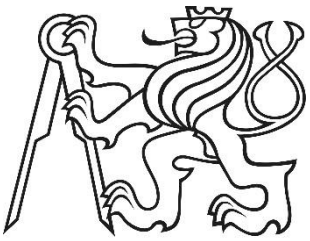
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra kybernetiky

Model města pro simulátor auta

Vladimír Čajka

Studijní program: Otevřená informatika (bakalářský)
Obor: Informatika a počítačové vědy

Vedoucí práce: Ing. David Sedláček, Ph.D.

10. ledna 2017

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Vladimír Č a j k a

Studijní program: Otevřená informatika (bakalářský)

Obor: Informatika a počítačové vědy

Název tématu: Model města pro simulátor auta

Pokyny pro vypracování:

1. Analyzujte požadavky na vnější scénu prostředí pro VR simulátor interiéru auta v systému VRUT založeném na raytrace rendereru.
2. Navrhněte rozsah scény (mapu) a základní grafický ráz budov. Vytvořte modely prostředí s ohledem na realtime zobrazení v systému VRUT. Modely nebudou vytvořeny jako low-poly, ale budou respektovat možnosti systému VRUT s předpokladem na celkový rozsah scény. Při vytváření modelů využijte ukládání specifických vlastností modelu do asociovaných textur (např. normal mapy, světelné mapy, atd.). Model bude vytvořen kombinací ruční práce v 3D modeláři a procedurálního generování v programu City Engine (CE).
3. V rámci textové části práce popište navržený řetězec pro tvorbu modelu (CE - 3D studio – VRUT), CGA gramatiky, a použité speciální techniky pro dosažení realistického zobrazení.

Seznam odborné literatury:

[1] http://www.cgg.cvut.cz/members/bittner/vrut/doc_cz.pdf
[2] Martin Šembera: CGA - příručka modeláře, BP FEL 2012 – http://leyfi.felk.cvut.cz/courses/mvr/CGA_prirucka/
[3] Jiří Žára, Bedřich Beneš, Jiří Sochor, Petr Felkel: Moderní počítačová grafika. Computer Press, 2005.

Vedoucí bakalářské práce: Ing. David Sedláček, Ph.D.

Platnost zadání: do konce letního semestru 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 17. 2. 2015

Poděkování / Prohlášení

V úvodu této bakalářské práce bych chtěl poděkovat svému vedoucímu práce Ing. Davidu Sedláčkovi, Ph.D. za připomínky, cenné rady a metodické vedení při jejím zpracování.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací

V Praze dne 10. ledna 2017

…….

Abstrakt / Abstract

Předmětem této bakalářské práce je vytvoření modelu města jako vnější scény pro VR simulátor interiéru automobilu.

Modelování dílčích objektů je prováděno dílem metodou polygonálního a dílem metodou procedurálního modelování.

U výsledného modelu, vytvářeného s důrazem na vysokou míru realističnosti modelovaných objektů, tato bakalářská práce ověřuje možnosti jeho vykreslování v reálném čase při využití různých renderovacích způsobů a tím i jeho vlastní použitelnost v modulu automobilového simulátoru vizualizačního nástroje VRUT.

Klíčová slova:
3D modelování, polygon, renderování, textura, vizualizace, virtuální realita

Main subject of the thesis is creation of external scene of a city model for VR car interior simulator.

Modeling of objects is a combination of polygonal and procedural modeling methods.

Resulting model is created with an emphasis on a high degree of realism of modeled objects. This bachelor thesis verifies possibilities of real time rendering using various rendering techniques and thus its own applicability in a VRUT visualization tool and it's driving simulator module.

Keywords:
3D modeling, polygon, rendering, texture, visualization, virtual reality

Obsah

Seznam obrázků

Kapitola 1
Úvod

V posledních letech proniká virtuální realita do stále většího počtu lidských činností a pracovních oborů. Modelování nejrůznějších objektů se používá nejen v oblastech zábavy (počítačové hry, filmy nebo reklama), ale velmi efektivně se uplatňuje i ve vědeckých, technických, dopravních a designerských oborech, v architektuře a urbanismu, zejména při simulaci nejrůznějších situací, mezních a krizových stavů a činností, kde virtuální realita představuje prostředí levnější, kdykoli dostupné a hlavně bezpečnější, než může být skutečné reálné prostředí.

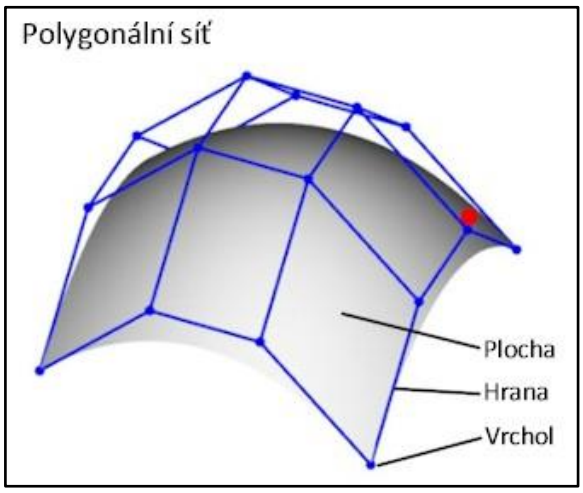
V souvislosti s intenzivním vývojem v oblasti výpočetní techniky a stále rostoucím a lépe dostupným výpočetním výkonem se zvyšuje poptávka po vytváření stále dokonalejších a realističtějších modelů, které se při nejrůznějších formách simulace virtuálního prostředí používají a stále více přibližují skutečné podobě a podmínkám reálného světa.

V rámci projektu VRUT, který byl vyvíjen ve spolupráci Fakulty elektrotechnické ČVUT a společnosti Škoda Auto, je stále používán modul automobilového simulátoru, který je provozován ve spojení s poměrně jednoduchým a málo realistickým modelem vnějšího prostředí, ve kterém se simulátor pohybuje jako ve virtuálním světě. V této souvislosti vzniklo zadání této bakalářské práce, které má za úkol vytvořil realistický 3D model města jako vnějšího prostředí pro tento simulátor a při jeho vytváření analyzovat a ověřit možnosti současných modelovacích metod a technik.

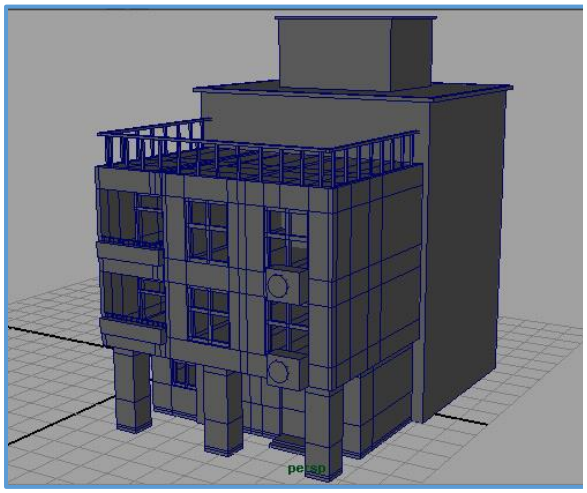
Kapitola 2
Metody 3D modelování

2.1 Polygonální modelování

Polygonální modelování je způsob modelování objektů založený na aproximaci jejich povrchu pomocí polygonů - mnohoúhelníků. Každý polygon se skládá ze tří nebo více vrcholů, reprezentovaných souřadnicemi v 3D prostoru, hran, které je spojují a ploch, které jsou těmito hranami ohraničeny. Modifikací těchto tří komponent jsou polygony vytvářeny a upravovány. Jednotlivé polygony spolu běžně sdílí vrcholy a hrany a vytváří tak povrch modelu, tzv. polygonální síť.



Obr. 2.1 Princip polygonálního modelování



Obr. 2.2 Polygonální model budovy

Při tvorbě modelu je nejprve postupně vytvořena polygonová síť ve 3D modelovacím nástroji. Následně tuto síť modelář pokryje povrchovými texturami a použije funkcionality, které aplikují vliv způsobu osvětlení, případně i jiné speciální efekty, které přispějí k vytvoření realistického výstupu.

Polygonové modelování je pro vytvoření objektů s nepravidelným a složitým povrchem a pro modelování objektů, u kterých není vyžadována naprostá přesnost, pro modeláře univerzálnější a jednodušší. Vykreslování polygonového modelu je také relativně rychlé.

Pro modelování objektů, jejichž tvar je založen na pravidelných a jednoduchých křivkách není tento způsob modelování nejvhodnější, protože zakřivené povrchy lze pouze aproximovat z konečného počtu rovných hran a polygonovým modelem nikdy nelze vytvořit dokonalé křivky.

Pro účely vytváření **neinteraktivní grafiky**, která nemusí reagovat v reálném čase na změny vstupů a může být vykreslována před vlastní prezentací, je možné zpracovat model s dostatečným počtem polygonů, aby nebyl znatelný rozdíl mezi dokonalou tvarovou křivkou modelovaného objektu a její polygonovou aproximací (high-poly). V tomto případě

se obvykle při vytváření modelu postupuje způsobem, kdy se zakřivené plochy modelují ručně do podoby hrubé polygonové sítě, která přibližně vystihuje tvar modelovaného objektu a následně se s vhodným stupněm vyhlazení aplikuje metoda (subdivision algoritmus), která vytvořené hranaté plochy rozdělí na příslušný počet menších polygonů a tím polygonovou síť modelu dostatečně zjemní a hranaté tvarové křivky žádoucím způsobem vyhladí.

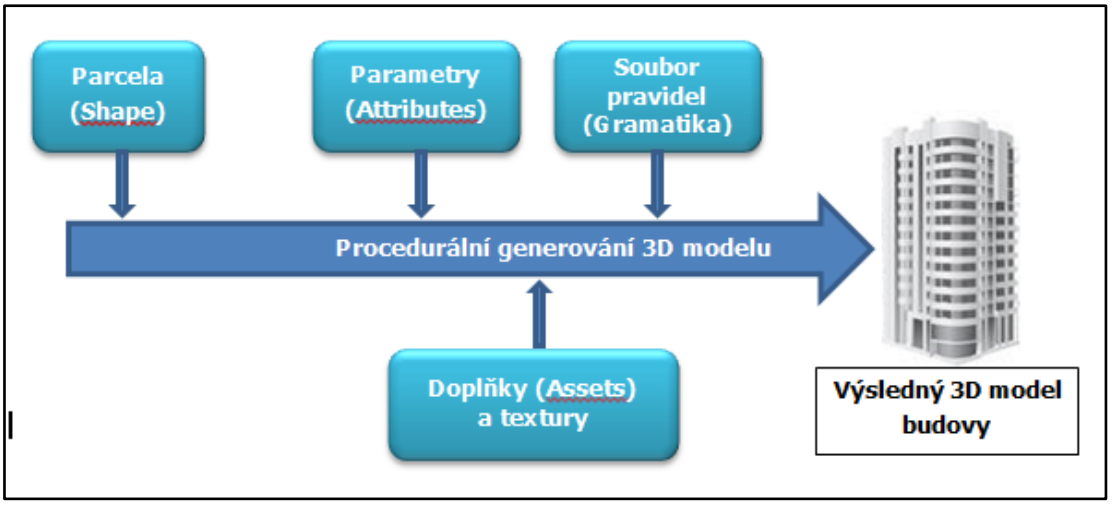
Při modelování objektu pro **interaktivní grafiku**, která musí v reálném čase reagovat na vstupy, je základním parametrem počet polygonů a v zásadě je snaha jej při vytváření modelu minimalizovat (low-poly). Polygonový limit je různý pro každý model a stanovuje se v závislosti na výkonu cílového hardwaru, složitosti celé modelované scény a na tom, zda je příslušný modelovaný objekt pro celou vytvářenou scénu klíčový a dominantní nebo se jedná pouze o modelování objektu v pozadí. V modelování objektů pro tento typ grafiky se prakticky nepoužívá subdivision algoritmus a malé tvarové detaily se obvykle nemodelují, ale zobrazují se pomocí textur. Pro zvyšování realističnosti vytvářených modelů se pro tento typ grafiky používají další nástroje a triky jako například textury typu bump map nebo normal map, kdy se mění osvětlení povrchu a vytvářejí se tak efekty složitějšího povrchu modelů. [16]

2.2 Procedurální modelování

Procedurální modelování je vhodnou metodou pro vytváření velmi složitých modelů. Umožňuje generování modelovaných objektů z jeho jednoduchých komponent s využitím určitých pravidel, které jsou závislé na vstupních parametrech.

Procedurální modelování na rozdíl od ostatních spíše manuálních metod, které vyžadují přímou spolupráci modeláře, je automatickým procesem, kdy jsou modely vytvářeny pomocí předem definovaných algoritmů. V těchto algoritmech jsou obsaženy požadavky modeláře a tyto algoritmy určují, jak se má příslušný 3D model vytvořit.

Tento způsob modelování je velmi dobře použitelný pro složené modelářské kompozice například pro modelování městského prostředí, kde se často uplatní modelace většího množství 3D objektů, kde se opakuje větší množství prvků, relativně jednoduchých tvarů nebo struktur. [17]



Obr. 2.3 Schéma procesu procedurálního modelování budovy

Postup vytváření 3D modelu města s využitím nástrojů procedurálního modelování je obvykle složený proces, který zahrnuje tyto základní kroky:

1. vytvoření územního plánu města a definice terénu,
2. vytvoření sítě komunikací,
3. rozdělení vzniklých bloků na jednotlivé parcely,
4. vymodelování budov a dalších městských prvků na jednotlivých parcelách.

Základními typy používaných typů souborů pravidel (gramatik) historicky byly a jsou Shape Grammars, L-systém, Split Grammar a CGA Shape.

Významné přednosti procedurálního modelování jsou:

- možnost tvorby široké palety výstupních modelů na základě několika vstupních parametrů,
- tímto způsobem lze rychle vytvořit náročné a komplexní modely,
- datová nenáročnost (komprese), generování modelu je prováděno až na vyžádání,
- tento způsob modelování není tak náročný na lidskou práci a je i finančně efektivnější.

Nevýhodou procedurálního způsobu modelování je:

- poměrně obtížné správně nadefinovat vstupní parametry a gramatiku tak, aby bylo dosaženo žádoucího výstupu a výsledky mohou být až nerealistické,
- podoba výstupního modelu je velmi citlivá i na malé změny vstupních parametrů,
- vstupní parametry a pravidla pro generování modelu se nedají intuitivně upravovat.

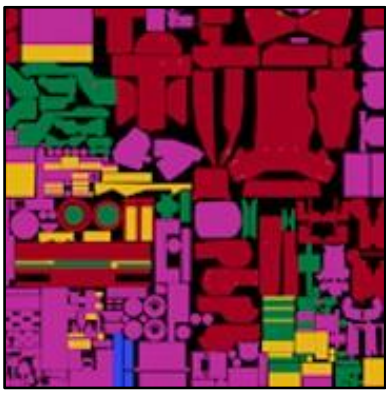
Kapitola 3
Materiály a textury

Každý objekt ve scéně by měl mít přiřazen materiál, aby dosáhl požadovaného vzhledu. Konečný materiál může být vytvořen kombinací více textur.

Pojmem textura je většinou míněn 2D obrázek, který lze určitým způsobem nanést (namapovat) pomocí texturových souřadnic na povrch 3D modelu, aby model nebyl jen bezbarvým tvarem. Existují také 3D textury, které nepotřebují texturovací souřadnice, ale vypočítávají výslednou barvu pixelu pomocí algoritmu. Takové však v této práci nebyly použity. Pro docílení větší realističnosti modelovaných objektů mohou modeláři využívat různé typy textur, které jsou schopny představovat vlastnosti povrchů, vyrobených z různých materiálů a měnit barvu povrchu v závislosti na pozici světla a úhlu pohledu pozorovatele.

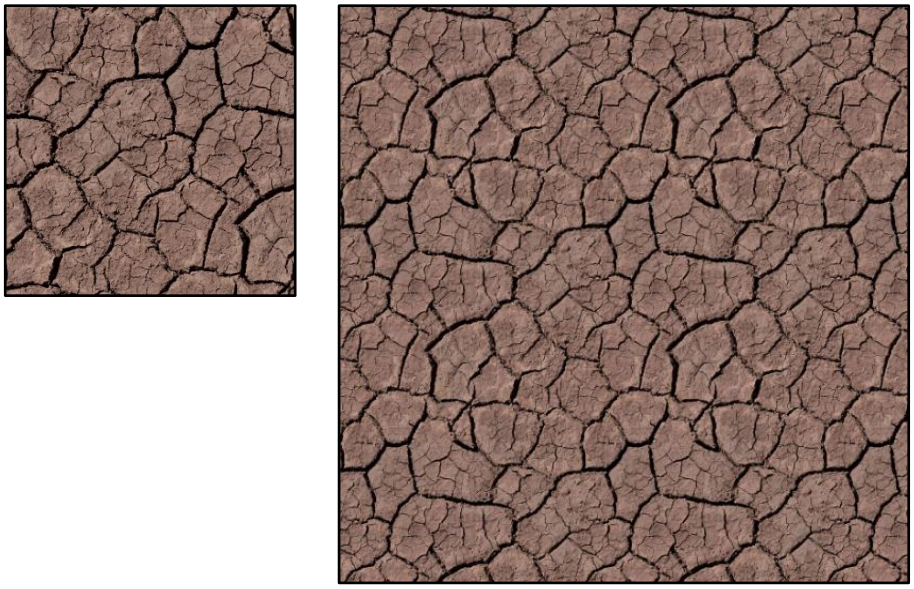
3.1 Color (Diffuse) map

Color map je základní typ textury, která definuje barvu a materiál v daném místě modelu. Materiál může mít více color map s různými mapovacími souřadnicemi. Tento typ textury se dá často použít i jako základ pro jiné typy textur, jelikož v místech, kde dochází ke změně barvy, také často dochází ke změně jiných vlastností povrchu modelu, jako například odrazivosti světla (specular map) či nerovnosti povrchu (bump map).



Obr. 3.1 Příklad textury typu color map [21]

U mnoha objektů můžeme pozorovat opakující se vzor po celém povrchu, například cihlová zeď, dlažební kostky a další. V takovém případě je výhodné použít tzv. opakovatelnou texturu, jejíž protilehlé hrany na sebe plynule navazují. Taková textura je pak několikrát promítnuta na povrch.



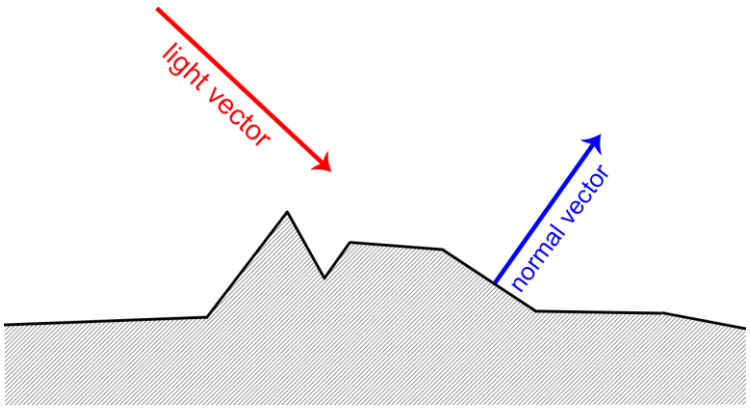
Obr. 3.2 Ilustrativní znázornění použití opakovací textury [22]

3.2 Bump, Normal a Displacement map

V dnešní době je již běžné používání textur, které modelu přidávají dodatečný detail. Ve skutečném světě se jen málo věcí skládá z dokonale rovných tvarů, proto se vhodně používají tyto textury k zobrazení miniaturních nerovností na povrchu modelu, které se nevyplatí polygonálně modelovat, jednak z přílišné složitosti nebo kvůli přílišné zátěži na systém při vlastním vykreslování výsledného obrazu.

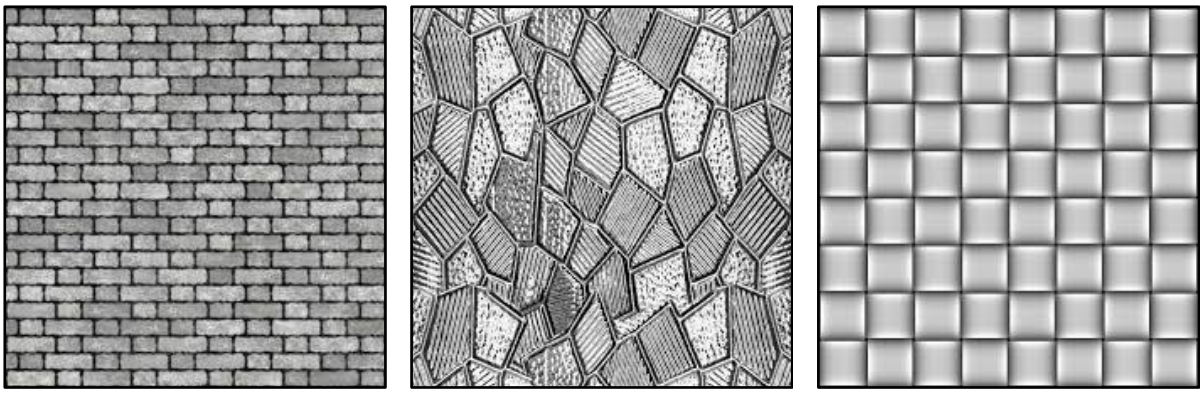
Bump map

Bump map je nejstarší a nejjednodušší ze zde zmíněných typů textur. Je reprezentována typicky obrázkem ve stupních šedi a v 8-bitové barevné hloubce. Je-li záměrem, aby daný bod působil dojmem, že je zapuštěný více v modelu, je mu přiřazena na bump mapě tmavší barva. Naopak čím světlejší bude oblast na bump mapě, tím více z modelu bude vystupovat. Při 50% šedé barvy se nepřidá žádný detail. Při vypočítávání osvětlení modelu pak dochází k vypočítání normály dané bump mapy, která se potom zkombinuje se skutečnou normálou na modelu v daném místě a použije tuto kombinaci jako konečnou normálu povrchu.



Obr. 3.3 Ilustrativní znázornění bump map textury

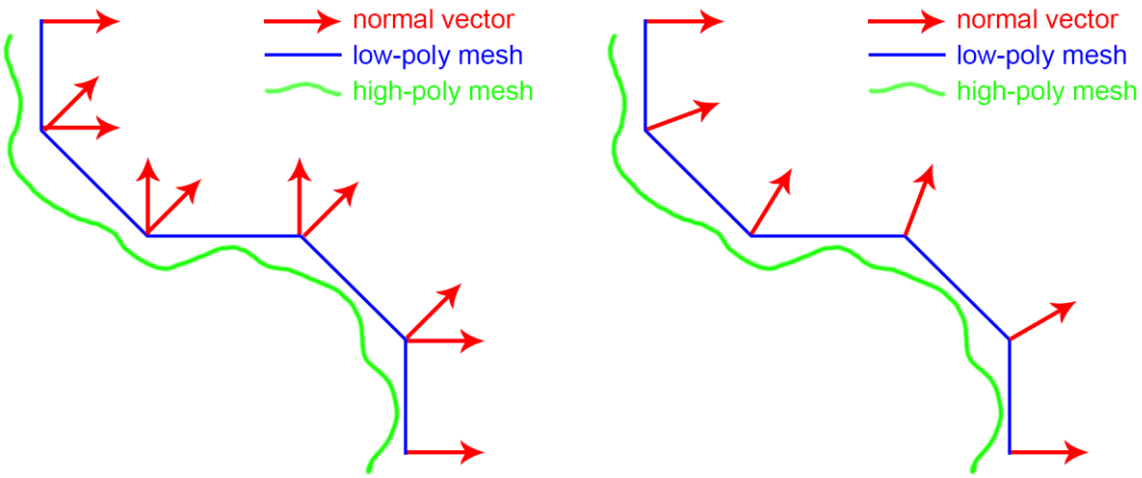
Je třeba si však uvědomit, že přidaný detail povrchu je falešný. Jde pouze o efekt zobrazení normály, který při velkém úhlu pozorování nefunguje správně a chceme-li detail pozorovat na siluetě modelu, vytrácí se zcela. Proto se tento typ mapy používá pouze pro miniaturní nerovnosti, například nerovnosti na zdech, u kterých nepotřebujeme, aby ovlivnili siluetu modelu.



Obr. 3.4 Příklady textur typu bump map [28]

Normal map

Normal map je novější a efektivnější typ textury nežli bump map, ale většinou je mnohem složitější ji vytvářet. Nejčastěji se model vytvoří ve dvou vyhotoveních. Detailním high-poly obsahujícím mnoho polygonů, které by však bylo příliš náročné na systém a zjednodušenou verzí low-poly. Ze složitějšího modelu se "upeče" normálová mapa, obsahující informace o normálách ve všech bodech modelu a ta se poté aplikuje na low-poly model. Místo stupně šedi normálová mapa používá barevnou RGB informaci, takže každá ze tří barev odpovídá jedné ose trojrozměrného prostoru.

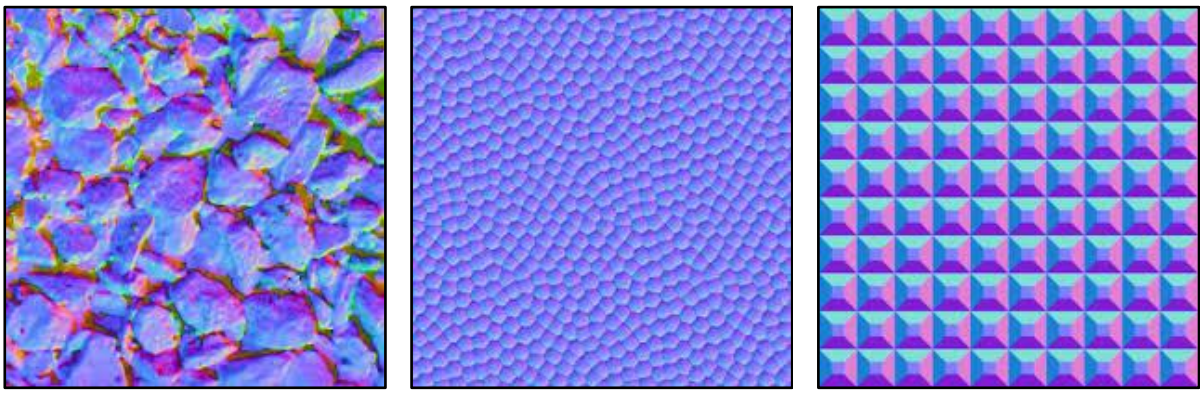


Obr. 3.5 Ilustrativní znázornění normal map textury [9]

Obrázek znázorňuje siluetu modelu. Zelená čára reprezentuje high-poly model. Modrá představuje low-poly. Na levém obrázku vidíme červenými čarami znázorněné normály modelu bez normálové mapy. V několika místech je pak vidět dvě normály zároveň, což

vyústí ve viditelný ostrý přechod mezi polygony, což je často nežádoucí. Použitím normál z high-poly modelu je tato informace přepsána a dojde k plynulému přechodu mezi polygony, které k sobě ve skutečnosti mohou svírat výrazný úhel. [9]

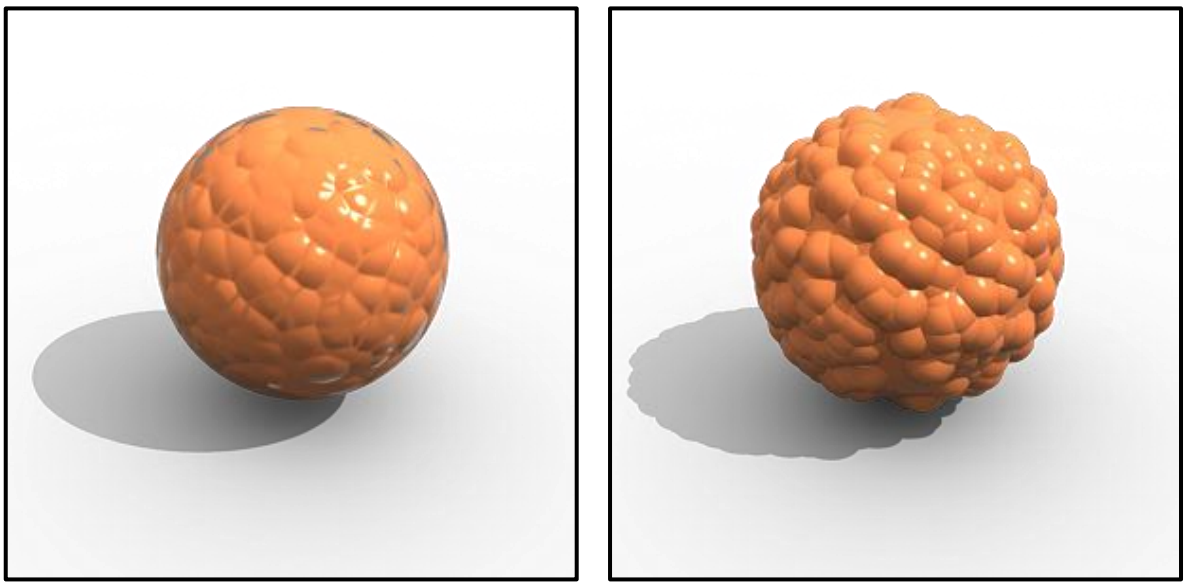
Stejně jako u bump map, je však docílení efektu větší složitosti modelu falešné, o čemž je možné se přesvědčit sledováním siluety modelu.



Obr. 3.6 Příklady textur typu normal map[29]

Discplacement map

Displacement map je jediný typ textur z této skupiny, u kterého se opravdu prostorově upravuje povrch modelu. Podobně jako u bump map jsou nerovnosti povrchu reprezentovány ve stupních šedi. Běžně se textury tohoto typu také "upečou" z detailnějších modelů. Často používané nástroje na přidávání detailů modelům jsou Autodesk Mudbox nebo Pixologic Zbrush. Pomocí této metody lze dosáhnout velmi věrných detailních výsledků. Bohužel je to však také metoda, která nejvíce zatěžuje systém, protože je u modelů vytvářena dodatečná geometrie. [8]



Obr. 3.7 Znázornění rozdílu mezi bump mapou vlevo a displacement mapou vpravo [30]

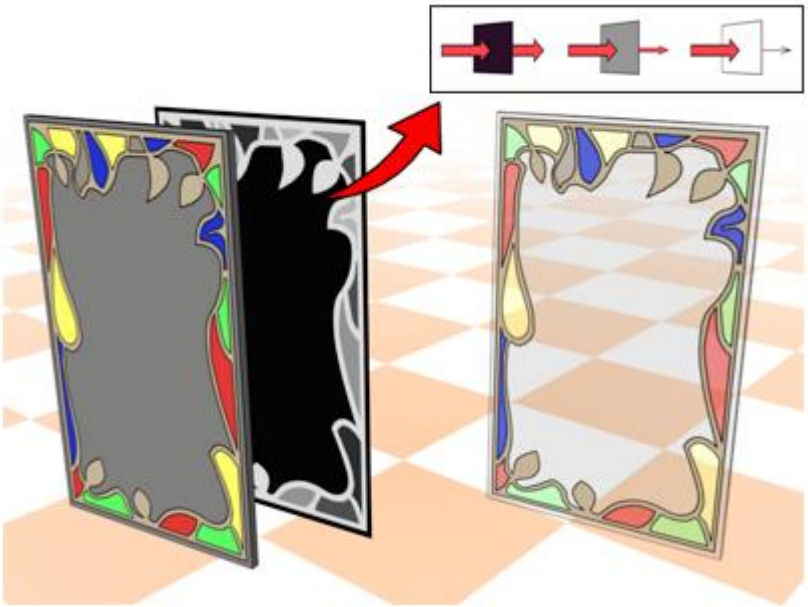
3.3 Specular map

Textury typu Specular map dodávají modelu jiskru. Pomocí tohoto typu textur můžeme dosáhnout odlesků na různých místech modelu. Model se často skládá z více různých materiálů. Kovové předměty odráží světlo více než dřevěné. Specular mapy jsou opět reprezentovány ve stupních šedi, kde tmavá barva představuje nízkou odrazovost světla a světlá naopak vysokou. [7]

Obr. 3.8 Ilustrace použití textury typu specular map [23]

3.4 Opacity map

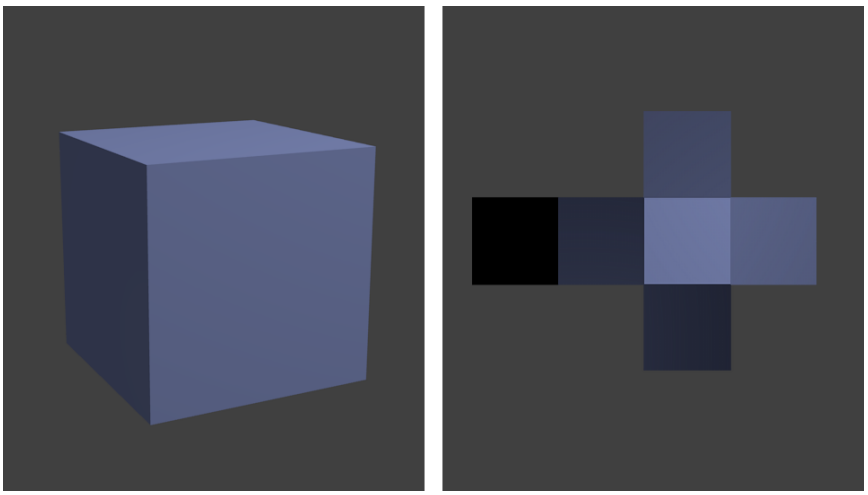
Textury Opacity map umožňují zachycovat průhlednost materiálu. Ať už částečnou například u oken tak i úplnou, kterou můžeme využít například u plotů. Tohoto efektu se dá také dosáhnout použitím průhlednosti v color mapě v podobě alpha kanálu.



Obr. 3.9 Příklad použití opacity map [2]

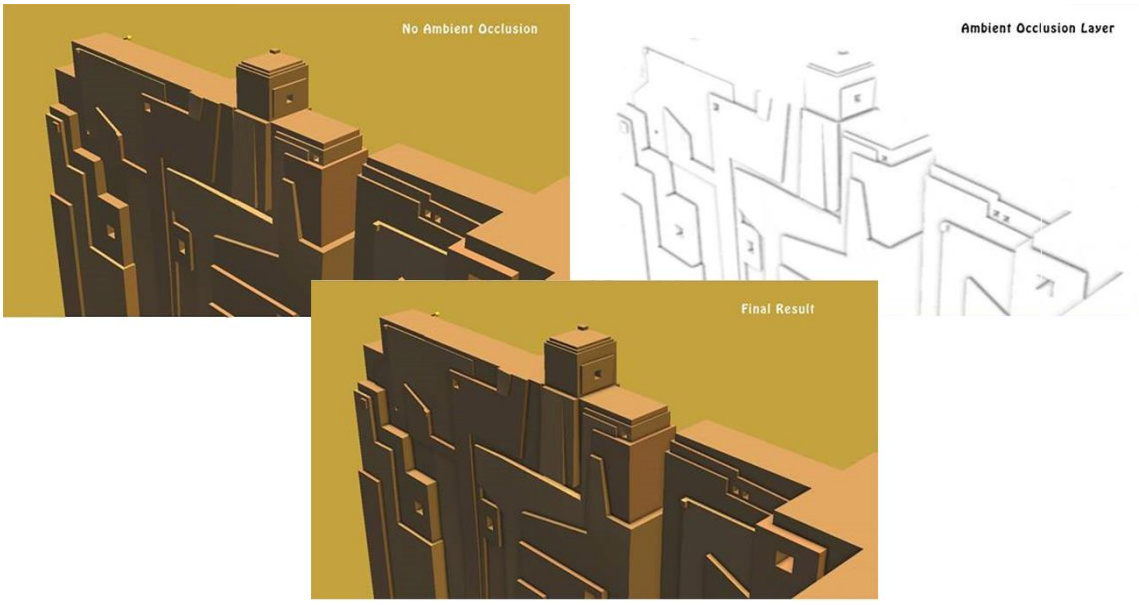
3.5 Light map

Light mapy udržují informaci o osvětlení objektu. Získává se předpočítáním množství světla, jaké na různé části objektu dopadá za daných světelných podmínek a při renderování je tato hodnota jen načtena z textury. Toho se využívá u statických objektů při renderování v reálném čase. Dynamické objekty se ve scéně pohybují a jejich nasvícení se mění, tudíž je potřeba jejich osvětlení počítat přímo při renderování.



Obr. 3.10 Ukázka light map textury [24]

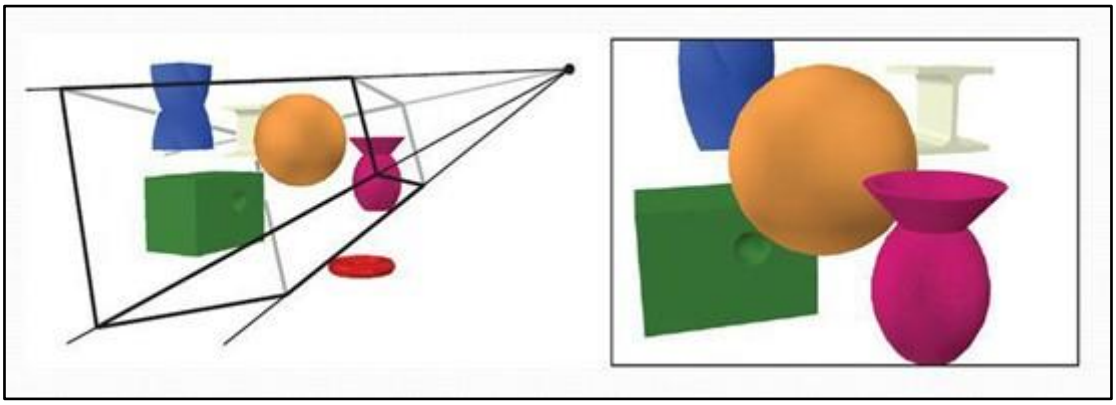
Často se používá light mapa s informací o ambient occlusion (zastínění okolím), která nezávisle na typu osvětlení, znázorňuje jak moc je dané místo vystavené či skryté v prostoru. Zakryté místo, například rohy místností, odráží méně ambientního světla než vystouplá místá objektu. Na obrázku 3.11 můžeme tento efekt pozorovat. [1]



Obr. 3.11 Ilustrace efektu ambient occlusion [25]

Kapitola 4
Renderování

Renderování je proces, ve kterém je vykreslován 2D obraz z připraveného 3D modelu. Prostorová scéna obsahující geometrii objektů, textury, světla a pozice kamery, ze které se má obrázek scény zachytit, jsou předány renderovacímu programu, který vypočítá konečný vzhled dvourozměrného rastrového obrazu.



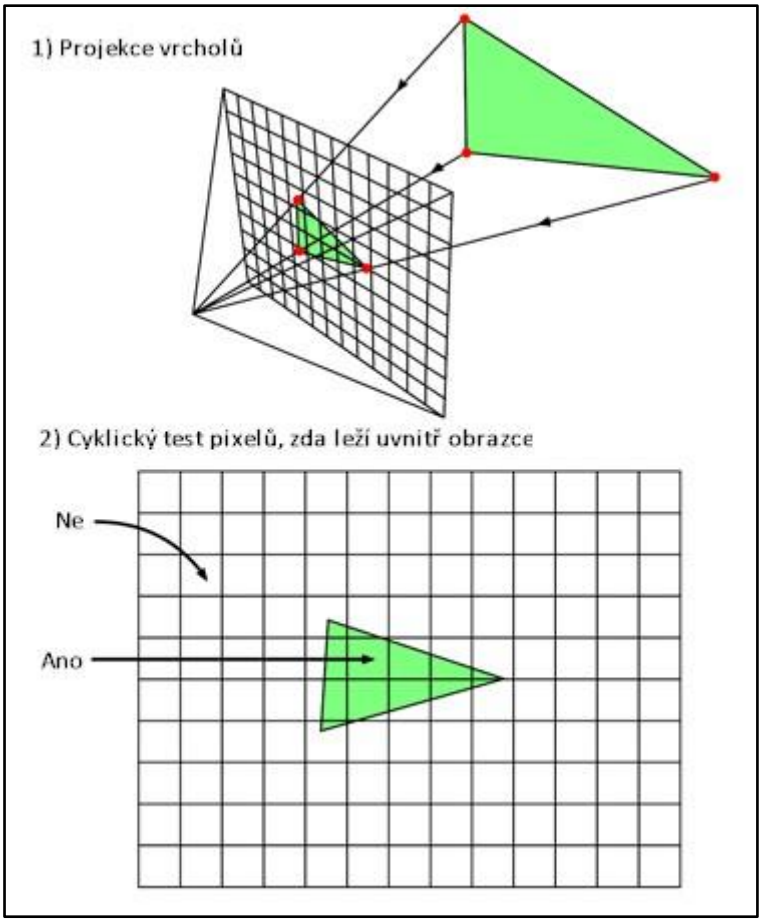
Obr. 4.1 Ilustrativní znázornění renderování [24]

Pro účely složitých renderovacích výpočtů, na které standardní procesor CPU (Central Processing Unit) nestačí, se používá specifický grafický procesor GPU (Graphics Processing Unit). Syntéza obrazu se snaží docílit co nejvíce realistického zobrazování, což se využívá zejména ve filmovém a herním průmyslu, architektuře, designu a tvorbě mnoha grafických programů.

K dispozici je několik metod, jak lze dojít ke konečnému obrazu, které se liší náročností výpočtu i ve výsledném zobrazení. Všechny se snaží vyřešit problém viditelnosti povrchu geometrickými výpočty, ovšem liší se pořadím, ve kterém se jednotlivé fáze procesu provádějí. Z pohledu viditelnosti povrchu proces renderování řeší způsoby, kterými lze určit, zda jsou dané části 3D scény viditelné z pozice kamery, zda jsou skryté za jiným předmětem, nebo se nachází mimo výseč kamery. Tato otázka se dá řešit dvěma hlavními metodami. Můžeme vést paprsek každým pixelem ve scéně, abychom zjistili, jakými objekty prochází (ray casting, ray tracing), což představuje obrazově založený přístup. Druhý, objektově založený přístup, naopak promítne polygon na obraz pomocí perspektivní projekce (rasterizace) [14].

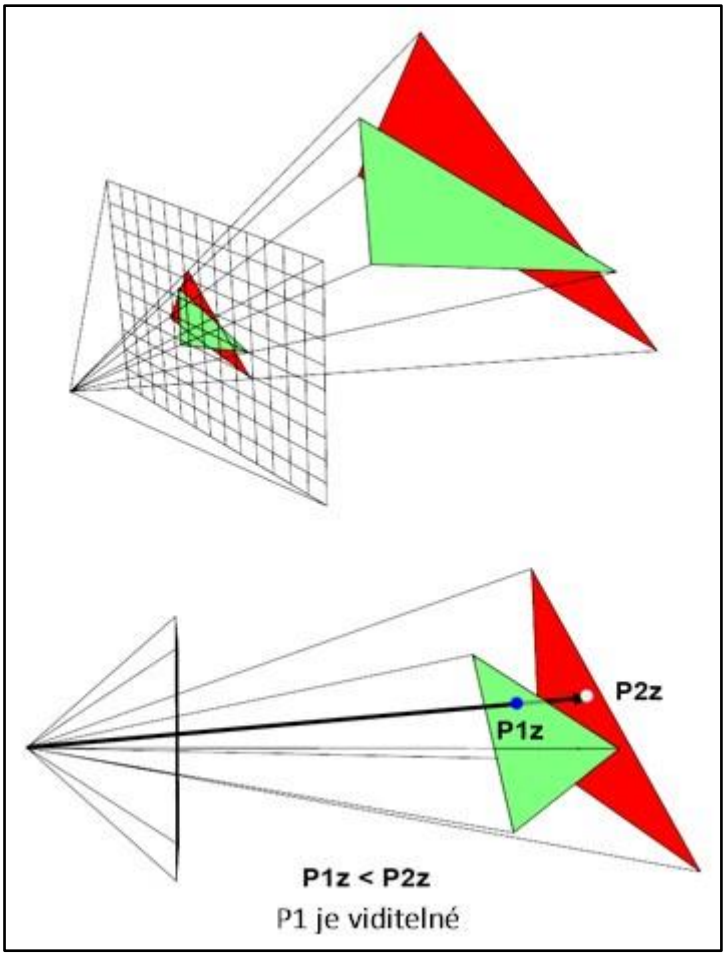
4.1 Rasterizace

Rasterizace je typ renderování, který používá objektově založený přístup. Tento proces pracuje s každým polygonem zvlášť a promítne jej na obraz, čili z 3D modelu vznikne jeho 2D zobrazení. Toho se docílí jednoduše promítnutím jeho vrcholů a následným určením, které pixely se nachází uvnitř již promítnutého 2D trojúhelníku.



Obr. 4.2 Ilustrace postupu při rasterizaci 1 [25]

Během procesu renderování je nutné někam ukládat renderovaný obraz. K tomu slouží tzv. frame buffer, který představuje dvoudimenzionální pole barev o velikosti stejné jako obrázek. Na začátku obsahuje samé pixely černé barvy. Jakmile dojde k rasterizaci polygonu, přepíše uloženou hodnotu odpovídajících pixelů. Jelikož se může několik polygonů promítnout na stejné místo, ale pouze jeden z nich je viditelný, musíme určit, který z nich je nejblíže. K tomu slouží tzv. depth buffer (nebo z-buffer), který je podobně jako frame buffer reprezentován dvoudimenzionálním polem o velikosti obrazu, ale místo barev pixelů ukládá vzdálenost počítaného polygonu od obrazu. Na začátku je pole zaplněno největšími možnými vzdálenostmi, podle použitého datového typu. Při určování viditelnosti polygonu proces vždy zkontroluje, zda vzdálenost aktuálního polygonu není delší než uložená hodnota, v takovém případě hodnotu barvy i vzdálenosti ponechá na původní hodnotě. Pokud je vzdálenost kratší, přepíše barvu pixelu i vzdálenost polygonu od obrazu.



Obr. 4.3 Ilustrace postupu při rasterizaci 2 [25]

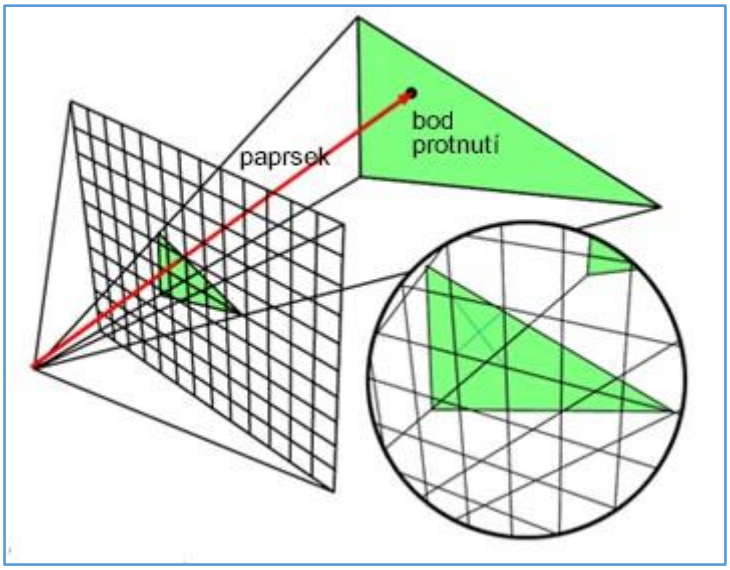
Rasterizace představuje objektově založený přístup, protože nejdříve prochází geometrii objektů a poté řeší, na jaké pixely v obrazu se zobrazí.

Oba tyto kroky jsou početně nenáročné a lineárně závislé na počtu polygonů ve scéně.

4.2 Ray tracing

U ray tracingu je již z názvu patrné, že tento způsob renderování pracuje se sledováním paprsku. Algoritmus vysílá paprsek každým pixelem obrazu a počítá vzdálenosti každého objektu, kterým projde. Objekt, který se na scénu nakonec zobrazí, je ten s nejmenší vzdáleností. Při renderování pomocí této metody dochází ke dvěma cyklům.

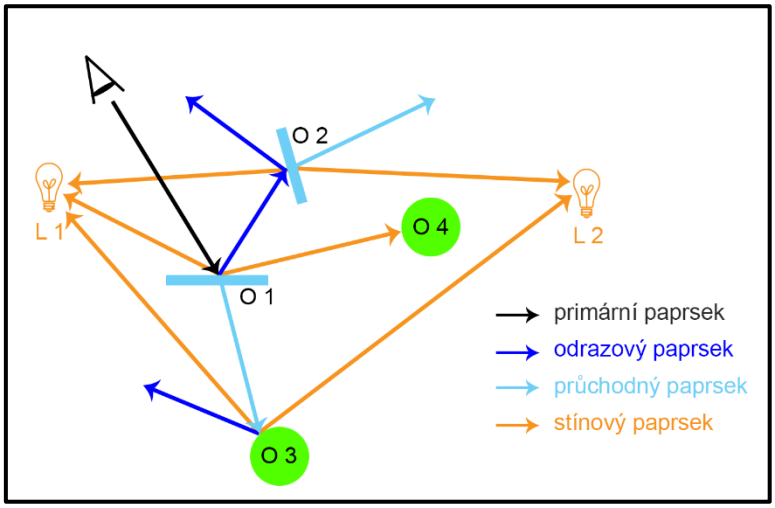
Renderovací algoritmus nejprve testuje po jednom všechny pixely, vede jimi paprsky a uvnitř tohoto cyklu prochází pro každý paprsek zvlášť všechny objekty ve scéně, aby zjistil, zda jimi paprsek prochází.



Obr. 4.4 Ilustrace postupu při raytracingu 1 [25]

Poté, co je určen nejblíže položený objekt v cestě paprsku (pokud vůbec na nějaký objekt vysílaný paprsek narazí), z místa dopadu je vedeno několik dalších nových paprsků, aby mohla být dopočítána výsledná barva, protože nezáleží pouze na základní barvě renderovaného objektu. Je třeba v této souvislosti určit, jestli se daná část objektu nenachází ve stínu, proto jsou z místa dopadu paprsku vedeny „stínové" paprsky směrem ke všem zdrojům světla ve scéně. Pokud stínový paprsek narazí na jiný objekt na své cestě ke zdroji světla, je zřejmé, že místo na prvním objektu je zastíněné dalším předmětem. Dále je veden z místa dopadu „odrazový" paprsek pod úhlem dopadu. Pokud prochází dalším objektem ve scéně, algoritmus upravuje výslednou barvu prvního objektu. Odrazový paprsek má smysl pro povrchy, které více odráží světlo a mohou zrcadlit povrch dalších objektů. Poslední paprsek, vysílaný z místa dopadu je „průchodný" paprsek, který prochází částečně průsvitnými objekty a pokračuje k dalším objektům, aby vrátil přídavnou informaci k výpočtu konečné barvy.

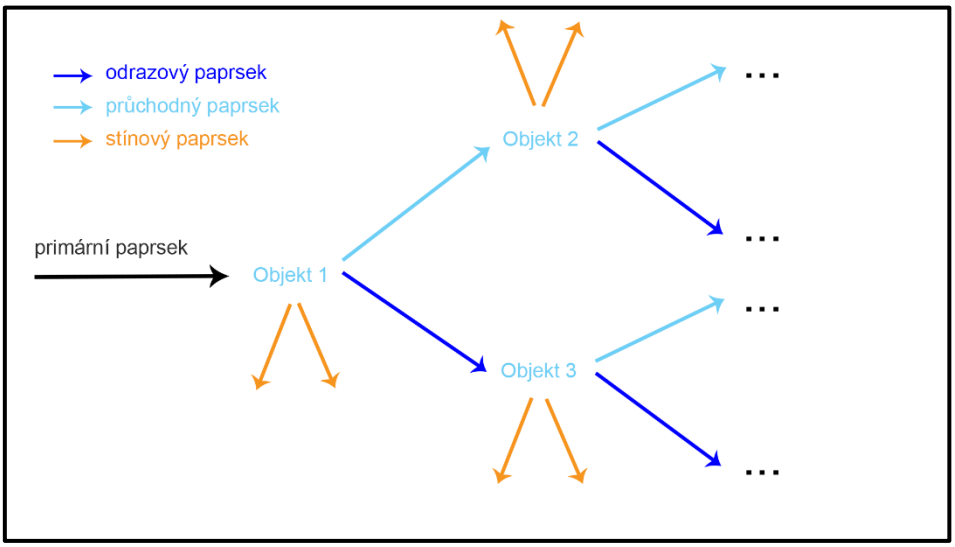
Toto lze popsat jako obrazově založený přístup, protože renderovací algoritmus nejdříve prochází pixely v obrázku a až poté jednotlivé objekty.



Obr. 4.5 Ilustrace postupu při raytracingu 2

Jak lze pozorovat na obrázku, původní paprsek nejprve naráží do objektu O1. Odtud vychází dva stínové paprsky směrem ke zdrojům světla L1 a L2. První dorazí k L1 bez překážky, ale druhý paprsek je zastaven objektem O4. Objekt O1 je tedy osvětlen pouze jedním světlem. Dále můžeme pozorovat odrazový paprsek, který na své cestě narazí na objekt O2, ve kterém dojde ke stejným operacím jako u objektu O1. A nakonec průchodný paprsek projde objektem O1 a dopadne na objekt O3, od kterého se opět odráží dál.

Jak je patrné, tento postup pokračuje stále dál, dokud, paprsek neopustí scénu bez srážky. Na tomto principu je možné vygenerovat paprskový strom, který je znázorněn na obrázku 4.6



Obr. 4.6 Paprskový strom pro renderování na principu ray tracingu

Výše zmíněný princip je při vykreslování obrazu renderovacím algoritmem postupně rekurzivně aplikován. Jelikož odrazové a průchodné paprsky závisí na odrazivosti a průhlednosti materiálu daného objektu, nemusí být u některých materiálů vygenerovány vůbec, nebo alespoň jejich intenzita postupně klesá. Toto znázorňuje následující vzorec.

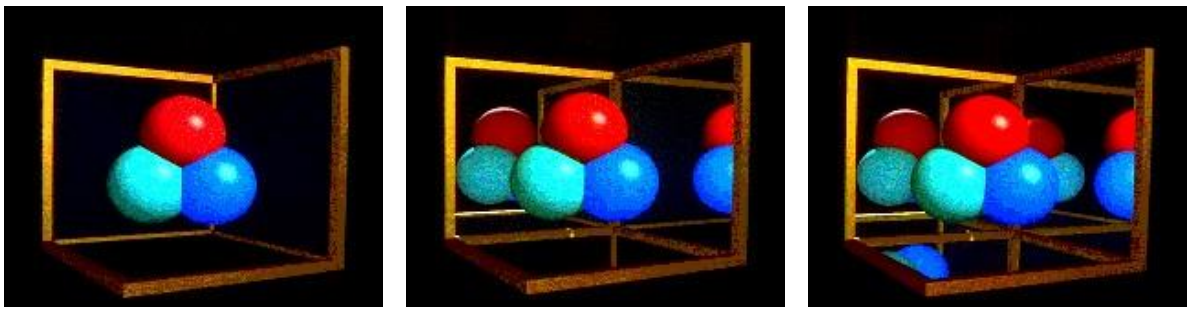
$$I = I_{local} + K_r * R + K_t * T$$

I_{local} – výpočet osvětlení lokálního objektu
R – intenzita světla odrazového paprsku
T – intenzita světla průchodného paprsku
K_r – odrazový koeficient
K_t – průchodný koeficient

Čím vyšší je stupeň rekurze při výpočtu (myšleno čím dál je sledovaný paprsek od původního paprsku), tím menší má objekt v cestě sledovaného paprsku vliv na výslednou barvu vykresleného pixelu v renderovaném obrazu.

Při renderování každého pixelu, ve scéně s vysoce odrazovými či průsvitnými materiály se mohou dramaticky zvýšit nároky na renderovací čas. Proto lze nastavit maximální hloubku paprskového stromu a omezit tak počet odrazů ve scéně. Na obrázku č 3.7 je vidět ilustrace porovnání třech případů. Levá část obrázku obsahuje vyrenderovaný obraz scény bez odrazových paprsků (hloubka stromu = 1), prostřední část obrázku ilustruje obraz scény

renderovaný s jednou vrstvou odrazů (hloubka stromu = 2) a v pravé části obrázku je vidět obraz scény, kde jsou objekty zrcadleny dvakrát (hloubka stromu = 3).



Obr. 4.7 Vliv nastavení stupně zrcadlení při ray tracingu [26]

Využití rasterizace a ray tracingu

V minulosti měli tyto dvě metody poměrně přesně vyhraněné využití. Rasterizace jako rychlejší, ale méně realistická metoda vykreslování obrazu prostorových scén se využívala v případech, kdy bylo nutné renderovat obraz v reálném čase, například v počítačových hrách nebo pohyblivých simulátorech. Ray tracing se naopak jako početně náročná metoda, která ale dosahuje lepších a realističtějších výsledků, využívala ve složitých scénách, ve kterých byl kladen důraz na realističnost výsledného zobrazení a obrazy mohly být vykreslovány předem. Dnes se ale už toto jednoznačné vymezování díky dostupnosti dostatečně výkonného hardwaru, na kterém renderování obrazu probíhá, stírá. V dnešní době je možné renderovat scény pomocí ray tracingu i v reálném čase a rasterizace se využívá stále méně. Rasterizace a její realističnost zobrazování je stále limitována tím, že vždy pracuje pouze s jedním objektem a nezohledňuje vliv dalších objektů ve scéně (stín, odraz, průhlednost) zatímco ray tracing umožňuje vyrenderovat pixel s kombinací odrazů nebo stínů více objektů.

Kapitola 5
Analýza možných postupů při vytváření modelu

5.1 VRUT

Zkratka VRUT představuje název Virtual Reality Universal Toolkit, který označuje programový nástroj pro zobrazování grafických dat nebo jiné operace, které lze nad grafickými daty provádět. Projekt, v rámci kterého byl VRUT vyvíjen, vznikl ve spolupráci katedry počítačové grafiky a interakce ČVUT FEL a společnosti Škoda Auto.

Podstatou VRUTu jako řešení je to, že dokáže spojovat různé úlohy pro práci nad grafickými daty, které jsou realizovány ve formě zásuvných modulů – pluginů a které mohou být do značné míry nezávislé na kmenové aplikaci napsané v jazyce ANSI C++. Tyto moduly se dají rozdělit do skupin:

- General modules – obsahující základní moduly potřebné k práci na scéně,
- Render modules – moduly umožňující renderování scény,
- Manipulatio modules – moduly umožňující různé způsoby nastavení interakce se scénou,
- IO modules – moduly na ukládání a načítání dat do scény.

Důležitým funkčním prvkem je graf scény, který reprezentuje strukturu načtené scény. Umožňuje vytvářet nové geometrie, světla, kamery, zdroje zvuku atd. a pracovat s nimi.

V rámci této bakalářské práce bude nástroj VRUT využit jako vizualizační aplikace pro testování 3D modelu města. Pro VRUT modul, který dokáže simulovat prostředí interiéru automobilu v roli simulátoru jízdy, je v praktické části této bakalářské práce vytvářen model města, který představuje venkovní scénu pro realistické vyjádření automobilového simulátoru. Vlastní model, jeho chování a faktická použitelnost v modulu simulátoru jako virtuálního vnějšího prostředí jsou pak testovány v simulátoru v kontextu možností, které jsou dostupné při vytváření modelu města.

5.2 Použitelné techniky modelování

Návrh rozsahu města a jeho půdorys bude vytvářen s ohledem na účel vytváření modelu jako venkovní scény pro simulátor auta.

Pomocí vhodného modelovacího nástroje bude vytvořen organický plán města s vhodným rozsahem ulic a s možností využívání cyklických tras v jízdním simulátoru. Vzniklé zastavěné bloky budou rozděleny na rozměrově přiměřené parcely, na které budou modelovány jednotlivé budovy virtuálního města. Rozsah modelu bude koncipován a nastaven na pokrytí přibližně 80-ti modely jednotlivých budov, které budou tvořit souvislou městskou zástavbu.

Pro modelování konkrétních trojrozměrných staveb lze použít některé modelovací techniky, popsané v kapitole 2. Městské stavby tvarově sice vycházejí z jednoduchých základních

geometrických tvarů, v detailech jsou ale komplikovanější, a proto metoda modelování Konstruktivní geometrie pevných těles CSG pro vytváření modelu městské zástavby není vhodná.

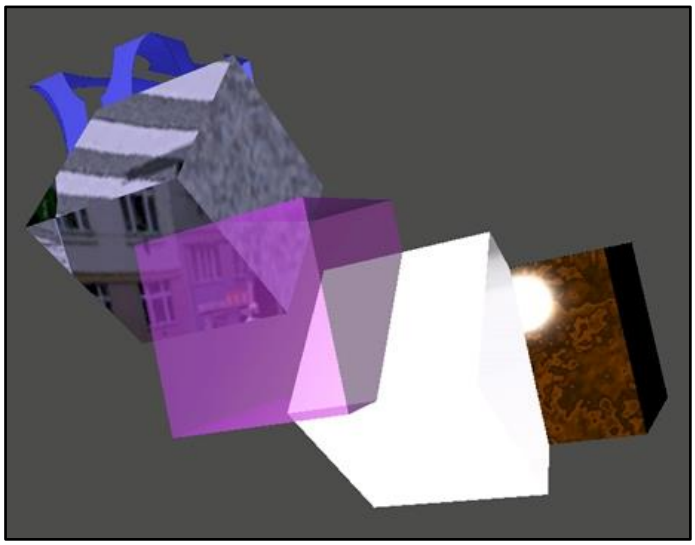
Modely části budov budou vytvářeny polygonálním modelováním, jehož princip je popsán v kapitole 2. Tento způsob je vhodný u tvarově složitějších, originálních a neopakovatelných budov, které obvykle přímo nevycházejí z půdorysu příslušné parcely. Procedurálně modelovat je vhodnější spíše tvarově podobné a jednodušší budovy, které vhodně doplňují bloky zástavby a více půdorysně kopírují tvar parcel. Princip tohoto způsobu modelování je nastíněn v kapitole 2.

Vzhledem k řádovému rozdílu v počtu polygonů v modelech budov vytvářených polygonálně nad procedurálními modely (což u polygonálních modelů také výrazně zvyšuje nároky na vykreslování jejich obrazů při zobrazování scén), bude vhodně volen poměr zhruba 10 polygonálních a 70 procedurálních modelů, které vytvoří celkový model zástavby města.

5.3 Použitelné typy textur pro vytváření povrchů

Na polygonálně a procedurálně vytvořené tvary modelů jednotlivých staveb a pozemní nezastavěné části modelu města budou namapované textury, které vytvoří obraz jejich povrchů, složených z různých materiálů a majících různé v obrazu zachytitelné vlastnosti, jako například barvu, nerovnost (plastičnost) a určitou průhlednost. Vhodná volba textur významně zvyšuje realističnost celého modelu a jednotlivých vykreslovaných scén. Jednotlivé typy textur jsou blíže popsány v kapitole 3.

Na obraze testovací scény test.fhs (obrázek č. 5.1), která obsahuje nastavení pro správné zobrazení podporovaných typů textur, je vidět jak a které typy textur vizualizační nástroj VRUT podporuje. Testovací scéna obsahuje pět krychlí, z nichž každá používá jiný typ textur. Můžeme vidět děrovanou krychli používající texturu s alfa kanály, krychli s reflection mapou, částečně průhlednou krychli a krychli pokrytou bump mapou.



Obr. 5.1 Obraz testovací scény test.fhs ve VRUTu

Vizualizační nástroj VRUT sám o sobě nepodporuje typ textury opacity map. Pro realistické zobrazení průhledných nebo částečně průhledných povrchů lze použít textury v grafických formátech, které podporují alfa kanály.

Alfa kanál je složka pixelu, která zachycuje hodnotu průhlednosti tohoto pixelu. Vhodným příkladem je barevný model RGBA, který kromě barevných složek RGB obsahuje i složku A vyjadřující průhlednost. Průhlednost pixelu znamená, že překrývá-li bitmapový obrázek s definovanou průhledností jiný obrázek, pak bude původní obrázek na pozadí zobrazen v příslušném bodě pixelu s intenzitou, která je dána mírou průhlednosti pixelu obrázku v popředí. Bitově může být alfa kanál vyjádřen 1 bitem (pixel je buď 100% průhledný nebo neprůhledný, což používá třeba grafický formát GIF) nebo 8 bity (což umožňuje využívat až 256 úrovní průhlednosti, s čímž pracuje například grafický formát PNG).

Jelikož VRUT zatím nepodporuje více sad texturovacích souřadnic namapovaných na jeden objekt, není možné použít opakovatelnou texturu v kombinaci s texturami namapovanými na objekt přímo, jako je například dirt mapa či light mapa. Proto budou textury vyhotoveny ve více provedeních.

1. Pro nynější použití bude opakovatelná textura vypečena spolu s dirt mapou a light mapou do jedné textury. Model tak příjde o kvalitu opakovatelné textury, ale zajistíme funkčnost v nástroji VRUT.

2. Pro budoucí použití s možností podpory více sad texturovacích souřadnic budou vyexportovány zvlášť opakovatelné textury, dirt mapy a light mapy. V nástroji bude prozatím možno použít jen jeden z nich.

Nakonec budou navíc vyexportovány light mapy textur ve dvou rozdílných denních dobách s různými světelnými podmínkami.

5.4 Použitelné způsoby renderování obrazu

VRUT má implementované renderovací moduly RenderGL, který pracuje na principu rasterizace a Raytracer, kterým podporuje renderovací metodu ray tracing. Obě tyto renderovací metody, které zajišťují vykreslování obrazu scén vizualizovaných 3D modelů, jsou blíže popsány v kapitole 4.

Pro vlastní fungování modulu simulátoru je tady nutné zvolit vhodnou renderovací metodu pro vykreslování obrazů v konkrétních místech jízdy uvnitř modelu města, navíc v reálném čase tak, aby se obraz plynule vykresloval a měnil podle situace a průběhu simulace jízdy automobilu virtuálním městem.

Pro účely zkušebního testování jsem v aplikací VRUT mohl vyzkoušet i dva vzorové 3D modely testovacích scén pro jízdní simulátor:

- bal2_skoda_navigace.fhs (307 uzlů, 217 objektů, 124 materiálů),
- real_trat.fhs (723 uzlů, 583 objektů, 201 materiálů).

Oba modely obsahují ve výsledku menší počet objektů a zejména výrazně jednodušší objekty vytvořené z řádově menšího množství polygonů.

Už při zkoušení renderovacích modulů VRUTu na těchto testovacích modelech a na běžně dostupném HW (notebook Lenovo Y50-70, CPU Intel Core i5-4210 2,90GHz, 8GB RAM, video Intel HD Graphic 4600, OS Windows 10 Home 64-bit) se ukazovalo, že na tomto hardwaru není vhodné použít pro účely vykreslování okolní scény pro jízdní simulátor v reálném čase renderovací modul Raytracer. Vykreslování obrazů neprobíhá plynule a v obraze se objevují černá místa. Výsledný obraz modelu je při tomto typu renderování velmi realistický, ale pro využití v modulu jízdního simulátoru na běžném hardwaru nepoužitelný. Plynulé renderování složitých modelů v reálném čase pomocí renderovacího modulu Raytracer by v konečném důsledku vyžadovalo úpravu především vlastního renderovacího modulu a výkonově velmi robustní hardware, který pro účely této práce není dostupný.

Renderování obrazu pro jízdní simulátor při využití modulu RenderGL nabízí sice méně realistický výstup, samotná simulace v obrazech vnějšího okolí neobsahuje stíny, více-stupňovou reflektivitu textur či částečně průhledné textury, ale vykreslování je plynulé a dostatečně rychlé i ze složitějších modelů, obsahujících velké množství polygonů, a použitelné i na dostupném hardwaru.

V rámci této bakalářské práce předpokládám, že pro účely použití v jízdním simulátoru budou obrazy vnějšího okolí z předmětného 3D modelu v reálném čase renderovány s využitím modulu RenderGL, který by měl nabízet uspokojivé výsledky.

Pro vykreslení statických ilustrativních obrázků vnějších scén použiji renderovací modul Raytracer s následujícími úpravami předmětného 3D modelu, abychom snížili nároky na dostupný výpočetní výkon. Pro každý snímek scény budou při renderingu vyhodnocovány jen objekty přímo viditelné z místa kamery čili renderování bude pracovat vždy jen s příslušnou omezenou částí modelu. Pro zachycení scény v místech, ze kterých lze dohlédnout na vzdálenější budovy, budou jako vhodnější použity jejich modelované low-poly verze. Některé low-poly modely jsou vytvořeny manuálně, ale u některých je možné použít optimalizační modifikátory ke snížení počtu polygonů.

Kapitola 6
Použitý software

6.1 Autodesk 3ds Max

Autodesk 3ds Max je profesionální softwarový nástroj pro 3D grafiku, animace a vizualizace od společnosti Autodesk. Nejčastěji se používá v postprodukci, při výrobě reklamních videospotů, filmů a v televizním průmyslu. Je velmi vhodný pro vytváření architektonických vizualizací a často je používán i k tvorbě grafiky do počítačových her.

V praktické části této bakalářské práce jsem tento nástroj použil pro polygonální modelování objektů, blíže popsané v kapitole 2. Dále je 3ds Max vhodný k renderovaní scén, k simulacím objektů ovlivněných okolními vlivy, animacím osob a k dalším podobným účelům. Nabízí propracované interaktivní grafické uživatelské prostředí a rychlý rendering. Aplikace obsahuje zabudovaný skriptovací jazyk MAXScript používaný například k zautomatizování opakujících se úkonů či vývoji nových rozhraní a pluginů. Pro 3ds Max je k dispozici bohatá knihovna doplňkových plug-in modulů dalších vývojářů.

3ds Max podporuje mnoho datových a souborových formátů. Standardně jsou modely a scény ukládány v souboru s příponou MAX, obsahující mnoho souvisejících informací (např. historie). Pro exportování vybraných informací nejčastěji používá soubory s formátem 3DS, FBX nebo OBJ. [3]

Použitá verze programu: **Adobe 3ds Max 2016**

6.2 Esri CityEngine

Esri CityEngine je 3D modelovací programový nástroj, který je určen pro modelování rozsáhlých urbanistických modelů od společnosti Esri. Tento systém byl poprvé představen v roce 2008 a podporuje několik způsobů modelování.

Procedurální modelování reprezentované CGA gramatikou umožňuje pomocí souborů pravidel generování mnoha různých budov na základě společných pravidel. Konkrétněji jsou principy procedurálního modelování popsáno v kapitole 2.2.

CityEngine umožnuje použít i polygonální způsoby modelování a lze v něm polygonálně vytvářet objekty podobně jako 3ds Max, ovšem v omezené míře a značně pomaleji. Tento přístup se však může vhodně kombinovat se soubory pravidel, což 3ds Max neumožňuje.

Podobně jako 3ds Max i CityEngine obsahuje skriptovací jazyk Python. Skripty zde mohou být kromě vytváření procedur uživatelských úkonů v interaktivním uživatelském rozhraní použity k vytvoření zformátovaných informačních reportů.[4]

Použitá verze programu: **Esri CityEngine 2014.0**

6.3 Adobe Photoshop

Adobe Photoshop je grafická aplikace a editor využitelný pro vytváření a úpravu bitmapových obrázků a grafiky od společnosti Adobe. Nástroj používá systém vrstev, který umožňuje nedesktruktivní upravování obrázků. Dále podporuje masky, kanály a nabízí mnoho dalších užitečných funkcionalit.

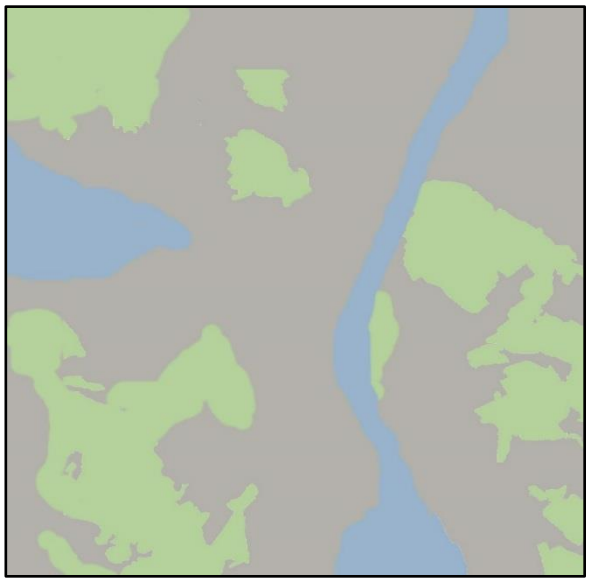
Dokumenty jsou ukládány s příponou PSD či PSB, pro velmi rozsáhlé grafické práce. Pro tento software se neustále vyvíjí velké množství pluginů.

Adobe Photoshop jsem v praktické části této bakalářské práce využil pro vytváření a modifikace textur, které jsem mapoval na připravené modely, pro zpracování některých grafických detailů a při tvorbě a úpravách texturových map pro návrh scény celkového modelu. [6]

Použitá verze programu: **Adobe Photoshop CC 2015.0.0**

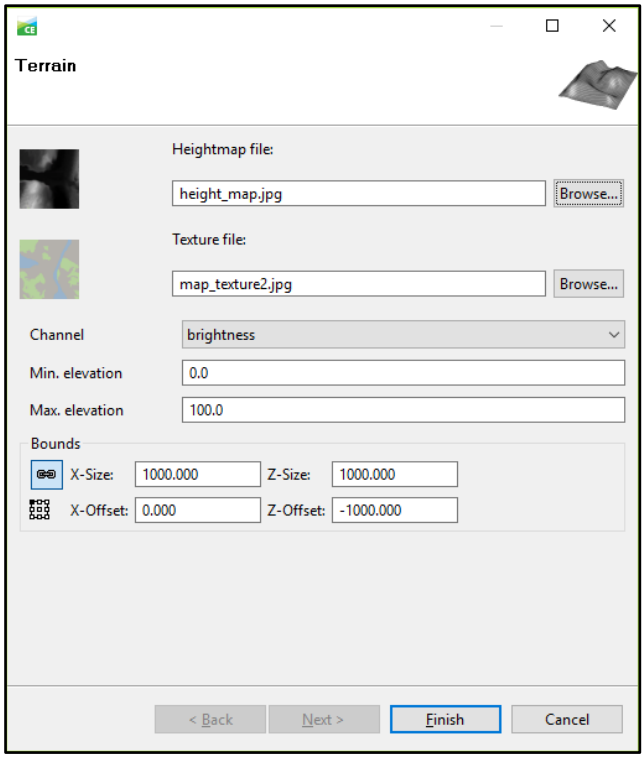
Kapitola 7
Popis realizace návrhu scény v CityEngine

Rozsah města byl nastaven tak, aby obsahoval kolem 120 parcel pro jednotlivé budovy. Nejprve byla nakreslena orientační mapa scény. Při jejím kreslení v aplikaci Adobe Photoshop jsem vycházel z ukázkové textury, kterou jsem ve Photoshopu upravil tak, aby přibližně odpovídala požadovanému rozsahu scény. Původní záměr vytvořit město na členitějším terénu jsem nakonec nerealizoval vzhledem ke zvýšené komplexnosti procedurálních pravidel ke generování budov.



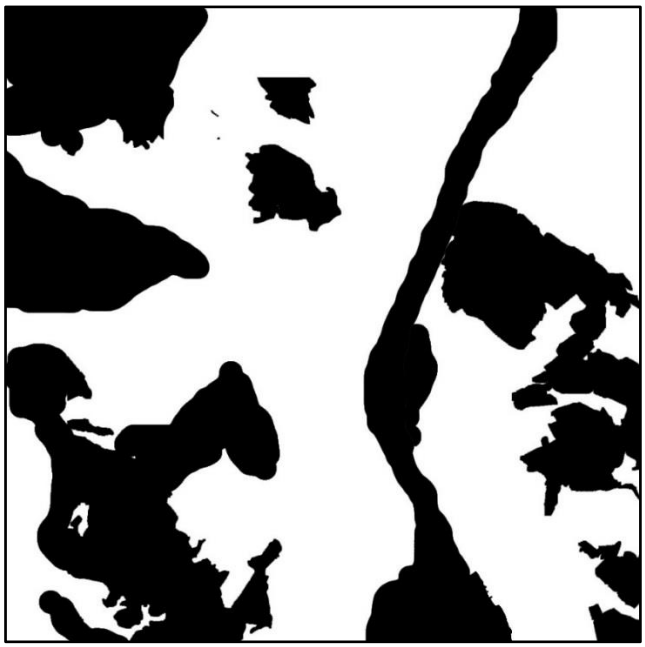
Obr. 7.1 Orientační mapa scény

Terén byl vytvořen pomocí aplikace CityEngine.



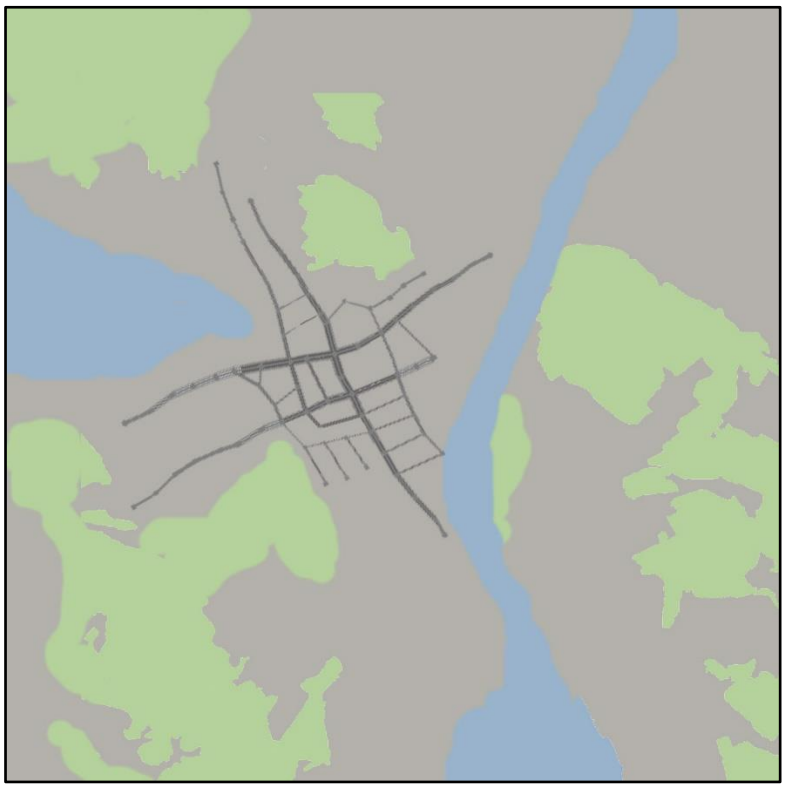
Obr. 7.2 Dialog pro základní nastavení terénu v aplikaci CityEngine

Silnice byly vygenerovány v aplikaci CityEngine pomocí funkce Grow Street, která náhodně vytváří síť ulic bez ohledu na texturu či výškovou mapu. Při výchozím nastavení se silnice vytvářely v nežádoucích místech, kde byla například plánována voda či velké převýšení. K docílení určité míry kontroly nad umístěním parcel byla použita tzv. Obstacle mapa, což je obrázek obsahující pouze černé a bílé pixely, které definují, zda se v daném místě může generovat zástavba či ne. Tuto mapu jsem vytvořil opět v aplikaci Adobe Photoshop.



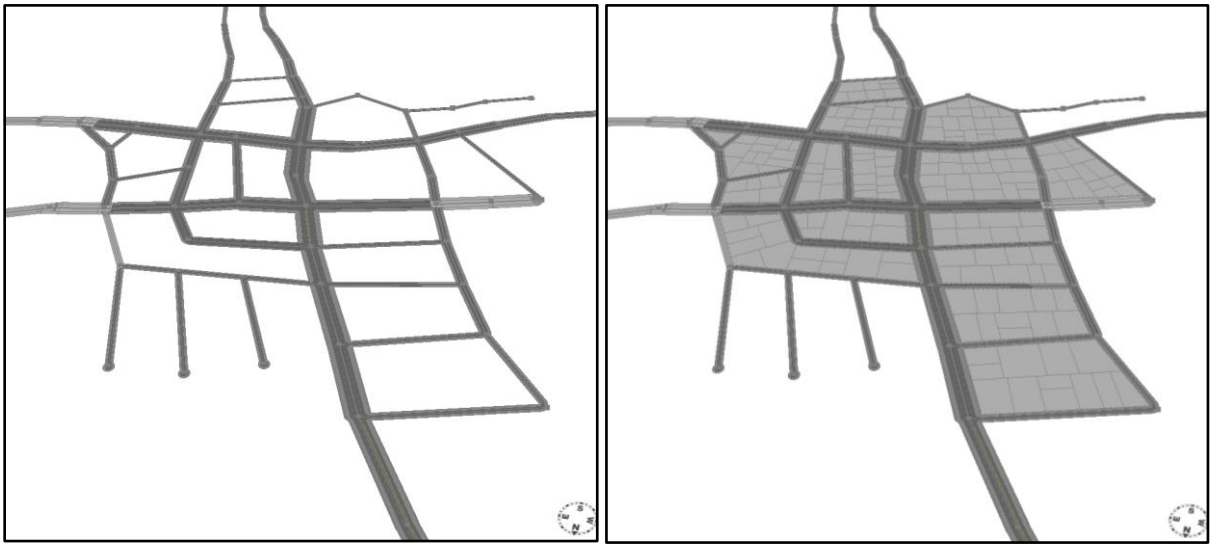
Obr. 7.3 Obstacle mapa

Vygenerovaná síť byla manuálně upravena a dotvořena pomocí kreslících nástrojů a funkce Cleanup Tool do požadované podoby.



Obr. 7.4 Mapa se sítí ulic

Bloky uzavřené v síťi ulic byly automaticky rozděleny na parcely a připraveny k modelování jednotlivých budov, které vytvoří virtuální městskou zástavbu. [18]

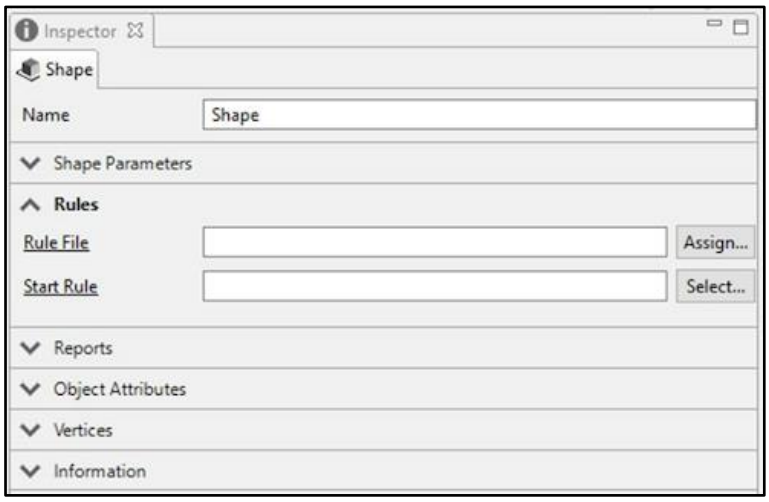


Obr. 7.5 Vygenerované bloky městské zástavby

Kapitola 8
Popis procedurálního modelování v CityEngine

Jak bylo již dříve zmíněno, většina typových budov městské zástavby modelu byla vytvořena metodou procedurálního modelování v programové aplikaci CityEngine. Tento způsob představuje sestavení příslušné gramatiky – souboru s pravidly (Rule File), podle kterého bude modelovaná budova v aplikaci generována.

Modelování každé budovy v CityEngine začíná výběrem parcely na mapě, na které má budova stát a přiřazením vhodného souboru, který již obsahuje nebo teprve bude obsahovat pravidla. Tato gramatika je pak spouštěna při generování budovy.



Obr. 8.1 Dialog pro spárování parcely a příslušné gramatiky v CityEnginu

Při sestavování souboru pravidel jsem začal definováním rámcových atributů budovy jako je výška celé budovy nebo výška patra.

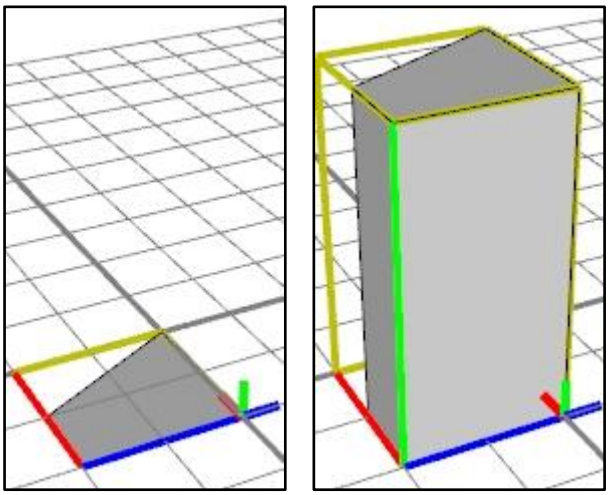
```
attr height = 30
attr floor_height = 4
```

Potom je třeba nadefinovat startovací pravidlo. Každá gramatika musí obsahovat alespoň jedno toto startovací pravidlo (Start Rule). Chceme-li zaplnit více parcel budovami vygenerovanými jedním souborem, ale potřebujeme jim pro generování v gramatice předepsat jiné vlastnosti, je vhodné nadefinovat více startovacích pravidel pro jednotlivé typy budov.

Startovací pravidlo je zpravidla pojmenováno řetězcem, který začíná klíčovým slovem "Lot". Například Lot, Lot1, Lot2...

Každé pravidlo běžně provede příslušnou operaci a poté pokračuje voláním dalšího pravidla. Na příkladu níže je znázorněno použití operace extrude, která z počátečního 2D tvaru parcely vysune 3D objekt s výškou 'height' a výsledný objekt předá pravidlu Building.

```
Lot -->
	extrude(height)
	Building
```



Obr. 8.2 Vysunutí tvaru budovy nad parcelou [27]

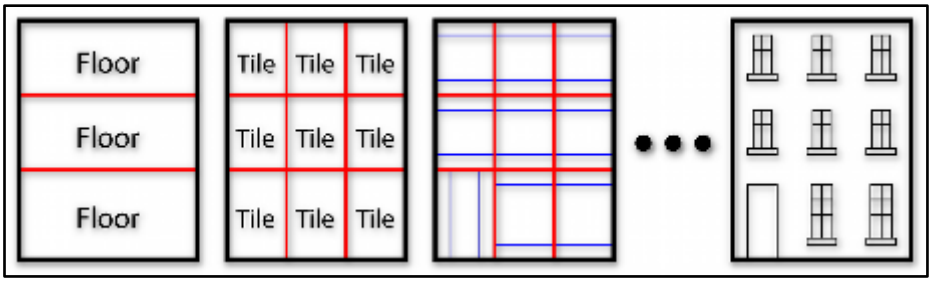
Pro práci s jednotlivými stěnami budovy, jsem 3D objekt rozdělil pomocí funkce comp(f), která umožňuje oddělit stěny směřující do ulice a stěny směřující dovnitř bloku a střechu.

```
Building-->
	comp(f){
		top : Roof
		|street.front : FrontFacade
		|all : Facade
	}
```

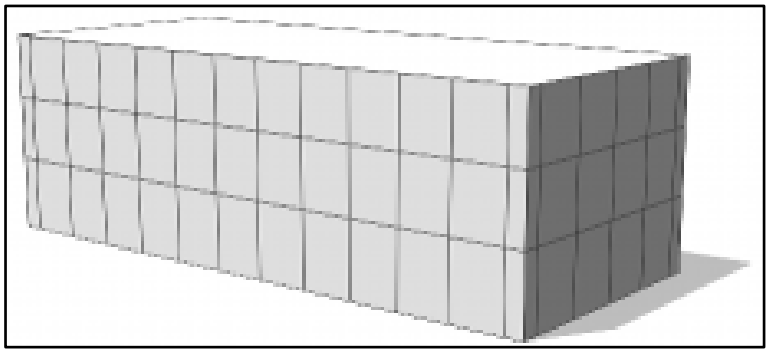
Pro oddělení jednotlivých podlaží v rámci jedné stěny byla použita funkce split, která s parametrem 'y' rozdělí stěnu horizontálně na libovolný počet částí. Stejná funkce s parametrem 'x' byla použita v případech, když bylo potřeba stěnu rozdělit vertikálně.

```
	split(y){
		groundfloor_height : GroundFloor
		|{~floor_height : Floor}*
	}
```

Na příkladu části kódu gramatiky můžeme pozorovat výškové rozdělení stěny na přízemí (GroundFloor) o výšce rovné parametru "groundfloor_height" a zbytku, který je rozdělen na ostatní patra (Floor). Hvězdička na konci představuje dělení na tolik pater s výškou 'floor_height', kolik se jich na výšku stěny vejde. Vlnovka před parametrem 'floor_height' přizpůsobuje výšku patra tak, aby se všechna patra vešla bez zbytkové části do výšky budovy.

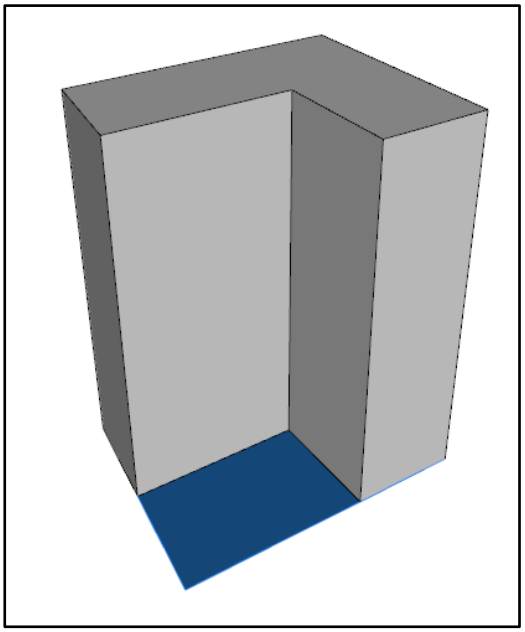


Obr. 8.3 Ilustrace horizontálního a vertikálního dělení stěny budovy [27]



Obr. 8.4 Ilustrace horizontálního a vertikálního dělení na 3D tvaru budovy [27]

CityEngine umožňuje také změnu tvaru půdorysu budov. Aby všechny neměly jen tvar klasických hranolů (většinou kvádrů), lze využívat při sestavování gramatik například funkce ShapeL nebo ShapeU, které jsou schopné upravit tvar budovy na parcele do tvaru L nebo U.



Obr. 8.5 Ilustrace výsledku použití funkce ShapeL [27]

```
shapeL(10, 15) {shape : Building | remainder: Walkway}
shapeU(10, 15, 15) {shape : Building | remainder: Walkway}
```

Ukázkový kód rozdělí parcelu na dvě části – Building, na které se dále postupuje v modelování stavby a Walkway, která může představovat například chodník, případně dvůr domu.

Jednotlivé funkce se mohou v CityEnginu volat s danou pravděpodobností. To umožňuje přiřadit soubor pravidel na celý blok a generovat odlišné budovy.

Stavby lze případně také variovat generováním náhodné velikosti atributů v daném rozsahu.

```
extrude(rand (10, 40))
```

Tato instrukce v gramatice vygeneruje budovu s náhodnou výškou mezi 10 a 40 m. [4]

Nejefektivnější způsob, jak jednotlivé budovy odlišit, je použití různých povrchových textur. Způsob mapování textur u procedurálně modelovaných objektů je nastíněn v kapitole 3.

Procedurální modelování se ukázalo také jako vhodný způsob, jak ve scéně rozmístit lampy, značky, zeleň, a další mobiliář. Tyto objekty byly poté exportovány odděleně pro lepší přehlednost v následné práci. [13]

Silnice a chodníky samotné byly vygenerovány pomocí ukázkového skriptu, přiloženého k CityEngine programu.

Výsledkem práce metodou procedurálního modelování v CityEnginu může být scéna obsahující rozmanité budovy, které se mohou lišit tvarem, výškou, texturami, stylem oken, balkóny a mnoha dalšími detaily. Spodní patra mohou obsahovat vchody do budov reklamních výlohy a cedule.

Pomocí operátoru i, je vhodné model doplňovat prvky vymodelovanými polygonálním způsobem.

```
i("assets/lamp")
```

Pro tento model byly tímto způsobem využity kromě vlastních modelu i 3 modely stromů z veřejné knihovny CityEnginu. (ESRi library > assets> vegetation).

Obr. 8.6 Ilustrace finální podoby procedurálně modelované budovy

Takto vytvořené modely byly exportovány do formátu FBX a dále upraveny v aplikaci Autodesk 3ds Max.

Formát FBX byl zvolen vzhledem k podpoře materiálů s více kanály.

Obsah vzorové CGA gramatiky je uveden v příloze A

Celá část modelu města, která byla vytvářena metodou procedurálního modelování v programu CityEnginu, obsahuje 1 481 objektů včetně ulic a chodníků, je složen z 255 466 polygonů a používá 496 materiálů.

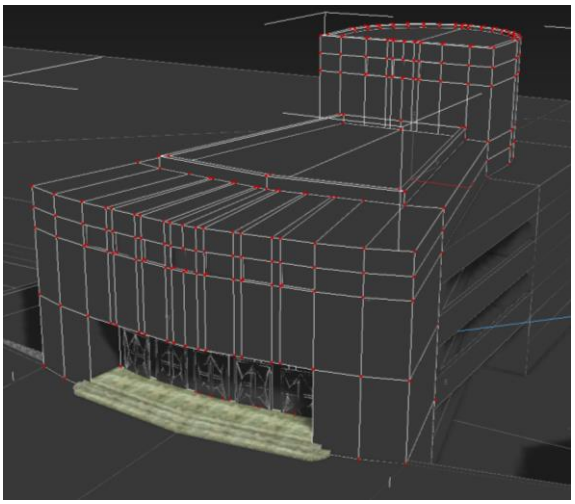
Kapitola 9
Popis polygonálního modelování v 3ds Max

Polygonální modelování bylo použito pro vytváření dominantních budov s neopakovatelnými tvarovými prvky ve scéně, pro které není použití procedurálního modelování efektivní a celkovou úpravu modelu, jako je například doplnění modelu o prvky typu pouličního osvětlení, zastávka hromadné dopravy, zábradlí, semafory nebo dopravní značky. Veškeré polygonální modelářské práce byly prováděny ve vhodné aplikaci Autodesk 3ds Max.

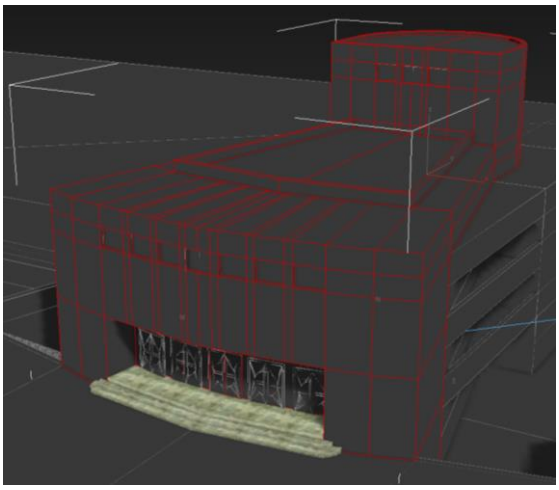
Přesný popis postupu při polygonálním modelování vzorové budovy by do značné míry přesahoval předepsaný rozsah náplně této práce, protože znamená popis velkého počtu opakujících se kroků, které pracují s jednotlivými dílčími prvky modelu budovy. Proto dále v této kapitole popisuji ty nejdůležitější a typové techniky, které jsem při modelování polygonálních modelů v programové aplikaci 3ds Max použil.

Edit poly

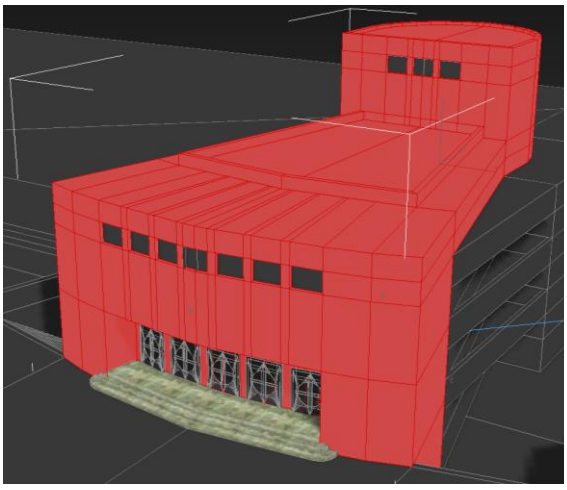
Technika Edit Poly (Select > Modify > Object-Space Modifiers > Edit Poly) je nejpoužívanější modifikátor ve scéně, který umožňuje explicitní úpravy různých sub-objektových úrovní (vrcholů, hran, polygonů). Po zvolení typu úrovně můžeme operovat pouze s vybranými částmi objektu - viz ilustrační obrázky č. 9.1 až 9.3. Na prvním operujeme s vrcholy (vertexy), na druhém s hranami a na třetím s polygony.



Obr. 9.1 Ilustrace techniky Edit Poly nad vrcholy

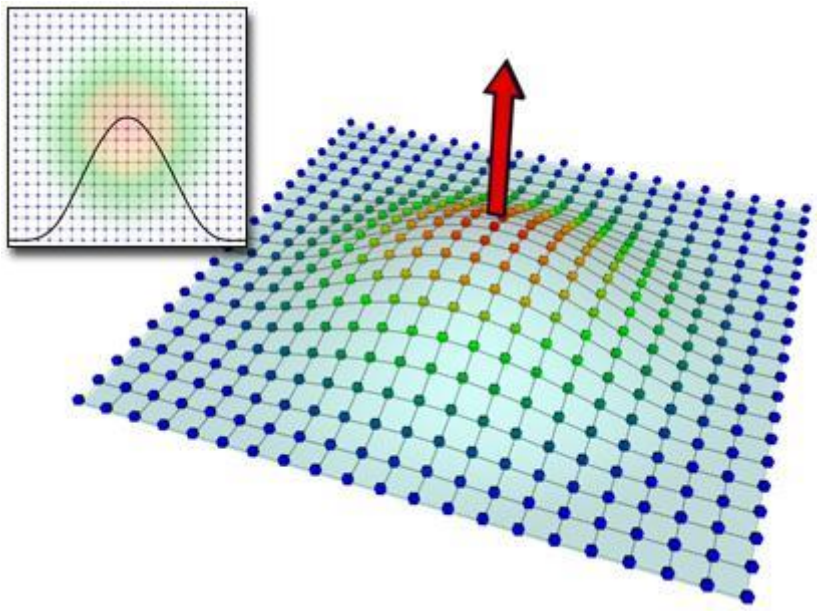


Obr. 9.2 Technika Edit Poly nad hranami



Obr. 9.3 Technika Edit Poly nad polygony

Ve spojení s technikou Edit Poly lze využít mnoho dalších nástrojů, např. **Měkký výběr (Soft Selection)**, který umožňuje částečně ovládat dílčí části objektů v závislosti na vzdálenosti od vybraného vrcholu (hrany, polygonu). [10] Takto jsem docílil hladšího přechodu mezi polygony. Tuto techniku jsem použil převážně při úpravě tvaru terénu. K přidávání dalších vektorů a hran byly použity nástroje **Connect** a **Cut**. Na vystouplé nebo vnořené prvky objektu jsem využil vhodné nástroje **Extrude**, **Hinge from Edge** a **Bridge**. Všechny zabudované a využitelné ve spojení s modifikátorem Edit Poly.

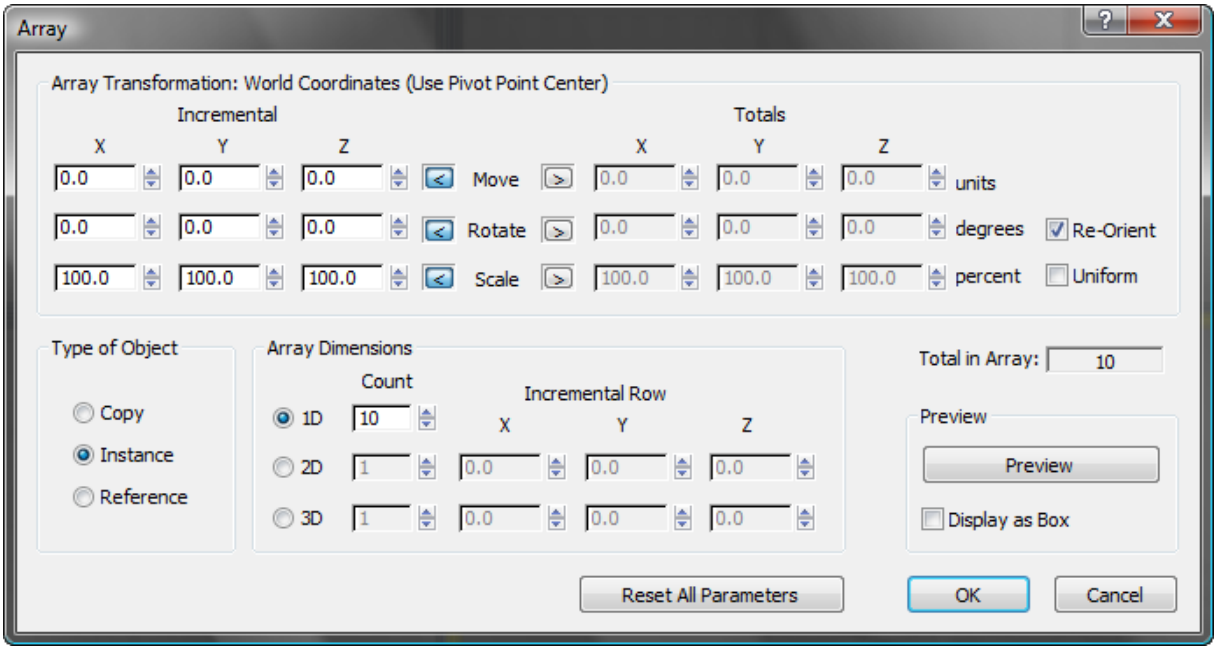


Obr. 9.4 Ilustrace principu nástroje měkký výběr (Soft Selection) [10]

Placement Tool

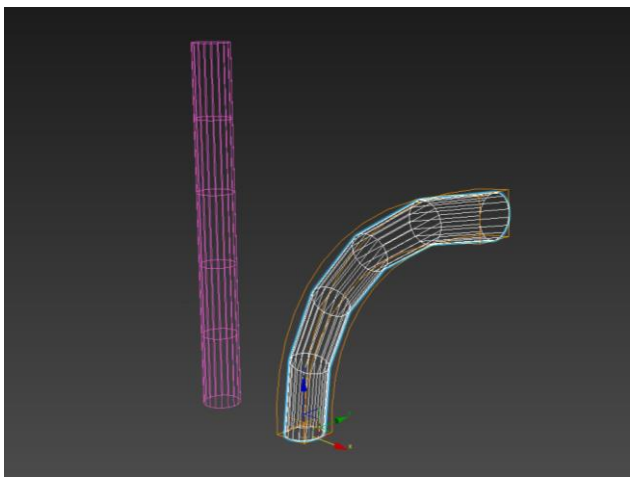
Placement Tool je velmi užitečný nástroj pro umístění objektu přesně na povrch dalšího jiného objektu. Toho jsem využil například při umístění budovy, zarovnání doplňkových objektů vzhledem k povrhu terénu a podobně.

Mnoho detailních prvků se na budovách ve scéně opakují (okna, dveře, sloupy a jiné stavební prvky). Takové objekty byly zkopírovány pomocí nástroje **Copy Instance**, která umožňuje promítnutí změn v jedné kopii na všechny ostatní. V případech, kdy bylo potřeba zkopírovat velké množství objektů, jsem efektivně použil nástroj **Copy Array** (Extras > Array), která umožňuje vytvoření mnoha instancí najednou a jejich kontrolu nad změnou umístění, rotací nebo změnou velikosti najednou při jejich vytvoření. [12]



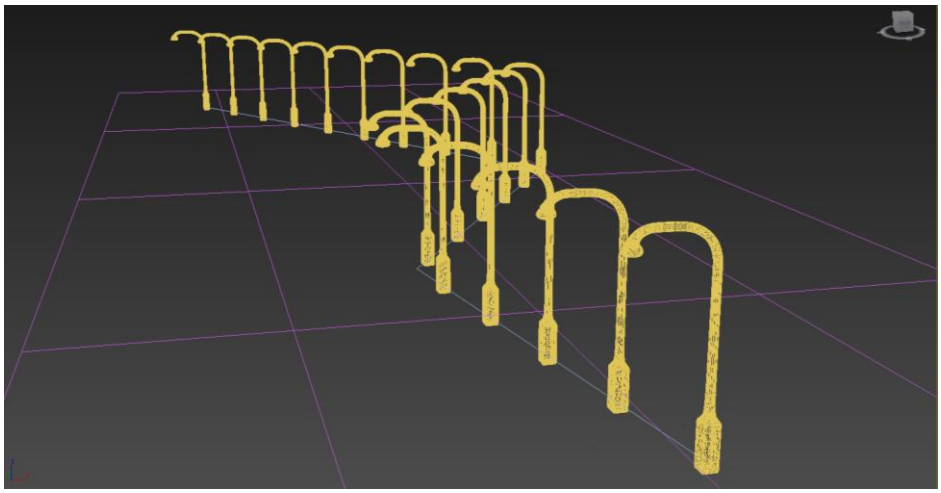
Obr. 9.5 Ilustrace dialogu pro nastavení atributů nástroje Copy Array v aplikaci 3ds Max

Zakroucení stojanu pouličních lamp jsem docílil pomocí modifikátoru **Bend** (Selection > Modify > Object-Space Modifiers > Turbo Smooth), který danou geometrii ohne o daný úhel v daném směru.



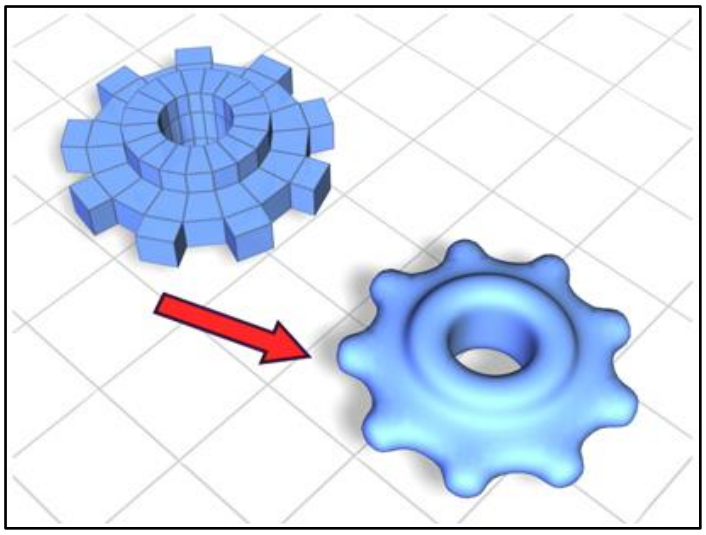
Obr. 9.6 Ilustrace použití modifikátoru Bend

Pro modelování plotů, zábradlí nebo rozmístění lamp je velmi efektivně využitelný nástroj **Spacing Tool** (Tools > Align > Spacing Tool), který umožnuje automatické rozmístění objektů podél křivky.



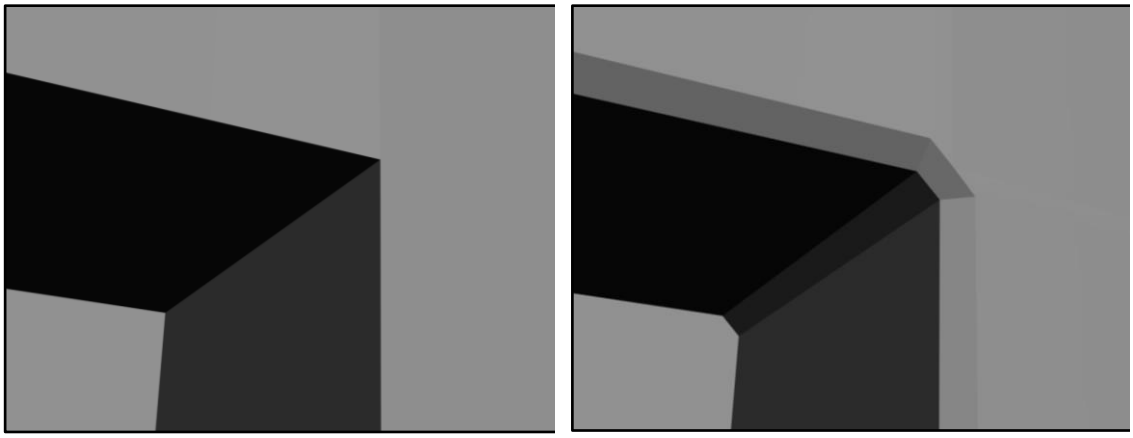
Obr. 9.7 Ilustrace použití nástroje Spacing Tool

Mnoho modelů nebo jejich dílčích prvků, které jsou vytvářeny a skládány z čistých polygonů, je potřeba uhladit a zjemnit přechody mezi jednotlivými polygony. V takových případech je velmi užitečný modifikátor **TurboSmooth** (Selection > Modify > Object-Space Modifiers > Turbo Smooth). Lze tak dosáhnout hladšího povrchu modelu, ale za cenu zvýšení počtu potřebných polygonů.



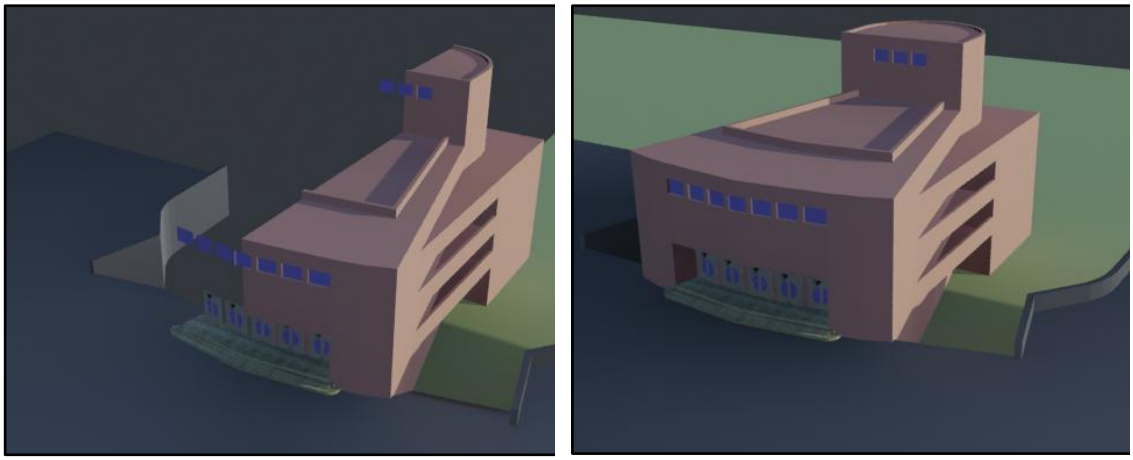
Obr. 9.8 Ilustrace použití modifikátoru TurboSmooth [10]

Ke zjemnění ostrých přechodových hran a rohů některých budov je vhodné použít modifikátor **Chamfer**, který vybrané hrany otupí přidáním dalších rovnoběžných hran.



Obr. 9.9 Ilustrace použití modifikátoru Chamfer

Při modelování symetrických budov je vždy praktické a efektivní vymodelovat pouze jednu polovinu a následně pomocí modifikátoru **Symetry** tuto polovinu podle příslušné roviny symetrie překlopit a vytvořit tak v jednom kroku celou druhou stranu modelované budovy najednou.[2]



Obr. 9.10 Ilustrace použití modifikátoru Symetry

Tímto způsobem bylo vytvořeno 7 typů budov, pouliční lampy, semafory, set dopravních značek, slunečník, židle, stůl, lavice, zastávka a další.

Polygonálně vytvořené dílčí modely budov a dalších prvků městské zástavby, tvoří 183 objektů, jsou složeny z 259 157 polygonů a na jejich povrch je použito 174 materiálů.

Tyto modely byly přidány do importované scény ze CityEnginu a vhodně zakomponovány do modelu města. Následně jsem provedl úpravy terénu a celkové scény.

Kapitola 10
Popis práce s texturami pro vyjádření materiálů

Textury pro vytváření materiálových povrchů budov a objektů jsem pro účely zpracování zadaného modelu města sám nevytvářel, pouze jsem některé upravoval, aby vyhovovaly účelu, pro který byly pak využity. Všechny použité textury nebo textury, které jsem využil až po další úpravě, jsem získal na následujících webových serverech, které nabízejí nespočet volně stahovatelných textur.

- http://texturer.com/
- http://www.textures.com/
- http://www.ftextures.com/

Všechny uvedené servery umožnují volné použití textur pro osobní i komerční účely. V souladu s podmínkami použití textur ze serveru http://www.textures.com/ přikládám následující citát.

"One or more textures on this 3D model have been created with photographs from Textures.com. These photographs may not be redistributed by default; please visit www.textures.com for more information."

10.1 Mapování textur při procedurálním modelování

Vzhledem k tomu, že při procedurálním modelování jsou jednotlivé modely budov generovány na základě společných pravidel, které jsou uvedeny v příslušné gramatice, je často nejefektivnější způsob jak jednotlivé budovy odlišit použití různých povrchových textur.

Pro tento účel lze použít například postup s funkcí randomFile, která umožňuje načítání náhodné textury z poskytnutého výběru. Následující příklad kódu v gramatice čte náhodně texturu ve složce images/windows se jménem začínajícím Window_, za kterým následují libovolné dva znaky, ve kterých se jména textur liší. Například Window_01.jpg, Window_02.jpg ...

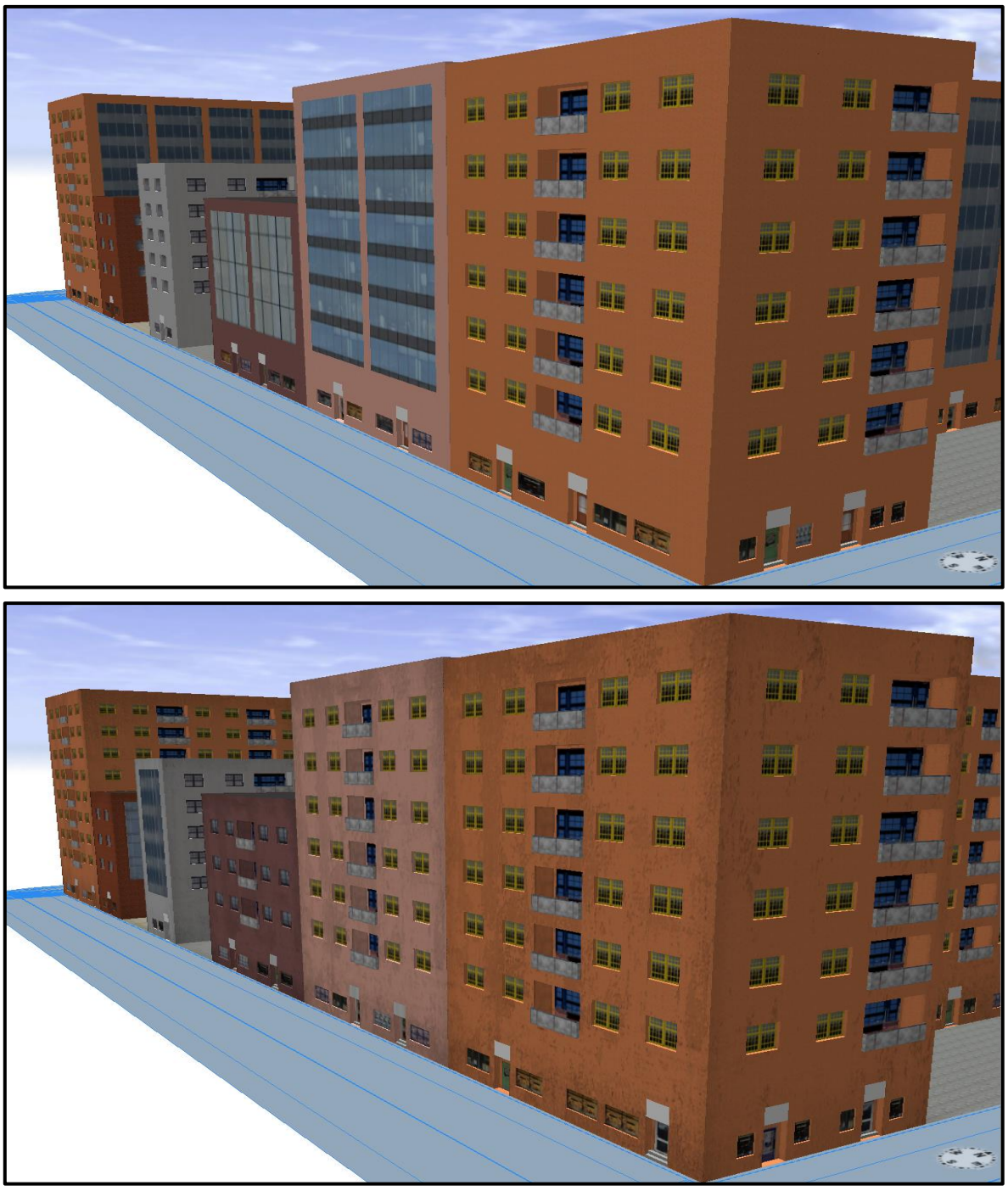
```
const randomWindowTexture = fileRandom("images/windows/Window_**.jpg")

      setupProjection(0, scope.xy, scope.sx, scope.sy)
      texture(randomWindowTexture)
      projectUV(0)
```

Zbytek ukázkového kódu řeší už přímo namapování textury na povrch objektu.

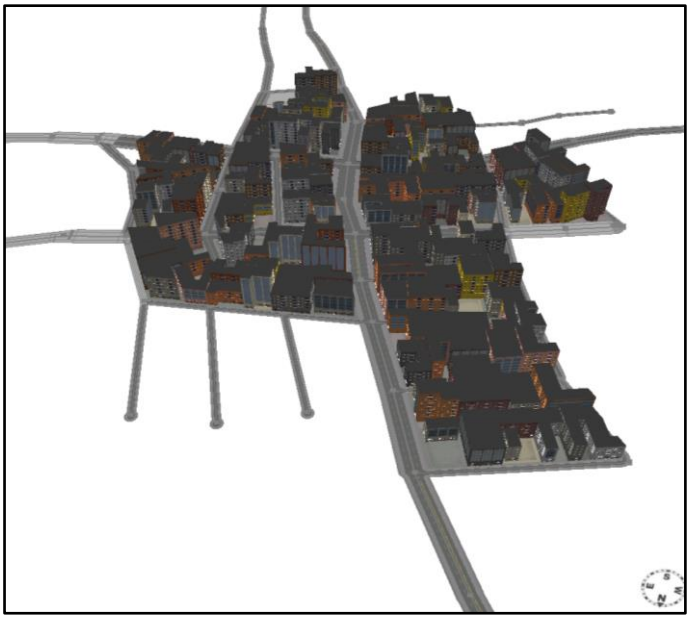
Jelikož jsou modely pokryty texturami, které jsou opakovaně pokládány vedle sebe, působí někdy velmi strojeným a nerealistickým dojmem. K eliminaci tohoto problému lze využít tzv. dirt mapy, což jsou černobílé textury namapované bez opakování na stěny budov, které ztmavují některé části vnějších stěn modelovaných budov. Na budovách v reálném

světě můžeme pozorovat tmavší místa na omítkách v místech nízko u země, pod střechou či pod okny. Vhodně vytvořené dirt mapy mohou simulovat právě tento efekt.

Na obrázku č 10.1 je pro porovnání scéna bez využití dirt mapy a s aplikovanou dirt mapou.



Obr. 10.1 Ilustrace použití dirt mapy



Obr. 10.2 Namapované textury silnic

Modely a textury silnic byly použity z ukázkových souborů knihovny ESRI.lib přiložené k CityEnginu.

Celkově jsem na procedurálně modelované části modelu namapoval 496 různých textur.

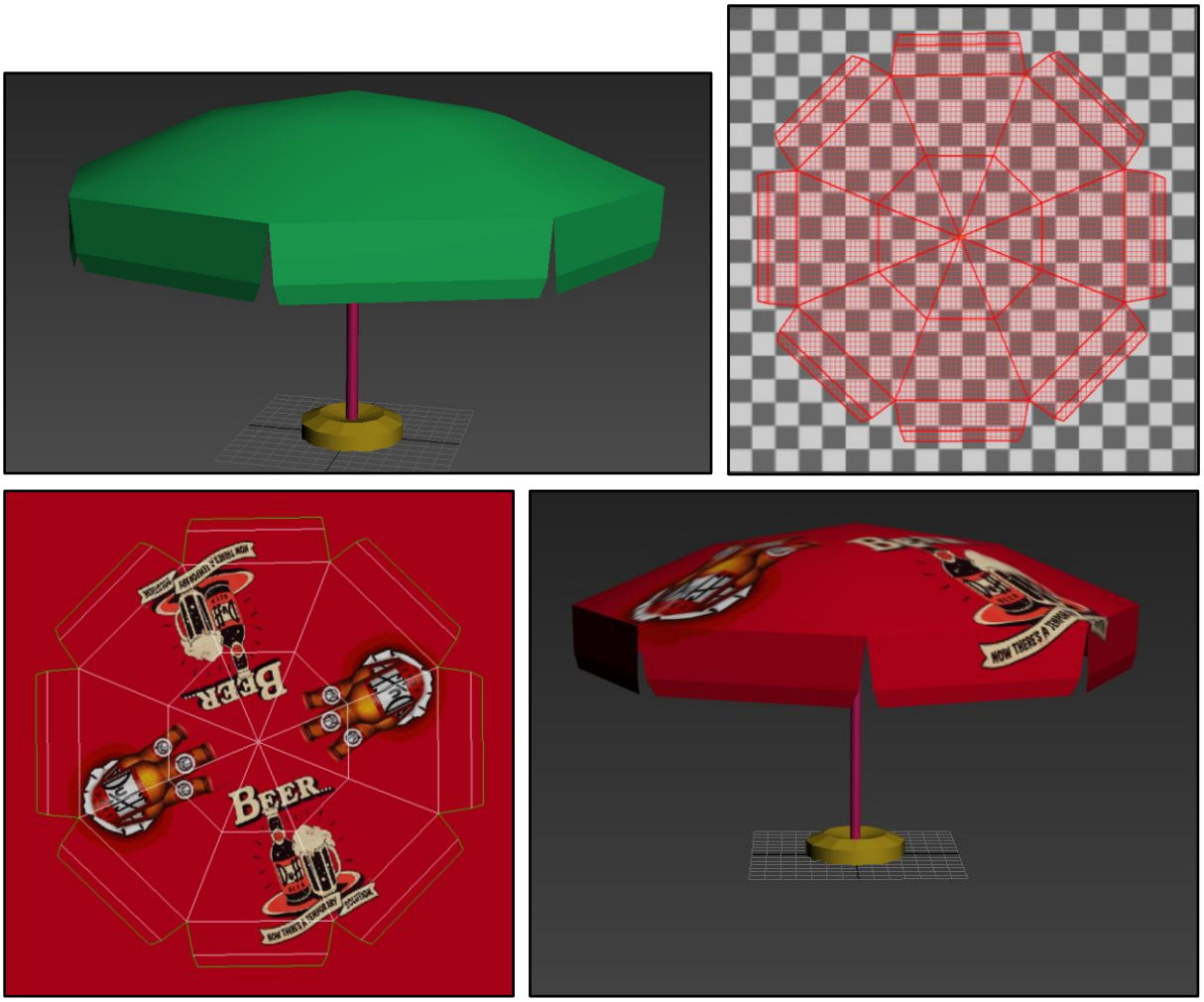
10.2 Mapování textur při polygonálním modelování

Materiály, které nevyžadovaly specifické vlastnosti, byly opatřeny pouze difuzní texturou (color map).

Pro správné namapování textur na objekty v aplikaci 3ds Max byly použity dva typy modifikátorů založených na UV mapování.

Mapování pomocí projekce textury z geometrických těles na povrch objektu byl použit modifikátor **UVWmap** (Select > Modify > Object-Space Modifiers > UVWmap), který umožňuje mapování planární (rovinné), válcové, sférické, krychlové a další. Tento modifikátor kromě běžného UV mapování umožňuje použití složky W, která je užitečná při renderování přídavných efektů jako je shadow mapping. Toho ale v projektu využito nebylo. U stěn budov jsem využil planárního mapování, případně mapování krychlové, u lamp a semaforů pak mapování válcové.

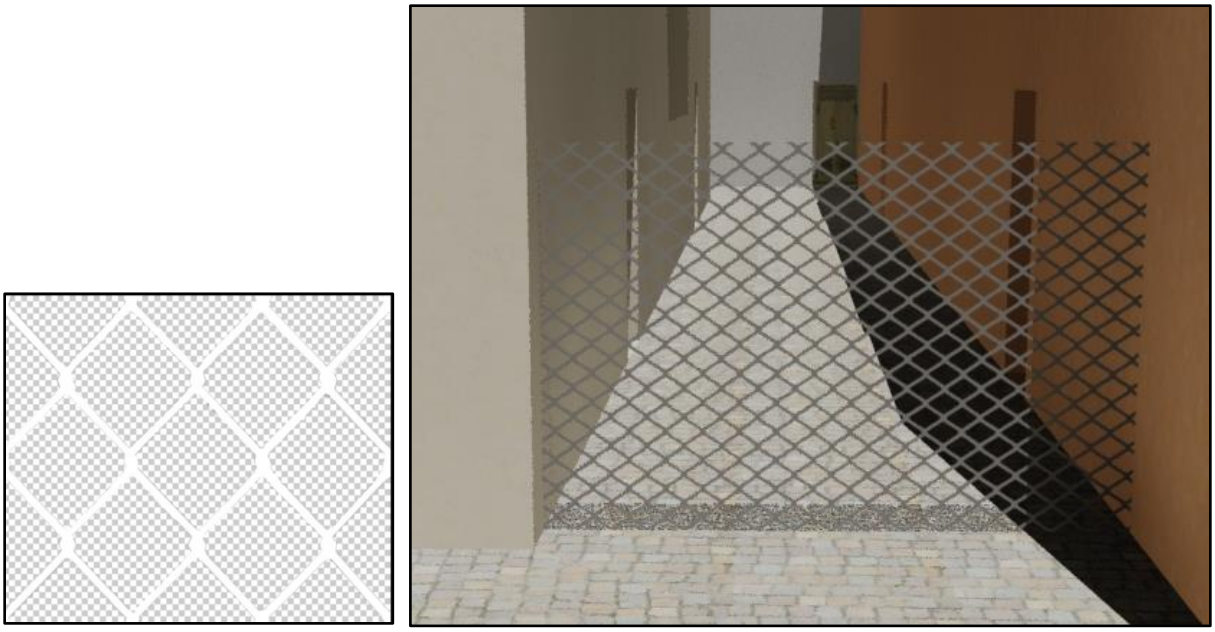
Jako druhý byl pro mapování textur použitý modifikátor **unwrap UVW** (Select > Modify > Object-Space Modifiers > unwrap UVW). Ten byl použit u modelů se složitější strukturou, kde bylo nutné souřadnice upravit manuálně. Příklad použití modifikátoru unwrap UVW vidíme na obrázcích, kde je zobrazeno mapování textury na model slunečníku.



Obr. 10.3 Ilustrační postup při mapování textury na slunečník

U materiálů použitých na objektech s nedokonale rovným povrchem, například u cihel, byly použity bump mapy nebo normal mapy. Mnoho volně dostupných textur obsahovalo i příslušné normal mapy, pro jiné byla normal mapa vytvořena pomocí Nvidia Ray pluginu v programu Adobe Photoshop.

Ploty a zábradlí byly reprezentovány rovinou, jejímuž materiálu byla přiřazena textura s alfa kanály umožňující průhlednost částí objektu.



Obr. 10.4 Ilustrační textury pro vyjádření průhledného plotu [2]

Pro vizualizaci skla v oknech byl použit materiál s černou difůzní složkou, hodnotou lesklosti (glossiness) 77, stupněm odrazivosti (specular level) 90 a průhledností (opacity) 30. Pro ukázku průhlednosti oken byla vytvořena jednoduchá interiérová scéna. Výsledný efekt můžeme pozorovat na následujícím obrázku.

Obr. 10.5 Ilustrace zpracování prosklené výlohy

Pro polygonálně vytvářené modely budov a další objekty, které byly využity v celém modelu města, jsem použil celkem 174 materiálových textur.

10.3 Vytváření light map výsledné scény

Vytváření světelných map probíhá v celkově zkompletované scéně, aby se na sebe jednotlivé objekty mohli vrhat stín. Jako první typ světelné mapy byla vytvořena mapa obsahující ambient occlusion pomocí vestavěného rendereru mentalray. V 3Ds maxu toho docílíme pomocí funkce "Render to texture" (Render > Render to texture) a nastavením v části "output" na "Ambient Occlusion (MR)". Tuto možnost podporuje pouze renderer mental ray. Tato funkce nerenderuje obraz z pohledu pozorovatele, ale místo toho vytvoří novou texturu vybraného objektu a zapíše do ní světelné informace z povrchu.

Podobně jsem postupoval při vytváření klasických light map znázorňujících denní osvětlení ve dvou různých časech. V nabídce output byly vytvořeny jednak „LightningMap" a jednak „CompleteMap". LightningMap obsahuje pouze informaci o vrženém stínu na objekt a používá se v kombinaci s dalšími mapami. CompleteMap „vypéká" do sebe kromě LightningMap i mapy v dalších kanálech (například dirt mapy) a umožňuje tak uložení do souboru OBJ více informací. Často tím však přijdeme o možné vyšší rozlišení textury. Jelikož více domů mohlo obsahovat stejnou texturu, byla vždy vypečená textura uložena do nepoužitého kanálu, aby nedošlo k nežádoucímu přepsání textury u jiných budov. Z důvodu velkého množství objektů jsem použil funkci „Automatic Unwrap", která automaticky namapuje souřadnice objektu do 2D prostoru. Ruční "rozbalování" textury, by bylo příliš časově náročné. 3ds Max defaultně ukládá vypečené textury do formátu TGA, který VRUT neumí přečíst, proto jsem v nastavení vybral PNG soubor.

Kapitola 11
Popis renderování scén modelu a výsledky

V rámci praktické části této bakalářské práce jsem vybral 3 scenérie (2 dílčí části modelu s různou úrovní složitosti modelu a scéna celého výsledného modelu) a testoval jsem míru realističnosti vyrenderovaných statických obrazů z jednotlivých scén a čas potřebný pro jejich vykreslení při použití renderovacích modulů Mental Ray, který obsahuje aplikace Autodesk 3ds Max, a RenderGL a Raytracer, které jsou jako moduly součásti vizualizačního nástroje VRUT. Renderovací moduly Mental Ray a Raytracer pracují s renderovací metodou ray tracing a modul RenderGL renderuje obrazy metodou rasterizace.

V této souvislosti byla také v praktické části testována vizualice celého modelu města v nástroji VRUT tak, aby bylo možné posoudit její použitelnost jako dostatečně realisticky a plynule vykresleného vnějšího okolí pro modul automobilového simulátoru VRUTu v reálném čase s ohledem na způsob renderování, tedy s využitím obou renderovacích modulů Render GL a Raytracer.

11.1 Popis prostředí

Hardware, na kterém probíhalo modelování a renderování obrazu modelových scenérií

Typ počítače	Lenovo Y50-70
Procesor	Intel(R) Core(TM) i5-4210 H CPU – 2.90 GHz
Paměť RAM	8 GB
Operační systém	Windows 10 Pro 64-bit
Grafická karta	Intel(R) HD Graphics 4600
Rozlišení okna renderovaných scén	1920 x 855

Charakteristiky vykreslovaných scenérií

Scéna č.1	
Název souboru	Restaurace
Krátký popis scény	Scéna s předzahrádkou se slunečníky a s portálem s výlohami a okny
Počet objektů	89
Počet polygonů	57 537

Scéna č.2	
Název souboru	Křižovatka
Krátký popis scény	Scéna s křižovatkou a modely v pozadí, na které jsou namapovány textury s normal map
Počet objektů	71
Počet polygonů	43 614

Scéna č.3 – celkový model města	
Název souboru	Město
Krátký popis scény	Celý výsledný model města
Počet objektů	2 564
Počet polygonů	712 879
Počet použitých materiálů	421

11.2 Výsledky renderování

	Scéna č.1	Scéna č.2	Celý model
Čas renderování v Mental Ray	3‘ 16“	2‘ 07“	-
Čas importu ve VRUT RenderGL	4“	3“	5“
Čas importu ve VRUT Raytracer	12“	16“	21“

Snímky obrázků všech scén, vyrenderovaných modulem Mental Ray jsou v příloze B (Výsledky renderování v modulu Mental Ray).

Snímky obrázků všech scén, vyrenderovaných modulem VRUT RenderGL jsou v příloze C (Výsledky renderování v modulu RenderGL).

Snímky obrázků všech scén, vyrenderovaných modulem VRUT Raytracer jsou v příloze D (Výsledky renderování v modulu Raytracer).

Kapitola 12
Závěr

Renderování dílčích scén i celkového modelu ve VRUT modulu Raytracer nevyhovuje podmínkám provozu automobilového simulátoru a vykreslování okolních městských scén na základě vytvořeného modelu v reálném čase. Při renderování statických snímku v tomto modulu lze dosáhnout výborných výsledků a velmi realistického obrazu, není to však vhodné pro rychlé renderování. Jediný funkční renderovací modul pro pro tento účel je RenderGL.

Modelování metodou procedurálního modelování je po určitém zvládnutí CGA jazyka a psaní příslušných gramatik velmi efektivní pro rozsáhlé scény.

V rámci praktické části této bakalářské práce jsem vybral 3 scenérie (2 dílčí části modelu s různou úrovní složitosti modelu a scéna celého výsledného modelu) a testoval jsem v nich míru realističnosti vyrenderovaných statických obrazů.

V této souvislosti byla také v praktické části testována vizualizace celého modelu města v nástroji VRUT tak, aby bylo možné posoudit její použitelnost jako dostatečně realisticky a plynule vykreslovaného vnějšího prostředí pro modul automobilového simulátoru VRUTu.

Jelikož je model města primárně vytvořen pro použití v modulu renderGL byly vytvořeny light mapy k simulaci vrhání stínu. Pro jejich korektní použití v nástroji VRUT bylo nutné "vypéct" light mapy, dirt mapy a opakovatelné textury do jednoho souboru. Ve výsledku bylo možné pozorovat na velkých domech zhoršení rozlišení opakovatelné textury. Tento problém bude možné v budoucnu vyřešit, pokud se nástroj VRUT doplní o podporu více sad texturovacích souřadnic.

Příloha A
Ukázková CGA gramatika

```
/**
 * File:    rule.cga
 * Created: 24 May 2016 11:57:35 GMT
 * Author:  cajka
 */

version "2014.0"

attr myFrontDepth = 10
attr myLeftWidth = 15
attr height_min = 15
attr height_max = 40
attr allyPercent = 20
attr floor_height = 4
attr groundfloor_height = 4

randomSignTexture = fileRandom("images/signs/Sign_***.jpg")
randomDoorTexture = fileRandom("images/Doors/FrontDoor_**.jpg")
randomAllyDoorTexture = fileRandom("images/Doors/AllyDoor_**.jpg")

Lot -->
      10% : Narrow
      else : Parcel

Narrow -->
      offset(-3)
      comp(f) { inside: Parcel | border: Walkway }

Parcel -->
      //rotateScope(0, 30, 0)
      50% : rotateScope(0, -90, 0)
                  shapeL(myFrontDepth,myLeftWidth) {shape : Building |
remainder: Walkway}
      20% : rotateScope(0, 180, 0)
                  shapeU(myFrontDepth,myLeftWidth, myFrontDepth) {shape :
Building | remainder: Walkway}
      else : Building


const randomWallTexture = fileRandom("images/concrete/b**.jpg")
randomDirtTexture = fileRandom("images/dirtmap/dirtmap_*.jpg")

Building -->
      translate(rel, world, 0, 0.2, 0)
      extrude(rand ( height_min, height_max ))
      comp(f){street.front : FrontFacade
                  |side : AllyFacade
                  |top : Roof
                  }

Roof -->
      color(0.2,0.2,0.2)
```

```
FrontFacade -->
	setupProjection(0, scope.xy, 1.5, 1)
	setupProjection(2, scope.xy, scope.sx, scope.sy)
	texture(randomWallTexture)
	set(material.dirtmap, randomDirtTexture)
	projectUV(0)
	projectUV(2)
	split(y){
		groundfloor_height : GroundFloor
		|~1 : UpperFacade
	}

const randomWindowTileTexture =
fileRandom("images/windows/Windows_**.jpg")

UpperFacade -->
	70% :
		split(y){
			2.5 : Wall
			| {~floor_height : Floor}*
			| 1.5 : Wall
		}
	else :
		split(x){
			1 : Wall
			|{ 0.5 : Wall| ~7 : WindowsBase | 0.5 : Wall}*

			|1 : Wall
		}

Floor -->
	split(x){
		1 : Wall
		| {~4 : Tile(split.index)}*
		| 1 : Wall
	}

Tile(index)-->
	case index % 3 == 0 :Balcony
	else : SmallWindow

Balcony-->
	split(y){
		3.2 :
		extrude(-1.3)
		comp(f){
			top : BalconyDoorTile
			| side : WallT
			| bottom : RailingTile
		}
		|~1 : Wall
	}

RailingTile-->
	split(y){
		1.2 : extrude(0.2)
			comp(f) {
				top : Railing
				|left : Railing
				| right : Railing
				| front : WallT
```

```
                  }
      }

Railing-->
      setupProjection(0, scope.xy, scope.sx, scope.sy)
      texture("images/windows/Balcony.jpg")
      projectUV(0)

BalconyDoorTile-->
      split(y){
            ~1 : split(x){
                        0.2 : Wall
                        | ~1 : BalconyDoor
                        | 0.2 : Wall
                  }
            | 0.2 : Wall
      }

BalconyDoor -->
      setupProjection(0, scope.xy, scope.sx, scope.sy)
      texture("images/windows/BalconyTile.jpg")
      projectUV(0)


SmallWindow -->
      split(x){
            ~1 : Wall
            | 2 :
                  split(y){
                        1.2 : Wall
                        |1.8 : UpperWindowTile
                        |~1 : Wall
                  }
            | ~1 : Wall
      }

UpperWindowTile -->
      extrude(-0.2)
      comp(f){
            side : WallT
            | top :
                  split(x){
                        ~1 : UpperWindow
                        |~1 : UpperWindow
                  }
      }

const randomWindowTexture = fileRandom("images/windows/Window_**.jpg")

UpperWindow -->
      setupProjection(0, scope.xy, scope.sx, scope.sy)
      texture(randomWindowTexture)
      projectUV(0)

WindowsBase -->
      split(y){
            2.5 : Wall
            | ~1 : FillWindows
            | 1.5 : Wall
      }
```

```
FillWindows -->
      setupProjection(0, scope.xy, 11.5, 11.5)
      texture(randomWindowTileTexture)
      projectUV(0)

GroundFloor -->
      split(x){
            1 : Wall
            | {~8 : ShopTile}*
            | ~1 : Wall
      }

Floor(floor_index) -->
      split(x){
            1 : Wall
            | {~4 : Tile(floor_index, split.index)}*
            | {~4 : Tile(floor_index, split.index)}*
            | ~1 : Wall
      }

ShopTile-->
      30% :split(x){
                  1 : Wall
                  | 1.5 : EntranceTile
                  | 0.5 : Wall
                  | ~2 : ShopWindowTile
                  | 0.5 : Wall
                  | ~2 : ShopWindowTile
                  | 1 : Wall
            }
      30% :split(x){
                  1 : Wall
                  | ~2 : ShopWindowTile
                  | 0.5 : Wall
                  | ~2 : ShopWindowTile
                  | 0.5 : Wall
                  | 1.5 : EntranceTile
                  | 1 : Wall
            }
      else : split(x){
                  1 : Wall
                  | ~2 : ShopWindowTile
                  | 0.5 : Wall
                  | 1.5 : EntranceTile
                  | 0.5 : Wall
                  | ~2 : ShopWindowTile
                  | 1 : Wall
            }

EntranceTile -->
      split(y){
            2.8 : Entrance
            | ~1 : AboveDoor
      }

AboveDoor -->
      split(y){
            0.6 : Wall
            | 0.01 : extrude(-0.5)SignBase
            | ~1 : Wall
      }
```

```
SignBase -->
	comp(f){
		top : SignSize
	}



SignSize -->
	offset(1)
	split(y){
		~1 : Wall
		|0.6 : SignPlace
		|~1 : Wall
	}


SignPlace -->
extrude(-0.7)
comp(f){
	side : Plast
	| bottom : Sign
}

Sign -->
	setupProjection(0, scope.xy, scope.sx, scope.sy)
	texture(randomSignTexture)
	projectUV(0)


ShopWindowTile -->
	split(y){
		1 : Wall
		| 1.5 : WindowTile
		| ~1 : Wall
	}

WindowTile -->
	extrude(-0.1)
	comp(f){
		side : WallT
		| top : Window
	}

randomShopWindowTexture = fileRandom("images/windows/ShopWindow_**.jpg")

Window -->
	setupProjection(0, scope.xy, scope.sx, scope.sy)
	texture(randomShopWindowTexture)
	projectUV(0)


Entrance-->
	extrude(-0.4)
	comp(f){
		side : WallT
		| top : DoorTile
	}
```

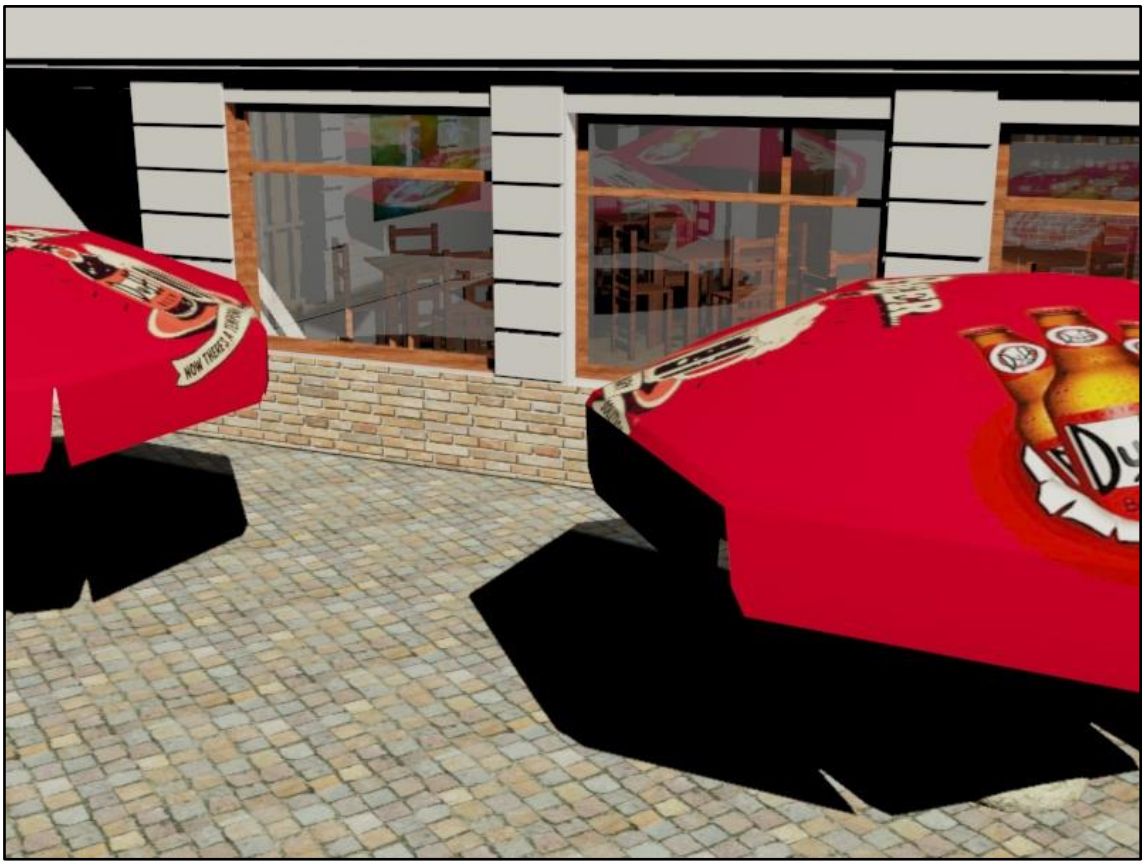
```
DoorTile -->
      30% :
            split(y){
                  2.3 : DoorTile2
                  | ~1 : Wall
            }
      30% :
            split(y){
                  0.2 : Step(0.2)
                  | 2.3 : DoorTile2
                  | ~1 : Wall
            }
      else :
            split(y){
                  0.2 : Step(0.4)
                  | 0.2 : Step(0.2)
                  | 2.3 : DoorTile2
                  | ~1 : Wall
            }

Step(height) -->
      extrude(-height)
      comp(f){all : StepTexture}

StepTexture-->
      setupProjection(0, scope.xy, scope.sx, scope.sy)
      texture("images/concrete/c43.jpg")
      projectUV(0)

DoorTile2 -->
      split(x){
            ~1 : Wall
            | 1 : Door
            | ~1 : Wall
      }


Door -->
      setupProjection(0, scope.xy, scope.sx, scope.sy)
      texture(randomDoorTexture)
      projectUV(0)


AllyFacade -->
      setupProjection(0, scope.xy, ~1, ~1)
      setupProjection(2, scope.xy, scope.sx, scope.sy)
      texture(randomWallTexture)
      set(material.dirtmap, randomDirtTexture)
      projectUV(0)
      projectUV(2)
      split(y){
            groundfloor_height : GroundFloorAlly
            |~1 : UpperFacade

      }

GroundFloorAlly-->
      split(x){
            {~8 : Wall | 1 : AllyEntranceTile | ~8 : Wall }*
      }
```

```
AllyEntranceTile-->
      split(y){
            2.2 : AllyDoorTile
            | ~1 : Wall
      }

AllyDoorTile-->
      extrude(-0.4)
      comp(f){
            top : AllyDoor
            |side : WallT
      }

AllyDoor-->
      setupProjection(0, scope.xy, scope.sx, scope.sy)
      texture(randomAllyDoorTexture)
      projectUV(0)

const randomWalkwayTexture = fileRandom("images/walkway/t**.jpg")

Walkway -->
      extrude(0.2)
      comp(f){ top : WalkwayTop
      }

WalkwayTop -->
      setupProjection(0, scope.xy, ~1, ~1)
      setupProjection(2, scope.xy, scope.sx, scope.sy)
      texture(randomWalkwayTexture)
      set(material.dirtmap, randomDirtTexture)
      projectUV(0)
      projectUV(2)

WallT -->
      setupProjection(0, scope.yz, ~1, ~1)
      setupProjection(2, scope.xy, scope.sx, scope.sy)
      texture(randomWallTexture)
      set(material.dirtmap, randomDirtTexture)
      projectUV(0)
      projectUV(2)
```

Příloha B
Výsledky renderování v modulu Mental Ray

Obr. B.1 Scéna 1 v Mental Ray

Obr. B.2 Scéna 2 v Mental Ray



Obr. B.3 Ukázka - Detail scény 1 ve Scanline rendereru

Obr. B.4 Ukázka - Detail scény 2 v Mental Ray

Příloha C
Výsledky renderování v modulu RenderGL

Obr. C.1 Scéna 1

Obr. C.2 Scéna 2

Obr. C.3 Scéna 3

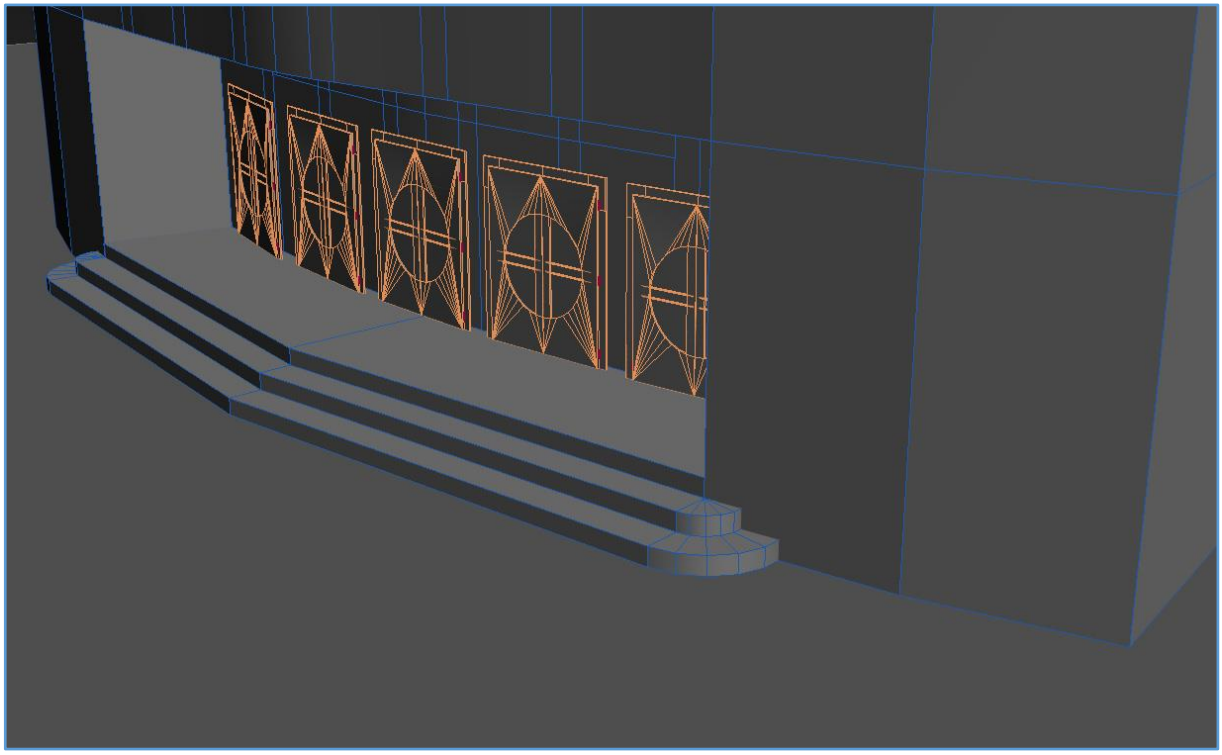
Příloha D
Výsledky renderování v modulu Raytracer

Obr. D.1 Scéna 1

Obr. D.2 Scéna 2

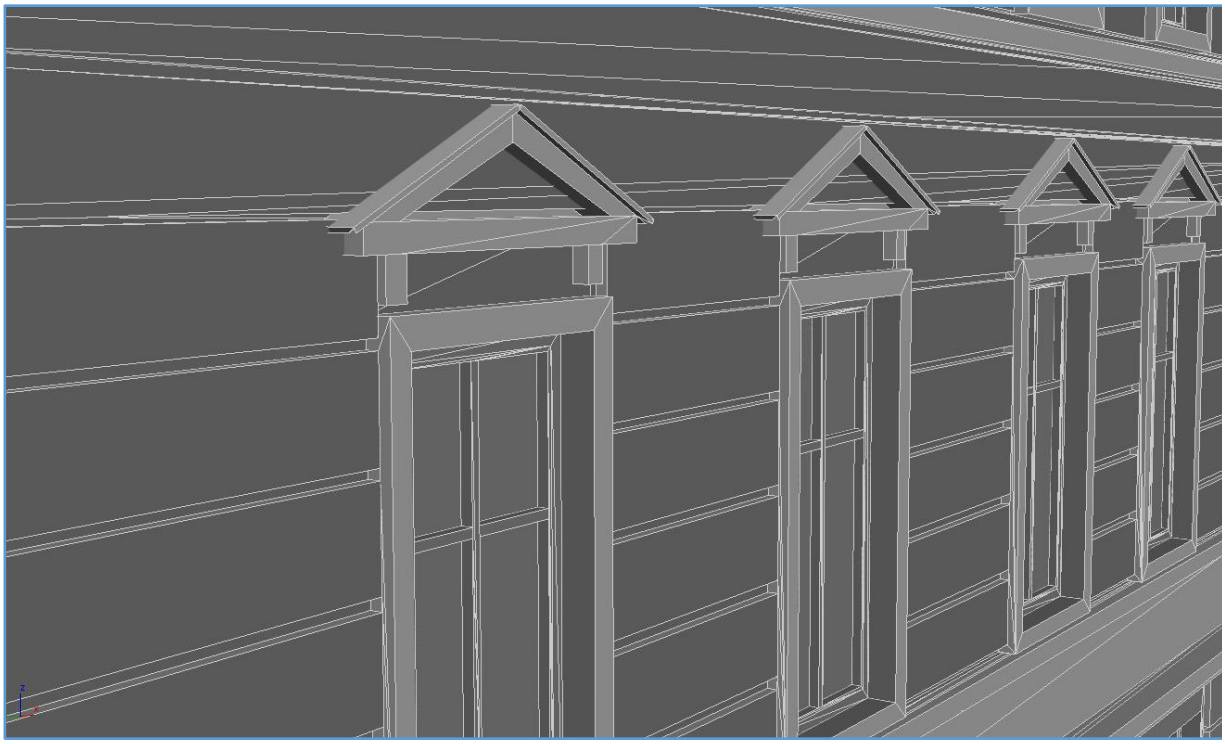
Obr. D.3 Porovnání scén vyrenderovaných modelem renderGL (nahoře) a raytracer (dole)

Příloha E
Ukázky jednotlivých modelů

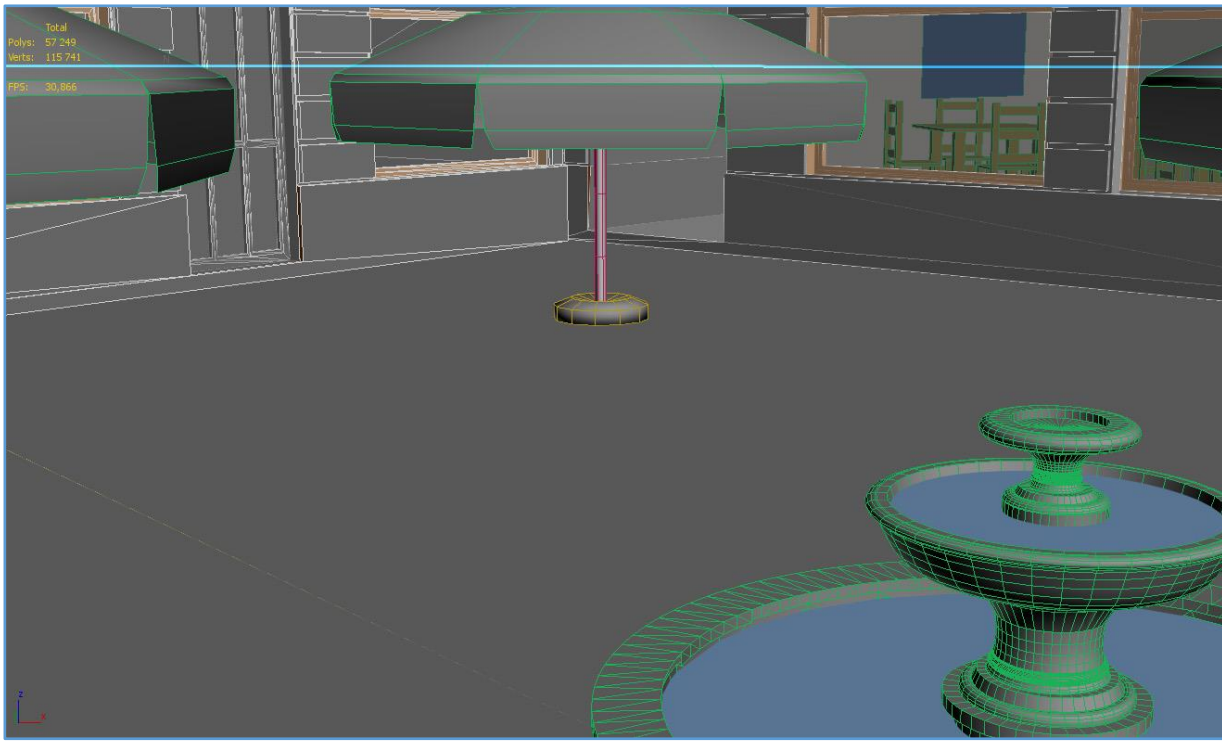


Obr. E.1 Detail modelu vstupního portálu budovy divadla

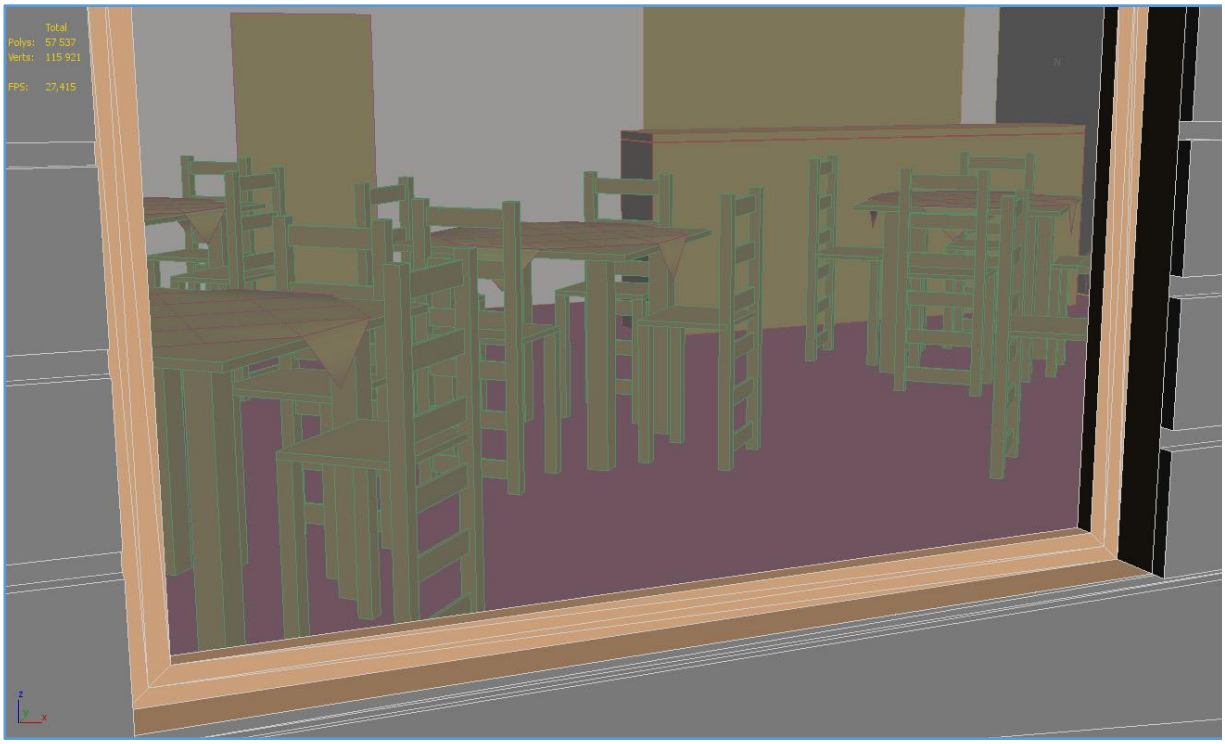
Obr. E.2 Detail modelu okenního průčelí jedné z budov



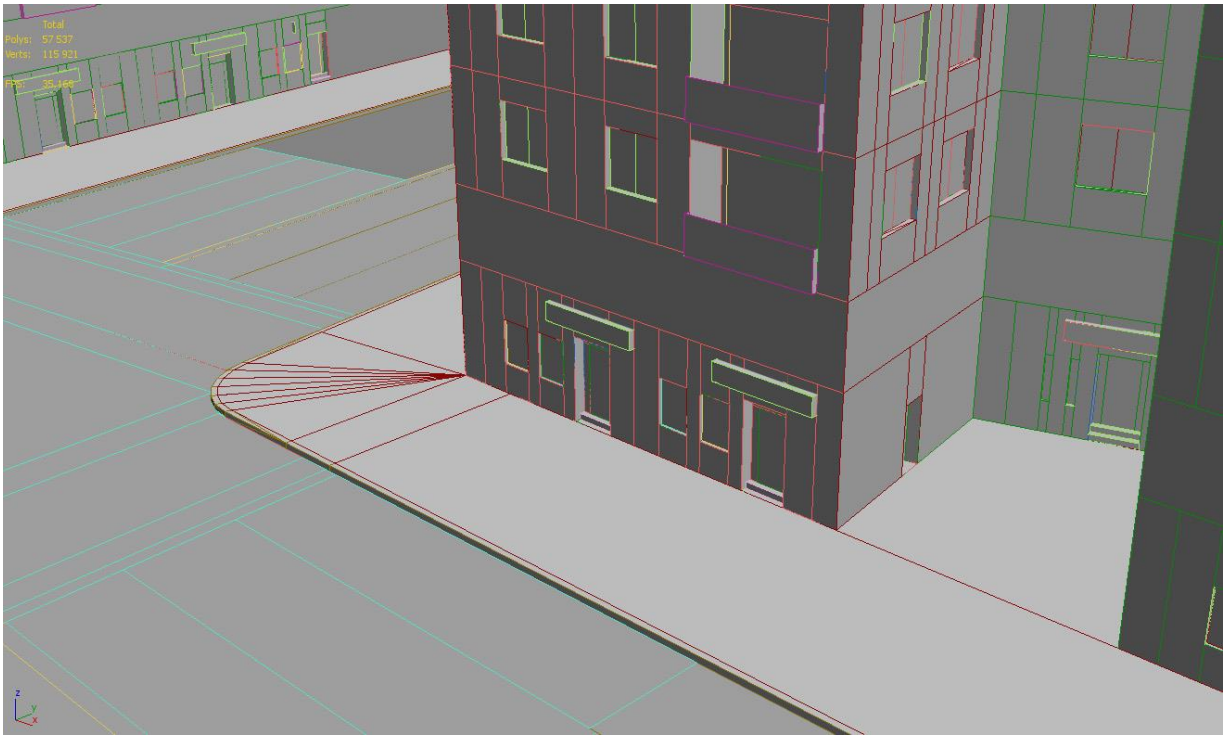
Obr. E.3 Detail modelu okenního průčelí jedné z budov



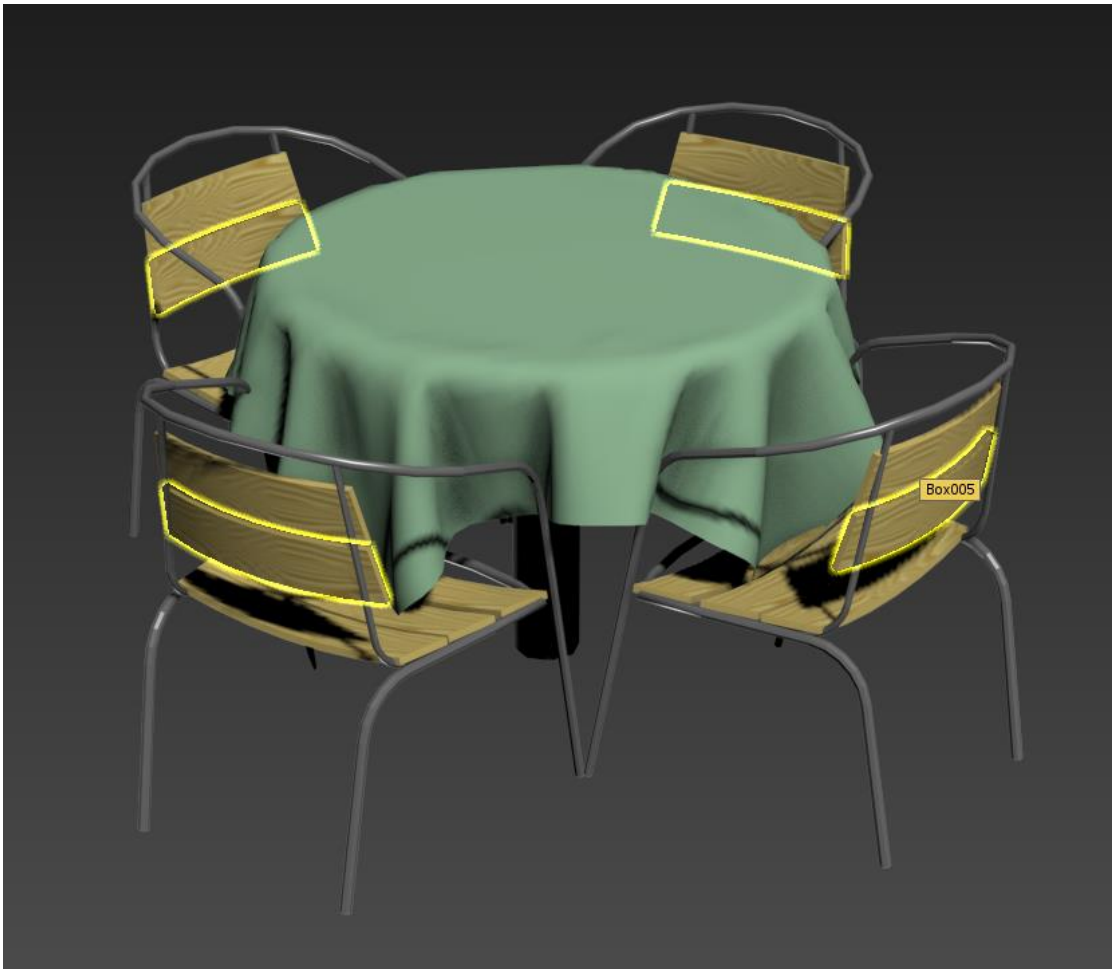
Obr. E.4 Detail modelu restauračního slunečníku a vodního prvku



Obr. E.5 Detail modelu interiéru viditelného za výlohou u jedné z budov

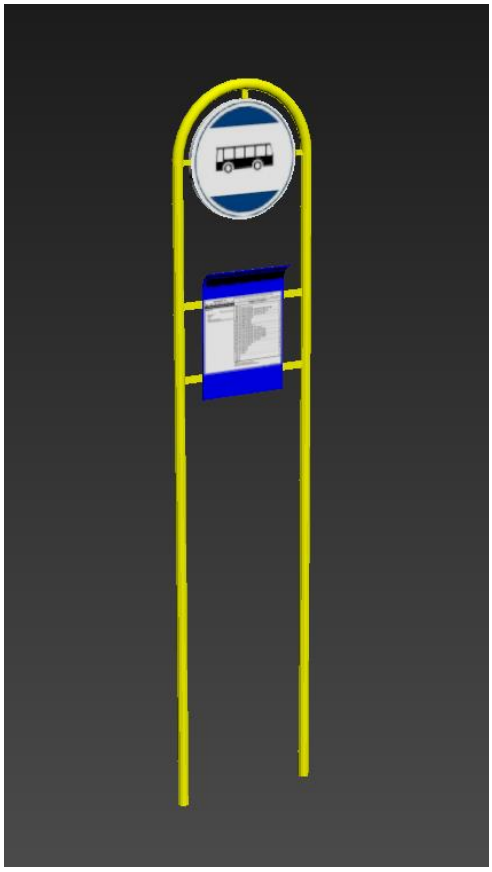


Obr. E.6 Ukázka procedurálně generované budovy v městské zástavbě

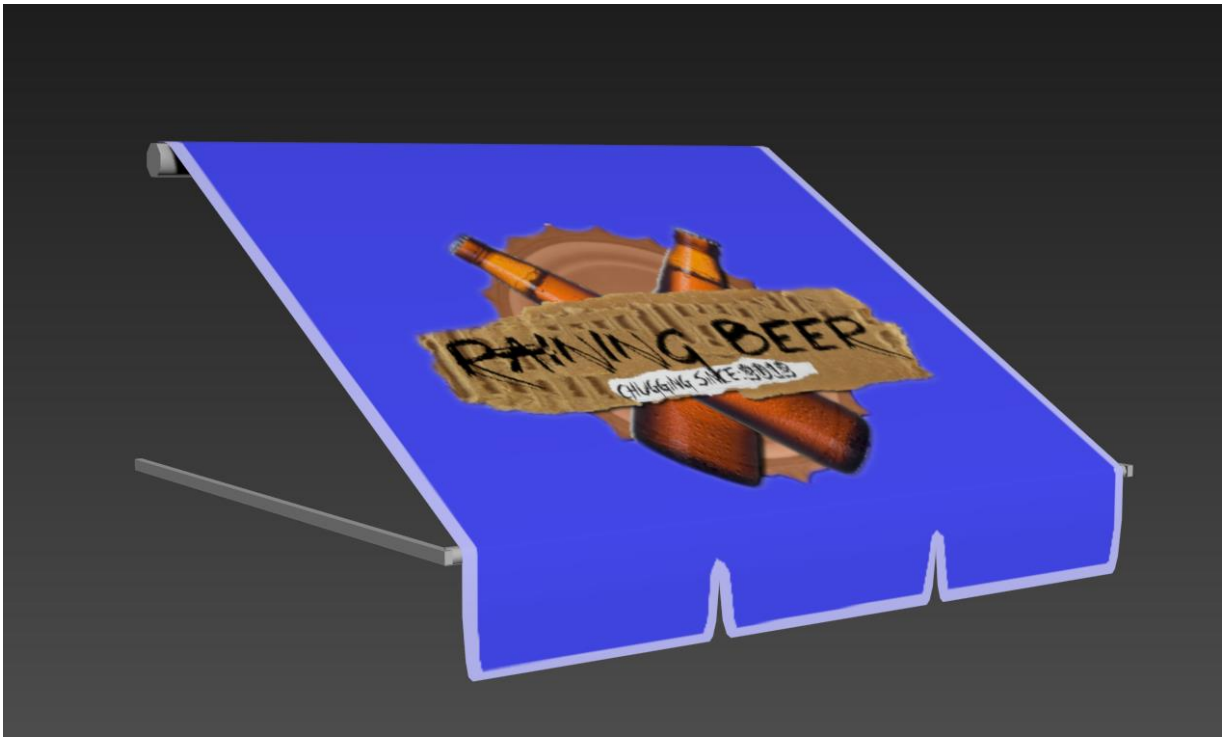


Obr. E.7 Model stolu a židlí

Obr. E.8 Model koše



Obr. E.9 Model Autobusové zastávky



Obr. E.10 Model přístřešku

Příloha F
Seznam použité literatury a informačních zdrojů

[1] Moderní počítačová grafika, Jiří Žára, Bedřich Beneš, Petr Felkel, 1998

[2] Oficiální tutoriály a báze znalostí k produktu 3ds Max
https://knowledge.autodesk.com/support/3ds-max

[3] Wiki stránka o produktu 3ds Max
https://en.wikipedia.org/wiki/Autodesk_3ds_Max

[4] Stránky popisující ArcGIS produkt CityEngine
https://www.arcdata.cz/produkty/arcgis

[5] Oficiální báze tutoriálů k CityEngine
http://desktop.arcgis.com/en/cityengine/

[6] Wiki stránka o společnosti Adobe
https://cs.wikipedia.org/wiki/Adobe_Systems

[7] Článek popisující různé texturové mapy
http://blog.digitaltutors.com/understanding-difference-texture-maps/

[8] Článek porovnávající bump, normal a displacement mapy
http://blog.digitaltutors.com/bump-normal-and-displacement-maps/

[9] Článek popisující teorii funkčnosti normal map
http://www.chrisalbeluhn.com/Normal_Map_Tutorial.html

[10] Popis modifikátorů 3ds Maxu
http://www.3dmax-tutorials.com

[11] Článek popisující plugin SketchUpu
http://www.cad-addict.com/2010/03/sketchup-plugins-extrude-with-rotation.html

[12] Oficiální pomocná stránka k práci v 3ds Maxu
http://docs.autodesk.com/3DSMAX/15/ENU/3ds-Max-Help/index.html

[13] Diplomová práce Procedural 3D modeling and visualization of geotypical Bavarian rural buildings in Esri CityEngine software (Ieva Dobraja, Technická univerzita Mnichov, 2015)
http://129.187.45.33/CartoMasterNew/fileadmin/user_upload/Dobraja_Report.pdf

[14] Wiki stránka o renderování
https://en.wikipedia.org/wiki/Rendering_(computer_graphics)

[15] Stránka popisující CSG stromy
http://www.generativeart.com/on/cic/papersGA2003/a11.htm

[16] Článek Úvod do problematiky polygonového modelování - Ondřej Karlík, Fakulta Elektrotechnická ČVUT, Praha
http://keymaster.powermac.cz/temp/b.pdf

[17] Prezentace Procedurální modelování (Pavel Strachota, FJFI ČVÚT, 2015)
http://http://saint-paul.fjfi.cvut.cz/base/sites/default/files/POGR/POGR2/06.proceduralni_modelovani.pdf

[18] Diplomová práce využití procedurálního jazyka v procesu modelování bloků městské zástavby (Bc. Lucie Koucká, Fakulta přírodovědecká UK Praha, 2015)
https://is.cuni.cz/webapps/zzp/download/120195916

[19] Diplomová práce Modulární 3D prohlížeč (Václav Kyba, FEL ČVUT, 2008)
https://dip.felk.cvut.cz/browse/pdfcache/kybav1_2009dipl.pdf

[20] Bakalářská práce Přesný 3D model společných prostor DCGI (Tomáš Kraus, FEL ČVUT, 2013)

[21] Ukázková color mapa
https://blog.sketchfab.com/art-spotlight-jp-mech/

[22] Ukázková opakovací textura
http://www.textures.com/download/soilcracked0103/46728

[23] Ukázková spekulární textura
https://luxion.atlassian.net/wiki/display/K6M/Specular+Maps#suk=

[23] Wiki stránka o light mapách
https://en.wikipedia.org/wiki/Lightmap

[23] Wiki stránka o ambient occlusion
https://en.wikipedia.org/wiki/Ambient_occlusion

[24] Slidy o renderování v reálném čase
http://slideplayer.com/slide/1507748/

[25] Stránka o rasterizaci
https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation

[26] Stránka o ray tracingu
https://www.siggraph.org/education/materials/HyperGraph/raytrace/rtrace1.htm

[27] CityEngine Help
http://cehelp.esri.com/help/index.jsp?topic=/com.procedural.cityengine.help/html/manual/cga

[28] Texture database
http://www.ftextures.com/Bump-Maps

[29] Texture database
http://www.textures.com

[30] Vray manual – displacement map
https://www.vray.com/vray_for_3ds_max/manual/vray_for_3ds_max_displacement_examples.shtml

Příloha G
Seznam použitých zkratek

2D	dvoudimenzioní
3D	třídimenzionální
CSG	Constructive Solid Geometry (Konstruktivní geometrie pevných těles)
NURBS	Non-uniform Rational Basis Spline
CGA	Computer Generated Architecture
Esri	Environmental Systems Research Institute
VRUT	Virtual Reality Universal Toolkit
ČVUT	České vysoké učení technické
FEL	Fakulta elektrotechnická
CPU	Central Processor Unit
RAM	Random Access Memory

Příloha H
Obsah přiloženého DVD

Přiložené DVD obsahuje následující adresářovou strukturu se soubory:

- **project**
  - **CityEngine** (obsahuje všechny soubory, které tvoří část projektu, vytvořenou v Esri CityEngine)
  - **polygonal model** (soubor s celkovým modelem města ve formátu MAX
    - **soubory dílčích modelů**
  - **VRUT scene** (OBJ a MTL soubory modelu, určené ke vstupu do vizualizačního nástroje VRUT)
    - **Dílčí části celkového modelu v OBJ**
  - **textures** (soubory texturami, které byly využité při tvorbě modelu)
- **images**
  - **scene 1** (obrazy scény 1 ve formátu JPG, renderované moduly Mental Ray, RenderGL a Raytracer)
  - **scene 2** (obrazy scény 2 ve formátu JPG, renderované moduly Mental Ray, RenderGL a Raytracer)
- **pdf** (elektronická verze bakalářské práce ve formátu pdf)