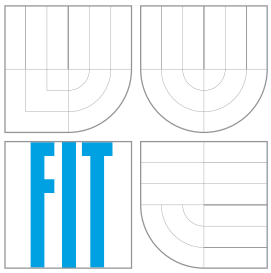
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VIZUALIZAČNÍ PROJEKT ZALOŽENÝ NA UNREAL DEVELOPMENT KITU

VISUALIZATION PROJECT BASED ON UNREAL DEVELOPMENT KIT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JIŘÍ BURY
AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN PEČIVA, Ph.D.
SUPERVISOR

BRNO 2011

Abstrakt

Tato práce se zaměřuje na tvorbu her v Unreal Development Kitu (UDK). Jsou zde probírány technologie pohánějící Unreal Engine 3 a nástroje, které UDK obsahuje. Dále se práce zabývá řešením plně dynamického denního cyklu, pokročilým zpracováním materiálů a obecně tvorbou scén určených pro herní a filmový průmysl. Podrobně je vysvětlen postup modelování objektů (v tomto případě na autě), s čímž souvisí skinning, rigging, texturování, import do UDK, přiřazení materiálů a propojení s UnrealScript kódem.

Abstract

This work focuses on game development in the Unreal Development Kit (UDK). There are discussed technologies powering the Unreal Engine 3 and tools that includes UDK. The thesis also deals with solution of a fully dynamic day cycle, advanced materials processing and generally scene creation for the game and cinematic industry. Related to skinning, rigging, texturing, importing into UDK, assigning materials and linking with the UnrealScript code, the modeling procedure of objects is explained in detail (in this case on a car).

Klíčová slova

Unreal Engine, Unreal Development Kit, UDK, herní engine, vývoj her, 3DS Max

Keywords

Unreal Engine, Unreal Development Kit, UDK, game engine, game development, 3DS Max

Citace

Jiří Bury: Vizualizační projekt založený na Unreal Development Kitu, bakalářská práce, Brno, FIT VUT v Brně, 2011

Vizualizační projekt založený na Unreal Development Kitu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. J. Pečivy, Ph.D.

.....
Jiří Bury
11. května 2011

Poděkování

Touto cestou děkuji panu Ing. J. Pečivovi, Ph.D. za nasměrování a poskytnutí cenných rad během vypracovávání bakalářské práce. Dále bych chtěl poděkovat lidem z Epic Games, Inc. za výbornou práci při vývoji Unreal Engine a UDK.

Obsah

1 Úvod **5**
- 1.1 Historie Unreal Engine . . . 5
- 1.2 Unreal Engine vs. UDK . . . 6
- 1.3 Přehled dokumentu . . . 6
- 1.4 Použitý software . . . 6

2 Unreal Engine 3 **7**
- 2.1 Objekty v Unreal Engine . . . 9
- 2.2 Osvětlení . . . 9
  - 2.2.1 Unreal Lightmass . . . 9
  - 2.2.2 Light mapy . . . 11
  - 2.2.3 Shadow mapy . . . 12
  - 2.2.4 Shadow Volumes . . . 12
  - 2.2.5 Shadow Buffer . . . 13
  - 2.2.6 Lighting Subdivisions . . . 13
  - 2.2.7 Light Environments . . . 13
  - 2.2.8 Light Functions . . . 14
- 2.3 Fyzikální systém . . . 15
  - 2.3.1 Rigid Body . . . 15
  - 2.3.2 KActor . . . 15
  - 2.3.3 KAsset . . . 16
  - 2.3.4 NVIDIA PhysX . . . 16
- 2.4 Animační systém . . . 17
  - 2.4.1 Skeletální animace . . . 17
  - 2.4.2 Morph animace . . . 17
  - 2.4.3 FaceFX . . . 17
- 2.5 Level Streaming . . . 18
- 2.6 UnrealScript . . . 18
  - 2.6.1 DLL Binding . . . 20

3 Unreal Development Kit **21**
- 3.1 UDK editor . . . 21
- 3.2 Unreal Material Editor . . . 22
- 3.3 Unreal Kismet . . . 23
- 3.4 Unreal Matinee . . . 24
- 3.5 Unreal Cascade . . . 25
- 3.6 Unreal PhAT . . . 25
- 3.7 Unreal AnimTree . . . 26

3.8 Unreal FrontEnd . . . 27
3.9 Tvorba uživatelského rozhraní . . . 27
3.9.1 UIScene . . . 28

4 Návrh a tvorba scény **29**
4.1 Koncept . . . 29
4.2 Terén . . . 29
4.3 Tunel . . . 31
4.4 Auto . . . 32
4.4.1 3DS Max . . . 33
4.4.2 UDK . . . 35
4.4.3 UnrealScript . . . 35
4.5 Objekty prostředí . . . 36
4.6 Osvětlení . . . 37
4.6.1 Obloha . . . 39
4.7 Voda . . . 40
4.7.1 Fyzikální vlastnosti . . . 41
4.7.2 Materiál . . . 42
4.8 Finální vzhled scény . . . 44

5 Závěr **47**
5.1 Využitelnost práce . . . 47
5.2 Budoucí vývoj . . . 47

A Obsah DVD **49**

B Manuál **50**

Seznam obrázků

2.1 Datový tok v Unreal Engine . . . 7
2.2 Interakce komponent Unreal Engine . . . 8
2.3 Událostmi řízená herní smyčka . . . 8
2.4 Použití světelných kanálů . . . 10
2.5 Srovnání klasického výpočtu s Global Illumination . . . 11
2.6 Srovnání kvality light map . . . 11
2.7 Shadow Volumes . . . 12
2.8 Shadow Buffer . . . 13
2.9 Light Environments . . . 14
2.10 Light Functions . . . 14
2.11 KActor s kolizním modelem . . . 15
2.12 KAsset s fyzikálními objekty interpretovanými kvádry . . . 16

3.1 Prostředí UDK . . . 21
3.2 Material Editor . . . 22
3.3 Kismet sekvence . . . 23
3.4 Unreal Matinee s animacemi řešené scény . . . 24
3.5 Model zbraně s jeho kostrou v PhAT . . . 26
3.6 Příklad AnimTree stromu . . . 27
3.7 Vztahy mezi herními prvky . . . 28

4.1 Koncept navrhované scény . . . 30
4.2 Terén s vrstvami . . . 31
4.3 Tunel s usazením do terénu . . . 32
4.4 Rozvržení referenčních obrázků v 3DS Max . . . 32
4.5 Rozpracovaný model auta . . . 33
4.6 Výsledný model auta . . . 33
4.7 UVW mapa . . . 34
4.8 Skelet auta . . . 35
4.9 Prostředí z ptačí perspektivy . . . 36
4.10 Scéna ve dne . . . 37
4.11 Scéna v noci . . . 38
4.12 Artefakty na povrchu terénu . . . 38
4.13 Sluneční paprsky . . . 39
4.14 Výsledná obloha . . . 40
4.15 Cube mapa . . . 40
4.16 Depth of Field efekt ve vodě . . . 41
4.17 Odlesk od hladiny . . . 42
4.18 Sekvence výrazových bloků tvořící odraz na vodní hladině . . . 43

4.19 Lom světla ve vodě 43
4.20 Začátek tunelu 44
4.21 Auto v tunelu při nočním osvětlení 45
4.22 Pohled z dálky 45
4.23 Osvětlení od světlometů 46
4.24 Západ slunce 46

Kapitola 1

Úvod

Vývoj v oblasti počítačových her je v dnešní době velmi rychlý a herní studia válčí o post nejlépe hodnocených her jak jen mohou. Aby zaujali potenciální zákazníky, musí předvést virtuální světy neomezené zábavy a nečekaných možností. Vytváření a následné „oživování“ těchto světů je náročná činnost, která si vyžaduje velké množství pracovní síly, času a zejména finančních prostředků (příkladem může být hra Mafia 2, jejíž vývoj trval 8 let a stál téměř miliardu korun). Vše se ale stejně děje za účelem zisku, jelikož mnohdy stojí v čele projektů investoři. Není tedy divu, že se herní studia snaží práci si co nejvíce zjednodušit a tím pádem vývoj urychlit. Za tímto účelem vznikají různé herní enginy a právě jedním z nich je i Unreal Engine, kterému se budu v této práci věnovat.

1.1 Historie Unreal Engine

V červnu roku 1998 vydala firma Epic Games svou první hru s názvem Unreal. Svým grafickým zpracováním, jedním z nejlepších outdoorových terénů a výbornou umělou inteligencí nastolila nový standard v oblasti počítačových her. Herní studia, které si licencovaly Unreal Engine, měly jasno, a sice dotvořit obsah, pozměnit chování Unreal Enginu a vydat hru. O rozdíly mezi platformami se staral Unreal Engine, takže je nebylo nutno pokaždé znovu implementovat. Velkou oblibu si získal i mezi tzv. modaři, což jsou z drtivé většiny fanoušci her, vytvářející vlastní herní obsah.

V listopadu roku 1999 vyšla multi-playerová hra Unreal Tournament, navíc s podporou hraní přes internet, což se znovu zarylo do pamětí hráčů.

Mezi březnem a červnem roku 2001 firma Epic Games zprovoznila Unreal Development Network (UDN), na které se nacházejí návody a dokumentace pro uživatele Unreal Engine, ať už jde o licencované studia, nebo modaře. Tato síť obsahuje desetitisíce dokumentů.

Velké úspěchy slavil i Unreal Engine 2, který se zlepšil už po tak výborné grafické stránce. Přínosem byla i výrazně vylepšená podpora pro modaře. Epic Games na něm postavili mnoho úspěšných her, nebudu je zde dále rozvádět.

V listopadu roku 2007 vyšla hra Unreal Tournament 3, postavená na Unreal Engine 3. Ten využíval pro zobrazení rozhraní DirectX 9.0, část osvětlení a materiálů v něm byla kompletně přepracována, obsahoval nový fyzikální systém aj. Koncem března roku 2011 vyšel Unreal Engine označený verzí 3.5, detailněji se na něho podíváme v následujících kapitolách.

1.2 Unreal Engine vs. UDK

Hlavní rozdíl mezi Unreal Engine a UDK je, že u projektu v UDK nelze přistupovat a upravovat zdrojový kód napsaný v jazyce C++, nicméně je k dispozici ovládání pomocí jazyka UnrealScript (přesto může být kód v C++ vykonán externě, je popsáno v sekci 2.6.1). Výhodu má UDK ale v tom, že pro studijní a nekomerční účely je zcela zdarma. Firma Epic Games, Inc. vychází vstříc individuálním vývojářům a poskytuje poměrně levnou licenci UDK (konkrétně $99) v případě komerčních projektů.

1.3 Přehled dokumentu

Unreal Engine je primárně určen pro tvorbu her, ale je využíván a má podporu i ve filmovém průmyslu. Cílen je na velká studia, protože není v silách jednotlivce zvládnout rozsáhlý projekt. Nicméně s dobrým nápadem mohou prorazit i tzv. „indie“ (independent), což jsou nezávislí vývojáři. Zvládnutí UDK vyžaduje opravdu velké množství času, na druhou stranu zájemcům o tvorbu virtuálních světů dává velkou zkušenost a určitě i výhodu při případném pohovoru do zaměstnání.

V Kapitole 2 probereme techniky, které využívá Unreal Engine 3. Nejedná se o úplný seznam technik, shrnuty jsou především ty, které jsou důležité z pohledu zaměření tvořené scény.

Kapitola 3 popisuje nástroje UDK, vysvětlen bude účel jednotlivých nástrojů a částečný popis práce s nimi.

Level Design je před samotnou fází tvoření obsahu (programování, modelování, ...) velmi často podceňován, nelze do scény přidávat a odebírat z ní objekty vždy jednoduše. Malé změny ve scénách nemusí způsobit žádné problémy, mohou ale také vést ke zborcení scény a částečné nebo kompletní nefunkčnosti (i té, která nejde na první pohled vidět). Na to je potřeba dávat při práci s UDK zřetel. Po návrhu následuje tvorba scény, obojím se budeme zabývat v kapitole 4. Postup je brán formou stručného návodu, jak docílit předem určených požadavků. S tím souvisí občas i nepředpokládané chování objektů ve scéně. Toto chování se vždy budeme snažit napravit tak, aby scéna působila dobrým optickým dojmem.

V závěru se pokusím objektivně zhodnotit Unreal Engine, nástroje UDK, i samotnou práci. Rovněž nastíním možnosti budoucího vývoje.

1.4 Použitý software

Nové revize UDK vycházejí pravidelně každý měsíc, stažitelné jsou ale všechny doposud vydané verze. Vývoj začal na verzi November 2010, s každou měsíční aktualizací byla scéna kompatibilní. Ve verzi UDK March 2011 byl Unreal Engine aktualizován na verzi 3.5 a u této jsem již zůstal. 100% kompatibilitu u novějších verzí nemohu zaručit.

Pro vytváření modelů jsem si vybral program 3DS Max 2011 Student. Tato verze je určena studentům pro nekomerční účely. Dodatečně byly nahrány doplňky ActorX a PhysX Plug-in.

Pro editaci textur jsem použil freeware Paint.NET. Je velmi rychlý, jednoduchý a existuje do něho velké množství užitečných doplňků.

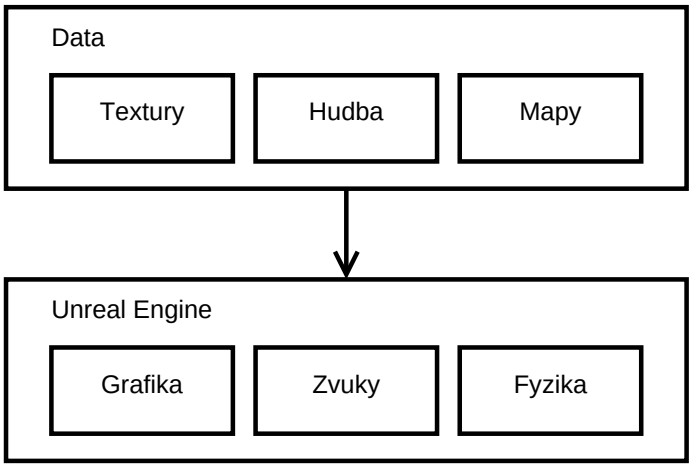
Posledním důležitým programem je freeware ConTEXT. Jedná se o textový editor, který podporuje zvýraznění syntaxe u jazyka UnrealScript.

Kapitola 2

Unreal Engine 3

Unreal Engine 3 je kompletní vývojové prostředí pro platformy PC, Xbox 360, iOS a Playstation 3, poskytující obrovské množství klíčových technologií, nástrojů pro vytváření obsahu a podporu infrastruktury.

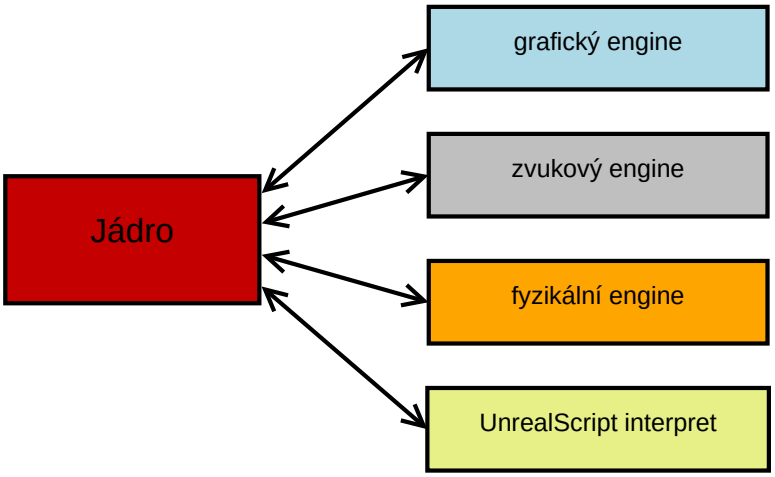
Unreal Engine 3 (dále jen UE3) byl navržen tak, aby mohli umělci a návrháři jednoduše vytvářet ve vizuálních prostředích tzv. assety (označuje se za ně obsah jako jsou textury, materiály, zvuky, charaktery, částicové efekty, ...) s pro ně relativně pochopitelnými programovacími technikami. Naproti tomu mají programátoři k dispozici vysoce modulární prostředí pro vytváření, testování a v neposlední řadě i kompletaci (překlad, komprese, vytvoření instalačního balíčku) her.



Obrázek 2.1: Datový tok v Unreal Engine

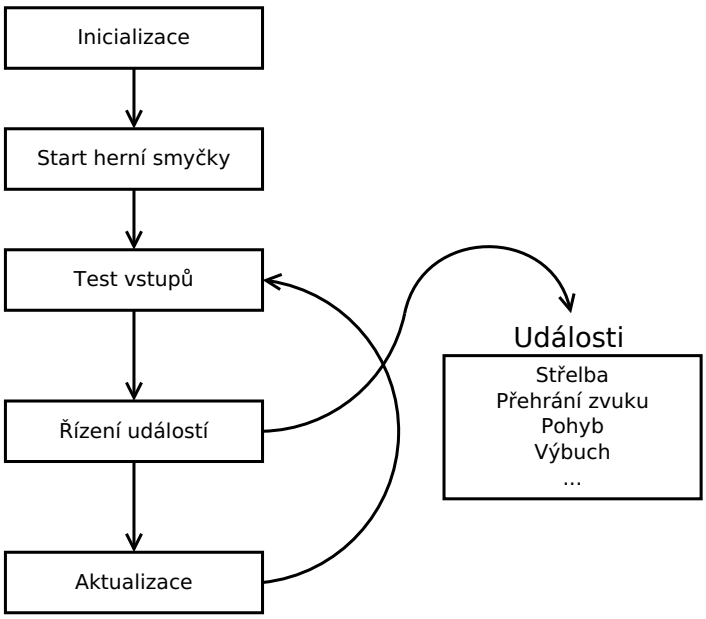
Architekturu UE3 tvoří více-vláknový renderovací systém zvaný Gemini, který se stará o výpočet globálního osvětlení metodou Photon mapping a věrohodné zobrazení scény. Umožňuje použít řadu post-processing efektů jako např. motion blur, depth of field, ambient occlusion, nebo také i nově přidaný bokeh efekt. Všechny tyto efekty obsahují mnoho parametrů. Podporovaná rozhraní jsou DirectX 9 a DirectX 11 pro platformu PC a OpenGL ES 2 pro iOS. S přínosem podpory DirectX 11 přibyla i podpora HW teselace (ovládáno přes

materiály, viz. sekce 3.2). Pro práci se zvuky využívá UE komponentu Unreal Audio System, který podporuje nejnovější kompresní formáty, pozicování zvuku v prostoru, filtraci, modulaci aj. Fyzikální systém UE je kompletně celý poháněn NVIDIA PhysX systémem [4]. Interakci komponent UE lze vidět na obrázku 2.2.



Obrázek 2.2: Interakce komponent Unreal Engine

UE je založen na událostmi řízené herní smyčce. To znamená, že obsahuje seznam událostí, které musí obsloužit. Tyto události pocházejí od komponent a jsou generovány různými zdroji, např. vstup ovládání, data z fyzikálního systému nebo i komunikace mezi samotnými komponentami [4]. Obecná událostmi řízená herní smyčka je znázorněna na obrázku 2.3.



Obrázek 2.3: Událostmi řízená herní smyčka

2.1 Objekty v Unreal Engine

Než popíšeme techniky, které UE používá, je potřeba porozumět některým pojmům. Objekty, které jsou umístěné ve scéně, lze rozdělit na statické a dynamické. Toto rozdělení není přesné, ale pro jednoduchost výkladu stačí. Statické objekty jsou všechny, které se po celou dobu ve scéně nemění. Nelze s nimi hýbat, nejde je zapínat a vypínat (světla), nemůžou se roztříštit nebo změnit kolizní model. Abychom mohli toto provádět, musíme je do scény vložit jako dynamický objekt. V případě importu polygonálního modelu z modelovacího programu do UE je tento asset (z modelu se stal zdroj v UE) označen ve specifických případech prefixem `StaticMesh`. Jedná se o statický model, který lze do scény přidat, ale interpretovat ho v ní lze mnoha různými způsoby. Dále pro jednoduchost uvažujme, že `StaticMesh` je vždy statický objekt. Druhým prefixem assetu může být ještě `SkeletalMesh`, který se liší od `StaticMesh` tím, že obsahuje kostru. Ta se využívá například k animaci modelu za běhu hry.

2.2 Osvětlení

Osvětlovací systém UE je závislý na mnoha faktorech, se kterými souvisí i HW nároky. Jedná se především o komplexnost scény a nastavení vlastností objektů v ní. Každý objekt ve scéně má přiřazené světelné kanály, které ho ovlivňují nebo naopak, které ovlivňuje on sám. Na obrázku 2.4 jsou 3 různě barevné bodové světla, každé z nich ovlivňuje jiný kanál (lze vidět na vlastnostech nad každým světlem). Dále jsou na obrázku 3 bedny, nesoucí na povrchu informace o zbarvení podle toho ke kterým kanálům jsou připojeny.

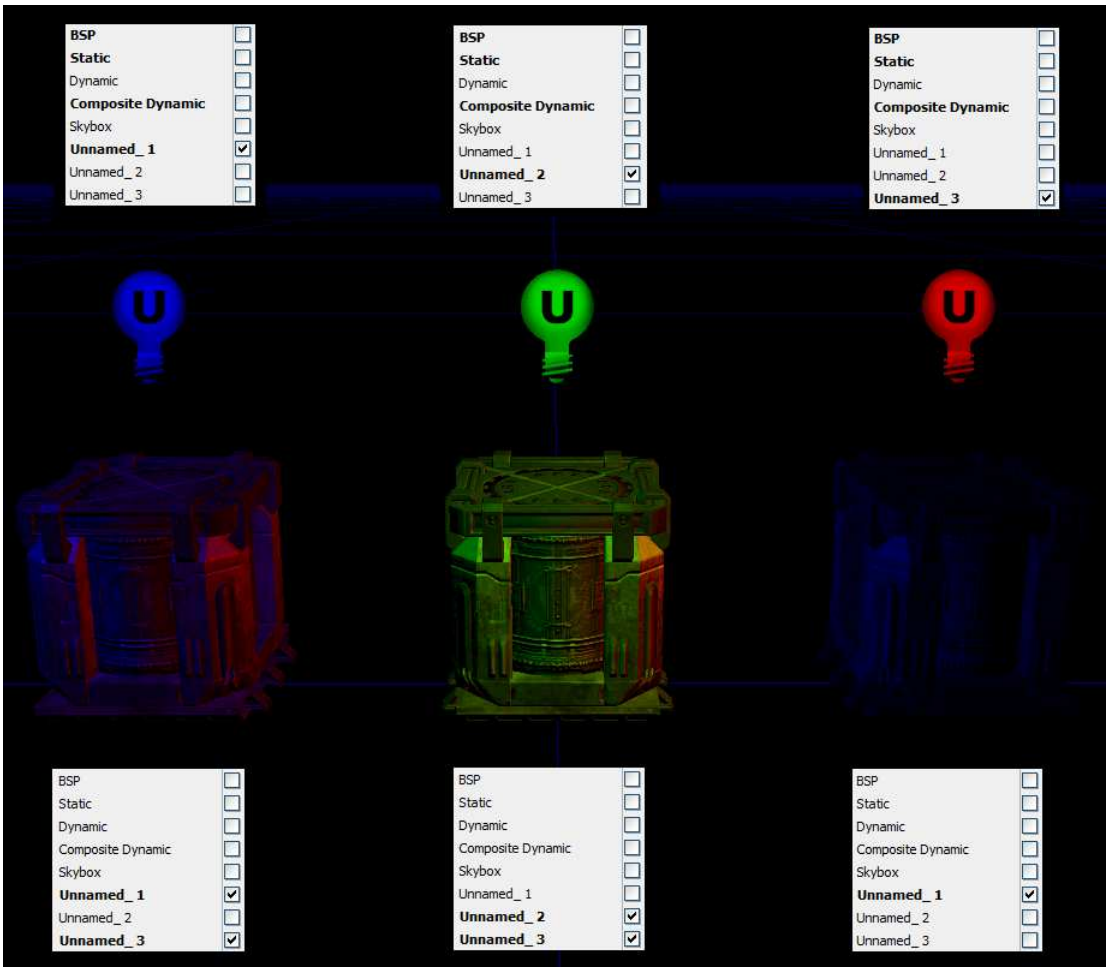
Teoreticky by měly být připojeny všechny objekty k jednomu kanálu, protože v reálném světě si objekty nemůžou vybrat jestli budou osvětleny nebo ne. Důvodem, proč tomu tak není, jsou většinou optimalizace. V některých případech mohou vznikat na objektech ve scéně nepříjemné artefakty, které lze právě pomocí těchto kanálů řešit.

Světla v UE můžeme rozdělit na statické a dynamické. Statické se oproti dynamickým nemohou nijak hýbat, nelze je vypnout a nemohou měnit ani jas ani barvu. Jakmile je scéna spuštěna, jsou proti jakýmkoli změnám uzamčeny. Dále lze rozdělit světla podle typu použití:

- `PointLight` – bodové světlo, svítí všemi směry
- `SpotLight` – reflektorové světlo, šíří se v předem zadaném kuželu
- `SkyLight` – okolní světlo, již se nepoužívá (nahrazeno Lightmass systémem)
- `DirectionalLight` – přímé světlo, určeno pro simulaci slunečních paprsků

2.2.1 Unreal Lightmass

Unreal Lightmass je systém, který řeší globální osvětlení a stínování ve scéně metodou photon mapping. Při výpočtech osvětlení se berou v úvahu i tzv. difúzní interreflexe, což znamená, že se fotony světla šíří prostorem a při odrazu od povrchu objektu mohou přebírat vlastnosti materiálů těchto objektů (změna barvy, snížení jasu), též zvané color bleeding. Výsledek těchto výpočtů se uloží do light map, které jsou využity dále jako náhrada méně kvalitních light map a statických shadow map při dynamickém osvětlení a stínování [2].



Obrázek 2.4: Použití světelných kanálů

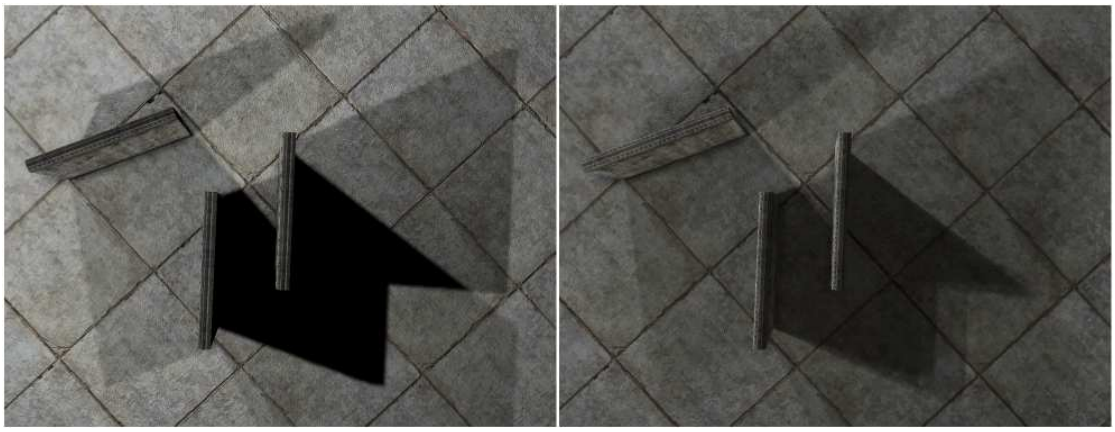
Na obrázku 2.5 jsou 2 totožné scény s malou místností, dvěma světly a třemi objekty, které vrhají stíny. U scény vlevo jdou vidět ostré stíny, u scény vpravo je zapnutý výpočet pomocí Lightmass[1].

Nejedná se o zabudovaný real-time systém, propojení UE a Lightmass zajišťuje Unreal Swarm, což je systém navržený pro masivní distribuované výpočty na renderovacích farmách. V případě komplexních scén jsou kritické sekce velikost paměti RAM a čas výpočtu.

Důležitým objektem ve scéně je `LightmassImportanceVolume`. Tento neviditelný objekt vymezuje hranice prostoru, ve kterém bude Lightmass počítat osvětlení. Příkladem může být rozdělení scény na oblasti, kde má každá vlastní `LightmassImportanceVolume` a `DirectionalLight` s odlišným parametrem `LightBounces` (počet odrazů fotonů) podle přístupnosti do dané oblasti.

Výpočet všeho osvětlení během hry by drasticky snížilo výkon, proto se používá předpočítané, které se uloží do light a shadow map a již se dále ve hře nepočítá. Platí to ovšem pouze pro statické objekty (viz. sekce 2.1).

[1] výpočet u této velmi jednoduché scény trval 40 sekund, kvůli vysokému rozlišení light mapy na zemi a nastavením osmi odrazů fotonů



Obrázek 2.5: Srovnání klasického výpočtu s Global Illumination

2.2.2 Light mapy

Light mapa je textura, vygenerovaná z výpočtů osvětlení povrchu statického objektu. Do tohoto výpočtu se zahrnují všechny světla, které osvětlují daný objekt. Výhoda spočívá v tom, že může více světel zasahovat stejnou část povrchu. Protože se jedná o předpočítané osvětlení, snížení výkonu je minimální. Výslednou kvalitu light mapy určuje rozlišení povrchu, na který se má light mapa aplikovat. Čím je nižší, tím se zlepšuje vizuální kvalita, ale za cenu vyšší paměťové náročnosti a mírného snížení výkonu [5]. Proto je důležité nalézt dobrý poměr mezi kvalitou a výkonem. Na obrázku 2.6 můžeme vidět srovnání vizuální kvality light map. Povrch ve scéně nalevo má nastavenou velikost light mapy 1.0, zatímco povrch aplikované light mapy ve scéně napravo 64.0. Na tyto light mapy připadá určitý počet lumelů, jenž je pro povrchy BSP objektů pevně daný, `StaticMesh` objekty mají pevnou velikost povrchu a specifikuje se počet lumelů light mapy, tzn. její rozlišení. Ze strany UE je toto značení poněkud matoucí.

Obrázek 2.6: Srovnání kvality light map

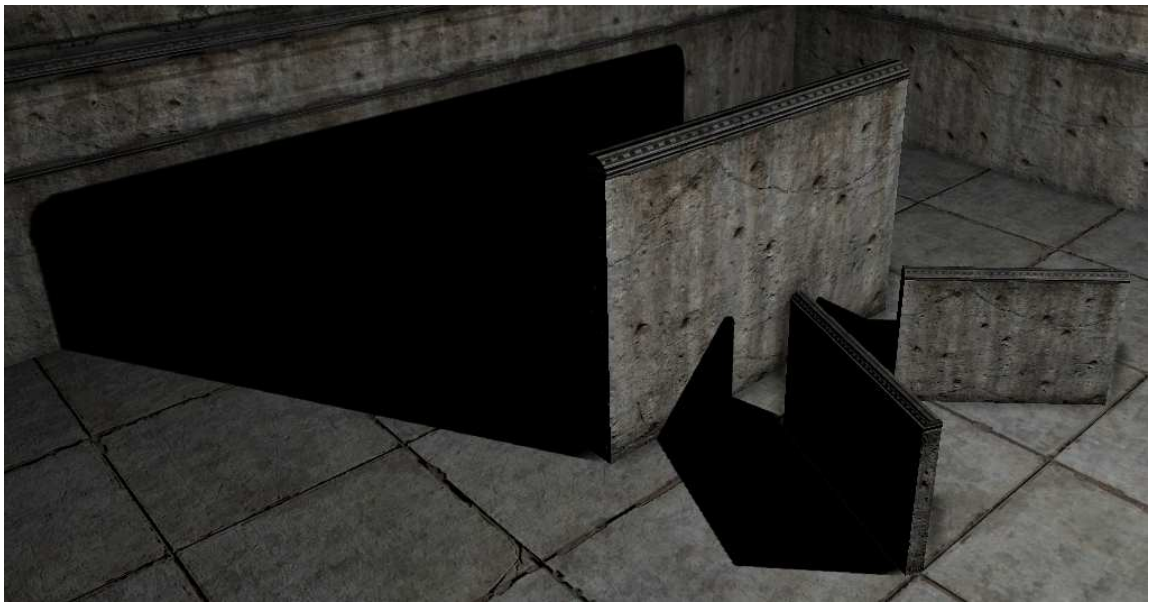
2.2.3 Shadow mapy

Shadow mapy jsou velmi podobné light mapám, ale místo osvětlení uchovávají informace o stínech. Základní rozdíl je v tom, že každé světlo ve scéně má svou vlastní shadow mapu pro každý z povrchů. V případě, že povrch objektu zasahuje více světel, jsou jejich shadow mapy zkombinovány. Shadow mapy poskytují lepší kontrolu osvětlení statických modelů než light mapy, mají ale daleko vyšší paměťovou náročnost, což se projeví i na mírném snížení výkonu [9].

2.2.4 Shadow Volumes

Předchozí vysvětlené techniky slouží pro statické osvětlení, tzn. je předpočítané, má vysokou kvalitu a během hry se nepohybuje. Jedna z technik, které UE3 používá pro dynamické osvětlení, se nazývá Shadow Volumes. Tuto techniku využívají výhradně dynamické světla, která osvětlují statické nebo dynamické objekty.

V případě dynamického světla, které osvětluje dynamický model, se při jakémkoliv pohybu světlem generují stíny pomocí Shadow Volumes i po výpočtu osvětlení pomocí Lightmass. Stíny generované od statického modelu jsou po výpočtu zobrazeny jako předpočítané stíny nebo Shadow Buffer stíny a v případě pohybu světla/objektu se změní na Shadow Volumes.



Obrázek 2.7: Shadow Volumes

Shadow Volumes neumí v UE pracovat s alfa kanálem materiálů. V případě průhledných části u materiálu přiřazeného k modelu dané části modelu vidět nepůjdou, ale budou vrhat stíny. Modely, u kterých se očekává vrhání dynamických stínů, tedy musí být vymodelovány s přesností.

2.2.5 Shadow Buffer

Technika Shadow Buffer má oproti Shadow Volumes jednu velkou nevýhodu, kvalita zobrazení stínu je závislá na vzdálenosti stínícího objektu od povrchu dopadajícího stínu. V UE se Shadow Buffer používá na stíny vržené hráčem a na tzv. Character self-shadowing, což jsou dynamické stíny na modelu hráče. Dále se využívá na zobrazení stínů dynamických objektů, které jsou osvětlovány statickým světlem [9]. Aby byly Shadow Buffer stíny korektně zobrazeny, musí mít objekt přiřazen kolizní model, viz. nástroj PhAT v sekci 3.6.

Obrázek 2.8: Shadow Buffer

Obrovskou výhodou Shadow Buffer je možnost využívat průhlednost materiálů. To se projeví zejména při použití objektů jako je například plot, kdy stačí do scény přidat jednoduchý model kostky a pletivo vyřešit alfa kanálem.

2.2.6 Lighting Subdivisions

Počínaje verzí UE3 se osvětlovací metoda změnila z per-vertex na Lightning Subdivisions. Osvětlování per-vertex značí, že se trojúhelník zobrazí osvětlený pouze pokud alespoň jeden z jeho vrcholů přijímá světlo. Problém nastával, pokud byl zastíněný pouze jeden vrchol, ve výsledném obraze často vznikaly artefakty, zvlášť pokud se jednalo o low-poly model.

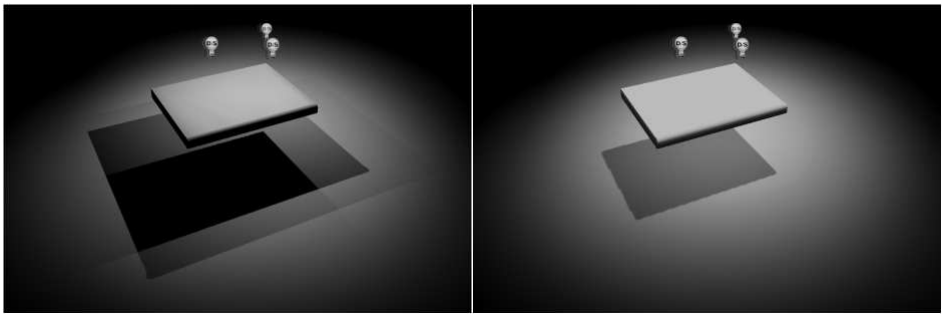
Lightning Subdivisions funguje na principu rozdělení polygonu na menší části, ve kterých se počítá osvětlení. Výsledek je smíchán dohromady a představuje osvětlení celého polygonu. Výsledné stíny jsou čistší, za cenu snížení výkonu [5].

2.2.7 Light Environments

Pokud spustíme scénu, kde je velké množství dynamických světel osvětlujících velké množství objektů, bude to mít katastrofální dopad na výkon. Proto může být objektům zapnuta možnost používání Light Environments, které povolí daným objektům vrhat pouze jeden stín. Tato technika pracuje na principu vytvoření okolního a přímého světla pro každý objekt.

Okolní světlo je využito pro osvětlení samotného objektu a přímé světlo pro vržení stínu. Barva stínu je určena jako váhový průměr barev všech světel, které osvětlují tento objekt a mají zapnutou vlastnost `AffectCompositeShadows`, přičemž váhu určuje jas světla. Směr stínu je počítán z jednotlivých vzdáleností světel od objektu, zároveň ale musí mít světlo zapnutou vlastnost `AffectCompositeShadowDirection` [1].

Výkonnostní rozdíl se v případě komplexních scén obrovský. Princip Light Environments můžeme vidět na obrázku 2.9, scény jsou zobrazeny v módu detailu osvětlení. Vlevo je objekt, který vrhá celkem 3 stíny, každý patří ke svému světlu. Scéna vpravo obsahuje již jen jeden stín, který využívá Light Environments a je společný pro všechny světla.



Obrázek 2.9: Light Environments

2.2.8 Light Functions

Light Functions poskytují možnost ovládat světla pomocí materiálů, což lze využít pro mnoho účelů, např. barevné skla. K tomu se využívá emisivní složka materiálů (viz. sekce 3.2). Směr projekce záleží na použitém typu světla (bodové, přímé, reflektorové). V případě použití Light Functions nemůže být světlo použito pro předpočítání stínů [4]. Na obrázku 2.10 je reflektorové světlo ovládané právě pomocí Light Functions a jednoduché textury.

Obrázek 2.10: Light Functions

2.3 Fyzikální systém

Fyzikální systém UE3 je postaven na PhysX enginu. Provádí všechny fyzikální simulace, včetně pohybu objektů a jejich kolizí. Většinou jsou zpracovávány tzv. rigid bodies, což jsou z pohledu fyziky ideální tuhé tělesa. UE3 rozeznává celkem 4 typy fyzikálních objektů:

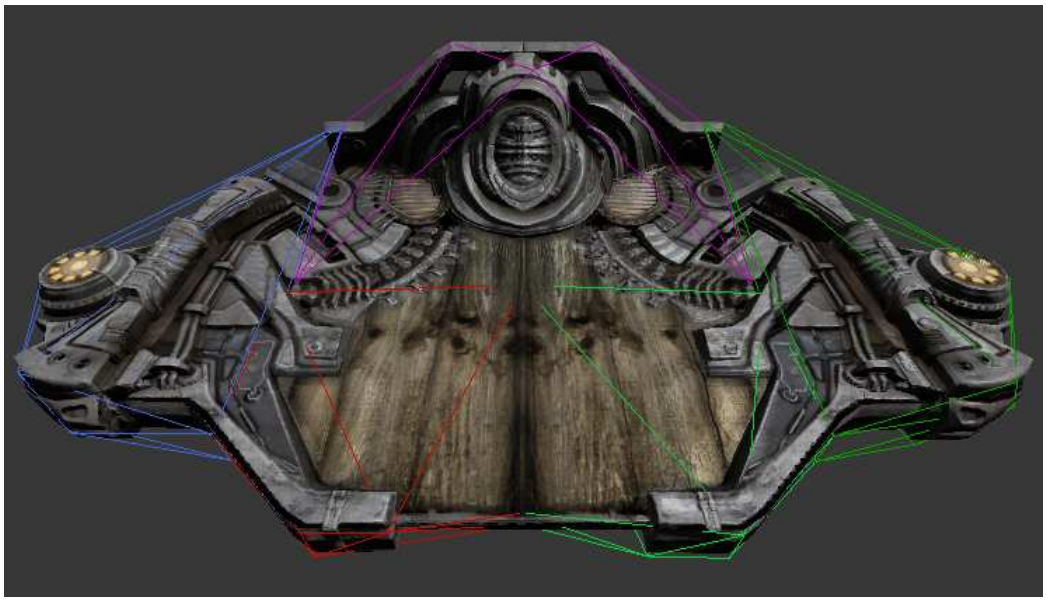
- `KActor`
- `KAsset`
- `Constraints`
- Impulzy a síly

2.3.1 Rigid Body

Rigid body je objekt, který se chová jako ideální tuhé těleso. Při působení vnější síly na povrch nedeformuje svůj tvar. Běžně jsou tyto objekty využívány ve fyzikálních simulacích kvůli relativně nízké výpočetní náročnosti. I když jejich popis nereprezentuje přesně chování jejich reálných vzorů, velmi se přibližují realistickému chování [5]. Rigid bodies také představují primární metodu kalkulace fyzikálních objektů.

2.3.2 KActor

Nejjednodušší z fyzikálních objektů, jako základ pro kalkulace je použit `StaticMesh`. Pro určení polohy, natočení, lineární a úhlové rychlosti používá typ fyziky `PHYS_RigidBody` [5]. Aby mohl být `StaticMesh` použit jako `KActor`, musí mít přiřazen zjednodušený kolizní model. Způsob vytvoření kolizního modelu v UDK a 3DS Maxu je různý, viz. sekce 3.1 (UDK) a 4.4.1 (3DS Max).

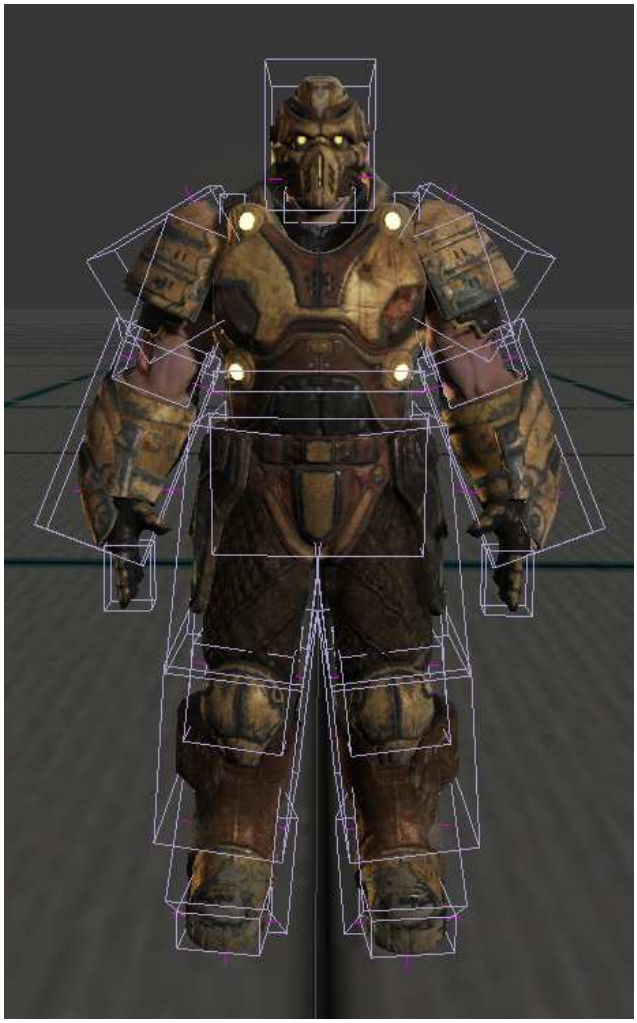


Obrázek 2.11: KActor s kolizním modelem

2.3.3 KAsset

`KAsset` je systém, ve kterém je `SkeletalMesh` animován pomocí rigid body simulací. Každá kost jeho skeletu je přiřazena určité části neviditelného fyzikálního těla, řekněme tedy samostatnému fyzikálnímu objektu. Tyto objekty mohou být interpretovány krychlí/kvádrem, kapslí („`Sphyl`“) nebo koulí a mají mezi sebou vazby typu zavěšení, kloub nebo kladka [5].

Poměr velikostí `KAsset`u ve scéně musí být vždy ve všech osách stejný, jinak může docházet k chybám při animaci.



Obrázek 2.12: KAsset s fyzikálními objekty interpretovanými kvádry

2.3.4 NVIDIA PhysX

PhysX Engine je plně integrovaný do UE3. Řeší kolize, spojování objektů, tření, nepřetržité detekce kolizí u zvolených objektů (používá se pro velmi rychle pohybující se objekty, např. kulky) a poskytuje plnou kontrolu vztahů mezi objekty (lze ušetřit výkon na objektech, kde není nutno řešit kolize) [6].

Velmi propracovanou částí je v UE3 simulace látek. Speciálním případem je mód, kdy můžou být látky použity pro simulaci plastických deformací.

Dále PhysX podporuje simulaci kapalin, plynů, částicových efektů, emitorů, měkkých tkání (včetně trhání) a vnějších sil jako je vítr, gravitace aj.

Zvláštní kategorií u PhysX je simulace vozidel, která je v UE3 propracovaná a počítá s komplexní geometrií řízení a tlumení. Vozidlo je řešeno jako jednoduché rigid body se speciálními zemními kontakty, které simulují chování kol. Třecí model pneumatik je také přítomen a řešen je pomocí fyzikálních materiálů.

2.4 Animační systém

Deformace objektů ve scéně můžou být ovlivněny dvěma zdroji, a sice animacemi pomocí klíčových snímků, nebo mísením těchto animací. Animace, které ovlivňují navzájem jiné kosti skeletu, mohou být spolu bez problému přehrávány. V případě, že více animací ovlivňuje jednu kost, výsledná animace se rovná kombinaci všech účastněných s ohledem na aktuální stav nasměrování objektu, rychlosti pohybu a jiných faktorů.

2.4.1 Skeletální animace

Základem skeletální animace je kostra, která je přiřazena k modelu. Ke každé kosti kostry jsou přiřazeny vrcholy modelu s určitou váhou v rozsahu 0.0 – 1.0. Tato váha vyjadřuje procentuální ovlivnění vrcholů při pohybu kosti.

Jakýkoliv model spojený s kostrou bere UE jako `SkeletalMesh`. Důvodem existence skeletální animace je urychlení animačního procesu, animace samotných vrcholů by byla složitá a časově náročná. Lehce jde vytvořit skok, běh, gestikulace rukou aj. UE3 rozlišuje objekty od jejich animací z důvodu možných pozdějších reprodukcí stejných animací na jiných objektech.

2.4.2 Morph animace

Morph animace odpovídá deformaci mezi dvěma modely. Vzniká interpolací vrcholů prvního modelu cílenou stavem vrcholů druhého modelu [3]. Morph animace vyžaduje minimálně dva modely, jeden zdrojový a druhý cílený. Cílené modely se pro každý zdrojový model v UE3 shlukují do kolekce `MorphTargetSet` z důvodu lepší organizace.

Nevýhoda této animace je, že musí být vrcholy modelů upravovány ručně. Počet vrcholů u všech cílených modelů musí souhlasit s počtem vrcholů zdrojového modelu, jinak nebude animace fungovat správně.

2.4.3 FaceFX

FaceFX je systém určený pro animaci obličeje. Je plně integrován do UE3 a umožňuje efektivnější a daleko rychlejší tvorbu animací. FaceFX umožňuje ovládání parametrů materiálů, což se může hodit pokud chceme udělat např. zčervenání. Zavedená je také podpora pro `MorphTargetSet`.

Základní prvek tvoří `FaceFXAsset`, který uchovává všechny data potřebné pro přehrávání animací založených na zvukových souborech. Pár základních póz musí být vytvořeno a naimportováno do FaceFX, kde už pak můžou být tvořeny vazby mezi nimi [5].

2.5 Level Streaming

Velmi důležitou technikou, kterou disponuje UE3, je tzv. level streaming. Tato technika umožňuje za běhu načítat data do paměti a naopak je i uvolňovat, lze tedy vytvářet obrovská herní prostředí.

Výhoda level streamingu je ta, že umožňuje mít plnou kontrolu nad obsahem paměti a zobrazení ve scéně. Další nespornou výhodou je dekompozice scény na levely, tzn. části, které jsou umístěny v samostatných souborech. Vývojáři mají možnost si levely mezi sebou rozdělit a pracovat na nich zároveň. Ve výsledku mohou (nemusí) být propojeny do jedné scény.

Metody streamingu existují dvě, buď pomocí vzdálenosti hráče/kamery od hranic levelu (využity jsou objekty `LevelStreamingVolumes`), nebo manuálním zpracováním v Kismetu (co je Kismet viz. sekce 3.3). Ve scéně pak je streaming prováděn např. při otevírání dveří, použití tlačítka atd. Kombinace obou metod jsou přípustné [4].

Obsah scény se dělí na levely perzistentní a streamované. Perzistentní level je přítomný ve scéně vždy, většinou bývá nahrán na začátku hry a obsahuje pouze objekt `PlayerStart` umístěný v prostoru tak, aby se nacházel na správném místě při nahrání prvního streamovaného levelu.

2.6 UnrealScript

UnrealScript je vyšší programovací jazyk, uzpůsobený pro potřeby vývoje her. Jeho hlavním zaměřením je podpora konceptů času, stavů, vlastností a vazeb. S jazykem C++ je spojena řada komplikací v případě programování herní logiky, na vytvoření je potřeba spousta času a často to vede k nepřehlednému kódu, který se špatně udržuje a ladí.

UnrealScript vychází z jazyka Java, z něhož má přejato sledování odkazů s Garbage Collectorem a jednoduché dědění tříd. Syntaxe je podobná jazykům C, C++ a Java. Kód, podílející se z velké části na výkonu UE, je napsán v C/C++, zbytek v UnrealScriptu. To umožňuje z pohledu vývojáře využívajícího UDK zásahy i do základnějších částí UE (např. správa objektů, level streaming, ...) [8].

Klíčové elementy UnrealScriptu:

- Každá třída dědí od jedné rodičovské třídy
- Každá třída patří k nějaké kolekci objektů, se kterýma je distribuována
- Funkce a proměnné třídy jsou přístupné pouze objektům, které patří k této třídě
- Globální funkce a proměnné neexistují
- Podporovány jsou všechny standardní klíčové slova jazyků C a Java
- Závorky a středníky mají stejné použití jako v C
- Klíčové slovo „`state`“, které lze využít pro oddělení kódu patřícího k určitému stavu objektu.

```
1  class TriggerLight expands Light;
2
3  // deklarace parametrů světla a triggeru
4  var ELightType InitialType;
5  var float InitialBrightness;
6  var float Alpha, Direction;
7  var actor Trigger;
8
9  // funkce volaná na začátku
10 function BeginPlay()
11 {
12     Disable('Tick');
13     InitialType = LT_Steady;
14     InitialBrightness = 1.0;
15     Alpha = 1.0;
16     Direction = 1.0;
17 }
18
19 // základní funkce volaná po každém vykreslení
20 function Tick(float DeltaTime)
21 {
22     Alpha += Direction * DeltaTime;
23     if(Alpha > 1.0)
24     {
25         Alpha = 1.0;
26         Disable('Tick');
27         if(Trigger != None) Trigger.ResetTrigger();
28     }
29     else if(Alpha < 0.0)
30     {
31         Alpha = 0.0;
32         Disable('Tick');
33         if(Trigger != None) Trigger.ResetTrigger();
34     }
35     LightBrightness = Alpha * InitialBrightness;
36 }
37
38 // funkce obsluhující stav přepnutí stavu triggeru
39 state() TriggerToggle
40 {
41     function Trigger(actor Other, pawn EventInstigator)
42     {
43         Trigger = Other;
44         Direction *= -1;
45         Enable('Tick');
46     }
47 }
```

2.6.1 DLL Binding

Možnost přistupovat do zdrojového kódu a přepisovat ho mají pouze licencovaní uživatelé plné verze UE3. V UDK toto není možné, nicméně kód C++ lze spouštět pomocí direktivy `DLLBind`. Jedna třída může na sebe navázat pouze jednu dynamickou DLL knihovnu a musí být koncová, tzn. nemůže už být dále děděna [7].

Následující kus kódu je jednoduchý UnrealScript soubor, který na sebe váže knihovnu `%UE3%\Binaries\Win32\UserCode\TestDLL.dll`. Jakmile je knihovna navázána, mohou být importované funkce klasicky volány v UnrealScript kódu.

```
1 class TestDLLPlayerController extends PlayerController
2     DLLBind(TestDLL);
3
4 dllimport final function vector CallDLL(string s, out int i[2]);
```

Další kus kódu představuje implementaci dynamické knihovny v C++, která se vztahuje i k výše uvedenému kódu.

```
1  extern "C"
2  {
3      struct FVector
4      {
5          float x, y, z;
6      };
7
8      __declspec(dllexport) FVector* CallDLL2(wchar_t* s, int i[2])
9      {
10         wchar_t temp[1024];
11         static FVector result;
12         result.x = (float) i[0];
13         result.y = -1.0;
14         result.z = (float) i[1];
15         swprintf_s(temp, 1024, L"i: {\%d,\%d}", i[0], i[1]);
16         return &result;
17     }
18 }
```

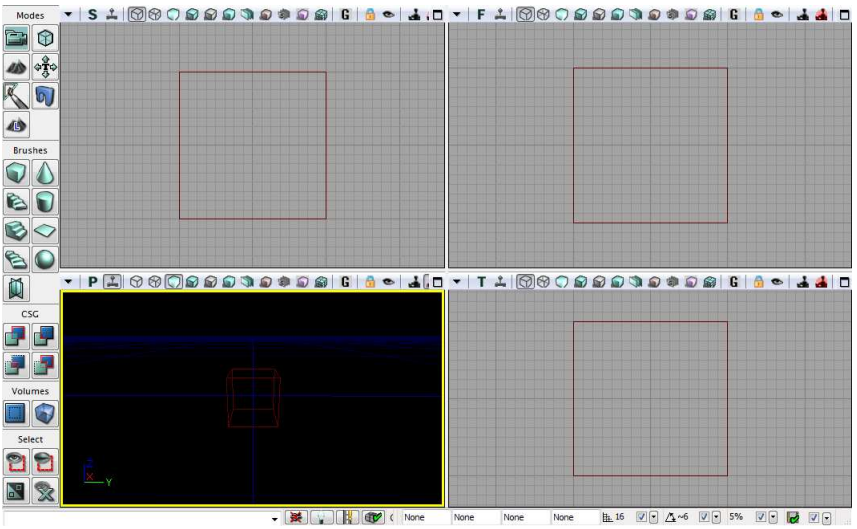
Kapitola 3

Unreal Development Kit

Unreal Development Kit (UDK) je, vyjma pár modifikací řečených dříve, synonymem pro Unreal Engine 3, ale zároveň i název samotného editoru, v němž se tvoří scéna. V této kapitole se seznámíme s většinou nástrojů, které obsahuje. Vývoj v UDK je možný pro platformu PC a iOS, což je segment mobilních aplikací pro zařízení značky Apple. Rozdíly mezi výkonem HW u těchto dvou platforem jsou velké, proto UDK obsahuje 2 verze editoru a spustitelné hry. K dispozici jsou dále programy SpeedTree Modeler, Unreal FrontEnd (je popsán v sekci 3.8) a Unreal iOS Configuration.

3.1 UDK editor

Prostředí editoru je podobné jako v 3D modelovacích programech, hlavní část tvoří 1 perspektivní a 3 ortogonální pohledy na scénu. Pro každý pohled má uživatel možnost zvolit si možnosti zobrazení scény, např. pomocí polygonálních sítí, detailu osvětlení, k dispozici je i možnost zobrazení jen určitých typů objektů. Užitečnými jsou módy Real-Time a Game View. První z nich se hodí při použití materiálů měnících se v čase (v základním nastavení je tato možnost vypnuta a scéna se kvůli výkonu aktualizuje pouze při pohybu kamerou). Druhý mód zobrazí scénu v pohledu tak, jak bude vypadat ve hře.



Obrázek 3.1: Prostředí UDK

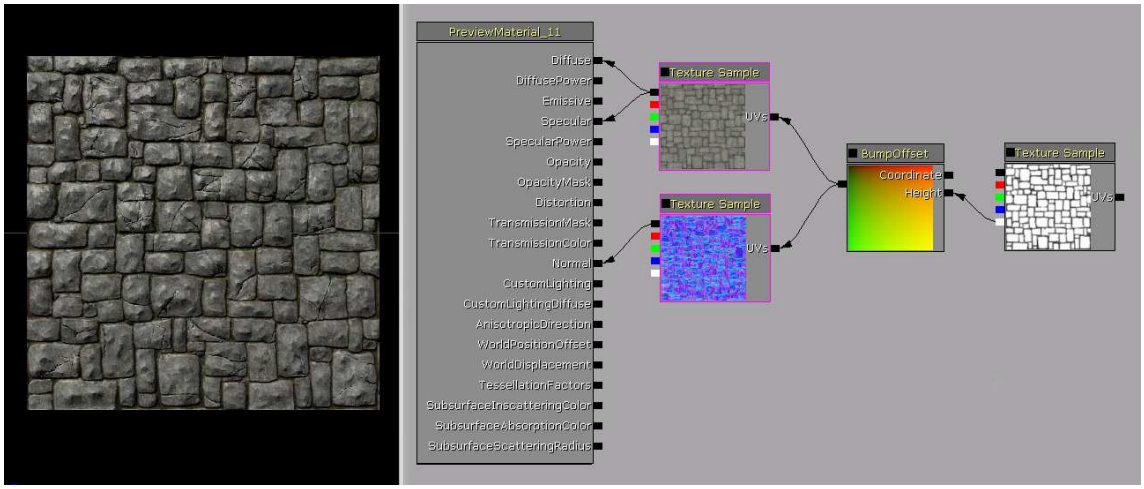
Při práci v editoru je aktivní vždy jen jeden editační mód. Uživatel má na výběr z těchto sedmi editačních módů:

- Camera Mode – volná manipulace s objekty ve scéně
- Geometry Mode – úprava geometrie BSP a CSG
- Terrain Editing Mode – úprava terénu, doporučuji nepoužívat
- Texture Alignment Mode – úpravy textur na povrchu (posun, měřítko, ...)
- Mesh Paint Mode – kreslení na modely
- Static Mesh Mode – úprava speciálních vlastností statických objektů
- Landscape Mode – nový mód pro práci s terénem

Pro urychlení tvorby CSG modelů pomocí BSP štětců poskytuje editor 9 přednastavených tvarů jako je koule, krychle, jehlan, ... U všech jsou nastavitelné základní parametry tvarů, složitější úpravy se řeší v Geometry Mode. Pomocí BSP štětců se přidávají do scény i tzv. Volumes, což jsou různé objekty představující např. ohraničení pro výpočet světla, prostor používající post-processing efekty atd. Dále obsahuje editor také funkce pro viditelnost objektů a navigaci ve scéně.

3.2 Unreal Material Editor

Material Editor slouží k vytváření a úpravám materiálů. Materiál je jediná možná cesta, jak přenést barevnou informaci na povrch modelu. Využívá různých elementů, pomocí kterých se dá ovládat např. průhlednost, lom světla, tesselace modelu a mnoho dalších. Mezi tyto elementy patří i textury, které nemohou být aplikovány na povrch objektu přímo. Základ materiálu tvoří `Material Node`, který obsahuje mnoho různých kanálů, např. `Diffuse`, `Emissive`, `Specular`. Na obrázku 3.2 je `Material Node` pojmenován jako `PreviewMaterial_11`. Jednotlivé kanály nebudu dále rozebírat, z názvů lze odvodit jejich použití.



Obrázek 3.2: Material Editor

V náhledu je možno zapnout zobrazení shader kódu v jazyce HLSL. Při konstruování materiálů je důležité sledovat počet instrukcí, které musí vykonat grafický procesor. Velký počet objektů se složitým materiálem může vést k výkonnostním problémům. Na druhou stranu ale lze udělat velmi komplexní materiály, které spotřebovávají minimum instrukcí. Vše je závislé na použití materiálových výrazů. Na předchozím obrázku jsou materiálové výrazy celkem čtyři, tři `TextureSample` a jeden `BumpOffset`. Výsledkem propojení těchto výrazů a `Material Node` kanálů je kamenná zeď nebo podlaha se slabým odleskem v levé horní části (světlo se tedy nachází také někde v těch místech) a velice pěkným 3D efektem na jednotlivých kamenech.

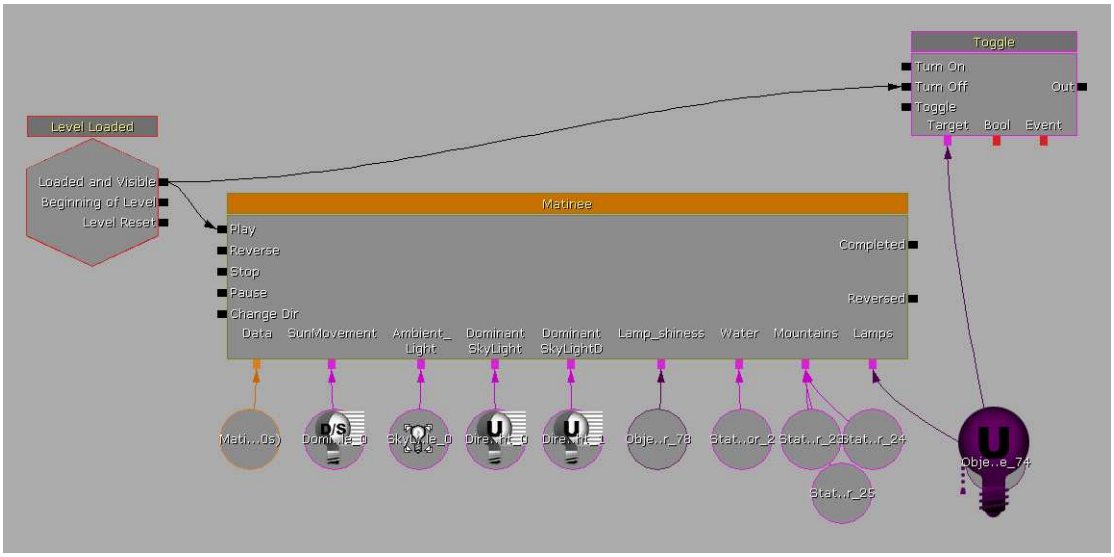
Důležitou vlastností materiálů je, že z nich mohou být odvozeny instance. V instancích pak lze měnit parametry materiálových výrazů, které byly v otcovském materiálu zkonvertovány na `Param`.

3.3 Unreal Kismet

Unreal Kismet je vizuální skriptovací systém, se kterým lze s minimálními programovacími znalostmi rychle a jednoduše vytvářet komplexní sekvence modulů [4]. Moduly jsou navzájem propojeny „dráty“, které mají za úkol přenášet informace.

Moduly Kismetu se dělí do kategorií:

- Příkazy – např. změna kolizního modelu u objektu
- Podmínky – porovnání hodnot, test stavu boje, ...
- Proměnné – objekty, Matinee data, jmenné proměnné, ...
- Události – aktivace triggeru, nahrání levelu, objekt zničen, ...



Obrázek 3.3: Kismet sekvence

Speciálním modulem je Matinee, což je modul obsluhující animace v levelu. Nejedná se o nastavení animací modelů, o to se stará nástroj AnimTree (popsán v sekci 3.7). Matinee modul se uvnitř dělí na skupiny a každé skupině jsou přiřazeny objekty, viz. obrázek 3.3.

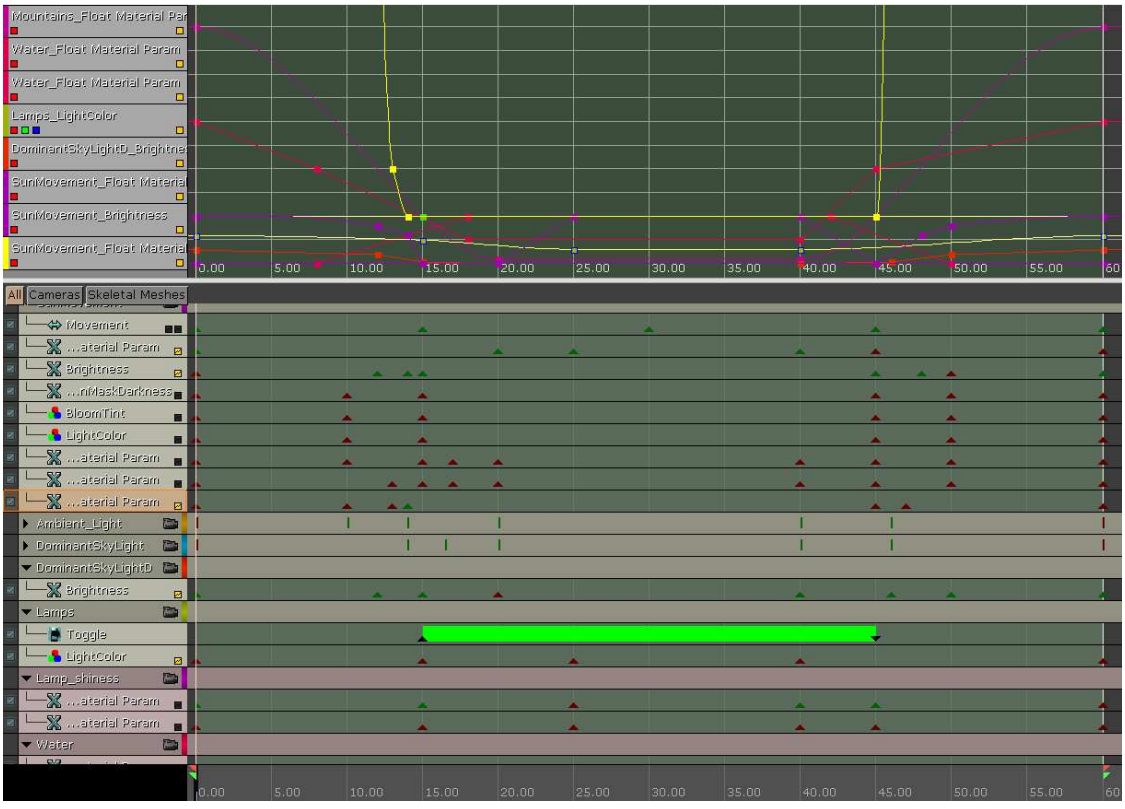
3.4 Unreal Matinee

Unreal Matinee je systém pro práci s animacemi. Ovládány mohou být parametry materiálových instancí, parametry světel, vytvářet se dají se scénou machinima, což už se řadí spíše do filmového průmyslu (hojně se využívají ale i ve hrách pro doplnění příběhové linie). Aktivovat se dají i samotné animace `SkeletalMesh`, FaceFX obličejový morphing, zvuky a plno dalších.

Součástí Matinee jsou:

- Stopy – každá stopa má na starost jednu vlastnost jedné skupiny
- Skupiny – mohou být do nich seskupeny stopy, patřící k jednomu objektu
- Křivky – vyznačují průběh hodnot animované vlastnosti
- Klíče – jsou využity k nastavení hodnoty ve zvoleném čase
- Tangenty – určují tvar křivky v klíčích

Speciální částí Matinee je režisérská skupina. Pro každou Matinee sekvenci, vytvořenou v Kismetu, může být přítomná maximálně jedna. Lze pomocí ní přidávat zvláštní stopy, které umí zpracovávat efekty jako střih kamer, blednutí (fade), slow-motion a jiné velmi specifické funkce.



Obrázek 3.4: Unreal Matinee s animacemi řešené scény

Na obrázku 3.4 můžeme vidět hlavní okno Matinee, které je rozděleno na 4 části. V levém spodním panelu se nacházejí skupiny (mají nalevo od názvu rozbalovací trojúhelník), pod které spadají jejich stopy. Je tam např. skupina `Lamps` se stopou `Toggle`, což způsobuje ve scéně zapnutí/vypnutí světel u pouličních lamp. Vedle tohoto panelu se nachází časová osa od 0 do 60 sekund. Na ní se nacházejí trojúhelníky, které představují klíče. Vertikálně je časová osa rozdělena podle stop nalevo. Klíče v jednom řádku patří ke stopě na stejné vertikální pozici. V levé horní části jsou zobrazeny všechny stopy, které mají nastavenou aktivní viditelnost. Pravá horní část představuje editor křivek, čtverečky na křivce označují hodnoty klíčů. Tento obrázek působí možná zmateně, ale dokazuje, že Matinee zvládá opravdu velké množství animací a je stále přehledný.

3.5 Unreal Cascade

Unreal Cascade je nástroj na vytváření částicových efektů. K tomu využívá emitory, což jsou body v prostoru, kde se částice generují a poté jsou vystřelovány dále do prostoru. Kolekce jednoho nebo více emitorů, které splňují určité pravidla, se nazývá částicový systém. Do scény je pak tento sytém přidán přes objekt `Emitter`. Všechny emitory jsou označovány jako „`Sprite`“. `Sprite` je čtvercová/obdélníková rovina, která je vždy natočena ke kameře a má přiřazený materiál [5].

Částicový systém obsahuje `TypeData` moduly, ty tvoří základ každého emitoru a dělí se na:

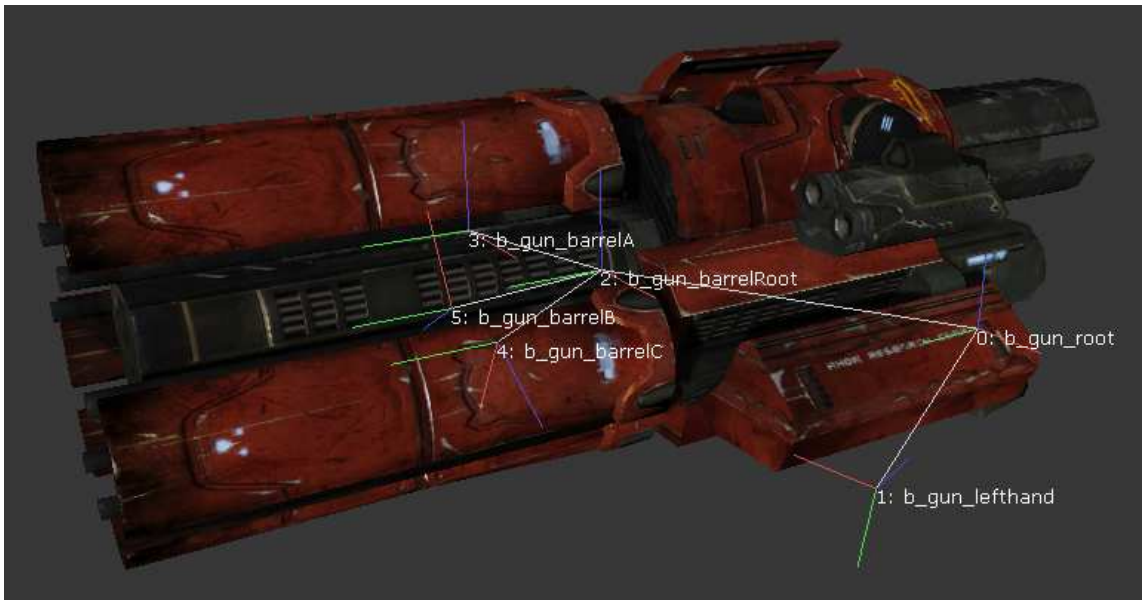
- `Beam` – částice se generují mezi 2 body a jsou v řadě na sebe napojené
- `Mesh` – místo částic se generují `StaticMesh` objekty
- `Trail` – částice jsou na sebe napojené a při pohybu objektu tvoří jeho stopu
- `Fluid` – simulace kapalin pomocí PhysX Fluid Solver

3.6 Unreal PhAT

Unreal PhAT je nástroj pro práci s fyzikou u `SkeletalMesh`, které pak mohou být do scény přidány jako `KAsset`. Odpadá tím časově náročná tvorba, přenášení a testování v 3D modelovacích programech. Primární úloha PhAT spočívá ve vylepšování a testování fyzikálního chování modelu. Pro správnou funkčnost musí být základní kosti skeletu přiřazeny nějaké vrcholy modelu s určitou váhou. V opačném případě PhAT nerozezná žádné navazující kosti a musí být vytvořeny manuálně.

Fyzika, která je použita u `SkeletalMesh`, je uložena do Physics Assetu. Pro případ jiného různého chování `KAsset` objektů, které používají stejný `SkeletalMesh`, stačí přiřadit těmto objektům různé Physics Assety.

U kostí se dají ovládat parametry jako maximální úhel otočení vůči ostatním připojeným kostím, maximální napětí, fyzikální materiál aj. Nástroj PhAT umožňuje také zobrazení kostry a pozicování kostí. Jednoduchý model s kostrou je zobrazen na obrázku 3.5.



Obrázek 3.5: Model zbraně s jeho kostrou v PhAT

3.7 Unreal AnimTree

Unreal AnimTree je vizuální editor určený pro tvorbu animačních sítí `SkeletalMesh` objektů. Svou funkčností je velmi blízký Kismetu nebo editoru materiálů.

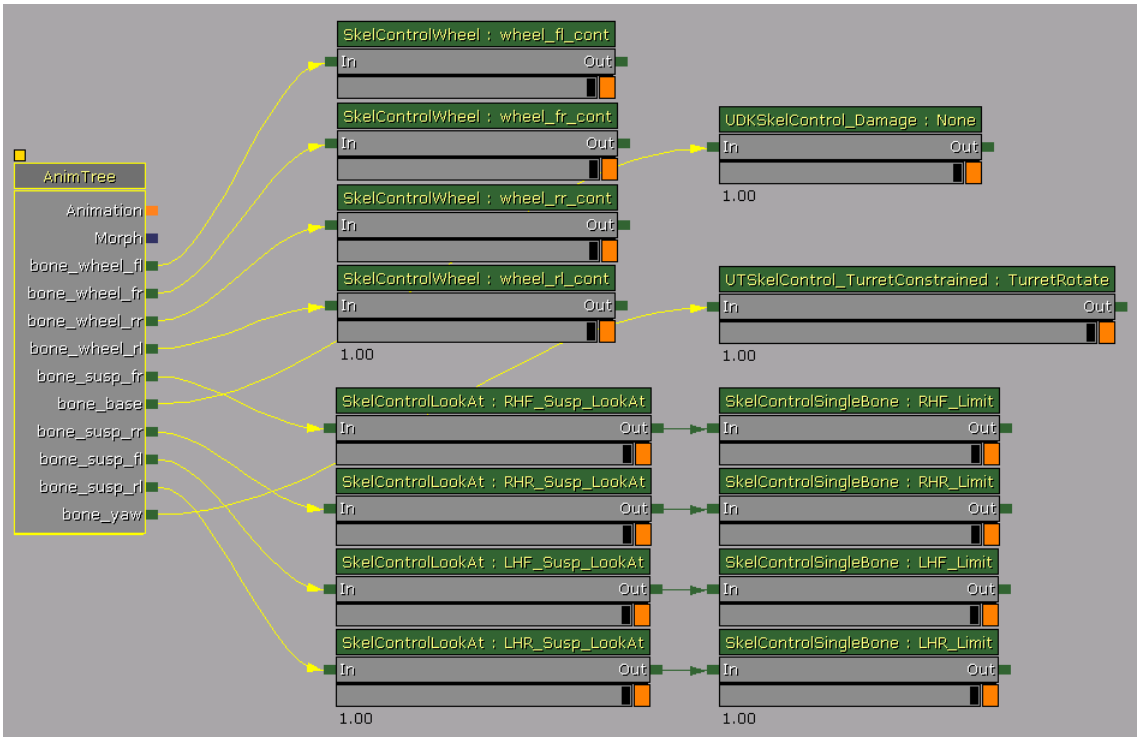
Podobně jako PhAT jsou data uložena v AnimTree assetech. Základem je `AnimTree Node`, ze kterého se hierarchicky odvozují synovské uzly, patřící k určitému objektu a vztahující se k určitému typu ovládání.

Přes AnimTree uzly mohou být ovládány:

- `Morph` animace
- `SkeletalMesh` animace pomocí klíčových snímků
- `SkelControls` – dovolují přepsat animaci na určité kosti jinou

Implicitně obsahuje `AnimTree Node` pouze uzly `Animation` a `Morph`, pomocí kterých se dají aktivovat `MorphTargetSet` animace přiřazené k modelu. Ačkoli AnimTree vypadá jednoduše, je určen spíše pro programátory vzhledem k obsáhlým parametrům uzlů a většinou i nutnému propojení s kódem. Editor podporuje zobrazení skeletu, jména kostí, váhu uzlů v části s grafem a real-time náhled modelu.

Na obrázku 3.6 je zobrazen AnimTree strom auta, tvořeného v rámci této práce. Nalevo se nachází `AnimTree Node`, který obsahuje synovské uzly vedoucí k různým `SkelControls`. `SkelControlsWheel` ovládají natočení kol u auta kolem osy Z. Dále `SkelControlsLookAt` mají na starost tlumení u kol a každý z nich je omezen limitem `SkelControlsSingleBone`. Pro doplnění, v nastavení mohou být `SkelControls` propojeny s UnrealScript kódem pomocí parametru `Control Name`.



Obrázek 3.6: Příklad AnimTree stromu

3.8 Unreal FrontEnd

Unreal FrontEnd je speciální nástroj na kompletaci scén. Umožňuje kompilaci UnrealScript souborů, balení dat do balíčků, vytvoření instalačního balíku a nahrání binárního IPA souboru na zařízení s operačním systémem iOS. Uživatel si může vytvářet a upravovat profily, přidávat k profilům různé mapy, zvolit mód Release/Debug a mnoho dalších nastavení.

3.9 Tvorba uživatelského rozhraní

Jedním z velmi důležitých aspektů her je uživatelské rozhraní (UI). V UE3 (UDK) uživatelské rozhraní poskytuje prostředky pro tvorbu tzv. HUD, které jsou většinou využity k zobrazování aktuálních informací během hry.

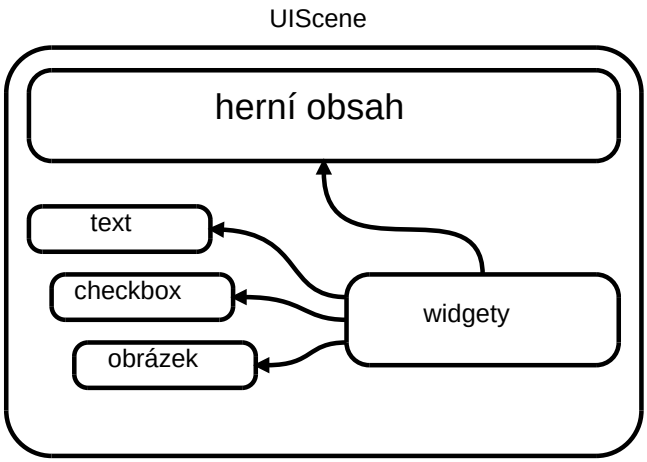
UI se v UE3 dělí do tří hlavních částí: UIScene, kolekce widgetů a Kismet sekvence. UI systém rozlišuje tvorbu UIScene od tvorby HUD a každá tato část sestává z jiných možností nastavení. Technika je ovšem velmi podobná.

Průběh tvorby lze rozdělit do tří kroků:

- návrh vzhledu
- funkcionalita UI
- implementace

3.9.1 UIScene

UIScene je první věc, kterou uživatel při spuštění hry uvidí. Skládá se z funkčních bloků zvané widgety, jenž zahrnují objekty jako jsou tlačítka, bary, posuvníky a další komponenty tvořící finální uživatelské rozhraní. Obrázek 3.7 znázorňuje vztah mezi prvky ve hře.



Obrázek 3.7: Vztahy mezi herními prvky

Uživatelské rozhraní v UE3 může být dále zpracováno pomocí Scaleform GFx, jenž umožňuje použití technologie Adobe Flash. Tuto možnost ale v práci rozvádět nebudeme.

Kapitola 4

Návrh a tvorba scény

Tato kapitola pojednává o návrhu a následné tvorbě finální scény. Je potřeba rozvrhnout si především rozměry scény a obsah. S tím souvisí i komplexnost objektů (např. počet polygonů u importovaných modelů) a náročnost na zpracování (počet instrukcí u materiálů).

Základní jednotkou rozměru v UDK je Unreal Unit (UU), která je ekvivalentní jednotce 3DS Unit a její rozměr reprezentuje délku 2cm. Tento poměr může být změněn, je ale potřeba dát pozor na správnou funkčnost ostatních komponent (např. změna šíření okolních zvuků). Je dobré vždy tvořit podle reálných proporcí.

4.1 Koncept

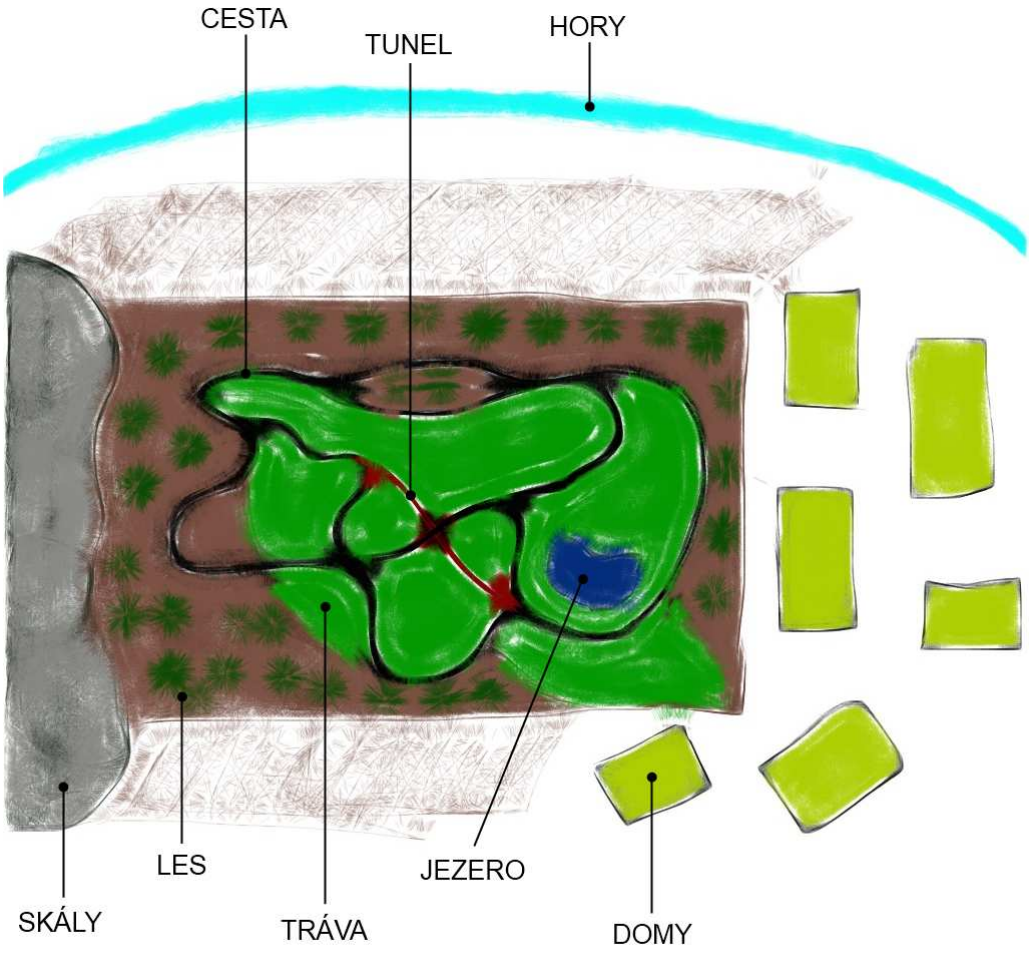
Základ návrhu stojí na myšlenkách autora, ze kterých by měl být utvořen koncept (nejlépe kreslený či psaný). Zahrnuty v něm musí být především prvky, které se výrazně podílí na scéně (např. tvar terénu, objekty ve scéně). Detaily typu odlesk od hladiny vody nejsou klíčové a mohou se domýšlet během návrhu. Samozřejmě čím detailněji je koncept zpracován, tím více je limitováno přepracovávání a z toho plyne i potencionální časová úspora.

Na obrázku 4.1 je zobrazen jednoduchý koncept prostředí, ze kterého budeme při tvorbě vycházet. Hlavní část tvoří terén, po kterém bude jezdit auto, tudíž musí být vytvořen jako první. Přímo na něj navazuje podzemní tunel, což sice není důležitý prvek scény, v určitých místech ale nahrazuje terén. Dalším aspektem je hratelnost scény a tu nejvíce ovlivňuje auto. Vymodelujeme ho v externím programu, importujeme do UDK a propojíme s kódem. Osvětlení je prvek, který má největší podíl na scéně. Vytvoříme komplexní denní cyklus a poté budeme postupně přidávat zbylé objekty do scény. Jako poslední vytvoříme jezero.

4.2 Terén

Prvním krokem je vložení nového terénu. Pro jeho editaci musí být aktivní Terrain Editing Mode. Detail terénu tvoří kopce a doliny, toho dosáhneme štětcem Paint s velkými hodnotami `Radius` a `Falloff`. Na terénu vzniknou různé výškové rozdíly, zjemnění provedeme štětcem Smooth. Veškeré úpravy tvaru povrchu je potřeba dělat na aktivní výškové mapě (`HeightMap`).

Vzhled povrchu terénu lze upravovat pomocí vrstev (`Layers`). Každé této vrstvě je přiřazen `Terrain Layer Setup`, který obsahuje seznam materiálů typu `Terrain Material`. `Terrain Material` v sobě ukrývá klasický materiál (typ Material) a nastavení jeho vlastností použití, např. otočení, velikost, způsob mapování. `Material` je pak složen z textury



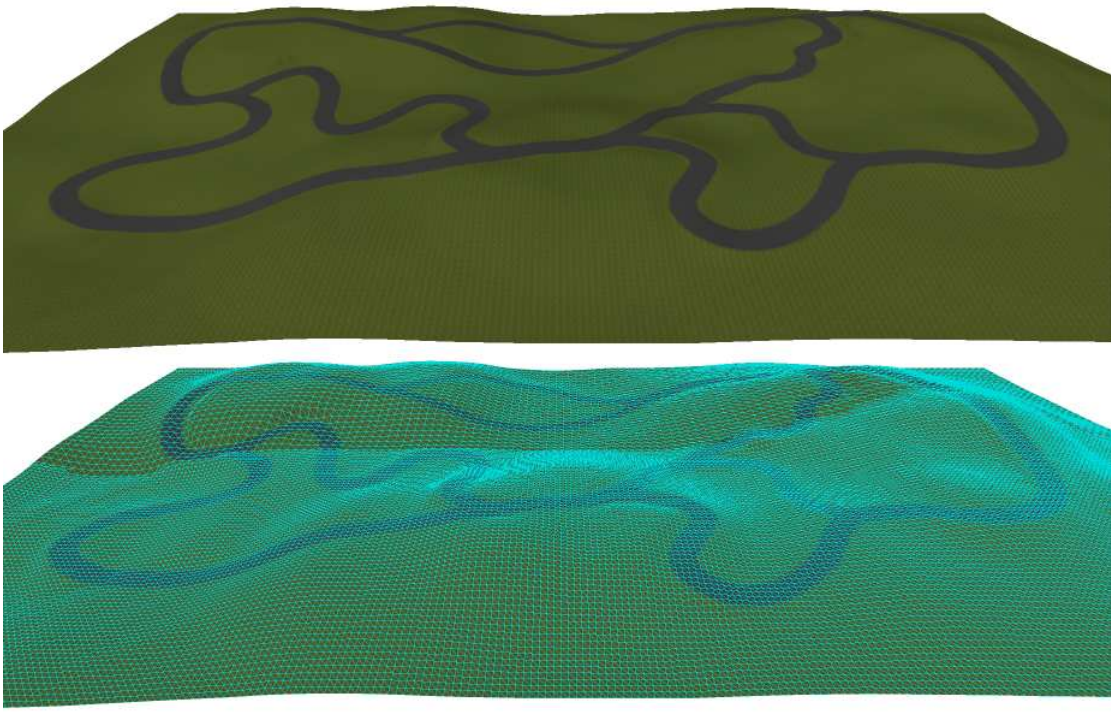
Obrázek 4.1: Koncept navrhované scény

a normálové mapy. My použijeme celkem 3 textury a to pro lesní půdu, travnatou zem a cestu. Při vložení první vrstvy se celý povrch terénu pokryje přiřazeným materiálem jako základ. Nanášení materiálu na povrch se děje při použití nástroje Paint s aktivním Terrain Layer Setupem v Terrain Editing Mode okně. V případě nanesení více materiálů na stejnou část plochy povrchu se zobrazí ten, jehož `Terrain Material Setup` je zobrazený v seznamu jako poslední z účastněných. Pořadí lze sice měnit, ale nedoporučuji.

U nanášení materiálu cesty se setkáváme s jedním nepříjemným efektem a tím jsou „zoubky“ u přechodů materiálů. Je to způsobeno nedostatečnou velikostí jednotek terénu. Terén je určen pro obrovské plochy a je potřeba brát v úvahu poměr mezi kvalitou a výkonem. V našem případě bude scéna velmi náročná na výpočetní výkon, zejména kvůli vysoké tesselaci. Po jejím zvýšení musíme ovšem povrch upravit štětcem Smooth, jelikož dojde ke zjemnění polygonální sítě, ale detail zůstane nezměněn. Tvar povrchu cesty je horizontálně rovný, dobrého výsledku docílíme jedním tahem se štětcem Average. Na vozovku a okraje pak použijeme odděleně znovu štětec Smooth. Náklon vozovky v zatáčkách neuvažujeme. Pokud je tvar cesty připraven, můžeme nanášet materiál asfaltu.

Při použití jedné vrstvy materiálu můžeme pozorovat efekt opakující se textury, což

můžeme i slabě vidět na travnatém povrchu u obrázku 4.2. Na horním terénu je ukázka použití více vrstev při vysoké tesselaci. Všimněte si jemné návaznosti okrajů cesty. U spodního terénu je zapnuta mřížka znázorňující dva stupně tesselace v závislosti na vzdálenosti kamery od tesselovaného povrchu. Při úplném přiblížení je terén o další 2 stupně jemnější.



Obrázek 4.2: Terén s vrstvami

4.3 Tunel

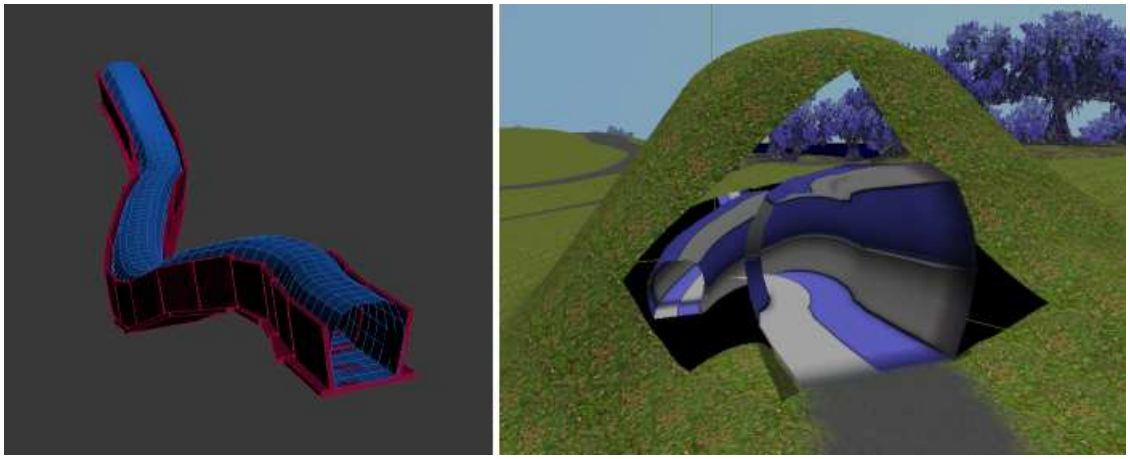
Podzemní tunel vytvoříme v externím modelovacím programu (v mém případě 3DS Max) a označíme jej nějakým názvem. Kolem tohoto modelu je potřeba ještě vytvořit tzv. kolizní model ohraničující tunel zespoda a zboku. Ten slouží pro to, aby nám po vjezdu auta do tunelu nepropadlo auto skrz tunel. Kolizní model musí mít stejný název jako model, ke kterému se váže, akorát s předponou „UCX_“. V opačném případě UDK kolizní model nerozpozná. Dále musí pro kolizní model platit, že je konvexní.

Pomocí modifikátoru `UVW unwarp` si navrhneme „texture layout“, který poté vyrenderujeme a v grafickém programu otexturujeme. Detailně je tento postup popsán v následující sekci modelování auta.

Tunel s kolizním modelem exportujeme do ASE souboru a ten importujeme do UDK. V Content Browseru se objeví `StaticMesh` s náhledem tunelu. Tunel můžeme vložit do scény jako `StaticMesh`, upravíme velikost tak aby nám seděla k cestě a přiřadíme mu materiál.

Aby se dalo tunelem projet, na obou stranách musí být skryt terén, který ho protíná. V Terrain Editing Mode zvolíme štětec Visibility a skryjeme pouze ty polygony, které zasahují dovnitř tunelu.

Na obrázku 4.3 můžeme vlevo vidět tunel (modrý model) s kolizním modelem (tmavě růžový) vymodelovanými v 3DS Maxu. Vpravo je terén, v němž je zasazen tunel, který navazuje na cestu (model má obrácené normály vnějšího povrchu). Důležité ovšem je, aby přesně seděl kolizní model tunelu, jinak může auto poskočit na zdánlivě rovném povrchu. V horších případech auto propadne nebo je nesmyslně katapultováno do vzduchu. Části, které zasahovaly do tunelu jsou skryty a musí být obklopeny jinými objekty, např. kameny.



Obrázek 4.3: Tunel s usazením do terénu

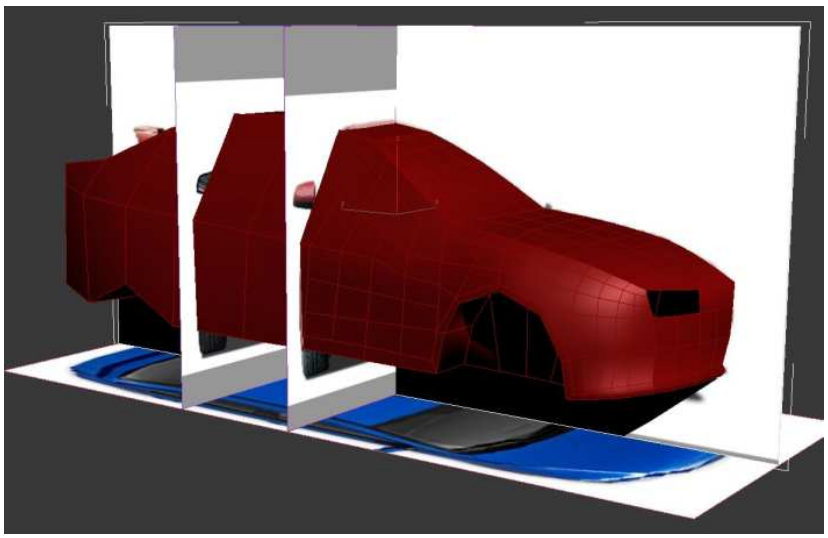
4.4 Auto

Vzor auta zvolíme dle libosti, v našem případě si vše vysvětlíme na Mitsubishi Lancer Evolution. V externím programu postupně vymodelujeme auto, nahrajeme do UDK, vytvoříme potřebný obsah a vše propojíme s kódem. Modelování auta je složitější a k zachování proporcí budeme potřebovat minimálně pohledy shora, zboku a zepředu.

Obrázek 4.4: Rozvržení referenčních obrázků v 3DS Max

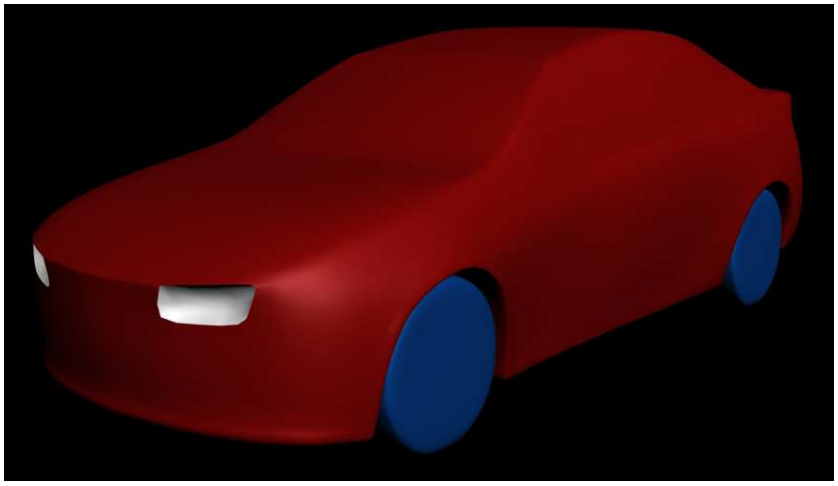
4.4.1 3DS Max

Pro správnou funkčnost modelu v UDK musí čelo modelu směřovat s kladnou osou X. Než započneme samotné modelování, vytvoříme si z referenčních obrázků základ ve formě kolmých rovin, které nám pomohou při editaci tvarů, viz. obrázek 4.4. Vložíme do scény kvádr podle velikosti auta na obrázku a v módu `Editable Poly` pomocí nástroje `Cut` řežeme polygony na povrchu. Vytvořené hrany a vrcholy posouváme podle okrajů výrazných rysů auta na fotkách. Pro přesnost a jednoduchost můžeme modelovat pouze polovinu auta, výsledný model pak vznikne použitím modifikátoru `Symmetry`.



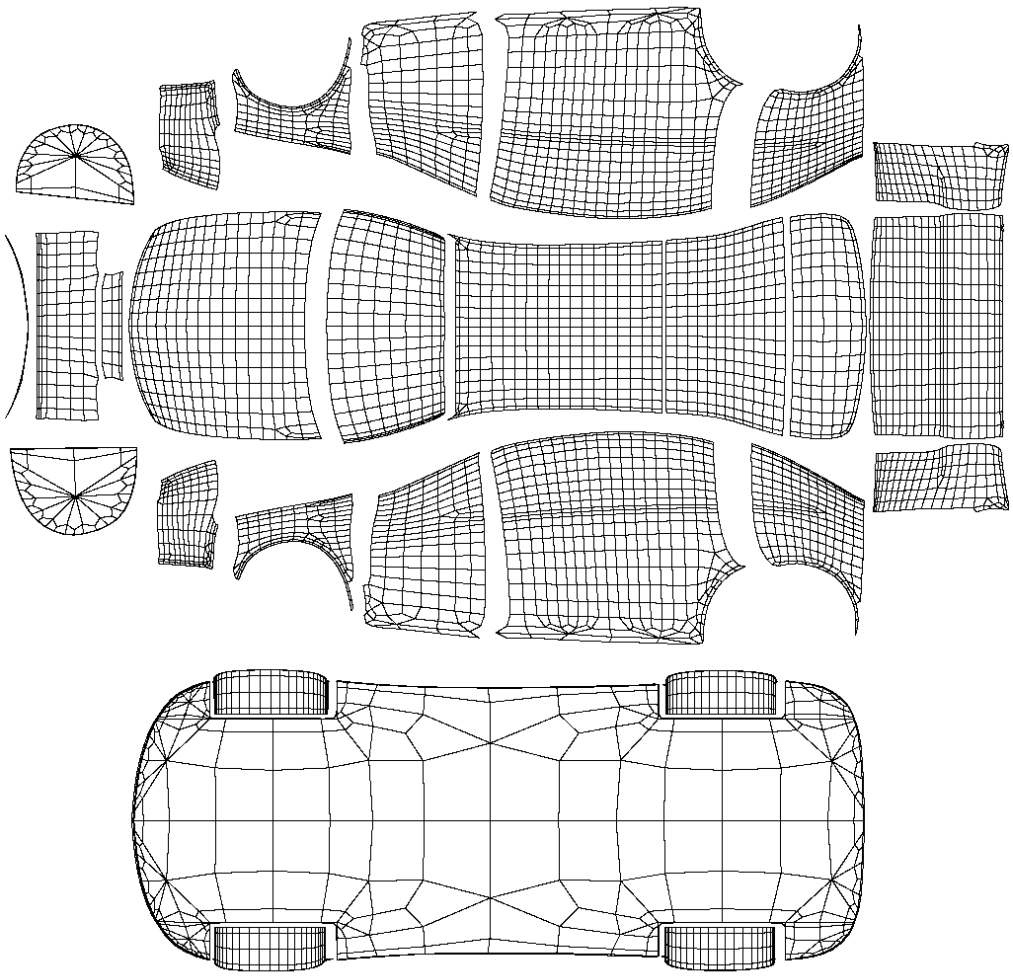
Obrázek 4.5: Rozpracovaný model auta

Abychom neměli model v konečném výsledku příliš hranatý, aktivujeme `Subdivision Surface`, stačí jedna iterace pro zachování relativně nízkého počtu polygonů. Kola a světla se musí modelovat jako objekty zvlášť a po nastavení materiálů se mohou objekty spojit do jednoho.



Obrázek 4.6: Výsledný model auta

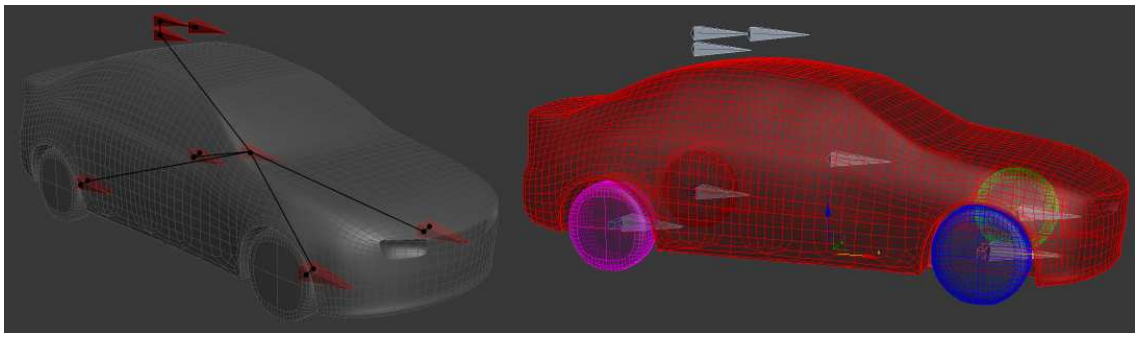
Materiálové kanály se správně do UDK naimportují pouze pokud má v 3DS Maxu každý objekt/element nastaven jiný `Material ID Channel` a je mu přiřazen materiál. Než spojíme kola, světla a auto dohromady, je potřeba aplikovat modifikátor `UVW unwarp` na každý z nich samostatně. Ten vygeneruje tzv. UVW mapu, která mapuje souřadnice textury na model. Automatické rozvržení UVW mapy u komplexních objektů dopadá velmi špatně, proto musíme vytvářet její části jednotlivě po shlucích polygonů. Výsledná UVW mapa je zobrazena na obrázku 4.7.



Obrázek 4.7: UVW mapa

Jako poslední krok musíme vytvořit skelet auta a jednotlivým kostem přiřadit váhu vrcholů. Vytvoříme proto objekt typu `Bones` u každého kola a jeho tlumiče. Tyto kosti budou sloužit pro natáčení kol a pružení tlumičů. Aby se navzájem ovlivňovaly, vytvoříme hlavní kost uprostřed auta, ta bude brána jako těžiště. Nastavíme hierarchii kostí, na model použijeme modifikátor `Skin` a jednotlivým kostem přiřadíme vrcholy modelu. Skelet je použit pro fyzikální simulaci, kterou má na starost systém PhysX. Podle výsledku simulace se pak chovají jednotlivé části auta, jedná hlavně o odpružení při zrychlení, brždění, zatáčení apod. Každá kost má přiřazené jméno, které je důležité pro jednoznačnou identifikaci elementů modelu v UnrealScript kódu.

Na obrázku 4.8 je zobrazeno auto s výsledným skeletem (nalevo) a barevně odlišenými elementy, které patří ke svým kostem (napravo). Pomocí plug-inu ActorX exportujeme model do souboru PSK.



Obrázek 4.8: Skelet auta

4.4.2 UDK

V UDK si vytvoříme nový package s názvem `VH_Lancer`. Importujeme do něho model vytvořený v 3DS Maxu a UVW mapu zpracovanou v grafickém programu – texturu. UDK rozpozná model jako `SkeletalMesh`. Pro správnou funkčnost musíme vytvořit a nastavit:

- `Material`
- `Physics Asset`
- `Physical Material`
- `AnimTree`

4.4.3 UnrealScript

V sekci s kódem vytvoříme třídu `UTVehicle_Lancer` odvozenou od `UT_Vehicle`, nastavíme vlastnosti vozidla a přiřadíme kola:

```
1 Begin Object Class=UTVehicleLancerWheel Name=RRWheel
2     BoneName="bone_wheel_rr"
3     BoneOffset=(X=0.0,Y=0.0,Z=0.0)
4     SkelControlName="wheel_rr_cont"
5 End Object
6 Wheels(0)=RRWheel
```

Analogicky pro ostatní kola. Ve zdrojovém kódu si můžeme všimnout jména `bone_wheel_rr` a `wheel_rr_cont`. První je totožné s názvem kosti zvoleném v 3DS Maxu a druhé slouží k propojení s AnimTree (zajišťuje animace). Kola jsou odvozeny od třídy `UDKVehicleWheel` a nastavení vlastností se nachází v samostatné třídě `UTVehicleLancerWheel`.

Abychom mohli ve scéně využít model auta jako vozidlo, musíme vytvořit UnrealScript soubor s třídou `UTVehicleFactory_Lancer` odvozenou od `UTVehicleFactory` a nastavit její vlastnosti. Po kompilaci skriptů se vozidlo zobrazí v seznamu Actor Classes.

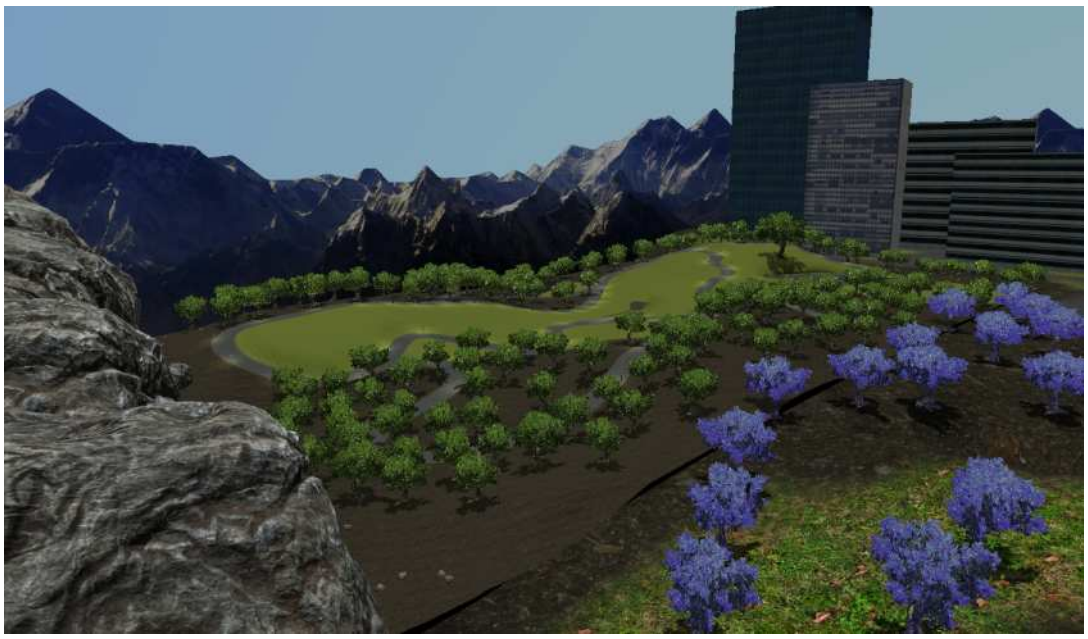
```
1 Begin Object Name=SVehicleMesh
2     SkeletalMesh=SkeletalMesh'VH_Lancer.Mesh.SK_Lancer'
3     Translation=(X=0.0,Y=0.0,Z=-60.0)
4 End Object
5 VehicleClassPath="Simulator.UTVehicle_Lancer_Content"
6 DrawScale=0.05
```

Poslední nutností je třída `UTVehicle_Lancer_Content` odvozená od `UT_Vehicle_Lancer`, kterou jsme vytvořili dříve. Ta zachází více do detailu a obsahuje nastavení o konkrétním objektu:

```
1 Begin Object Name=SVehicleMesh
2     SkeletalMesh=SkeletalMesh'VH_Lancer.Mesh.SK_Lancer'
3     AnimTreeTemplate=AnimTree'VH_Lancer.Anims.AT_Lancer'
4     PhysicsAsset=PhysicsAsset'VH_Lancer.Mesh.PA_Lancer'
5 End Object
```

4.5 Objekty prostředí

Na obrázku 4.9 můžeme vidět výslednou scénu, které se budeme snažit přiblížit. Obsahuje navíc stromy, pouliční lampy, domy, hory a skály.



Obrázek 4.9: Prostředí z ptačí perspektivy

Skály vytvoříme pomocí skládání zvětšených kamenů, které jsou obsaženy v UDK. Domy a hory vymodelujeme v 3DS Maxu, s použitím modifikátoru `UVW unwarp` otexturujeme v grafickém programu UVW mapu a importujeme vše do UDK. V něm vytvoříme a přiřadíme materiály importovaným `StaticMesh` objektům a doplníme je do scény.

U lamp a stromů je situace složitější, je jich totiž ve scéně hodně. Při klasickém umísťování objektů do scény by mnoho stejných objektů zabíralo velkou část paměti a umisťování by trvalo dlouho. UDK obsahuje různé metody pro práci s těmito případy a my využijeme dvě z nich. První se jmenuje `Archetype` a použijeme ho pro pouliční lampy. Druhý slouží k nanášení objektů jako vrstvu u terénů, tzv. `DecoLayer`.

V 3DS Maxu si vymodelujeme lampu, dále pomocí UVW unwarp rozložíme mapu na sloup a baňku. Po importu vytvoříme pro tento `StaticMesh` objekt materiál, ve kterém pro baňku využijeme proměnnou – parametr a `Emissive Channel`. Z materiálu zdědíme `MaterialInstanceTimeVarying`, který budeme ovládat následně pomocí Matinee. `StaticMesh` přidáme do scény, vytvoříme z něho `Archetype` a nahradíme jím `StaticMesh`. Dále do scény přidáváme pouze vytvořený `Archetype`. V případě 70-ti lamp ve scéně je v paměti uložen `Archetype` na rozdíl od `StaticMesh` pouze jednou.

Abychom mohli přidat do scény stromy, přepneme se do Terrain Editing Mode a vytvoříme novou vrstvu `DecoLayer`. Ve seznamu vlastností terénu vybereme u této `DecoLayer` u parametru `Factory` objekt `StaticMeshComponentFactory`. Dále objektu přiřadíme strom – jakýkoliv `StaticMesh` obsahující UDK. U vlastností `StaticMeshComponentFactory` pak nastavíme vlastnosti jako `Density`, `Rand Seed` apod. Nyní je vrstva `DecoLayer` nastavená a my můžeme pomocí štětce Paint (pozor na aktivní vrstvu `DecoLayer` vs. `HeightMap`) nanášet na povrch terénu stromy.

4.6 Osvětlení

Osvětlení scény je kritickou oblastí, budeme se snažit vytvořit plně dynamický denní cyklus. Techniky popsané zde jsou diskutabilní, dovedou nás ovšem k požadovanému výsledku.

Základní funkci osvětlení ve scéně plní `DominantDirectionalLightMovable` (DDLM). Jedná se o pohyblivé přímé světlo simulující sluneční paprsky, závislé pouze na nastaveném směru svitu. Slunce by správně mělo být ve scéně nahrazeno typem `PointLightMovable` a mělo by se otáčet kolem scény, výpočet osvětlení by však trval velmi dlouho a prostor kolem scény by musel být mnohokrát větší. V našem případě umístíme do scény DDLM s konstantně měnícím se směrem svitu.

Obrázek 4.10: Scéna ve dne

Objekty nyní (obrázek 4.10) vrhají černé stíny, ve scéně totiž chybí okolního osvětlení. Do scény tedy přidáme světlo typu `SkyLight` s takovým nastavením, aby působila přirozeně. V momentě, kdy „slunce zapadne“, se ale scéna nepromění v absolutní tmu, nýbrž je stále osvětlena okolním světlem. Nastavení tedy změníme tak, aby působila přirozeně v noci. Osvětlení ve dne budeme korigovat dvěma světly typu `DirectionalLight` pomocí Matinee. Jedno přidáme do scény s paprsky směřujícími dolů a druhé s paprsky směřujícími nahoru. Scéna v noci je zobrazena na obrázku 4.11.

Obrázek 4.11: Scéna v noci

Při velmi tupých úhlech svírající směr svitu světla s terénem vznikají na povrchu terénu nepříjemné artefakty, viz. obrázek 4.12. Řešení spočívá buď ve vypnutí vrhání stínů u terénu a nebo zamezením těchto situací, což v našem případě představuje umístění skal a budov.

Obrázek 4.12: Artefakty na povrchu terénu

U DDLM ve scéně zapneme vlastnost `Render Light Shafts` a nastavíme patřičné hodnoty u parametrů `Bloom`, což má za následek vykreslování slunečních paprsků. Scéně to dodá na realističnosti. Jelikož západ a východ slunce působí jinými barvami, vše musíme animovat přes Matinee. Přidané sluneční paprsky můžeme pozorovat na obrázku 4.13.

Obrázek 4.13: Sluneční paprsky

4.6.1 Obloha

Na předchozím obrázku se již na obloze nachází „slunce“. To, jak vypadá obloha, ovšem se světly ve scéně nemá skoro vůbec nic společného. Obloha je ve skutečnosti `StaticMesh` kopule, které je přiřazen materiál. Tento objekt ignoruje veškeré kanály osvětlení a neprodukuje stíny, slouží pouze k vizualizaci. Pokud by měla být na minulém obrázku scéna, kterou v průběhu tvoříme, obloha by byla černá a vidět by šly pouze paprsky.

Do scény tedy přidáme `StaticMesh SkyDome` a vytvoříme pro něho nový materiál. Ten bude využívat materiálové kanály `Emissive` pro oblohu a `CustomLighting` pro slunce. Aby scéna vypadala korektně, použijeme výrazové bloky `Camera Vector` (směr, kterým se díváme) a `Light Vector` (směr paprsků světla, v tomto případě DDLM). Propojením s jinými výrazovými bloky vytvoříme gradient slunce, přidáme mu koronu a zapojíme do `CustomLighting` kanálu.

Samotnou oblohu vytvoříme ve dne gradientem odstínu modré a modrobílé barvy, noc uděláme s pomocí textury nebe. Přechody mezi dnem a nocí vyřešíme přidáním bloku `Lerp`, což je lineární interpolace mezi dvěma stavy ovládaná parametrem.

Výslednou oblohu můžeme vidět na obrázku 4.14, všimněme si jemného gradientu na pozadí a korony kolem obvodu slunce.

Obrázek 4.14: Výsledná obloha

Z materiálu nebe vytvoříme `MaterialInstanceTimeVarying`, nastavíme počáteční hodnoty a přiřadíme ke `SkyDome` ve scéně. Veškeré animace potřebné pro oživení scény vytvoříme v Matinee.

4.7 Voda

V rozpracované scéně bude voda tvořit jezero s mnoha vlastnostmi. Nejprve je potřeba získat referenční obrázky pro vytvoření dvou Cube map (Cube mapa použitá pro odlesky při denním osvětlení je zobrazena na obrázku 4.15), které slouží k vytvoření iluze odlesků.

Obrázek 4.15: Cube mapa

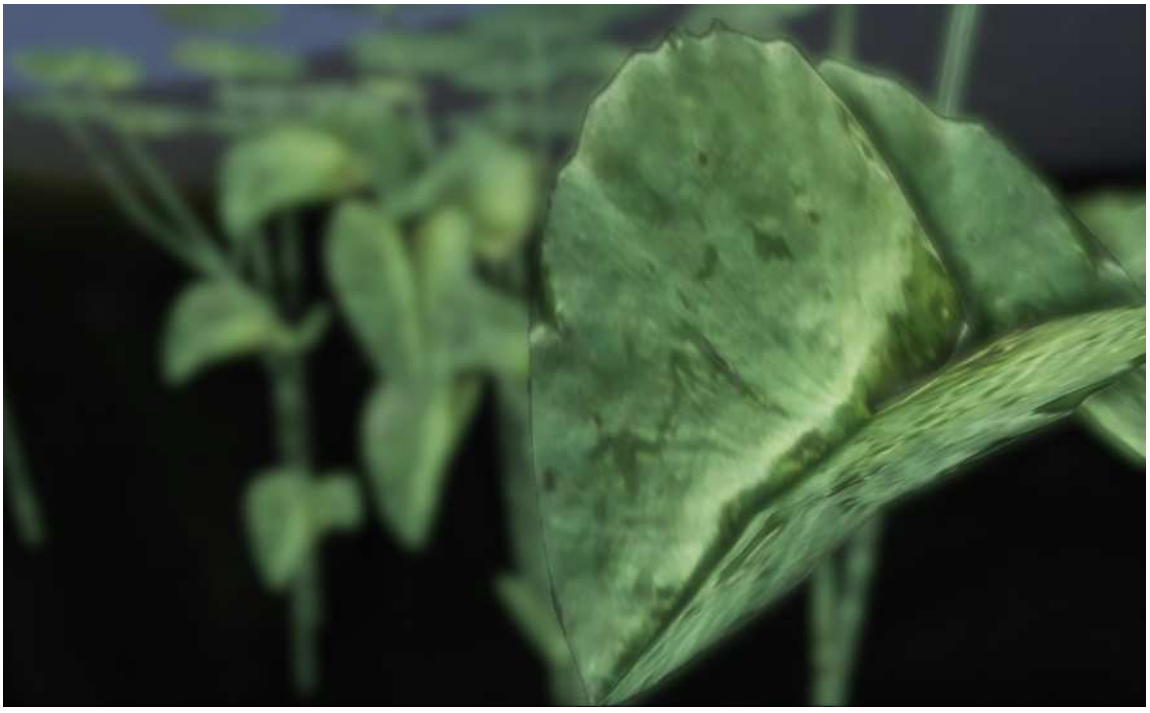
Obrázky pro Cube mapy získáme např. pomocí printscreenu hotové scény z pozice vody. To značí 6 obrázků pro odlesk ve dne a 6 obrázků pro odlesk v noci. Po zpracování je importujeme do UDK jako textury a ty pak vložíme do Cube mapy určené pro denní, resp. pro noční odlesky na hladině.

Pomocí BSP štětců vytvoříme čtvercovou plochu v místech návrhu tak, aby nikde nepřesahovala terén. Poté vytvoříme materiál, z něhož zdědíme `MaterialInstanceTimeVarying` a ten přiřadíme vytvořené ploše. Po dokončení materiálu a nastavení implicitních hodnot instance ji budeme animovat přes Matinee.

4.7.1 Fyzikální vlastnosti

Hlavními vlastnostmi vody je pro nás barva, odlesky, vlnění, lom světla, mlžení a tření. Pro poslední dvě z vyjmenovaných využijeme tzv. Volume, což je vymezený prostor, v němž je aplikován efekt nebo obecněji funkcionalita v tomto prostoru podle typu Volume.

Na obrázku 4.16 můžeme vidět efekt mlžení, které je pro lidské oči pod vodou přirozený. Tohoto efektu dosáhneme tak, že celý prostor pod vodní hladinou vykryjeme BSP štětcem a vytvoříme `PostProcessVolume`. V jeho vlastnostech aktivujeme `bEnableDOF`, což zapne pro hráče ve vymezeném prostoru efekt Depth of Field.



Obrázek 4.16: Depth of Field efekt ve vodě

Pro totožný BSP štětec vytvoříme na stejném místě `UTWaterVolume`, jedná se o speciální `PhysicsVolume` s přednastaveným chováním objektů v prostředí vody. Ovlivňuje rychlost pohybu v prostoru, tření s povrchem, gravitaci a v neposlední řadě i úbytek zdraví simulující topení.

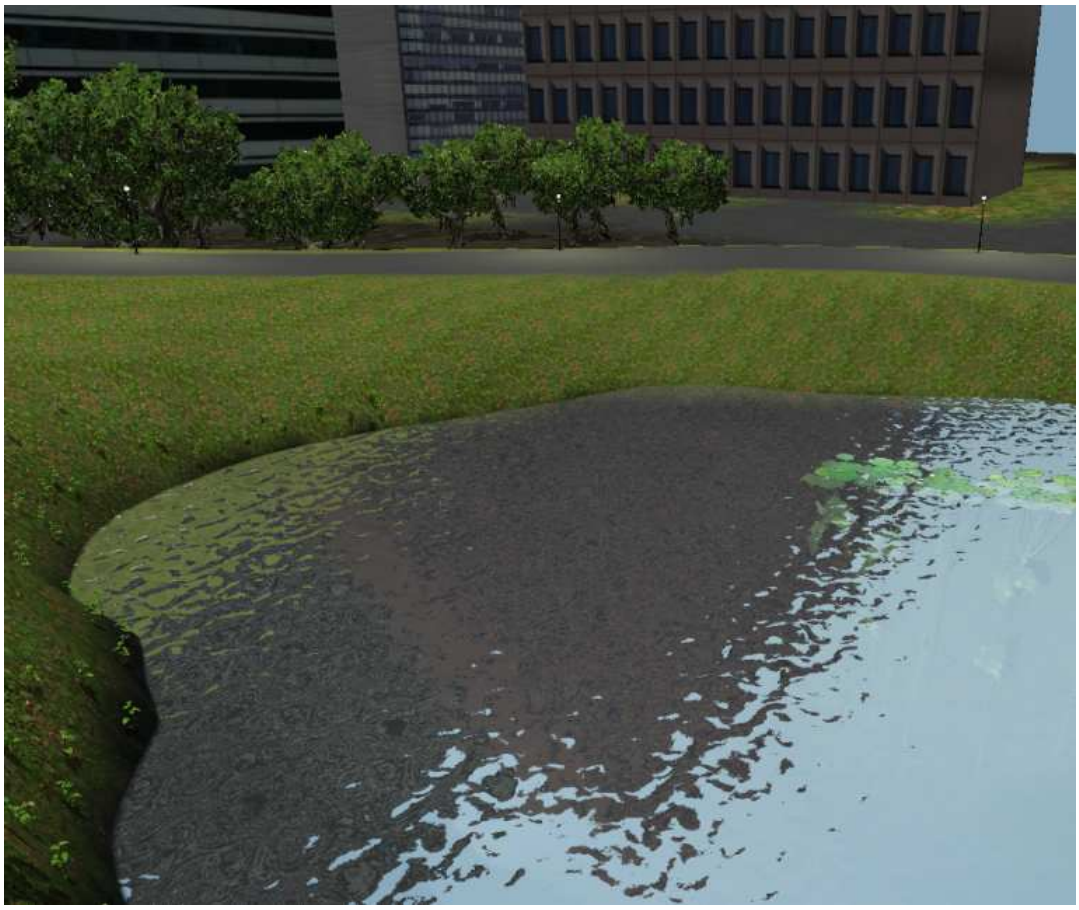
4.7.2 Materiál

Materiál vody, který dále vytvoříme, bude zahrnovat bloky

- barva povrchu
- odraz okolí
- odraz slunce
- vlny
- průhlednost
- lom světla

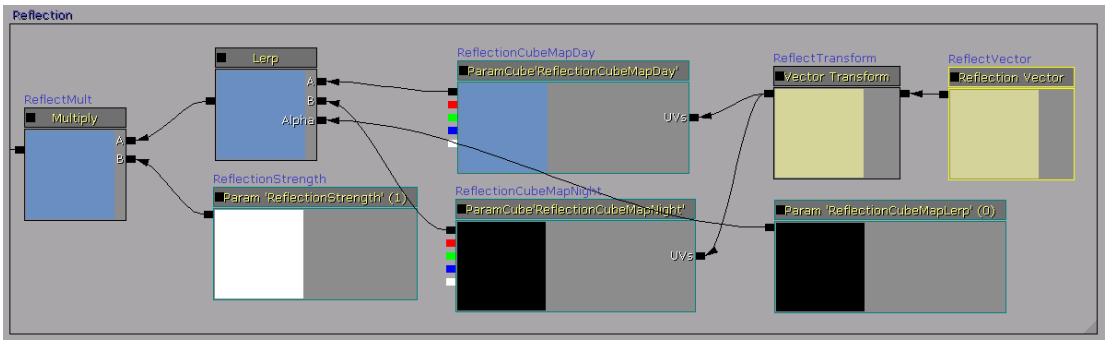
a využívat kanály `Emissive`, `Opacity`, `Normal`, `CustomLighting` a `Distortion`.

Barvu vody určuje jeden vektor vynásobený alfa kanálem normálové mapy. K němu přidáme odlesk okolí, závislém na úhlu pohledu, použijeme tedy výraz `Reflection Vector`. Reflexi vytvoříme z připravených Cube map pro den a noc s pomocí lineární interpolace ovládané přes Matinee. Výsledek připojíme do kanálu `Emissive`. Odraz na vodní hladině můžeme vidět na obrázku 4.17, všimněme si na ní rozmazané budovy.



Obrázek 4.17: Odlesk od hladiny

Odraz slunce vytvoříme pomocí `Light Vector`, `Reflection Vector`, vektorů barev a matematických výrazů. Výsledek pak připojíme do kanálu `CustomLighting` a `Lighting Model` nastavíme na `MLM_Custom`. Sekvenci výrazů můžeme vidět na obrázku 4.18



Obrázek 4.18: Sekvence výrazových bloků tvořící odraz na vodní hladině

Vlny na vodní hladině je pouze iluze, kterou získáme využitím výrazů `Panner`, `Time`, `Mask` a `Texture Sample` (využijeme texturu s náhodnými barevnými gradienty). Tyto výrazy budou tvořit základ pro normálové mapy, které dodají plasticitu a dále se podílejí na barvě hladiny.

Pro správnou funkčnost průhlednosti musí mít `Material Node` nastavenu vlastnost `Blend Mode` na hodnotu `BLEND_Translucent`. Výšku vody dostaneme odečtením hodnoty výrazu `PixelDepth` od hodnoty výrazu `DestDepth`. Podle rozdílu pak lineární interpolací zajistíme průhlednost materiálu připojením ke kanálu `Opacity`.

Efekt lomu světla v UDK vytvoříme na odlišném principu než jaký zaujímá fyzikální pohled. V našem případě se jedná pouze o vnášení chyb do obrazu. Jako zdroj různých hodnot nám poslouží normálová mapa vodní hladiny, kterou vynásobíme vektorem, tvořeným náhodně zvolenými hodnotami. Ty upravujeme tak dlouho, dokud nebudeme s vizuálním výsledkem spokojeni, viz. obrázek 4.19.

Obrázek 4.19: Lom světla ve vodě

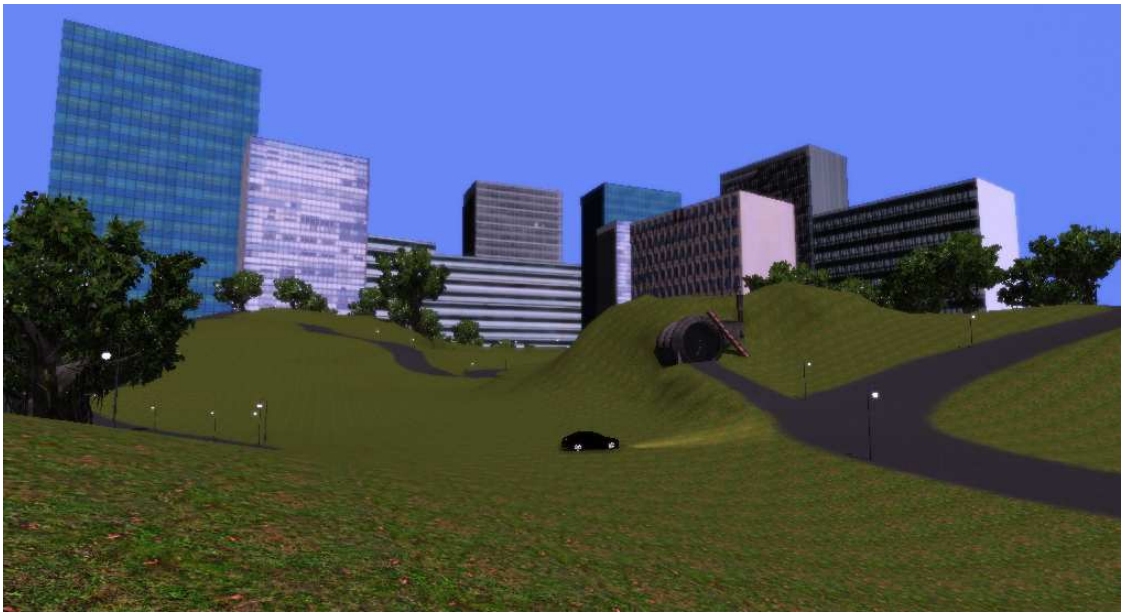
4.8 Finální vzhled scény

Pro celkově živější vzhled scény upravíme hodnoty středních a vysokých tónů ve `World Info`. Do `Game Types Supported` přidáme nový typ hry `UDKGame` a nastavíme informace o mapě. Nakonec musíme nastavit hodnotu `Kill Z` v sekci `Zone Info`, aby se hráč při vylétnutí z mapy objevil znovu na počáteční pozici. Na následujících obrázcích pak můžeme vidět finální scénu, vytvořenou v této práci.

Na obrázku 4.20 je zobrazen terén s tunelem. Kolem tunelu jsou umístěny objekty zakrývající díry v terénu, které jsou potřeba k projetí místa protnutí tunelu s terénem. Obrázek 4.21 ukazuje průjezd auta tunelem v noci. Osvětlení pochází od světlometů na autě. Pohled z dálky je zobrazen na obrázku 4.22 a jedná se o pohled z první osoby. Detailní záběr na osvětlení od světlometů auta znázorňuje obrázek 4.23. A jako poslední můžeme vidět na obrázku 4.24 propracovaný západ slunce s vodou a autem.

Obrázek 4.20: Začátek tunelu

Obrázek 4.21: Auto v tunelu při nočním osvětlení



Obrázek 4.22: Pohled z dálky

Obrázek 4.23: Osvětlení od světlometů

Obrázek 4.24: Západ slunce

Kapitola 5

Závěr

V této práci jsme se seznámili s hlavními technikami, které využívá Unreal Engine 3. Dále byly rozebrány nástroje Unreal Development Kitu pro práci s assety, čtenář by tedy měl mít základní vědomosti k čemu slouží a jak s nimi pracovat. Formou návodu byly popsány klíčové postupy při tvorbě scén nejen herních, ale i filmových. V neposlední řadě by čtenář měl být obeznámen s důležitostí návrhu scény před samotnou tvorbou, díky čemuž lze během ní předcházet řadě problémů. Tvorba scény byla dekomponována na části, které jsme vytvářeli postupně od základních (terén) až po specifické (voda). Zároveň jsme si vysvětlili od počátku postup a důležité kroky k vymodelování auta a zprovoznění v UDK.

Unreal Engine a UDK je komplexní řešení, které odděluje problémy a náročnost nízkoúrovňového programování od tvorby obsahu, na níž se uživatel může plně soustředit.

Při vypracovávání jsem se seznámil s profesionálními nástroji a technologiemi pro tvorbu velkých her a získal jsem velice dobrou a především realistickou představu o průběhu nejen vytváření samotného obsahu, ale i průběhu po stránce finanční a marketingové.

Splněny byly všechny body zadání. Výsledkem této práce je trailer a spustitelná aplikace demonstrující otevřenou scénu s objekty zde tvořenými.

5.1 Využitelnost práce

Obsah práce může být využit jako částečný návod jak dosáhnout určitých požadavků v UDK. Vzhledem k tomu, že se tvorba her stala samostatnou, intenzivně se rozvíjející oblastí, může být obsah využit také při vyučování studentů v předmětech, zaměřujících se právě na tvorbu her s využitím UDK nebo kteréhokoliv jiného profesionálního řešení.

5.2 Budoucí vývoj

Rozšíření této práce by mohlo spočívat například v oblasti uživatelského rozhraní nebo síťové komunikace, kdy by bylo možné vytvořit závodní hru po síti. Nicméně rád bych pokračoval v dalším studiu na FIT VUT a v rámci diplomové práce zrealizoval plně funkční hru s pokročilým zpracováním pro platformu iOS, konkrétně zařízení iPad/iPhone.

Literatura

[1] Golding, J.: Lighting Reference [online]. rev. 2011-05-05 [cit. 2011-05-06].
URL http://udn.epicgames.com/Three/LightingReference.html

[2] Haines, S.: Unreal Lightmass – Static Global Illumination for Unreal Engine 3 [online]. rev. 2011-05-04 [cit. 2011-05-05].
URL http://udn.epicgames.com/Three/Lightmass.html

[3] James M. Van Verth, Lars M. Bishop: *Essential Mathematics For Games And Interactive Applications*. Burlington: Morgan Kaufmann, 2008, ISBN 978-0-12-374298-8, 704 s.

[4] Jason Busby, Zak Parrish, Jeff Wilson: *Matering Unreal Technology, Volume I: Introduction to Level Design with Unreal Engine 3*. Indianapolis: Sams, 2009, ISBN 978-0-672-32991-3, 912 str.

[5] Jason Busby, Zak Parrish, Jeff Wilson: *Matering Unreal Technology, Volume II: Advanced Level Design Concepts with Unreal Engine 3*. Indianapolis: Sams, 2009, ISBN 978-0-672-32992-0, 1080 s.

[6] NVIDIA: PhysX Features [online]. 2011 [cit. 2011-05-06].
URL http://developer.nvidia.com/physx-features

[7] Porter, J.: Calling DLLs from UnrealScript [online]. rev. 2011-05-05 [cit. 2011-05-07].
URL http://udn.epicgames.com/Three/DLLBind.html

[8] Sweeney, T.: UnrealScript Language Reference [online]. rev. 2011-05-05 [cit. 2011-05-07].
URL http://udn.epicgames.com/Three/UnrealScriptReference.html

[9] Wright, D.: Shadowing Reference [online]. rev. 2011-05-05 [cit. 2011-05-06].
URL http://udn.epicgames.com/Three/ShadowingReference.html

Příloha A

Obsah DVD

Na přiloženém DVD se nachází:

- tato práce v elektronické podobě
- zdrojové soubory s návodem ke zprovoznění v UDK
- instalační soubor aplikace UDK Scene, obsahující spustitelnou scénu
- instalační soubor UDK verze March 2011
- trailer vytvořené scény
- plakát
- readme

Příloha B

Manuál

Po instalaci a startu aplikace UDK Scene se zobrazí vytvořená scéna, ve které se dá pohybovat. Pro určení směru je využito ovládání pomocí myši. Klávesy, které je možno použít ve scéně:

- W – pohyb dopředu
- S – pohyb dozadu
- A – úkrok doleva
- D – úkrok doprava
- E – nastoupení/vystoupení do/z vozidla
- F7 – funkční klávesa pro zobrazení módů vykreslování (mesh, lit, unlit, ...)
- ‘ – vyvolání konzole
  - `exit` – konec
  - `restartlevel` – restartování levelu
  - `show collision` – zapne zobrazování kolizí
  - `stat XXX` – zobrazí statistiky XXX (fps, game, ...)
  - `fly` – mód létání

Aplikaci lze ukončit voláním z konzole nebo křížkem u okna aplikace.