Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Simulátor dopravních situací pro výuku řízení

Bc. Vít Bejček

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Vít Bejček

Informatika

Název práce

Simulátor dopravních situací pro výuku řízení

Název anglicky

Traffic situation simulator for driving schools

Cíle práce

Diplomová práce je zaměřená na vývoj softwaru pro výuku řízení vozidel automobilové dopravy způsobem ohodnoceného řešení dopravních situací. Cílem je navrhnout nástroj využitelný v rámci výukových osnov autoškoly. Dílčím cílem je popsat práci s vybraným enginem pro tvorbu 3D aplikací.

Metodika

Metodika řešené práce je založena na analyticko-syntetickém přístupu. Bude provedena analýza odborných informačních zdrojů a na základě syntézy zjištěných poznatků a požadavků na software bude navržena a implementována aplikace pro simulaci dopravních situací při výuce řízení. Vývoj aplikace bude popsán, výsledná aplikace bude otestována a budou navrženy možnosti jejího případného dalšího rozvoje.

Doporučený rozsah práce

60-80 stran

Klíčová slova

Unreal Engine, C++, Blueprints, dopravní situace, umělá inteligence, výuka řízení vozidel, dopravní předpisy, model vozidla, simulace lokální dopravy

---

Doporučené zdroje informací

CORERA, A. *Visual C++ .NET pro programátory v C++.* Brno: Computer Press, 2003. ISBN 80-7226-860-0.

Epic Games. Unreal Engine 4 Documentation. [Online] 2015. https://docs.unrealengine.com.

Karamouzas, Ioannis, a další. A Predictive Collision Avoidance Model for Pedestrian Simulation. 2009.

MFF UK. AMIS – Artificial Minds for Intelligent Systems. [Online] 2015. http://artemis.ms.mff.cuni.cz.

Millington, Ian. Arteficial Intelligence for Games. San Francisco : Elsevier Inc., 2006. ISBN-13: 978-0-12-497782-2.

Předběžný termín obhajoby

2016/17 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 20. 2. 2016

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 20. 2. 2016

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 15. 03. 2017

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Simulátor dopravních situací pro výuku řízení" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 20.3.2017 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Jiřímu Brožkovi, Ph.D. za podnětné rady při vedení diplomové práce a svým blízkým za podporu při studiu.

Simulátor dopravních situací pro výuku řízení

Souhrn

Diplomová práce *Simulátor dopravních situací pro výuku řízení* se zabývá vývojem aplikace v prostředí Unreal Engine 4. Zkoumána jsou teoretická východiska klíčová pro následnou praktickou část práce. Pozornost je věnována herním enginům, nízko-úrovňovým programovým rozhraním pro vykreslování a enginům pro fyzikální simulace. Hlavním cílem práce je tvorba aplikace, která umožní začínajícím řidičům procvičit řízení osobního automobilu a dodržování pravidel silničního provozu. Vlastní softwarové řešení je implementováno s využitím frameworku Unreal Engine 4 v kombinaci s aplikacemi pro tvorbu grafických prostředků 3ds Max, Blender, Gimp a Photoshop. Logický model aplikace je objektově orientovaný a jeho tvorba proběhla v programovacím jazyku C++ a vizuálním skriptovacím jazyku BluePrint.

Klíčová slova: Unreal Engine 4, Epic Games Inc., C++ OOP, BluePrint, 3ds Max, Blender, 3D aplikace v reálném čase, 3D grafické prostředky, simulace vozidla, logický model vozidla, dopravní situace

Traffic situation simulation for driving schools

Summary

Diploma thesis *Traffic situation simulation for driving schools* deals with application development under Unreal Engine 4 environment. It examines theoretical basis which is essential for further production. The emphasis is given to understanding games engines, low level application interfaces for rendering graphics and physical simulation mechanics. Main goal is to develop software application which allows unexperienced drivers to practice vehicle manipulation under the supervision of traffic regulations. Software solution is implemented in Unreal Engine 4 environment in conjunction with tools which are used for producing graphical assets, such as 3ds Max, Blender, Gimp, Photoshop. Underlying logical model of software is object oriented and is implemented in C++ and visual scripting language BluePrint.

Keywords: Unreal Engine 4, Epic Games Inc., C++ OOP, BluePrint, 3ds Max, Blender, 3D application in real-time, 3D art assets, vehicle simulation, logic model of vehicle, traffic situation

Obsah

1 Úvod

Rozvoj osobních počítačů, jejich vybavení a jiných technologií není ničím překvapivý. Co lze však považovat za překvapivé, je rychle rostoucí trend v užívání 3D aplikací. [1] Tento trend může být způsoben značným nárůstem ve výkonu zařízení, které dokáží bez větších problémů zpracovávat a vykreslovat komplexní 3D scény v reálném čase. V tomto ohledu je důležité pozastavit se nad termínem „zpracování v reálném čase“, který je pro tématiku diplomové práce klíčový. Například výpočet vědeckých projektů nebo vykreslení animovaných sekvencí pro filmy nevyžaduje prvek interaktivity, a proto je kladen větší důraz na jejich přesnost než nízký čas odezvy. Takto odlišné požadavky mají následně vliv na výběr vhodných technologií. Jedním z dalších činitelů rostoucího počtu uživatelů 3D aplikací bude určitě postupné rozšíření zařízení do domácností konzumních uživatelů. A právě tyto důvody dávají vzniknout rozmachu v herním průmyslu, který cílí na koncové uživatele. V dnešní době rozhodně již nelze považovat společnosti v herním průmyslu za nevýznamné účastníky trhu.

Vývoj hardwaru a softwaru jde odjakživa ruku v ruce, ale právě při vývoji aplikací herního typu lze pozorovat, že je toto provázání ještě silnější. Ať už je příčinou neoblomný a dobře mířený marketing společností, nebo jen lidská okouzlenost virtuální realitou, vývoj spěje rychle kupředu. Výzkum hardwaru (především čipů CPU a GPU) a herních aplikací se často prolíná, a tak nástrahy představené jednou stranou, nemusí být tou druhou chápány jako překážka. Vzniká výzva, při jejímž překonání se společnost dostává do konkurenční výhody. Jako ilustrační příklad lze uvést dlouhodobě vznikající poptávku po systému, který by dokázal v reálném čase provádět fyzikální simulace. S dobrým řešením přišla firma NovodeX AG, která byla ve svém provedení natolik úspěšná, že byl její produkt draze odkoupen a začala se pro něj vyvíjet hardwarová akcelerace pro urychlení výpočtů s názvem PhysX PPU (physics processing unit). V dnešní době je soubor těchto technologií již v rukou jednoho z předních světových výrobců grafických čipů společnosti NVIDIA. Od zakoupení netrvalo dlouho a společnost představila řadu grafických karet s názvem GeForce, které umožňují hardwarovou akceleraci pro systém PhysX. O tom, zda byla investice výhodná, lze jen polemizovat. Přesvědčit čtenáře může ale výčet obstojného množství aplikací a herních titulů, které podporují, nebo dokonce vyžadují tuto technologii. [2]

Rozmanitost 3D aplikací samozřejmě nekončí pouze u zábavního průmyslu a stále více projektů nachází uplatnění i v jiných odvětvích. Uživatel si může dopřát prohlídku interiéru rozsáhlých objektů, nebo částečně doplnit svou výuku ve 3D simulátorech. Tématika diplomové práce spadá právě do oboru aplikací, které simulují vybrané aspekty dopravního provozu za účelem výuky.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem diplomové práce s názvem „Simulátor dopravních situací pro výuku řízení“ (dále jen: diplomová práce) je vývoj softwaru pro podporu při výuce řízení osobních vozidel. Navržena bude 3D aplikace zpracovávaná v reálném čase. Je považováno za nezbytné, aby byla interaktivní, reagovala na vstup uživatele a poskytla mu odpovídající grafický a zvukový výstup. Konečný produkt umožní uživateli řídit osobní automobil ve vytvořeném prostředí. Poskytnuta bude zároveň dostatečná zpětná vazba, která analyzuje jeho dosavadní výkon a dodržování dopravních předpisů. Aplikace bude vyvíjena na frameworku Unreal Engine 4 od společnosti Epic Games Inc.

Dílčí cíle práce byly určeny následovně v souladu s možnostmi výše zmíněného frameworku.

1. Budou připraveny grafické a zvukové prostředky, které jsou nezbytné pro zobrazení objektů reálného světa v připravovaném prostředí. Důraz bude kladen na jejich realistické vzezření. V případě využití prostředků třetích stran nebude porušeno licenční ujednání.
2. Narýsován bude obecný model krajiny, který bude následně obsazen připravenými grafickými prostředky. Mezi tyto úkony patří i aplikace materiálů na povrch a tvorba silniční sítě.
3. Navržen bude logický model pro simulované osobní vozidlo, který napodobí reálné jízdní vlastnosti dopravního prostředku.
4. Vytvořený simulátor bude vyžadovat zpětnou vazbu v oblasti kontroly dopravních předpisů. Z uvedeného důvodu bude připraven systém, který poskytne uživateli dostatečnou zpětnou vazbu.
5. Bude vytvořeno uživatelské rozhraní, které je nezbytné pro specifikaci množiny podporovaných vstupů a patřičných výstupů. Cílem je ulehčit uživateli ovládání a pohyb napříč aplikací a zároveň zajistit vykreslení důležitých informací.
6. S ohledem na různorodé výkonnostní možnosti cílových zařízení bude zohledněna škálovatelnost aplikace.
7. Aplikace bude umožňovat rozšiřitelnost, nastane-li možnost dalšího vývoje.

2.2 Metodika

Metodika diplomové práce reflektuje obsah řešené problematiky pro tvorbu 3D aplikace v reálném čase se zaměřením na simulaci osobního vozidla a dopravních situací v prostředí Unreal Engine 4. Provedena bude analýza informačních zdrojů v oboru vývoje 3D aplikací. Ze zjištěných poznatků a obecných předpokladů budou následně metodou dedukce zvoleny postupy, které budou využity při tvorbě aplikace.

Některé novější technologie a metody doposud postrádají kvalitativně uspokojující informační zdroje splňující požadovanou akademickou úroveň. Z tohoto důvodu budou využity informace a rady z odborných internetových diskuzí. Takto nabyté znalosti budou podrobeny metodě vlastního experimentování. Cílem empirického experimentování bude najít řešení, které je kompromisem mezi optimalizací výkonu a kvalitou aplikace. Jednotlivé iterace testování budou zkoušet různé konfigurace použitých prostředků a logických modelů. Důvody pro výběr konkrétního řešení budou uvedeny v jednotlivých kapitolách.

Klíčové oblasti vyvíjeného softwaru, na které budou použity metody experimentování a teoretické metody poznání:

1. Tvorba a vykreslování grafických prostředků
2. Engine pro simulaci fyziky
3. Objektově orientované programování v jazycích C++ a BluePrint
4. Práce v prostředí Unreal Engine 4 - Editor
5. Programové rozhraní Unreal Engine 4

Tvorba 3D prostředků proběhne v softwarech Blender a 3ds Max. Experimentováno bude především se složitostí polygonové sítě a jejím dopadem na výpočetní náročnost scény. 2D grafické textury, normálové a UV mapy budou připravovány v kombinaci výše zmíněných nástrojů s aplikacemi Photoshop, Gimp a UE4 editor. Snahou bude dosáhnout dostatečného detailu při zachování malého rozlišení rastrových dat. Závěrem práce bude shrnutí praktické části. Výsledky budou diskutovány a dojde ke zhodnocení výstupů vytvořeného softwaru a jejich průnik s vytyčeným záměrem. Shrnuty budou nalezené nedostatky a aplikace bude vystavena výkonnostnímu testu na strojích s různou konfigurací. Stroje budou vybírány na základě dostupnosti. Navrženy budou rovněž potenciální rozšíření pro produkt v případě jeho budoucího vývoje.

3 Přehled řešené problematiky

Kapitola přehled řešené problematiky se bude zabývat teoretickými východisky, která jsou nezbytná pro uskutečnění vlastní práce. Simulátor dopravních situací pro výuku řízení bude realizován v prostředí Unreal Engine 4 od společnosti Epic Games, Inc. Za účelem lepšího porozumění bude sestaven přehled důležitých oblastí pro tvorbu 3D aplikací, který bude výstupem studia odborných informačních zdrojů. V rámci diplomové práce se lze setkat s cizojazyčnými termíny, a to z důvodů nevhodného či neexistujícího překladu do českého jazyka. Tyto termíny budou při jejich prvním výskytu náležitě vysvětleny.

3.1 Herní engine

Herní engine[1] je nástroj, který ulehčuje práci s vývojem aplikací herního typu. Pro vývojáře to znamená, že se nemusí věnovat programování nízko-úrovňových funkcí a svou pozornost mnohou věnovat hlavní myšlence tvorby. Celý proces vývoje 3D aplikace je velmi zdrojově náročný, a proto je představa znovu-využitelného middlewaru[2] z časového i ekonomického hlediska lákavá. Právě znovu-využitelnost je přednost herních enginů a bývá realizována hned na několika úrovních. Nejdříve se jedná o samotný herní engine, který je souborem modulů, které jsou integrované do vývojového prostředí. Tyto individuální moduly jsou v mnoha případech postaveny na jiných znovu-použitelných softwarech. Příkladem mohou být komponenty pro grafické vykreslování scén, které jsou běžně realizovány obecně použitelnými grafickými knihovnami OpenGL a DirectX, nebo fyzikální engine zastoupený rozhraním PhysX nebo Bullet. Uvedené API[3] jsou využity hned v několika soudobých řešeních. [1] [2] [3]

Herní enginy jsou koncipovány tak, aby byly schopny pokrýt širokou škálu zařízení. Podporováno bývá vše od osobních počítačů přes herní konzole až po mobilní zařízení. Podstatná část pozornosti je věnována i zařízením pro virtuální realitu, jako jsou Oculus Rift, Samsung Gear VR a Steam VR. [1] Abstrakce od platformy je zajištěna modulární architekturou, která umožňuje zaměňovat komponenty dle potřeb dané platformy. Některé

[1] Engine je jádrem počítačové aplikace, která realizuje klíčové funkce.

[2] Middleware je specializovaný software, který slouží jako prostředník jiným aplikací.

[3] API (Application Programming Interface) je rozhraní, které slouží pro tvorbu aplikací. Jedná se o soubor protokolů, tříd, metod atd. s účelem poskytnout vývojáři rozšiřitelnou funkcionalitu.

enginy dokonce obsahují křížové překladače, které dovolují generování kódu spustitelného na jiné platformě. Například Unreal Engine 4 dokáže vydat produkt na platformy: iOS, Windows 32/64 bit, Linux, Android, AppleTv, HTML5 [1]

Snahou herního enginu je dosáhnout co nejvíce rozmanitého prostředí vhodného pro vývoj 3D aplikací. Specifikace, co by měl takový software obsahovat, nejsou nikde upřesněné a bývá čistě na zpracovateli, jakou funkcionalitu do enginu zahrne. Ve většině vývojových prostředí lze pozorovat modul pro grafické vykreslení scény, engine pro simulování fyzikálních jevů, knihovny pro správu zvuku, síťovou podporu, modul pro umělou inteligenci, správu obsahových prostředků a graf scény. [1] [3] Za obsahové prostředky jsou považovány 3D modely, textury, audio a video záznamy apod. Samotná výroba prostředků bývá obvykle přenechána aplikacím, které jsou pro daný účel specializované. Tyto aplikace často využívají stejné výstupní formátování pro soubory[4] a herní engine je dokáže rozpoznat a umožňuje jejich nahrání. [1] [2] [3]

Aplikační rozhraní pro Unreal Engine 4 je styčnou plochou mezi vývojářem a produktem, a proto je nezbytné zaopatřit jej patřičnou dokumentací. Epic Games Inc. spravuje soubor dokumentace v online režimu[5]. Snahou vydavatele je udržet dokumentaci kompletní a aktuální, ale nijak se k tomu nezaručuje. Občasné nedostatky se mohou projevovat nedostatečným nebo zastaralým popisem řešené problematiky. [3]

[4] Mezi používané formáty patří například FBX pro 3D prostředky, waw a MP3 pro zvukové prostředky, BMP, TIF a Targa pro rastrovou grafiku. [2]

[5] Dokumentace pro Unreal Engine může být nalezena online zde: https://docs.unrealengine.com/latest/INT/API/

3.2 Grafické vykreslování 3D scén

Grafické vykreslování je proces pro převedení aplikačních grafických dat do rastrové podoby, která je určena pro projekci na uživatelském monitoru. Operace vykreslování lze realizovat více způsoby.

Jedním ze způsobu, kterým lze dosáhnout vykreslení grafického výstupu na monitor, je čistě softwarový přístup, který k výpočtům využívá CPU. Softwarový přístup uchovává celou 3D scénu pro vykreslení v paměti a pracuje metodou pixel po pixelu. Uchování celé 3D scény umožňuje softwarovému vykreslování flexibilnější řešení požadavků, protože přímý přístup do paměti dovoluje lépe získat požadované objekty. [4]

Druhým způsobem je hardwarová akcelerace GPU. Vykreslování pomocí hardwarové akcelerace používá odlišný přístup. Hardwarový přístup si neudržuje v paměti 3D scénu, ale uchovává si všechny vstupní pixely. Vykreslování je prováděno na jednom trojúhelníku v daný okamžik, a ten je po zpracování zanesen do framebufferu[6]. Přístup hardwarové akcelerace GPU má několik úskalí. Neexistence 3D scény v paměti neumožňuje zohlednit ostatní přítomné objekty, které mohou mít vliv na zpracovávaný trojúhelník (například odrazy světla, stíny atd.). Nicméně pro výše uvedené limitace jsou vytvořeny náhradní řešení, které je částečně obchází. Ale i přes uvedené nedostatky je pro vykreslování náročných 3D scén považována hardwarová akcelerace za vhodnější. Hlavním důvodem je velké množství dat, které je potřeba v herním prostředí zpracovat pro jeden snímek. [4] Vzhledem k předmětu diplomové práce a jejího silného propojení s vývojem video her v 3D aplikacích bude zaměřena pozornost právě na druhý z výše uvedených přístupů.

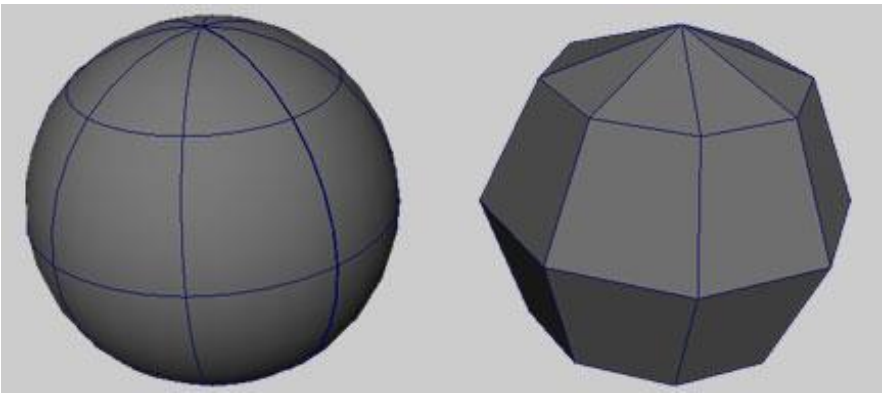
[6] Framebuffer je část paměti, která uchovává snímek v podobě rastrových dat.

3.2.1 Modely v 3D grafickém prostředí

Drátové modely jsou důležitým prvkem pro vytvoření grafické 3D scény. Jsou nezbytné pro tvorbu animací, her a vědeckých vizualizací. [5] V počítačové grafice se využívají tři základní přístupy pro vytvoření tří-dimenzionálních geometrie. Modeluje se prostřednictvím polygonů, křivkového přístupu NURBS a dělení povrchů (v angl.: subdivision surfaces). [6] [5]

Polygonové modelování

Polygonové modelování je pravděpodobně nejrozšířenější způsob vytváření 3D obsahu pro různé aplikace. Polygony (mnohoúhelníky) jsou tří a vícestranné útvary s vrcholy, které jsou definované v trojrozměrném prostoru. Na rozdíl od geometrického mnohoúhelníku může dojít k tomu, že vrcholy neobývají stejnou rovinu, a proto jsou často mnohoúhelníky převáděny na trojúhelníky (triangulace), u kterých je to zaručeno[7]. Každý polygon je složen z vrcholů, hran a jedné stěny. Celý model je sestaven z mnoha polygonů. Sousedící polygony sdílí vrcholy a hrany. [6] Uspořádání jednotlivých polygonů se snaží aproximovat svůj vzor. Nevýhodou polygonového modelování jsou ostré přechody mezi jednotlivými stěnami. [5] Tento nedostatek je možné oslovit později v průběhu vykreslování výsledné scény[8].



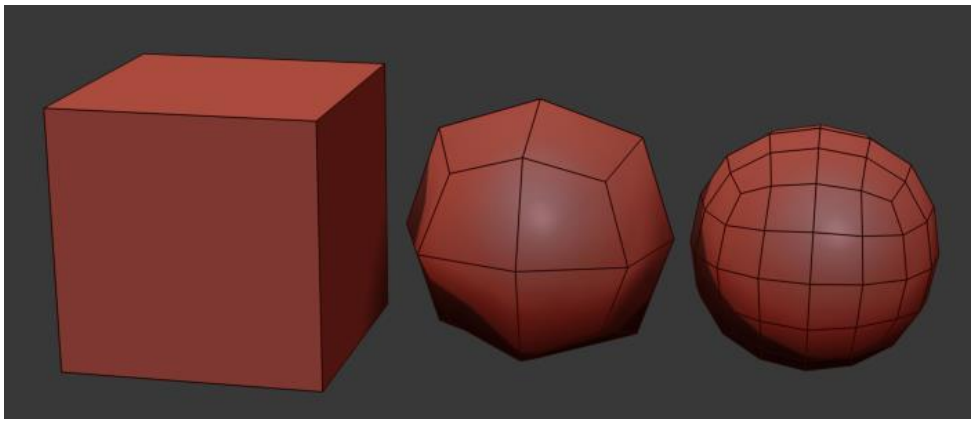
Obrázek 1: Objekt vytvořený za použití NURBS (vlevo), objekt vytvořený běžnými polygony (vpravo). Zdroj: SW Autodesk Maya.

[7] Při modelování polygonů může dojít k situaci, kdy vrcholy nejsou koplanární.

[8] Plynulých přechodů nelze v polygonálním modelování dosáhnout. Organický tvar se realizuje stínováním.

Non Uniform Rational Basis Spline

NURBS(Non-Uniform Rational Basis Spline) je matematický model pro generování zakřivených povrchů. Předností tohoto způsobu modelování je přirozeně organický vzhled díky hladkým křivkám a absenci ostrých hran. Samotné modelování za pomocí NURBS je považováno za intuitivní a chování povrchů za předvídatelné. Tvar povrchu definují bézierovy křivky, jejichž zakřivení je ovládáno kontrolními body. [6]



Obrázek 2:Modelování metodou subdivision surfaces s využitím aproximačního modelu Catmull-Clark. Výchozí polygon tvaru krychle (vlevo). První iterace dělení povrchu (uprostřed). Druhá iterace dělení povrchu (vpravo). Zdroj: SW Autodesk 3Ds Max.

Dělení povrchů

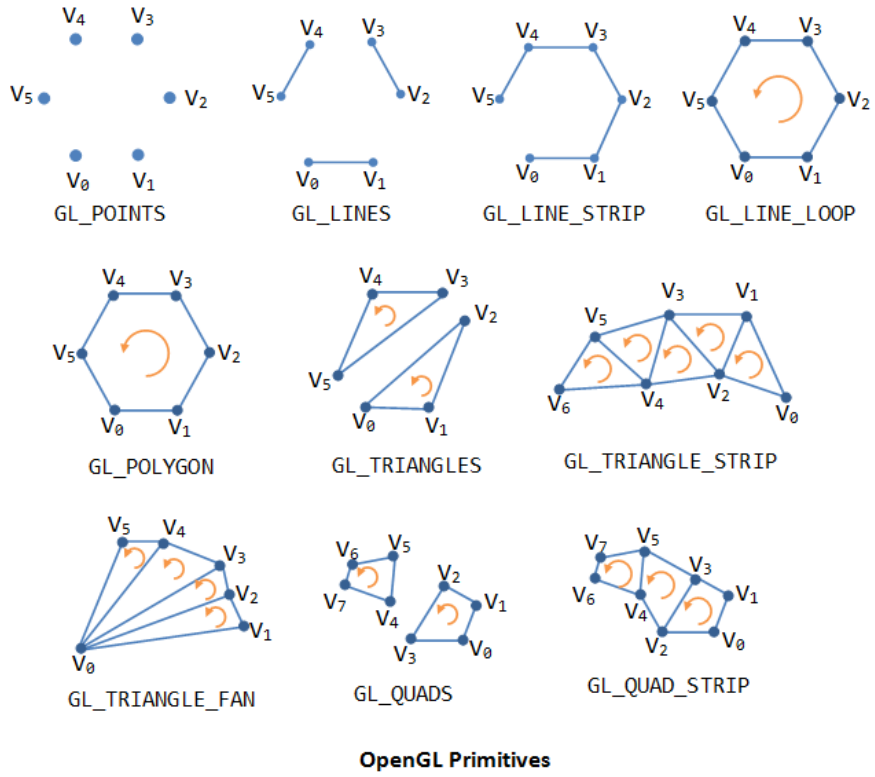
Dělení povrchů (subdivision surfaces) je metoda, která upravuje původní topologii polygonového modelu. Pro napodobení vzorového tvaru jsou stěny polygonu děleny a pozice nových vrcholů jsou interpolovány. Způsob dělení a pozice nových vrcholů jsou dány výběrem aproximačního modelu. Generování výsledné geometrie se provádí rekurzivně a s růstem počtu iterací obecně roste kvalita produkovaného modelu. [6] Je ale nutno mít na paměti, že model s větším detailem je náročnější na vykreslení.

3.2.2 Grafické programové prostředí

Grafická programová prostředí (dále jen grafická API) nebo grafické knihovny jsou určeny pro ulehčení práce při generování grafických výstupů. Již delší dobu jsou nedílnou součástí vývoje různorodých aplikací, které vyžadují vykreslení grafických objektů na monitor. Za pomocí těchto knihoven odpadá vývojářům práce s vytvářením vlastních a mnohdy neoptimalizovaných funkcí pro sestavování konečného výstupu.

Grafická primitiva

Za použití grafických API lze běžně vykreslovat pouze základní grafická primitiva, která se mohou v závislosti na zvolené knihovně mírně lišit. Například pro standard OpenGL jsou definována následující grafická primitiva: izolovaný bod, úsečka, posloupnost úseček, cyklická posloupnost úseček, konvexní polygon, trojúhelník, pás trojúhelníků, trs (vějíř) trojúhelníků, čtyřúhelník, pás čtyřúhelníků. [7] [8]



Obrázek 3: Grafická primitiva pro OpenGL. [7]

Knihovna Direct3D od společnosti Microsoft využívá všechna uvedená grafická primitiva s výjimkou polygonů s více než třemi vrcholy. Takové polygony jsou převedeny na trojúhelníky (triangulace) a je tak u nich zajištěn planární povrch. Polygony s více než třemi vrcholy mohou mít povrch zakřivený a jejich vykreslování je neefektivní. [9]

Vykreslování jednotlivých primitiv je realizováno příkazy s použitím příslušného počtu parametrů. Za použití funkcí lze výše uvedená primitiva skládat do složitějších útvarů, ze kterých jsou eventuálně vytvořeny celé scény. [7]

Rastrová data

V terminologii grafických knihoven jsou definovány pojmy bitmapa a pixmapa. Pro oba platí, že patří do kategorie rastrových dat. Jinými slovy, jedná se o informace uložené ve 2D mřížkách. Často dochází k záměně pojmů. [7]

Například pro OpenGL jsou termíny bitmapa a pixmapa definovány následovně: Bitmapa je pole často o tvaru obdélníku, kde každý prvek může nabývat hodnot 0 nebo 1. V bitmapě je tedy vyhrazen pro každý bod právě jeden bit. Takto vytvořené rastry plní úlohu různých vykreslovacích masek. V místě, kde je prvek nastaven na 1, dojde k vykreslení pixelu a naopak. [7]

Pixmapa je také obdélníkové pole, ale každý prvek má větší bitovou hloubku. V tomto smyslu je lze považovat za rastrový obrázek. Bitová hloubka bývá často 8 nebo 32 bitů o třech až čtyřech kanálech. [7]

Rozlišení těchto termínů nemusí být dodrženo pro každou grafickou knihovnu. Například dokumentace ke knihovně DirectX uvádí bitmapy, tak jak jsou pro OpenGL definovány pixmapy. [9]

Vykreslovací řetězec

Vykreslovací řetězec není nic jiného než posloupnost jednotlivých stádií vykreslovacího procesu scény. Jednotlivé fáze se mohou lišit v závislosti na použité grafické API a její verzi. S vývojem knihoven jsou dodávány neustále nové funkce. Jako příklad je možné uvést modul teselace zodpovědný za přidání geometrických detailů polygonům, který byl do knihovny Direct3D přidán až ve verzi 11. [9]

Moduly jsou samostatné jednotky pracující individuálně a jejich operace je řízena daty (data-flow architektura) a nepředávají se řídící instrukce. Výstupní data jsou vstupem pro následující modul. [7]

Při zjednodušení vstupují do vykreslovacího řetězce dva typy dat. Jedním jsou geometrická data v podobě již zmíněných primitiv a data rastrová. Proces byl navržen tak, aby bylo možné paralelně zpracovávat oba druhy dat odlišnými jednotkami. Komplexní operace je snadné dekomponovat do jednodušších kroků. Tok dat je často proudový (data jsou předávána

mezi jednotlivými moduly), to je úsporným opatřením, neboť sběrnice mezi grafickým čipem a pamětí není zbytečně zatěžována. [7] [9] Důležitým poznatkem je, že data nemusí nutně projít celým řetězcem jen jednou. Často dochází k přepracování již použitých dat. Tato data jsou obvykle po první iteraci uložena do některého z bufferů a následně jsou načtena počátkem některého z logických cyklů řetězce. Samozřejmě musí být dodržen typ dat. Nelze načíst data rastrová do geometrické jednotky a naopak.

Obrázek 4 ilustruje příklad vykreslovacího řetězce společnosti Microsoft pro Direct3D 10. Každá ze zobrazených fází je konfigurovatelná v grafickém API. Zaoblené obdélníky reprezentují programovatelné moduly, jež lze obsluhovat pomocí programovacího jazyka HLSL[9]. Klasické (nezaoblené) obdélníky jsou stádia s fixními funkcemi. Cílem tohoto návrhu je dosáhnou maximální flexibility pro vývoj aplikací. [9] Následuje popis jednotlivých fází řetězce.

Input-assembler

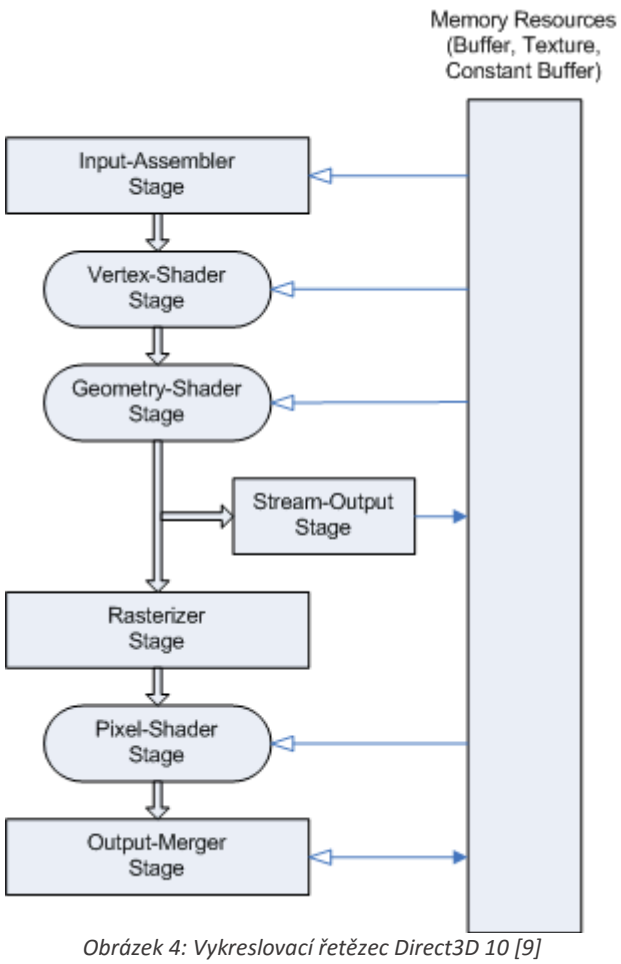
Input-Assembler je první z fází a je zodpovědný za shromáždění dat a jejich sestavení do podoby grafických primitiv pro použití v dalších stádiích vykreslovacího řetězce. [9]

Vertex Shader

Úkolem vertex shaderu je transformace vrcholů. Operace jsou prováděny právě nad jedním vrcholem v daný okamžik a upravovány jsou některé z atributů: pozice, osvětlení atd. Výstupem je právě jeden transformovaný vrchol. [9]

Geometry Shader

Geometry shader zpracovává celá grafická primitiva. Mimo informací o



Obrázek 4: Vykreslovací řetězec Direct3D 10 [9]

[9] HLSL (High Level Shading Language) Je programovací jazyk odvozený od jazyka C. Je určen pro programování shaderů ve vykreslovacím řetězci. Vyvíjen je od verze DirectX 9. [9]

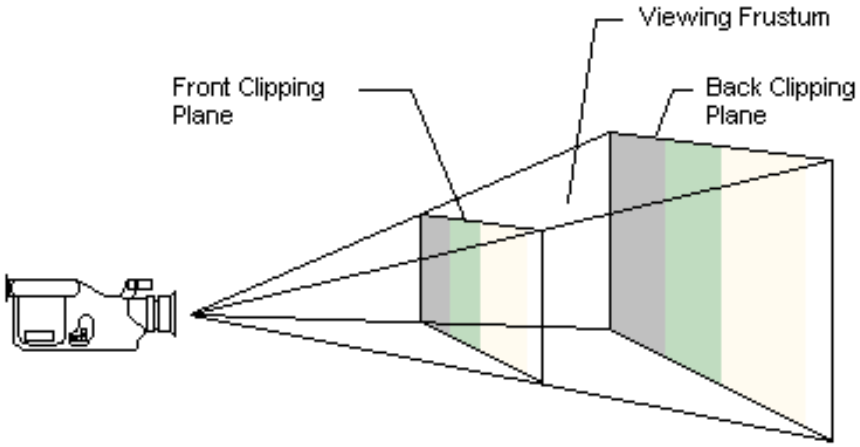
vlastních vrcholech si tato primitiva uchovávají informaci o vrcholech hranou přilehlých primitiv. Výsledkem modulu může být úplné odstranění celého primitiva, nebo vytvoření jednoho a více nových. [9]

Stream output stage

V této fázi vystupují data ve své současné podobě do paměťového bufferu. Tato data ještě neprošla rasterizerem a jsou stále v geometrické podobě, a proto je lze načíst zpět do vykreslovacího řetězce v některých jeho předešlých fázích. [9]

Rasterizer

Rasterizer převádí vektorové informace na rastrové složené z pixelů. Data pixelů jsou interpolována napříč primitivem. Úkolem rasterizeru je i ořezávání primitiv do vykreslovacího prostoru (čtyřboký komolý jehlan). Vykreslovací prostor je definován přední a zadní ořezávací plochou a zorným úhlem, kterým snímá kamera scénu (Field of View). Část primitiva mimo vykreslovací prostor je odstraněna a v místě styku jsou vytvořeny nové vrcholy. [9]



Obrázek 5: Vykreslovací prostor je čtyřboký komolý jehlan (Viewing Frustum) . [9]

Pixel Shader

Pixel shader je invokován pro každý pixel resterizerem. V běžných případech je pixel zpracováván jen jednou, výjimkou je multisampling u textur, kde je proveden odpovídající počet iterací. Právě v pixel shaderu jsou implementovány komplexní operace včetně per-pixel osvětlení a post-processingu. Do zpracování jsou zahrnuta texturová data, interpolované hodnoty vrcholů a konstantní proměnné. Výstupem je barevná hodnota obvykle v 8, 32-bitech o čtyřech složkách RGBa[10]. Pokud je pixel v průběhu shaderu odstraněn, není barva na výstupu uvedena. [9]

[10] RGBa je rozšířením barevného aditivního modelu RGB, kde *a* znamená kanál alfa a určuje průsvitnost.

Framebuffer

Výsledná data, která prošla vykreslovacím řetězcem, jsou ukládána do takzvaného framebufferu. Implementace framebufferu je závislá na použité knihovně. Pro OpenGL je framebuffer realizován dílčími buffery. Framebuffer je pamětí uchovávající rastrová data, kde je každý prvek pole nazýván fragment. [7] [10]

Color buffer

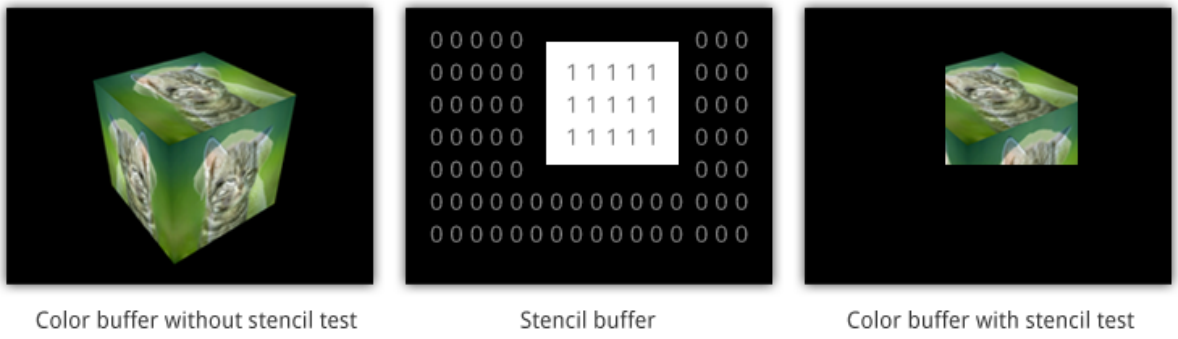
Barvový buffer (color buffer) ukládá barevnou informaci o scéně pro jednotlivé fragmenty. Tyto informace jsou běžně uloženy v informaci RGB. V případě použití některých metod, jako jsou mnohonásobný buffering nebo úprava pro stereoskopický pohled, je barvových bufferů vytvořeno hned několik. [7]

Depth buffer

Hloubkový buffer (depth buffer, Z-buffer) ukládá informaci o vzdálenosti fragmentu od projekční roviny. Při zpracování fragmentu dojde ke srovnání s již zaznamenanými fragmenty. Tento proces se nazývá test hloubky (depth testing) a je volně programovatelný. Obvykle dojde k zápisu fragmentu s menší vzdáleností od projekční roviny a odstranění vzdálenějšího, ale lze dosahovat jiných i opačných efektů. Ve většině případů je použit pouze jeden hloubkový buffer. [7] [10]

Stencil buffer

Paměť šablony (stencil buffer) slouží k ořezávání fragmentů z vykreslované scény. Funkčně je velmi podobný hloubkovému bufferu s tím rozdílem, že vývojář má větší kontrolu nad vykreslovanými fragmenty. Pro určení hodnoty je využit stencil test, který je volně programovatelný. [10] Paměť šablony nachází uplatnění pro tvorbu uživatelských rozhraní nebo při tvorbě speciálních efektů. [7]



Obrázek 6: Využití Paměti pro šablony v OpenGL. [10]

Accumulation buffer

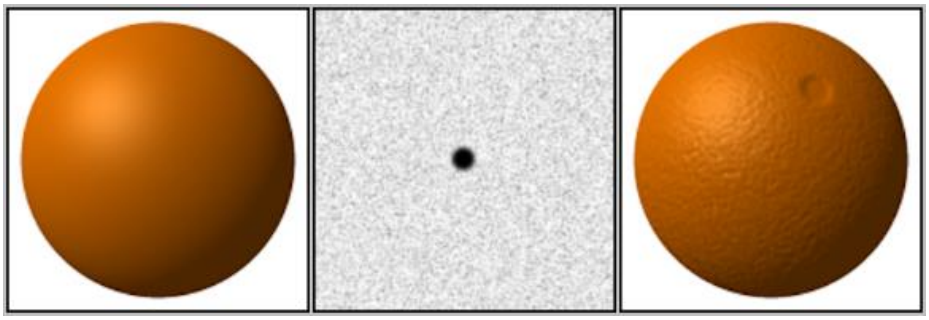
Akumulační buffer (accumulation buffer) slouží k uchování rastrových obrázků. Pro každý pixel je uchována informace RGBa, tím se neliší od barvového bufferu. Akumulace spočívá v možnosti sčítat (akumulovat) jednotlivé vykreslované scény a vytvářet některé speciální efekty post-processingu. V OpenGL se využívá například k tvorbě efektů, jako jsou: motion-blur, depth-of-field a antialiasing. [7]

3.2.3 Techniky pro úpravu povrchu

Účelem úpravy povrchu je přinést dodatečný detail pro povrch modelu, ať už se jedná o jeden polygon nebo celek. Tohoto efektu je možné dosáhnout hned několika způsoby, jejichž přístup se značně odlišuje.

Bump mapování

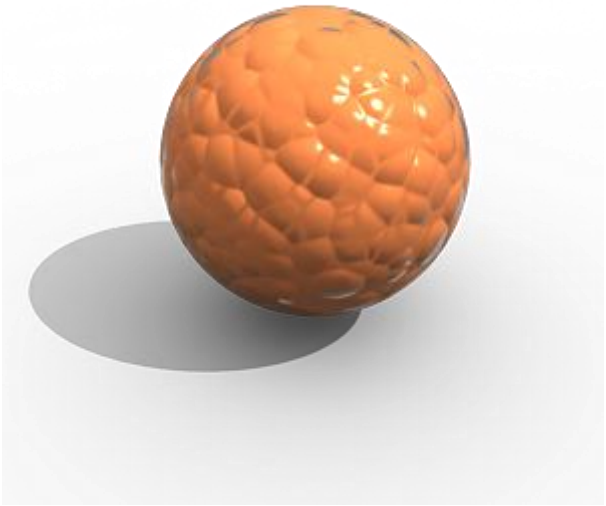
Bump mapování bylo používáno na počátcích 3D grafiky. Představeno bylo v roce 1978 Jamesem Blinnem. Zvrásněný efekt povrchu je falešný, protože metoda nezasahuje do topologie síťového polygonálního modelu ani nemanipuluje s jednotlivými vrcholy. [5] Výsledného dojmu je dosaženo pomocí úprav výpočtů osvětlení. [11]



Obrázek 7: Bump mapování. Zleva: 1. Objekt s difuzní mapou. 2. Bump mapa. 3. Objekt s difuzní a bump mapou. Zdroj: en.wikipedia.org

Bump mapy jsou běžně jednokanálové rastry s 8 bitovou barevnou informací. Jedná se tedy o stupnici šedi o 256 odstínech. Každý z odstínů reprezentuje výstup, nebo prohlubeň zvrásnění. Čím jsou pixely světlejší (blíže k bílé), tím více vystupují z povrchu a opačně, pixely přibližující se černé barvě tvoří prohlubně. Důkaz pouhé iluze zvrásnění lze spatřit v obrysu samotného objektu a jeho stínu. Tento obrys bude odpovídat geometrii původního objektu.

Bump mapy jsou principiálně velmi jednoduché a lze je vytvořit v jakémkoliv nástroji pro 2D grafiku. Bohužel, mapování tímto způsobem přináší hned několik nevýhod. Při nesprávném úhlu pohledu na povrch, především při profilovém úhlu, dojde k zániku vrásnění. [11] Bump mapy s nízkou barevnou informací mohou tvořit nežádané artefakty v podobě nepřirozeně „zubatého“ povrchu. Z těchto důvodů se dle potřeb používají bump mapy s vyšší barevnou informací 16, 32 bitů, které umožňují plynulejší přechody. Další nevýhodou metody bump mapování je neexistence sebe-zastínění. Sebe-zastínění je jev, který umožňuje vrhat stíny výstupů na vlastní povrch.



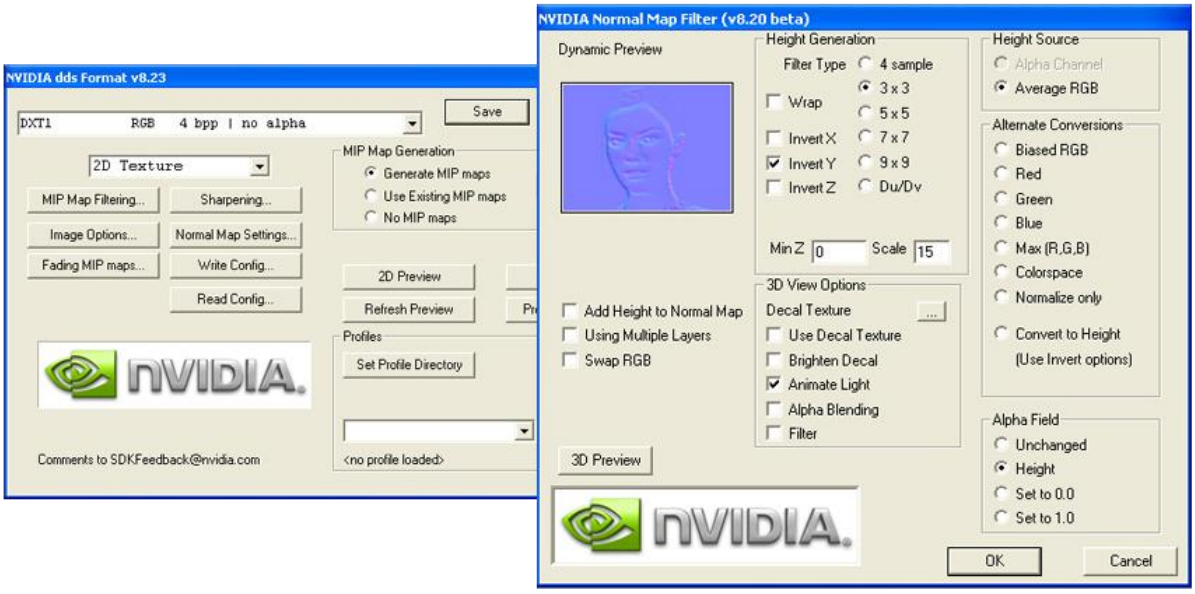
Obrázek 8: Hladký obrys a stín odpovídá geometrii objektu. Zdroj: Chaosgroup.com

Normálové mapování

Normálové mapování je odlišnou technikou bump mapování a někdy je nazýváno *Dot3 bump mapování*. Vzhledem k tomu, že se jedná stejný princip jako u bump map, nevznikají na modelu dodatečné geometrické detaily a konečný efekt je pouze iluzí. Normálové mapy jsou rastry s třemi kanály RGB. Hodnota jednotlivých kanálů odpovídá prostorovým souřadnicím X, Y, Z. Kombinace informací říká přesný směr normály z každého bodu daného povrchu. Touto informací je určeno stínování pro daný bod.

Na rozdíl od obyčejných bump map je normálová mapa obtížná na tvorbu. Jen stěží jí lze vytvořit manuálně ve 2D grafické aplikaci a mnohdy jsou využívány postupy „pečení“ textur (v angl.: texture baking). Jedná se o postup, kdy je algoritmem ve 3D aplikaci převedena informace ze složitějšího modelu na bitmapu, která je následně aplikována na zjednodušený model.

Jiným způsobem jsou generátory z 2D bitmap, které vytvářejí pouze jednodušší verzi normálové mapy. Nevýhodou tohoto přístupu je, že vyžadují velmi kvalitní vstupní rastry. Pokud textury nesplňují požadavky generátoru, může s velkou pravděpodobností dojít k tvorbě nesprávné normálové mapy.



Obrázek 9: Doplněk Adobe Photoshop Normal Map Filter od společnosti NVIDIA. Zdroj: developer.nvidia.com

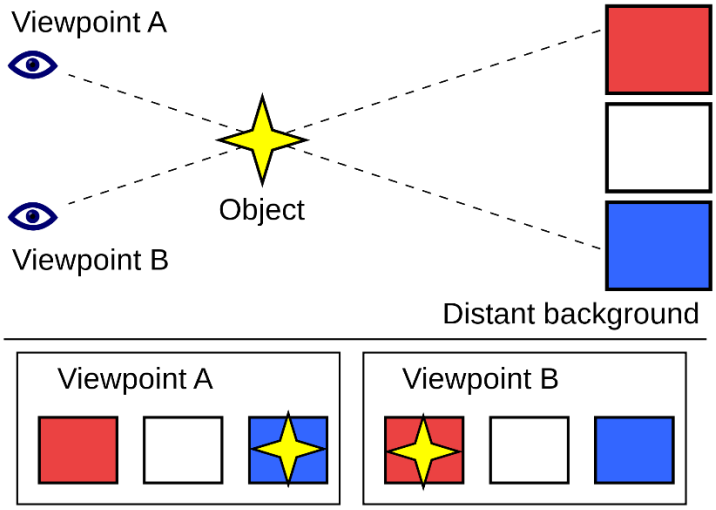
Jako příklad lze uvést doplněk komerčního softwaru Adobe Photoshop od společnosti NVIDIA s návem Normal Map Filter. V případě NVIDIA je vyžadována bitmapa v odstínech

šedi s výškovou informací. [12] Druhým generátorem je online aplikace NormalMapOnline, která přijímá stejné vstupní rastry jako NVIDIA Normal Map Filter. [13] V obou příkladech si lze prohlédnout výsledný efekt v reálném čase a přizpůsobit jej dle potřeb.

Parallax Oclusion mapování

Parallax occlusion mapování (dále jen POM) je technika, která dále rozšiřuje bump mapování. Jedná se tedy o další metodu, která vytváří iluzi zvrásněného povrchu. Oproti původním bump a normálovému mapování přidává silnější efekt hloubky výslednému povrchu. Stejně jako u bump mapování pracuje operace POM výhradně s pixel shaderem a neupravuje geometrii výchozího polygonu. [14]

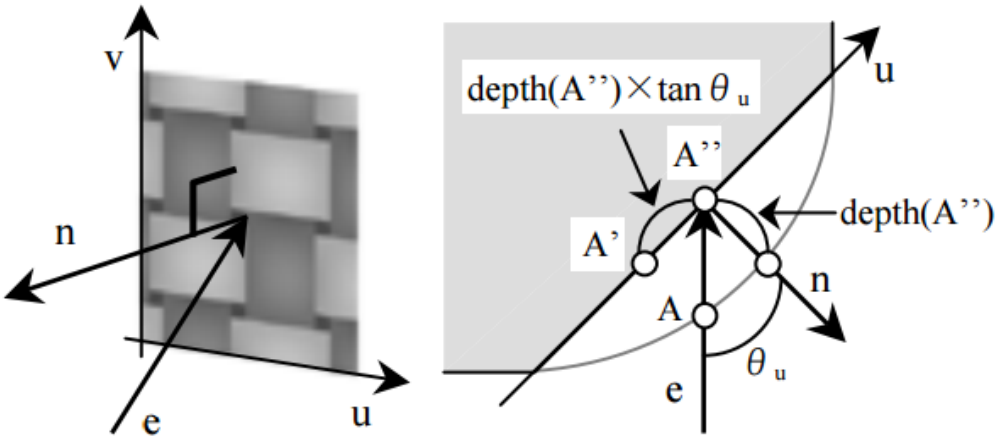
POM je kompromis přístupů bump a displacement mapování. Při představení metody POM v roce 2001 (Kaneko a spol.) byly pro její uplatnění vzneseny dva hlavní argumenty. Prvním je nekonzistence vrásnění povrchu bump mapováním při různých úhlech sevřených vektorem pozorovatele a pozorované plochy, kde při nízkých úhlech zcela zaniká detail vrásnění a silně se projevuje reálná geometrie povrchu. Druhý argument říká, že klasické mapování textur neumožňuje zdánlivý posun pozorovaného bodu vzhledem k pozadí při změně úhlu pohledu, jde o takzvaný efekt paralaxy.



Obrázek 10: Efekt paralaxy. Zdroj en.wikipedia.org

POM získává informace z výškové mapy (v angl.: height-map), která přiřazuje hodnoty jednotlivým souřadnicím u, v pro každý polygon objektu. Pro zachycení vyžadovaných efektů, jmenovitě jde o vrásnění a paralaxu, je zapotřebí provést transformaci souřadnic dle specifických parametrů. Za tyto parametry jsou považovány informace získané z výškové mapy pro daný bod a úhel pohledu pozorovatele.

Obrázek 11 znázorňuje výpočet zmíněných korekcí pro bod A. Tak, aby byl bod A vnímán pozorovatelem na své určené pozici na křivce, je nutné transformovat souřadnice bodu A' na A''. [15]



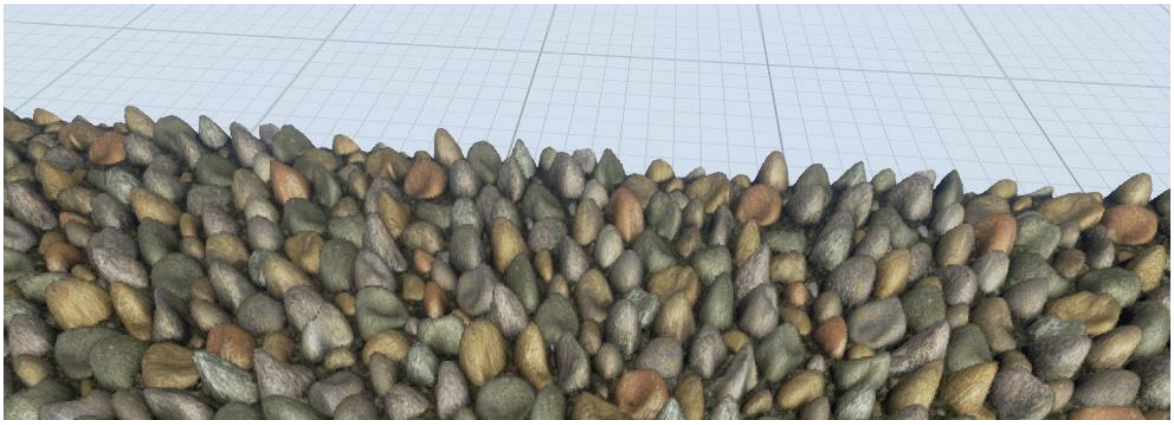
Obrázek 11: Korekce souřadnic textury metodou POM [15]

Transformace souřadnice bodu z textury je provedena dle následujícího vzorce:

$$u' = u + \tan\theta_u \cdot depth(u, v)$$

Směrový vektor pozorovatele e a normála polygonu n svírají úhel θ. Funkce hloubky $depth(u, v)$ je aplikována na výškovou mapu, která je přiřazená polygonu a její hodnota odpovídá výškové informaci v daném bodě. Pro souřadnice v platí analogicky to samé co pro u.

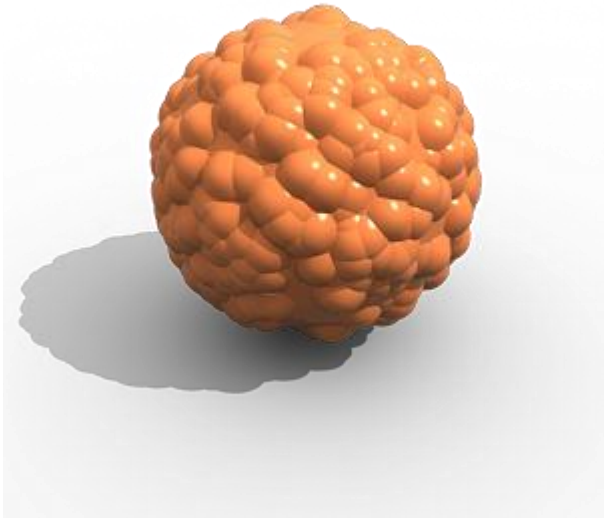
Obrázek 12 znázorňuje jediný polygon a již na první pohled jsou znatelné hlavní přednosti metody POM: sebe-zastínění, zvrásnění povrchu i obrysu objektu. Na rozdíl od značné výpočetní náročnosti displacement mapování je v praxi POM reálně aplikovatelné a může být využito ve větším měřítku pro konečné vykreslení rozsáhlejších 3D scén. [15]



Obrázek 12: Materiál vytvořený v UE4 s využitím POM. Zdroj: forums.unrealengine.com

Teselace a displacement mapování

Pro vytváření dodatečných detailů na povrchu objektu je metoda teselace nejvěrohodnější. Na rozdíl od výše zmíněných způsobů je detail na povrch opravdu přidán. Dochází k vytvoření nových vrcholů a hran pro každý polygon. Detailní geometrie přináší výhody a nevyžaduje řešení problémů, jako jsou: sebe-zastínění, hladké obrysy, dojem plochého povrchu při nízkých zorných úhlech. Proces teselace bere informace z takzvané displacement mapy, kterou lze vytvořit metodou „pečení“ z vysoce detailního modelu, nebo manuálně jako bump mapy. Displacement mapa je rastrovou mřížkou s 8, 16 nebo 32 bitovou informací pro každý prvek pole. Zapotřebí je pouze jedna barevná složka (stupnice šedi), která jako v případě bump mapy určuje odsazení vrcholů v geometrii. Pro displacement mapy je doporučeno využít alespoň 16 bitů na každý prvek, protože u nižších hodnot nemusejí být detaily dostatečně rozlišitelné a model se nebude podobat zamýšlenému vzoru. [5] [11]



Obrázek 13: Displacement mapování upravuje geometrii celého objektu. Projev lze sledovat na obrysu útvaru i jeho stínu. Zdroj: Chaosgroup.com

I přes nejkvalitnější úpravu povrchu se teselace v herním prostředí využívá zřídka. Hlavním důvodem je neúměrná výpočetní a paměťová náročnost. Při vzniku nové geometrie vznikají na objektu nové vrcholy a o každém z nich je potřeba udržovat informace. Vznikají složitější grafické modely, vykreslovací řetězec musí aplikovat transformace na větší počet grafických primitiv. Další nevýhodou teselace je, že ji je možné aplikovat pouze na celé objekty a nejen jeho části, a to omezuje vývojáře a designéry při optimalizaci. [5]

3.2.4 Globální osvětlení

Globální osvětlení (dále jen GI, v angl.: Global Illumination) je název procesu zodpovědného za vykreslení 3D scény se zahrnutím přímého osvětlení a následně odraženého světla. GI simuluje fyzikální jev šíření světla, kdy emitor vysílá částice fotonů, které se odráží od povrchu objektů a přejímají jeho energii (barvu), odražené fotony následně dopadají na povrch jiného objektu, na jehož povrch přenáší získanou barvu. Scény vykreslené za použití GI vykazují oproti scénám s pouhým přímým osvětlením fotorealistickou kvalitu. Mezi hlavní efekty nepřímého osvětlení patří odrazy, lom světla a stíny.

GI nachází široké uplatnění v různých disciplínách počítačové 3D grafiky. Například se využívá při tvorbě fotorealistické grafiky a počítačově generovaných animací pro potřeby filmařského průmyslu. Bohužel v kontextu herních enginů se vyskytují jistá omezení, která nelze dnešními technologiemi jednoduše překonat. Důvodem je náročnost výpočtů pro osvětlení celé scény. Pro simulování osvětlení a následného vykreslení scény v reálném čase, kdy je potřeba generovat co největší počet snímků v co nejkratším časovém intervalu, se jedná o obtížně realizovatelný úkol. Pro již zmíněné příklady, kde je GI přípustnou cestou, nepředstavuje výpočetní náročnost takový problém, jelikož doba vykreslování jednoho snímku není určujícím kritériem. I přes uvedené omezení si globální osvětlení našlo cestu do dnešních herních enginů. Často se lze ale setkat se zjednodušeným přístupem. [16]

Tak aby bylo možné začlenit GI do simulací v reálném čase, je nutné vzdát se některých ze vznesených požadavků. Pro herní prostředí je možné využít GI pouze pro objekty, jejichž poloha je známá předem a zůstane dále neměnná. Takové objekty se označují za statické, a proto se lze často setkat s pojmem *statické globální osvětlení*. [16] [17] Výpočet statické globální osvětlení není realizováno při běhu aplikace, nýbrž předem ve vývojové fázi. Pro specifikované statické objekty jsou procesem, zvaným budování osvětlení (v angl.: baking lightmaps, build lighting), vytvořeny datové struktury, které jsou uloženy k pozdějšímu využití. Struktury s názvem *lightmapy* obsahují statickou informaci globálního osvětlení povrchů. Takto uložené osvětlení má při běhu procesu zvladatelnou výpočetní režii. Přesunutím výpočtu mimo samotný běh programu je možné dosáhnout velmi kvalitních efektů. [16]

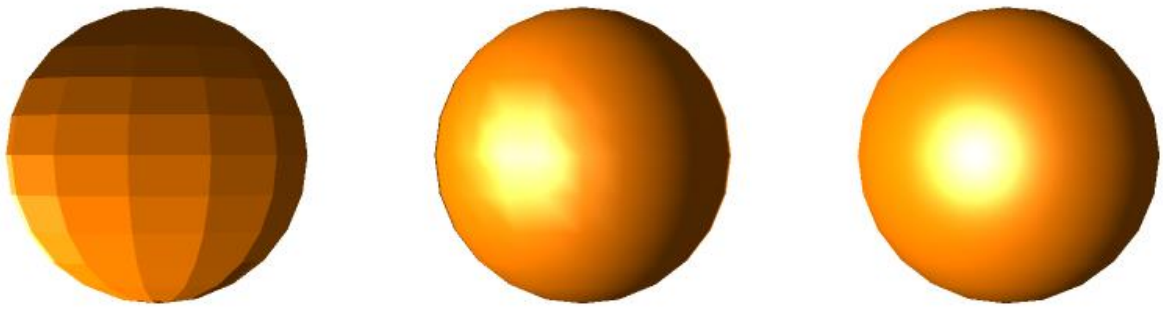
Nevýhody statického GI je jeho fixní povaha. Například úpravou statického objektu nedojde ke změně vrženého stínu, ten zůstane vykreslen dokonce i po úplném odstranění objektu. Stejně tak nelze upravovat atributy světla, jako je barva a intenzita. Dalším nedostatkem je nezahrnutí dynamických objektů do konečného osvětlení scény. Za dynamické objekty lze považovat ty, u kterých se při běhu aplikace předpokládá nějaký typ transformace. Transformací se rozumí pohyb, změna velikosti, rotace atd. Dynamické objekty tak nemohou ovlivňovat výsledné osvětlení scény. [16] [17]

Tak, aby nebyly dynamické objekty vytrženy z barevného kontextu scény, jsou navržena řešení. Jedním z nich je využití *Indirect Lighting Cache*, který předložila společnost Epic Games, Inc. Řešení spočívá v tom, že jsou napříč virtuálním prostředím rozmístěny světelné vzorky. Ty mohou být uspořádány například v trojrozměrné mřížce. Během budování statického osvětlení je do každého vzorku zachyceno nepřímé osvětlení ze všech směrů. Při průniku dynamického objektu a vzorku dochází k provedení testů pro přítomnost světla, pokud test neuspěl, dochází k aplikování interpolované barevné hodnoty pro příslušný vzorek na povrch dynamického tělesa. Dynamický objekt přijímá odražené světlo, ale nemůže jej odrážet zpět na ostatní objekty, ať už se jedná o objekty statické nebo dynamické. [17] Light Probes od společnosti Unity Technologies řeší problematiku podobným způsobem. [16]

Obrázek 14: Trojrozměrná mřížka Indirect Lighting Cache v UE4. [17]

3.2.5 Stínování

Stínování je proces, který se snaží vytvořit u 3D modelů dojem hloubky proměnlivou intenzitou světla a stínů. Objekty 3D grafiky jsou sestaveny z rovinných stěn, a proto je výzvou dosáhnout organického tvaru a spojitě zakřivený povrchů, které je možné pozorovat u reálných vzorů. Pro simulaci těchto efektů se využívají algoritmy stínování, jež interpolují hodnotu stínování napříč povrchem.



Obrázek 15: Vykreslení objektu za pomocí různých stínovacích algoritmů. Ploché stínování (vlevo), Gouraudovo stínování (uprostřed), Phongovo stínování (vpravo). [18]

Ploché stínování

Ploché stínování (v angl.: flat shading) je výpočetně nejjednodušší algoritmus pro stínování. Intenzita světla a následná barva pixelů je dána úhlem mezi normálou povrchu a směrem světla. Pro všechny pixely náležící danému povrchu je následně jednotně nastavena stejná barva. [18] Nevýhodou plochého stínování jsou na první pohled znatelné přechody mezi jednotlivými polygony. Čím je směr normál povrchů více odlišný, tím je rozdíl barvy stran markantnější.

Gouraudovo stínování

Gouraudovo stínování vypočítává osvětlení pro každý vrchol polygonálního modelu, jeho intenzita je dána úhlem mezi normálou vrcholu a úhlem světla. Normála vrcholu vychází z průměrné hodnoty normál povrchů, jimž vrchol náleží. Následně dochází k interpolaci hodnoty napříč všemi pixely pro daný polygon. Jedná se o relativně nenáročný algoritmus, který přináší jednoduchý výsledek. [18] Nevýhodou je, že při využití u modelů s nedostatečně detailní geometrií vznikají artefakty prozrazující tvar polygonů. Obrázek 15 ilustruje, že při

použití Gouraudova stínování lze sledovat ostré hranice stínů, které prozrazují kostru modelu tvořenou čtyřúhelníky.

Phongovo stínování

Phongovo stínování je výpočetně nejvíce náročné a není velmi často využíváno pro interaktivní scény. Výpočet barvy je prováděn pro každý jednotlivý pixel. Intenzita je zjištěna interpolací normály povrchu pro všechny pixely a následně je zjištěna barva pro daný bod. [18] Na rozdíl od Gouraudova stínování přináší dobré výsledky i při málo detailní geometrii tělesa. Obrázek 15 znázorňuje efekt, kde se tvar nasvícené plochy přibližuje kruhu.

3.3 Fyzikální engine

Fyzikální enginy jsou oborem počítačových simulací se snahou aproximovat s co největší přesností některé fyzikální jevy. Předmětem zájmů jsou především simulace chování tuhých a deformovatelných těles, kapalin a částicových systémů. Realizace fyzických simulací je relativně výpočetně náročná, a proto je často zátěž přenášena z CPU na jiné hardwarové doplňky. Jedním ze způsobů je zavedení *Physics Processing Unit* (dále jen: PPU). Jedná se o dedikovaný mikročip určený pro řešení fyzických simulačních problémů. Distribuce dnešní výpočetní síly pro fyzikální simulace se ale ubírá směrem k využití grafických čipů. Název metody je General Purpose processing on Graphics Processing Unit (dále jen: GPGPU). Pro využití různých čipů, které lze najít v osobních počítačích a dalších zařízeních, bylo vytvořeno standardizované rozhraní a jazyk OpenCL (v angl.: Open Computing Language) umožňující paralelní programování na těchto čipech. [19] Tento standard dnes nejvíce rozšiřují společnosti NVIDIA s architekturou Computer Unified Device Architecture (CUDA) a AMD s technologií AMD Fire Stream. Obě společnosti jsou členy Khronos Group, která je zodpovědná za přípravu norem pro OpenCL. [20]

3.3.1 Vysoce-precizní fyzikální simulace

Přístup fyzikálních simulacích se značně liší v závislosti na času nezbytném pro provedení výpočtů. Pro vědecké a fotorealistické potřeby, kdy čas výpočtů není klíčovým kritériem, se volí vysoce precizní výpočetní modely s vysokým počtem vstupních parametrů a simulovaných prvků. Například pro výzkumu molekulární dynamiky a chování materiálů bylo simulováno současně více než 320 miliard atomů. Experiment byl proveden na architektuře IBM Blue Gene/L. Superpočítač Blue Gene/L dokáže zpracovávat až 5.6 GFLOPS, což odpovídá výkonu pro zpracování 10^9 operací s desetinnými čísly za vteřinu. [21] Mezi dosud největší fyzikální simulaci je považována Q Continuum, která byla realizována na superpočítači Titan, jenž byl postaven v Oak Ridge National Laboratory. K výpočtům využívá akcelerované GPU a při zátěžových testech mu byl naměřen výkon $17.59 \cdot 10^{15}\ FLOPS$ s teoretickým výkonem až $27 \cdot 10^{15}\ FLOPS$. Předmětem simulace bylo vytvoření modelu zrodu vesmíru a jeho vývoje v průběhu 13.8 miliard let. Simulovaný prostor se rozprostírá na $1300 \cdot 10^{18}\ pc^3$ a zahrnuje více než půl bilionu částic. [22] Ať už se jedná o fotorealistické nebo vědecké účely, vysoce-precizní simulace značně vytěžují hardwarové prostředky a často je jejich trvání dlouhodobého charakteru.

3.3.2 Simulace v reálném čase

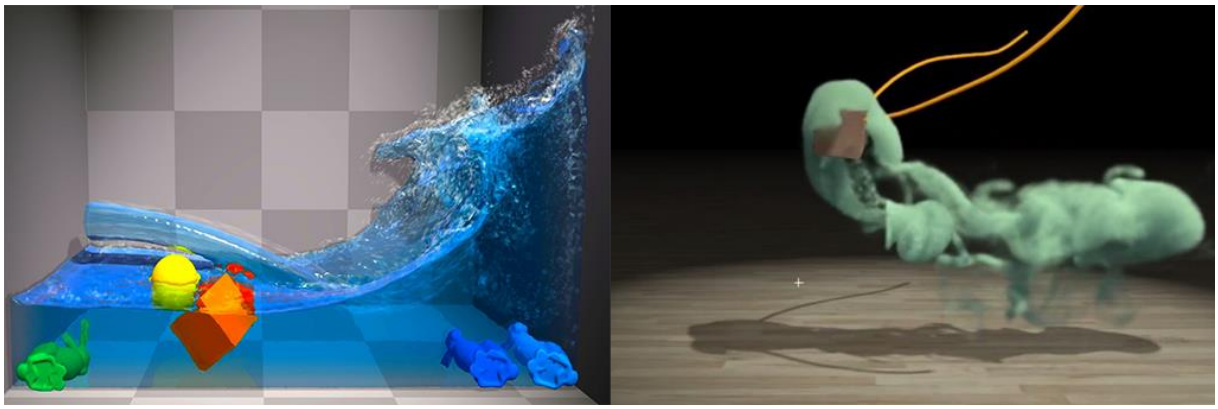
Pro simulace v reálném čase se využívají zjednodušené modely výpočtů. Jejich cílem je spíše docílení uvěřitelného efektu pro pozorovatele, než dosažení přesného výsledku simulace fyzikálních zákonů. Většina simulací je omezena jen na hrubé napodobení chování reálných těles. Ve velké většině případů se pracuje pouze s tuhými tělesy, jejichž simulace je nejméně výpočetně náročná. Do nových produktů se ale čím dále tím častěji začínají prosazovat simulace oděvů, textilií, kapalin a destrukcí. Simulování deformovatelných těles v reálném čase je realizovatelné, ale ve větším měřítku je pro současný hardware těžce dosažitelné.

NVIDIA PhysX

NVIDIA se svou technologií APEX postavenou na enginu PhysX, pokrývá značnou část dnes žádaných vlastností. Fyzikální engine PhysX má široké uplatnění a nachází se v mnoha herních titulech. Je možné se s ním setkat ve videohrách například: Batman Arkham Asylum, The Witcher 3: Wild Hunt a aplikacích pro tvorbu 3D grafiky a animací: Autodesk Maya, 3ds Max. Je využit pro simulování fyziky i v middleware softwarových řešeních, jako jsou UDK, Unreal Engine 4 a Unity. [4] Účelem škálovatelného rozhraní je poskytnou vývojářům vysokoúrovňovou API. Celé rozhraní APEX je rozděleno do modulů s charakterizujícím zaměřením:

- oděvy (APEX Clothing)
- destrukce (APEX Destruction)
- částicové systémy (APEX Particles)
- turbulence (APEX Turbulence)
- vegetace (Apex Vegetation)

Nově přidává NVIDIA do své sady Gameworks modul PhysX FLEX. Jedná se o technologii založenou na simulacích částic. Na rozdíl od tradičních přístupů, kdy jsou efekty tvořeny nezávislými objekty s algoritmy určující jejich individuální chování, využívá FLEX uniformní částicovou reprezentaci pro všechny objekty, a tak může v reálném čase efektivně simulovat jejich vzájemné interakce. Mezi klíčové vlastnosti FLEXU patří dynamické napodobování kapalin, plynů, látek, provazů, částic, tuhých a deformovatelných těles a mnohem více. Všechny výše zmíněné objekty dokáží akcí vyvolávat reakci na ostatních objektech umístěných v prostředí. [5]

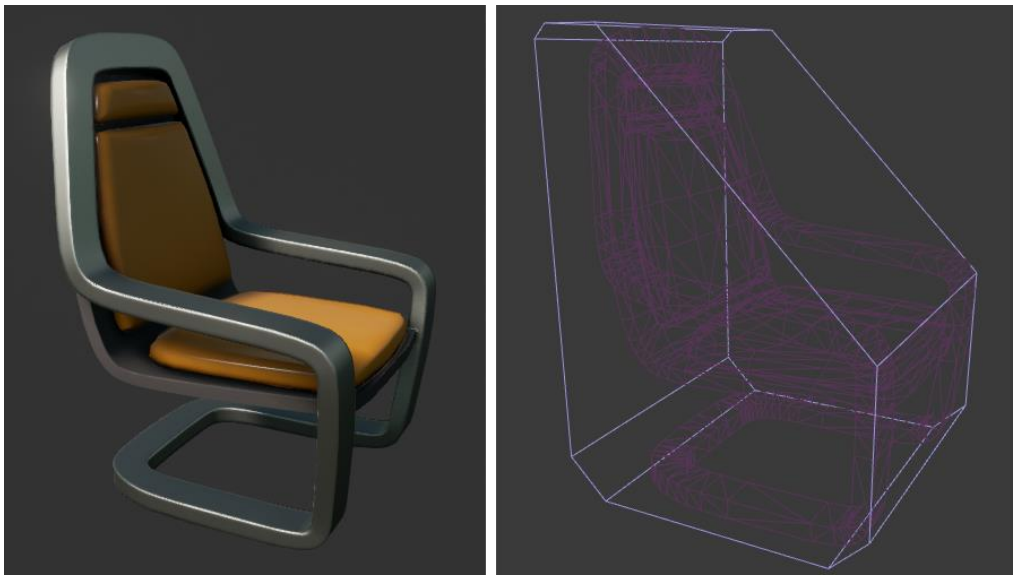


Obrázek 16: Představení NVIDIA FLEX. Vzájemné interakce objektů: kapalina a pevná tělesa (vlevo); plyn, pevné těleso a provaz (vpravo). [5]

Kolizní modely

Při simulaci fyzických jevů často dochází ke kolizi existujících objektů. Jednotlivé objekty jsou reprezentovány relativně komplexními polygonovými modely, které určují jejich pozorovatelný tvar[11]. Pro odlehčení výpočetní náročnosti jsou vysoce detailní polygonové modely objektů nahrazovány jednoduchou kolizní geometrií. Původní drátový model objektu je v rámci fyzikálního enginu ignorován a veškeré simulace jsou prováděny na kolizním modelu. Výhodou je, že engine nemusí pracovat se složitou strukturou vrcholů a hran. Tvar kolizních útvarů není nikde stanoven a uživatel si je může definovat sám. V mnoha SW se objektům přiřazuje výchozí kolizní geometrie, obvykle se lze setkat s kvádry, válci a konvexními mnohostěny. Výsledného kolizního modelu lze dosáhnout i skládáním více geometrických útvarů. [6]

[11] Pozorovatelným tvarem se rozumí polygonový model, který bude zpracován vykreslovacím řetězcem a následně předložen v rastrové podobě uživateli.



Obrázek 17: 3D model židle v UE 4. Dratový model židle (tmavě fialová) a její kolizní geometrie (světle fialová). Zdroj: UE4

3.3.3 Pevná a deformovatelná tělesa

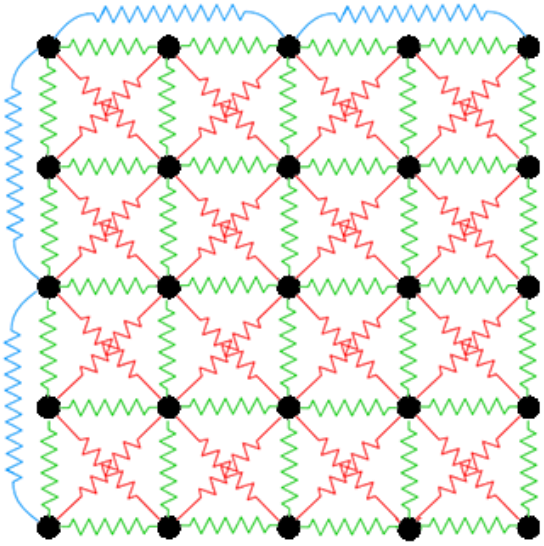
Jedním ze způsobů rozlišení deformovatelnosti fyzikálních objektů, je identifikování jeho reakce po doručení počátečního impulsu. Hypotetická tělesa, která nevykazují po aplikování externích sil známky deformace, se označují za tuhá. Taková tělesa nelze v reálném světe nalézt a své uplatnění nacházejí v situacích, kde postačí zjednodušený matematický model. Působení vnějších sil nezpůsobuje změnu objemu a vzdálenosti jednotlivých bodů, které definují jeho tvar, ty jsou vždy konstantní. Parametry potřebné k popsání systému tuhého tělesa jsou redukovány na rotaci a pohyb. Na druhou stranu tu jsou tělesa deformovatelná, která přenášejí působení externí síly na svůj vnitřní stav. U deformovatelných těles dochází ke změně relativní vzdálenosti vnitřních bodů, a tak se mění objem a tvar tělesa. Pro popsání vlastnosti deformace narůstá počet parametrů. [7]

Rozhodnutí pro výběr typu tělesa záleží na více kritériích, ale jedním z hlavních je zcela zjevná vyšší výpočetní náročnost deformovatelných systémů. Pro výběr typu tělesa se zřetelem na optimalizaci 3D aplikace je třeba si položit následující otázky:

- O jaký typ fyzikální simulace se jedná, je v ní třeba docílit vysoké vypovídající hodnoty?
- V jakém množství se těleso ve scéně objeví?
- Jakou roli objekt hraje? Jedná se o klíčový předmět? Bude dostatečně zřetelný?

Mass-Spring model

Simulace hmotných bodů (také částic) a pružin je v dnešní době asi nejvíce využívaná metoda pro napodobování deformovatelných těles. Měkká tělesa jsou definována souborem fiktivních pružin a hmotných bodů rozmístěných do struktury. Tyto struktury mohou nabývat více rozměrů, například pro simulování vlasů a vláken se využívá jednorozměrné struktury, pro napodobení chování textilií se využívají dvojrozměrné mřížky a pro měkkou tkáň trojrozměrná síť. Mezi uzly vytvořené struktury jsou následně vloženy pružiny. [7] Umístění pružin a nastavení jejich sil je účelné a bude záviset na požadovaném chování materiálu.



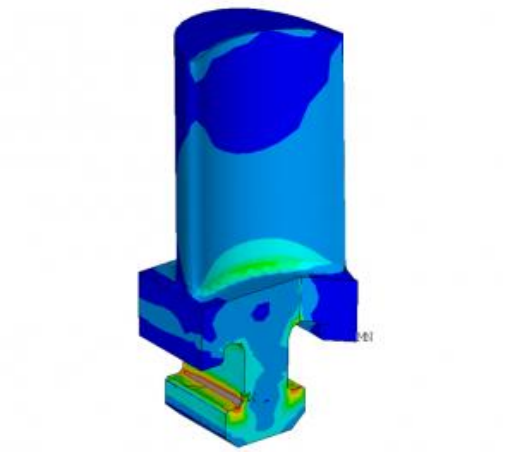
Obrázek 18: Model pro simulaci deformací látky. Hmotné body (černé), strukturální pružiny (zelené), diagonální pružiny (červené), ohýbání (modré). Zdroj : gorkin.com

Ve všech krocích fyzikální simulace je pro každý hmotný bod zjištěn vliv vnějších a vnitřních sil. Vnější síly působí v podobě gravitace, odporu, impulzu vzniklém při kolizi objektů atd. Vnitřní síla je výsledkem součtu sil vyvíjených pružinami v daném bodě. Pohyb dané částice je zjištěn po aplikování příslušných fyzikálních výpočtů. V praxi se často využívá druhého Newtonova pohybového zákonu: $F = m \cdot \frac{\partial v}{\partial t}$. [7] [8]

Obrázek 18 znázorňuje příklad modelování prostřednictvím metody mass-spring. Jedná se o dvojrozměrnou mřížku pro simulaci chování látky. Při uvedení látky do pohybu působí vnější síly na jednotlivé částice. Uvedený model rozlišuje tři základní typy sil. První silou je napínání látky strukturálně podle její geometrie. Pro simulaci jsou vytvořeny strukturální pružiny (zelené), které vykazují vysoký odpor, neboť tento efekt aproximuje napínání textilie podél neflexibilního vlákna. Pro diagonální napínání jsou vytvořeny pružiny (červené), které leží na uhlopříčkách všech pomyslných čtyřúhelníků a jejich odpor je nízký. Látka klade odpor i při jejím ohýbání mimo rovinu, který je realizován modrými pružinami. [8]

Metoda konečných prvků

Metoda konečných prvků (v angl.: Finite Element Method, dále jen: FEM) je způsob simulace deformovatelného tělesa. FEM nachází největší uplatnění v oboru strukturální mechaniky pro analýzu strojních dílů. Předmětem simulací jsou například zátěžové a teplotní testy materiálu. [9]



Obrázek 19: Predikce únavové životnosti oběžných lopatek parních turbín. [9]

Principem metody je vytvoření spojitého prostředí z diskrétních bodů (prvků). Každý z bodů nese své fyzikální vlastnosti, které jsou vyžadované pro cílovou analýzu. Charakteristiky prvku bývají v podobě hustoty, tepelné vodivosti, pružnosti použitého materiálu atd. Uspořádání prvků odpovídá povaze a tvaru simulovaného předmětu a často bývá 2D mřížkou nebo 3D sítí. [9] Na rozdíl od metody Mass-spring, FEM velmi dobře zachycuje objem tělesa, který zůstává reprezentován i po jeho deformaci. Nevýhodou FEM je značná výpočetní náročnost a tím i obtížná realizovatelnost pro simulace v reálném čase. [10]

4 Vlastní práce

Tvorba projektu v herních enginech zahrnuje činnosti ze širokého prostředí tvorby 3D aplikací. V rámci této kapitoly bude popsán proces tvorby finálního projektu v jednotlivých jeho krocích.

4.1 Příprava grafických a zvukových prostředků

Simulátor dopravy pro výuku řízení se snaží napodobit reálné situace. K navození cílového dojmu jsou využity prostředky, které jsou dostupné v odvětví 3D aplikací a zároveň jsou vhodné pro simulace v reálném čase. Klíčovými kritérii pro jejich výběr a tvorbu byla optimalizace výkonu a vzbuzení esteticky pozitivního dojmu. Za účelem efektivní manipulace jsou obsahové prostředky po importování do UE4 přiřazeny vytvořeným třídám jako jejich atributy. Nově vzniklé třídy plní veškeré vlastnosti objektově orientovaného paradigmatu. Tímto způsobem lze třídy skládat do složitějších, nebo děděním specializovat jejich nastavení.

Většina obsahových prostředků použitých v této diplomové práce pochází z externích softwarových aplikací. Tvorba 3D polygonových modelů byla realizována v nástrojích Autodesk 3ds Max 2016 a Blender 2.74. Rastrová 2D grafika určená pro texturování povrchů byla vytvořena v SW Adobe Photoshop 2016. Finální úprava a konfigurace prostředků byla provedena v middlewaru Unreal Enginu 4.

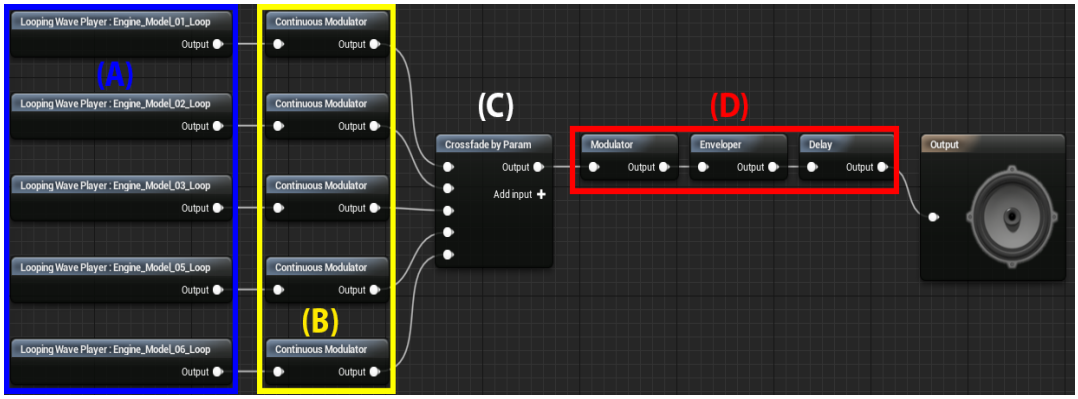
4.1.1 Osobní vozidla

Pro potřeby dopravního simulátoru byly vytvořeny dva základní typy vozidel. Výběr typů vozidel je omezen, protože modelování komplexních objektů jako jsou vozidla, je časově náročné.

Ozvučení osobních vozidel

Pro ozvučení vozidel byly použity prostředky dostupné v základním vybavení UE4. Ve své základní podobě existují pouze jako zvukové stopy, které často zachycují zvukovou smyčku opakující se v určitém intervalu nebo jednorázový zvukový projev. Bohužel jsou tyto prostředky velmi neflexibilní, a proto je důležité přiřadit jim kontrolovatelné chování. Toho lze docílit v Sound Cue editoru, který je součástí základního vybavení UE4. Výstupem je audio soubor s názvem Sound Cue, který lze upravovat dle specifikovaných požadavků. Mimo jiné

lze u každého objektu nastavit rádius útlumu. Při zvyšování vzdálenosti posluchače od zdroje zvuku dochází k postupnému snižování výsledné hlasitosti.



Obrázek 20: Sound Cue pro ozvučení motoru vozidla vytvořená v SW UE4.

Obrázek 20 ilustruje Sound Cue pro zvuk motoru osobních vozidel. Kategorie (A) reprezentuje zvukové stopy ve své surové podobě. V tomto případě se jedná o pět odlišných zvukových smyček chodu motoru. Skupina (B) je tvořena průběžnými modulátory hlasitosti a výšky zvuku. Pro obě vlastnosti je stanovena je minimální a maximální hodnota, vstupním parametrem je počet RPM motoru vozu, který je do tohoto intervalu normalizován. Položka (C) zajišťuje plynulé prolínání jednotlivých stop. Rozhodujícím parametrem je opět RPM vozu, na jehož základě je stanovena hlasitost jednotlivých zvukových stop. V jednom časovém okamžiku bývá přehráváno více zvukových stop s odlišnou intenzitou. Skupina uzlů (D) se stará o přidání náhodné složky hlasitosti a výšky zvuku. Vytvořen je zde i efekt pro postupný nárůst hlasitosti na počátku přehrávání stopy. Zvuková montáž pro zvuk motoru je použita u všech osobních vozidel a je aktivována s nastartováním motoru.

Při kolizi osobního vozidla s jiným objektem je střet doprovázen vlastním audio doprovodem. Sound Cue kolize vozu je realizována podobným způsobem jako zvuk motoru. Za vstupní parametr, který udává hlasitost a výšku zvuku, je zde využita síla impulzu vzniklá při nárazu. Zvuková komponenta je ve výchozím stavu vypnutá. Vznikne-li událost kolize, dojde k jednorázovému přehrání. Komponenta je po ukončení přehrávání znovu uvedena do neaktivního stavu. Posledním ozvučeným prvkem osobních vozidel je zvukový doprovod signalizačního osvětlení. V tomto případě nebyl využit Sound Cue editor a zvuk je prostou zvukovou smyčkou. Při aktivování směrové signalizace dojde k opakovanému přehrávání souboru, dokud není signalizace opět vypnuta.

Osobní vozidlo třídy A

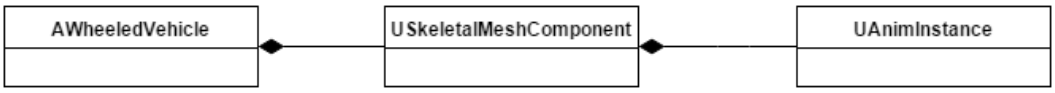
Obrázek 21: Osobní vozidlu třídy A.

První vozidlo je určené pro obsluhu uživatele. Model byl pořízen z trhu, který je provozován společností Epic Games, Inc. Trh je dostupný pro vývojáře v prostředí UE4. Jedná se o velmi detailní model, který byl již při zakoupení vybaven šesti diskrétními úrovněmi Level Of Detail[12] (dále jen: LOD). Sítě modelů jsou sestaveny v rozmezí od 200 tisíc pro LOD 0 do 26 tisíc polygonů při nejméně detailní variantě LOD. To lze pro vykreslování 3D grafických prvků v reálném čase považovat za složitý model, a proto je nutné zajistit jen omezený počet instancí tohoto typu vozidla. Vozidlo je vytvořeno z hierarchické struktury dílčích komponent, což umožňuje designérovy animovat jednotlivé části vozu.

Obrázek 22: Animované komponenty v interiéru vozidla A: otáčení volantu (A), ručičky tachometru (B), polohování řadicí páky (C).

[12]Diskrétní level of detail je metoda pro optimalizaci polygonových modelů, která nahrazuje složité modely jednodušší variantou a naopak. V rámci diplomové práce se bude vždy hovořit o diskrétním LOD.

Ve vozidle A je kamera uživatele umístěna do kabiny přibližně v úrovni očí, tak aby co nejvěrohodněji napodobovala výhled řidiče. Vzhledem k tomuto umístění kamery byly zvýšeny nároky na detaily v interiéru vozu. Animování jednotlivých částí vozu bylo realizováno na základě vlastností vozidla (tzn. na základě vnitřního stavu objektu). Tento přístup je umožněn díky vztahům tříd, a tak může objekt animace přistupovat k jednotlivým atributům vozidla.



Obrázek 23: Diagram tříd pro znázornění vztahů vozu, síťového modelu a animací.

V interiéru vozu je animován volant, jehož otáčení po ose X je závislé na úhlu natočení předních kol vozu: $\angle otočení\_volantu = (\angle kolo_{LP} + \angle kolo_{PP}) \cdot \frac{3}{2}$. Pozice ručiček tachometru vychází z hodnot rychlosti vozidla a počtu otáček motoru za minutu (v angl.: revolutions per minute, dále jen: RPM). Poloha řadicí páky je stanovena v diskrétních hodnotách a odpovídá aktuálnímu převodovému stupni.

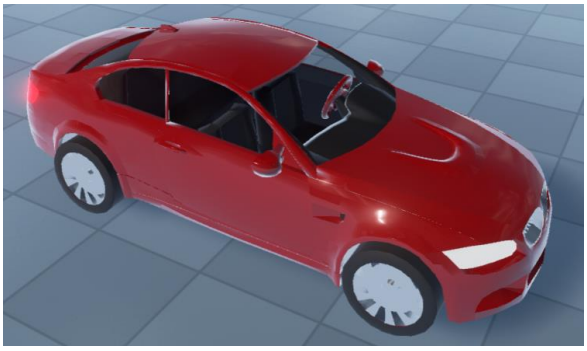
Vozidlo typu A má náhon na přední nápravu. Otáčení předních kol je závislé na RPM motoru, rychlost otáčení zadních kol odpovídá rychlosti vozu. Zatažením ruční brzdy dojde k jejich zablokování. Vozidlo typu A je vybaveno kotoučovou brzdou. Kotouč brzdy je součástí polygonového modelu kola. Brzdové čelisti jsou samostatnou komponentou a bylo nutné zajistit, aby nerotovali po ose kol (osa Y), ale přejímaly jejich rotaci při zatáčení (osa Z).

Obrázek 24: Dynamické parametrizované materiály: vypnuté osvětlení vozidla (vlevo), zapnuté osvětlení (uprostřed), brzdové osvětlení (vpravo).

Součástí pořízeného balíku byly materiály pro jednotlivé povrchy modelu. Některé byly upraveny tak, aby bylo možné dynamicky zasahovat do jejich vzhledu prostřednictvím skalárních a vektorových parametrů. Dynamické materiály byly použity pro manipulaci světel, kde vektorové parametry reprezentovaly barvu a skalární intenzitu emise světla.

Kolizní model vozu je vytvořen pěti nezávislými konvexními mnohostěny, které jsou uspořádané tak, aby co nejlépe vystihovaly karosérii vozidla. Vznikne-li událost kolize, lze rozpoznat, které z kolizních primitiv úder zaznamenalo a následně provést příslušné reakce na vzniklou událost. Tímto způsobem je možné odhadnout místo nárazu a aplikovat v daném místě lokální poškození spolu s příslušným zvukovým doprovodem. Každému ze čtyř kol je nadále přiřazen samostatný kolizní mnohostěn aproximující válec, který odpovídá jejich průměru a šířce.

Osobní vozidlo třídy B



Obrázek 25: Osobní vozidlu třídy B.

Druhé vozidlo bylo navrženo s důrazem na jednoduchost polygonové sítě, a proto je bez většího dopadu na výkon vhodné pro husté obsazení scény. Jeho tvorba proběhla v SW Autodesk 3ds Max manuální redukcí topologie složitějšího modelu. V tomto procesu byly slučovány „nadbytečné“ vrcholy a hrany. Odstraněním detailů byl model zjednodušen z původních 360 tisíc polygonů na zlomek původního počtu. Optimalizovaný model má dvě úrovně LOD s deseti tisíci trojúhelníky pro LOD 0 a při odstranění interiéru bylo u LOD 1 dosaženo necelých třech tisíc polygonů. Hierarchická struktura modelu je velmi jednoduchá. Kořenem struktury je trup vozidla, který má pouze čtyři potomky v podobě kol. Možnost animování vozu je tím značně snížena.

Zřetel na zamýšlenou nízkou náročnost vozidla typu B byl brán i při tvorbě materiálů. Použité materiály nevyužívají textur a výsledného vzhledu bylo dosaženo nastavením základní barvy a hodnot pro kontrolování hrubosti a kovového vzezření povrchu. Pro dosažení unikátního dojmu každého vozu jsou při jeho umístění do scény vygenerovány náhodné hodnoty ve stanoveném rozmezí, které upravují základní vlastnosti povrchu. Na první pohled mohou tímto způsobem vznikat naprosto odlišné výsledky. Vůz může být například bílý s matným povrchem odrážející pouze zlomek světla nebo červený s téměř zrcadlovým odrazem. Nevýhodou přístupu bez mapování textur je uniformní vzhled materiálu pro celý povrch. Kolizní model vozu B byl také zjednodušen. Pro karosérii vozu je vyhrazen pouze jeden kvádr. Kolize kol jsou řešeny stejným způsobem jako v případě vozu A.

4.1.2 Dopravní značení

Součástí vyvíjeného prostředí jsou i modely a textury použité pro vykreslení dopravních značek. K tomu aby bylo použití dopravních značek univerzální, bylo k tvorbě přistupováno modulárně. Značky byly rozděleny na tři základní komponenty, které lze následně v prostředí UE4 skládat do finálního produktu. Tím vzroste počet variací a s minimálním úsilím lze dosáhnout nových typů dopravního značení. Značení je rozděleno na informační plochu (kruhová, čtvercová, obdélníková a osmistěnná), sloupky v různých velikostech a tvarech a upevnění, které připojuje informační plochu k sloupku. Pro dopravní značení bylo připraveno několik materiálů. Nejprve, pro upevnění informační plochy a sloupku, byly připraveny variace shaderů s kovovým vzhledem. Na informační plochy jsou použity jednoduché materiály se základní rastrovou texturou bez dalších vlastností. Důvodem výběru takto jednoduchých materiálů byla potřeba maximalizovat jasnou a nezkreslitelnou informační povahu, kterou by mohly složitější materiály svými efekty omezovat.

4.1.3 Krajina

Krajina bývá často nezbytným aspektem 3D aplikací, a proto má v prostředí UE4 vyhrazen svůj vlastní nástroj pro její tvorbu. Nejhrubší topologii udává tvar krajinného povrchu[13]. Detail krajiny je později dotvářen designéry, kteří do prostředí umisťují připravené 3D prostředky, generují vegetaci a malují povrch.

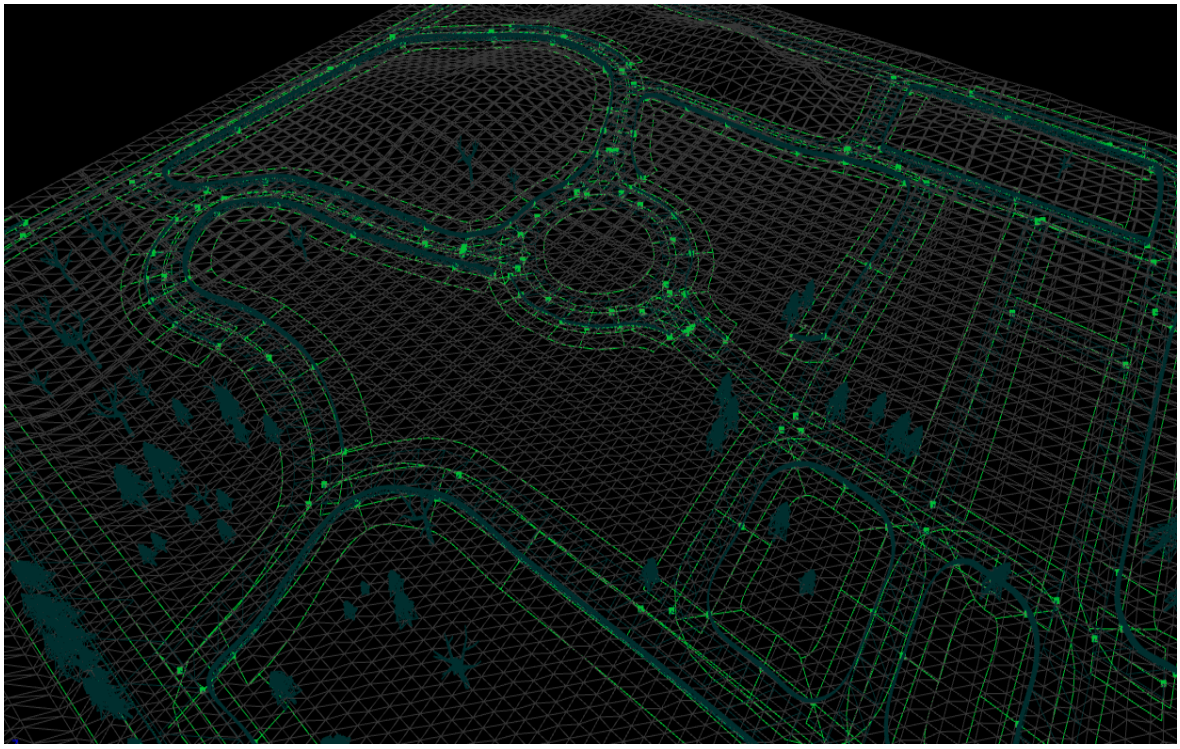
Polygonová síť krajiny

Tvorba krajinného povrchu proběhla v SW UE4 v režimu správa krajiny. Povrchem se rozumí síť tvořená čtvercovými polygony. Celá síť je rozdělena na komponenty, které jsou dále složeny ze sekcí. Na sekcích jsou prováděny techniky LOD a culling[14], a proto je jejich velikost klíčová pro zátěž vyvinutou na CPU a GPU. Menší sekce dokáží využít obou technik s větší přesností, ale jejich nároky jsou mnohem vyšší. Pro modelování rozsáhlých a otevřených prostředí se využívají větší sekce. To způsobuje, že se komponenta skládá z méně sekcí a tím je redukován počet na sobě nezávislých úrovní LOD, které musí být zpracované.

[13] Pro krajinný povrch se v UE4 používá termín landscape (v angl.: krajina).

[14] Culling je proces odmítání objektů, které tak nebudou využity ve výsledném vykreslení. Rozlišují se techniky softwarové (CPU) a hardwarové (GPU). V případě tvorby krajin v UE4 se bude vždy jednat o způsob filtrování objektů s využitím CPU. [3]

Pro potřeby diplomové práce byla vytvořena krajina rozprostírající se na půl čtverečním kilometru. Z optimalizačních hledisek byl terén sestaven z šestnácti komponent tvořících čtverec. Každá z použitých komponent je tvořena právě jednou sekcí s rozlišením 63x63 jednotek. Celkové rozlišení sítě je tedy 252x252 jednotek. V UE4 dochází k triangulaci polygonových modelů, a proto je výsledný počet polygonů dvojnásobný. Zvolené rozlišení se ukázalo být jako dostatečné pro zachycení nezbytných rysů krajiny se zachováním co nejnižšího počtu polygonů.



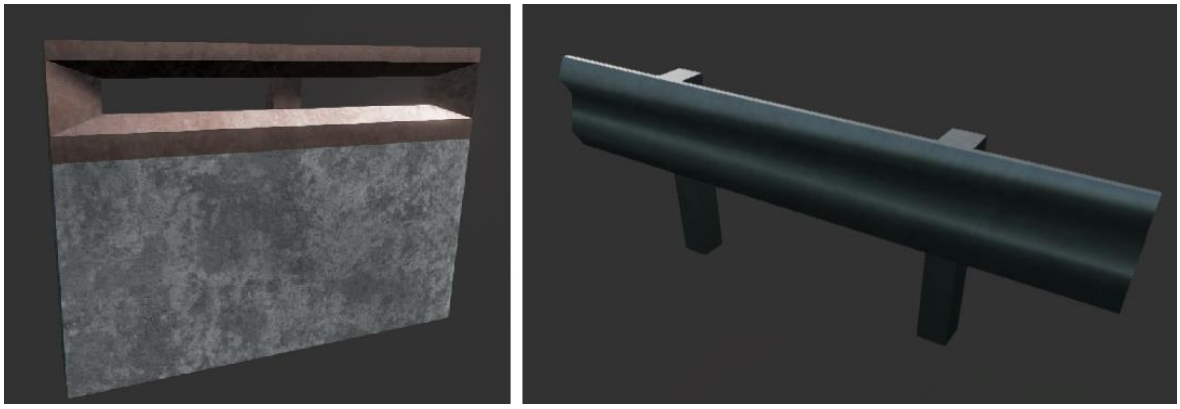
Obrázek 26: Zachycení polygonové sítě povrchu a dalších krajinných objektů.

Příprava krajiny

Po vytvoření krajinného povrchu přichází na řadu jeho tvarování. K tomu dochází přemisťováním pozic vrcholů předem vytvořené sítě bez zásahů do vnější topologie. Pro tvarování povrchu krajiny je v editoru dostupná rozsáhlá sada nástrojů. Frekventovaně byl využíván vyřezávací nástroj (v angl.: Sculpting tool), u kterého lze nastavit poloměr a sílu působení. Nastavitelný útlum může na základě specifikované křivky přizpůsobit sílu působení v závislosti na vzdálenosti od středu štětce. S tímto nástrojem lze snadno dosáhnout přirozeně hladkých přechodů na rozsáhlých plochách. Nástroje pro tvorbu eroze (v angl.: Erosion tool) jsou vhodné pro přidání zvrásnění povrchu, které je v přírodě způsobené přenosem nezpevněných usazenin horniny. Možnosti nástroje dovolují simulovat erozi způsobenou působením větru i vody. Tímto způsobem bylo dosaženo nepravidelného a organického tvaru příkopů a skalisek přiléhajících vozovce.

Spojité objekty a křivkový nástroj

Křivkový nástroj (v angl.: Splines tool) slouží k tvorbě útvarů, které lze označit za jednodimenzionálně spojité. Mezi takové útvary patří vozovky a jiné typy silnic, řeky, oplocení a další objekty, jejichž obecný tvar zachycuje právě křivka. Práce s křivkovým nástrojem se v UE4 neliší od jiných softwarů pro 2D nebo 3D grafiku, které využívají nástroje pro úpravu Bézierovy křivky. Křivka je souborem kontrolních bodů a segmentů. Orientace kontrolních bodů určuje tvar křivky a do segmentů lze dosadit jeden i více 3D modelů. Jednotlivá nastavení segmentů samozřejmě umožňují i detailní manipulaci vybraných 3D modelů. Nastavit lze orientaci, velikost, způsob opakování nebo vyplnění segmentu po jeho délce.



Obrázek 27: Ukázky použitých 3D modelů tvořící spojitý objekt.

Pro tvorbu vozovky byl navržen základní blok s rozměry 800x1200 cm. Připravený blok se v rámci segmentu opakuje a nikdy nepřesahuje své původní rozměry. Délka bloků

v segmentu je automaticky upravována (vždy jen zkracována) a šířka bloku je nezávisle na délce konstantní. Takové neproporcionální úpravy modelu vedou k vizuální deformaci použitého materiálu. Z těchto příčin je nutné dbát na zvolení správné délky segmentu.

Materiály krajiny

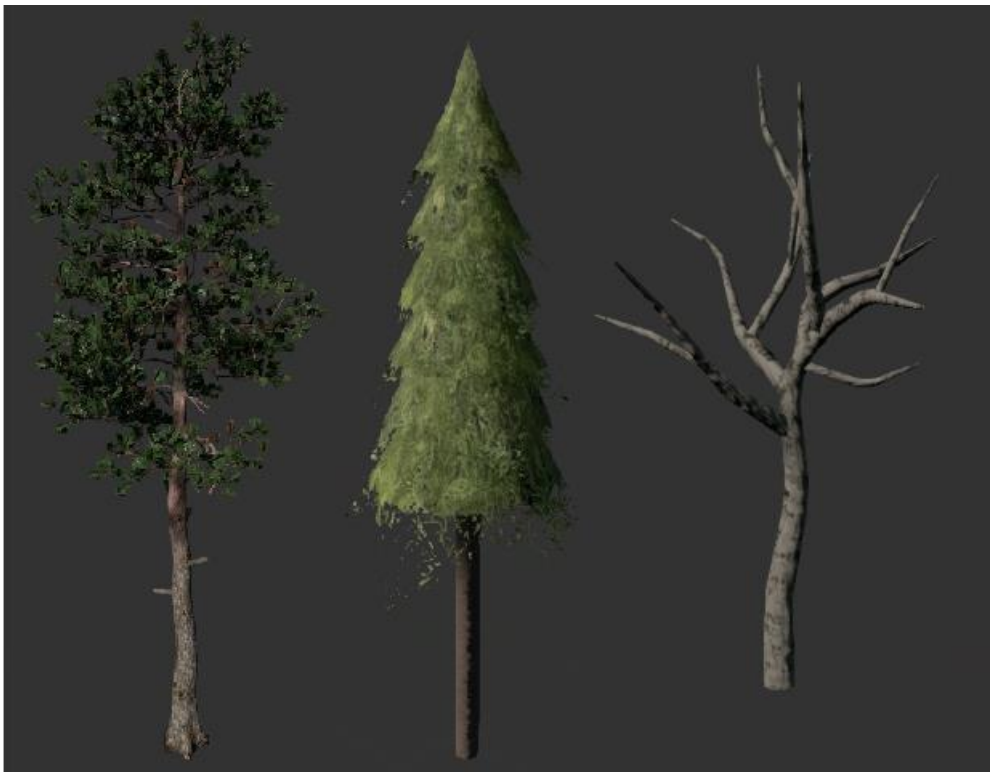
Dokončením úprav na modelu krajiny přichází na řadu její texturování. K tomu je využit nástroj malování (v angl.: Paint tool). V procesu malování je na předem vytvořený povrch aplikována informace zaznamenávající přítomnost materiálu. Tímto způsobem lze vytvářet materiálové vrstvy, které lze v různé intenzitě překrývat. V rukách zkušeného vývojáře je toto chování velmi silným nástrojem, neboť umožňuje dosažení nových povrchů, které přejímají kombinované vlastnosti původních materiálů. Na následujícím obrázku je zobrazen plynulý přechod materiálu pro půdu a trávu.

Obrázek 28: Malování na povrch krajiny - plynulý překryv dvou rozdílných materiálů.

V krajině lze nalézt čtyři základní materiály povrchu: travnatý povrch, půdu, sníh a asfalt, který je použit pro dopravní komunikace. UE4 umožňuje definovat takzvané fyzické materiály (v angl.: Physical material), které slouží jako kolekce vlastností a ovlivňují chování a interakce jiných objektů s povrchem, na kterém je fyzický materiál aplikován. Mezi jednu z těchto vlastností patří tření, které určuje výslednou kluzkost povrchu. Pro dosažení unikátního chování každého povrchu byly vytvořeny tři základní konfigurace. Nastavení s vysokým třením bylo použito pro asfaltový povrch vozovky, střední pro travnatý a hliněný povrch a nízké tření pro sněhový pokryv. S využitím fyzických materiálů doznává automobil naprosto jedinečného chování na odlišných typech povrchu.

Vegetace krajiny

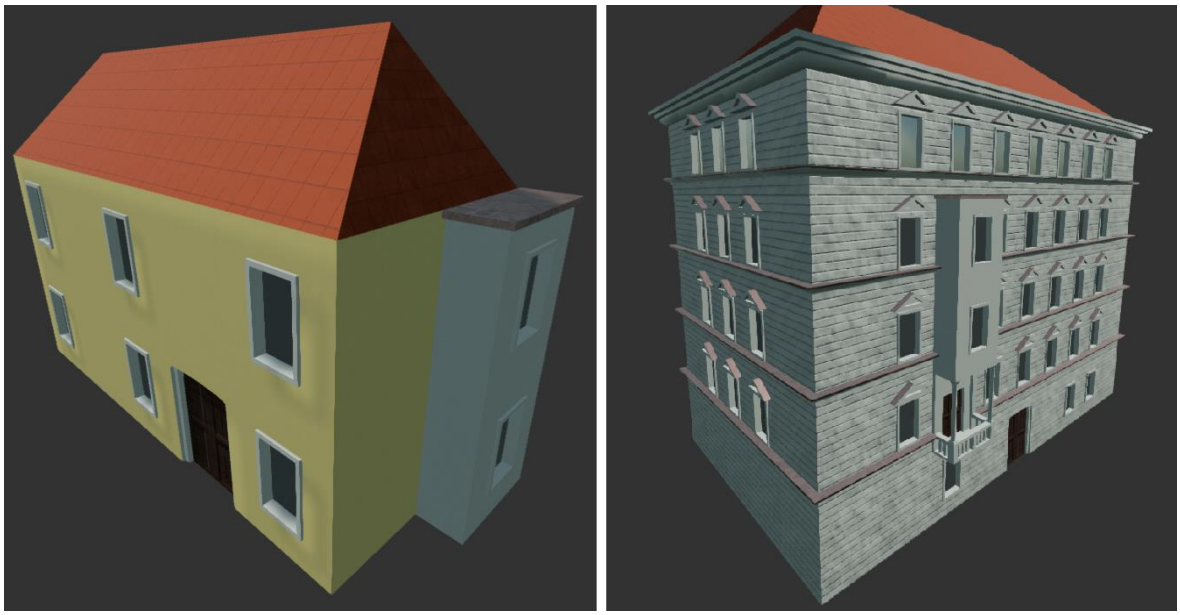
Stromy byly v krajině rozmístěny nástrojem pro tvorbu vegetace. Do výběru nástroje jsou zahrnuty požadované typy vegetace, které jsou reprezentovány jednotlivými grafickými prostředky. U každého prostředku je nastavena míra nahodilosti, která je stanovena vlastnostmi jako je pozice, měřítko a rotace. Pro uvedené vlastnosti je nastaven interval, ve kterém se budou hodnoty pohybovat. Následně jsou štětcem nanášeny na povrch krajiny zvolené objekty, přičemž každý je svým způsobem unikátní, jelikož jeho vlastnosti jsou stanoveny náhodně ve zvoleném intervalu. Hustotu nanášené vegetace lze nadále upravit v nastavení vlastností štětce. Při tvorbě vegetace je nutné zohlednit grafickou náročnost, jelikož množství stromů a jiných rostlin bývá obvykle vysoké, a proto je nezbytné vybrat správné 3D modely pro danou situaci. Použity byly dva typy modelů stromů s nízkým počtem polygonů a texturami o nízkém rozlišení a jeden s vysoce detailním modelem, texturami a animací. Stromy s nízkým detailem se uplatnily při tvorbě zalesnění. Strom s komplexním modelem byl přidělen na předem vybraná místa.



Obrázek 29: Použité typy vegetace.

4.1.4 Exteriér

Okolní krajina je zasazena do prostředí maloměsta, a proto bylo nutné připravit modely a textury, které budou vzbuzovat patřičný dojem zamýšleného prostředí. Důraz byl kladen na stavby a především na obytné budovy. Tvorba budov probíhala v softwaru 3ds Max. Vymodelovány byly modulární bloky staveb s přibližnou plochou patnáct metrů čtverečních, a ty byly následně skládány do celku. Tímto způsobem byly připraveny dva základní polygonové modely. Na následujícím obrázku vlevo je umístěna menší obytná budova, jejíž polygonová síť je tvořena třemi sty polygony. Stavba napravo je znázorněna ve větším detailu. Síť je tvořena tři a půl tisíci polygony. Takto náročný model by mohl mít ve větším využití značný dopad na výpočetní náročnost scény, a proto byla připravena i jeho redukovaná verze pro LOD 1 s polovičním počtem polygonů. Tvorba kolizních modelů byla ponechána automatickému generování, které UE4 nabízí.



Obrázek 30: Příklad použitých budov.

Tvorba modelů rozsáhlých staveb může být časově velmi náročná, a proto je celkový počet staveb relativně nízký. To znamená velkou míru opakujících se prvků napříč krajinou. I přes omezený repertoár modelů bylo klíčové, aby každé místo působilo svým vlastním a unikátním dojmem. K tomu, aby bylo dosaženo vjemové rozmanitosti výsledné scény, došlo k vytvoření několika materiálů a textur určených pro povrchovou úpravu budov. Tento časově nenáročný krok dal prostředí značnou barvitost. Následující obrázek ukazuje některé z použitých vzorů. Barevný motiv a hloubku zvrásnění lze u každého vzoru následně upravovat.

Dosáhnout toho lze velmi snadnou úpravnou hodnot skalárních a vektorových parametrů dané materiálové instance. Na následujícím obrázku jsou příklady vnějších fasád určených pro budovy a jiné stěny. Rozdílu mezi první a druhou byly dosaženo pouhou úpravou základní barvy.

Obrázek 31: Příklady použitých materiálů na povrchu budov.

4.2 Logický model vozidla

Chování vozidla představovalo v rámci diplomové práce jednu z nejdůležitějších částí. Cílem tvorby osobního vozidla bylo dosáhnout co nejvíce realistického napodobení skutečných jízdních vlastností. Zároveň bylo podstatné zohlednit různé typy čtyřkolových osobních vozidel, které se od sebe liší právě jízdními vlastnostmi, typem karoserie atd. Ačkoliv byl brán zřetel na možnou budoucí úpravu systému, není v současné době připravena podpora jiných typů dopravních prostředků. Pro začlenění dopravních prostředků jako jsou nákladní automobily a motocykly by musel připravený systém projít úpravou.

Dílčím cílem práce, kterému je věnovaná pozornost v této kapitole bylo vytvořit šablonu pro uživatelem ovládané osobní vozidlo, které bude dosahovat následujících požadavků:

1. Vozidlo bude ovladatelné uživatelem.
2. Jízdní vlastnosti se budou snažit přiblížit skutečnému chování vozidla.
3. Osobní vozidlo bude uchovávat informace o svých vnitřních stavech. Tyto informace bude možné sdělit uživateli příslušným způsobem.
4. Připravené řešení bude univerzální, a proto následné přidávání nových typů vozidel nebude obtížné.
5. Automobil bude kontrolovat uživatele při dodržování dopravních předpisů.

4.2.1 Hierarchie tříd logického modelu vozidla

Unreal Engine 4 nabízí ve své API celou řadu připravený řešení, které lze využít jako výchozí body při vývoji aplikace. Hojně využívá konceptů objektově orientovaného programování. Aplikační rozhraní je postaveno na principu dědičnosti a skládání, a proto je dobré pochopit, kde se v hierarchii systému nacházejí požadované prvky.

Pro splnění prvního požadavku o schopnosti dopravního prostředku být ovládaný uživatelem je možné využít třídu APawn. APawn je třídou, která je děděním odvozena od třídy AActor, a proto přebírá i její atributy. AActor je vysoko ve stromu dědičnosti a je zobecněna pouze třídou UObject, která je kořenem a všechny třídy i jejich instance jsou potomky právě tohoto základního objektu. APawn má jako jednu ze svých komponent AController, který není fyzicky reprezentovaný a slouží uživateli jako přemostění mezi jeho vstupem a vnějším rozhraním ovládaného objektu. Každý objekt typu AController může mít ve výchozím nastavení vztah pouze s jedním objektem APawn. Instanční objekty třídy AController mohou

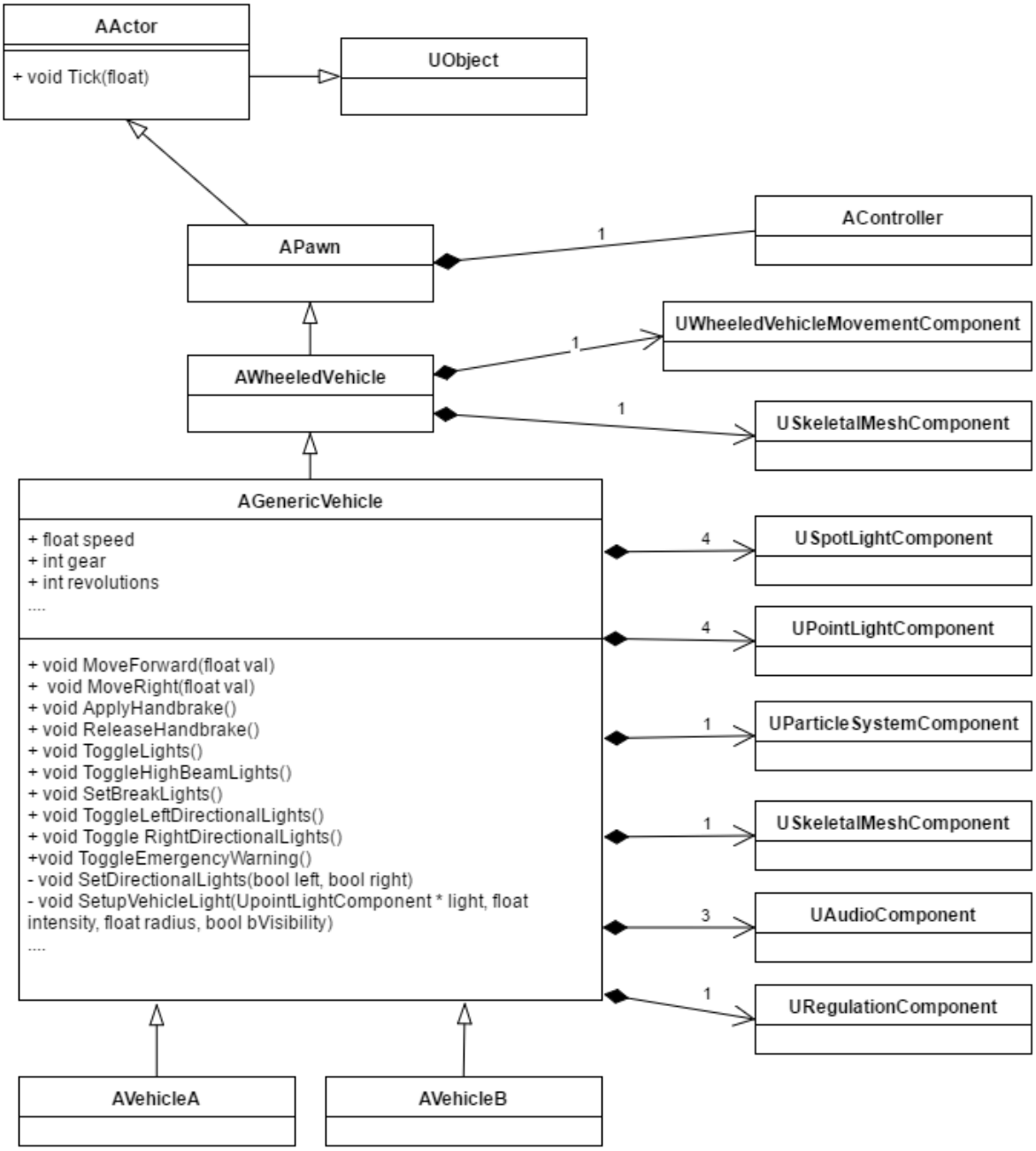
ovládnout cílový objekt metodou `AController::Possess(APawn* InPawn)`, nebo se ho naopak vzdát metodou `AController::UnPossess()`.

V aplikačním rozhraní UE4 se nachází i částečné řešení druhého vzneseného požadavku na realistické jízdní vlastnosti dopravního prostředku. Využito bylo třídy AWheeledVehicle, která rozšiřuje přímo nadřazenou třídu APawn a doplňuje ji o atribut typu UWheeledVehicleMovementComponent. Tento atribut má za úkol obstarat celou logiku pro simulaci jízdy vozidla.

4.2.2 Třída AGenericVehicle

Pro splnění zbývajících požadavků bohužel nestačí programové rozhraní UE4, a proto byly ve zdrojovém kódu projektu v jazyce C++ a ve vizuálním skriptovacím jazyce BluePrint připraveny další rozšíření tříd a komponenty. AWheeledVehicle byl rozšířen vlastní třídou AGenericVehicle, který slouží jako nejbližší společný předek všech logických modelů použitých vozidel. Pro jednotlivé typy vozidel byly pak v BluePrint připraveny další třídy, které již obsahují konkrétní konfigurace pro daná vozidla.

Dodatečná funkcionalita základních tříd je dosažena přidáváním komponent. Komponenta je objekt typu UActorComponent a slouží k seskupení chování nebo logiky, která je znovupoužitelná pro více objektů typu AActor. Třídy AActor a UActorComponent jsou ve vztahu kompozice s kardinalitou vazby 1:N, kde AActor může mít libovolný počet pod-objektů. Objekty AActor mohou využívat rozhraní komponenty, ale je tomu i naopak. Příkladem může být pod-objekt typu UMovementComponent, který dokáže manipulovat s jinou komponentou vlastnícího objektu. Komponenty jsou instanciovány v konstruktoru objektu AActor při jeho tvorbě. Jednou z vlastností komponenty je možnost aktivovat/deaktivovat se v průběhu simulace. S deaktivováním objektu dojde samozřejmě i k dočasnému přerušení veškerého přidaného chování. Komponenty mají svoji vlastní hierarchii dědičnosti, která má kořen v již zmíněné třídě UObject.



Obrázek 32: Zjednodušený diagram tříd pro třídu AGenericVehicle.

Na předchozí ilustraci je znázorněn diagram tříd pro třídu AGenericVehicle, která převzala v názvu uvozující A, tak aby byla dodržena jmenná konvence UE4 spjatá se všemi potomky třídy AActor. Diagram je oproti programové struktuře diplomové práce zjednodušený, neboť reálný rozsah všech atributů tříd a jejich vztahů je mimo možnosti zobrazení.

Mimo již zmíněné komponenty, které třída AGenericVehicle dědí od svých předků jsou přidány nové. Pro osvětlení vozu je vytvořeno osm světelných komponent. Pro objekty typu AGenericVehicle bylo navrženo rozhraní metod, které slouží ke správě jednotlivých vlastností vozidel. Pro ilustraci lze uvést metodu `AGenericVehicle::ToggleLights()`, která pracuje s některými komponentami pro osvětlení a přepíná jejich stav. Vývojář by neměl mít neomezený přístup k vlastním atributům třídy a přistupuje k nim prostřednictvím definovaného rozhraní. Atributy jsou skryty jako privátní, což je považováno za dobrou praktiku, jelikož to omezuje množinu stavů, do kterých by se mohl objekt dostat. K privátním atributům nemohou dědící objekty přistupovat napřímo a musí využívat metod, jejichž viditelnost pro ně není skrytá. Tímto způsobem je snížen počet nekonzistentních stavů vozidla. Jako příklad lze uvést situaci, ve které dojde nezabezpečeným přístupem ke komponentě dálkového světla k její aktivaci, ale nebude nastavena proměnná `AGenericVehicle::bHighBeamLightsOn`, která je dále využívaná v uživatelském rozhraní nebo přepínání mezi běžným a dálkovým osvětlením. Při využití stanoveného rozhraní musí instanční objekty dědících tříd využít metodu `AGenericVehicle::ToggleHighBeamLights()`, která se postará o přepnutí běžného a dálkového osvětlení a nastavení zmíněné proměnné.

Události při každém snímku

U instančních objektů typu AActor je dobré upozornit na metodu `AActor::Tick(float DeltaSeconds)`, která je invokována při každém snímku, a proto je na ní často navěšena funkcionalita, která musí být automatizována v reálném čase. Při volání metody je poskytnut argument, který odpovídá časové periodě mezi snímky tzn. $T = \frac{1}{f}$, kde f je frekvence obnovení scény. Metoda Tick je ve výchozím nastavení vypnuta a je nutné ji povolit. Důvodem je značná výpočetní režie takových objektů, a proto by měl vývojář důkladně zvážit, zda je kýžené chování objektu nutné provádět v každém snímku. Metoda Tick byla ve třídě AGenericVehicle přetížena a průběžně se stará o aktualizaci hodnot reprezentující různé stavy vozidla. Metoda kontroluje, zda splňují brzdová světla podmínky pro jejich rozsvícení a upravuje skalární parametry příslušných dynamických materiálů. Aktualizovány jsou rovněž obecné vlastnosti vozu včetně jeho rychlosti, převodového stupně, počtu otáček motoru za minutu a úhel natočení předních kol. Tyto vlastnosti jsou atributy samotné třídy, a lze je zpřístupnit i v jiných scénářích. Například pro vykreslování uživatelského rozhraní vozidla jsou využívány právě hodnoty: rychlost, RPM a převodový stupeň. Na atributech závisí i objekty

určené pro animaci kostry vozu. Natočení volantu, poloha ručiček na tachometru a poloha řadící páky vychází také z uvedených hodnot. Zvuk motoru vozidla je přizpůsoben jeho RPM.

```
void AGenericVehicle::Tick(float DeltaSeconds){

	SetBreaklights();
	UpdateHUD();
	UpdateSound();
	FLWheelSteerAngle = GetVehicleMovementComponent()->Wheels[0]->GetSteerAngle();
	FRWheelSteerAngle = GetVehicleMovementComponent()->Wheels[1]->GetSteerAngle();


	//Stop Sign
	if (bCheckingIfStopped) {
		if (Speed < 1) {
			bHasStopped = true;
		}
	}

	//Speed limit
	if (Speed > MaxAllowedSpeed) {
		TimeMaxAllowedSpeedExceeded += DeltaSeconds;
	}
}


void AGenericVehicle::UpdateSound() {
	EngineSound->SetFloatParameter("RPM", Revolutions);
}


void AGenericVehicle::SetBreaklights(){
	float BrakeEmissiveValue;
	float TailEmissiveValue;
	if (bHandbrakeOn || bReverseOn) {
		BrakeEmissiveValue = 1400.f;
		TailEmissiveValue = 1400.f;
	}
	else if(bLightsOn) {
		BrakeEmissiveValue = 100.f;
		TailEmissiveValue = 100.f;
	}
	else {
		BrakeEmissiveValue = 0.f;
		TailEmissiveValue = 0.f;
	}

	BrakeLight->SetScalarParameterValue("BrakeIntensity", BrakeEmissiveValue);
	TailLight->SetScalarParameterValue("ReverseIntensity", TailEmissiveValue);
}


void AGenericVehicle::UpdateHUD() {
	UWheeledVehicleMovementComponent* WVMC = GetVehicleMovementComponent();
	Speed = abs(int((WVMC->GetForwardSpeed() / 100000) * 3600));
	Gear = WVMC->GetCurrentGear();
	Revolutions = WVMC->GetEngineRotationSpeed();
}
```

Obrázek 33: Metoda AGenericVehicle::Tick aktualizuje stav vozidla v každém snímku.

Událost kolize

Metoda `AActor::NotifyHit(args)` je volána v případě, že dojde ke kontaktu z některých kolizních modelů (AActor může mít současně několik kolizních komponent s vlastní geometrií) s okolím nebo kolizním modelem jiného objektu typu AActor. Metoda je invokována s argumenty vypovídajícími o vzniklé události.

Předány jsou následující argumenty:

1. Reference na kolizní komponentu, která zaznamenala událost.
2. Reference na objekt typu AActor se kterým došlo ke střetu.
3. Reference na kolizní komponentu střetnutého objektu.
4. Booleova hodnota, která určuje původce kontaktu.
5. Souřadnice určující vznik události.
6. Vektor normály nárazu.
7. Vektor působení síly nárazu.
8. Struktura obsahující informace o nárazu. Zahrnuty jsou informace o povrchu, typu fyzikálního materiálu atd.

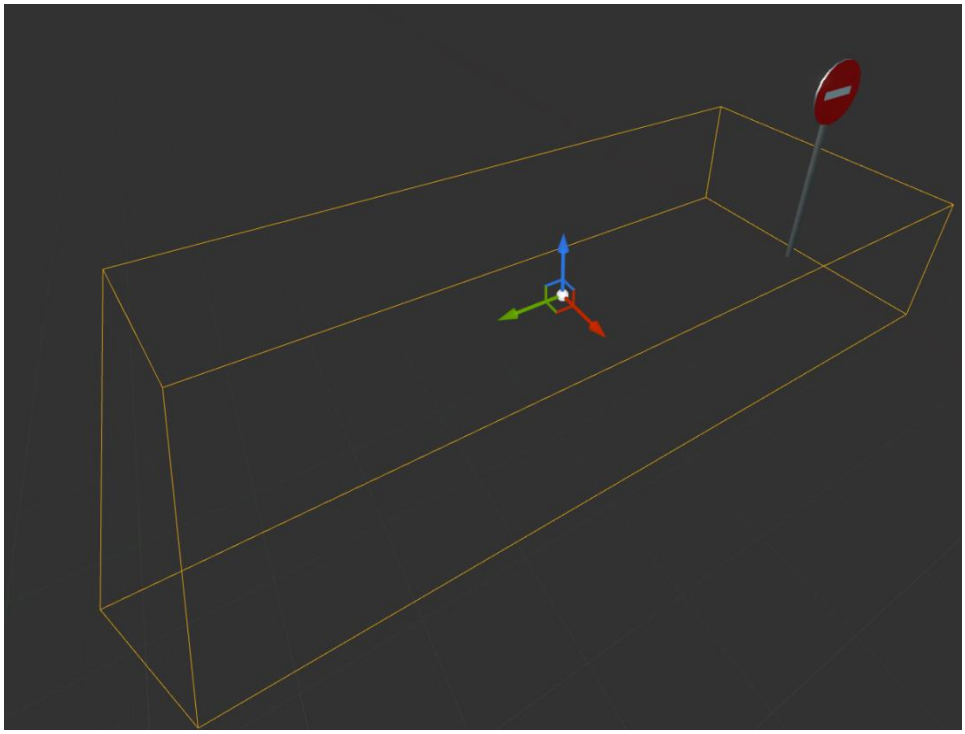
Metoda byla přetížena pro potřeby třídy AGenericVehicle. Kolizní událost přehraje zvukovou komponentu určenou pro náraz vozidla. Před samotným přehráním zvukové stopy je provedena kontrola, zda stopa již nehraje a následně je nastaven parametr „Speed“. Na základě tohoto parametru je upravena intenzita a výška výsledného zvuku. Pokud síla nárazu převyšuje uvedenou hranici je navýšen počet zaznamenaných kolizí.

```
void AGenericVehicle::NotifyHit(UPrimitiveComponent* MyComp, AActor* Other,
UPrimitiveComponent* OtherComp, bool bSelfMoved, FVector HitLocation, FVector HitNormal,
FVector NormalImpulse, const FHitResult& Hit) {

	if (!HitSound->IsPlaying()) {
		HitSound->SetFloatParameter("Speed",
		UKismetMathLibrary::NormalizeToRange(Speed, 0.f, 100.f));
		HitSound->Play();
	}

	if (NormalImpulse.Size() > 150000.f) {
		RegulationComponent->CollisionHits = CollisionHits + 1;
	}

}
```

Obrázek 34: Metoda AGenericVehicle::NotifyHit je invokována v případě zaznamenání styku dvou kolizních komponent.

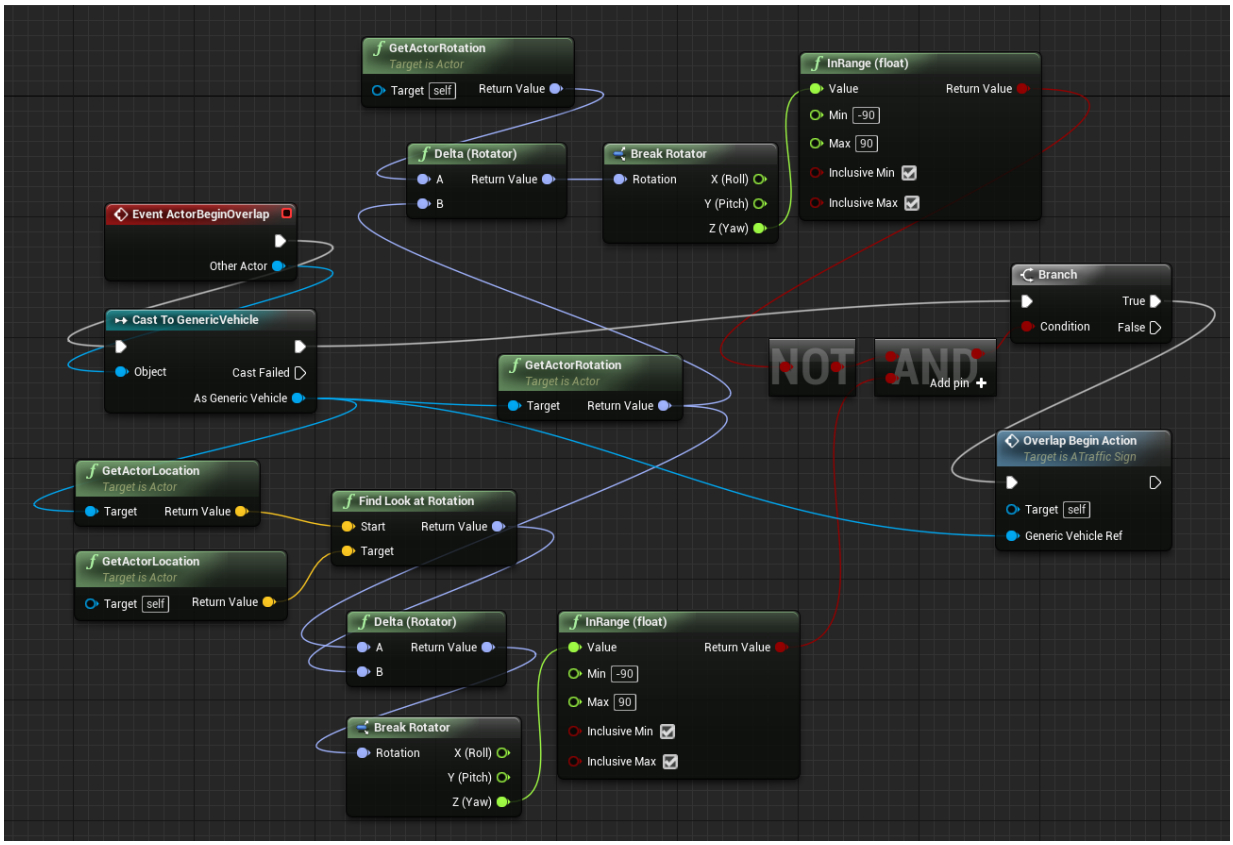
4.3 Systém pro kontrolu pravidel silničního provozu

Systém pro kontrolu pravidel silničního provozu byl jeden z požadavků diplomové práce. Ke splnění toho úkolu bylo přistupováno dvěma způsoby. První způsob by se dal nazvat *„Chytré dopravní značky“*, protože jsou to právě instanční objekty typu ATrafficSign, které jsou zodpovědné za provedení kontroly. Třída ATrafficSign je předkem všech ostatních typů dopravních značek a sdružuje chování, které je pro všechny potomky společné. Každá značka má jednu nebo více komponent, které určují její pole působnosti. Běžně je toto území tvořeno kvádrem, který není v průběhu simulace pro uživatele viditelný. Následující ilustrace znázorňuje pole působnosti jedné z dopravních značek.



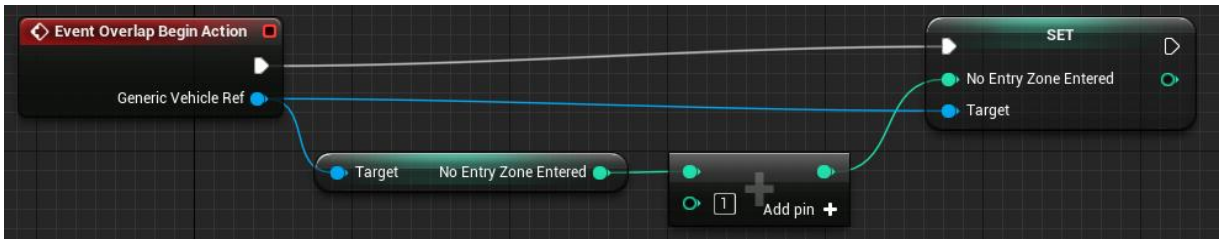
Obrázek 35: Dopravní značení a pole působnosti.

Dojde-li k průniku objektu typu AGenericVehicle a vytyčeného pole, vznikne událost AActor::AActorBeginOverlap, která přijímá jako svůj argument druhého činitele této události. Na následujícím obrázku je ve skriptovacím jazyku BluePrint implementována událost, která vyhodnotí, zda je činitel opravdu typu AGenericVehicle a pokud splňuje další požadovaná kritéria. Mezi kritéria patří například správná rotace vozidla vůči poli působnosti. Pokud jsou podmínky splněny, dojde následně k invokování metody ATrafficSign::OverlapBeginAction. Podobným způsobem je navržena událost AActor::AActorOverlapEnd, která vzniká v případě, že objekt opustil pole působnosti dané značky.



Obrázek 36: Metoda ATrafficSign::AActorBeginOverlap

Invokované metody OverlapActionBegin a OverlapEndAction jsou definovány jako abstraktní a opět přijímají účastníka události typu AGenericVehicle jako argument. Tento návrh má za následek jednoduchou rozšiřitelnost, neboť stačí u derivované třídy pouze přetížit příslušné metody a k jejich volání dojde pak již automaticky. Jako příklad lze uvést třídu ANoEntry, která slouží jako předek pro všechna dopravní značení zákazu vjezdu. Na následující ilustraci je uvedena implementace metody. Jinými slovy, vjede-li automobil do pole působnosti takové značky, přičti hodnotu jedna na jeho počítadle.



Obrázek 37: Metoda ANoEntry::OverlapBeginAction navýší o 1 počítadlo vedené v objektu typu AGenericVehicle

Druhá část systému je řešena uvnitř objektů typu AGenericVehicle. Je to z důvodu, že ne všechna pravidla silničního provozu mohou být kontrolována uvnitř pole působnosti dopravních značek. Příkladem může být překročení maximální povolené rychlosti, které je nutno hlídat všude. Stejným způsobem je řešeno zaznamenávání kolizí dopravního prostředku s objekty okolního prostředí. Tyto konkrétní příklady jsou řešeny uvnitř metody AGenericVehicle::Tick, která byla více popsána v kapitole *Logický model vozidla*.

Kontrola některých dopravních předpisů kombinuje oba přístupy. Dopravní značka „Stůj, dej přednost v jízdě!“ při vzniku události ATrafficSign::AActorBeginOverlap spustí na objektu vozidla kontrolu, která hlídá zastavení vozidla. Běžící kontrola se aktualizuje v každém snímku na objektu vozidla a porovnává jeho vnitřní stavy. Opustí-li objekt pole působnosti, vznikne již zmíněná událost ATrafficSign::AActorOverlapEnd, která invokuje metodu AGenericVehicle::StopSignVerify, a ta ukončí kontrolu a zaznamená výsledek.

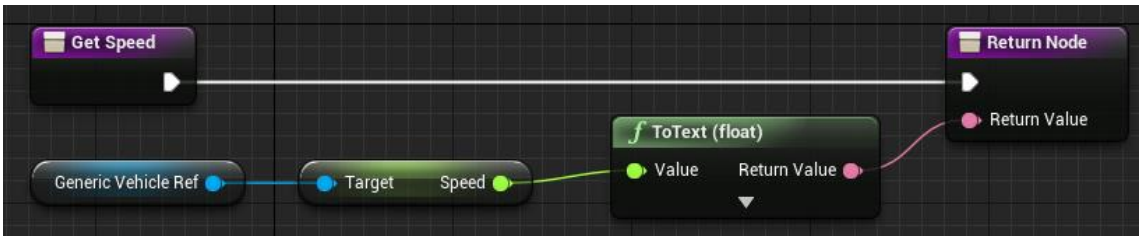
```
void AGenericVehicle::StopSignVerify() {
	if ((!bHasStopped) && (bCheckingIfStopped == true)) {
		StopSignsIgnored = StopSignsIgnored + 1;
	}
	bCheckingIfStopped = false;
	bHasStopped = false;
}
```

Obrázek 38: Metoda AGenericVehicle::StopSignVerify ukončuje běžící kontrolu a zaznamenává výsledek

4.4 Uživatelské rozhraní

Uživatelské rozhraní je nezbytnou součástí každé aplikace, která má umožnit uživateli interakci s běžícím procesem a poskytnout výstup pro důležité informace. K tvorbě uživatelského rozhraní byl využit nástroj, který je integrovaný přímo ve vývojovém prostředí UE4 – Editor s názvem *UMG UI Designer* (v angl.: Unreal Motion Graphics User Interface Designer, dále jen UI Designer). UI Designer nabízí tvorbu jednotlivých prvků rozhraní ve vizuální podobě a úzce spolupracuje se skriptovacím jazykem BluePrint. Připravené UI je rozděleno do několika základních widgetů[15].

Je důležité zmínit, že UI nepřidává žádnou vlastní logiku, která by nebyla přímo spjata s jeho způsobem vykreslování. Zobrazené informace jsou řešeny interní logikou jiných objektů, a to převážně uvnitř AGenericVehicle. Objekt UI je unikátní pro každého uživatele, a proto je možné i zpětně z kontextu objektu UI zjistit objekt uživatele. Vzhledem k tomu, že každý uživatel má vlastní objekt typu APawn, který jej reprezentuje v prostředí, není obtížné se z UI dostat k referenci na daný objekt. V tomto případě je ovládaný objekt typu APawn zároveň typu AGenericVehicle. Widgety tak pouze zpřístupňují veřejné atributy referenčních objektů a starají se o jejich zobrazení.

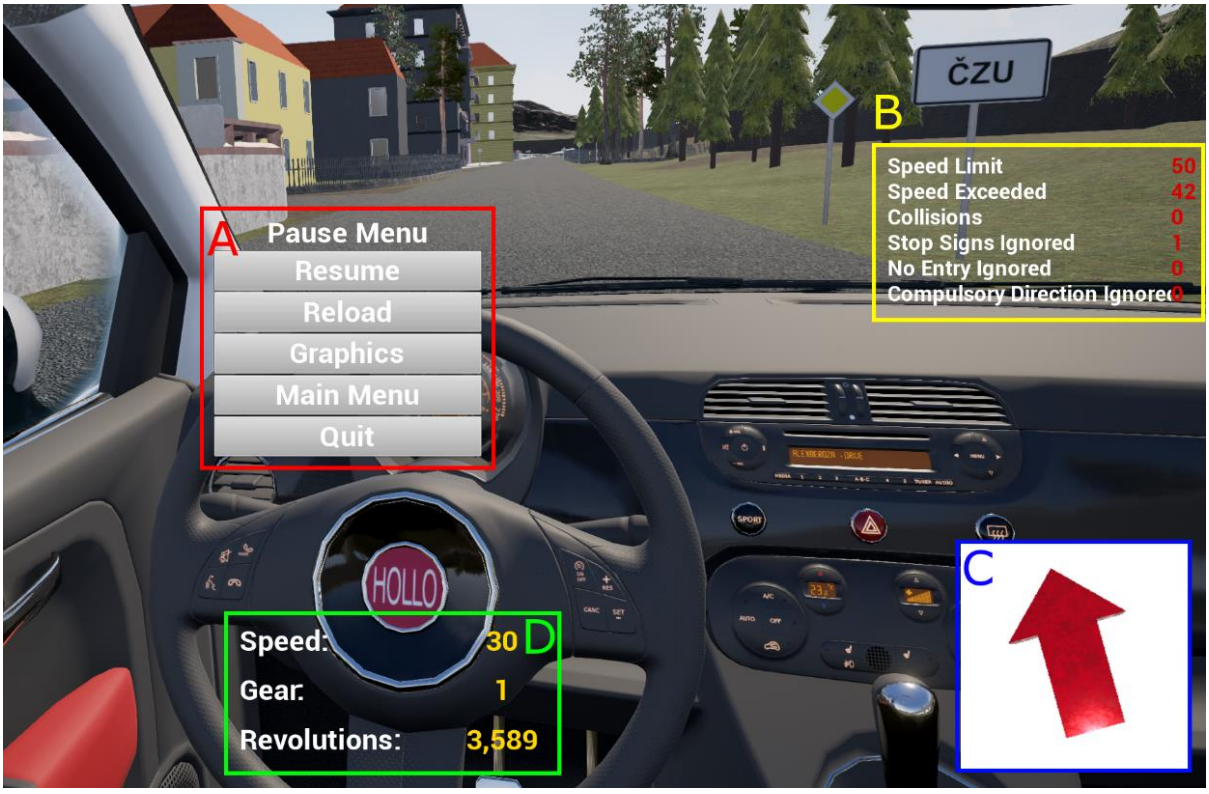


Obrázek 39: Metoda UI pro zjištění rychlosti referenčního objektu typu AGenericVehicle.

[15] Prvky uživatelského rozhraní se nazývají v terminologii UE4 widgety.

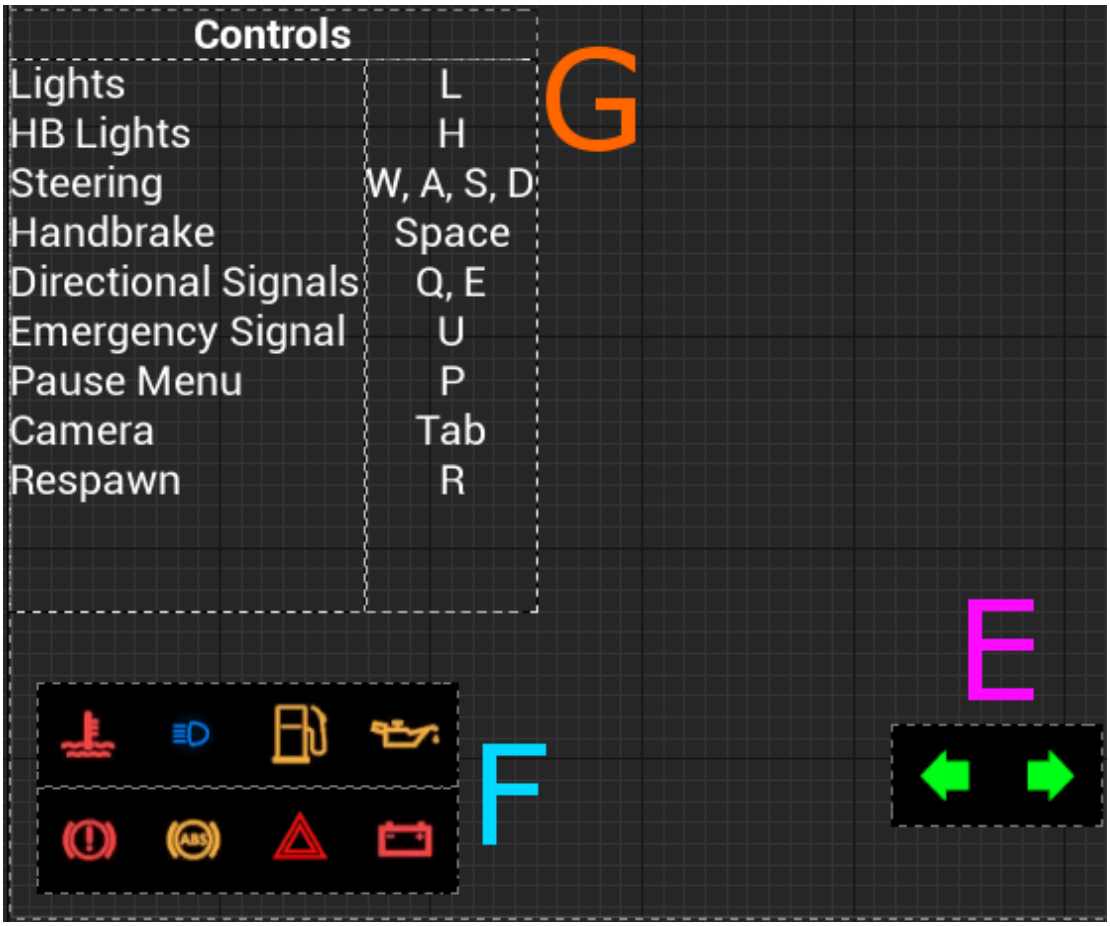
Umístění a vzhled všech widgetů je možno pozorovat na následujících ilustracích.

A. Nabídka pozastavení hry je prvek, který je využíván k základnímu pohybu v aplikaci. Umožňuje výběr grafického nastavení, rozlišení displeje a výběr mapy.
B. Informační panel o dodržování pravidel silničního provozu realizuje zpětnou vazbu pro uživatele. Na základě tohoto výstupu může být řidič ohodnocen.
C. Směrový indikátor ukazuje umístění kontrolního bodu, který je po dosažení přemístěn. Vykreslovaná šipka aktualizuje svůj směr v závislosti na rotaci a pozici vozu a pozici kontrolního bodu.
D. Informace o rychlosti, převodovém stupni a otáčkách za minutu automobilu jsou aktualizované v každém snímku.



Obrázek 40: Uživatelské rozhraní. A - hlavní nabídka, B - pravidla silničního provozu, C - směrový indikátor, D – rychlost, převodový stupeň, RPM

E. Kontrolky pro směrová světla jsou umístěny ve středu displeje.
F. Ostatní kontrolky vozidla jsou zobrazeny v levém spodním rohu displeje. Viditelné jsou jen v případě, že se objekt vozidla dostal do příslušného stavu.
G. Panel klávesových zkratek informuje uživatele o možnostech ovládání. V Současné době není možné klávesové zkratky měnit.



Obrázek 41: E - kontrolky směrových světel, F - obecné kontrolky, G - panel klávesových zkratek

Klávesové zkratky slouží pro nenáročnou navigaci napříč aplikací a řízení automobilu. Následující seznam klávesových zkratek je přístupný pouze v režimu simulace. V kontextu hlavní nabídky je aplikace řízena pouze kurzorem polohovacího zařízení.

- W, A, S, D alternativně klávesy šipek – řízení automobilu
- C – otevření/zavření informačního panelu klávesových zkratek
- Mezerník – ruční brzda
- Q, E – zapnutí/vypnutí směrových světel
- L – zapnutí/vypnutí osvětlení vozu
- H – přepínání dálkových/potkávacích světlometů, je nutné nejprve zapnout osvětlení
- U – zapnutí/vypnutí varovných světel
- P, ESC – vyvolání hlavní nabídky
- Tabulátor – přepínání režimu kamery
- Polohovací zařízení (osy X, Y) – otáčení kamery
- R – resetování polohy a rotace automobilu

5 Výsledky a diskuse

5.1 Splnění dílčích cílů práce

Jedním z dílčích cílů práce byla příprava grafických a zvukových prostředků, které budou využity v prostředí UE4. Z důvodů neočekávané časové náročnosti tohoto kroku, byly využity některé grafické a zvukové prostředky, které jsou dostupné v základní knihovně UE4 – editor. Jeden model vozidla byl zakoupen z komunitního trhu UE4. Zbytek prostředků byl vytvořen v softwarech třetích stran: Blender, 3ds Max, Photoshop a Gimp a následně zkompletován v UE4. Do množiny vlastních výrobků patří několik budov, desítky dopravních značek, oplocení, model vozovky atd. Motorové vozidlo je částečně animováno. Uživatel může zaznamenat otáčení volantu, které je závislé rotaci předních kol. Pozice ručiček tachometru a otáčkoměru odpovídají příslušným hodnotám. Výsledný estetický vzhled použitých prostředků předčil očekávání.

Tvorba krajiny proběhla v softwaru UE4 – Editor, který skýtá vlastní sadu nástrojů určenou pro tvorbu map. Krajina se rozprostírá na půl čtverečním kilometru a má nepravidelný reliéf. Vytvořeny byly krajinné útvary jako jsou kopce a prohlubně. Připravené materiály byly aplikovány na povrch krajiny a mají vlastní fyzikální vlastnosti. Umisťování grafických prostředků proběhlo v UE4 manuálně i plošně s využitím náhodného generování. Celková délka silniční sítě přesahuje pět kilometrů.

Klíčovým cílem diplomové práce bylo vytvořit logický model, který bude řídit chování osobního vozidla. S využitím programového rozhraní UE4 bylo dosaženo chování, které napodobuje reálné jízdní vlastnosti automobilu. Vozidlo dokáže akcelerovat, brzdit a zatáčet. Fyzikální engine UE4 přenáší na automobil působení vektorových sil a vozidlo je tak podrobeno vlivu gravitace a setrvačnosti. Vybrané objekty si rovněž dokáží předávat sílu navzájem. Implementovaná převodovka umožnuje motoru převod mezi pomyslnými převodovými stupni. Osvětlení automobilu je dosaženo připojením světelných komponent. Mezi osvětlení patří čtyři směrová světla, dva potkávací a dva dálkové světlomety. Uživateli je umožněno přepínat osvětlení dle potřeby. Logický model vozu splnil požadavky.

Kontrola dodržování dopravních předpisů je nezbytnou zpětnou vazbu pro uživatele. Připraven byl systém, který dokáže monitorovat některé aspekty pravidel silničního provozu.

Kontrolováno je překročení maximální povolené rychlosti, které je reprezentováno celkovým časem, při kterém uživatel jel rychleji, než by měl. Navrženo bylo *chytré dopravní značení*, které dokáže samo vyhodnotit, zda se kolemjedoucí vozidlo provinilo proti kýženému chování. Mezi tyto dopravní značky patří například značka „Stůj, dej přednost v jízdě!“, která vyžaduje úplné zastavení projíždějícího automobilu. Jiné značky zakazují vjezd, nebo přikazují směr jízdy a při jejich porušení je uživatel upozorněn. Připravený systém zároveň zaznamenává kolize automobilu s jinými objekty.

Uživatelské rozhraní bylo připraveno v softwaru UE4 – Editor a je rozděleno na dvě základní skupiny. Do první skupiny patří kontextové nabídky, které umožňují pohyb napříč aplikací, grafické nastavení a výběr mapy. Druhá skupina uživatelských rozhraní je vykreslována v průběhu simulace automobilu. Patří sem informace o vnitřních stavech vozidla včetně rychlosti, převodového stupně a RPM motoru. Znázorňuje rovněž údaje o dodržování pravidel silničního provozu, které slouží uživateli k ohodnocení jeho výkonu. Pro určení cílové destinace, do které by se měl uživatel vozidlem dopravit, byl navržen indikátor směru. Provedení indikátoru bylo řešeno směrovou šipkou, která je umístěna v pravém dolním rohu displeje. Připravené uživatelské rozhraní splňuje stanovené požadavky.

Škálovatelnosti softwaru bylo dosaženo připravením konfiguračních skupin, které zahrnují čtyři stupňující se grafická nastavená. Při výběru úrovně nastavení dojde ke změně v kvalitě efektů, post-processingu, stínů, rozlišení textur, antialiasingu a k úpravě vykreslované vzdálenosti. Uživatel může zároveň využít tří úrovní pro rozlišení obrazu. Škálovatelnost může být upravena při běhu aplikace v příslušné kontextové nabídce.

Ve výsledném softwarovém balíčku není implementován způsob pro vlastní tvorbu doplňků nebo modulů. Pro úpravu aplikace a přidávání nových objektů a chování by musel budoucí vývojář upravit zdrojový kód, nebo otevřít projekt v softwaru UE4 – Editor. Při vývoji bylo dbáno na dodržení principů OOP, a proto je připravené programové rozhraní jednoduše rozšiřitelné a jeho objekty znovupoužitelné.

5.2 Testování výkonu

Výkon aplikace byl testován na třech odlišných strojích, které se liší ve svých konfiguracích. Výběr strojů proběhl na základě dostupnosti. Při testování byla pozornost zaměřena na výkon aplikace v jejích různých nastaveních. Předmětem zájmu byla veličina počet snímků za vteřinu (v ang.: frames per second, dále jen FPS), kterou bylo možno měřit v prostředí UE4 editor. Pro měření byl použit příkaz `stat fps`, který se zadává do konzole aplikace. Druhým testovacím kritériem byla plynulost uživatelské zkušenosti.

5.2.1 Zařízení Lenovo IdeaPad B590

Prvním testovaným zařízením je notebook od společnosti Lenovo IdeaPad B590. Toto zařízení představuje nízkou třídu na trhu s počítači a jeho kupní cena se pohybovala okolo osmi tisíc korun. Notebook byl výrobcem doporučen ke kancelářskému využití, a proto by neměl být dostatečně výkonný pro vykreslování 3D aplikací v reálném čase.

Lenovo IdeaPad B590	
Procesor	:Intel Core Processor i3-3110M
Operační paměť	4GB DDR3 1600 MHz
Grafický čip	Intel HD Graphics 3000
Pevný disk	HDD 500GB 5400 otáček/s
Platforma	Windows 8

Notebook má znatelné problémy s během aplikace. Integrovaný grafický čip a procesor jsou nedostatečné a příjemné uživatelské zkušenosti s hladkým průběhem není plně dosaženo. Na nejnižší nastavení s rozlišením 1366x768 nepřekročí FPS hodnotu 30. Vyšší úrovně grafického nastavení, se zapnutým dynamickým stínováním a post-processing úpravou jsou nedostupné a FPS klesá i pod hodnotu 1.

5.2.2 Vlastní sestava 1

Druhým testovacím zařízením je vlastní sestava pořízená v roce 2012. Cena všech uvedených komponent se pohybovala okolo šestnácti tisíc korun, a tak spadá do střední třídy.

Sestava 1	
Procesor	Intel Core i5-2500K
Operační paměť	8GB, 2x Kingston DDR3 1333MHz CL9
Grafická karta	MSI N560GTX-Ti-M2D1GD5/OC
Základní deska	GIGABYTE GA-Z77-D3H - Intel Z77
Pevný disk	Kingston HyperX 3K - 240GB
Platforma	PC, Windows 7

Při testování aplikace bylo na všechny nastavení s výjimkou úrovně „Epic“ dosaženo vysokých hodnot FPS, které se ustálily na hranici 59, která je daná nastavením grafické karty V-Sync při snaze synchronizovat obnovovací frekvenci monitoru a aplikace. Při nastavení nejvyšších grafických detailů došlo k razantnímu poklesu výkonu, který se pohyboval v intervalu 20 – 50 FPS.

5.2.3 Vlastní sestava 2

Třetím testovacím zařízením je opět vlastní sestava, která byla zakoupena na konci roku 2016. Kupní cena se pohybuje okolo dvaceti dvou tisíc korun a je na pomezí střední a vyšší třídy stolních počítačů.

Sestava 2	
Procesor	Intel Core i5-6600K
Operační paměť	16GB,2x Kingston HyperX Fury DDR4 2133Mhz
Grafická karta	ASUS GeForce GTX O6G 1060, 6GB GDDR5
Pevný disk	Kingston HyperX 3K - 240GB
Platforma	PC, Windows 10

Vytvořená aplikace byla na všechny úrovně grafického nastavení plynulá. Počet FPS přesahoval hodnotu 100 ve všech konfiguracích s výjimkou nastavení *Epic*, které dosahovalo hodnot v intervalu 60-100.

5.3 Nalezené nedostatky

V průběhu testování aplikace bylo odhaleno několik nedostatků. Příčiny, které měly silný dopad na chování aplikace byly odstraněny. Prvním objeveným problémem byly grafické artefakty, které se vytvářeli za rychle se pohybujícími předměty. Nejzřetelněji mohl být jev pozorován okolo kol vozidla a u jeho obrysů. Jednalo se o vážný nedostatek, který narušoval estetický dojem, a proto došlo k nápravě. Důvodem bylo špatně nastavené pohybové rozmazání (v angl.: motion blur), které slouží k navození pocitu plynulého pohybu a kompenzuje diskrétní povahu vykreslování v jednotlivých snímcích. Vypnutím toho nastavení byl problém odstraněn.

Při nejnižší úrovni grafického nastavení nefungují správně směrová světla. Při přechodu na nejnižší nastavení je proveden příkaz `sg.EffectsQuality = 0;`, který deaktivuje jednu z vlastností UE4 a komponenta světel nemůže využívat přiřazené funkce, které kontrolují způsob osvětlování scény. Směrová světla využívají světelnou funkci, která způsobuje jejich přerušované svícení. Problém nebyl zatím odstraněn.

Dojde-li k vyvolání hlavní nabídky klávesovou zkratkou P nebo ESC, nedostane kurzor myši správný kontext. Uživatel musí provést dvě kliknutí pro potvrzení jakékoliv položky v této nabídce. Příčina problému nebyla odhalena.

Obrázek 42: Chybějící kolizní geometrie u některých typů stromů.

Bylo zjištěno, že některé druhy vegetace nemají svoji kolizní geometrii. Tento nedostatek umožňuje jiným objektům nerušený průchod skrze polygonový model. Při zhodnocení vlivu této chyby byl vyvozen závěr, že nebude provedena oprava. Kolizní komponenta přidává další geometrická primitiva do 3D scény, o kterých si počítač musí uchovávat informace v paměti. Počet těchto stromů v připravované krajině je vysoký a náprava by měla značný dopad na výpočetní náročnost celého prostředí.

6 Závěr

Diplomová práce byla zaměřena na oblast pro vývoj 3D aplikací v reálném čase. Hlavním cílem práce bylo vytvořit *Simulátor dopravních situací pro výuku řízení*. Před samotným započetím vlastní práce došlo ke studii odborných informačních zdrojů, které se této problematice věnují. Z nabytých informací došlo k určení zájmových oblastí pro nadcházející vývoj aplikace. V kapitole *Přehled řešené problematiky* byla zpracována teoretická východiska nezbytná pro pochopení dané tématiky. Předmětem zájmu byla tvorba 3D aplikací v herních enginech se zaměřením na UE4. Dále byl přiblížen proces grafického vykreslování na různých softwarových rozhraních, které jsou využívané v dnešní produkci. Popsány byly metody, které jsou využívány odborníky při tvorbě 3D modelů a způsoby pro úpravu povrchu. Pozornost byla rovněž zaměřena na aspekty soudobých fyzikálních enginů.

Kapitola *Vlastní práce* popisuje kroky vývoje aplikace. Příprava jednotlivých prvků grafických a zvukových prostředků proběhla v softwarech třetích stran a jejich kompletování bylo provedeno v UE4 – Editor. Použité grafické prostředky jsou dostatečné pro potřeby této práce a dobře zachycují své reálné obrazy. Navržený logický model vozidla je schopen dostát vzneseným nárokům a zajišťuje základní i pokročilou možnost jeho ovládání. Implementovaný systém pro kontrolu dodržování pravidel silničního provozu je schopen rozeznat pochybení uživatele. Uživatelské rozhraní je vizuálně jednoduché, ale nabízí dostatečnou množinu vstupů a poskytuje všechny nezbytné informace.

V kapitole *Výsledky a diskuse* bylo zdokumentováno dosažení jednotlivých dílčích cílů, které byly předem stanoveny. Proběhl krátký test aplikace na několika dostupných zařízeních, které reprezentovaly různé výkonnostní kategorie. Překvapením byla vysoká škálovatelnost připraveného softwaru. S nejnižším nastavením dokázala aplikace pracovat i na velmi slabém stroji Lenovo IdeaPad B590, který je výrobcem určený ke kancelářským účelům. Na nejvyšší úrovní grafického nastavení je aplikace esteticky uspokojivá. Během testování softwaru byly rovněž objeveny chyby. Závažné nedostatky byly zpětně odstraněny, a ty přetrvávající nemají zásadní vliv na výslednou funkcionalitu.

Podstata všech dílčích cílů práce byla naplněna. Z tohoto důvodu lze konstatovat, že diplomová práce *Simulátor dopravních situací pro výuku řízení* splnila své zadání.

Seznam použitých zdrojů

[1] Statista, „Value of games industry,“ [Online]. Available: https://www.statista.com/statistics/259455/value-of-games-industry-mergers-and-acquisitions-worldwide/. [Přístup získán 4 7 2016].

[2] NVIDIA, „GPU Applications,“ 3 7 2016. [Online]. Available: http://www.nvidia.com/object/gpu-applications.html. [Přístup získán 3 7 2017].

[3] Epic Games, Inc., „Unreal Engine 4 Documentation,“ 2015. [Online]. Available: https://docs.unrealengine.com/latest/INT/index.html. [Přístup získán 2 6 2015].

[4] NVIDIA, „NVIDIA APEX,“ 2016. [Online]. Available: http://www.nvidia.com/object/apex.html. [Přístup získán 4 3 2016].

[5] NVIDIA, „NDVIDIA FLEX,“ 2016. [Online]. Available: https://developer.nvidia.com/flex. [Přístup získán 4 3 2016].

[6] Epic Games, Inc., „Physics Simulation,“ 2015. [Online]. Available: https://docs.unrealengine.com/latest/INT/Engine/Physics/index.html. [Přístup získán 5 2 2016].

[7] M. Desbrun, P. Schröder a A. Barr, „Interactive animation of structured deformable objects,“ 1999. [Online]. Available: http://www.multires.caltech.edu/pubs/GI99.pdf.

[8] D. Baraf a A. Witkin, „Large Steps in Cloth Simulation,“ v *Computer Graphics Proceedings*, Orlando, 1998.

[9] TechSim Engineering, „Strukturální mechanika FEA,“ [Online]. Available: http://techsim.cz/vypocetni-simulace/strukturalni-mechanika-fea/. [Přístup získán 8 3 2016].

[10] J. Mesit, *Modeling and Simulation of Soft Bodies,* Orlando, Florida: University of Central Florida, 2010.

[11] Unity Technologies, „Unity Manual,“ 2015. [Online]. Available: http://docs.unity3d.com/Manual/index.html. [Přístup získán 3 2 2016].

[12] Blender Foundation, „Blender Reference Manual,“ [Online]. Available: https://www.blender.org/manual/contents.html. [Přístup získán 7 3 2016].

[13] Autodesk, „Hardware vs. Software Rendering,“ 2008. [Online]. Available: http://download.autodesk.com/us/maya/2009help/mr/manual/node57.html. [Přístup získán 24 2 2016].

[14] P. Kovač, „Svět Hardware,“ 23 2 2012. [Online]. Available: http://www.svethardware.cz/graficke-enginy-her-a-realny-svet/18297. [Přístup získán 2 1 2016].

[15] Autodesk Inc., „Autodesk Knowledhe Network,“ 2015. [Online]. Available: https://knowledge.autodesk.com/. [Přístup získán 1 20 2016].

[16] P. Tišnovský, „Grafická knihovna OpenGL,“ 2003. [Online]. Available: http://www.root.cz/serialy/graficka-knihovna-opengl/#ic=serial-box&icc=title. [Přístup získán 12 1 2016].

[17] „OpenGL SoftwareDevelopment Kit,“ Silicin Graphics Inc., 2016. [Online]. Available: https://www.opengl.org/sdk/docs/. [Přístup získán 10 1 2016].

[18] Microsoft, „DirectX Graphics and Gaming,“ 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/ee663274(v=vs.85).aspx. [Přístup získán 1 13 2016].

[19] Open.GL, „OpenGL Application,“ [Online]. Available: https://open.gl/. [Přístup získán 26 2 2016].

[20] E. Russell, „Eliminate Texture Confusion: Bump, Normal and Displacement Maps,“ Digital Tutors, [Online]. Available: http://blog.digitaltutors.com/bump-normal-and-displacement-maps/. [Přístup získán 22 1 2016].

[21] NVIDIA, „NVIDIA Texture Tools for Adobe Photoshop,“ 2016. [Online]. Available: https://developer.nvidia.com/nvidia-texture-tools-adobe-photoshop. [Přístup získán 22 2 2016].

[22] C. Petry, „NormalMap Online,“ [Online]. Available: http://cpetry.github.io/NormalMap-Online/. [Přístup získán 22 2 2016].

[23] J. Zink, „gamedev.net,“ 9 8 2013. [Online]. Available: http://www.gamedev.net/page/resources/_/technical/graphics-programming-and-theory/a-closer-look-at-parallax-occlusion-mapping-r3262. [Přístup získán 2 23 2016].

[24] T. Kaneko, T. Takahei, M. Inami, N. Kawakami, Y. Yanagida, T. Maeda a S. Tachi, „Detailed Shape Representation with Parallax Mapping,“ v *ICAT 2001*, Tokyo, 2001.

[25] Unity Technologies, „Lighting,“ 2015. [Online]. Available: http://docs.unity3d.com/Manual/LightingOverview.html. [Přístup získán 1 12 2015].

[26] Epic Games, Inc., „Lighting the Environment,“ 2015. [Online]. Available: https://docs.unrealengine.com/latest/INT/Engine/Rendering/LightingAndShadows/index.html. [Přístup získán 2 1 2016].

[27] Tech Soft 3D, „HOOPS 3D Graphics System,“ 2016. [Online]. Available: http://docs.techsoft3d.com/. [Přístup získán 28 1 2016].

[28] Khronos Group, „OpenCL,“ 2016. [Online]. Available: https://www.khronos.org/opencl/. [Přístup získán 3 2 2016].

[29] NVIDIA, „CUDA Parallel Computing Platform,“ 2016. [Online]. Available: http://www.nvidia.com/object/cuda_home_new.html. [Přístup získán 3 2 2016].

[30] K. Kadau, T. Germann a P. Lomdahl, „Molecular dynamics comes of age: 320 billion atom simulation,“ *International Journal of Modern Physics C,* 12 12 2006.

[31] K. Heitmann, N. Frontiere, C. Sewell, S. Habib, A. Pope, H. Finkel, S. Rizzi, J. Insley a S. Bhattacharya, „The Q Continuum Simulation: Harnessing the Power of GPU Accelerated Supercomputers,“ *The Astrophysical Journal,* 21 8 2015.

Seznam ilustrací