení auta sleduje předem vytyčené body a aproximuje zatáčky. Důraz je kladen na přesné sledování vytyčené trasy a jednoduchost jejího zadání.

Na tomto základě potom vytvořit jednoduchý program a demonstrovat použití získaných algoritmů .

Klíčová slova

OpenGL, Coin3D, simulace, 3D scéna, generování krajiny, metoda zlomů, výšková mapa, transformační matice.

Poděkování

Rád bych poděkoval svému vedoucímu, kterým byl pan Ing. Jan Pečiva, že mi při práci na projektu pomohl a také za to, že napsal tak srozumitelný tutoriál k Coin3D.

Abstract

The goal of this project is to create a car movement simulation in a 3D scene, controled by user on keyboard or automatically. The automatic driver follows given points and aproximates curves. The weight in this project is set to exact following the route and to easy creating of the route.

Then create a simple program and demonstrate the usage of the algorithms.

Keywords

OpenGl, Coin3D, simulation, 3D scene, landscape generation, fault formation algorithm, height map, transformation matrix.

Obsah

1 Úvod

Tento ročníkový projekt a jeho praktická část (program *CarSim*) je zaměřený na navržení a vytvoření systému modelů aut pro herní použití, který bude vykazovat co nejvěrohodnější chování modelů aut při jízdě po zvlněném terénu v trojrozměrném prostředí. Není cílem dosáhnout dokonalé shody simulačního modelu s reálným chováním auta, ale umožnit jednoduše přidat do 3D scény několik pohyblivých modelů aut a ovlivnit jejich pohyb např. pro doplnění vzhledu simulace nebo hry. Vytvoření takového systému lze rozložit na několik částí:

Grafický systém: V projektu využívám systému OpenGL s nadstavbou Coin3D [2].

Model terénu: Vytvoření modelu krajiny, její výškové mapy a případné pokrytí terénu texturou.

Model auta: Prvotní nastavení parametrů pohybu a vzhledu modelu (např. měřítko).

Ovládání modelu auta: Program reaguje na vstupy z klávesnice přes knihovny Coin3D

Řídící body pro modely aut: Systém řídících bodů umožňuje ovládání modelů aut, které jsou řízeny automatem.

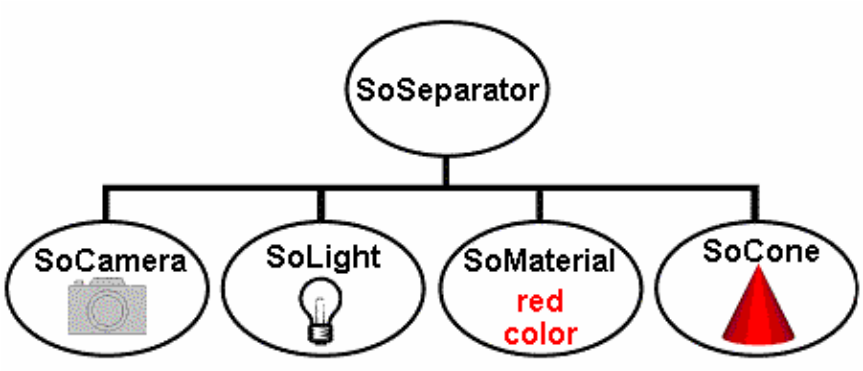
Vstupní parametry: Pro načtení vstupních parametrů je využijeme kombinace parametrů příkazové řádky a dat v externích textových souborech.

2 Coin3D

2.1 Obecně

Coin3D je nadstavbou grafického systému OpenGL. Jeho použitím se práce s 3D grafikou výrazně zjednodušuje. Využívá stromové struktury pro vyjádření závislostí jednotlivých komponent scény. Scéna má potom kořenový uzel a všechny komponenty scény (modely aut, model terénu, světelný zdroj, kamera atd.) jsou jeho synové (nodes). Do tohoto grafu scény se zařazují také „nehmotné“ prvky scény: prvek pro obsluhu klávesnice, časovač atp. Vhodným poskládáním jednotlivých jednoduchých komponent může tedy jednoduše vzniknout komplexní simulace na vysoké úrovni.

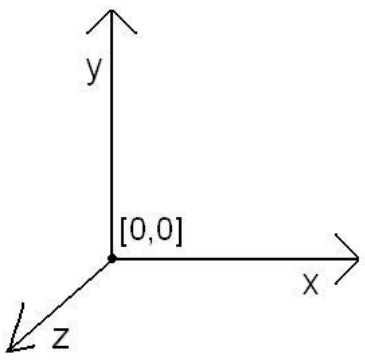
Popis jednotlivých uzlů scény použitých v projektu je uveden u popisu jednotlivých struktur.



Obrázek 1. Příklad grafu scény. Převzato z [1]

2.2 Coin3D scéna

Scénu tvoří „nekonečný“ prostor, do kterého jsou vkládány jednotlivé objekty scény. Jejich poloha je definována třemi souřadnicemi pravoúhlého souřadného systému.



Obrázek 2. Souřadný systém scény Coin3D.

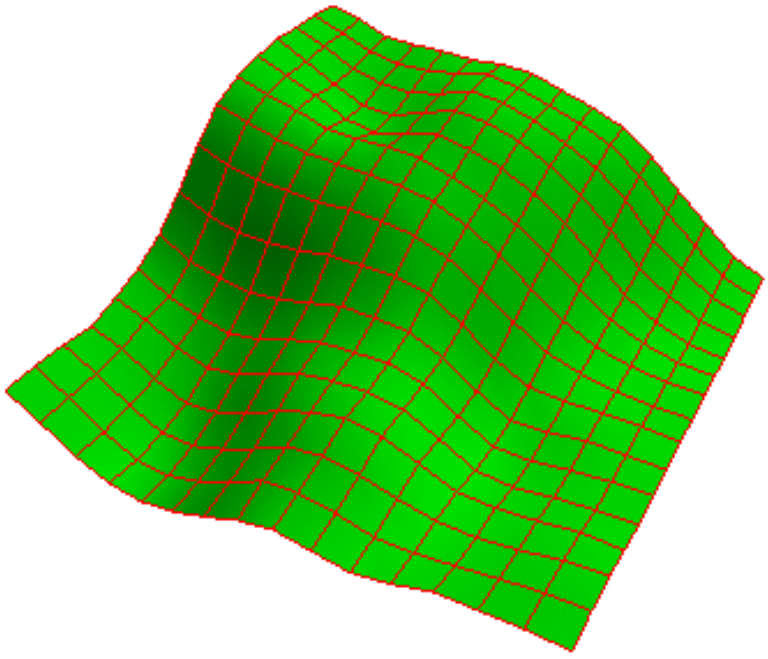
3 Modelování terénu

3.1 Algoritmus

Pro modelování terénu pro tento projekt jsem využil Fault formation algoritmus (metoda zlomů). Používá velmi jednoduchý princip - náhodně vygeneruje "zlom" v krajině a pak ke všem bodům jedné poloviny připočte nějakou hodnotu. ([1] – článek: Terrain Generator 2) Nakonec je celý povrch vyfiltrován filtrovací funkcí, aby bylo dosaženo hladšího a přirozenějšího vzhledu.

3.2 Datová struktura

Vygenerovaná datová struktura obsahuje model krajiny se všemi daty o trojúhelnících a normálách, ale pro potřeby dalšího využití tohoto modelu jako podkladu pro scénu s pohybujícími se auty především obsahuje tzv. **výškovou mapu.** Bylo by neekonomické z hlediska paměťové náročnosti uchovávat informace o výšce každého bodu ve scéně, proto je výšková mapa pouze dvourozměrné pole hodnot, které uchovává informace o výšce bodů ve vrcholech základní mřížky modelu krajiny.



Obrázek 3. Výšková mapa terénu. Převzato z [1]

Mezi těmito základními body výšku aproximujeme z okolních bodů.

4 Model auta

4.1 Model ve scéně

Jako každý model ve scéně Coin3D se i model auta skládá z několika uzlů. Modelu auta ve scéně je reprezentován třemi uzly:

Transformační uzel (SoMatrixTransformation): Uzel reprezentuje dvourozměrnou matici transformací 3D objektu. Pomocí této matice lze nastavit posun modelu ve scéně, jeho natočení, naklopení a měřítko (Více o transformační matici ve [3]).

Uzel rotace (SoRotation): Umožňuje rotovat model kolem specifikovaného vektoru o zadaný počet stupňů. Použito kvůli rozdílným souřadnicovým systémům výpočtů a prostoru scény.

Uzel se souborem modelu (SoFile): Obsahuje model auta načtený ze souboru.

4.2 Model v programu

V programu je model auta reprezentován řadou datových struktur popisujících jeho aktuální rychlost, maximální rychlost, limit rychlosti, aktuální souřadnice výšky, maximální zrychlení a zpomalení, maximální zatočení, směr, stav autopilota, jestli je na něj nasměrována kamera atd.

Všechny tyto atributy modelu určují, jak se model zachová a co se bude dít v následujícím časovém okamžiku (viz kapitola 5, kde najdeme také přesnější popis atributů modelu auta).

5 Princip simulace

Protože chceme, aby simulace pohybu auta probíhala v reálném čase, potřebujeme, aby všechny změny atributů modelu probíhaly v se změnou systémového času. Coin3D nám v tomto vychází vstříc. Můžeme definovat funkci, která se bude opakovaně spouštět pokaždé, když skončí renderování scény. V ní už jednoduše zjistíme časový okamžik, který uplynul od posledního spuštění a můžeme pomocí něj parametrizovat všechny děje týkající se pohybu a reakcí modelů. Také v ní obsloužíme vstupy z klávesnice a případné reakce na ně.

```
While not(konec) {
        Dt:=zjisti_uplynulý_čas();
        Ošetři_vstupy_z_klávesnice();
        Proveď_výpočty_na_modelech(dt);
        Uprav_objekty_scény_podle_nových_parametrů();
        Vykresli_scénu();
}
```

Zjednodušený algoritmus enginu simulace

Z výše uvedeného mimo jiné vyplývá, že přímo nezáleží na výpočetním výkonu stroje na kterém simulace běží (př. auta by měla být po 5 minutách simulace na stejných místech na pomalém i rychlém počítači). Následující dvě kapitoly popisují algoritmické výpočty změn parametrů, které probíhají v jednom časovém okamžiku a jsou implementovány v souboru **CarModel.h** v metodě **timeTick();**. Jejími parametry jsou časový interval, ukazatel na podklad (model krajiny) a ukazatel na kameru, jejíž pozice se mění s pozicí a směrem právě sledovaného modelu auta.

5.1 Výpočty u modelu bez autopilota

Model bez autopilota znamená, že uživatel model řídí z klávesnice a přímo ovlivňuje směr jízdy a rychlost. Zde využijeme podporu klávesnice, aby nám řekla, které klávesy jsou stisknuté.

Změna rychlosti: pokud je žádáno **zvýšení** rychlosti, vypočítá se nová rychlost jako:

$$v_{new} = v_{old} + \Delta t \cdot a ,$$

kde **v** je rychlost, **a** je maximální zrychlení auta definované pevně pro každé auto a Δt je čas od posledního volání funkce. **Snížení** rychlosti (brždění) se vypočítá obdobně:

$$v_{new} = v_{old} - \Delta t \cdot d \ .$$

Pokud dosáhne vozidlo své maximální dopředné (zpětné) povolené rychlosti, nemůže dále zrychlovat (zpomalovat). Nastane-li situace, že vozidlo se pohybuje nenulovou rychlostí, ale není stisknuta žádná z kláves ovlivňující rychlost, vozidlo pomalu snižuje svoji rychlost k nule podle vzorce

$$v_{new} = v_{old} \pm \Delta t \cdot a \ .$$

Změna se přičítá, blížíme-li se k nule zdola, odečítá, když shora.

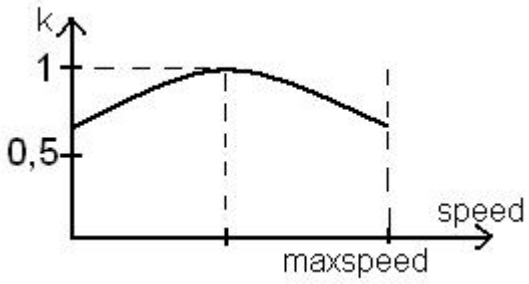
Změna směru: vozidlo má jako jeden ze svých atributů **směr**. Tento směr můžeme chápat jako azimut, odklonění od nulového směru. Otáčí-li se model vlevo, azimut roste, otáčí-li se vpravo, klesá. Hodnota tohoto atributu se pohybuje v rozsahu od 0 do 360. Při požadavku na změnu směru (stisknutí příslušné klávesy) se přírůstek směru dá vyjádřit jako:

$$h_{new} = h_{old} \pm \Delta t \cdot k \cdot tr \ ,$$

kde **h** je natočení (odklon od nulového natočení), **tr** je konstanta definující maximální zatočení auta a **k** je speciální hodnota, která zohledňuje schopnost auta zatočit v závislosti na aktuální rychlosti. Vypočet vypadá takto:

$$k = 1 - \left(\frac{v - v_{\max} / 2}{v_{\max}} \right)^2 ,$$

a průběh této funkce vidíme na obrázku 4.



*Obrázek 4. Závislost konstanty **k** na aktuální a maximální rychlosti.*

Pokud je rychlost auta menší nebo rovna nule, výpočet proběhne podle následujícího vzorce:

$$v_{new} = v_{old} \pm \Delta t \cdot tr \cdot \left(v / \left(v_{\max} / 2 \right) \right) .$$

5.2 Výpočty u modelu s autopilotem

Než se pustíme do vysvětlování algoritmu pro výpočet trasy automatu, je potřeba si vysvětlit, jak jsme přišli k datům, která v něm používáme. To, jaká data potřebujeme, určí použitý algoritmus výběru cesty.

5.2.1 Logika cesty

Největší problém u auta ovládaného autopilotem je rozhodnout, podle jakého algoritmu se bude pohybovat a jestli jej půjde nějak ovlivnit. Je několik možností, jak nechat automat řídit, mezi kterými jsem vybíral:

- Nechat auto jezdit zcela náhodně, případně podle některých náhodně generovaných křivek.
- Určit pevné body a nechat automat hledat mezi nimi optimální cesty.
- Určit pevné body a stanovit přesná pravidla pro průjezd mezi body.

Vycházel jsem z toho, že půjde o doplněk nějaké hry, nebo simulace, tedy bude potřeba přesně řídit, kudy auto projede (ulice města, mosty atp.). Rozhodl jsem se pro implementaci poslední ze jmenovaných možností - systému řízení modelů aut pomocí „waypointů“ (předem pevně určených bodů v prostoru), kterými model auta bude postupně projíždět po pevně stanovené trase podle daných pravidel. Tak se tedy určitě nestane, že by vozidlo jezdilo tam, kde nemá. Je ale zřejmé, že pro správné naplánování cesty touto metodou, je nutné znát předem terén a systém silnic. Při použití této metody odpadá rovněž nutnost implementace některé z metod pro hledání optimální cesty. Toto je ale vykoupeno podstatně vyššími nároky na systém pomocných bodů, vazeb mezi nimi a metainformací, které budou potřeba pro bezchybný průjezd vozidla systémem.

Rychlost simulovaného vozidla řízeného autopilotem je vždy stanovena na nejvyšší možnou a automat se snaží jet vždy jak nejrychleji to okolnosti dovolují. O limitování rychlosti se dočteme v kapitole 5.2.3.

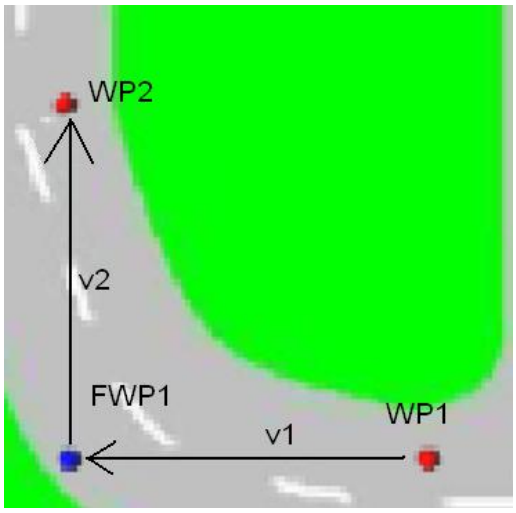
5.2.2 Systém pomocných bodů

Tento systém stanoví, jak se automatem řízené vozidlo chová při průjezdu systémem. Po pečlivém zvážení účelu simulace jsem dospěl k závěru, že pokud každý pomocný bod bude moci ukazovat na dva další, bude to plně postačující. Jednoduché křižovatky tak půjdou vytvořit bez problémů a složitější vhodnou kombinací jednoduchých.

Úspěch celého systému záleží hlavně na tom, jakým způsobem dokáže automatu sdělit informace potřebné k tomu, aby byl automat schopen aproximovat průjezd zatáčkou alespoň po

kružnici. Tento systém musí tedy hlavně „vědět“, od kterého bodu začíná zatáčka, kde končí a kterým směrem je auto natočeno při příjezdu k zatáčce a jak je natočeno potom.

Tento problém se mi podařilo vyřešit přidáním jisté redundance do systému pomocných bodů. Systém obsahuje pravé a tzv. falešné „waypointy“. Mezi pravými waypointy očekáváme rovný úsek dráhy. Pokud je ale mezi nimi falešný waypoint, znamená to, že ohraničují zatáčku. Tyto falešné navigační body v sobě nesou veškeré důležité informace o zatáčce, ve které leží. Protože aproximujeme zatáčky po kružnicích, tak mezi důležité informace o zatáčce patří **změna směru**, ke které dojde po projetí zatáčkou, **poloměr** zatáčky a **dráha** pro projetí zatáčkou. Postup výpočtů:



Obrázek 5. Výpočet informací o zatáčce.

Podle obr. 5 vidíme, že známe-li pozice bodů WP1, WP2 a FWP1, zjistíme vektory v1 a v2 jako

$$\vec{v1} = \overrightarrow{FWP1} - \overrightarrow{WP1} \quad \text{a} \quad \vec{v2} = \overrightarrow{WP2} - \overrightarrow{FWP1}.$$

Zjistíme jejich orientaci v našem prostoru (0 směřuje dolů a doleva se přičítá) a zjistíme změnu směru:

$$zmena\_smeru = rot\left(\vec{v2}\right) - rot\left(\vec{v1}\right) = 180 - 270 = -90 .$$

Poloměr můžeme spočítat takto:

$$r = \frac{\left|\vec{v1}\right|}{\tan\left(zmena\_smeru / 2\right)}.$$

Poslední údaj o zatáčce, projetou dráhu, spočteme potom takto:

$$s = \frac{zmena\_smeru \cdot 2 \cdot \pi \cdot r}{360}$$

Tyto výpočty provádíme ještě před spuštěním simulace, při načítání pomocných bodů.

5.2.3 Limitování rychlosti

Aby bylo možné správně projet zatáčkou, je zapotřebí určit pro každou zatáčku maximální rychlost, kterou dané vozidlo může zatáčkou projet. Protože dopředu nevíme, jaké vlastnosti (hlavně maximální zatočení auta) bude projíždějící auto mít, musíme tento limit dopočítávat pokaždé, když má auto zatáčkou projet. A protože na začátku zatáčky už musí auto jet maximální přípustnou rychlostí, případné zpomalení provádíme už na předcházejícím úseku. Limit rychlosti tedy počítáme o úsek dopředu. Pokud následující bod označuje křižovatku, vezme se menší z obou limitů.

Limit rychlosti spočteme a automat začne zpomalovat hned, když zjistíme, že za následujícím waypointem je zatáčka (falešný waypoint). Bohužel to vše má za následek, že auto zpomalí do menší ze zatáček, ale pak do ní nemusí vjet.

$$v_{\lim it} = tr \cdot k \cdot \frac{delka\_zatacky}{zmena\_smeru}$$

V rovnici je **tr** maximální zatočení auta, **k** spočteme stejně jako v kapitole 5.1 a zbývající dvě konstanty jsme získali z příslušného falešného pomocného bodu.

5.2.4 Výpočty

Výpočet **změny směru** se rozpadá na dvě možnosti. Pokud se automat s autem **nachází v zatáčce** (poznal to podle falešného waypointu), má vše pro to, aby projel zatáčkou po kruhové trajektorii. Změna směru se tedy vypočítá takto:

$$h_{new} = h_{old} + \frac{(\Delta t \cdot v \cdot zmena\_smeru)}{delka\_zatacky}.$$

V této rovnici je **v** aktuální rychlost, Δt časový úsek od posledního výpočtu a ostatní konstanty jsou získány z informací o zatáčce.

Pokud se v **zatáčce nenachází**, zjistí si automat odchylku od požadovaného směru (k pozici následujícího waypointu) a pokud je tato odchylka větší než požadovaná přesnost, začne provádět opravu v požadovaném směru. Nový směr se tedy v tomto kroku vypočítá se stejnou změnou směru, jako v poslední zatáčce. V předchozím kroku se ale také mohla provádět tato oprava a pokud se prováděla do opačného směru než bychom chtěli opravovat teď, je naše oprava pouze desetinová. Tím se zamezí nepříjemným oscilacím.

Výpočet změny rychlosti automatu podléhá původní myšlence: auto se snaží jet tak rychle, jak to jen v dané situaci jde. Jestliže tedy je aktuální rychlost vozidla větší než povolená, rychlost snižujeme, v opačném případě rychlost zvyšujeme. Vzorce pro výpočet:

$v_{new} = v_{old} + \Delta t \cdot a$ pro zrychlení a $v_{new} = v_{old} - \Delta t \cdot d$ pro zpomalení

5.3 Úpravy parametrů objektů scény

V okamžiku, kdy jsou spočteny změny směru a rychlosti, je potřeba změnit atributy objektů ve scéně, aby mohly být systémem Coin3D správně zobrazeny.

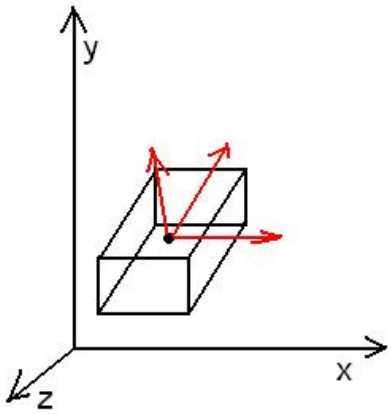
5.3.1 Aktuální pozice

Nejprve je potřeba zjistit, kam se model auta za uplynulý čas posunul. Toto zjistíme z aktuální rychlosti, normalizovaného vektoru aktuálního směru, změna času od posledního výpočtu a původní pozice takto:

$$\overrightarrow{pos_{new}} = \overrightarrow{pos_{old}} + v \cdot \Delta t \cdot \overrightarrow{smer_{norm}}$$

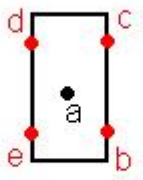
5.3.2 Naklopení a výška ve scéně

Protože se model auta pohybuje po terénu, který je zvlněný, je nutné, aby se podle terénu po kterém jede naklápěl. Toho docílíme použitím již zmíněného uzlu pro maticovou transformaci modelu. Nastavíme jej pomocí tří směrových vektorů. Tyto vektory jsou navzájem kolmé a svým natočením vůči souřadnému systému scény určují natočení modelu (viz obr.6).



Obrázek 6.

Vektory získáme několika výpočty nad výškovými souřadnicemi bodů a,b,c,d,e z obrázku 7, kde je vyobrazena poloha auta a tyto speciální body. Bod **a** je těžiště modelu, ostatní reprezentují kontakt modelu auta s povrchem (kola). Pozice směrových vektoru **x** ve výškové mapě je daná aktuálním vektorem směru. Vektor **z** je vůči **x** otočen o 90 stupňů. Výšková souřadnice vektoru **z** je průměrem výšek bodů **d** a **c**, výšková souřadnice vektoru **x** je průměrem bodů **c** a **b**. Směrový vektor **y** získáme vektorovým součinem vektorů **x** a **z**. Atribut výšky vlastního modelu je průměrem všech červených bodů.



Obrázek 7.

Jednotlivé složky právě získaných vektorů se zapíší do matice transformací následovně:
Vektor **x** vyplní první sloupec transformační matice (viz [3]) tak, že x-ová souřadnice v prvním, y-ová ve druhém a z-ová souřadnice ve třetím řádku. Vektor **y** vyplní podobně druhý sloupec a vektor **z** třetí sloupec.

5.3.3 Nový waypoint

Po provedení všech oprav pozice a natoční je nutné zjistit, zda už není auto dostatečně blízko ke svému současnému cíli a jestli už není čas přepnout se na další cíl cesty. V průběhu všech výpočtů dochází k zaokrouhlování a jiným nepřesnostem, je tedy nastavena jistá tolerance . Pokud se tedy model nachází v dosahu waypointu, dojde k aktivaci dalšího v řadě. V případě, že se jednalo o křižovatku, je další cesta vybrána náhodně.

Tímto novým cílem může být i falešný waypoint. Z něj jsou potom načtena data o zatáčce a jako aktuální waypoint nastaven jeho následník.

Zároveň je spočten i limit rychlosti.

5.3.4 Úprava pozice kamery

Novou pozici kamery, která sleduje vybrané auto, zjistíme z aktuální pozice a vektorem posunutí kamery (tento je přepočten podle natočení auta a požadované polohy kamery za autem)

6 Ukázková aplikace

Program SimCar načte počáteční informace o systému waypointů a modelech aut z textových souborů. Informace o waypointech jsou předány do příslušných datových struktur a následně zřetězeny. Data modelů aut se zinicializují, modely zasadí do scény na startovací waypoint a spustí se simulace. Program přebírá od uživatele vstupy z klávesnice a způsobem popsaným v kapitole 5. je zpracuje.

6.1 Ovládání

Šipky – manuální ovládání auta
A - zapíná a vypíná autopilota u auta, které je právě zaměřeno kamerou
Tab - přepíná pohled kamery na další auto ve scéně
Enter – přepíná kameru na sledování celé scény shora a zpět
Esc – konec aplikace
N – může přepnout na další waypoint (pouze u auta s manuálním ovládáním)
Space – přepíná mezi způsoby zobrazení waypointů ve scéně

7 Závěr

Ve vytvořené aplikaci byly implementovány algoritmy uvedené v teoretické části projektu. Výsledná aplikace umožňuje přidat do scény teoreticky neomezené množství modelů aut a jejich waypointů. Také je možné jeden z modelů ovládat přímo a přepínat se mezi nimi. Simulace sice není a ani nemá být fyzikálně přesná, ale splňuje požadavky, které jsme na ni v úvodu kladli.

7.1 Návaznost na jiné projekty

Tento projekt byl tvořen s vizí, že se výsledky zkombinují s některými ostatními projekty vyvíjenými souběžně. Bohužel se ukázalo, že většina těchto projektů zaměřených na práci s náhodně vygenerovaným 3D terénem se ubírá jiným směrem a nešlo by tyto projekty rozumně sloučit do logického celku. Kompatibilita zde ovšem vzhledem k použitým datovým strukturám je.

7.2 Možné pokračování

V průběhu testování a ladění programu jsem však narazil na několik věcí, které jsou nad rámec zadání a nemají na funkčnost simulačního algoritmu vliv, ale možná by usnadnily práci uživateli a zlepšily výsledný efekt aplikace. Zde jsou některé z nich:

Zadávání pomocných bodů pomocí textového souboru je pro uživatele je poměrně pracné. Zadat je bez drobných chyb, které se později těžko odstraňují, se ukázalo jako téměř nemožné při větší složitosti systému. Vytvoření editoru waypointů na bázi grafického editoru by práci uživateli jistě usnadnilo.

Práce s kamerou je v aplikaci na základní úrovni, vlastně téměř chybí. Jako dalším rozšířením aplikace by tedy mohlo být doplnění ovládání kamery myší např. ve stylu některých her (Homeworld atp.).

Optimalizace výpočtů se při vyšších počtech simulovaných modelů ukázala být dalším logickým krokem ve vývoji použitých algoritmů. Může se totiž stát, že model během své neaktivní fáze přejede svůj waypoint. Ovšem pro zprůhlednění a jednoduché pochopení algoritmu jsem tento krok zatím vynechal. Pokud ovšem budeme chtít kód využít na pozadí jiné aplikace, nebo s větším počtem modelů, budou některé úpravy nutné.

Přesnější model ovládání auta přes klávesnici. V aplikaci je použit jednoduchý model ovládání auta klávesnicí. K demonstračním účelům ovšem plně postačuje.

Načítání terénu ze souboru by odstranilo některé problémy s editací systému waypoiontů spojené s neznalostí povrchu terénu (generuje se náhodně). Mohla by se např. do 3D scény vrátit vodní plocha nebo některá jiná vylepšení.

Literatura

[1] Ing. Pečiva, Jan: *Seriál Open Inventor*.
URL projektu: http://www.root.cz/clanky/open-inventor/ (duben 2005)

[2] Coin3D, knihovna k dispozici ke stažení zde: http://ftp.coin3d.org/coin/src/Coin-2.4.0.tar.gz (duben 2005)

[3] Doc.Dr.Ing. Černocký, Jan, 5. přednáška z předmětu Základy počítačové grafiky: Transformace ve 2D/3D, k dispozici pro studenty FIT-VUT. URL: https://www.fit.vutbr.cz/study/courses/IZG/private (duben 2005)