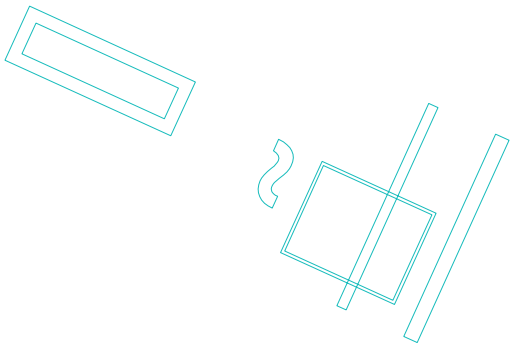
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DEFERRED SHADING

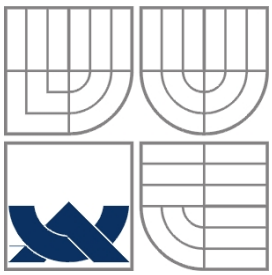


DIPLOMOVÁ PRÁCE
MASTER'S THESIS

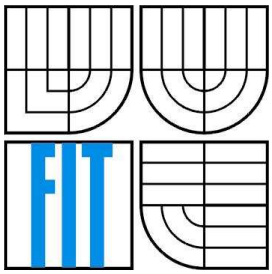
AUTOR PRÁCE
AUTHOR

BC. PETR STARÝ

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DEFERRED SHADING

DEFERRED SHADING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — BC. PETR STARÝ
AUTHOR

VEDOUCÍ PRÁCE — ING. RADOVAN JOŠTH
SUPERVISOR

BRNO 2009

Abstrakt

Práce se zabývá problematikou návrhu a implementace výukového programu pro demonstraci techniky deferred shading a jejich možností. Intuitivní a interaktivní formou se snaží vysvětlit principy osvětlování a stínování pouze viditelných pixelů obrazu na základě atributové mapy vytvořené při rasterizaci geometrie. Zpracování geometrie je tedy kompletně odděleno od samotného procesu stínování.

Klíčová slova

Zpožděné stínování, osvětlení, stínování, metodika výuky, výukový program, vykreslovací řetězec, demonstrace, atributová mapa

Abstract

Work deals with design and implementation a tutorial for demonstration deferred shading technique and its possibilities. It explains lighting and shading principles in intuitive and interactive way. Deferred shading is a technique which determines pixel color after the geometry rasterization of the entire scene. In other words the processing of geometry does not interfere with the shading process.

Keywords

Deferred shading, rendering pipeline, lighting, shading, methodology of education, demonstration, tutorial, g-buffer, geometry buffer, fat buffer

Citace

Starý Petr: Deferred Shading, diplomová práce, Brno, FIT VUT v Brně, 2009.

Deferred Shading

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Radovana Joštha. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Petr Starý
25. května 2009

Poděkování

Děkuji svému vedoucímu Ing. Radovanovi Jošthovi za jeho velmi vstřícný přístup a ochotu, kterou mi projevoval při tvorbě této diplomové práce. Zároveň chci i poděkovat osobám, které vyplnili dotazník.

© Petr Starý, 2009.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod

Tempo evoluce grafických karet je nesmírně vysoké. Každý rok vychází nová generace grafických karet, přičemž každá další generace je mnohem výkonnější a implementuje nové vlastnosti. Díky těmto novým vlastnostem se stejně rychle objevují nové metody a algoritmy vykreslování objektů. Stejně tak se ale vynořují zpoza historie i již vynalezené metody, které tehdy kvůli vysokým technickým nárokům nebylo možné implementovat. Zářným příkladem takovéto techniky je i deferred shading, metoda stínující pouze viditelné pixely. V dnešní době nachází deferred shading své uplatnění jak ve vědeckých vizualizacích, tak i v počítačových hrách.

V této dokumentaci nejprve ještě v další části úvodu popíši motivace, které mne vedli k výběru tohoto zadání, a obecně popíši cíl diplomové práce.

Ve druhé kapitole shrnu získané teoretické poznatky o charakteristice výuky. Klasifikuji jednotlivé metody výuky a několik z nich i popíšu. Na základě těchto teoretických poznatků se pokusím navrhnout systém výuky pro moji diplomovou práci.

Třetí kapitola se již věnuje technice deferred shading. Nejprve obecně popíši architekturu vykreslovací pipeline a poté přejdu k jednotlivým částem pipeline. Od přípravy objektů před vykreslováním, přes samotné vykreslování objektů do G-Bufferu a osvětlení scény na základě uložených atributů až po aplikaci post processing efektů. Vysvětlíme si i použitou terminologii a velkou pozornost budeme věnovat i vysvětlení G-Bufferu a možnostem optimalizace tohoto atributového prostoru.

Celá čtvrtá kapitola je věnována návrhu učební pomůcky. Popíši požadavky kladené na aplikaci z hlediska čistoty návrhu a zároveň i užitných vlastností. Dále si popíšeme návrh grafického uživatelského rozhraní a všech prvků tvořících toto rozhraní. Taktéž se budeme věnovat návrhu jednotlivých scén zobrazených uživateli, přičemž si každou scénu popíšeme a zevrubně vysvětlíme. V neposlední řadě by měl být součástí učební pomůcky i dotazník ověřující její užitné vlastnosti, tudíž i tato oblast bude popsána v samotném návrhu.

Pátá kapitola popisuje implementaci vycházející z návrhu. Jsou zde vysvětleny základní stavební kameny celé aplikace rozdělené do několika kategorií. Každá kategorie obsahuje množinu tříd, jejichž funkcionalitu si zevrubně vysvětlíme. Je zde popsán audio podsystém, princip fungování grafického uživatelského prostředí a v neposlední řadě i Deferred Shader Framework pomocí kterého můžeme velmi snadným způsobem použít techniku deferred shading i v jiných programech.

Poslední kapitolou diplomové práce je závěr. V této kapitole zhodnotím stav diplomové práce, navrhnu možnosti budoucího rozšíření a vyhodnotím vyplněné dotazníky.

1.1 Motivace

Motivací, které mne vedli k výběru tohoto tématu diplomové práce, bylo několik.

První motivací bylo vytvoření učební pomůcky a s tím související nutnost získání teoretických poznatků o správné výuce. Schopnost předávat své znalosti a zkušenosti správným způsobem je velmi důležitá pro jakýkoliv obor a jako taková se setkává s velmi dobrým ohodnocením v zaměstnání.

Další motivací byla touha po implementaci techniky, která se začíná implementovat v počítačových hrách současnosti ve větší míře a jejíž vnitřní architektura vykazuje velmi dobrý potenciál pro budoucí využití grafických karet díky příchodu unifikovaných shader jednotek.

1.2 Cíl práce

Cílem této práce je vytvoření intuitivní učební pomůcky, která by osvětlovala princip vykreslování grafických primitiv pomocí techniky Deferred Shading díky sadě interaktivních a názorných příkladů prokládaných vysvětlujícím textem to celé spojené do jedné aplikace.

Samotná výuka bude rozdělena na několik lekcí popisující jednotlivé části zpožděného vykreslovacího řetězce. Každá lekce může být rozdělena na jednotlivé podsekce kvůli zvýšení přehlednosti celé výuky.

První lekce bude obecným představením samotné techniky popisující terminologii, princip činnosti, výhody a nevýhody použití.

Dále bude vysvětlen architektonický rozdíl mezi klasickým dopředným vykreslovacím řetězcem a zpožděným vykreslovacím řetězcem. Do této lekce též patří vysvětlení rozdílu v toku dat a reprezentaci těchto dat.

Poté se posuneme do části vykreslování geometrie do atributového prostoru a detailně popíšeme atributový prostor pojmenovaný jako G-Buffer.

V pořadí již čtvrtá lekce bude o možných typech světel, osvětlení, stínování a vykreslování stínů za použití informací uložených v G-Bufferu. Poslední lekce se bude věnovat aplikaci post processing efektům, jako jsou například depth of field, bloom filter, edge detection filter a anti-aliasing.

Nyní je nezbytné popsat prostředky použité při implementaci. Celá práce bude implementována v programovacím jazyce C++. Dále je použito přenositelné grafické rozhraní OpenGL pro komunikaci s grafickou kartou, spolu s technologií CG od společnosti nVidia ke zpřístupnění programovatelných shader jednotek na grafické kartě. Pro přístup ke zvukové kartě použiji rozhraní OpenAL.

Na závěr je potřeba se zmínit, že následující dvě kapitole vycházejí ze semestrální práce.

2 Proces získávání informací

„… není, myslím, účelem střední školy, aby absolvent podržel slovíčka a vzorce, nýbrž myšlenkové metody, na kterých to vše visí. Umět – to je dočasné, ale rozumět – to je trvalé obohacení ducha“

Karel Čapek

Od okamžiku narození neustále vnímáme okolní svět a nejprve samotným pozorováním rozpoznáváme, jak věci fungují. O něco později si osvojíme řeč a zavalíme své rodiče otázkami začínající na známé „a proč …?“. Rodiče nám tedy začnou vysvětlovat zákonitosti světa, které nám již nejsou na první pohled tak jasné. Po několika letech začneme chodit do školy, kde tuto roli rodičů nahradí učitelé, kteří mají za úkol nás nejprve naučit číst, psát a počítat. A právě tímto okamžikem dochází ke zlomu, protože se již neučíme poznávat to, co bychom chtěli my, ale co chtějí ostatní.

Výuka je ve své podstatě vzájemné působení mezi učitelem a studentem. Nelze ji proto uskutečnit bez vzájemné snahy mezi oběma subjekty. Studenti samotní musí klást za svůj cíl osvojení si dané problematiky a učitelé se musí snažit své vědomosti předat co nejoptimálnějším způsobem.

Ve druhé polovině 20. století započal vědecký pokus, který měl za cíl komplexně popsat výuku a systémově vytvořit optimální metodiku výuky napříč všemi obory. Díky tomuto výzkumu se klasifikovali jednotlivé typy a metody výuky nutné pro správný rozvoj znalostí studenta.

2.1 Klasifikace metodik výuky

Jednotlivé metodiky výuky mohou být rozdělený do jednotlivých kategorií vzhledem k charakteru osvojování si informací získaných komunikací mezi učitelem a studentem jak je uvedeno v [1].

Metodika informačně receptivní je nejobvyklejším typem výuky. Jedná se o předávání již známých informací formou ústního výkladu, psaného textu, pomůcek, či praktických ukázek. Od studenta se v tomto případě požaduje, aby věnoval předávání informací pozornost, přičemž se souběžně snažil pochopit a zapamatovat předávanou informaci. Příkladem může být například vysvětlení učitele, jakým způsobem se řeší kvadratické rovnice.

V návaznosti na předcházející metodiku se obvykle realizuje reproduktivní metodika. Existuje několik forem reprodukce poznatků. Učitel může ústně zopakovat již jednou naučenou látku z jiného úhlu pohledu, případně se řeší učitelem vytvořená sada cvičení. Osvojený poznatek tak dokáže student aplikovat na základě předem definovaného vzoru. Příkladem budiž neustálé řešení kvadratických rovnic v hodinách matematiky.

Metodika založená na vědeckém přístupu k řešení problému se nazývá metodika problémového výkladu. Učitel nastolí určitý problém a postupně ho sám řeší. Studentům vysvětluje své myšlenkové

postupy a vztahy mezi jednotlivými přechody. Tato metodika klade velké nároky na samotné studenty, protože studenti musí aktivně sledovat učitelův výklad a logiku jeho výroků, stejně jako kontrolovat jeho přesvědčivost. V případě jakýchkoliv pochybností musí být vznesena otázka, protože nepochopení každého kroku vede k neosvojení si celé problematiky a v budoucnosti ke špatným návykům při učení.

2.2 Klasická forma sdělování informací

Existuje několik forem sdělování informací směrem od učitelů ke studentům. Nejznámější formou sdělování informací na základních a středních školách je vysvětlování. Na vysoké škole již převládá forma přednášek. Doplňující a velmi užitečnou formou sdělování informací je názorná demonstrace.

Každá z těchto forem má svá specifika a postupy jak správně předávat informace.

2.2.1 Vysvětlování informací

Vysvětlování je dodání takových informací, které studenty dovedou k pochopení příčin, důsledků a podstaty daného problému. Je kladen velký důraz na úsudek studenta, jeho analytické a logické myšlení, schopnost zobecňování. Učitel musí jednotlivé informace předkládat postupně, systematicky a logicky. Je potřeba též udržovat kontakt se studenty a v případě potřeby okamžitě reagovat na podněty ze strany studentů. Každé vysvětlení by mělo splňovat několik charakteristik, aby bylo lépe pochopeno.

- Musí být postaveno na znalostech, které již studenti mají
- Musí být předkládáno přesvědčivě a trpělivě
- Musí být dostatečně jednoduché i za cenu vynechání důležitých, ale složitých detailů
- Musí obsahovat pouze informace nutné pro jasný a logický popis

Rozdíl vysvětlování oproti přednášení je větší míra bezprostřednosti mezi učitelem a studentem. Tato bezprostřednost se projevuje mnohem větší možností řídit postup myšlenkových operací studentů, účinně pokládat rozšiřující otázky kdykoliv během výkladu a přistupovat ke každému studentovi individuálně dle schopností každého z nich.

Při samotném vysvětlování je potřeba nejprve začít konkrétními případy, zvlášť pokud se snažíme vysvětlit abstraktní pojem. Schopnost studenta pracovat správně s abstraktním pojmem nemusí znamenat, že je schopen onen abstraktní pojem i vysvětlit. Je důležité si uvědomit, že informace přijímané formou vysvětlování nám aktivizují levou hemisféru mozku. Abstraktní pojmy se nám ale formují v pravé hemisféře a bez konkrétních příkladů nejsme schopni tyto abstraktní

pojmy správně asociovat a zařadit. Tím se též vysvětluje, proč není student schopen vlastními slovy vysvětlit pojem, kterému dosud úplně neporozuměl.

Použití analogie, přirovnání a dovedení informace ad absurdum zvyšuje schopnost studentů zapamatovat si informace a nalézt vztahy mezi již známou věcí a novou informací, i když samotná analogie či přirovnání může silně pokulhávat.

Dovedení informace ad absurdum je technika používaná pro odnaučení špatně naučené informace tím, že demonstrujeme postup použití informace tak, aby vedla k nepravdě.

2.2.2 Přednášení informací

Přednášení informací neboli přednáška představuje nejnáročnější formu předávání informace pro studentyy vůbec. Student se musí po celou dobu výkladu dokonale soustředit na přednášené informace a zároveň musí udržovat myšlenkový kontakt s učitelem. Přednáška nepočítá s aktivitou studentů při výkladu a převážně pouze předkládá již utříděná fakta, takže student ani nemá tendence nad danými informacemi nadále přemýšlet a zpracovávat je. I když je na přednášce studentům dovoleno se dotazovat učitele i v průběhu výkladu, jenom málo studentů tuto možnost využije. Navíc je doba, po kterou jsou schopni se soustředit kratší než u jiných technik z důvodu velkého množství informací získaných během krátkého časového období.

Přednášení též klade velké nároky na učitele. Učitel nesmí být při výkladu nudný a používat monotónní řeč. Díky neexistenci zpětné vazby se od učitele vyžaduje dodatečné ujištění, že studenti výkladu porozuměli. Začínající učitelé též mají sklon předávat informace příliš rychle.

Výhodou přednášek je rychlý postup v probírané látce. Dále tato forma předávání informací učí studenty lépe porozumět mluvenému slovu a cvičí je v pozornosti, soustředění se a ve výběru a zachycení právě těch důležitých myšlenek ve výkladu.

2.2.3 Názorná demonstrace informací

Názorná demonstrace je návrat do dětských let, kdy jsme získávali informace o světě na základě pozorování. Studenti touto cestou získávají pravdivé poznatky opírající se o přímé poznávání skutečnosti. Tato technika se dá použít pouze pro omezený počet nejrůznějších jevů. V okamžiku, kdy není možné jednoduchým způsobem demonstrovat daný jev je skutečnost nahrazena za kresby, schémata, či počítačové modely. Takto vytvořená objektivní realita je poté reprezentována pomocí nejrůznějších modelů představující systémy vztahů a souvislostí popisujících jev.

Zajímavou technikou tohoto typu výuky je demonstrace daného jevu, aniž by učitel studentům vysvětloval probíhající demonstraci. Studenti jsou nuceni se usilovně soustředit a pečlivě pozorovat demonstraci, aby poté byli schopni odpovídat na učitelovi otázky ohledně demonstrace.

Rozmach informačních technologií nyní umožňuje demonstrovat nejrůznější jevy studentům interaktivní formou v bezpečném a kontrolovaném prostředí. Studenti mohou na pokyny učitelů upravovat jednotlivé vstupní parametry jevu a okamžitě vidět odezvu na vytvořeném modelu.

2.3 Moderní forma sdělování informací

Poslední dobou se objevuje nová skupina metodik, které se snaží využívat všech dostupných informačních kanálů neotřelým způsobem. Obecnou snahou těchto metodik je poskytovat studentům co nejširší škálu informačních kanálů, kterými lze informace získávat. Dále se snaží cíleně vyvolávat situace, kde tyto informace naleznou své uplatnění a tím pomáhají k trvalému osvojení si takto získaných informací.

Autoři knihy ‚Klíčové kompetence a jejich rozvíjení' [2] rozdělují tyto moderní techniky výuky do pěti skupin, kde několik nejvýznamnější popíšu níže. Je pravdou, že tyto nové vyučovací metody vyžadují intenzivní práci učitelů a nekonvenční myšlení, na druhou stranu jejich největší přínos lze nalézt v přinesení radosti z učení jak studentům, tak i samotným učitelům, což má za následek mnohem větší úspěšnost pochopení nových poznatků a i jejich trvalejší osvojení si oproti tradičním metodám výuky.

2.3.1 Informativní metodika

Jak již z názvu vyplývá, snahou této metodiky je přeměnit informace na znalosti. Znalost není nic jiného než pochopená, dalo by se říci zpracovaná, informace. Tradiční informativní metody předkládají informace způsobem, který se zaměřuje převážně na levou hemisféru mozku. Levá hemisféra mozku je zodpovědná za logické a symbolické myšlení, řeč a lineární úvahy. Intuitivní myšlení, fantazie, celostní úvahy a tušení je doménou pravé hemisféry mozku. Moderní informativní metodika se snaží předkládat informace způsobem, aby byly zapojeny obě dvě hemisféry mozku propojením řečového a obrazového myšlení. Při předávání informací touto formou jsou kombinovány schopnosti celého mozku a dochází ke zbystření paměti, zvýšení koncentrace a získání celkového přehledu rychlejším způsobem.

Kvůli změně formy výuky je potřeba věnovat zvláštní pozornost následujícím bodům a zároveň se snažit tyto body dodržet při výuce.

- **Názornost**: informace se opírají o zkušenosti studentů a jsou názorné
- **Jednoduchost**: čím obtížnější téma, tím jednodušší by měla být řeč
- **Vlastní činnost**: musí být vyvolána aktivita u studentů, vést je ke kladení otázek
- **Jasnost**: je potřeba vybrat několik ústředních pojmů kvůli přehledu
- **Živost**: učitel musí informace předkládat z několika různých směrů

- **Soulad**: informace by měli mít obecnou platnost a obstát před novými vědeckými poznatky
- **Využitelnost**: informace musí být pro studenty použitelná

Příkladem metod aplikujících informativní metodiku může být audiovizuální přednáška, kvízové otázky, dohledávání informací v médiích či jedna z nejnovějších metod myšlenkového mapování.

2.3.2 Narativní metodika

Narativní metodika je založena na vyprávění. Hlavním komunikačním kanálem zprostředkovávající informace je řeč, jejímž prostřednictvím se mají studenti naučit slovně vyjadřovat a slovy adekvátně vystihnout různé situace. Studenti se učí strukturovat informace a ty poté prezentovat ostatním. Díky tomuto přístupu se zvyšuje kompetence ústního jednání studentů. Zároveň jsou schopni se věcně a bez řečnických problémů tázat na postoje a mínění ostatních, oceňovat postoje ostatních a zároveň zdůvodňovat svoje vlastní názory přesvědčivým způsobem. Nezanedbatelnou výhodou je, že studenti aplikací této metodiky nevědomky získávají správnou řeč těla.

Příklady mohou být práce v malých skupinkách a následné prezentace výsledků, besedy s odborníkem, různé diskusní metody, případové analýzy, či techniky sněhové koule. Nejzajímavější metodou aplikace narativní metodiky je právě technika sněhová koule, která předkládá studentům určitý problém. Každý student individuálně zpracuje posudek a poté svůj názor konzultuje ve dvojici s někým jiným, přičemž se snaží nalézt společný názor. Jakmile dvojice zaujme svůj společný názor, vyhledají další dvojici a situace se opakuje. Tato metoda se velmi často používá u výběrových řízení na pozice vedoucích pracovníků, či konzultantů právě díky odhalování schopnosti argumentace.

2.3.3 Intuitivní metodika

Intuitivní metodika oslovuje, jak již z názvu napovídá pravou hemisféru mozku zodpovědnou za intuici. Tato metodika byla velmi dlouho dobu opomínána, čímž jsme mohli přijít o velké množství kreativních postupů k vyřešení problémů. Řešení problémů není založeno na znalostech či zkušenostech, ale dochází k němu na základě pocitů, či vnitřních vnuknutí.

Cíleným uplatňováním intuitivních metod pomocí správné hudby a obrazů se studenti mohou dozvědět nové věci sami o sobě, o svém rozpoložení a zároveň mohou objevit dosud neznámé mechanismy řešení tím, že se učí pozorněji naslouchat svým vnitřním hlasům a účelně jich využívat při hledání řešení.

Mezi známé metody patří brainstorming, meditace, relaxační poslech hudby či vizualizace. Příkladem této intuice v programování může být špatný pocit ze zdrojového kódu s tím, že to musí jít nějak lépe. Při zvládnutí intuitivní metodiky je programátor schopen mnohem rychleji přijít na elegantnější řešení než v případě programátora bez znalostí této metodiky.

3 Deferred shading

Deferred shading v překladu znamená odložené stínování. Tento název přesně popisuje princip fungování této techniky. Odložením se myslí posunutí výpočtu osvětlení a stínování ve vykreslovacím řetězci až za část zodpovědnou za vykreslení geometrie. Nejprve vykreslíme celou scénu do pomocné struktury nazvané G-Buffer (detailně popsané v kapitole 3.3) a poté na základě informací z tohoto G-Bufferu aplikujeme osvětlení a stínování pouze na viditelné pixely scény.

3.1 Motivace

Deferred shading se stal velmi atraktivním prostředkem pro zvýšení vizuálního dojmu i pro herní vývojové týmy. Existuje několik projektů, které kombinují možnosti klasických vykreslovacích metod s metodou deferred shading. Her založených pouze na metodě deferred shading již ale také existuje několik.

S.T.A.L.K.E.R [4] je střílečka z první osoby, která se odehrává na území Černobylu. Jedná se o první hru, která implementovala techniku deferred shading s pokročilými vlastnostmi a tím ukázala cestu i všem ostatním. Hra byla několikrát odložena, přičemž v pokročilé fázi vývoje se tým rozhodl vyměnit dosavadní klasické stínování za deferred shading. Vzhledem k velmi detailní geometrii ve hře a použití velkého počtu dynamických stínů docházelo k bottle-necku na straně vertex shaderů, přičemž právě deferred shading je cestou jak velmi výrazně ulehčit vertex shader jednotkám a použít detailnější geometrii scény bez významného vlivu na čas strávený při výpočtu osvětlení scény.

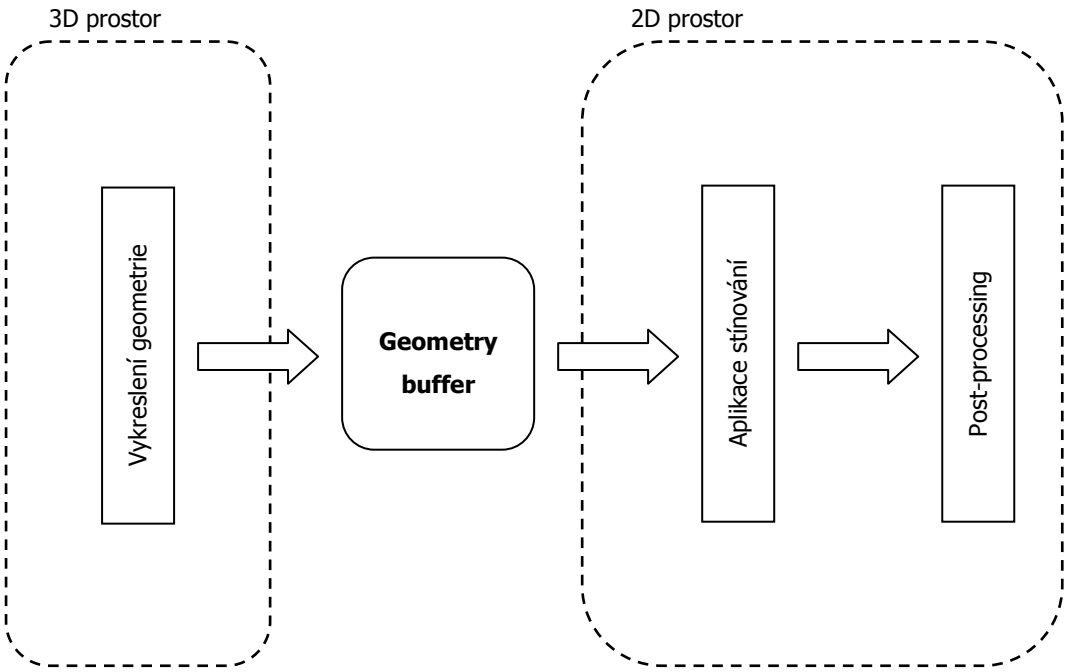
Obrázek 3.1: Snímky ze hry S.T.A.L.K.E.R. ukazující vizuální rozdíl po změně metody stínování. Na pravé straně je snímek získaný při použití techniky deferred shading.

Na základě úspěchu hry S.T.A.L.K.E.R. se i další studia rozhodla využít deferred shading jako primární zobrazovací metodu. Další významný dokončený titul je online hra Tabula Rasa [5]. Velké

naděje na vysokou vizuální kvalitu se též upírají na titul Killzone 2 pro herní konzolu Playstation 3 využívající právě deferred shading [6].

3.2 Vykreslovací řetězec

Architektura opožděného vykreslovacího řetězce lze snadno rozdělit do několika na sobě nezávislých fází, jak můžeme vidět na obrázku 3.2. Fáze první vykreslí geometrii scény a atributy uloží do G-Bufferu. Druhá fáze je zodpovědná za stínování jednotlivých pixelů a poslední nepovinná fáze umožňuje aplikovat post-processing efekty. Důležitým poznatkem je, že naplněním informací o scéně do G-Bufferu jsme se z objektového prostoru posunuli pouze do obrazového prostoru, ve kterém provádíme všechny další operace.



Obrázek 3.2: Schéma zpožděného vykreslovacího řetězce

Porovnáním tradičního dopředného vykreslovacího řetězce s opožděným vykreslovacím řetězcem získáme lepší přehled o celkové funkčnosti techniky deferred shading. V případě dopředného vykreslovacího řetězce s per-pixel stínováním rozlišujeme dva případy.

První případ je zpracování osvětlení v jednom kroku a druhý případ je zpracování osvětlení v několika krocích. Zpracování osvětlení v jednom kroku se realizuje pomocí jednoho fragment shader programu, kterému předáváme informace o světlech. Tento typ se používá v případě menšího počtu světel ve scéně, přičemž každé další přidané světlo úměrně zvětšuje velikost fragment shader

programu.

Pro každý objekt:

Vykresli objekt a aplikuj všechna světla najednou

Výpis 3.1: Výpočet osvětlení v jednom kroku

Druhým případem je zpracování osvětlení v několika krocích, kdy každý krok představuje výpočet osvětlení jedním světlem. Geometrie je tedy vykreslována tolikrát kolik máme světel na scéně. Z optimalizačních důvodů se proto snažíme vykreslovat pouze geometrii, která je ovlivněna daným světlem, což si ale žádá velké nároky na procesor. V nejhorším případě provedeme `pocet_svetel * pocet_objektu` operací při výpočtu osvětlení.

Pro každé světlo:

Pro každý objekt ovlivněný světlem:

Vykresli objekt a přičti výslednou barvu do framebufferu

Výpis 3.2: Výpočet osvětlení v několika krocích

Odložený vykreslovací řetězec naproti tomu pro výpočet osvětlení v nejhorším případě vyžaduje pouze `pocet_svetel + pocet_objektu` operací. Ideální situací pro využití této přednosti u techniky deferred shading je velký počet lokálních světel.

Pro každý objekt:

Vykresli objekt do několika atributových map

Pro každé světlo:

Aplikuj osvětlení formou 2D post-procesu

Výpis 3.3: Výpočet osvětlení metodou deferred shading

3.3 Geometry buffer

Geometry buffer neboli G-Buffer si můžeme představit jako sadu 2D textur. Každá textura poté ve svých pixelech ukládá různé informace o geometrii scény. G-Buffer je tedy mezičlánek spojující krok vykreslení geometrie s krokem výpočtu osvětlení. Při vykreslování geometrie dochází k zápisu

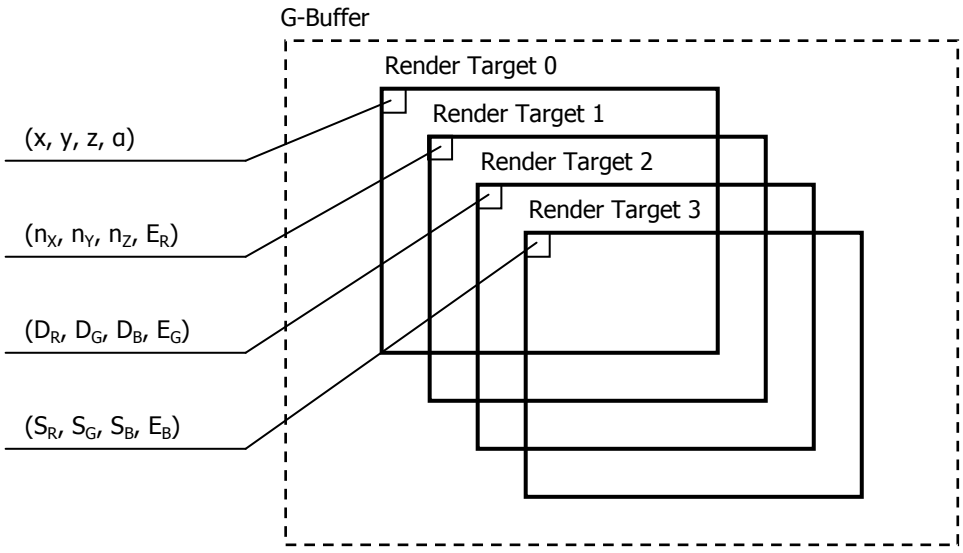
informací do G-Bufferu. Informace uložené v G-Bufferu jsou poté použity v dalším kroku při výpočtu osvětlení a stínování. Jedná se o nejdůležitější součást opožděného vykreslovacího řetězce, který významným způsobem ovlivňuje rychlost výpočtu a finální vizuální kvalitu.

G-Buffer zabírá na grafické kartě velké množství paměti a pro své vytvoření vyžaduje od grafické karty několik vlastností:

- Podporu pro floating point textury
- Podporu pro floating point blending
- Podporu vykreslování do několika cílů najednou (MRT)
- Podporu shader modelu 2.0

Požadavek na podporu MRT není úplně oprávněný. Ve své podstatě stačí, pokud grafická karta podporuje akorát off-screen rendering. Pro vytvoření G-Bufferu je ale poté potřeba vykreslit celou scénu několikrát, což rapidně zvyšuje časovou náročnost a deferred shading se stává pro komplexnější scény prakticky nepoužitelným.

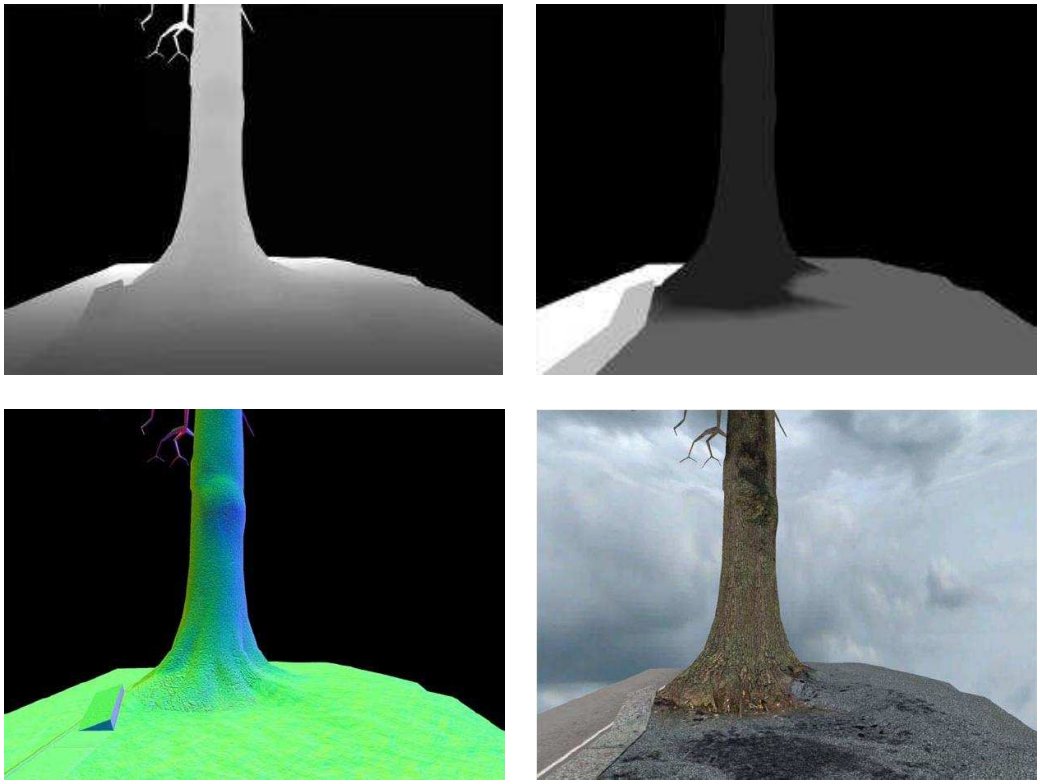
Na obrázku 3.3 můžeme vidět příklad G-Bufferu tvořeného čtyřmi texturami, kde každá složka pixelu textury je číselného formátu R32f (pixel poté zabírá 128 bitů paměti). První textura obsahuje informace o pozici a α materiálu (konstanta odlesků), druhá textura obsahuje normálu a červenou složku vyzářeného světla, třetí textura obsahuje difúzní barvu materiálu a zelenou složku vyzářeného světla a poslední textura obsahuje spekulární barvu materiálu a modrou složku vyzářeného světla. Velmi snadno si můžeme spočítat, že při použití rozlišení 1280x1024 bude takovýto G-Buffer v paměti grafické karty zabírat přesně 80 MB.



Obrázek 3.3: Schéma G-Bufferu tvořeného pomocí čtyř atributových map

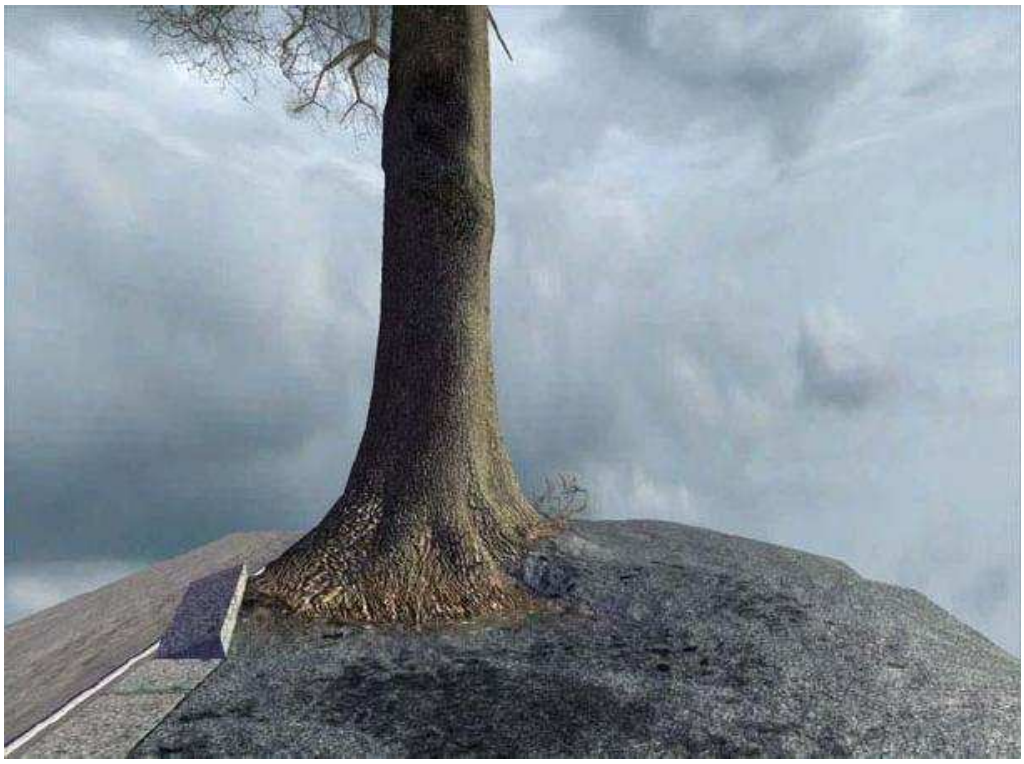
Při návrhu struktury G-Bufferu musíme nejprve vědět, jaký osvětlovací model budeme chtít použít, jelikož každý osvětlovací model má své charakteristické požadavky. Vzhledem k velikosti G-Bufferu je také potřeba vzít v úvahu techniky, které jsou schopny optimalizovat jeho velikost.

Na obrázku 3.4 můžete vidět jednotlivé buffery tvořící G-Buffer, tak jak je Shawn Hargreaves představil na konferenci GameDevelopers Conference v roce 2004.



Obrázek 3.4: Vyobrazení bufferů hloubky, spekulární složky, normál a difúzní barvy

Výsledek výpočtu osvětlení na základě bufferů hloubky, normál, difúzní barvy a spekulární složky můžeme vidět na obrázku 3.5.



Obrázek 3.5: Výsledný obraz vyprodukovaný pomocí techniky deferred shading

3.3.1 Optimalizace velikosti

Zmenšení velikosti G-Bufferu můžeme dosáhnout pouze snížením počtu ukládaných atributů. Počet ukládaných atributů lze redukovat tím, že budeme ukládat pouze takové atributy, na jejichž základě jsme schopni dopočítat původní informace. Nevýhodou je nárůst počtu instrukcí v shader programech při výpočtu osvětlení. V následujícím textu popíši dvě nejznámější možnosti optimalizace.

Všímavý čtenáři si mohli na obrázcích výše povšimnout toho, že Shawn Hargreaves použil buffer hloubky. Obvykle je pozice v G-Bufferu uložena relativně vůči kameře (eye-space). Za předpokladu, že známe vektor pohledu si můžeme dopočítat x-ovou a y-ovou složku pozice pouze na základě složky z (hloubky) pro každý pixel. Správný vektor pohledu pro každý pixel získáme tak, že nejprve vypočítáme čtyři vektory z pozice kamery do všech okrajů obrazu (obdoba vyslání paprsku při raytracingu). Poté takto vypočtené vektory předáme do fragment shaderů, kde poté automaticky získáme díky bilineární interpolaci vektorů odpovídající vektor pohledu $\boldsymbol{v}$ pro každý fragment obrazu. Rovnice pro výpočet souřadnic $\boldsymbol{x}$, $\boldsymbol{y}$ je poté následující (vzorec 3.1):

$$p_x = \frac{v_x}{v_z} p_z \qquad\qquad p_y = \frac{v_y}{v_z} p_z$$

Vzorec 3.1: Výpočet x-ové a y-ové složky vektoru pozice

Další metodou optimalizace je výpočet ***z*** hodnoty normály na základě ***x***, ***y*** hodnoty. Požadavkem ale je, aby velikost vektoru normály byla rovna 1 a zároveň měli všechny normálové vektory v OpenGL kladnou hodnotu složky ***z***. V případě, že jsou tyto požadavky splněny můžeme určit ***z*** hodnotu normály pomocí této rovnice (vzorec 3.2):

$$z = \pm \sqrt{1.0 - (x^2 + y^2)}$$

Vzorec 3.2: Výpočet složky **z** normalizovaného normálového vektoru

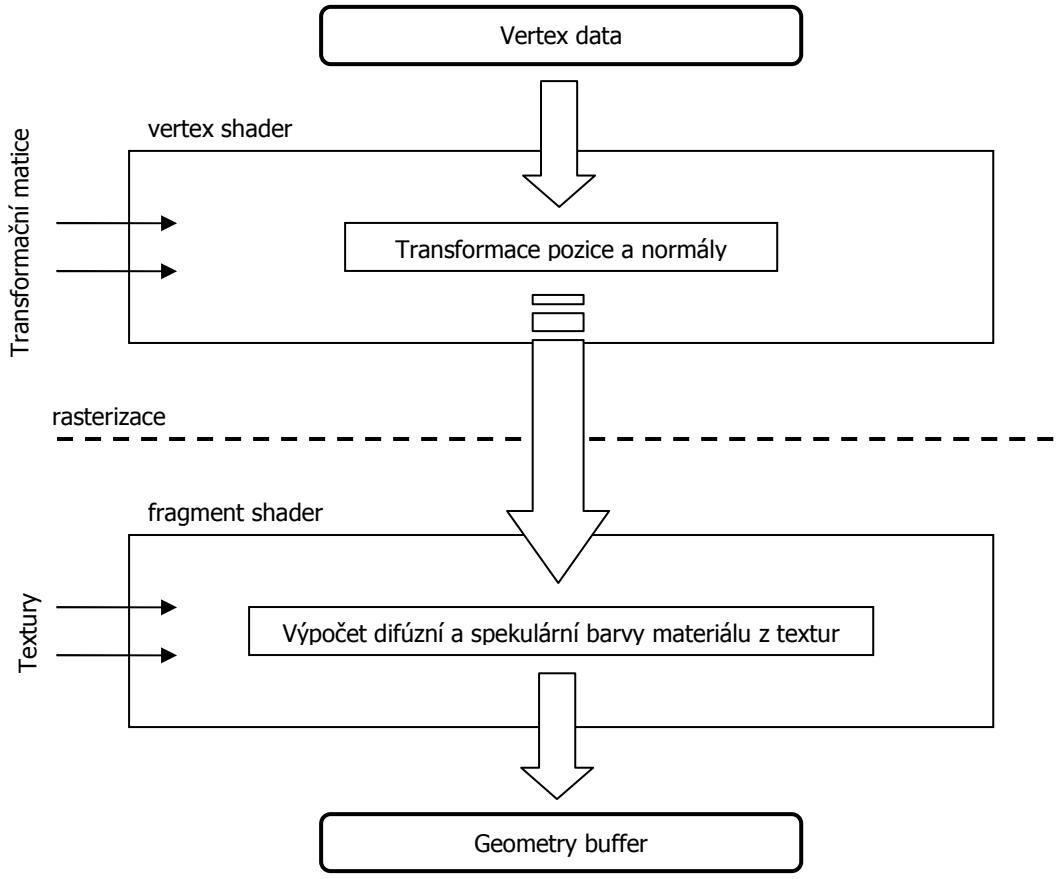
3.4 Vykreslení geometrie

V předchozí podkapitole jsme si popsali funkcionalitu geometry bufferu. Nyní si popíšeme, jakým způsobem dochází k zápisu atributů do tohoto G-Bufferu. Je důležité si uvědomit, že záleží jenom na nás jaké atributy do tohoto G-Bufferu budeme ukládat. V případě phongova osvětlení potřebujeme mít v G-Bufferu pro každý pixel uloženy informace o pozici, nebo hloubce, poté normalizovanou normálu a informace o materiálu. Na základě těchto údajů jsme schopni jednoznačně aplikovat phongovo osvětlení. Pokud bychom se ale rozhodli použít například nefotorealistickou metodu stínování, budeme do G-Bufferu pravděpodobně ukládat jiné pro tuto metodu důležité atributy. Obrázek 3.6 schématicky reprezentuje tento proces uložení atributů do G-Bufferu.

První část odloženého vykreslovacího řetězce, který vykresluje geometrii a zapisuje atributy do G-Bufferu je velmi podobný klasickému vykreslování. Objekt je definován odkazy na použité textury a dále vrcholy v lokálním prostoru, kde každý vrchol obsahuje dodatečné informace o pozici, normále, barvě, texturových koordinátech. Tyto informace jsou předány jako vstup do vykreslovacího řetězce, potažmo vertex shader programu.

Vertex shader aplikuje na každou pozici vrcholu a jeho normálu model-view transformační matici pro převod do pohledového systému kamery. Ostatní informace ze vstupu projdou vertex shaderem nepozměněny, protože budou použity až ve fragment shaderu.

Fragment shader má po rasterizaci na vstupu interpolované hodnoty z vertex shaderu. Vektor pozice a normály se zapíše v nezměněné formě do definovaných atributových map. Na základě texturovacích koordinátů určíme difúzní a spekulární barvu pro daný pixel a taktéž je uložíme do příslušných atributových map.



Obrázek 3.6: Schématické znázornění toku dat při naplňování geometry bufferu

Všechny informace uložené v G-Bufferu jsou definovány relativně vůči kameře. Nyní, když máme G-Buffer naplněný atributy, můžeme přejít k výpočtu osvětlení a stínování.

3.5 Osvětlení a stínování

Jak jsme si již řekli v kapitole 3.2, výpočet osvětlení u deferred shading má lineární složitost. Každý nový objekt, či světlo nám lineárně zvyšuje počet operací, které je nutno v nejhorším možném případě provést. Této složitost je docíleno faktem, že výpočet osvětlení je pouze závislý na velikosti rozlišení a ne na počtu objektů ve scéně. Oddělení fáze vykreslení objektů od samotného osvětlení a stínování nám přináší ještě jeden zajímavý poznatek. Pokud se v následujícím snímku pouze změnilo nasvícení scény, můžeme vypočítat osvětlení z údaju v G-Bufferu naplněného v předcházejícím snímku a tím se vyhnout opětovnému vykreslení geometrie.

Osvětlení lze rozdělit na dvě nezávislé fáze. První fáze je aktualizační, kdy se pro každý světelný zdroj určí jeho pozice na scéně. Druhá fáze je již samotný výpočet osvětlení. Každý světelný zdroj je zpracován shader programem a výsledek osvětlení je additivně smícháván s výsledným

obrazem. Proces výpočtu osvětlení nejprve určí příspěvek osvětlení globálního, ambientního světla a teprve poté se určují příspěvky k osvětlení na základě lokálních světel.

Samotný výpočet osvětlení funguje tak, že vykreslíme obdélník o velikosti obrazovky, čímž dosáhneme aplikace fragment shader programu na všechny pixely. Vstupem pro fragment shader program jsou poté jednotlivé atributové mapy tvořící G-Buffer a informace o světelném zdroji. Každý typ světelného zdroje má svůj vlastní shader program zohledňující jiné vstupní atributy.

3.6 Post-processing

Poslední a volitelnou fází opožděného vykreslovacího řetězce je aplikace post-processing efektů. Mezi nejobvyklejší efekty patří mlha, volumetrická mlha, depth of field, bloom filter a edge detection filter. Edge detection filter je primárně použit jako základ pro anti-aliasing formou rozostření viditelných hran. Hardwarový anti-aliasing není možné u deferred shading použít vzhledem k tomu, že neexistuje hardwarová podpora anti-aliasingu u vykreslovacích cílů. Proto je jedinou cestou jak aplikovat anti-aliasing využití právě fáze post-processingu.

Aplikace post-processing efektů je obdobná jako u výpočtu osvětlení. Nejprve vykreslíme polygon přes celou obrazovku, případně jenom přes určitou zájmovou oblast v závislosti na typu efektu, čímž dojde ke spuštění fragment shaderů daného efektu pro každý pixel obrazovky.

Jednotlivé typy efektů použitých v diplomové práci jsou vysvětleny v podkapitole 4.3 věnované scénám, kde se právě tyto post-processing efekty budou vyskytovat.

3.7 Shrnutí

Deferred shading je technika stínování pouze viditelných pixelů scény. Rozděluje proces vykreslování objektů na obrazovku na dvě části. První část se stará o vykreslení geometrie a naplnění G-Bufferu, přičemž druhá část aplikuje osvětlovací model formou 2D efektů.

Velkou výhodou této techniky je výpočet osvětlení v nejhorším případě s lineární složitostí. Deferred shading je proto velmi vhodný pro zobrazování scény s velkým počtem lokálních zdrojů světla.

Nevýhodou deferred shading je absence hardwarového anti-aliasingu a velká náročnost na velikost paměti grafické karty.

4 Návrh

Cíl práce (obecný popis je uveden v kapitole 1.2) je jasně daný. Vytvořit intuitivní a interaktivní učební pomůcku, popisující techniku deferred shading. Na základě informací z předcházejících kapitol nyní mohu zadefinovat požadavky na aplikaci a rozvést celkový logický návrh aplikace. V další kapitole bude popsána samotná implementace.

4.1 Požadavky

Před vytvářením detailního návrhu implementace je potřeba zadefinovat požadavky na aplikaci. Požadavky lze klasifikovat do následujících kategorií.

- **Požadavky na funkce**
- **Požadavky na výkonnost**
- **Požadavky na vlastnosti**
- **Požadavky na vnější rozhraní**

V mém případě se zaměřím na zadefinování požadavků na funkce, vlastnosti a vnější rozhraní. Samotný detailní návrh bude s těmito požadavky počítat a těmto požadavkům se přizpůsobí celkový design aplikace.

4.1.1 Požadavky na vnější rozhraní a funkce

Vzhledem k povaze aplikace, která má být zároveň i učební pomůckou, musíme definovat dvě rozdílné kategorie požadavků na obecné úrovni. První kategorie souvisí s požadavky na výuku a druhá kategorie definuje požadavky ze strany správného softwarového návrhu z hlediska zásad objektově orientovaného programování.

Co se požadavků na výuku týče, tak aplikace a forma zobrazených informací musí splňovat následující kritéria:

- **Přehlednost**
- **Srozumitelnost**
- **Názornost**
- **Interaktivita**
- **Intuitivní ovládání**

Přehlednosti lze docílit správným rozdělením výuky do jednotlivých částí, kterým budeme říkat lekce. Každá lekce musí popisovat jasnou a srozumitelnou formou danou oblast výuky. Přechod mezi jednotlivými lekcemi by měl být plynulý a logický. Každá lekce by měla využít potenciálu možnosti názorně za použití demonstračních ukázek prezentovat jednotlivé děje, přičemž student by měl mít možnost ovládat různé parametry demonstrace, aby mohl interaktivně měnit celý proces demonstrace a hlouběji pochopit příčiny a důsledky změn jednotlivých parametrů.

Požadavky na správný softwarový návrh z hlediska zásad objektově orientovaného programování lze srhnout do těchto kritérií, tak jak je uvedeno v knize „Návrhové vzory“ [9]:

- **Programovat proti rozhraní**
- **Důsledné skrytí implementace**
- **Zapouzdření a odpoutání části kódu, které by se mohly změnit**
- **Přednost skládání před dědičností**
- **Soudržnost**
- **Minimální vzájemná provázanost**
- **Návrh řízený odpovědnostmi**
- **Vyhýbání se duplicitám v kódu**

Splnění těchto kritérií lze dosáhnout správným navržením tříd a použitím návrhových vzorů.

4.1.2 Požadavky na vlastnosti

Vzhledem k povaze diplomové práce je potřeba klást dodatečné požadavky na zpracování a kvalitu provedení diplomové práce z uživatelského hlediska. Z tohoto důvodu definuji tyto dodatečné požadavky na vlastnosti.

- **Lokalizace**
- **Namluvení**

Lokalizace představuje cestu, jak přeložit veškeré texty do jiných světových jazyků a tím zpřístupnit tuto diplomovou práci širší veřejnosti.

Namluvení představuje cestu jak zprostředkovat předkládané informace větším množstvím informačních kanálů. Tato vlastnost aplikace byla specificky vybrána na základě analýzy metod výuky, přesněji Informativní metodiky (popsané v podkapitole 2.3.1).

4.2 Grafické uživatelské prostředí

Nedílnou součástí každé výukové aplikace musí být velmi dobře navrhnuté grafické uživatelské prostředí [13, 15]. Zobrazení malého počtu ovládacích prvků spolu s jednoduchým ovládáním je tou správnou cestou pro navržení vhodného grafického uživatelského rozhraní.

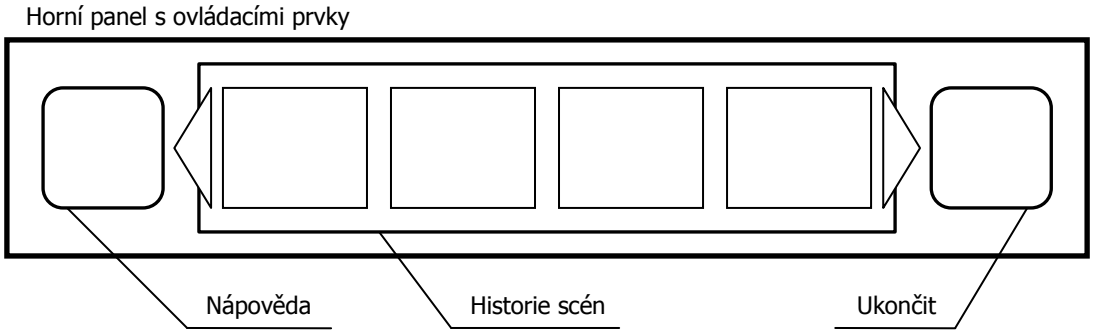
4.2.1 Design aplikace

Nejprve je potřeba se zamyslet nad požadavky na přehlednost a intuitivní ovládání. Požadavek na intuitivní ovládání splním takovým návrhem aplikační logiky, která bude počítat pouze s ovládáním pomocí myši a využitím levého a pravého tlačítka myši. Přehlednosti docílím návrhem správného rozložení prvků v aplikaci.

Na obrázku 4.1 můžeme vidět prvotní návrh rozložení prvků. Celá aplikace je tvořena ze tří částí. Ovládací prvky jsou zobrazeny v horním a spodním panelu okna. Mezi těmito panely je prostor, ve kterém se budou vykreslovat jednotlivé scény popisující principy techniky deferred shading (podrobný popis nalezneme v podkapitole 4.3).

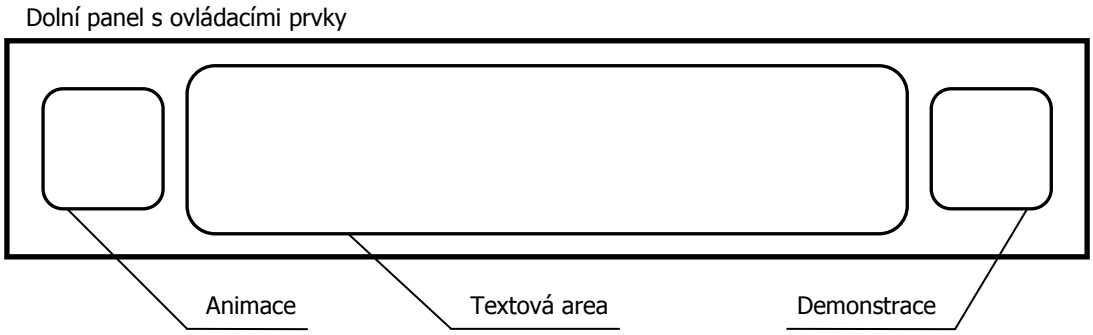
Obrázek 4.1: Návrh rozložení grafických prvků v aplikaci

Horní i spodní panel s ovládacími prvky bude stále zobrazovat stejné prvky bez ohledu na právě zobrazovanou scénu. Na obrázku 4.2 a 4.3 je předběžný návrh rozložení použitých ovládacích prvků v těchto panelech. Podrobný návrh ovládacích prvků použitých na těchto obrázcích a jejich funkcionalita je uvedena v podkapitole 4.2.2.



Obrázek 4.2 – Návrh ovládacích prvků v horním panelu aplikace

Horní panel obsahuje tři základní ovládací prvky. Dvě tlačítka reprezentující možnosti zobrazení nápovědy a ukončení aplikace. Ve střední části panelu je ovládací prvek zobrazující historii scén, kterými student procházel.



Obrázek 4.3 – Návrh ovládacích prvků ve spodním panelu aplikace

Spodní panel taktéž obsahuje tři ovládací prvky. Vlevo máme tlačítko umožňující spustit animaci, vpravo tlačítko spouštějící demonstraci. Uprostřed panelu máme oblast, kde se bude zobrazovat text. Stav těchto ovládacích prvků je velmi silně závislý na právě zobrazované scéně.

Největší oblast aplikace je vyhrazena pro zobrazování scény. **Scéna** obsahuje různé objekty popisující, případně reprezentující, daný subsystém techniky deferred shading. Pro účely výukového programu navrhuji tři různé druhy scén.

- Statická
- Animační
- Demonstrační

Statická scéna pouze zobrazuje objekty, o kterých si uživatel může nechat vypisovat dodatečné informace, případně provádět další akce popsané v následující podkapitole. V případě, že se dá na statické scéně spustit animace, resp. demonstrace je vysvíceno tlačítko *Animace*, resp. *Demonstrace*.

Scéna **animační** naproti tomu je plně automatizovaná a uživatel nemá možnost ovlivnit animaci jinak, než jejím předčasným ukončením. Tento typ scény je aktivován stisknutím tlačítka *Animace* v levé části spodního ovládacího panelu. Animace je tvořena zobrazovaním informačního textu spolu se synchronizovaným mluveným slovem a zobrazováním objektů na scéně.

Při stisku tlačítka *Demonstrace* v pravé části spodního ovládacího panelu se aktivuje **demonstrační** scéna. Demonstrační scéna používá přímo techniku deferred shading a ukazuje různé možnosti a vlastnosti této techniky v závislosti na právě probírané látce.

4.2.2 Ovládací a zobrazovací prvky

Grafické uživatelské rozhraní nemůže fungovat bez reakcí uživatele. Aby uživatel mohl určitým způsobem ovládat aplikaci je třeba vytvořit ovládací prvky. Na druhou stranu potřebujeme taktéž i vizualizovat změny v aplikaci v závislosti na reakcích uživatele. Z tohoto důvodu musíme navrhnout i takzvané zobrazovací prvky.

Přehled jednotlivých typů prvků navržených pro výukovou aplikaci je následující:

- Tlačítko
- Obrázek
- Plovoucí nápověda
- Oblast pro zobrazení informací
- Historie navštívených scén
- Deferred Shading

Tlačítko patří do kategorie ovládacích prvků. Je použit pro jakýkoliv objekt, který nějakým způsobem má reagovat na uživatele. Uživatel má možnost nad zobrazeným tlačítkem stisknout levé tlačítko resp. pravé tlačítko myši, přičemž dojde k vyvolání určité akce. Každému tlačítku přiřadíme příslušnou akci, která se má provést v reakci na kliknutí myši. Po zhodnocení všech vlastností výukového programu navrhuji tuto konečnou množinu akcí:

- Ukončení aplikace
- Zobrazení nápovědy
- Spuštění animace
- Spuštění demonstrace
- Zobrazení informačního textu

- Přechod na jinou scénu

Akce *Ukončení aplikace* a *Zobrazení nápovědy* není třeba rozebírat. Dále zde máme akce pro *Spuštění animace* a *Spuštění demonstrace*. Pojem animace a demonstrace jsme si vysvětlili v předcházející podkapitole. *Zobrazení informačního textu* se použije pro výpis podrobného text o funkcionalitě stisknutého tlačítka (znovu opakuji, že tlačítko je pouze typ objektů reagujích určitým způsobem na uživatele), přičemž dojde i k přehrání namluveného textu. Poslední akce *Přechod na jinou scénu* odstraní současnou scénu a zobrazí scénu novou.

Obrázek patří do skupiny zobrazovacích prvků tvořících objekty, které nereagují na reakce uživatele.

Dalším zástupcem skupiny pouze zobrazovacích prvků je **plovoucí nápověda**. Plovoucí nápověda představuje cestu jak stručně popsat objekty na scéně i v ovládacích panelech. Pokud se uživatel zastaví s myší nad nějakým objektem po dobu alespoň jedné vteřiny, zobrazí se plouvoucí nápověda k tomuto objektu. Jediné dva typy prvků, které nemají definovanou plovoucí nápovědu je plovoucí nápověda sama a prvek demonstrace Deferred Shading.

Posledním příslušníkem zobrazovacích prvků je **oblast pro zobrazení informací**. Pokud uživatel nad některým objektem stiskne tlačítko myši, pro které je zadefinována akce *Zobrazení informačního textu*, tak právě tato oblast je určena pro zobrazování informací. Informační text je plynule posouván v této oblasti tak jak plyne mluvené slovo.

Historie navštívených scén se řadí do kategorie ovládacích prvků. Většina scén obsahuje objekty (tlačítka), na které je namapována akce *Přechodu na jinou scénu*. Tímto způsobem se uživatel přesunuje z jedné oblasti výuky do dalších. Prvek historie zobrazuje právě přechody mezi těmito scénami a umožňuje uživateli okamžitý přechod do jakékoliv již navštívené scény. V okamžiku přechodu na jinou scénu se do historie přidá miniatura předcházející scény. Historie zobrazuje pouze čtyři posledně navštívené scény, přičemž poskytuje tlačítka *Zpět* a *Dopředu*. Tlačítkem *zpět* a *dopředu* můžeme listovat skrze všechny zobrazené scény od spuštění aplikace.

Posledním typem je ovládací prvek **deferred shading**. Deferred shading prvek zároveň představuje demonstrační scénu. Řadí se mezi ovládací prvky, jelikož bude demonstrační scéna obvykle podporovat reakce na stisk a pohyb myši uživatele v závislosti na zobrazené demonstraci. Přehled všech deferred shading demonstrací je uveden v podkapitole 4.3 jako součást přehledu o každé scéně.

4.3 Scény

Celá výuková aplikace je rozdělena do takzvaných scén. Scéna obsahuje jednotlivé entity vysvětlující aktuální látku, případně demonstraci techniky deferred shading. Samotný návrh všech scén a jejich propojení mezi sebou bylo během implementace několikrát předěláno z důvodu neefektivní, či

nepřehledné struktury. V této podkapitole si tedy představíme všechny finálně navržené scény a popíšeme si i celkovou strukturu učebního plánu.

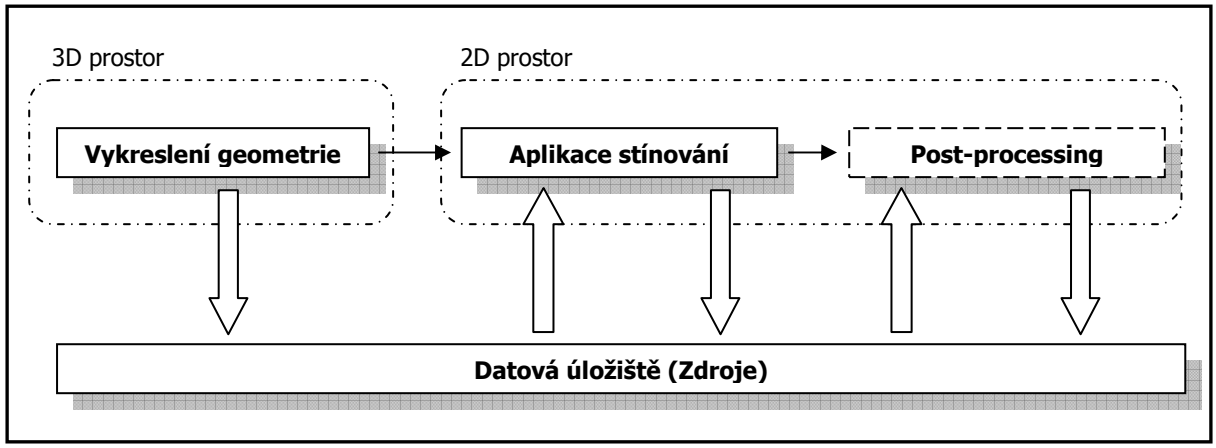
Každá scéna je koncipiována tak, aby obsahovala co nejméně entit. V případě, že není možné aby scéna obsahovala méně než sedm entit (počet entit je vybrán na základě všeobecně uznávaného poznatku, že krátkodobá paměť je schopna si pamatovat v jednom okamžiku čtyři až devět elementů) se musíme snažit důležité entity nějakým způsobem zvýraznit oproti ostatním entitám. Tímto způsobem dokážeme předkládat všechny informace přehledně. Můžeme si představit scénu, kde máme zobrazeny čtyři důležité entity, které jsou provázany mezi sebou (vztah může být reprezentován čárovou entitou vedoucí z jedné entity do druhé). Vztahy v tomto případě jsou pouze dodatečně informativní a pro nás nepříliš významné. Proto těmto vztahům přiřadíme nevýrazné barvy, zatímco ony čtyři důležité entity se budeme snažit pomocí výrazných barev co nejvíce zvýraznit.

Jednotlivé scény jsou propojeny mezi sebou, přičemž student má ihned po spuštění aplikace možnost se neomezeně pohybovat ve všech scénách bez jakýchkoliv restrikcí. Návrh sekvence scén je vytvořen tak, abychom přecházeli od obecnějších informací k těm konkrétnějším.

V následujících podkapitolách budou zobrazena schémata scén, přičemž entita popsaná tučným textem představuje zároveň i navigační entitu. Celou hierarchii přechodů mezi scénami představím v poslední podkapitole 4.3.15.

4.3.1 Vykreslovací řetězec

Ihned po spuštění aplikace se zobrazí úvodní scéna. Scéna by měla zobrazovat vykreslovací řetězec techniky deferred shading tak jak je uveden na obrázku 4.4.



Obrázek 4.4: Schéma úvodní scény

Jak si můžeme všimnout, tento obrázek je upravenou verzí schématu zpožděného vykreslovacího řetězce popsaného v třetí kapitole a zobrazeného na obrázku 3.2, proto jej nebudu ani vysvětlovat. Pozornost si ale zaslouží entita Post-processing a její ohraničení. Můžete si všimnout, že

hranice entity je vyobrazena čárkovaně. Tímto způsobem se snažím nenásilně upozornit na to, že post-processing patří do množiny volitelně aplikovatelných entit. Tedy pokud nechceme použít žádný post-processing efekt, tak nemusíme a nijak to neovlivní samotnou techniku deferred shading.

Pro tuto scénu je definována jedna animace. Animace bude znázorňovat postup techniky deferred shading. Nejprve se vysvítí entita Vykreslení geometrie, poté přechod do entity Datová úložiště, přičemž vysvítíme i entitu zdroje. Stejným způsobem se dostaneme ze zdroje do entity Aplikace stínování, přičemž po vysvětlení této fáze se opět vrátíme do datového úložiště, abychom mohli obdobný proces provést i nad entitou post-processing. V celém průběhu animace se v textovém poli zobrazuje text popisující co se právě děje, přičemž tento text bude předkládán i zvukovou cestou díky namluvení.

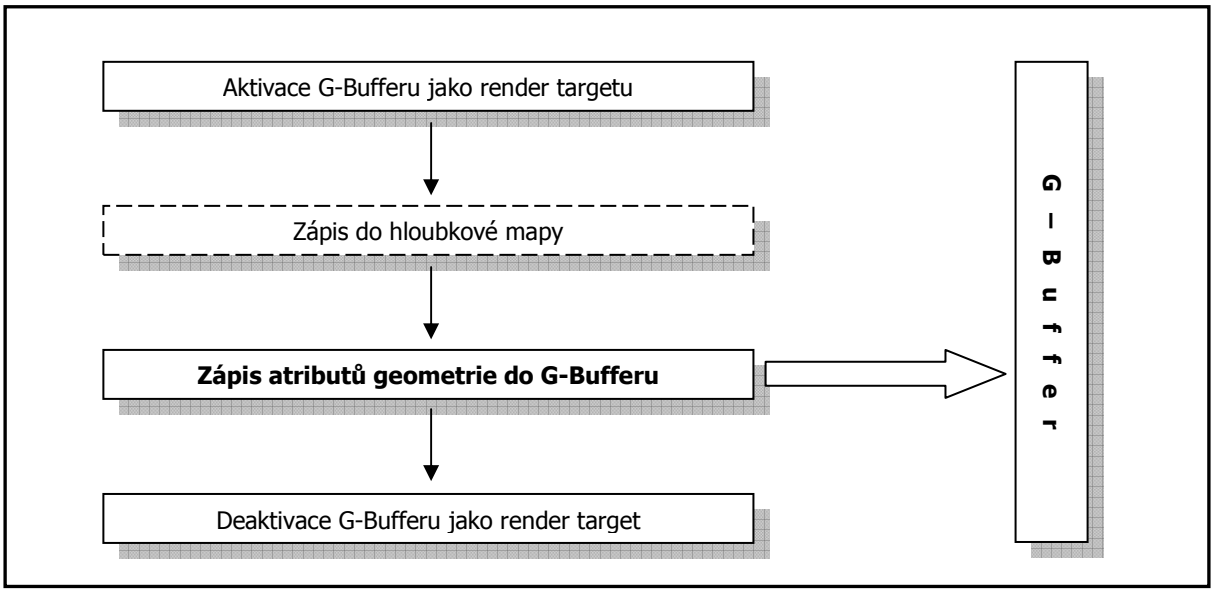
Jedna demonstrace bude připravena i pro úvodní scénu. Demonstrace by měla znázorňovat složitější objekt spolu s velkým počtem lokálních světel, které by měli okolo tohoto objektu rotovat. Zároveň by se měli využít post-processing efekty bloom a antialias filter. Počet světel by mělo jít jednoduchým způsobem upravovat (přidávat, odebírat), stejně jako by mělo být možné zapnout, resp. vypnout fázi zpracování geometrie, stejně jako i jednotlivé post-processing efekty.

4.3.2 Vykreslení geometrie

V této scéně bude osvětlen základní mechanismus vykreslování dat do G-Bufferu. Nejprve je třeba nastavit jednotlivé atributové mapy G-Bufferu jako vykreslovací cíle pro budoucí vykreslování mimo obrazovku. Dále máme nepovinnou část vykreslení dat do hloubkové mapy.

Pro zápis do hloubkové mapy potřebujeme posílat na grafickou kartu pouze vrcholy polygonů bez normál, texturových koordinátů a tak dále. Jakmile začneme vykreslovat v další fázi, přičemž používáme složitější fragment shadery, ušetříme vykonání složitých fragment operací přeskočením fragmentů, které jsou vzdálenější než fragment uložené v hloubkové mapě vytvořené v předchozím kroku. Přístup, kdy nejprve zapíšeme informace do hloubkové mapy, abychom urychlili následující vykreslovací fázi se nazývá early Z checking [16].

Po volitelném využití zápisu do hloubkové mapy je třeba zapsat všechny atributy do G-Buffer. Nakonec zrušíme nastavení G-Bufferu jako vykreslovacího cíle. Celý tento proces je zobrazen na následujícím schématu.



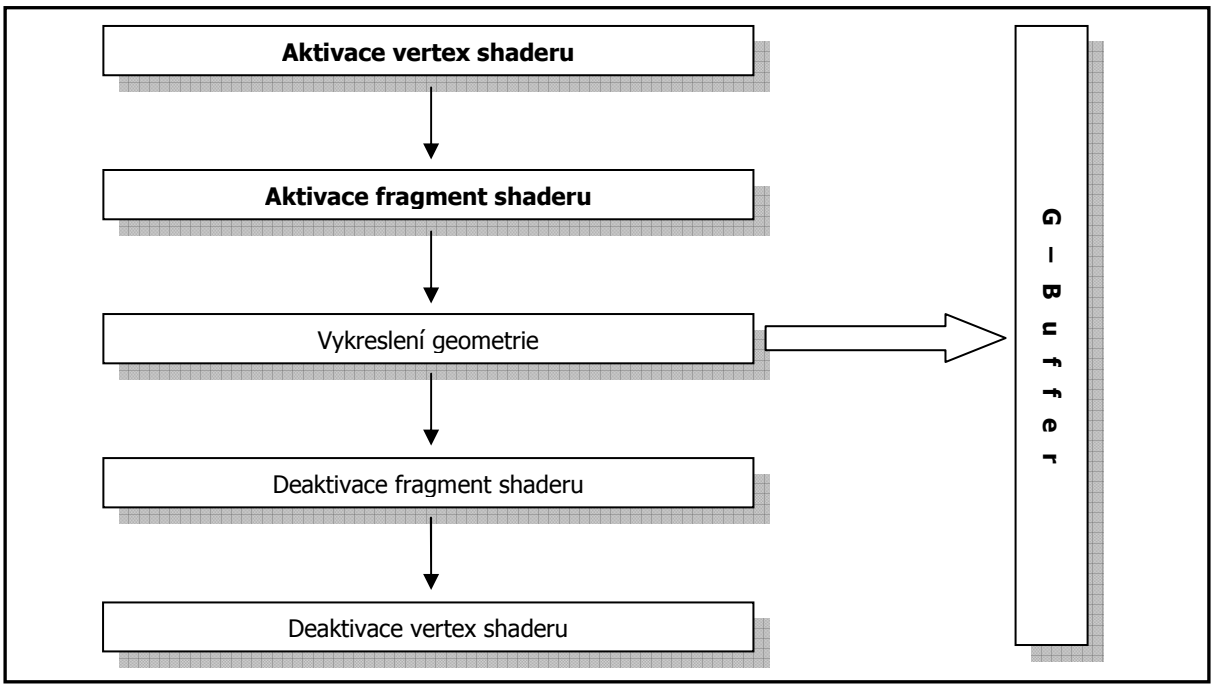
Obrázek 4.5: Schéma scény vykreslení geometrie

Součástí této scény bude i možnost spustit animaci. Animace postupně krok za krokem vysvětlí funkčnost a tok dat na této scéně.

Demonstrace nebude součástí této scény.

4.3.3 Zápis atributů geometrie do G-Bufferu

Zápis atributů geometrie do G-Bufferu je jednou z nejdůležitějších fází techniky deferred shading. Scéna bude zobrazovat nastavení vertex a fragment shaderů pro vykreslení do několika atributových map (tvořící G-Buffer) a samotný proces vykreslení geometrie, tak jak je uvedeno na obrázku 4.6.



Obrázek 4.6: Schéma scény vykreslení geometrie do G-Bufferu

Animace pro tuto scénu nebude dostupná.

Demonstrace bude pro tuto scénu připravena. Obsahem demonstrace bude zobrazení jednoduché scény (převzané z demonstrace použité v úvodní scény popisující vykreslovací řetězec) a výsledných atributových map G-Bufferu. V demonstraci nebude použito světlo, ani žádný post-processing efekt.

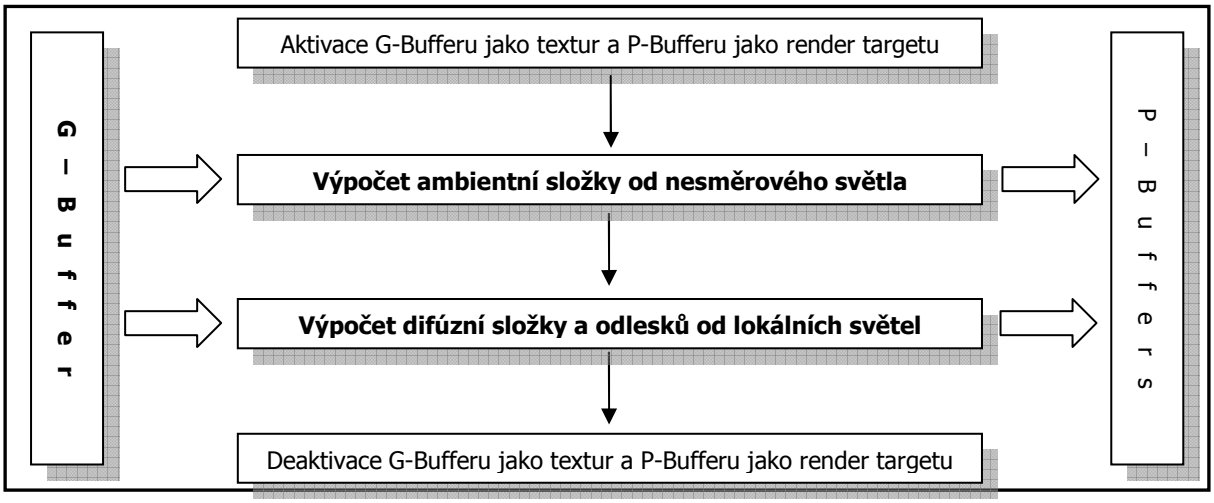
4.3.4 Vertex a fragment shader pro vykreslení geometrie

V předcházející scéně jsme aktivovali vertex a fragment shadery pro vykreslení geometrie. Nynější scény budou zobrazovat okomentovaný zdrojový kód těchto shader programů.

Zároveň bude možné i spustit animaci, ve které se jednotlivé řádky zdrojové kódu budou označovat s vysvětlením co se právě děje se vstupními daty.

4.3.5 Aplikace stínování

Druhá nejdůležitější fáze techniky deferred shading je aplikace stínování na základě informací v G-Bufferu. Scéna popisuje základní kroky této fáze. Nejprve musíme nastavit jednotlivé atributové mapy G-Bufferu jako vstupní textury pro shader programy a P-Buffer jako vykreslovací cíl. Poté se vypočítá příspěvek nesměrového světla. Po výpočtu příspěvku nesměrového světla je potřeba pro všechny světla na scéně vypočítat i jejich příspěvky. Schématický návrh této scény je popsán na obrázku 4.7.



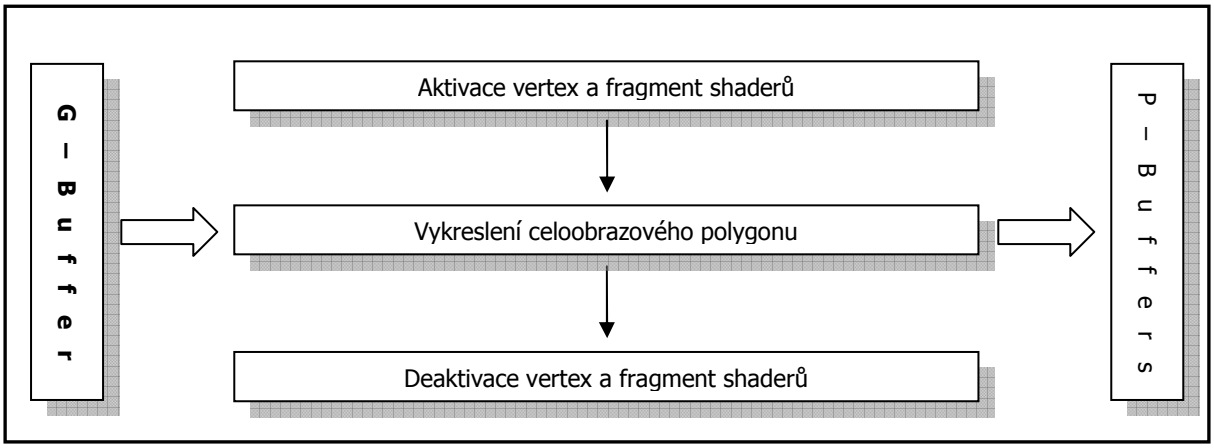
Obrázek 4.7: Schéma scény popisující aplikaci stínování

Scéna popisující aplikaci stínování patří mezi ty nejsložitější, tudíž je potřeba mít animaci osvětlující činnost. Animace se nejprve zaměří na vysvětlení nutnosti použití G-Bufferu jako textury a mapování jednoho z P-Bufferu jako vykreslovacího cíle. Poté je obecně vysvětleno jakým způsobem dosáhneme výpočtu ambientní složky materiálů a jejich zápis do P-Bufferu. Obdobně je vysvětlen i výpočet difúzní složky a odlesků od lokálních světel.

Demonstrace bude zobrazovat netriviální model osvětlený různými typy světelných zdrojů. Bude možnost přidávat, resp. odebírat jednotlivé světelné zdroje. Dále bude možnost nechat zobrazit jednoduchý model světelného zdroje, aby bylo dobře vidět jakým způsobem se určuje, zda-li daný světelný zdroj bude využit při výpočtu stínování.

4.3.6 Výpočet ambientní složky od nesměrového světla

Nesměrové světlo je unikátní v tom slova smyslu, že není možné určit zdroj světelného zdroje a je aplikován vždy na celou vykreslovanou geometrii, přičemž tento typ světla má být použit pro simulaci několikrát odražených světelných paprsků z lokálních zdrojů světla. Technika deferred shading s tímto typem světel samozřejmě počítá a schéma výpočtu můžete nalézt na obrázku 4.8. Princip výpočtu ambientní složky nesměrového světla je vysvětlen právě v této scéně. Nejprve je potřeba nastavit příslušné vertex a fragment shader programy, předat fragment shaderu konstantu definující barvu světla a poté vykreslit polygon překrývající celou obrazovku. Tímto krokem docílíme použití shader programů na každý pixel výsledného obrazu a provedení jednoduchého výpočtu ambientní složky na základě informací uložených v G-Bufferu.



Obrázek 4.8: Schéma scény výpočtu ambientní složky od všesměrového světla

Animace ani demonstrace nebude součástí této scény.

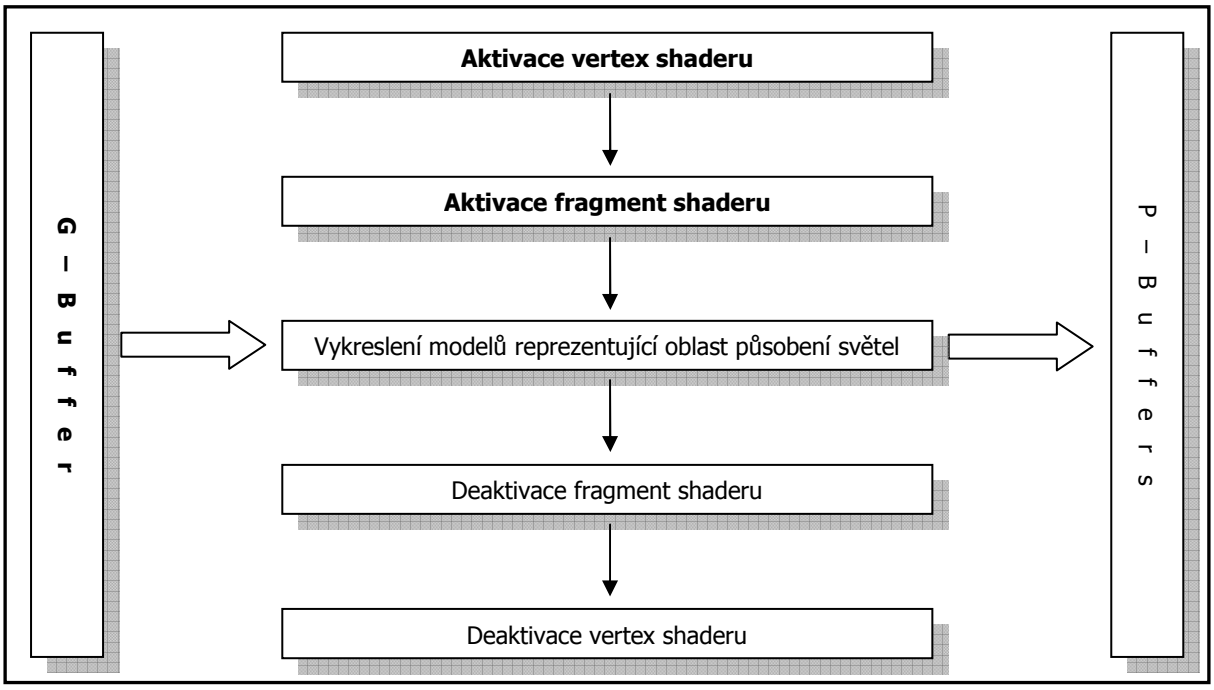
4.3.7 Výpočet difúzní složky a odlesků od lokálních světel

Všechna lokální světla, která ovlivňují pouze určitou oblast geometrie, jsou zpracovávana v této scéně. Nejprve musíme nastavit vertex a fragment shader, který bude vypočítávat výslednou barvu difúzní složky materiálů a odlesků.

Po nastavení daných vertex a fragment shader programů vykreslíme pro každé světlo model reprezentující jeho oblast působení. V případě všesměrového typu světel vykreslíme jednoduchou kouli se středem v pozici všesměrového světla a s poloměrem daným dosahem tohoto typu světla. Pro typ světla typu reflektor vykreslujeme jednoduchý kužel. Jedinou vyjímkou je směrový typ světla,

který není lokálním typem světel a u kterého je potřeba vykreslit polygon překrývající celou obrazovku, abychom mohli aplikovat tento druh světla na celou geometrii.

Na obrázku 4.9 je znázorněno schéma této situace.



Obrázek 4.9: Schéma scény vypočítavající difúzní složku a odlesky

K této scéně nebude možnost spustit animaci osvětlující celý proces výpočtu výsledné barvy každého pixelu.

Demonstrace bude v tomto případě obsažena taktéž, přičemž si student bude moci vybrat ze tří demonstrací. První demonstrace bude věnována pouze všesměrovým světlům a bude zobrazovat jednoduchou scénu na které bude zobrazeno jedno všesměrové světlo i s jeho zjednodušeným modelem koule, aby šlo názorně vidět jak se výpočet osvětlení odrazí pouze na pixelech, které jsou v dosahu této koule. Druhá demonstrace bude obdobným způsobem jako první demostrace věnována reflektorovým typům světel a třetí demonstrace bude určena výhradně pro směrové světla.

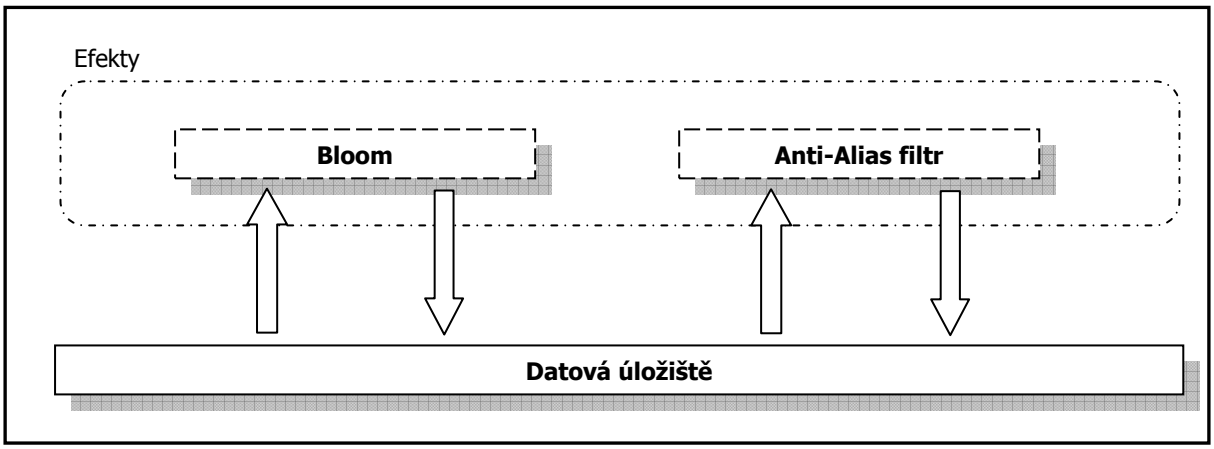
4.3.8 Vertex a fragment shadery pro aplikaci stínování

Na těchto scénách bude zobrazen okomentovaný zdrojový kód vertex a fragment shader programů. Součástí scén je i animace postupně vysvětlující jednotlivé řádky zdrojového kódu.

4.3.9 Post-processing

Volitelnou částí techniky deferred shading je aplikace post-processingu na výsledný obraz. Jednotlivé efekty mohou vycházet jak z údajů uložených v G-Bufferu, tak pouze z údajů o výsledné barvě pro

každý pixel výsledného obrazu. Jak můžeme vidět na obrázku 4.10, tato scéna obsahuje dva efekty – bloom a anti-alias filter.



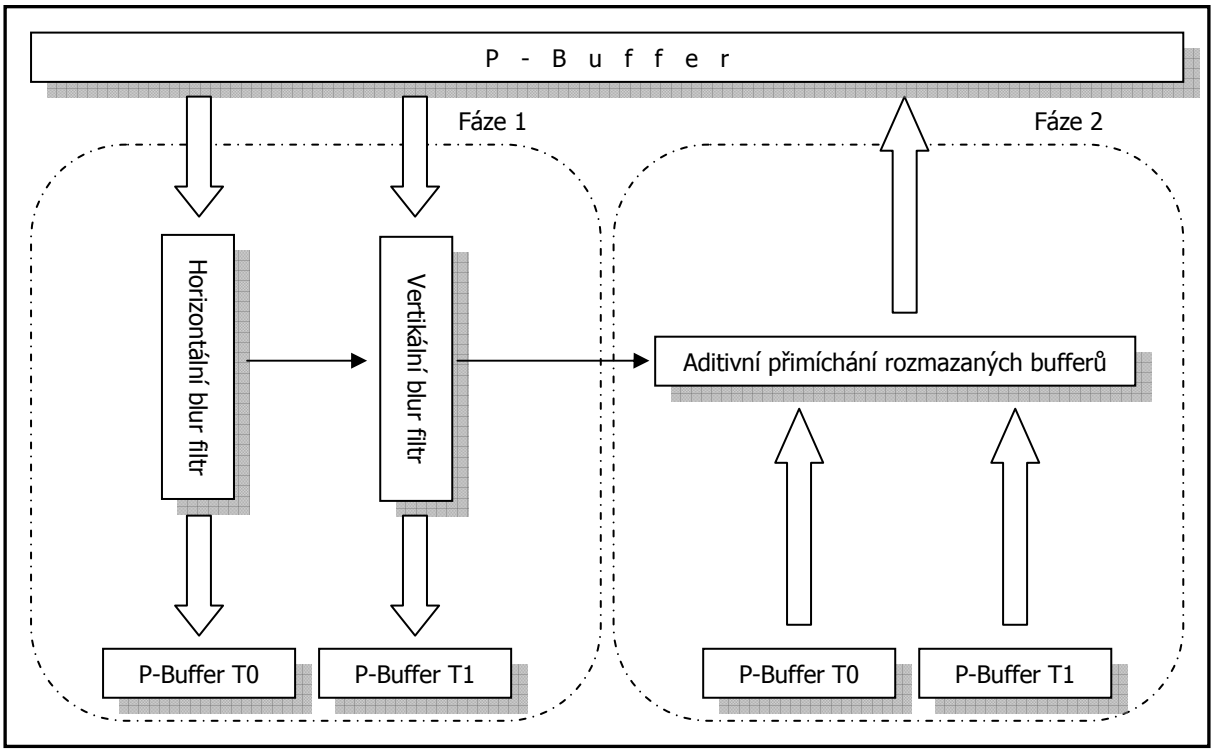
Obrázek 4.10: Schéma post-processing scény

Animace ani demonstrace nebude pro tuto scénu k dispozici.

4.3.10 Bloom

Bloom je způsoben rozptýlením světelných paprsků v čočce oka. To má za následek záři okolo světlých oblastí a utlumení kontrastu v temných oblastech. Pro vytvoření efektu Bloomu jsou potřeba dva průchody.

V prvním průchodu aplikujeme horizontálně na vstupní obraz blur filter s velikostí jádra 7x7. Ve druhém průchodě aplikujeme stejný blur filter vertikálně. Výsledky těchto průchodů jsou ukládány do dočasných textur s menším rozlišením, čímž dosáhneme uspokojivé rychlosti vykreslování. Tyto rozmazané textury jsou poté aditivně přimíchány k výslednému obrazu scény, tak jak je ukázáno na obrázku 4.11.



Obrázek 4.11: Schéma aplikace bloom efektu

Scéna obsahuje animaci popisující aktivity, kterými je třeba projít než dosáhneme aplikace efektu bloom. Nejprve bude vysvětlena první fáze tohoto efektu, kdy je třeba vytvořit dočasné atributové mapy obsahující rozmazanou verzi vstupního obrazu. Poté přejdeme k druhé fázi, kde dojde k aditivnímu přimíchání takto vytvořených bufferů se vstupním obrazem.

Demonstrace bude ukazovat jednoduchou scénu na kterou bude aplikován efekt bloom, přičemž bude viditelný i obsah atributových map P-Buffer T0 a P-Buffer T1.

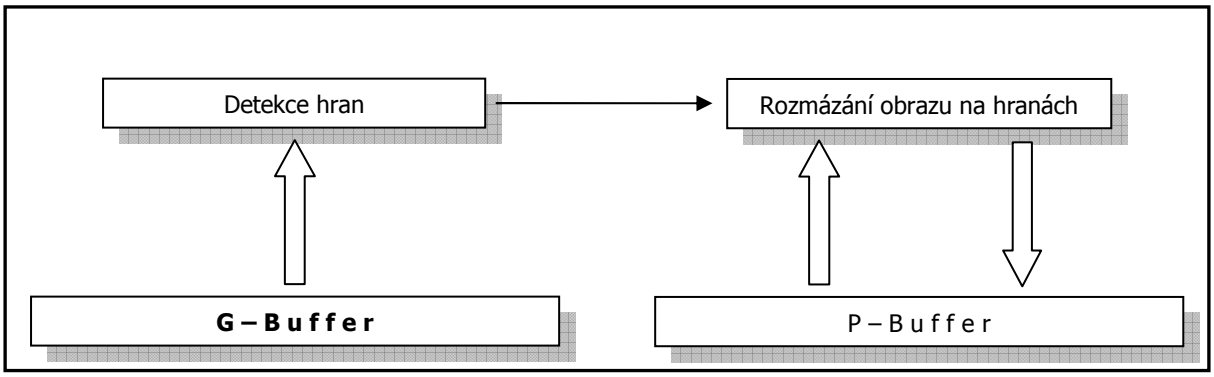
4.3.11 Anti-Aliasing filtr

V podkapitole 3.6 jsme se dozvěděli, že není možné použít hardwarový anti-aliasing. Máme ale možnost použít fázi post-processing a aplikovat svůj vlastní anti-aliasing filtr na výstupní obraz. Princip je naznačen na obrázku 4.12. Aplikace anti-aliasing filtru se provádí ve dvou krocích a našim cílem je vytvoření rozmazaného obrazu pouze u pixelů reprezentujích hrany.

Prvním krokem je tedy samotná detekce hran. Na základě porovnávání sousedních pixelů v normálové a hloubkové mapě (tvořících součást G-Bufferu) můžeme určit pro každý pixel faktor rozmazání.

Jakmile známe faktor rozmazání, tak barvu výsledného pixelu tvoří všechny sousední pixely vynásobené s daným faktorem rozmazání a zároveň i váhou. Váha určuje jakou sílu přikládáme faktoru rozmázání.

Tímto postupem aplikujeme rozmázání pouze na pixely, které reprezentují hranu.



Obrázek 4.12: Návrh scény popisující princip anti-aliasing filtru

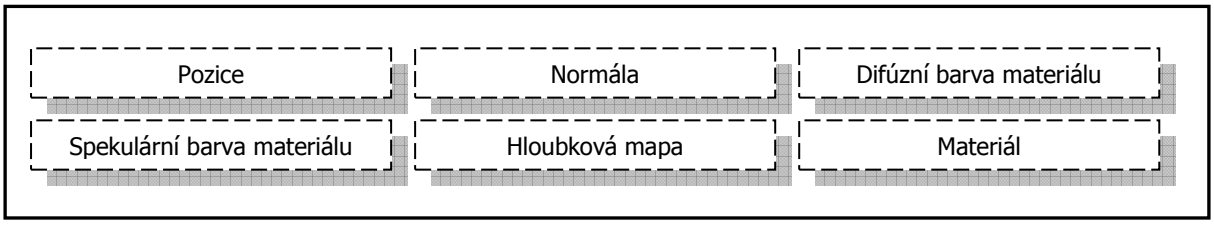
Animace nebude součástí této scény. Demonstrační ukázka aplikace anti alias filtru bude dostupná, přičemž by mělo být možné si nechat zobrazit i výsledek detekce hran.

4.3.12 Datová úložiště (Zdroje)

Datová úložiště reprezentují veškeré datové prostředky, které musíme na grafické kartě vyhradit pro techniku deferred shading. Tyto zdroje jsou pojmenovány jako framebuffer, geometry buffer (G-Buffer) a picture buffer (P-Buffer). Vzhledem k jednoduchosti této scény nebudu uvádět schéma scény.

4.3.13 G-Buffer

G-Buffer neboli Geometry Buffer je esenciálním prvkem celého vykreslovacího řetězce. Veškeré atributy o geometrii jsou uloženy právě v tomto bufferu. Na obrázku 4.14 je vyobrazeno schéma scény popisující G-Buffer. Ten je tvořen několika atributovými poli, ukládající nejrůznější informace o geometrii. V závislosti na použitém osvětlovacím modelu ukládáme do těchto atributových polí požadované informace. V našem případě je geometry buffer tvořen několika poli. První atributové pole ukládá informace o pozici, druhé pole ukládá informace o normále v daném pixelu. Třetí buffer ukládá difúzní barvu materiálu a tak podobně. Na obrázku 4.13 je scéna navržena tak, aby si student uvědomil, že je pouze na nás jaké informace budeme chtít v G-Bufferu ukládat.

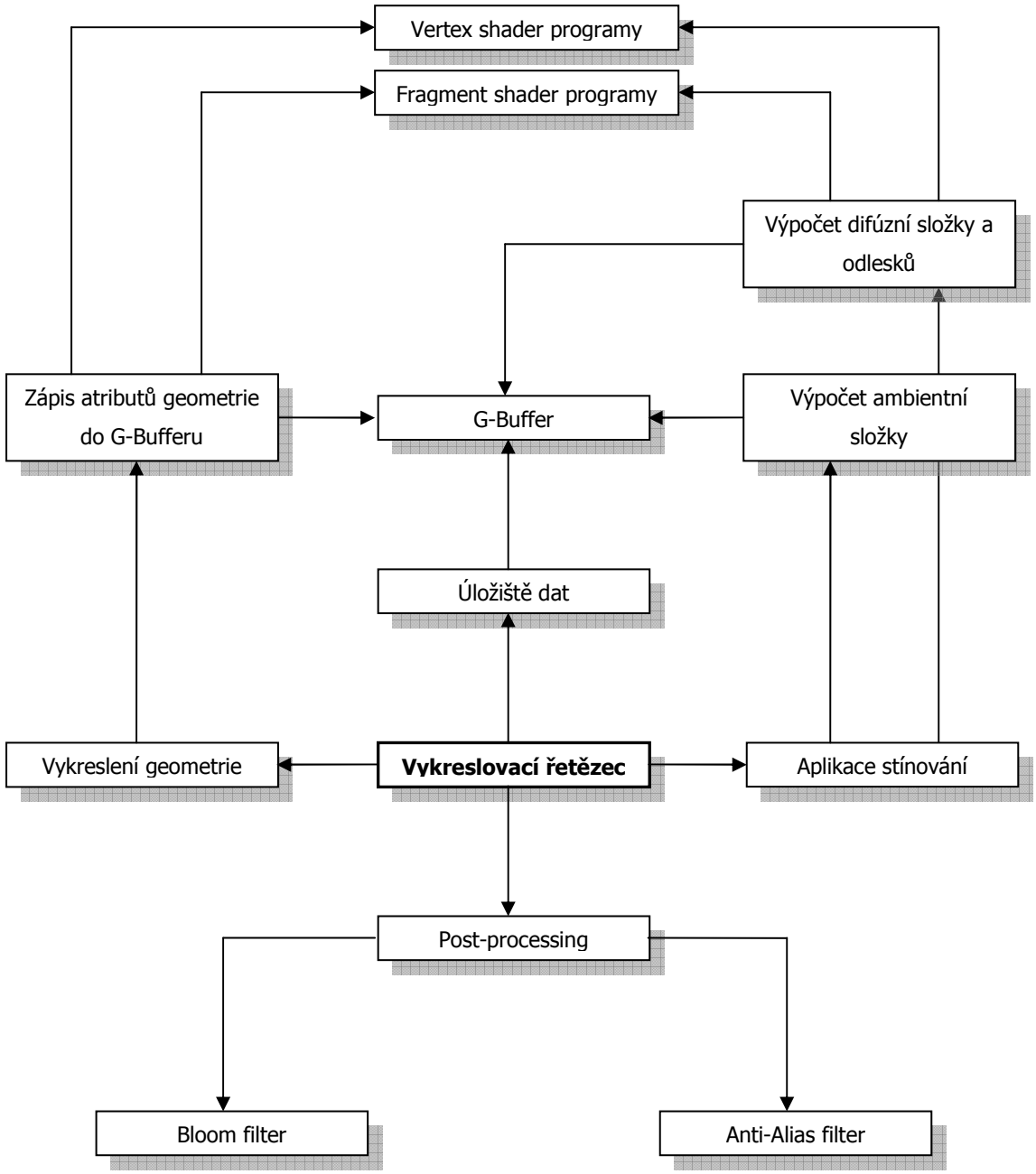


Obrázek 4.13: Návrh scény datová úložiště

Animace ani demonstrace není součástí scény G-Buffer.

4.3.14 Hierarchie přechodů mezi scénami

V tomto okamžiku již známe všechny navržené scény. Poslední věc, kterou musím k tématu scén zmínit je hierarchie přechodů mezi scénami tak jak je vyobrazen na obrázku 4.16.



Obrázek 4.16: Návrh hierarchie přechodů mezi scénami

4.4 Dotazník

Součástí diplomové práce by mělo být taktéž vytvoření anonymního dotazníku. Tento dotazník poslouží pro zhodnocení jednotlivých aspektů výukové aplikace ze strany studentů, a proto je nutné věnovat struktuře dotazníku velkou pozornost. Při určování struktury dotazníku a správné metodiky otázek jsem vycházel z informací v odborných knihách [11, 12].

4.4.1 Typy otázek

Správným výběrem otázek se snažíme dosáhnout situace, kdy výsledné odpovědi jsou relevantní a co nejvíce přesné. Otázky v dotazníku mohou být tří typů.

- Uzavřené
- Otevřené
- Škálové

Uzavřené otázky nabízejí pouze striktní a velmi omezenou množinu absolutních odpovědí typu Ano, Ne, Nevím. Výhodou těchto otázek je větší jednotnost měření a jednodušší korelace statistických výsledků. Nevýhodou je nemožnost dostat se pod povrch odpovědi a v neposlední řadě i určitý tlak na tázaného vybrat nějakou alternativu, jen aby zakryl svoji vlastní nevědomost.

Naproti tomu otevřené otázky kladou velmi málo omezení na odpovědi. Odpovědi jsou obvykle psané a co nejvíce popisující postoj tázaného k dotazu. Omezením tohoto typu otázek je velká časová náročnost na tazatele a poté velmi složité statistické vyhodnocení odpovědí.

Škálové otázky se typicky vyznačující možností odpovědi v určité škále. Správně definovaná posuzovací škála zajišťuje určitou objektivnost a zároveň zachycuje jev v kvantitativní rovině. Posuzovací škála trpí u některých posuzovatelů při hodnocení chybou z přísnosti, znehodnocovací chybou (deprecionalizace), případně chybou shovívavosti (favorizace). Proto je nutné pro každý dotazník vyhodnotit zda-li netrpí jednou z uvedených chyb a případně tuto chybu kompenzovat.

4.4.2 Požadavky na metodu dotazníku

Metoda dotazníku musí splňovat nejrůznější kritéria, aby se o ní dalo říci že získáváme fakta. Těmito požadavkami jsou.

- Objektivnost
- Spolehlivost
- Standardnost
- Validita

Objektivnost metody je určena stupněm nezávislosti osoby vyplňující dotazník na dotazovanou oblast. Omezuje nebezpečí, že by tazatel zkreslil fakta, aby získal žádoucí výsledky a tím přispívá k jednoznačnosti výsledků. Jinými slovy objektivita metody je definována takovou formulací otázek, které nemanipulují s osobou tak, aby odpověděl ve prospěch tazatele.

Spolehlivost lze prokázat až časem a opakovaným použitím metody dotazníku. Vysokou spolehlivost mají metody, které jsou stálé v čase. Na druhou stranu metody, které vykazují velké rozdíly ve výsledcích oproti předchozímu měření mají spolehlivost velmi nízkou. V podstatě existuje několik způsobů jak určit spolehlivost metody. Prvním způsobem je opakované použití metody u stejných osob za předpokladu, že se sledovaná vlastnost nezměnila. Vnitřní konzistenci metody a její spolehlivost lze taktéž určit rozdělením dotazníků na dvě části a porovnáním výsledků každé části zvlášť. Míra homogenity těchto výsledku nám poté definuje spolehlivost metody.

Každá osoba vyplňující dotazník musí následovat stejný proces, aby byl zabezpečen požadavek na standardnost. Všichni zúčastnění by měli mít stejné podmínky pro vypracování dotazníku. Dodržením standardnosti se vylepšuje parametr spolehlivosti a validity metody.

Validita reprezentuje míru platnosti výsledků. V každé metodě dotazníku se projevují systematické i náhodné chyby, jež nám zkreslují výsledný obraz. Proto je třeba validitu empiricky ověřovat, nejlépe porovnáním výsledků vůči něčemu o čem nepochybujeme. Jako příklad si můžeme představit dotazník o předmětu vyplňovaný po ukončení zkouškového období na naší fakultě. Na otázku „Dostupnost slajdů a jiných materiálů k přednáškám“ můžeme dostat od studentů různé odpovědi, přičemž jsme schopni v systému přesně registrovat okamžik, kdy byly slajdy dostupné a empiricky určit tu odpověď o které nepochybujeme.

4.4.3 Struktura dotazníku

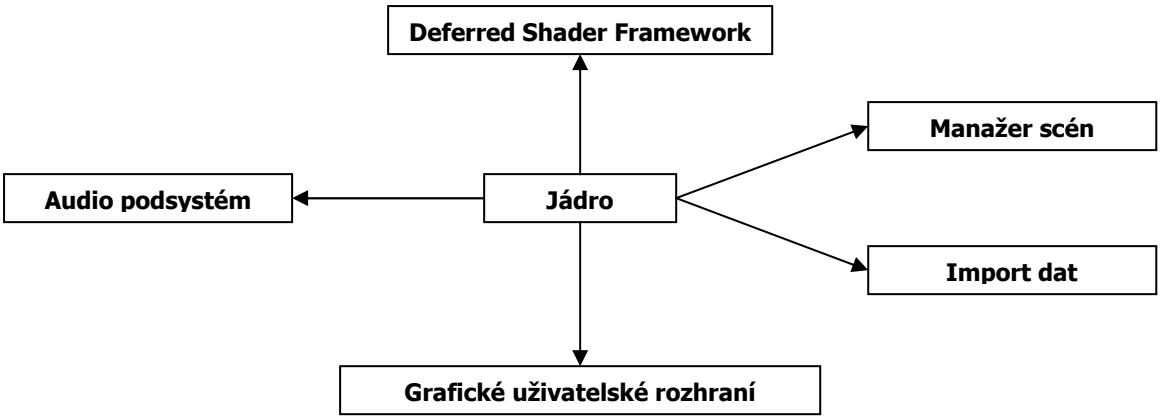
Nyní přejdeme k samotné struktuře dotazníku. Na základě předchozích teoretických východisek o metodě dotazníku a znalosti posuzované diplomové práce bude nejoptimálnější použít škálové otázky. Škálové otázky mohou velmi výrazně zvýšit celkovou objektivnost metody. Požadavek na standardnost bude zajištěn následujícím procesem. Každá osoba vyplňující dotazník bude pozvána do centra výpočetní techniky na fakultě, kde bude spuštěna výuková aplikace, přičemž bude informován, že je potřeba si nejprve přečíst dokument popisující ovládání programu a teprve poté projít výukovou aplikaci sám, bez dotazů na ostatní. Po ukončení výuky pomocí výukové aplikace Deferred Shading bude dané osobě předložen dotazník. Spolehlivost a validitu metody dotazníku nebudu řešit vzhledem k náročnosti na zpracování a přípravu přesahující čas vyhrazený na tuto část diplomové práce.

Dotazník je koncipiován tak, abych získal měřitelné výsledky z oblasti návrhu uživatelského prostředí a kvality výuky. Otázky, které tvoří dotazník lze nalést v příloze A. Statistické výsledky a zhodnocení vyplněných dotazníků je popsáno v závěru této diplomové práce.

5 Implementace

V předcházející kapitole jsme se věnovali návrhu. V této kapitole si popíšeme jednotlivé podsystémy, které bylo nutno naimplementovat než jsme mohli přistoupit k vytvoření scén, animací a demonstrací.

Při implementaci jsem využíval poučky správného objektového návrhu popsaného taktéž v předcházející kapitole. Nejprve si popíšeme podsystém zpřístupňující zvukovou kartu a přehrávající namluvený text. Poté se budeme věnovat architektuře grafického uživatelského prostředí z hlediska implementovaných tříd a mechanismů, díky kterým grafické uživatelské prostředí reaguje na reakce studenta. Dále si rozebereme princip organizace scén a přístup k externím souborům. Poslední část této kapitoly bude věnována implementaci třídy umožňující jednoduchým způsobem aplikovat techniku deferred shading na vizualizaci trojrozměrné scény. Obecný přehled jednotlivých podsystémů implementovaných v rámci diplomové práce lze nalézt na obrázku 5.1.



Obrázek 5.1: Přehled jednotlivých podsystémů aplikace

5.1 Audio podsystém

Audio podsystém je reprezentován jedinou třídou *Audio*. Třída Audio je implementována jako singleton. Singleton je návrhový vzor, který nám umožňuje vytvořit pouze jedinou instanci v celém systému. Díky tomu, že v celém programu máme pouze jednu instanci, můžeme definovat statickou metodu vracející ukazatel na tuto jedinečnou instanci. Pro zpřístupnění zvukové karty jsem využil rozhraní OpenAL.

Třída Audio implementuje tři metody *Play*, *PlayAsync* a *Stop*. Metoda Play přehraje daný zvukový soubor a pozastaví vykonání programu do doby, než bude celá audiosekvence přehrána. Oproti tomu metoda PlayAsync spustí audiosekvenci a okamžitě se ukončí, aby mohl program dále pokračovat. Metoda Stop slouží pro zastavení aktuálně přehrávané audiosekvence.

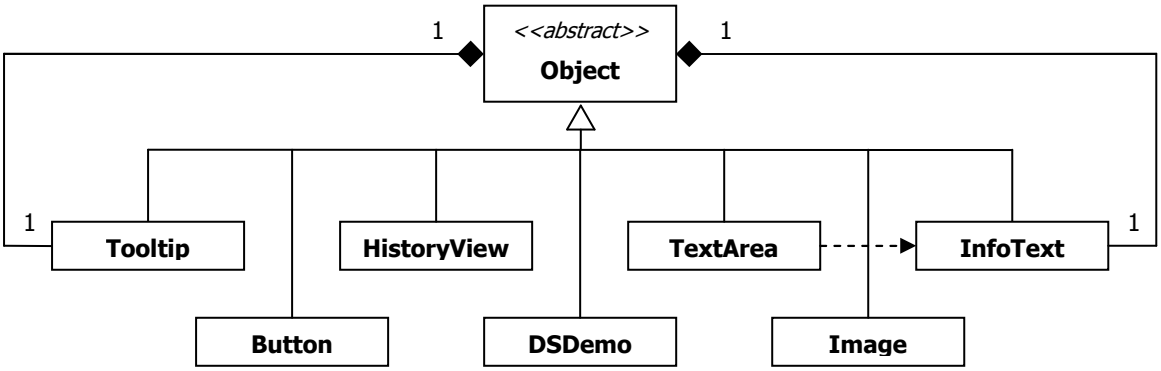
Omezením tohoto audio subsystému je, že dokáže přehrávat pouze audiosekvence uložené ve formátu WAV.

5.2 Grafické uživatelské rozhraní

Implementace grafického uživatelského rohraní lze rozdělit na dvě části. První část tvoří entity uživatelského rozhraní a druhou část tvoří logika.

5.2.1 Entity

Nejprve se tedy zaměříme na třídy tvořící grafické uživatelské rozhraní z pohledu viditelných entit na scéně. Na obrázku 5.2 můžeme vidět diagram tříd tvořících viditelné entity v aplikaci. Jak lze vidět, návrh počítá s vytvořením univerzální třídy, ze které všechny zobrazitelné entity dědí. Tímto způsobem mohu jednoduchým a unifikovaným způsobem zpracovávat všechny entity na scéně. V následujících odstavcích velmi stručně popíši účel každé třídy.



Obrázek 5.2: Diagram tříd reprezentující objekty grafického uživatelského rohraní

Třída **Object** reprezentuje předka všech tříd a obsahuje metody pro získávání obecných informací a dále virtuální metody zodpovědné za aktualizace stavu (metoda *update*), či vykreslení objektu (metoda *Render*). Jak si můžeme všimnout, třída Object využívá vztah kompozice na instance tříd Tooltip a TextInfo.

Třída **Tooltip** zapouzdřuje zobrazení textu a odpovídajícího grafického vzevření v okamžiku, kdy se pozastavíme na určitý časový úsek nad jakoukoliv entitou. Tooltip implementuje virtuální metody *Update* a *Render* zděděné z třídy Object.

Button je klasické tlačítko. Implementuje virtuální metody *update* a *Rrnder*. Zároveň obsahuje tři stavy popisující jeho vzevření. Tlačítko může být neaktivní či aktivní. V případě, že je tlačítko aktivní ještě rozlišujeme, zda-li je uživatelem právě vybrán. Každý tento stav je reprezentován jiným grafickým vzevřením.

HistoryView implementuje třídu Object a zároveň je definován jako singleton. Instance této třídy je zobrazena v horní části aplikace a slouží pro navigaci mezi již prošlými scénami. Pro každou scénu je předem vytvořena miniatura, která se po přechodu na jinou scénu zobrazí na pro tento účel vyhrazených pozicích na obrazovce.

Třída **DSDemo** slouží pro zobrazení scény využívající deferred shading. Dochází zde k nastavení světel, geometrických objektů a dalších informací nutných pro úspěšné zobrazení demonstrační scény. Z této třídy bude pro každou demonstrační scénu zděděna nová třída, ve které bude umístěn zdrojový kód pro logiku dané demonstrace. Třídy zděděné z bázové třídy DSDemo využívají třídy popsané v sekci 5.4 věnované návrhu Deferred Shader Frameworku.

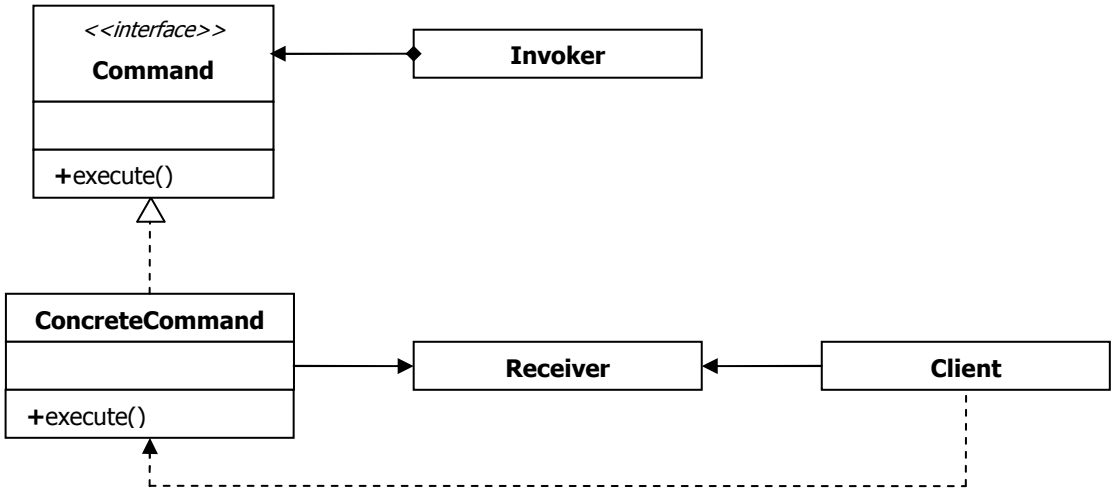
Singleton **TextArea** opět dědí ze třídy Object. Jeho význam spočívá v zobrazení instance třídy TextInfo na obrazovce a spuštění příslušné audiosekvence. V případě, kdy je informace uložená v instanci třídy TextInfo příliš rozsáhlá a nemůže být zobrazena najednou na vyhrazeném místě je potřeba tento text postupně posouvat synchronně s aktuálně namluvenou sekvencí.

InfoText zapouzdřuje detailní informace o textu zobrazovaného v instanci třídy TextView a zároveň obsahuje i informace nutné k identifikaci asociovaného audiovizuálního souboru taktéž použitého ve třídě TextView.

Třída **Image** je nejjednodušší třídou vůbec. V její zodpovědnosti je pouze zobrazení grafické entity na daných souřadnicích.

5.2.2 Logika

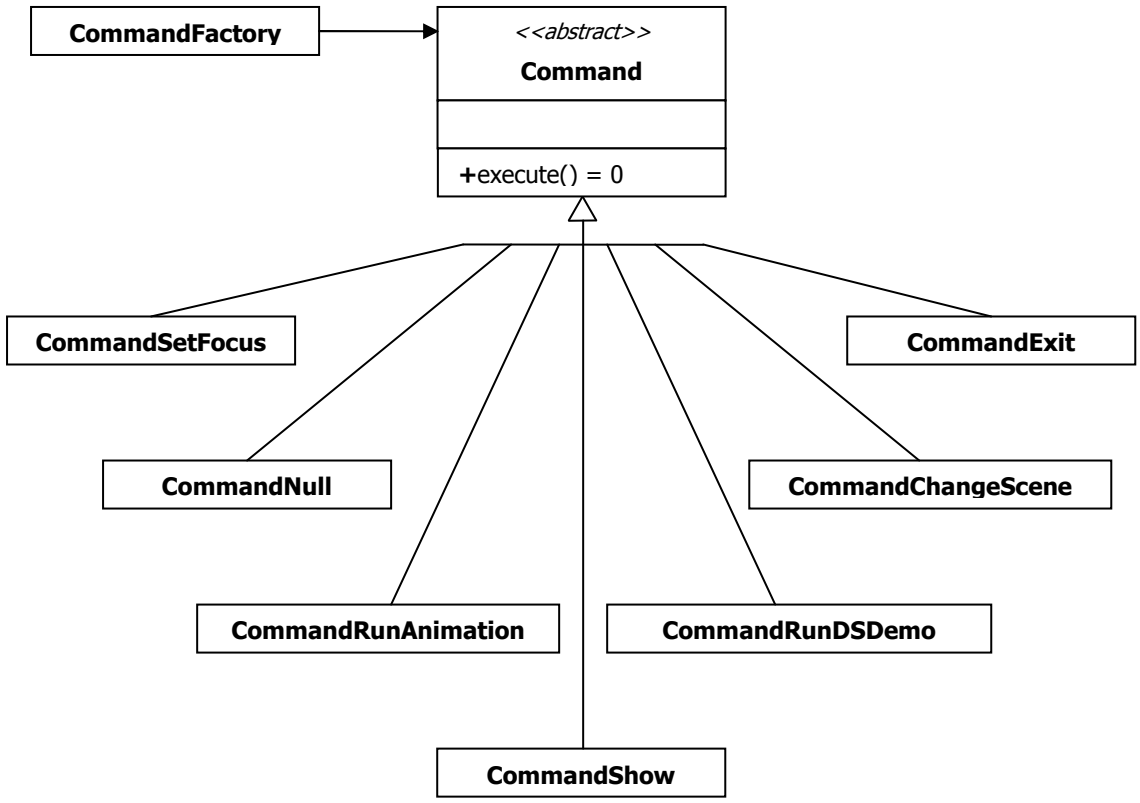
Princip fungování grafického uživatelského rozhraní z hlediska vnitřních vztahů a operací jsem vystavěl na návrhovém vzoru Command [9]. Command je návrhový vzor ve kterém instance třídy reprezentuje a zapouzdřuje všechny informace nutné k vykonání příkazu. Command využívají tři entity a to klient, spouštěč a vykonavatel. Na obrázku 5.3 můžeme vidět UML diagram tříd reprezentující tento návrhový vzor.



Obrázek 5.3: Diagram tříd zobrazující návrhový vzor Command

Klient vytvoří příslušnou instanci třídy Command a zadefinuje její vlastnosti. Spouštěč určí okamžik, kdy dojde ke spuštění příkazu. A poslední vykonavatel je samotná instance třídy, která vykoná daný příkaz.

V případě diplomové práce implementuji abstraktní třídu Command a od ní odvozenou množinu konkrétním příkazu tak jak lze vidět na obrázku 5.4.



Obrázek 5.4: Diagram tříd jednotlivých příkazů

Každá entita popsaná v předcházející podkapitole reaguje na čtyři události a to na stisknutí levého tlačítka myši nad entitou, na stisknutí pravého tlačítka myši nad entitou, na vstup kurzoru myši do oblasti entity a na opuštění kurzoru myši z oblasti entity. Každé této události můžeme nastavit některý z příkazů, který se vyvolá.

Abstraktní třída **Command** pouze definuje virtuální metodu *execute* a slouží jako bázová třída pro všechny ostatní konkrétní příkazy.

CommandFactory je navržen dle idiomu simple factory [9]. Implementuje statickou metodu *getCommand* sloužící pro vytváření instancí konkrétních příkazů na základě žádosti od klienta. Důvodem pro použití CommandFactory je snaha o zapouzdření kódu, který se často mění, do své vlastní třídy.

Třída **CommandSetFocus** je vyvolána v okamžiku vstupu, resp. opuštění kurzoru myši do, resp. z oblasti entity. V závislosti na události je poté dané entitě nastaven příznak určující zaměření entity. Daná entita poté sama může měnit své grafické vzevření na základě tohoto příznaku.

Další třída **CommandNull** implementuje idiom null object [9]. Null object je obvykle jediná instance dané třídy (z tohoto důvodu je tato třída implementována jako singleton), která se používá v okamžicích kdy nemáme z různých důvodů definovány všechny objekty v systému, přičemž se nechceme neustále ptát, zda-li daný objekt není null v případech, kdy se daný objekt používá. V mém případě je instance této třídy nastavena jako reakce entity na události na které nechci nijak reagovat.

CommandRunAnimation vytváří instance zodpovědné za provedení animací popsaných ve 4. kapitole.

Třída **CommandShow** je zodpovědná za vytvoření, resp. zrušení a zobrazení, resp. skrytí entity grafického uživatelského rozhraní. Díky tomuto příkazu můžeme dynamicky vytvářet, či rušit jednotlivé entity na scéně v závislosti na reakcích uživatele.

Nyní se podíváme na třídu **CommandRunDSDemo**. Tento příkaz inicializuje třídu DSDemo a nastaví všechny její atributy. Poté dojde k zobrazení demonstrační ukázky.

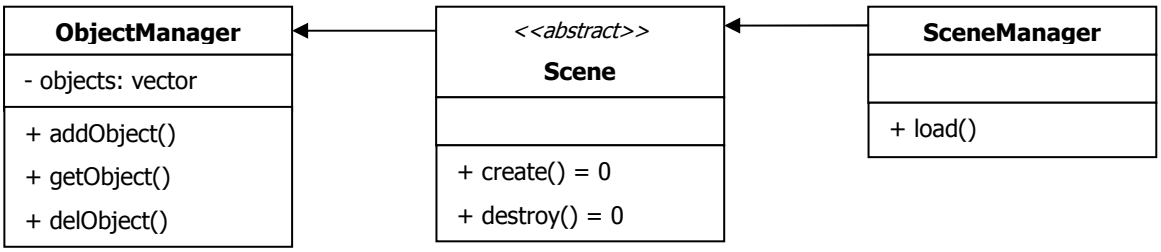
Instance třídy **CommandChangeScene** slouží pro přechod mezi scénami. Aktuální scéna musí být nejprve odstraněna z paměti, poté se vytvoří nová scéna a aktualizuje se entita reprezentovaná třídou HistoryView.

Poslední třída **CommandExit** reprezentuje ukončení ať již aplikace, či jiných činností zadefinovaných mezi vstupními parametry.

5.3 Manažer scén

V této podkapitole se dočteme jakým způsobem dochází k vytváření objektů tvořící entity na scéně.

Obrázek 5.5 představuje diagram tříd zodpovědné za výše zmíněné činnosti. Jedná se o tři třídy pojmenované jako ObjectManager, SceneManager a Scene.



Obrázek 5.5: Diagram tříd zodpovědné za vytváření objektů na scéně a import dat

ObjectManager obsahuje všechny entity zobrazitelné na obrazovce a zároveň poskytuje metody pro jejich přidávání (*addObject()*), získávání (*getObject()*) a odebírání (*delObject()*) objektů.

ObjectManager je též zodpovědný za aktualizaci stavů všech aktivních entit pomocí metody *update()* a vykreslení všech viditelných entit pomocí metody *render()*. Taktéž je zodpovědný za volání destruktoru, resp. odstranění instancí z paměti.

Nejvýznamnější třídou v této podkapitole je **Scene**. Od této bázové třídy jsou zděděny další třídy reprezentující jednotlivé scény aplikace. Tyto zděděné třídy vytváří instance všech entit na scéně dle jejich popisu. Každá scéna má svůj vlastní popis entit, přičemž každá entita má svůj vlastní jedinečný identifikátor, kterým se můžeme na tuto entitu odkazovat. V okamžiku vytvoření instance třídy reprezentující typ entity a nastavení všech atributů této instanci dojde k zavolání metody *addObject()* třídy ObjectManager pro přidání této nové instance do pole všech instancí. Při zavolání metody *destroy()* je poté na základě seznamu entit pro danou scénu volána metoda *delObject()* třídy ObjectManager. Třída, která vytváří jednotlivé instance třídy Scene dle požadavků aplikace se nazývá SceneManager.

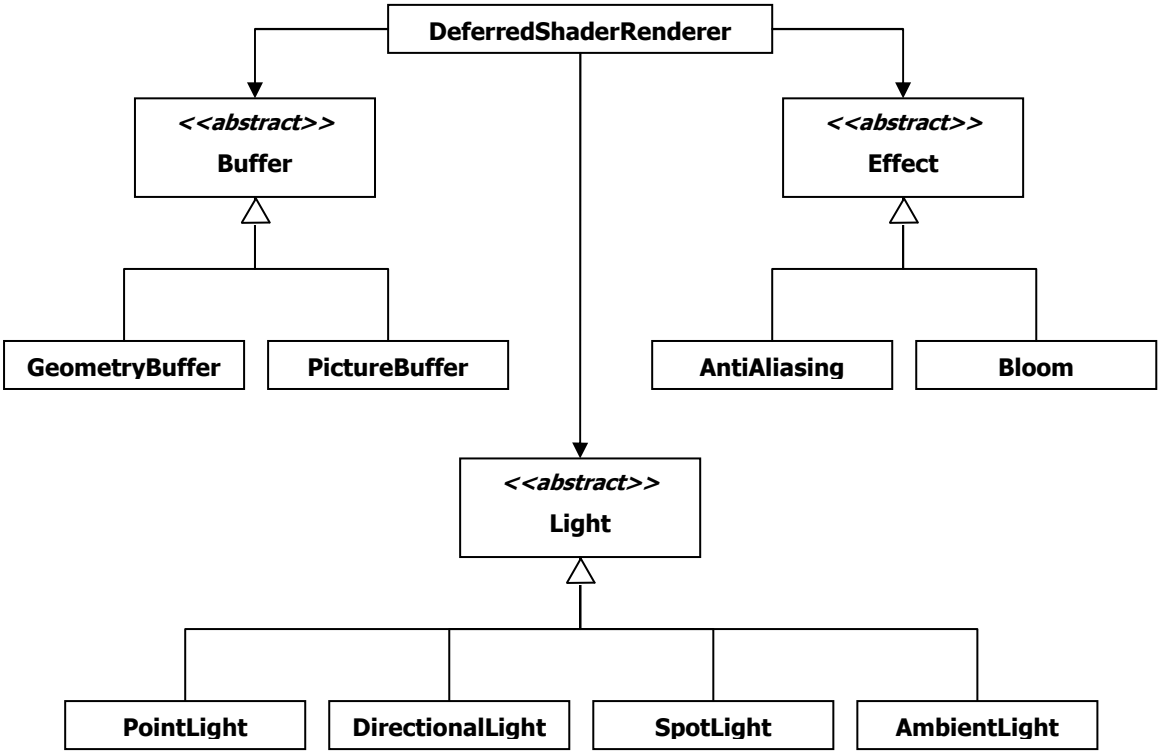
SceneManager je třída vytvářející instance zděděných tříd z bázové třídy Scene dle požadavků aplikace. Tato třída tedy obstarává logiku potřebnou pro změnu scény. Při změně scény nejprve zavolá metodu *destroy()* současné scény a poté vytvoří instanci nové scény a zavolá metodu *create()*.

5.4 Deferred Shader Framework

Dostáváme se ke stěžejní části diplomové práce. Deferred Shader Framework obsahuje kolekci tříd umožnující snadným způsobem využít techniku deferred shading v jakémkoliv projektu. Vzhledem k možnému využití tohoto frameworku i v jiných projektech je návrh frameworku velmi jednoduchý, přehledný a s důrazem na maximální možnost úpravy dle požadavků.

Vykreslovací řetězec je tvořen třemi fázemi. Každá fáze vykreslovacího řetězce závisí na různých objektech. Fáze první, která naplní G-Buffer pomocí geometrie, závisí právě na struktuře G-Bufferu. Ve druhé fázi, kdy se snažíme aplikovat osvětlení jako 2D post efekt, je vše závislé na samotných světlech. Třetí a poslední fáze post-processingu je definována pouze samotnými efekty. Z tohoto důvodu je věnován velký důraz na možnosti různé implementace právě těchto objektů.

Obrázek 5.6 představuje diagram tříd celého deferred shader frameworku. Vidíme zde třídy reprezentující buffer, světla a post-processing efekty. Instance těchto tříd jsou využity právě ve třídě DeferredShaderRenderer v jednotlivých fázích vykreslovacího řetězce. Důležitou poznámkou je, že právě tyto třídy sebou nesou odkaz na použitý vertex a fragment shader program. V následujících podkapitolách zevrubně popíši jednotlivé třídy a jejich funkčnost. Pro podrobnější informace je třeba prostudovat okomentované zdrojové soubory.



Obrázek 5.6: Diagram tříd deferred shader frameworku

5.4.1 Buffer

Třída **Buffer** představuje abstraktní třídu, ze které poté dědí třídy GeometryBuffer a PictureBuffer. Definuje metody, které umožní nastavit daný buffer buďto jako vykreslovací cíl, do kterého se zapisuje (*bindAsRenderTarget()*), či jako texturu ze které se údaje čtou (*bindAsTexture()*).

GeometryBuffer reprezentuje nám známý G-Buffer. Jedná se o kolekci bufferů s pevně určeným formátem dat. Zároveň tato třída poskytuje metody, pro nastavení vertex a fragment shader kódu, který se použije ve fázi vykreslení geometrie. Důvodem proč je příslušný vertex a fragment shader kód obsažen v této třídě místo, aby se nastavoval ve třídě DeferredShaderRenderer, je co největší oddělení proměnlivých částí kódu od samotného jádra frameworku. Je důležité si uvědomit, že při použít techniky deferred shading vždy zapisujeme do G-Bufferu a jenom my víme jaké data potřebujeme ukládat.

Další třída **PictureBuffer**, neboli P-Buffer je jednoduchý buffer, ve kterém jsou uloženy pouze informace o barvě každého pixelu. P-Buffer může stejně jako G-Buffer sloužit buďto pro čtení, či pro zápis.

5.4.2 Light

Dostáváme se k druhé fázi vykreslovacího řetězce reprezentovaného výpočtem osvětlení na základě informací uložených v G-Bufferu. Abstraktní třída **Light** definuje metody, které musí každé světlo implementovat pro korektní běh programu, jako například metodu *apply()*. Stejně jako v případě třídy GeometryBuffer i zde nastavujeme pro všechny zděděné třídy cestu ke zdrojovému kódu vertex a fragment shaderu.

Třída **PointLight** popisuje chování všesměrového světla. Tomuto typu světla definujeme pozici, faktor utlumování světelných paprsků a příslušnou barvu vyzařovaného světla.

DirectionalLight představuje směrový typ světla. Nastavuje tedy směr odkud světlo přichází a informace o barvě světla.

Reflektorový typ světla je implementován ve třídě **SpotLight**. Světlu musíme nastavit pozici, směr vyzařovaného světla, maximální úhel paprsku světla ve vztahu ke směru světla a samozřejmě i barvu světla.

Posledním typem světla je nesměrové světlo popsané ve třídě **AmbientLight**. Pro tento typ světla je potřeba nastavit pouze barvu světla.

5.4.3 Effect

Post-processing efekty se aplikují v poslední fázi vykreslovacího řetězce. Efekt může vycházet pouze z informací uložených v P-Bufferu vytvořeného z fáze výpočtu osvětlení, ale zároveň se může dotazovat i na informace uložené v G-Bufferu. Abstraktní třída **Effect** opět pouze poskytuje metody nutné pro správné vykonání. Mezi tyto metody se řadí *setPriority()*, která nastavuje každému efektu prioritu při zařazování do fronty efektů. Čím větší je číslo, tím větší má efekt prioritu a je umístěn více na počátek fronty. Důvodem této priority je fakt, že některé post-processing efekty se mohou navzájem negativně ovlivňovat, přičemž použití priorit nám dává možnost takovým situacím zabránit jednoznačným určením pořadí aplikovaných efektů. Stejně jako u bufferů a světel nastavujeme cestu k příslušným vertex a fragment shader programům právě u těchto efektů.

AntiAliasing efekt se snaží odstranit alias efekt na hranách objektů. Pro tento druh efektu nastavujeme faktor váhy. Faktor váhy je stručně vysvětlen v podkapitole 4.3.11.

Poslední dobou hojně v počítačových hrách využívaný efekt je reprezentován třídou **Bloom**. Aplikace tohoto efektu má za následek zvýraznění kontrastu u osvětlených oblastí a utlumení kontrastu v temných oblastech. Díky tomu, že tato třída pracuje výhradně na základě informací uložených v P-Bufferu není třeba nastavovat jakýkoliv parametr a zdrojový kód vertex i fragment shaderu bude vždy totožný.

5.4.4 DeferredShaderRenderer

Nyní se dostáváme k jádru celého frameworku. Třída **DeferredShaderRenderer** se stará o přístup k shader jednotkám na grafické kartě, nastavuje veškeré atributy OpenGL nutné pro správné vykreslení objektů a spouští jednotlivé fáze vykreslovacího řetězce.

Co se atributů třídy DeferredShaderRenderer týče, tak obsahuje jeden buffer třídy PictureBuffer, do kterého se zapisuje ve fázích výpočtu osvětlení a post-processingu. Tento P-Buffer tvoří finální obraz scény.

Nejprve si popíšeme jakým způsobem vykonáme fázi vykreslení geometrie. Tato fáze je odlišná od dalších dvou fázi z hlediska použití třídy DeferredShaderRenderer. Provedení této fáze je podmíněno použitím dvou metod. Za prvé musíme použít metodu *beginGeometryStage()*, která přebírá jako parametr instanci třídy G-Buffer. Poté můžeme vykreslovat geometrii způsobem, který odpovídá popisu vstupů u vertex a fragment shader programů. Jakmile vykreslíme všechny objekty, musíme zavolat metodu *endGeometryStage()*. Zavoláním této metody ukončíme fázi vykreslení geometrie, přičemž G-Buffer bude naplněn odpovídajícími hodnotami. Důvodem pro vytvoření dvou metod je oddělení samotného rendereru od popisu geometrie, která musí být dodána zvlášť.

Dalším krokem je výpočet osvětlení. Celý krok je proveden metodou *performLightingStage()*. Je potřeba předat instanci třídy G-Buffer získané v předcházejícím kroku zároveň se seznamem instancí světel. Seznam světel je reprezentován pomocí STL datového kontejneru vector. Instance světel jsou vytvořena uživatelem dle jeho preferencí. Optimalizací budiž setřídění seznamu světel před vykonáním samotného procesu výpočtu osvětlení způsobem zajišťujícím co nejmenší počet výměn vertex a fragment programů na grafické kartě. Poté je pro každé světlo vykreslen odpovídající zjednodušený model a shader programy jsou aplikovány pouze na pixely scény v dosahu světla.

Poslední fází je aplikace post-processing efektů. Obdobně jako v předcházející fázi se volá pouze jediná metoda *performPostProcessStage()*. Parametry tvoří naplněný G-Buffer a seznam instancí třídy Effect. Efekty jsou před aplikováním na výsledný obraz seřazeny do fronty dle priority jednotlivých efektů.

Nakonec je potřeba akorát přesunout obsah P-Bufferu třídy DeferredShaderRenderer do framebufferu, aby došlo k jeho vykreslení na obrazovku o což se postará metoda *flush()*.

5.4.5 Zhodnocení

Deferred Shader Framework je díky malému počtu tříd velmi jednoduchý a zároveň dostatečně flexibilní na nejrůznější použití. Kvalita tohoto frameworku může být prokázána až jejím použitím v dalších projektech. Pro moji diplomovou práci je tento framework naprosto dostačující a plně vyhovuje mým požadavkům.

6 Závěr

Deferred shading patří mezi znovu objevené vykreslovací metody, které je nyní možné díky evoluci grafických karet aplikovat. Předností je oddělení části zodpovědné za vykreslování objektů od části vypočítávající osvětlení a stínování. Stínování je takzvaně odloženo.

Bohužel diplomová práce nesplnila všechny požadavky, které jsem na ni osobně kladl. Lokalizace není dosud naimplementována, ačkoliv je návrh tříd natolik uzpůsoben, že by mohla být v rámci budoucího rozšíření jednoduše doimplementována.

Na druhou stranu se podařilo vytvořit interaktivní učební pomůcku, která studentům objasní techniku deferred shading zajímavým a neotřelým způsobem. Z tohoto hlediska diplomová práce splnila všechny požadavky a tedy i zadání.

Součástí diplomové práce je i statistické vyhodnocení dotazníku definovaného v příloze A. Dotazník nakonec vyplnilo šest osob mužského pohlaví ve věku od 23 let do 26 let. Výsledky jsou uvedeny v tabulce 6.1.

Otázka	Odpověď			
	Ano	Spíše ano	Spíše ne	Ne
Bylo pro Vás ovládání aplikace intuitivní?	5	1		
Uvítal jste namluvení aplikace?	3	2		1
Myslíte si, že by byl ženský hlas lepší na namluvení?	6			
Byla pro Vás navigace mezi scénami jednoduchá?	4	2		
Zdá se Vám současné členění informací přehledné?	3	2	1	
Myslíte si, že mluvený text je dostatečně obsáhlý?	2	3	1	
Pochopil jste jak technika deferred shading funguje?	3	3		
Pomohli Vám animace lépe pochopit situaci na scéně?	4	2		
Byly pro Vás demonstrační ukázky názorné?	2	4		
Uvítal byste větší počet demonstračních ukázek?	1	4	1	
Pomohli Vám demonstrační ukázky lépe pochopit látku?	5	1		

Tabulka 6.1: Odpovědi na dotazník

.Na základě těchto odpovědí můžeme nyní vydedukovat některé poznatky. Vzhledem ke vztahu osob jež vyplňovaly dotazníky k mé osobě lze bohužel vidět velmi kladné odpovědi, což poněkud znehodnocuje celý dotazník. Pokud bych se ale pokusil statisticky vyhodnotit odpovědi se zanedbáním vlivu známosti, tak výsledky jsou následující. Grafické uživatelské rozhraní je navrženo velmi dobře. Studenti nemají prakticky žádný problém s ovládáním aplikace. Namluvení aplikace se líbilo, i když by všichni uvítali spíše ženský hlas. Co se výukové části týče, tak prakticky všichni

pochopili jakým způsobem technika deferred shading funguje. Demonstrační ukázky a animace tvořili velmi významný prvek pro pochopení výuky.

Z pohledu dalšího vývoje projektu jsem již zmínil implementaci lokalizace, která by umožnila rozšířit tuto učební pomůcku i do jiných zemí. Další možnost vývoje projektu vidím ve vytvoření unifikovaného rozhraní pro tvorbu učebních pomůcek na základě nyní definované struktury. Toto rozhraní by poté mohlo být využito pro tvorbu dalších interaktivních učebních pomůcek. Zároveň je možné i tuto učební pomůcku nadále rozšiřovat. Může se rozšířit oblast post-processing efektů o nové efekty, přidat oblast optimalizace deferred shadingu. Taktéž je potřeba zmínit možnosti dalšího vývoje deferred shader frameworku. V současné podobě je framework velmi jednoduše navržen pro účely výukové aplikace. Je možné naimplementovat podporu i pro další techniky deferred shadingu jako jsou možné například u nefotorealistického vykreslování či jako možnost urychlení techniky volume rendering. Dále vidím možnosti rozšíření v implementaci stínů a průhlednosti.

Tvorba interaktivní učební pomůcky osvětlující techniku zpožděného stínování byla časově náročná a zároveň i velmi poučná. Pochopil jsem, že není jednoduché podat předem připravenou látku způsobem, který by všichni univerzálně chápali. Určité oblasti výuky jsou závislé na znalostech, které by již student měl znát. Bez možnosti vstupu učitele do procesu výuky zároveň i existuje pravděpodobnost, že reprezentované informace budou špatně, či mylně interpretovány.

Dle mého názoru bude budoucnost výuky patřit právě podobným samostatným učebním pomůckám. Samozřejmě již tyto učební pomůcky budou mnohem vyspělejší, propojené k decentralizované databázi veškerého poznání lidstva, kdy s rozvojem expertních systémů a vývojem pokročilých umělých inteligencí se budou tyto učební pomůcky schopny přizpůsobit znalostem každého studenta individuálně a předkládat mu danou látku pro něho tím nejoptimálnějším způsobem.

Literatura

[1] Morkesová, K.: *Vyučovací metody*. Ostravská univerzita, 2005.

[2] Belz, H., Siegrist, M.: *Klíčové kompetence a jejich rozvíjení*. Praha, 2001.

[3] Žára, J., Beneš, B., Sochor, J., Felkel, P.: *Moderní počítačová grafika*. Brno, Computer Press, 2004.

[4] Shishkovtso, O.: *GPU Gems 2 – Chapter 2 - Deferred Rendering in S.T.A.L.K.E.R.*, Addison-Wesley, 2005, s. 345 - 366.

[5] Koonce, R.: *GPU Gems 3 – Chapter 3 - Deferred Shading in Tabula Rasa*. Addison-Wesley, 2007, s. 429 – 459.

[6] Valient, M.: *Deferred Rendering in Killzone 2* [online]. Brighton, 2007.
URL: http://www.guerrilla-games.com/publications/dr_kz2_rsx_dev07.pdf

[7] Calver, D.: *Photo-realistic Deferred Lighting* [online]. Beyond3D, 2003.
URL: http://www.beyond3d.com/content/articles/19/

[8] Gruber, L.: *DeShade – A Deferred Shading Frame Work for Complex Lighting*. Graz University of Technology, 2008.

[9] Pecinovský, R.: *Návrhové vzory*. Brno, Computer Press, 2007

[10] McConnell, S.: *Dokonalý kód*. Brno, Computer Press, 2006

[11] Kohoutek, R.: *Metoda dotazníku pro pedagogy*. Brno, CERM akademické nakladatelství, 1998

[12] Kohoutek, R.: *Dotazník* [online].
URL: http://www.ped.muni.cz/wpsy/stud_materialy/koh_dotaznik.htm

[13] Cortes, L.: *Designing a Graphical User Interface* [online]. New York, 1997
URL: http://www.medicalcomputing.org/archives/0agui.php

[14] Myers, Brad A.: *Why are Human-Computer Interfaces Difficult to Design and Implement?* [online] Pittsburgh, 1993
URL: http://reports-archive.adm.cs.cmu.edu/anon/1993/CMU-CS-93-183.ps

[15] Hobart, J.: *A Principles of good GUI Design* [online]. 1998
URL: http://axp16.iie.org.mx/Monitor/v01n03/ar_ihc2.htm

[16] FiringSquad: *FS Guides: Occlusion Culling* [online], 2002
URL: http://www.firingsquad.com/guides/occlusionculling/page2.asp

Seznam příloh

Příloha A

Dotazník k aplikaci Deferred Shading Tutorial

	ANO	SPÍŠE ANO	SPÍŠE NE	NE
1. Bylo pro Vás ovládání aplikace intuitivní?	☐	☐	☐	☐
2. Uvítal jste namluvení aplikace?	☐	☐	☐	☐
3. Myslíte, že by byl ženský hlas vhodnější pro namluvení?	☐	☐	☐	☐
4. Byla pro Vás navigace mezi scénami jednoduchá?	☐	☐	☐	☐
5. Zdá se Vám současné členění informací přehledné?	☐	☐	☐	☐
6. Myslíte si, že mluvený text je dostatečně obsáhlý?	☐	☐	☐	☐
7. Pochopil jste jak technika deferred shading funguje?	☐	☐	☐	☐
8. Pomohli Vám animace lépe pochopit situaci na scéně?	☐	☐	☐	☐
9. Byly pro Vás demonstrační ukázky názorné?	☐	☐	☐	☐
10. Uvítal byste větší počet demonstračních ukázek?	☐	☐	☐	☐
11. Pomohli Vám demonstrační ukázky lépe pochopit látku?	☐	☐	☐	☐