VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

3D ZÁVODNÍ HRA

3D RACING GAME

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **JAN VLACH**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. MICHAL TÓTH**
SUPERVISOR

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Vlach Jan**
Obor: Informační technologie
Téma: **3D závodní hra**
3D Racing Game
Kategorie: Počítačová grafika

Pokyny:

1. Prostudujte knihovnu OpenGL a její nadstavby.
2. Navrhněte 3D závodní hru s množstvím grafických efektů
3. Implementujte navrženou aplikaci
4. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu
5. Vytvořte video pro prezentování projektu.

Literatura:

- Podle pokynů vedoucího

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Tóth Michal, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato práce se zaměřuje na návrh a implementaci 3D závodní hry s grafickými efekty. Cílem této práce je vytvořit závodní hru s grafickými efekty tvořeny především emitory částic. K detekci kolize je ve hře využita fyzikální knihovna *Bullet*. Díky ní všichni hráči reagují na objekty scény. Terén je generován z výškové mapy a podporuje techniku *multitexturing*, která dovoluje nanést více textur na povrch terénu. Herní trať je vymezena pomocí kolizní mapy. V práci je kladen důraz na vytvoření editoru pro snadnou tvorbu a úpravu map. Editor dále umožňuje kreslení křivek *Kochanek-Bartels*, které jsou použity k určení pohybu oponentů.

Abstract

This bachelor thesis focuses on desing and implementation of 3D racing game with graphical effects. The goal of this thesis is to create a racing game with the help of particle emitors. For collision detection is used *Bullet* physics library. Because of it, all players react to other scene objects. Terrain is generated from height map and supports *multitexturing* technique, which allows for multiple textures to be applied to terrain's surface. Game's racing track is defined by collision map. In thesis, there is an emphasis on creating editor for easy map creation and editation. Editor allows for drawing *Kochanek-Bartels* splines, which are then used to determine the movement of the oponents.

Klíčová slova

3D hra, křivka Kochanek-Bartels, interpolace, výšková mapa, multitexturing, OpenGL, částicové efekty, editor map.

Keywords

3D game, Kochanek-bartels spline, interpolation, height map, multitexturing, OpenGL, particle effects, map editor.

Citace

VLACH, Jan. *3D závodní hra*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Tóth Michal.

3D závodní hra

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Tótha. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Vlach
17. května 2017

Poděkování

Rád bych poděkoval svému skvělému vedoucímu, panu Ing. Michalovi Tóthovi za jeho čas, pomoc a skvělé nápady. Také bych rád poděkoval své rodině a blízkým přátelům za podporu, hlavně Bc. Ondřeji Hájkovi za pomoc při odhalování chyb při jízdě vozidel.

Obsah

Kapitola 1

Úvod

První hry se začínaly objevovat kolem roku 1970 a využívaly velmi jednoduché grafiky. S postupem času se grafická stránka her začala vyvíjet. Z éry 2D her se s posupným nárůstem výkonu začaly objevovat rané 3D hry, jako *Star Fox* nebo *Quake*. Jak výkon počítačů neustále roste, vzrůstají i vizuální požadavky. Nejnovější hry se snaží přiklonit k filmovějšímu zpracování.

Cílem práce je vytvořit závodní hru s grafickými efekty, kvůli kterým jsou implementovány subsystémy jako je částicový systém, systém animací, či umožňující pohyb počítačem řízených protivníků. Implementovaná aplikace používá fyzikální knihovnu **Bullet** pro detekci kolizí mezi herními entitami. Aby bylo možné grafickým modelům přiřadit jejich kolizní obal, je součástí výsledné aplikace i editor úrovní pro snadné vytváření nového herního obsahu. Aplikace využívá grafické API **OpenGL** ve verzi 3.2. V této verzi už užívá dynamické pipeline, díky které lze přesunout část výpočtů na grafickou kartu a ušetřit procesorový výkon. Jednotlivé grafické modely, co jsou použity ve hře, byly vytvořeny v modelovacím software **Blender**.

Druhá kapitola seznamuje s množstvím základních pojmů z oblasti počítačové grafiky jako jsou *Homogenní souřadnice*, *souřadné systémy* a *animace*. Dále rozebírá téma interpolačních křivek, jejich výhod a nevýhod. V poslední řadě jemně seznamuje s *OpenGL*.

Ve *třetí* kapitole je diskutován návrh herních mechanizmů a požadavků na implementaci, které z návrhu plynou. Kapitola seznamuje s konceptem editoru a možnostmi, jako je kreslení křivek užitých pro určení trasy oponentů.

Předposlední kapitola popisuje kompletní implementaci hry a všech a podporovaných herních entit. Kapitola se detailněji zaměřuje na implementaci terénu, částicových efektů a chování hráčů. Kapitolu završuje implementace editoru a algoritmy pro něj, jako ovládání modelů myší nebo kreslení křivek.

Poslední kapitolou je závěr, obsahující zhodnocení práce a možnosti rozšíření hry.

Kapitola 2

Analýza a specifikace

Kapitola seznamuje se základními afinními transformacemi a homogenními souřadnicemi, s výhodami rotace vyjádřené pomocí kvaternionů. Dále stručně popisuje rozdíly mezi souřadnými systémy, redefinuje pojem model v kontextu práce a seznamuje s několika interpolačními metodami.

2.1 Podobné hry

Her se závodní tématikou je opravdu hodně. Mezi nimi autor vybral jednu, se kterou je velice dobře obeznámen, a to **Crash team racing**. Ve hře jsou dostupné herní módy typu *Race*, což jsou standardní závod proti počítačem řízeným oponentům, *time trial*, ve kterém se závodník snaží sám projet trať co nejrychleji, *battle*, který je dostupný jen pro více hráčů. Ve hře závodníci na trati využívají různé zbraně, které jsou jim náhodně přiděleny pro projetí krabic, roztroušených po mapě. Herní prvky této hry byly inspirací pro vytvoření této práce.

Obrázek 2.1: Crash team racing, převzato z webu[1].

[1]Dostupné z http://gamesretrospect.com/2014/12/crash-team-racing-review/

2.2 Hra

Hra se obecně skládá z obrázků třídimenzionálního či dvoudimenzionálního světa, ve kterém hráč kontroluje postavu, zvíře, nebo vozidlo.[1, s.8] Hra je real-time interaktivní simulace s agenty, simulující definovaný model. Tento model je pouze aproximací reality, simulované subsystémy jsou pouze zjednodušeným popisem jejich předloh. Agenti v tomto případě jsou jiná vozidla, která mezi sebou a hráčem interagují.

2.3 Homogenní souřadnice a afinní transformace

Mezi základní geometrické transformace se řadí rotace, změna měřítka, zkosení nebo posunutí. První tři jsou reprezentovány 3x3 maticemi. Rozšířením těchto matic do **Homogenních souřadnic** znamená, že je možné uniformně pracovat s nimi a s posuvem.

Bod v homogenních souřadnicích

Bod v kartézských souřadnicích v 3D prostoru je uspořádaná trojice $[x, y, z]$. Stejný bod bude v homogenních souřadnicích reprezentován uspořádanou čtveřicí čísel $[x, y, z, w]$, kde w je váha bodu a je rovna 1. Při realizaci vektoru je parametr w roven 0.

Afinní matice je 4x4 transformační matice, co zachovává paralelismus přímek a jejich relativní vzdálenosti od sebe, ale ne nezbytně absolutní délky a velikosti úhlů. **Afinní matice** je libovolná kombinace matic základních geometrických transformací.[1, s.158]

Rotace kolem obecné osy

Rotační matice existují celkem tři, pro každou dimenzi jedna. Rotaci v libovolné ose je pak nutné vyjádřit jako skládanou operaci rotací v jednotlivých osách. Je třeba říci, že na pořadí rotace záleží, a proto je vhodné určit konvenci rotace a držet se jí. Při skládání rotací může nastat případ, kdy rotace kolem jedné osy způsobí, že dvě osy budou ve stejné rovině. Potom nastal tzv. **Gimbal Lock** a rotace ztrácí jeden stupeň volnosti.

Rotace kolem obecné pomocí kvaternionů

Kvaterniony jsou rozšířením komplexních čísel. Oproti skládaným rotacím u kvaternionů **Gimbal Lock** nikdy nenastane. Oproti maticovému vyjádření lze kvaterniony snadno mezi sebou interpolovat.

2.4 Souřadné systémy

Díky *afinním* transformacím se body a vektory vždy vyjadřují relativně k souřadnému systému (prostoru).

Model space (Modelový prostor)

Je prostor, ve kterém je vyjádřen *model*. Aplikací **Modelové** matice M_M je vertex v transformován do prostoru světa $v_{M \to W} = M_M * v$.

World space (Prostor světa)

Aplikací **Modelové** matice je vertex nyní vyjádřen relativně k počátku scény. **View** matice M_V transformuje vertex v_M do prostoru kamery $v_{M \to V} = M_V * v_M$.

View space (Prostor kamery)

Reprezentuje kameru v **OpenGL**. Souřadnice vektoru jsou transformovány z pohledu kamery. Další transformační maticí, kterou je nyní **Projection** matice M_P, je vertex převeden do **Clip Space** $v_{V \to P} = M_P * v_V$.

Clip space

Při vytvoření M_P je specifikován interval koordinát pro každou dimenzi (**Viewing frustum**). Ten má tvar dle typu projekce (ortogonální či perspektivní). Pokud pozice vertexu v_P nespadá do intervalu, není vertex vykreslen.

Normalized device space

Souřadnice výstupního zobrazovacího zařízení jsou definovány na intervalu <-1,1> ve dvou dimenzích.

2.5 3D herní model

V úvodní sekci 2.2 je model definován v kontextu simulace. V kontextu práce bude modelem dále označena síť vertexů, uv souřadnic, a normál, tvořící logický celek. Specifikováno je i pořadí vertexů a typ polygonů. Polygony jsou téměř vždy trojúhelníky a popisují vizuální strukturu modelu v *modelovém prostoru*. Herní model také může obsahovat informaci buďto o barvě (pak je v grafických formátech navíc uvedena *RGBA* hodnota pro každý vertex), nebo je model opatřen texturou a namísto *RGBA* hodnot obsahuje *uv souřadnice*, které popisují mapování textury na něj. Výše uvedené je spíše minimum, modely vetšinou obsahují i jiné informace, například normály, tangenty, kosti a váhy vertexů, využívané při skeletální animaci a jiné.

Nová instance modelu

Standardně pro vykreslení modelu je třeba data modelu poslat na grafickou kartu spolu s modelovou maticí, co reprezentuje transformaci z *Model space* do *World space*, jak bylo zmíněno v sekci 2.4. Protože jsou transformace nezávislé na modelu, pro přidání dalšího **stejného** modelu není třeba na grafickou kartu opět odeslat data modelu, ale pouze novou *Modelovou* matici.

2.6 Animace

Animace je proces změny viditelných parametrů. Ve hře to může být například změna tvaru entity, rotace, měřítka nebo pozice herní entity v čase. Podle času lze animace dělit:

- **Vykreslení animace v reálném čase (Real-time rendering):** Animace je zobrazena v reálném čase, a parametry musí být plynule zobrazeny několikrát za vteřinu.

Animace tak klade hlavní důraz na rychlost a standardně animované jsou méně detailní. Detaily mohou být pak simulovány například modifikací normálových vektorů, nebo dopočítány pomocí **tesalace**. Tento přístup se používá ve hrách.

- **Vykreslení animace snímek za snímkem (Frame by Frame rendering):** Téměř výhradně používající se ve filmovém průmyslu, u Frame by Frame odpadá požadavek na rychlost, každý dílčí snímek animace je zaznamenán v paměti, a následně jsou snímky spojeny v celistvou animaci.

Standardně animátor specifikuje jen parametry animace pro klíčové pózy(keyframes) a ostatní snímky, tzv. mezisnímky jsou generovány automaticky pomocí interpolace.

Pro vyjádření animace pohybu se často používají animační křivky. Tvorba klíčových snímků pak probíhá z vytvořené interpolační křivky podle definované rychlosti pohybu. Následně je určena i orientace objektu, pohybujícího se po křivce a to z lokálního souřadnicového systému objektu.

2.7 Částicové systémy

Částicové systémy se velmi často užívají k produkci vizuálních efektů. Standardně se jich ve scéně vyskytuje velké množství, nebo popisují deformaci složitých modelů. Částice lze použít jako jednoduchou náhradu komplexní simulace, jako např. efektů kouře, nebo k zobrazení velkých skupin malých živočichů.

Systém částic je reprezentován vlastnostmi několika vlastnostmi, jako jsou změna barvy, polohy, rychlosti a směru pohybu. Částice dynamicky vznikají a zanikají. V částicích je důležitým prvkem náhodnost. Ta se charakterizuje jednoduchým vztahem

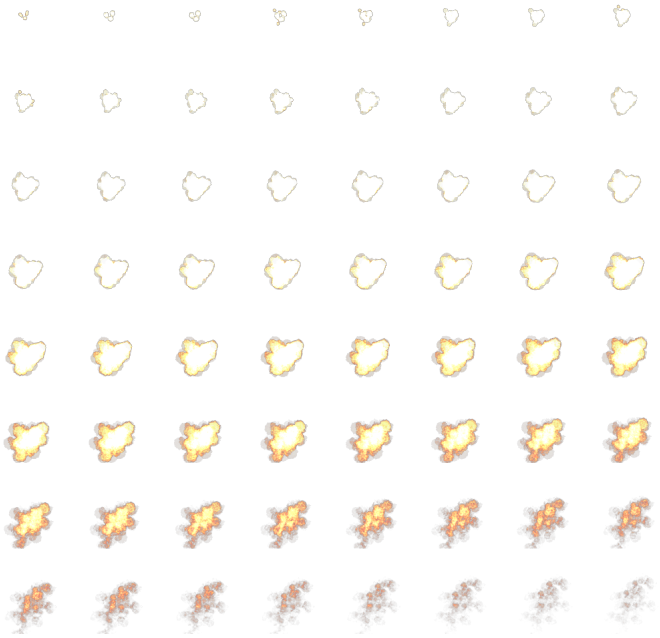
$$x = s + rand * v \tag{2.1}$$

- x je modifikovaný parametr (délka života částice, její barva, atd...)
- s je střední hodnota pravděpodobnosti
- v je rozptyl
- $rand$ je vygenerované náhodné číslo

Při generování částice jsou nastaveny se její parametry. Následně proběhne aktualizace parametrů všech částic emitoru. Částice, které překročili dobu života, jsou z emitoru odebrány. Nakonec jsou všechny částice emitoru vykresleny.

Texture atlas

Přepínání textury je pro grafickou kartu nákladná operace. Aby se tak dělo co nejméně, je vhodné více malých obrázků vložit do jedné textury. Této textuře se říká *Texture atlas* a používá se při nanášení textury na objekty *GUI* a pro animaci částic.



Obrázek 2.2: Příklad **texture atlasu**, převzato z webu.[2]

2.8 Interpolace a metody

Je numerická metoda aproximace reálné funkce, kdy je v intervalu známo několik uzlových bodů. Z těchto kontrolních bodů lze vypočítat předpokládanou hodnotu funkce. Matematicky řečeno, je hledána funkce $g(x)$ tak, že musí platit $g(x_i) = f(x_i),\ i = 0, 1 \ldots n$.

Na místě jsou dvě skupiny metod a to:

- **Interpolace polynomem**
- **Interpolace křivkou**

Interpolace polynomem

Metoda prokládá množinou bodů interpolační polynom. Tím je hledána jedna funkce, procházející všemi uzly interpolace. Funkce má tvar polynomu, kde stupeň polynomu závisí na počtu uzlů interpolace. S každým stupněm polynomu se stupňuje složitost výpočtu. U interpolačního polynomu nastává velký problém: při vyšším počtu uzlů interpolace má výsledná funkce(křivka) tendenci oscilovat. Proto všechny metody spadající pod tuto kategorii jsou *nevhodné*.

Interpolace křivkou

Metoda se nesnaží proložit všechny uzlové body naráz. Hledá se jeden funkční předpis platný pro jeden interval, specifikovaný uzlovými body. Pak hledaná funkce bude na intervalech definována různě.

[2] http://www.opengl-tutorial.org/intermediate-tutorials/billboards-particles/particles-instancing

Příkladem může být:

$$f(t) = \begin{cases} 1 & t < -1 \\ g(t) & t > -1 \end{cases}$$

Interpolace křivkou netrpí neduhem oscilace jako polynomiální interpolace. V praxi se používají křivky 3.řádu (neboli kubické). Následující metody patří do této kategorie.

Přímka

Přímka je nejjednodušší interpolační křivkou. Má spojitost pouze prvního řádu. Jednotlivé části mezi uzly interpolace jsou nahrazeny přímkou, přechody mezi intervaly výsledné funkce jsou velmi ostré.

$$S_i(x) = f(x_i) + \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}(x - x_i), \quad x\epsilon < x_i, x_{i+1} > \tag{2.2}$$

Metoda není vhodná k použití.

Kubická křivka

Každý interval $< x_i, x_{i+1} >$ je nahrazen kubickým polynomem ve tvaru:

$$P(x) = c_3x^3 + c_2^2 + c_1x + c_0 \tag{2.3}$$

Výsledkem je je křivka, který velmi dobře interpoluje zadané uzly, přechody mezi intervaly $< x_i, x_{i+1} >$ jsou velmi jemné. Metoda Kubické křivky pracuje pouze s hodnotami. Následující metody umožňují specifikovat interpolaci například koeficientem tuhosti. Ty ovšem nahrazují v intervalu nejen hodnoty, ale i jejich derivace.

Hermitova kubika

Oproti obyčejné kubické křivce jsou derivace g_k, g_{k+1} koncových bodů y_k, y_{k+1} specifikovány explicitně. Proto je třeba najít koeficienty c_3, c_2, c_1, c_0 z rovnice kubické křivky [2.3] tak, aby $P(x_k) = y_k$, $P(x_{k+1}) = y_{k+1}$ a aby derivace splňovali podmínky $P(x_k) = g_k$ a $P(x_{k+1}) = g_{k+1}$.

Při dosazení podmínek do rovnice:

$$\begin{gathered} c_3x_k^3 + c_2x_k^2 + c_1x_k + c_0 = y_k \\ 3c_3x_k^2 + 2c_2x_k + c_1 = g_k \\ c_3x_{k+1}^3 + c_2x_{k+1}^2 + c_1x_{k+1} + c_0 = y_{k+1} \\ 3c_3x_{k+1}^2 + 2c_2x_{k+1} + c_1 = g_{k+1} \end{gathered} \tag{2.4}$$

Definujme proměnnou u na intervalu $[x_k, x_{k+1}]$, která umožňuje vyjádřit posuv bodu na intervalu. Používá se v lineární interpolační funkci:

$$f_k(u) = (1 - u)y_k + uy_{k+1} \tag{2.5}$$

Hermitova kubika používá speciálně navržené křivky pro každý uzel.

$$\begin{gathered} h_1(u) = 2u^3 - 3u^2 + 1 \\ h_2(u) = -2u^3 + 3u^2 \\ h_3(u) = u^3 - 2u^2 + u \\ h_4(u) = u^3 - u \end{gathered} \tag{2.6}$$

Pokud zkombinujeme rovnice [2.4] a [2.6], dostáváme:

$$f_k(u) = y_k(2u^3 - 3u^2+) + y_{k+1}(-2u^3 + 3u^2) + \triangle_k g_k(u^3 - 2u^2 + u) + \triangle_k g_{k+1}(u^3 - u) \quad (2.7)$$

kde $\triangle k = x_{k+1} - x_k$ specifikuje interval. Nyní už stačí specifikovat chování derivací g_k a g_{k+1}

Kochanek-Bartels křivky (TBC křivky)

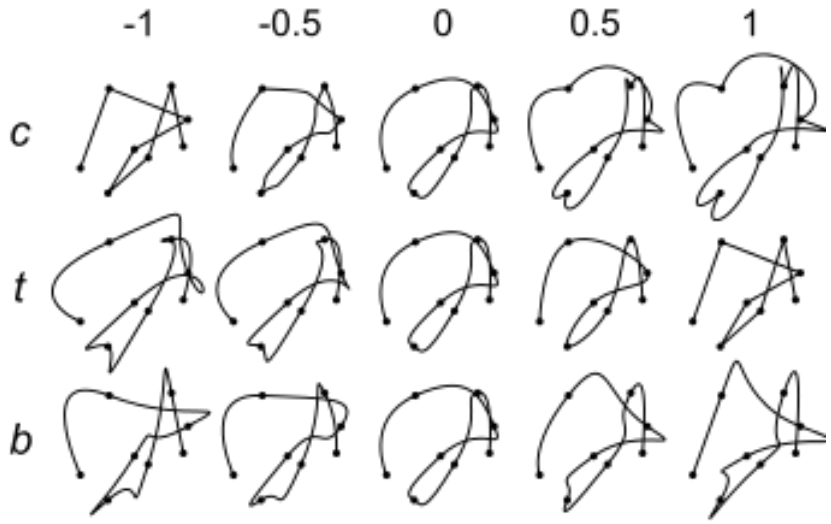
Vychází z Hermitovské kubiky. Křivka **Kochanek-Bartels** se standardně využívá při animaci, kvůli třem parametrům, s jejichž pomocí lze snadno měnit průběh interpolace. Tyto parametry jsou:

- **Napětí (tension)** a
- **Šikmost (bias)** b
- **Spojitost (continuity)** c

Pro každý uzlový bod křivky je definován vstupní(l_i) a výstupní(r_i) tečný vektor vztahem 2.8.

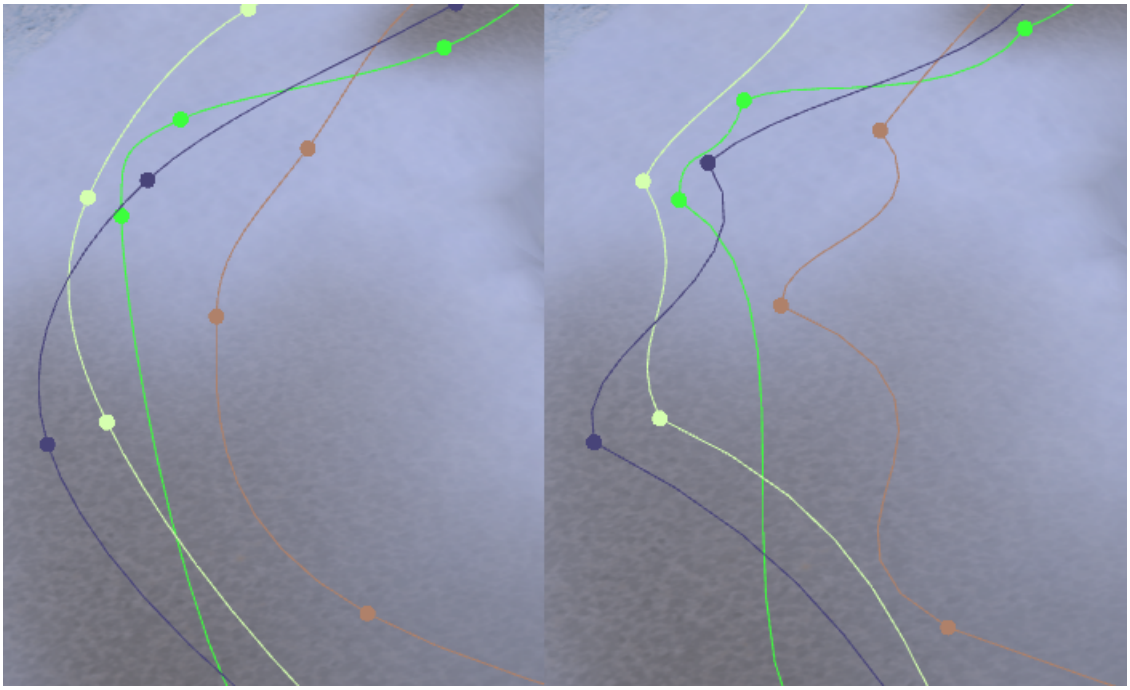
$$\begin{aligned} l_i &= \tfrac{(1-a)(1+b)(1-c)}{2}(P_i - P_{i-1}) + \tfrac{(1-a)(1-b)(1+c)}{2}(P_{i+1} - P_i) \\ r_i &= \tfrac{(1-a)(1+b)(1+c)}{2}(P_i - P_{i-1}) + \tfrac{(1-a)(1-b)(1-c)}{2}(P_{i+1} - P_i) \end{aligned} \quad (2.8)$$

Při parametrech $a = 0$ $b = 0$ $c = 0$ jsou tak hodnoty vstupního a výstupního vektoru určeny pouze polohou uzlových bodů.



Obrázek 2.3: Ukázka ovlivnění parametry průběh interpolace. Převzato z webu. [3]

[3] https://en.wikipedia.org/wiki/Kochanek-Bartels_spline



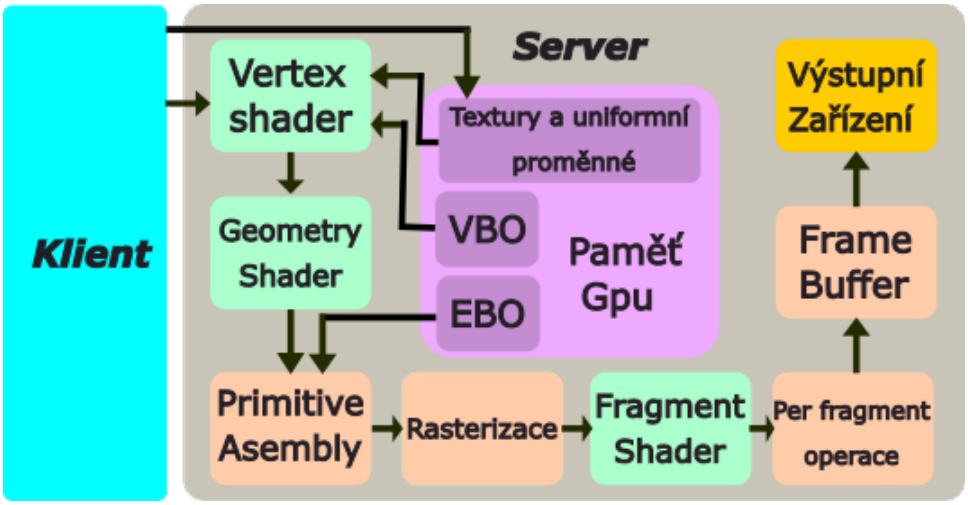
Obrázek 2.4: Srovnání křivek Kochanek-Bartels s Kardinální křivkou

2.9 OpenGL

OpenGL je rozšířené grafické API pro práci s 3D a 2D grafikou. Od verze 2.0 používá programovatelnou pipeline. V práci je využita verze 3.2, která obsahuje volitelný **Geometry shader**. Shadery jsou programovatelné části pipeline a v práci jsou využity tyto tři:

- **Vertex shader** se využívá typicky pro aplikaci transformačních matic. Počítá se v něm tzv. per-vertex část osvětlení. Ve vertex shaderu jsou vertexy zpracovávány paralelně a programově je vždy přístupný právě jeden. Přiřazením do speciální proměnné *gl_Position* je vertex předán na výstup shaderu.
- **Geometry shader** je volitelný shader *OpenGL Pipeline*. Sídlí mezi *Vertex shaderem* (pokud není použita další volitelná fáze *Tessalation shader*)[1] a *Vertex Post-Processing* fází. Shader umožňuje generovat ze vstupního primitiva nula nebo více primitiv. Jeho primární funkcí je **Transform feedback** (umožňuje data zapsat zpět do bufferů a zpřístupnit je *CPU*), která není v práci využita.
- Ve **Fragment shaderu** je často počítáno osvětlení.[2] Jeho vstupem jsou fragmenty, z předešlého stupně pipeline, rasterizace. Ve fragment shaderu je přiřazena barva jednotlivým fragmentům a hodnota hloubky. Fragment shader je nepovinná část pipeline.

[1] Ve verzi 3.2 není *Tessalation shader* standardně dostupný
[2] Per-vertex osvětlení je počítáno ve **Vertex shaderu**



Obrázek 2.5: Vizualizace pipeline OpenGL 3.2

OpenGL podporuje širokou škálu programovacích jazyků a platforem. Kromě standardní verze *OpenGL* existuje *OpenGL ES*, které je používáno pro mobilní telefony, herní konzole nebo pro vestavěná zařízení a *WebGL* pro akcelerování grafiky ve webových prohlížečích. Nově je také dostupné *API Vulkan*, které je zaměřeno svojí složitostí a možnostmi spíše na velké firmy.

Kapitola 3

Návrh aplikace

Kapitola se věnuje návrhu herní aplikace vzhledem ke kladeným požadavkům na hru.

3.1 Herní mechanismy a návrh

Hra je simulací modelu závodu motokár. Tak, jako v každých závodech, motokáry závodí na závodní dráze, která je vymezena na vygenerovaném povrchu. Povrch je možné buďto generovat obrázku. Ve hře může každý hráč využít i tzv. **power-upy**. Ve hře jsou zastoupeny dva herní módy, a to závod proti oponentům a závod na čas, při kterém se snaží za co nejkratší čas dokončit trasu a sebrat přitom co nejvíce mincí.

Power-up

Power-up je reprezentován pomocí modelu na trati, který lze sebrat v průběhu závodu a zvýšit tak své šance na vítěství. Pokud hráč projede přes model, je danému hráči přidána možnost užití sebraného *power-upu*, indikovaného ikonou v *Gui*. Model je následně odebrán ze scény, a to včetně jeho fyzikální obálky na předem stanovenou dobu.

Power-upy jsou celkem tři, dva umožňují dosáhnout vyšší rychlosti pohybu. Rozdíl mezi nimi je malý, první aktivuje možnost dosáhnutí vyšších rychlostí vozidla po omezenou dobu hned při jeho přejetí a druhý, zrychlovací *power-up* je uložen pro pozdější užití. Aktivace zrychlujících efektů je indikována efektem ohnivého kouře, vystupujícího se z výfuků vozidla vytvořeného pomocí emitoru částic.

Třetím *power-upem* je raketa, která má za úkol penalizovat hráče, kteří s ní budou kolidovat. Při vystřelení rakety je vytvořena instance modelu rakety, a dvou instancí emitoru částic, a to kouře a exploze. Jak raketa letí dopředu, emitor kouře generuje částice a simuluje tak kouř. Pokud dojde k vylétnutí rakety mimo dráhu, nebo kolizi rakety vozidlem, dojde k explozi. Pří explozi se raketa postupně zmenšuje až zmizí úplně a mezitím emitor exploze zobrazí předgenerovanou sadu částic. Pokud nastala kolize rakety a vozidla hráče, je hráč přesunut na naposledy projetý *checkpoint*.

Checkpoint

Je významné místo, kterým hráči musí projet. Při projetí hráčů *checkpointy* jsou zaznamenány jejich časy, případně kola. Díky tomu, že hráči musí projet korektní sekvencí *checkpointů*, si hráči nebudou moci zkracovat trať, neboť kolo pak není započítáno a hráč musí

Obrázek 3.1: Letící raketa na zrychlující vozidlo

kolo jet znovu. Kromě toho jsou opěrnými body v prostoru a využívají se pro určení pořadí vozidel při jejich kolizi.

Vymezení tratě

Závodní trať je třeba vymezit primárně proto, aby hráči nemohli vynechat kusy tratě a aby rakety, které hráči mohou vypálit, mohli být zničeny a odebrány tak ze scény co nejdříve a tím šetřili paměť a výkonnostně zatěžovali počítač co nejméně.

K vymezení tratě se nabízí obalová tělesa, ale k jejich použití by bylo nutné je nejdříve vytvořit externím modelovacím editoru. Lepším řešením proto je vytvoření *kolizní mapy*. To je textura, na níž je vyznačena validní trasa vybranou barvou. Ta je vzorkována na pozici hráče. Při získání specifické hodnoty se potom rozhodne, zdali je vozidlo stále přítomno na validní trati. Postup je výrazně jednodušší už jen kvůli nutnosti vyrobení jednoduché textury oproti vytvoření značně sofistikovanější obálky kolem celé tratě.

Oponenti

Oponenti by se měli chovat velmi podobně jako hráč, udržovat směr a rychlost auta nebo použít získaný *power-up*. K určení správné orientace vozidla dopomáhá oponentům křivka. Díky ní je pak oponent schopen jednoduše korigovat svůj směr a nestane se, že by vyjel z dráhy. Oponenti, stejně jako hráč, musí reagovat na vnější podněty hry, jako jsou kolize mezi sebou a hráčem, na kolizi s raketou, nebo na možnou změnu rychlosti díky získání zrychlujícího *Power-upu*.

3.2 Editor úrovní

Při vytváření složitějších her je herní editor nutností. Tyto editory jsou navrženy pro specifické potřeby každé hry. Některé umožňují editaci herních misí (např. editor hry *warcraft3*), jiné umožňují tvořit mapy.

Motivace k implementaci editoru map je jednoduchá a rychlá tvorba závodních úrovní. Proto je kladen v editoru důraz na jednoduché používání. Jednotlivé entity je například možné posouvat po herní mapě po výběru entity myší. Pak entita kopíruje pohyb myši. Entity mají sadu parametrů, které je vhodné nastavit. Nastavení těchto parametrů je možné v externím souboru, ale není to nezbytně nutné (až na jeden případ). Editor dovoluje nastavení těchto parametrů skrze jeho Gui.

Mezi modifikovatelné parametry v editoru je pozice modelů, světel, nastavení jejich rotace v jedné ose, změny měřítka. Lze přidat další instance modelů a změnit velikost kolizní obálky, což afektuje všechny instance daného modelu. Zároveň pak je možné vytvářet a mazat křivky, které jsou použity pro pohyb oponentů.

Některé herní entity mají předdefinované chování (např. *powerupy*). Je tedy nutné u entit mít parametr, který bude rozlišovat, jak se má entita při závodění chovat.

Výčet podporovaných entit:

- **Model**

  Standardní model

- **Model s osvětlením**

  Je stejný jako standardní model, navíc ale je emitorem světla. Barvu světla je třeba nastavit v souboru, implicitně je světlo vypnuto.

- **Emitor částic**

  Emitor částic je velmi často využit s konkrétním vozidlem, dle kterého nastavuje svou vlastní pozici ve scéně a proto emitor pouze umožňuje jeho přidání do scény a všechny jeho parametry je třeba manuálně nastavit v externím souboru.

- **Terén**

  Terén je generován z textury, v současné době podporuje pouze 8-bitovou hloubku textury, ze které vznikne hrubší terén. Ideální rozlišení výškové mapy je 256x256 a 512x512. Výšková mapa ve vyšších rozlišeních působí při aplikaci rotace dle topologie terénu na model rušivě, a je tedy vhodné alespoň terén zvětšit.

- **Skybox**

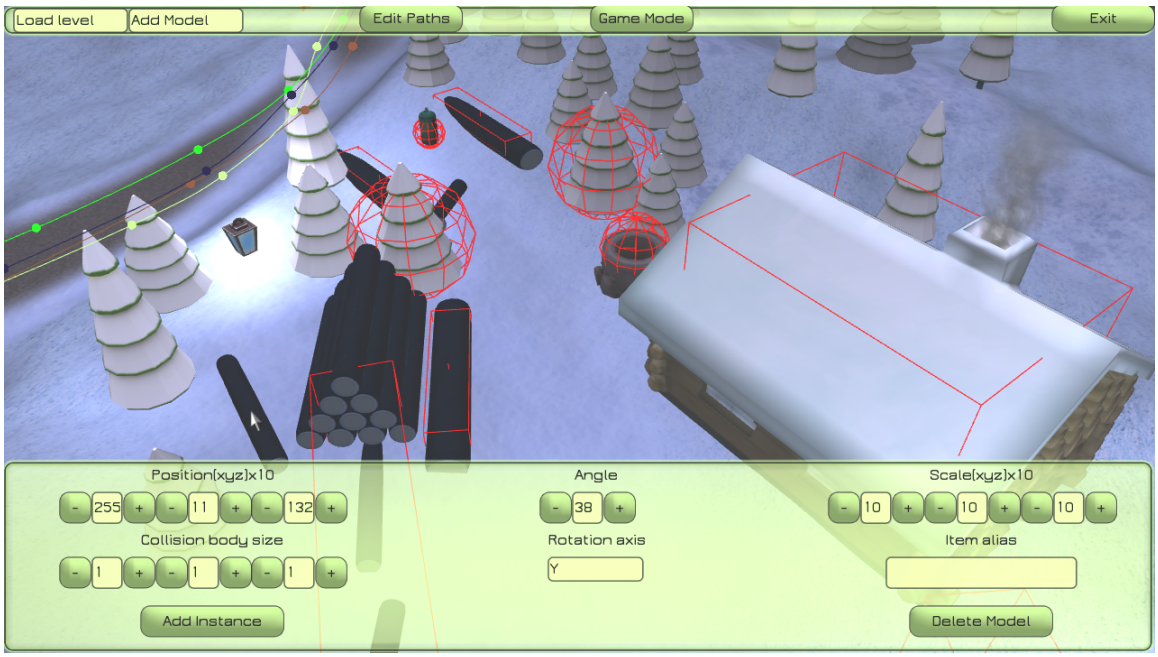
  Skybox reprezentuje pozadí herní scény. Zde je potřeba specifikovat správně celkem šest textur, pro každou stranu jednu.

Libovolné instanci vybraných entit(model, emitor částic, světlo) lze přiřadit alias, kterým lze specifikovat úlohu, kterou bude objekt ve hře plnit. Některé příklady:

- **PLAYER** Vybraná instance modelu bude ovládána hráčem.
- **POWERUP _ BOOST** Při kolizi vozidla hráče a modelu zrychlujícího *power-upu* je hráči přidána možnost jej později použít. Instance modelu má předdefinovanou animaci a rotuje na místě.

- **CHECKPOINT _ START**: Definuje start a zároveň cíl.
- **CPU _ N**: Reprezentuje `N`-tého oponenta. Pokud se stejný *alias* vyskytuje ve křivkách, jsou křivky přidány k oponentovi jako jeho dráha.
- **R _ CAR _ SMOKE** Emitor částic je použit jako šablona pro vytvoření počtu instancí dle počtu hráčů kouř z pravého výfuku. Emitor pak automaticky udržuje pozici kolem pravého výfuku. Tento emitor funguje jako šablona a všem hráčům jsou automaticky vytvořeny instance se stejnými parametry.

Tohle je pouze výběr z více možností, zachycující popis chování entit editoru. Některé entity jako například emitor částic má více parametrů, které je vhodné specifikovat, jako např. počet generovaných částic za jednotku času. Nastavení těchto parametrů editor neumožňuje, a je třeba je nastavit ve výstupním souboru.



Obrázek 3.2: Prostředí editoru

Výstupní soubor

Výstup je realizován do *xml* souboru, ten zajišťuje dobrou čitelnost a možnost rychlé editace. Do výstupního souboru je zapsáno pouze po zvolení možnosti **Save** v editoru. Pak jsou uloženy parametry a typy všech načtených entit ve scéně. Ke čtení *xml* souboru je využita knihovna *libxml2*.

Kapitola 4

Implementace

Kapitola se zabývá popisem implementace nastudovaných algoritmů. K implementaci je využito objektově orientované paradigma jazyka *C++* a návrhového vzoru *Singleton*.

Použité knihovny

Program využívá knihovny *SDL* pro vytvoření okna, knihovnu *Assimp* pro načítání modelů, a knihovnu *Bullet* k detekci kolize.

Načtení Gui

Při puštění aplikace jsou vytvořeny jednotlivé ovládací prvky hlavního menu, editoru, a zobrazovací prvky při závodění. Proto se načtení seznamu jednotlivých map (pro každou mapu je pak vytvořeno tlačítko se jménem mapy) děje při spuštění hry. Pokud tedy je vytvořena a uložena nová mapa v editoru, aby byla zpřístupněna ke hraní, je třeba hru restartovat.

4.1 Načítání Scény

Scéna je načtena ze vstupního *xml* souboru. Při načítání modelů zároveň inicializuje přenos dat modelu (vertexů, uv souřadnic, normál) na grafickou kartu. Scéna dle potřeby(resp. podle parametru *name* u herních entit) vytvoří vazby mezi objekty hráčů a entitami.

4.2 Herní smyčka

Nejdříve je zkontrolováno, zda-li nemá být přidána nějaká entita. V herní scéně to může být raketa, která byla vystřelena některým z hráčů, nebo to mohou být jednotlivé *power-upy* na mapy. Stejně tak mohou být některé entity odebrány. V herní scéně je to většinou opět raketa, nebo dříve sebraný *power-up*. V editoru to pak může být libovolná entita. Následuje fáze přečtení vstupů od hráče.

Na základě získaných vstupů a kontextu herní scény může být aktualizován pohyb hráče a oponentů. Hráč totiž mohl hru pozastavit stiskem klávesy *escape* a v tom případě nejsou entity aktualizovány (mimo emitory částic). Následně je proveden krok fyzikální simulace

knihovny bullet.

Algoritmus 1: Pseudokód hlavní smyčky programu

```
while !quit do
    Scéna->PřidejEntity();
    Scéna->OdeberEntity();
    HlavníHráč->PřečtiVstupy();
    HlavníHráč->VýpočetPohybu();
    for i ← 0 to PoetHr do
        Hráč->VýpočetPohybu();
    end
    Bullet->KrokSimulace();
    Skybox->Vykresli();
    Lights->SeřaďSvětla();
    Terén->Vykresli();
    Modely->Aktualizuj();
    Modely->Vykresli();
    Světla->Aktualizuj();
    Světla->Vykresli();
    Emitory->Aktualizuj();
    Emitory->Vykresli();
    Gui->Vykresli();
end
```

Před vykreslením entit jsou aktualizovány *modelové matice* aktivních herních entit. To jsou takové entity, které se pohybují nebo nějak interagují s hráči, např. částicové efekty, *power-upy*, a modely přiřazené jednotlivým hráčům. V editoru jsou aktivní všechny modely, protože je možné měnit jejich parametry.

4.3 Vykreslení modelů a světel

Světla se od modelů liší pouze přítomností informaci o barvě světla. Před jejich vykreslením je provedeno jejich seřazení relativně ke kameře.

Metoda *Aktualizuj()* vytvoří modelové matice z parametrů instancí. Pak před vykreslením jsou nastaveny uniformní proměnné (*modelová, projekční a view* matice). Ve *vertex shaderu* jsou *vertexy* těmito maticemi transformovány. *Fragment shader* vzorkuje texturu na *uv* souřadnicích. Ve hře je implementován *Phongův* osvětlovací model bez spekulárních odlesků s podporou více světel. Počet světel je omezen na čtyři dynamická plus jedno globální, a to kvůli výkonu. Hra podporuje přidání neomezeného množství světel a proto jsou do *fragment shaderu* poslány pozice nejbližších čtyř světel vzhledem ke kameře.

4.4 Ovládaný hráč

Herní entita hráče reaguje na stisky kláves, kdy klávesy **w s a d** slouží k pohybu a zatáčení, stisknutím **mezerníku** hráč vyskočí a stiskem klávesy **shift** je použit *power-up*, v případě, že ho hráč dříve získal. Následně jsou předány reference všem hráčům na herní entity, a to dle parametru **name** entity.

Obrázek 4.1: Scéna hry s oponenty a zdrojem světla

- **PLAYER** nebo **CPU _ N**:

  Herní entita typu model předá hráči referenci na sebe. Hráč pak ovládá její transformace na základně stisknutých kláves a dění ve hře. Pokud byl textový řetězec instance herní entity **CPU _ N**, namísto přiřazení k hráči je vytvořen nový oponent, ke kterému je následně přidána. Tato herní entita reprezentuje vozidlo.

- **ADIN _ PLAYER** nebo **ADIN _ CPU_N**

  Herní entita je přidána buďto ke hráči, nebo k oponentovi. Je umožněno mít tak další model, jenž bude přiřazen k hráči. Ve hře je možnost použita k přiřazení postavy závodníka k vozidlu. Je tak možné mít jednu instanci herní entity vozidla, ke které je přidána libovolná instance entity postavy.

- **ROCKET _ FIRING, ROCKET _ EXPLOSION, ROCKET _ SMOKE**

  Všechny dílčí entity jsou automaticky vygenerovány v editoru a přidány do scény. Všechny dohromady popisují vizuální reprezentaci letící rakety.

- **L _ CAR _ FIRE, R _ CAR _ FIRE, L _ CAR _ SMOKE, R _ CAR _ SMOKE**

  Opět automaticky vygenerovány, jsou to částicové emitory k zobrazení efektu kouře a ohně.

Zatáčení hráče

Zatáčet lze pouze při dopředném nebo zpětném pohybu hráče. Při stisku příslušných kláves přičtena předem definovaná konstanta ke všem herním entitám náležícím hráči.

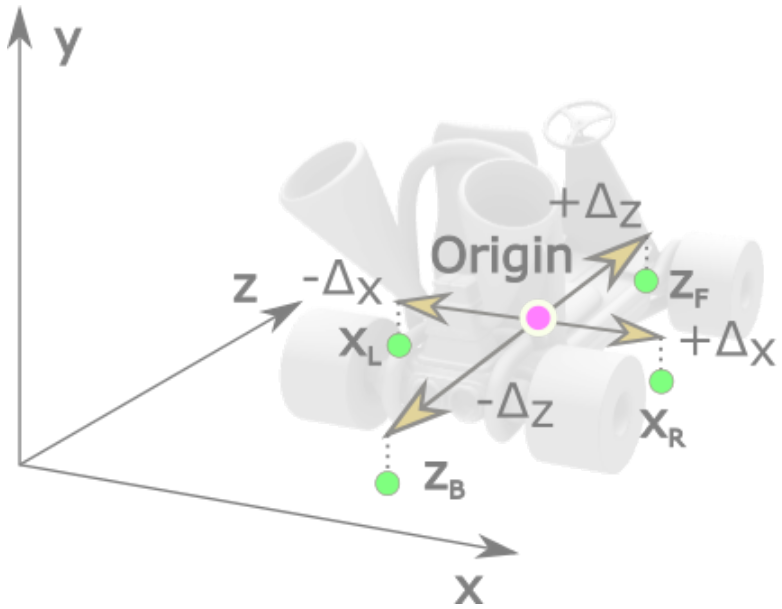
Pohyb hráče

Podobně jako u zatáčení hráče je definován rychlostní přírůstek, který je přidán k celkové rychlosti po stisku příslušných kláves. Hráčům je průběžně nastavována souřadnice y dle výšky terénu na pozici (implementace v sekci 4.7)

Hráč rozlišuje několik stavů skoku. Při standardním skoku (po stisknutí mezerníku) je nastavena rotace auta po celou dobu jeho trvání. Při sjezdu vysokého srázu je naopak aplikována poslední rotace auta k terénu po celou dobu pádu.

Rotace vozidla dle povrchu terénu

Vozidlo se snaží natáčet tak, aby jeho orientace odpovídala topologii terénu. Výšky terénu jsou získány na čtyřech souřadnicích, jak ukazuje schéma 4.2.



Obrázek 4.2: Získání souřadnic terénu k rotaci vozidla

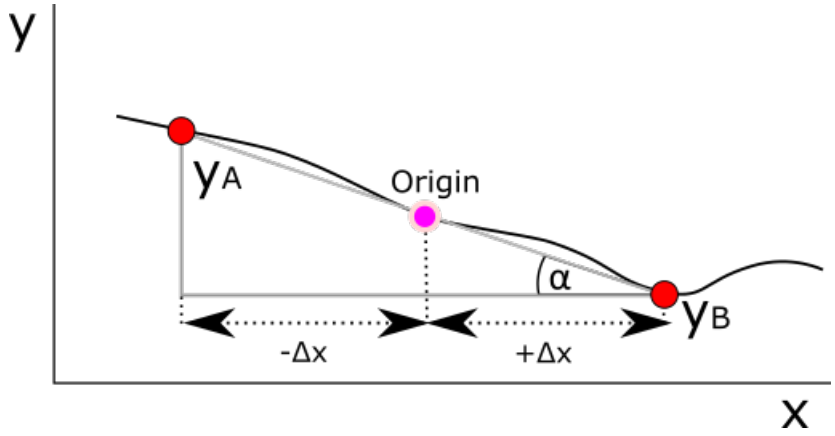
Pak je pomocí trigonometrie zjištěn úhel natočení. Obrázek 4.3 ukazuje zjištění úhlu z jedné strany. Výsledný náklon vozidla je složenou rotací a je vyjádřen pomocí kvaternionu.

Kamera

V herním režimu kamera sleduje pohyb hráče a je vždy za ním. Přírůstek pozice kamery je vyjádřen vztahem:

$$\vec{\Delta p} = (d * sin(\alpha), d * cos(\alpha)) \tag{4.1}$$

kde α je úhel natočení hráče a d je vzdálenost mezi kamerou a hráčem. Tu lze kolečkem myši měnit. Výsledné přírůstky jsou odečteny od pozice kamery. Rotace kamery je nastavena dle úhlu α.



Obrázek 4.3: Získání souřadnic terénu k rotaci vozidla

4.5 Oponenti

Při načítání hry byly oponentům přidány sloty s křivkami, které jim pomáhají k pohybu. Pokud oponentovi nebyl slot přidělen, zůstává stát a nehýbe se.

Algoritmus 2: Pseudokód výpočtu pozice počítačem řízeného hráče

Data: $pozice, uhel, rychlost, interval, tuhostKrivky$
Result: Aktualizované hodnoty $pozivce, uhel, interval$
start:
$\Delta procenta = RychlostNaInterval(rychlost, interval)$;
$procenta+ = \Delta procenta$;
if $(procenta > 1)$ **then**
  $procenta = 1$;
  $novaIterace = true$;
  $interval+ = 1$;
end
$bodKrivka = CatmullRom(interval, procenta)$;
$\Delta pozice = DiferencePozice(bodKrivka, pozice, procenta)$;
$\Delta uhel = DiferenceRotace(bodKrivka, pozice, uhel, tuhostKrivky)$;
$pozice+ = \Delta pozice$;
$uhel+ = \Delta uhel$;
if $novaIterace = true$ **then**
  go to start;
end

Křivka je rozdělena na intervaly, vymezené jejími kontrolními body. Metoda *RychlostNaInterval()* změří, zda-li lze posunout hráče o vzdálenost vyjádřenou *rychlostí* na *intervalu* křivky. A pokud by pozice přesahovala hranice intervalu, inkrementuje *interval* tak dlouho, dokud není splněna podmínka $speed < delkaIntervalu$. V takovém případě nastaví $rychlost = rychlosu - i * delkaIntervalu_i$, kde i je dáno počtem intervalů a $delkaIntervalu_i$ jejich délkou. Výsledný posuv je pak vyjádřen v procentech vůči *intervalu*.

Pokud se hráč nacházel ve stejném intervalu už při poslední iteraci algoritmu, je nová procentuální pozice přičtena k předchozí. Jestli-že hodnota překročí interval(>1), je algoritmus spuštěn znovu s aktualizovanými hodnotami.

Po interpolaci s vypočtenými parametry je znám cílový bod ležící na křivce. Tento bod určuje směr pohybu.

$$\vec{\Delta p} = ||(c - p)|| * v \tag{4.2}$$

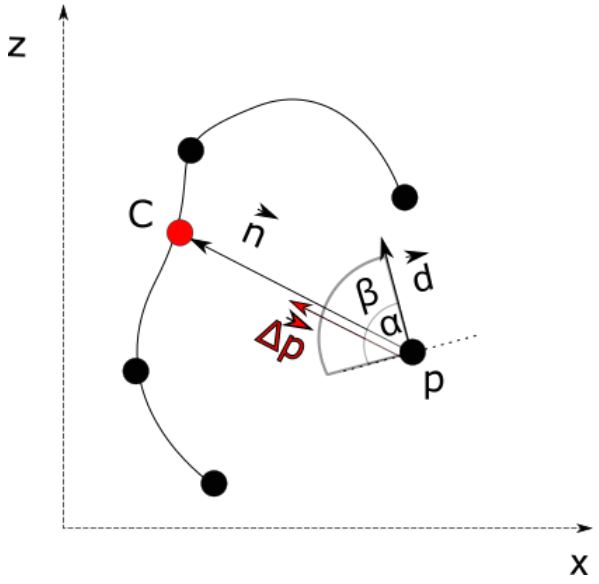
Rovnice popisuje vytvoření směrového vektoru od pozice hráče p k cílovému bodu c na křivce. Směrový vektor je následně zvětšen o rychlost v. Výsledný vektor $\vec{\Delta p}$ je přičten k původní pozici. Pro novou pozici je zjištěna výška terénu a dle ní nastavena příslušná souřadnice vektoru pozice. Ke správnému určení rotace je nejprve vyjádřen směr hráče $\vec{d}$ dle jeho orientace.

$$\vec{d} = (cos(\alpha), sin(\alpha)) \tag{4.3}$$

Následně je vytvořen směrový vektor $\vec{n}$ a vypočítán výsledný úhel β. Konstanta t značí tuhost rotace. Nižší hodnoty tuhosti znamenají nižší výsledný úhel, a vozidlo se natáčí pomaleji. Pro $v = 30$ je vhodným koeficientem tuhosti $t = 30$.

$$\vec{n} = \vec{c} - \vec{p} \tag{4.4}$$

$$\beta = t * \frac{d_x * n_x - d_y * n_y}{\hat{d} * \hat{n}} \tag{4.5}$$



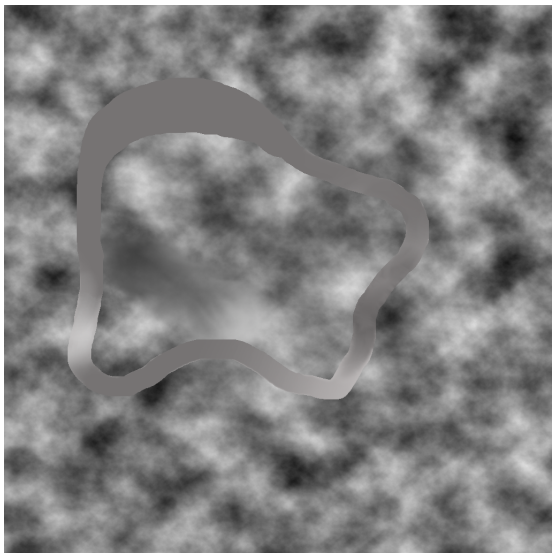
Obrázek 4.4: Schéma výpočtu posuvu a rotace protivníků

4.6 Terén

Program vytvoří trojúhelníkovou mříž, kde hustota trojúhelníků koreluje s rozlišením textury. Příliš vysoké rozlišení může mít za následek zbytečně velké paměťové nároky. Normála bodu je vypočítána z jeho okolí.

Výšková mapa

Standardní výšková mapa je v odstínech šedé, kde bílá barva reprezentuje nejvyšší možnou výšku vertexu (parametr *MaxHeight*), černá reprezentuje naopak nultou. Terén ze standardní 8-bitové textury má tendenci vypadat hrubě, a proto je na texturu aplikován konvoluční aproximační filtr, který redukuje výškové skoky.



Obrázek 4.5: Příklad výškové mapy

Multitexturing a Tiling

Při aplikaci textury by textura by byla protažena, což by se projevilo ztrátou detailů. Pro jejich zachování by musela textura být obřích rozměrů, což by si vyžádalo velké množství paměti. Řešením je *Tiling*, opakované nanesení malé bezešvé textury v libovolném měřítku. Detaily tak nejsou ztraceny a textura je stále ostrá. Vzniká však alias, lze na terénu vidět šachovnicový vzor. Aby byl tento dojem co nejmenší, terén využívá *Multitexturing*, a je na něj aplikováno více textur najednou. Poměr barev z jednotlivých textur určuje *blendmapa*.

Implementovaný *Multitexturing* podporuje celkem čtyři textury, pro každý barevný kanál jednu,[1] plus základní. Navíc zachovává gradient a přechody textur jsou tak plynulé.

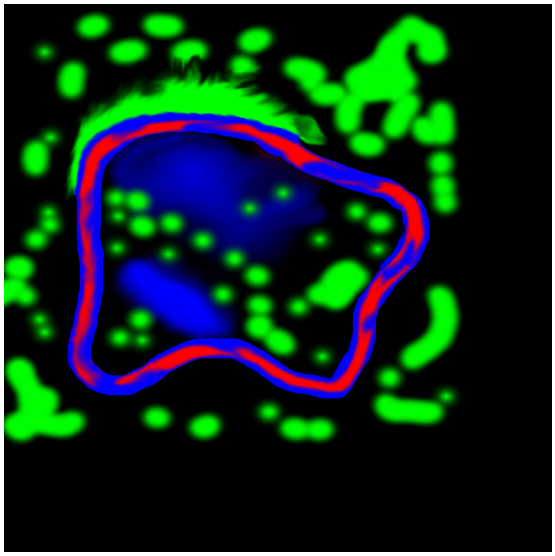
Zjemnění terénu

Při generování rotačního kvaternionu pro vozidla je díky zvolené metodě jeho výpočtu problém. Vozidlo sice kopíruje svojí rotací topologii terénu, ten je však díky použití výškové mapy s 8-mi bitovou hloubkou velmi hrubý. Proto je při načtení výškové mapy na vertexy terénu aplikován filtr s dolní propůstí.

Byly vyzkoušeny celkem dvě varianty[2], první varianta používala konvoluční jádro s diskrétní aproximací *Gaussova rozložení*, které váženě průměruje jednotlivé vertexy.[4] Druhá varianta, která byla nakonec v programu ponechána, je standardní aproximační filtr.

[1] Mluvíme o RGB prostoru, ne RGBA

[2] Obě používají konvoluční jádra o velikosti 5x5



Obrázek 4.6: Blendmapa

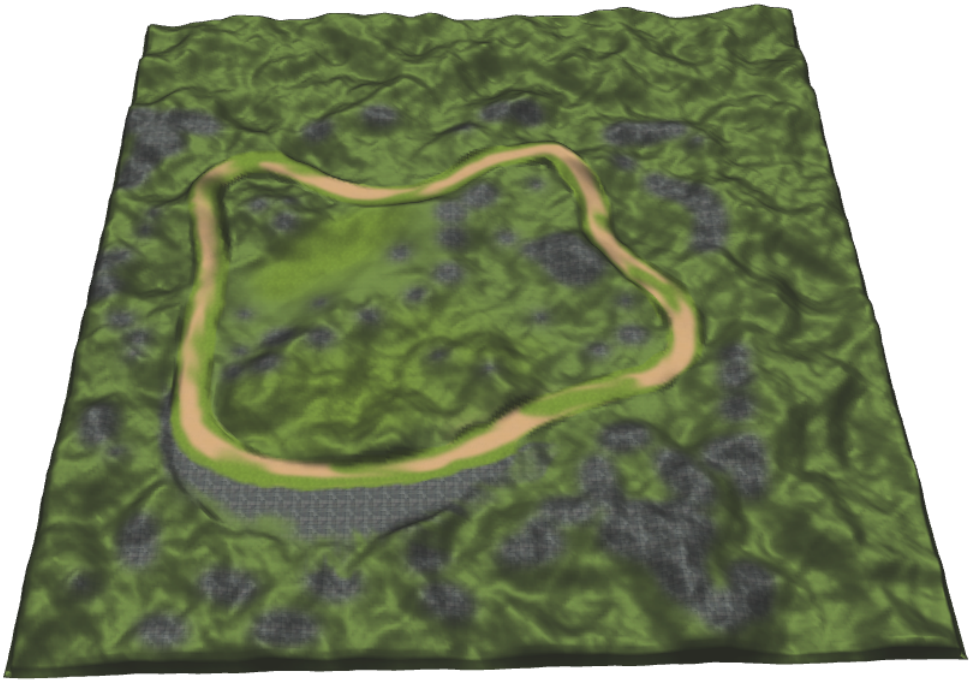
4.7 Kolize

Kolizi lze ve hře dělit do tří skupin, a na to kolize herních entit (respektive jejich modelů ve scéně), kolizi herních entit a terénu a na kolizní mapu.

Kolize mezi entitami

K detekci kolize mezi entitami je použita knihovna *Bullet*. Kolize je detekována pomocí tzv. *callback funkcí*, které mohou být kdykoliv invokovány. Při kolizi v závodě na sebe entity reagují dle skupin:

- **Hráči naráží mezi sebou:** Při kolizi oponentů a hráče je zjištěna vzdálenost mezi nimi a nejbližším následujícím *checkpointem*. Hráči, kterému náleží vozidlo, nacházející je blíže *checkpointu*, je zvýšena rychlost. Hráči, se vzdálenějším vozidlem je rychlost snížena.
- **Raketa a hráči:** Vystřelená raketa je po malém časovém zpoždění zničena (časová prodleva je kvůli animaci exploze). Hráč je, stejně jako při vyjetí z dráhy, přesunut na poslední *checkpoint*. Zničení rakety je provedeno buďto při kolizi s hráčem, kolizí s jiným objektem, nebo při vyletění rakety mimo dráhu.
- **Hráči a *power-up*:** Hráčům je přidán *power-up*, který lze následně mohou použít. Pokud hráč už *power-up* měl, není mu přidán. *Power-up* je odebrán ze scény na krátkou dobu, aby jej hráč nemohl používat opakovaně. Také je díky tomu redukován maximální počet raket ve scéně a tedy jsou sníženy paměťové nároky. Po několika vteřinách je *power-up* znovu přidán do hry.
- **Hráči a *checkpoint*:** Hráči je aktualizováno počítadlo *checkpointů*, pouze v případě, že hráč popořadě projel jednotlivými *checkpointy*. Je-li *checkpoint* startovacím *checkpointem*, je hráči připsáno kolo, což je indikováno zobrazením času kola v *Gui* a když je počet kol roven třem, hra je ukončena tabulkou výsledků.



Obrázek 4.7: Výsledný otexturovaný terén

Kolize modelů a terénů

Pohyb modelu v osách **x, z** je plně vymezen reakcemi na vstup uživatele a v ose **y** je vymezen částečně (v sekci 4.4). Pokud hráč momentálně není ve stavu skoku, je jeho pozice nastavena podle *y-nové* souřadnice terénu, po kterém jezdí.

Pro zjištění výšky stačí zjistit pozici trojúhelníku, odpovídajícímu zadaným souřadnicím. Výsledná hodnota je rovna interpolací výšek dříve zjištěného trojúhelníku, což popisuje algoritmus 3.

Algoritmus 3: Pseudokód pro zjištění výšky terénu na konkrétní pozici

```
Data: XZ vektor Pozice
Result: Interpolovaná Vyska
if (!NaTerenu(Pozice)) then
    Vyska = 0;
else
    velikostCtverce = TerrainDimensionSize/TerrainDimensionCount ;
    Grid = floor(Pozice.x/velikostCtverce), floor(Pozice.z/velikostCtverce) ;
    if (Grid.x <= 1 − Grid.z) then
        Vyska = BaricentrickaInterpolace(VyssiTrojuhelnik(), Grid);
    else
        Vyska = BaricentrickaInterpolace(NizsiTrojuhelnik(), Grid);
    end
end
```

Kromě zjištění výšky lze použít stejný přístup pro získání hodnoty kolizní mapy dráhy.

Kolizní mapa

Kolizní mapa umožňuje vymezit specifickou oblast části mapy. Podle interpolované hodnoty, která je z ní získána, je určena platnost polohy vozidla. Pokud je hodnota ve správném intervalu, vozidlo se pohybuje na mapě. Při jiných hodnotách vozidlo vyjelo z dráhy a musí být vráceno na poslední *checkpoint*. Přidáním dalších intervalů by bylo možné reagovat na jiné podněty na dráze. Kolizní mapou lze tedy řešit kolize objektů, ale pouze v 2D prostoru.

Obrázek 4.8: Kolizní mapa

4.8 Skybox

Je krychle, na jejíž strany jsou aplikovány navazující textury pozadí herní scény. *Skybox* opticky vypadá vzdáleně a to nezávisle na pohybu kamery. Při hrách s otevřenými světy je standardně od poloviny výšky obrázku aplikován barevný přechod v barvě mlhy. Efekt mlhy umožňuje dynamicky přikreslovat modely, aniž by byl moment viditelný. Protože jednotlivé mapy jsou relativně malé, efekt mlhy (aplikace gradientu barvy při zobrazování textur na základě vzdálenosti modelu od kamery) není využíván.

4.9 Částicové efekty

Částicový emitor je objekt co zastřešuje správu částic ve scéně. Atributem *pps* jsou udává počet vygenerovaných částic za vteřinu.

Generování částic

Emitor generuje částice z jednoho bodu. Při vygenerování nové částice je vytvořen náhodný vektor z intervalu udaného směrovými vektory *minMaxSide* a *minMaxDepth*. Zkombinovaný s parametrem *height* vytvoří vektor rychlosti. Kromě rychlosti je částici při jejím vytvoření náhodně vygenerována rotace, kterou si pak udržuje po celou dobu života.

Nová pozice částic

Nová rychlost je aktualizována podle gravitace $g = 10$ a parametru *gravityAfection* g_a ovlivňující gravitaci 4.6.

$$v_{i+1} = v_i + g * g_a * \Delta t; \tag{4.6}$$

Rychlost je následně přičtena k pozici.

Generování primitiv v Geometry shaderu

Do *geometry shaderu* jsou posílány jednotlivé pozice částic. V shaderu jsou vygenerovány čtyři body výsledného čtverce a k nim *uv* souřadnice. Souřadnice jsou vytvořeny dle poměru momentální k maximální délce života částice. Tímto poměrem L a počtem obrázků C(pro každou dimenzi zvlášť) v *texture atlasu* lze získat jeho index I (vztah 4.7). *Texture atlas* obsahuje vždy **stejný** počet obrázků v dimenzích.

$$I = L * C \tag{4.7}$$

Podíl Indexu a počtu horizontálních obrázků určí uv_x a operací modulo indexu a počtů svislých obrázků je určena uv_y (vztah 4.8).

$$\vec{uv_{rel}} = (\frac{I}{C_x}, I - \frac{I}{C_y} * C_y) \tag{4.8}$$

Po získání relativního $\vec{uv}$ je aplikován na rohy obrázku a jsou tak získány finální uv_f souřadnice bodu. Příkladem může být horní levý roh (vztah 4.9).

$$uv_f = (\frac{uv_{relx}}{C_x}, \frac{uv_{rely}}{C_y}) \tag{4.9}$$

Aby byly přechody mezi jednotlivými obrázky plynulé, jsou generovány *uv* souřadnice pro index I a I_{+1}. Výsledná barva textury je tak interpolována mezi uv_f a uv_{f+1}.s

Rotace částic

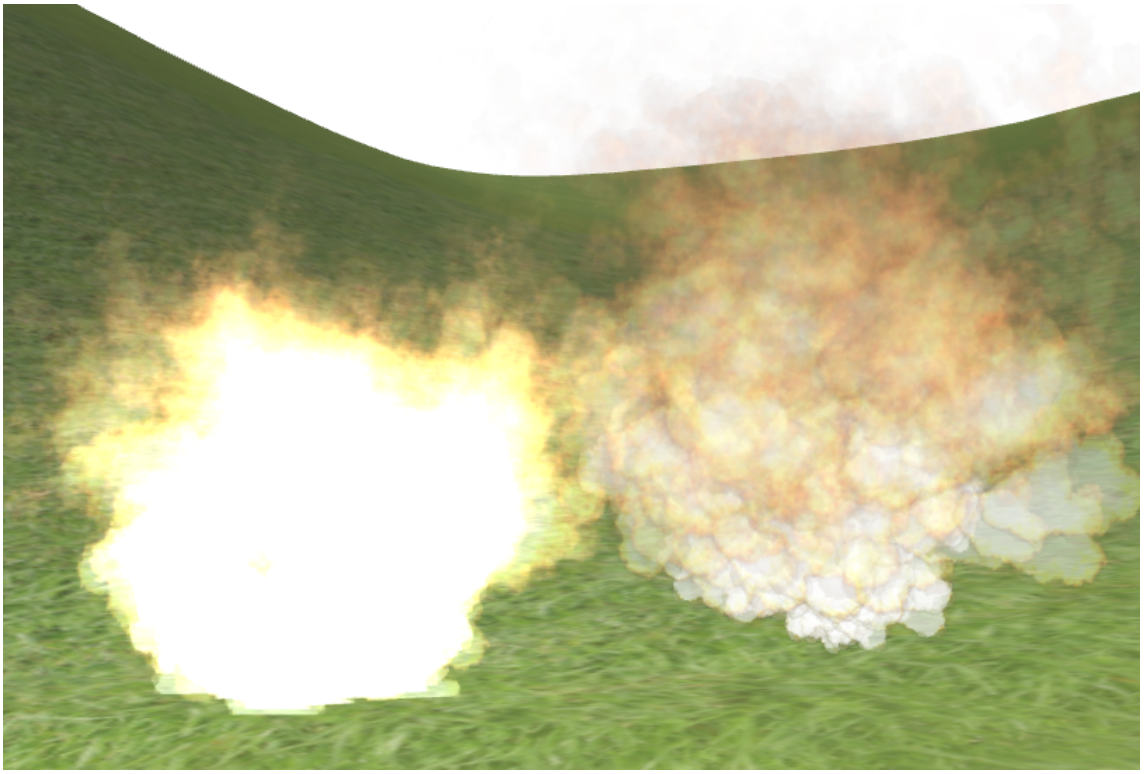
Vizuálně jsou částice pouze 2D obrázky, a proto je vhodné, aby byly vždy otočeny směrem ke kameře. Jednou z možností, jak docílit „vypnutí“ rotace v nepotřebných osách je odstranění rotace částice vzhledem ke kameře, což zobrazuje vztah 4.10. Aby bylo správně zachováno pořadí multiplikace matic, je *modelová* matice rozdělena na dílčí transformace (T je matice posuvu, S je matice změny měřítka a R_z je matice rotace v ose z).

$$gl_position = M_P * M_V * T * S * M_V^{-1} * R_z * v_i \tag{4.10}$$

Částice je tak vždy otočena ke kameře. V shaderu se dynamicky mění velikost částice v závislosti na její délce života.

Viditelnost a blending částic

Obrázky částic obsahují *alfa kanál*, aby byla zachována iluze sofistikovaného objektu, namísto obyčejného čtverce s nanesenou texturou. To je ovšem problém při použití **Depth**



Obrázek 4.9: Částicové emitory se standardním mícháním barev(vlevo) a aditivním(vpravo)

testu[3]. Se zapnutým **Depth** testem nejsou překryté části částic vykresleny i když je překrytá část částice průhledná. Pokud je **Depth** test vypnut, dochází k vizuálním artefaktům. Řešením může být buďto seřazení částic vzhledem ke kameře (což je při vyšším počtu částic výpočetně náročné), nebo použít *aditivní blending*[4]. *Aditivní blending* je nezávislý na pořadí, což je velmi výhodné. Odpadá tak nutnost řazení částic. Nevýhodou ovšem je, že *aditivní* blending sčítá jednotlivé barvy. To se projevuje výrazným zesvětlením částic. Editor umožňuje výběr mezi oběma variantami, přičemž částice kvůli výkonu neseřazuje.

Animace

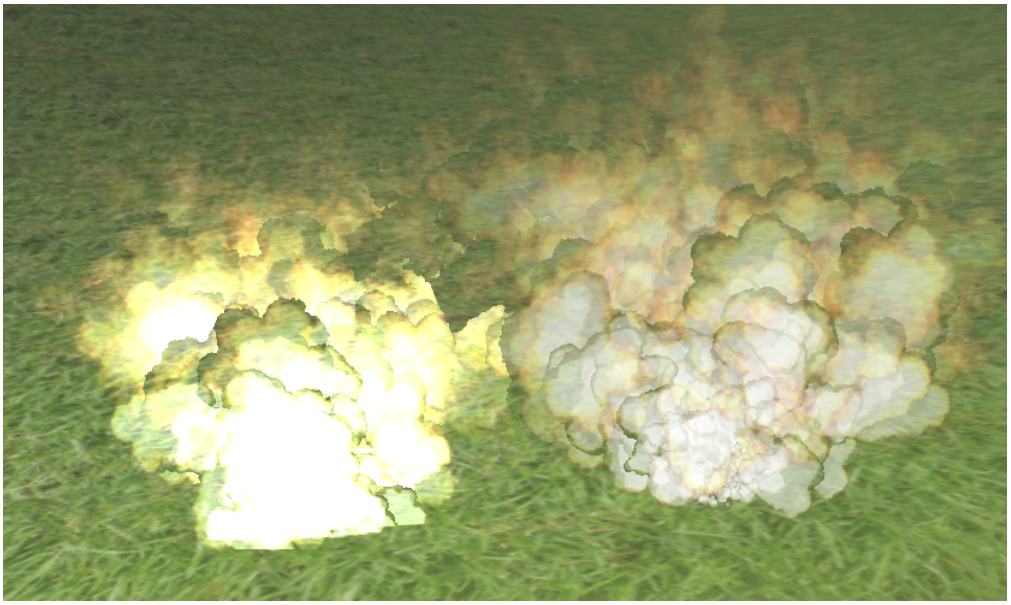
Hra podporuje animaci rotace, změny měřítka a posuvu jednotlivých modelů. Pomocí specifikace několika klíčových pozic parametrů *keyframes* a délky animace jsou modely animovány. Jednotlivé *mezisnímky* jsou interpolovány lineární interpolací (sekce 2.8) v případě posuvu, změny měřítka a sférickou interpolací pro rotační kvaternion.

Animace jsou pevně definovány pro *power-upy*. Ty neustále rotují a pokud nimi vozidlo projede, je přidána animace zmenšení modelu.

Po uplynutí malé časové prodlevy je model zvětšen do původní velikosti.

[3] Zaručuje, že překryté objekty nejsou vykresleny

[4] Česky míchání barev



Obrázek 4.10: Obě varianty částicových emitorů se zapnutým **Depth testem**

4.10 Implementace editoru

Editor úrovní umožňuje mimo jiné při přidání herní entity vybrat její kolizní reprezentaci ze dvou kolizních obálek, a to z koule, nebo z obdélníkového boxu. *Bullet* sice dokáže vygenerovat obálku na i základě modelu, nicméně pro jednoduchou detekci jsou sféra a box naprosto dostačující. Editor si lze představit jako standardní herní scénu, která pomocí *Gui* dokáže dynamicky přidávat a upravovat parametry herních entit. Při vykreslení scény editoru jsou oproti herní scéně vykresleny křivky a kolizní obálky modelů.

Při vstupu do editoru jsou do herní scény vždy vygenerovány šablonové entity (například emitor částic, reprezentující dým stoupající z výfuků). Poté po výběru typu herního módu, je možné začít pracovat na tvorbě herní mapy. Při načtení již uložené úrovně jsou automaticky zničeny všechny herní entity a nahrazeny novými entitami, specifikovanými v externím *xml* souboru. Při uložení mapy jsou reprezentace veškerých herních entit s jejich parametry zapsány do externího souboru, a to včetně šablonových entit.

Když je přidána instance herní entity v editoru stiskem tlačítka **Add instance**, nová instance automaticky zdědí všechny atributy právě vybrané entity. U tvorby nové mapy by měl jako první entita přidán terén, protože pohyb kamerou ve světě je počítán relativně k němu.

Editor umožňuje vybírat jednotlivé herní entity přesunem kurzoru na ně (resp. jejich kolizní obálky) a stisknutím tlačítka myši. Ke zjištění, zda-li kurzor překrývá nějakou herní entitu z relativního pohledu kamery, je vygenerován paprsek, procházející scénou. *Bullet* pak dokáže detekovat kolizi paprsku s kolizní obálkou modelu a určit kterou paprsek protal.

4.11 Ovládání modelů v editoru pomocí paprsku

Téma souvisí s převodem jednotlivých prostorů, vysvětlených na straně 26. Paprsek je standardní vektor, vycházející z kamery do scény. Souřadnice paprsku jsou určeny polohou

Obrázek 4.11: Ukázka změny velikosti *power-upu* při sebrání

myši, které jsou specifikovány v *normalized device space* (dále *NDS*). Oproti tomu jsou modely specifikovány ve *World Space*. Souřadnice myši je tedy nutné transformovat z *NDS* do *World Space*. Pro následující rovnice platí, že:

- $x_1 a x_2$ jsou souřadnice myši
- $y_1 a y_1$ jsou rozměry obrazu
- $y_1 a y_1$ jsou rozměry obrazu
- n je paprsek v *NDS*
- w je paprsek v *World space*

$$\vec{n} = (\frac{x_1}{y_1 - 0.5} * 2, -\frac{x_2}{y_2 - 0.5} * 2, -1, 1) \tag{4.11}$$

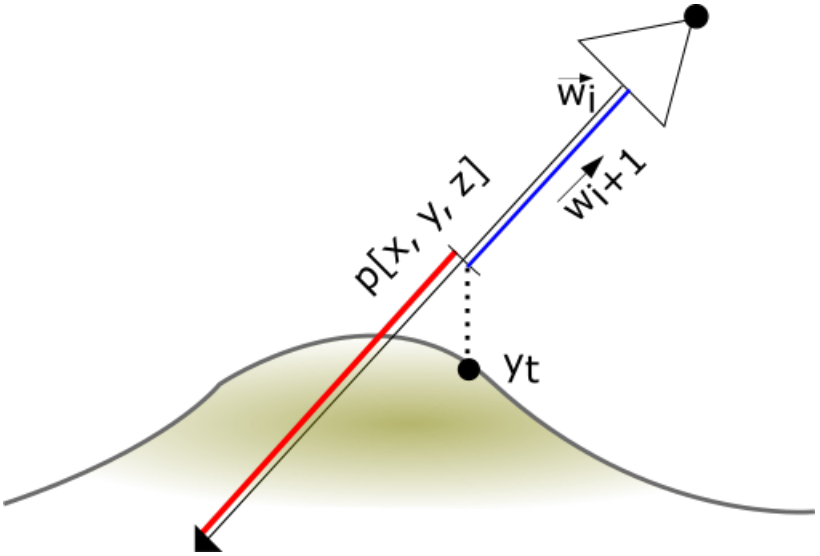
$$\vec{w} = M_V^{-1} * M_P^{-1} * \vec{n} \tag{4.12}$$

Vztahem 4.11 je vytvořen paprsek. První dvě složky vektoru $\vec{n}$ jsou souřadnice paprsku v *Normalize device coors*. Pro transformaci paprsku z *NDC* do *Clip space* není třeba provádět inverzní operaci k perspektivnímu dělení. Stačí pouze rozšířit vektor tak, aby jeho třetí souřadnice indikovala směr míření paprsku do scény.

Nyní stačí invertovat *projection* a *view* matice a vynásobit nimi $\vec{n}$ (vztah 4.12). Po normalizaci $\vec{w}$ je paprsek připraven.

Posouvání entit myší po terénu

K posuvu entit při po zisku směrového paprsku je paprsek zvětšen o dostatečně velkou konstantu k. Nyní je třeba zjistit, ve kterém místě protíná paprsek terén. K tomu je použito binární vyhledávání. Ve středu intervalu paprsku je vytvořen bod, jehož výška je porovnána s výškou terénu na jeho souřadnicích. Podmínka $y_t > p_y$, generuje nový interval paprsku $w_{i+1} \in \langle k * w_i; p \rangle$ v případě pravdy, jinak je nový interval opačný. Vyhledávání má pevně stanovený počet iterací, po kterém je výsledný bod dostatečně přesný. Pozice entity je pak nastavena na zjištěnou pozici.



Obrázek 4.12: Binární vyhledávání souřadnic terénu

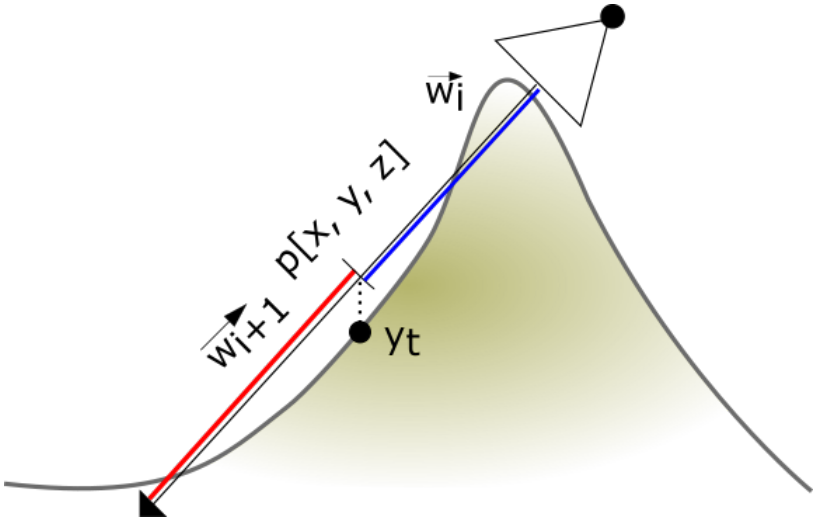
Počet iterací je pevně určen a pro zvýšení rychlosti je možné počet iterací snížit za cenu větší nepřesnosti. Algoritmus funguje dobře při terénu, který nemá příliš prudké výchylky ve výšce. Mohlo by se stát, že paprsek mine příliš vysoký kopec. Protože jde ale kameru v editoru oddalovat a přibližovat, při vyšším oddálení vetšinou není problém posouvat entity i na strmé vyvýšeniny.

4.12 Kreslení křivek v editoru

Pro kreslení křivek je nutno vytvořit nový slot. Slot reprezentuje skupinu křivek, která je přidána instancím herních entit. Mezi sloty je možné přepínat pomocí *comboboxu* v editoru po zvolení **curves**. Pro přiřazení slotu herní entitě je třeba zvolit parametr **name** tak, aby byl stejný, jako v případě herní entity reprezentující hráče(v sekci 4.4).

Ke generování křivky je použita interpolační metoda **Catmull-Rom**. Rovnice 4.13 ukazuje maticový zápis *TBC* křivky definované v rovnici 2.8.

$$I = \begin{bmatrix} t^3 \\ t^2 \\ t^1 \\ 1 \end{bmatrix} * \begin{bmatrix} -2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix} * \begin{bmatrix} P_i \\ P_{i+1} \\ l_i \\ r_i \end{bmatrix} \tag{4.13}$$



Obrázek 4.13: Chybné určení nového intervalu

T reprezentuje interval 0-1 mezi dvěma uzlovými body. Po vytvoření nové křivky a kliknutím na místo terénu je přidán nový uzlový bod. Při třech a více uzlových bodech je křivka vykreslena (pouze v editoru).

Kapitola 5

Závěr

Cílem práce bylo prostudování knihovny *OpenGL* a vytvoření hry s množstvím grafických efektů. Návrh hry byl inspirován podle existující komerční hry *Crash team racing*. Ve hře jezdí motokáry na předem vyhrazené trati. Trať je vymezena pomocí kolizní mapy. Hráči mohou sbírat zrychlující *power-upy*, které zvyšují jejich rychlost, nebo rakety, které mohou vystřelit. Při vyjetí hráče z trati, nebo při kolizi jeho vozidla a rakety, je hráč přenesen zpět na trať. Hra využívá knihovnu *Bullet* pro detekci kolizí.

Jedním z dalších úkolů bylo implementovat množství grafických efektů. Těch je dosaženo především pomocí emitorů částic a podporou více světel ve scéně. Na základě prostudování knihovny *OpenGL* a po návrhu herních mechanizmů byla vybrána verze 3.2, obsahující *Geometry Shader*, kterým jsou generovány částice na grafické kartě.

Protihráči jsou při závodech naváděni pomocí křivek, díky nimž udržují směr. Aplikace využívá křivek *Kochanek-Bartels*, s jejichž pomocí je výsledný pohyb vozidel hráčů plynulý. Navíc stačí pouze specifikovat kontrolní body, tangenty určující průběh a zakřivení křivky není třeba specifikovat, což bylo velmi výhodné při implementaci kreslení křivek.

Ve hře byl implementován i editor map. Editor umožňuje kreslení křivek, dle kterých se oponenti pohybují, importovat modely do scény, nastavovat velikost a typ jejich kolizních obálek, generovat terén pomocí výškové mapy, přidat *skybox* nebo emitor částic. Modely je možné v editoru přesouvat pomocí myši. Terén podporuje *Multitexturing*. Mapu navrženou v editoru lze bezproblémově použít ve hře, případně je možné ji i externě upravit, protože je uložena v *xml*. Všechny demonstrační mapy dostupné ve hře byly vytvořeny v editoru.

Při návrhu editoru bylo nutné prostudovat dostupné knihovny pro vytvoření *Gui*. Nakonec byla vybrána knihovna *CEGUI*.

Editor využívá pro načítání modelů knihovnu *Assimp*. Všechny modely použité ve hře byly vytvořeny modelovacím programem *Blender*.

Výsledkem práce je hra, pro kterou lze snadno přidávat další obsah v podobě nových map. Závodník je ovládán buďto klávesnicí, nebo *taneční podložkou* ***X-PAD basic***.

Možné rozšíření práce spočívá v přidání podpory skeletální animace. Skeletální animace je nyní standard pro animování modelů. Zajímavá by mohla být v kombinaci s pohybovým senzorem *Kinect*. Senzor by nahrál pohyb uživatele a podle něj by byla vytvořena kostra s animací, kterou by pak hra aplikovala na model závodníka.

Dalším rozšířením hry po grafické stránce by mohlo být například tvoření stínů pomocí stínových map. Herní editor by mohl být rozšířen o podporu materiálů a importu spekulárních a normálových textur.

Literatura

[1] Gregory, J.: *Game engine architecture*. Mass, A K Peters, 2009, ISBN 1568814135.

[2] Hocevar, S.: *Particles / Instancing*. [Online; navštíveno 15.05.2017].
URL http://www.opengl-tutorial.org/intermediate-tutorials/billboards-particles/particles-instancing

[3] Kochanek, D. H. U.; Bartels, R. H.: *Interpolating splines with local tension, continuity, and bias control*. ACM SIGGRAPH Computer Graphics[online]. 1984, **18**(3), 33-41[cit. 2017-05-15], DOI: 10.1145/964965.808575. ISSN 00978930.
URL http://dl.acm.org.ezproxy.lib.vutbr.cz/citation.cfm?doid=964965.808575

[4] Kubínek, J.: *Techniky používané pro ostření a rozmazávání obrazu*[online].Vysoké učení technické v Brně. Fakulta informačních technologií. 2007 [cit. 2017-05-15], Bakalářská práce. Vysoké učení technické v Brně. Fakulta informačních technologií. Ústav počítačové grafiky a multimédií. Vedoucí práce Jiří Venera.
URL http://hdl.handle.net/11012/56264

[5] Lawrence, J.: *Catmull-Rom Curve Fitting and Interpolation Equations*. International Journal of Mathematical Education in Science and Technology [online]. 2010, **14**(6), 842-849[cit. 2017-05-15], DOI: 10.1080/0020739X.2010.486449. ISSN 0020739X.
URL http://www.tandfonline.com.ezproxy.lib.vutbr.cz/doi/abs/10.1080/0020739X.2010.486449

[6] Wright, R.; Lipchak, B.; Haemel, N.: *OpenGL superbible: comprehensive tutorial and refence*. Upper Saddle River, N.J., 2007, ISBN 9780321498823.

[7] Žára, J.; Beneš, B.; Sochol, J.; aj.: *Moderní počítačová grafika*. Computer Press, 2004, ISBN 80-251-0454-0.

Přílohy

Příloha A

Obsah CD

- /video/xvlach14.mp4 - demonstrační video práce
- /binary - složka se spustitelnou verzí hry a dynamickými knihovnami
- /binary/game.exe - spustitelná hra
- /binary/Resources - složka s externími modely, texturami a mapami hry
- /binary/Shaders - složka se shadery
- /source_code.zip - komprimovaná složka obsahující zdrojové soubory hry, externí modely a shadery
- /latex_source.zip - komprimovaná složka obsahující zdrojový kód pro sazbu programu latex
- /xvlach14_bc.pdf - bakalářská práce ve formátu pdf
- /readme.txt - textový soubor s návodem hry