FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Generátor fantasy interiérů
Student:	Michal Pilař
Vedoucí:	Ing. Radek Richtr, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Počítačová grafika
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem práce je pomocí pokročilých technik modeláže a především pomocí procedurálního generování vytvořit generátor fantasy budov včetně jejich interiérů.

1) Proveďte krátkou rešerši problematiky procedurálního generování prostředí. Zaměřte se na obdobné projekty a použití.
2) Vytvořte sadu objektů (židle, skříň, stoly, ...), které bude model obsahovat. Zaměřte se na jednotný vizuál.
3) Vytvořte sadu parametrizovatelných Blender pluginů použitelných pro typové fantasy budovy a interiéry.
4) Demonstrujte možnosti pluginu pomocí sady vytvořených budov.

Elektronicky schválil/a Ing. Radek Richtr, Ph.D. dne 4. února 2022 v Praze.

Bakalářská práce

GENERÁTOR FANTASY INTERIÉRŮ

Michal Pilař

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Radek Richtr, Ph.D.
10. května 2022

Obsah

Poděkování vii

Prohlášení viii

Abstrakt ix

Seznam zkratek x

Úvod 1

1 Rešerše 3
- 1.1 Tvorba prostředí počítačové hry . . . 3
- 1.2 Vytváření modelů . . . 4
- 1.3 Materiály a textury . . . 6
  - 1.3.1 UV unwrapping . . . 7
  - 1.3.2 Typy textur . . . 7
- 1.4 Úvod do procedurálního generování . . . 9
  - 1.4.1 Návrh obsahu pomocí PCG . . . 9
  - 1.4.2 Modální PCG . . . 9
  - 1.4.3 Segmentovaný PCG . . . 10
- 1.5 Výhody a nevýhody PCG . . . 10
  - 1.5.1 Nevýhody . . . 10
  - 1.5.2 Výhody . . . 11
- 1.6 Přístupy k PCG . . . 11
  - 1.6.1 Přístup založený na vyhledávání . . . 11
  - 1.6.2 Metody konstruktivního generování . . . 12
- 1.7 PCG v praxi . . . 14
  - 1.7.1 Procedurálně generované světy . . . 14
  - 1.7.2 Sadow of Mordor . . . 15
  - 1.7.3 Dungeon Alchemist . . . 15

2 Analýza 17
- 2.1 Tvorba modelů . . . 17
  - 2.1.1 Modelování . . . 17
  - 2.1.2 Sculpting . . . 18
  - 2.1.3 Materiály . . . 18
  - 2.1.4 Realistické vs. stylizované modely . . . 19
- 2.2 Implementace algoritmu PCG . . . 20
  - 2.2.1 Použité technologie . . . 22
  - 2.2.2 Použitá metoda generování . . . 22

3 Návrh **25**
3.1 Využití pluginu . . . 25
3.2 Struktura budov . . . 25
3.2.1 Místnosti . . . 25
3.2.2 Objekty . . . 26
3.3 Funkčnost pluginu . . . 26
3.3.1 Logika generování . . . 26
3.3.2 Návrh pravidel . . . 29
3.4 Návrh uživatelského rozhraní pluginu . . . 30

4 Realizace **33**
4.1 Tvorba modelů . . . 33
4.1.1 Modelování . . . 33
4.1.2 UV unwrapping . . . 34
4.1.3 Tvorba materiálů . . . 34
4.1.4 Výsledné modely . . . 34
4.2 Implementace pluginu . . . 36
4.2.1 Implementace procedurálního generování . . . 36
4.2.2 Implementace uživatelského rozhraní . . . 38
4.2.3 Výsledky . . . 41
4.2.4 Instalace pluginu . . . 41

Závěr **45**

A Další výsledky **47**

Bibliografie **54**

Obsah přiloženého média **55**

Seznam obrázků

1.1 Model vytvořen technikou modelování 5
1.2 Model vytvořen technikou sculpting 6
1.3 UV Unwrapping 7
1.4 Ukázka metallic-roughness a specullar-glossines přístupů 8
1.5 Dělení prostoru s využitím BSP stromu 13
1.6 Příklad procedurálně generovaného skřeta ze hry Shadow of War 15
1.7 Příklad vygenerovaného dungeonu pomocí nástroje Dungeon Alchemist 16

2.1 Snímek obrazovky ze hry The Last of Us Part II 19
2.2 Posun v realističnosti u videoherní série Uncharted 20
2.3 Příklad stylizovaných světů 21
2.4 Porovnání realistického a stylizovaného modelu 21
2.5 Příklad využití *geometry nodes* při tvorbě budov 23

3.1 Návrh uživatelského rozhraní 31

4.1 Model stolu se stoličkami 34
4.2 UV mapa modelu stolu se stoličkami 35
4.3 Albedo mapa modelu stolu se stoličkami 35
4.4 Finální render modelu stolu se stoličkami 37
4.5 Uživatelské rozhraní 39
4.6 Příklady vygenerovaných budov 43

A.1 Střední budova se čtyřmi patry 48
A.2 Malá budova se třemi patry 48
A.3 Tři budovy rozmístěné do prostoru 49
A.4 Dvě budovy rozmístěné do prostoru 49
A.5 Průřez malou budovou 50
A.6 Průřez středně velkou budovou 50
A.7 Vytvořený model postele 51
A.8 Vytvořený model kuchyňského stolu 51
A.9 Vytvořený model skříně 52
A.10 Vytvořený model ohniště 52

Seznam tabulek

3.1 Tabulka vztahů mezi jednotlivými objekty a místnostmi 26

4.1 Tabulka vytvořených modelů 36

Seznam výpisů kódu

4.1 Metoda zajišťující generování půdorysu 38
4.2 Úryvek konstruktoru třídy FloorArea 39
4.3 Implementace třídy *Hallway* 40
4.4 Metoda *wallDirection* 41
4.5 Implementace uživatelského rozhraní 42

Chtěl bych poděkovat především své rodině, svým blízkým a své přítelkyni za podporu během studia. Dále bych rád poděkoval vedoucímu této práce Ing. Radku Richtrovi, PhD. za jeho rady a cennou zpětnou vazbu.

Prohlášení

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.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 10. května 2022 .....

Abstrakt

Počítačové hry jsou součástí lidských životů již desítky let. Zatímco dříve měla o počítačové hry zájem pouze úzká skupina lidí, dnes je toto odvětví schopné konkurovat filmovému průmyslu – ať už počtem lidí podílejících se na daných projektech, tak rozpočtem těchto projektů. Tato práce popisuje proces tvorby světů pro počítačové hry a ukazuje, jak je do tohoto procesu možné zapojit procedurální generování. Zároveň je v praktické části vytvořen nástroj, který realizuje právě použití procedurálního generování při tvorbě videoherního světa.

Klíčová slova 3D, procedurální generování obsahu, počítačová hra, virtuální svět, model, polygon, textury, Python, Blender

Abstract

Computer games have been part of human lives for decades. While previously only a small group of people were interested in computer games, today, the industry is able to compete with the film industry – both in terms of the number of people involved in the projects and the budget of those projects. This thesis describes the process of creating worlds for computer games and demonstrates how it is possible to involve procedural generation in this process. Also, a tool that implements the use of procedural generation in the creation of a video game world is created and the process of its creation is described.

Keywords 3D, procedural content generation, computer game, virtual world, model, polygon, textures, Python, Blender

Seznam zkratek

PCG	Procedurální generování obsahu (Procedural Content Generation)
RPG	Hra na hrdiny (Role-Playing Game)
QA	Zajištění kvality (Quality Assurance)
BSP	Binární dělení prostoru (Binary Space Partitioning)
OOP	Objektově Orientované Programování

Úvod

S virtuálními světy se setkáváme čím dál častěji. V dnešní době se dávno nejedná pouze o počítačové hry, ve kterých se můžeme s takovými světy setkat. Ať už se jedná o virtuální světy vytvořené za účelem pořádání konferencí nebo například světy postavené nad technologií blockchain, jejichž tvůrci si slibují budoucí přesun lidstva převážně do online světa, nových případů užití stále přibývá. Za poslední dobu má na rozšíření virtuálních světů bezpochyby zásluhu i koronavirová krize, která vypukla na jaře roku 2020. To, co bylo možné, se muselo co nejrychleji přetransformovat do digitální podoby, a i díky tomu se začaly objevovat nové možnosti, jak virtuální světy smysluplně využít.

Spousta nově vzniklých projektů využívajících nějakou formu virtuálního světa se s velkou pravděpodobností v budoucnu ukáže jako slepá cesta. Opravdu například potřebujeme pořádat konference ve virtuálních světech, a tím z konference dělat prakticky počítačovou hru, když dnes existují nástroje, které nám umožní konferenci pořádat, aniž bychom uživatele zatěžovali například používáním brýlí na virtuální realitu? Není virtuální svět v tomto případě spíš rušivým elementem? Odpovědi na tyto a další otázky přinese jen čas.

Počítačové hry jsou dnes běžnou součástí životů mnoha z nás. Dávno se nejedná o zábavu pro úzkou zájmovou skupinu, na kterou zbytek společnosti nahlíží skrz prsty. Vývoj počítačových her je zároveň bezpochyby zaměření, díky kterému vytváření virtuálních světů nejvíce vzkvétá. Aby také ne, když se dodnes nejspíš jedná o jediný případ (respektive jeden z mála případů), kde dávají virtuální světy rozhodně smysl.

Existuje nespočet metod a postupů, které se při tvorbě virtuálních světů (ať už do počítačových her či k jiným účelům) používají. Jednou z takových metod je i metoda procedurálního generování, která tvůrcům například usnadňuje vytváření obrovských světů. Tato metoda spočívá v tom, že je část práce svěřena do rukou počítače. Zda dává, či nedává smysl tuto metodu použít v zásadě záleží na množství následně ušetřené práce a mnoha dalších aspektech. Vytvářet nástroj, kterým procedurálně vygeneruji tři budovy by ve výsledku bylo pravděpodobně nákladnější a časově náročnější, než vytvořit dané tři budovy ručně. Jakmile ovšem místo pouhých tří budov budeme chtít vytvořit tři města, využití této metody začne dávat mnohem větší smysl. Metoda procedurálního generování s sebou tedy nese jak spoustu výhod, tak nevýhod – ty je důležité si při jejím používání uvědomit a následně je zohlednit v samotném procesu. Tato metoda se dá ovšem použít i v mnoha jiných odvětvích, než jen při vytváření virtuálních světů. Procedurálně generovaná může být hudba, konverzace nebo například hádanka. Způsobů využití procedurálního generování může být takřka neomezeně a záleží jen na tvůrci, zda se touto cestou rozhodne vydat.

Cíl práce

Tato práce má zanalyzovat produkční postup tvorby virtuálních světů a v tomto kontextu použitou metodu procedurálního generování. Cílem je popsat výhody a nevýhody této metody a vysvětlit, kde procedurální generování použít a kde se mu naopak vyhnout. Zároveň budou zanalyzovány obdobné projekty a jejich výsledky.

V praktické části bude následně vytvořen nástroj umožňující procedurální generování budov do virtuálních světů. Tento nástroj bude vytvořen tak, aby byl v budoucnu lehce rozšiřitelný do komplexnějšího nástroje generujícího například celá města. Zároveň budou vytvořeny jednotlivé modely, které bude zmíněný nástroj při generování používat.

Kapitola 1
Rešerše

Tato kapitola se věnuje vytváření videoherního prostředí. Je v ní vysvětlen produkční proces při vytváření takového prostředí, hlavně jednotlivých objektů. Následně je popsána metoda procedurálního generování, jakým způsobem ji použít při tvorbě herního světa, jaké jsou její výhody a nevýhody.

1.1 Tvorba prostředí počítačové hry

V této části je definován pojem *prostředí počítačové hry*. Následně je vysvětlen rozdíl mezi prostředím počítačové hry a level designem. Závěrem se tato část věnuje produkčnímu procesu při vytváření takového prostředí.

Při vývoji počítačových her se můžeme setkat s řadou pojmů. Pro účely této práce je důležité si definovat pojem *prostředí počítačové hry* a vysvětlit si rozdíl mezi tímto pojmem a pojmem *level design*. V knize *Creating Game Environments in Blender 3D* [1] je prostředí počítačové hry definováno následovně:

▶ **Definice 1.1** (Prostředí počítačové hry)**.** *Prostředí počítačové hry (Game Environment) je dimenzí spojující pravidla, cíle, předměty a teoretické aspekty do jednoho celku, přinášející interaktivní tok aktivit.*

Pojem *game environment* je často zaměňován s pojmem *level design*. Tyto dva pojmy se ovšem zásadně liší. Zatímco *game environment* nám udává estetické a architektonické působení dané hry, *level design* určuje například to, jaké výzvy musí hráč během svého hraní překonat, nebo jak je celý svět poskládán do jednoho smysluplného celku. Podrobněji popsal tento rozdíl *Josh Bycer* ve svém článku *Level Design vs. Environment Design* [2] nebo *Luke Ahearn* v knize *3D game environments* [3]. Zjednodušeně by se dalo prohlásit, že *game environment* je určitou podmnožinou *level designu*.

Nedílnou součástí každého prostředí počítačové hry je videoherní obsah (Game Assets). Ten je v již zmíněné knize *Creating Game Environments in Blender 3D* [1] popsán následující definicí:

▶ **Definice 1.2** (Videoherní obsah (Game Assets))**.** *Videoherním obsahem se rozumí cokoliv, co je součástí hry – objekty, scény, materiály, textury, osvětlení a další.*

V následujících částech této kapitoly jsou popsány praktické části procesu tvorby prostředí počítačové hry – nevěnujeme se tedy level designu obecně. Je zde popsán proces tvorby takového videoherního obsahu, který určuje vizuální působení prostředí počítačové hry.

1.2 Vytváření modelů

Nedílnou součástí vytváření jakéhokoliv virtuálního prostředí je tvorba jednotlivých 3D modelů, které reprezentují jednotlivé objekty ve hře. To může být komplikovanější proces, než by se mohlo na první pohled zdát.

Jak popisuje *Renee Dunlop* ve své knize *Production Pipeline Fundamentals for Film and Games* [4], to, jakým způsobem jsou modely vytvořeny, se odvíjí od toho, k jakému účelu modely slouží. Je totiž velký rozdíl, zda se vytváří modely, které mají sloužit při tvorbě vizuálních efektů filmu, nebo zda bude model použit v počítačové hře. Narozdíl od počítačové hry, u filmu nemá konzument možnost ovlivnit, jak se bude jeho obsah vyvíjet. Akce, úhly kamery a tempo, ve kterém se děj odvíjí jsou celé rozhodnuty již při produkci a výsledek je lineární obrazovou a zvukovou sekvencí.

Oproti tomu, ač u počítačových her produkční tým zpravidla vymýšlí příběhovou linii, a rozhoduje vizuální a zvukový styl celé hry, je to nakonec samotný konzument, který rozhodne, co se bude s vytvořeným obsahem dít. Hráč rozhoduje, z jakého úhlu se bude akce zobrazovat, v jakém pořadí se bude zobrazovat a často i co se bude zobrazovat.

To, jakým způsobem model vytváříme, se přímo odvíjí od toho, zda je potřeba ho zobrazovat v reálném čase (jako u zmíněných počítačových her), nebo zda si můžeme dovolit vykreslovat jeden snímek v řádu minut až hodin (jak tomu je při tvorbě vizuálních efektů). Ač při vytváření jednotlivých komponent často tvůrci filmů a počítačových her používají stejné nástroje, liší se způsob, jakým jsou tyto nástroje použity. Pokud totiž chceme vykreslit jednotlivé snímky v reálném čase, potřebujeme, aby zobrazení modelu nebylo příliš výpočetně náročné. Z toho důvodu si například nemůžeme dovolit mít veškeré detaily přímo v geometrii modelu, ale musíme si pomoci jiným způsobem, což zde jsou zpravidla textury.

Samotné modelování je pravděpodobně nejkomplexnější proces při tvorbě kompletních modelů. Jednotlivé modely se skládají ze tří základních prvků – z bodů, hran a ploch. Následující definice jsou převzaty z knihy *Beginning Blender* [5]:

▶ **Definice 1.3** (Vrcholy)**.** *Model je sestaven z řady bodů, které nazýváme vrcholové body (vertex points / vertices). Tyto vrcholy jsou samy o sobě nic neříkající a potřebují něco víc k tomu, aby definovaly objekt.*

▶ **Definice 1.4** (Hrany)**.** *Jednotlivé vrcholy jsou propojeny přímkami, které nazýváme jednoduše hrany (edges). Pomocí hran je již možné zobrazit objekt jako formu drátěného objektu (wireframe).*

▶ **Definice 1.5** (Plochy)**.** *Plochy (faces), též nazývané polygony, nám vznikají, pokud je prostor mezi několika (minimálně třemi – trojúhelník) spojenými hranami vyplněný, čímž tvoří pevný povrch. Z ploch jsme schopni vytvořit solidní objekt. Jednotlivé plochy mohou být různě zbarvené za účelem vytvoření realistických textur objektu.*

R. Dunnlop ve své knize [4] popisuje, že počet ploch při produkci filmu může dosahovat až desítek milionů. U her mají modely zpravidla nižší rozlišení a detaily jsou přidány později, například pomocí promyšleného texturování a stínování. Zvyšující se výkon herního hardwaru ovšem umožňuje i růst počtu ploch u počítačových her.

Problém optimalizace modelů je popsán v knize *Creating Game Environments in Blender 3D* [1]. Polygony mohou mít teoreticky libovolný počet hran, ovšem při vykreslování jsou rozděleny na jednotlivé trojúhelníky. Obecně pak můžeme prohlásit, že čím větší je počet trojúhelníků, ze kterých se objekt skládá, tím je objekt detailnější. Se zvyšujícím se počtem trojúhelníků ovšem roste náročnost vykreslení daného objektu počítačem. K optimalizaci celé scény je tudíž důležité držet počet trojúhelníků, ze kterých se objekt skládá, co nejnižší.

Obrázek 1.1 Model vytvořen technikou modelování z článku *Modeling vs Sculpting* [6].

Modelování vs Sculpting

Samotné vytváření modelů můžeme rozdělit do dvou podkategorií, respektive do dvou technik, které je možné k vytváření modelů použít – *modelování* a *sculpting*[1]. *James Taylor* ve svém článku [6] popisuje rozdíly mezi těmito dvěma technikami a zároveň zde vysvětluje, pro jaký typ modelů se jaká technika více hodí. Ač se na první pohled může zdát, že pomocí obou technik dojdeme snadno ke stejným výsledkům – obě techniky slouží k tomu, abychom vytvořili geometrii pro naše postavy, prostředí a další videoherní obsah – rozdíly jsou zde zásadní.

Modelování

Modelováním se rozumí proces, kdy manipulujeme s jednotlivými polygonálními tvary. S objekty se pracuje na úrovni jednotlivých komponent – tedy vrcholů, hran a ploch – s cílem vytvořit komplexnější objekt. Při správné čtyřúhelníkové topologii modelu můžeme velmi efektivně manipulovat s velkými oblastmi našeho modelu pomocí rychlého výběru kružnic a smyček skládajících se z na sebe navazujících hran. Zároveň můžeme dosáhnout vysoké přesnosti výběrem a manipulací individuálních komponent našeho modelu. To nám umožní provádět velmi specifické změny na modelu – ovšem za cenu zpomalení pracovního postupu.

Modelování se hodí při vytváření předmětů s tvrdým povrchem, což jsou zpravidla strojově či člověkem vytvořené předměty, které se skládají například z rovnoběžných linek nebo pravých úhlů. Modelování je také důležité z technické stránky – schopnost ovlivnit topologii daného modelu je klíčová, pokud vytváříme model určený k animování. Vyhnout se modelování je naopak rozumné při vytváření organických objektů. Příklad modelu, při jehož tvorbě byla použita technika modelování je na obrázku 1.1.

Obrázek 1.2 Model vytvořen technikou sculpting z tutoriálu *Concept Sculpting for Film and Games* [7] od autorů webové stránky *www.flippednormals.com*.

Sculpting

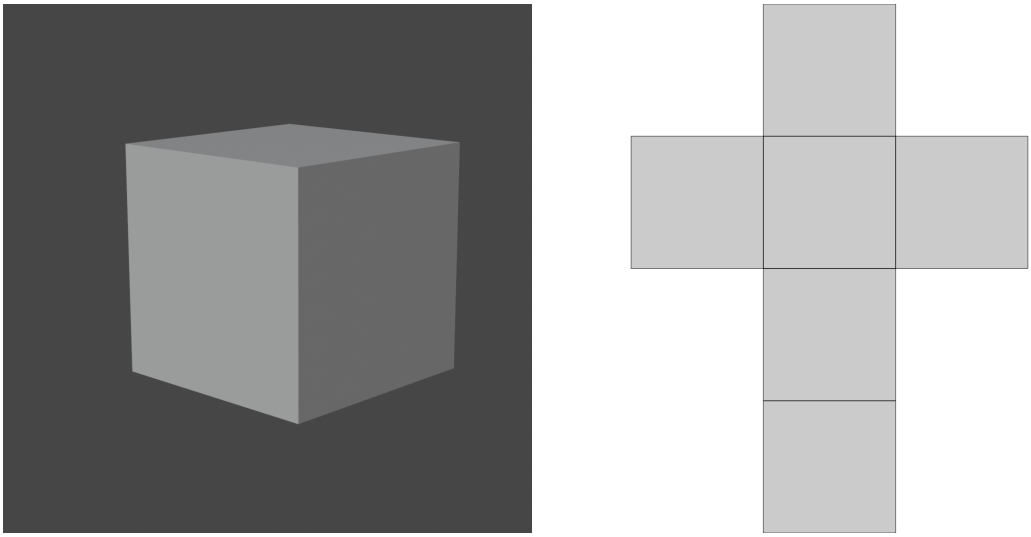
Sculpting se oproti modelování spoléhá na manipulaci s objektem pomocí štětce. Při sculptingu nemanipulujeme s konkrétními komponenty modelu, ale místo toho model deformujeme pomocí nejrůznějších štětců. To činí sculpting extrémně rychlou a intuitivní metodou – nevybíráme zde totiž jednotlivé vrcholy, se kterými budeme manipulovat, ale manipulujeme s větším počtem vrcholů najednou pomocí štětce. Sculpting ovšem oproti modelování postrádá přesnost – je zde velmi obtížné, až nemožné, manipulovat s jednotlivými komponenty modelu. Jednotlivé štětce zároveň většinou vytvářejí hladký povrch, kvůli čemuž je náročné vytvářet ostré hrany.

Sculpting je nejlepším nástrojem pro vytváření organických modelů, jako jsou postavy, příšery, rostliny nebo oblečení. Pomocí sculptingu lze také velmi rychle vytvořit vysoce detailní modely. Jak už z textu vyplývá, sculpting se nehodí na vytváření objektů s tvrdým povrchem. Ač u většiny sculpting nástrojů najdete možnost, jak takové modely vytvořit, je jednodušší pro modely vyžadující přesnost použít modelování. Objekty vytvořené sculptingem mají také téměř vždy moc vysoké rozlišení na to, aby mohly být použity k animování – u takových modelů je potřeba provést proces retopologie. Retopologií se rozumí vymodelování stejného modelu, jaký byl vytvořen pomocí sculptingu, ovšem s mnohem nižším detailem a se správnou topologií. Detailní model vytvořený sculptingem je zde následně použit k vytvoření textur, které dodají detail tam, kde ho model s nízkým rozlišením postrádá. Příklad modelu, při jehož tvorbě byla použita technika sculpting je na obrázku 1.2.

1.3 Materiály a textury

Materiály přidávají objektům nejrůznější optické vlastnosti, například s cílem, aby se jejich vzhled co nejvíce přiblížil vzhledu objektu v reálném světě. Jsou nejčastěji reprezentovány sadou textur.

[1] Sculpting by se dal přeložit jako *sochání*, ovšem i mezi česky mluvícími 3D grafiky se z pojmu *sculpting* stal známý termín, který je lepší ponechat bez překladu, aby nemátl čtenáře a ten byl schopen si bez problému na toto téma dohledat další informace.



(a) (b)

Obrázek 1.3 Na obrázku 1.3a je zobrazen 3D model. Na obrázku 1.3b je zobrazena UV mapa (2D reprezentace) tohoto modelu.

Jejich cílem mimo jiné je, aby přidaly detaily tam, kde není možné je reprezentovat samotnou geometrií. Přiřazování textur (respektive materiálů) objektu nazýváme *mapování textur*. Definice tohoto pojmu je převzata z knihy *Beginning PBR Texturing* [8] a zní následovně.

▶ **Definice 1.6** (Mapování textur)**.** *Mapování textur (Texture mapping) je proces přiřazování barev jednotlivým pixelům na 3D modelu s cílem simulovat čtyři atributy: barvu (color), odráženou barvu (specular color), nerovnost povrchu (surface perturbation) a průhlednost (transparency).*

1.3.1 UV unwrapping

Vzhledem k tomu, že materiály pro 3D modely jsou zpravidla reprezentovány pomocí 2D obrázků, neboli textur, je potřeba u každého modelu, kterému chceme přiřadit materiál, projít procesem, který se nazývá *UV unwrapping*. Tento proces definujeme následovně:

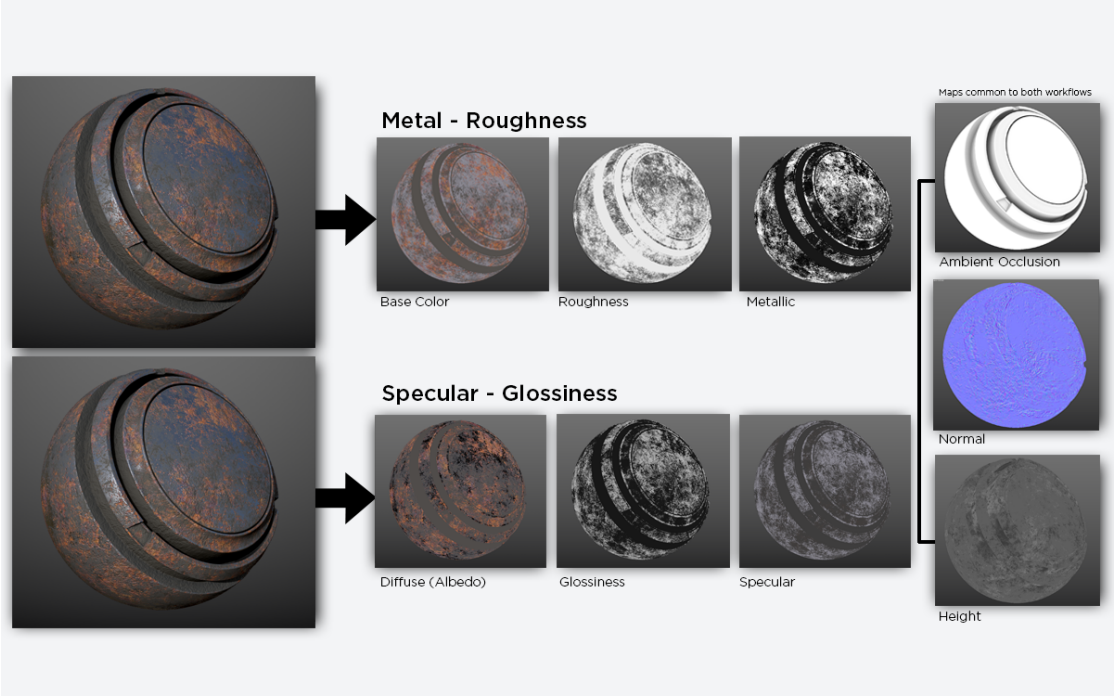
▶ **Definice 1.7** (UV unwrapping)**.** *UV unwrapping je metoda, pomocí které vytváříme takzvané UV mapy, neboli 2D reprezentaci libovolného 3D modelu.*

Příklad UV mapy se nachází na obrázku 1.3. UV unwrapping je potřeba provést u každého modelu, který chceme později texturovat. Dnes již ovšem existují nástroje, které nám tento proces značně usnadní nebo ho za nás provedou celý. Automaticky provedený UV unwrapping ovšem nemusí vyhovovat našim potřebám a zásahu do finální podoby UV mapy se tedy často nevyhneme.

1.3.2 Typy textur

V knize *Beginning PBR Texturing* [8] jsou popsány jednotlivé typy textur, jejichž kombinací vzniká požadovaný vzhled daného modelu. To, jaké konkrétní typy textur se pro daný model použijí, závisí na použitém herním enginu a na pracovním procesu studia vyvíjejícího hru. Typy textur mohou být následující.

1. *Diffuse map.* Difuzní mapa reprezentuje celkovou barvu modelu. Jedná se o nejběžnější typ textury. Tato textura zároveň nese informaci o světlech a stínech.



Obrázek 1.4 Ukázka *metallic-roughness* a *specullar-glossines* přístupů z tutoriálu publikovaného společností *Adobe* na téma PBR texturování [9].

2. *Albedo map.* Albedo mapa je podobná difuzní mapě, pouze s tím rozdílem, že nenese žádnou informaci o světle a stínech.

3. *Transparency/alpha map.* Většinou se jedná o mapu obsahující pouze stupně šedi. Tradičně bílá barva na této mapě znamená, že objekt je zcela viditelný a černá barva naopak znamená, že objekt je zcela průhledný. Odstíny šedi mezi černou a bílou barvou nesou informaci, jak moc je objekt průhledný.

4. *Specular map.* Tato mapa určuje odlesk povrchu v daných místech.

5. *Bump map.* Bump mapa je určena k tomu, aby vytvořila iluzi nerovnosti povrchu. Může se použít k vytvoření škrábanců, odřenin a jiných drobných povrchových detailů. Iluze hloubky závisí na úhlu kamery mířící na objekt. Čím více se hodnota mapy blíží k bílé, tím více detaily vystupují z povrchu a naopak. Výsledný efekt nezávisí na jednotlivých vrcholech modelu.

6. *Normal map.* Tento typ mapy se používá k zobrazení detailů s vysokým rozlišením na modelu s nízkým rozlišením. Textura mění směr normály v místě daného pixelu pomocí chorochromatické mapy.

7. *Displacement map.* Tato mapa na rozdíl od bump mapy závisí na vrcholech modelu. Displacement mapa generuje fyzickou hloubku modelu podle toho, jaký odstín šedé se nachází v daném místě mapy.

8. *Ambient occulsion map.* Tato mapa se používá pro generování stínů a tmavých míst na základě blízkosti objektů.

Existují i další typy map, které mohou být užitečné pro dosažení nejlepších možných výsledků, jako jsou metallic mapy, parallax mapy, glossiness mapy, roughness mapy a další. Jak již bylo zmíněno dříve, to, jaké mapy budou při vývoji počítačové hry potřeba závisí na tom, jaký

engine se při vývoji používá. Pokud je používaný engine založen na takzvaném *mettalic-roughness* přístupu, musí být použity albedo, metallic, roughness, normal a ambient occulsion mapy. Tento přístup používá například engine *Unreal Engine 4*. Pokud ovšem daný engine používá *specular-glossiness* přístup, musí být použity diffuse, specular, glossiness, normal a ambient occulsion mapy. Na tomto přístupu je založen například herní engine *Unity*. Příklad použití těchto dvou přístupů je zobrazen na obrázku 1.4.

1.4 Úvod do procedurálního generování

Jak je již zmíněno v úvodu, procedurální generování je metoda, díky které lze předat část tvůrčího procesu do rukou algoritmu. Původ a dodnes největší využití má procedurální generování ve vývoji počítačových her.

Definice se v různých pramenech drobně liší, ovšem myšlenka zůstává stejná. *Gillian Smith* [10] například definoval procedurální generování obsahu následovně (jedná se o velmi obecnou definici, která pokrývá veškeré formy procedurálního generování):

▶ **Definice 1.8** (Procedurální generování obsahu – PCG)**.** *Procedurálním generováním obsahu se rozumí použití formálního algoritmu k vygenerování obsahu, který by byl jinak vytvořen lidskou rukou.*

Tanya X. Short a *Tarn Adams* ve své knize [11] popisují, že ručně vyráběný obsah může paradoxně působit více uměle, než obsah vytvořený počítačem. Lidští designéři mají často potřebu to přehánět s množstvím designových prvků a vést hráče až moc lineární cestou – to může způsobit, že se hráč cítí méně jako dobrodruh a více jako malé dítě vedené za ruku. V tomto smyslu může být lhostejnost algoritmu vůči hráči velikou výhodou. Stejně jako skutečný svět, procedurálně generovaný obsah se nezajímá o to, zda na konci naší cesty čeká něco zajímavého. Používání procedurálního generování ovšem neznamená, že do vytvořeného díla nezasáhne vůbec lidská ruka. Tento zásah ovšem udává pouze širší směr, kterým se bude výsledný obsah ubírat. V knize [11] jsou následně popsány tři příklady, ve kterých lze PCG použít – *návrh obsahu pomocí PCG*, *modální PCG* a *segmentované PCG*.

1.4.1 Návrh obsahu pomocí PCG

PCG je ve vývoji her často použité k vygenerování velkého množství obsahu, který je následně ručně upraven. Typický případ tohoto přístupu jsou hry s velkým otevřeným světem, jako je například *Skyrim*, jehož součástí je právě velký svět, který může hráč prozkoumat. Dalším příkladem mohou být logické hry – v tomto případě generátor vytvoří stovky logických úloh odpovídajících dané hře a člověk ručně vybere ty nejzajímavější. V těchto případech přichází rozhodnutí použít PCG brzy, ovšem jeho implementace není klíčová v počátcích projektu. U generátoru krajiny pro RPG hru (hru na hrdiny), jako je již zmíněný Skyrim, nám stačí, když je schopen generovat hrubý návrh – všechny důležité detaily, se kterými hráč interaguje jsou následně doplněny ručně. Díky tomu je možné lépe předpovědět celkovou časovou náročnost projektu. V tomto případě použití PCG začíná a končí v raných fázích projektu, následně se pouze upravuje vygenerovaný obsah.

1.4.2 Modální PCG

Některé hry spoléhají minimálně na PCG, ale mají speciální módy hraní, které využívají procedurálně vygenerovaný obsah. Jedná se o vedlejší část hry, která je nějakým způsobem napojena na část hlavní – hráč má například možnost vedle plnění hlavní příběhové linie prozkoumávat procedurálně generované jeskyně, ovšem může se tomuto obsahu zcela vyhnout. Vzhledem k tomu,

že se jedná o oddělené módy, mohou být vytvářeny až v pozdější fázi projektu nebo dokonce vydány jako rozšíření dávno po vydání původní hry.

Procedurálně generovaný mód, který nějakým způsobem napodobuje původní hru, se může ukázat jako méně nákladný, co se zdrojů týče, neboť návrháři již mají jasnou vizi, jak má výsledný produkt vypadat a jak mají fungovat jednotlivé herní mechaniky.

Na druhou stranu, tento způsob svádí k vytváření nepříliš zábavného obsahu, který je více náhodný, než procedurální. Většina her, do kterých byl takový mód implementován se často spoléhá například pouze na vlny náhodně generovaných nepřátel, namísto vytváření zábavného a pohlcujícího procedurálně generovaného herního zážitku.

1.4.3 Segmentovaný PCG

Určité segmenty ve hře mohou být odděleny od ostatního videoherního návrhu s cílem použít PCG. Lineární, ručně navržená hra může mít například procedurálně generovanou hudbu. Díky tomu, že se jedná o segment oddělený od zbytku hry, je možné při výskytu nějakého problému přejít k ručně vytvořenému obsahu, aniž by se výrazně zvýšila časová náročnost.

1.5 Výhody a nevýhody PCG

Většina zásadních výhod a nevýhod již vyplývá z předchozích částí textu. V této části jsou shrnuty zásadní výhody a nevýhody použití PCG, jež jsou podrobněji popsány v již zmíněné knize *Procedural Generation in Game Design* [11].

1.5.1 Nevýhody

Zajištění kvality

Jedním z největších problémů při použití PCG ve hrách, konkrétně v rozsáhlých AAA titulech je zajištění kvality, zkráceně QA. Hra, která hojně využívá PCG nemusí být schopna garantovat správnou funkčnost ve všech případech. Je prakticky nemožné otestovat každou možnou iteraci procedurálního generování. Generátor může chybovat v pouhých 0,1% případů, na které se nepřijde, dokud není hra vydána pro širokou veřejnost.

Časová omezení

Ač je PCG často chápáno jako nástroj, který vývojářům ušetří čas, opak může být pravdou. Vývoj generátoru může být náročnější, než se původně očekávalo a tím výrazně zvýšit časovou náročnost celého projektu. Práce na generátoru může být zároveň potřebná i po jeho dokončení, kvůli nutnosti častých úprav, které jsou potřeba v závislosti na dalším vývoji. Ruční vytváření obsahu lze mnohem snadněji časově odhadnout.

Přehnaná náhodnost

Pokud projekt nemá dostatečné zdroje nebo dostatečný čas k vytvoření zajímavého procedurálně generovaného obsahu, může obsah ve hře skončit jako až moc náhodný, čímž se hra stává nudnou a repetitivní. V takovém případě je lepší se použití PCG vyhnout.

1.5.2 Výhody

Šetření času

Oproti časovým omezením uvedeným v předchozí části může PCG při správném použití ve skutečnosti čas šetřit. To se projeví hlavně v případě, kdy chceme vygenerovat takové množství obsahu, že vytvořit ho ručně by mohlo být prakticky nemožné. Za příklad můžeme dát opět RPG hry s velkým otevřeným světem, kde ač je vygenerovaný výsledek ručně upravován, šetří zde PCG spoustu času a zdrojů.

Rozsah

Pomocí PCG mohou herní světy nabírat obrovských rozměrů. Lze generovat zdánlivě nekonečné světy, jako například celé galaxie. Hry jako *No Man's Sky* – které se věnuje následující část – je téměř nemožné vytvořit ručně.

Znovuhratelnost

Unikátní obsah vytvořený pomocí PCG může zajistit, že hra bude pro hráče při každém průchodu jiná a tím zůstane zábavná i při několikátém hraní. PCG může v tomto případě přimět hráče se ke stejné hře opakovaně vracet.

1.6 Přístupy k PCG

Existuje vícero možností, jak k PCG přistupovat. Jednotlivé možné přístupy jsou popsány v knize *Procedural Content Generation in Games* [12]. V této části jsou popsány jak přístupy, které jsou aplikovatelné při řešení problému procedurálního generování budov, tak přístupy hodící se při generování jiného obsahu.

1.6.1 Přístup založený na vyhledávání

Přístup založený na vyhledávání spočívá v hodnocení každého vygenerovaného artefaktu. Artefakt je přijat pouze pokud splní minimální požadované hodnocení, jinak je generován znovu. Artefakty, které hodnocení splňují, je následně možno kombinovat nebo ještě vylepšovat.

Základní komponenty jsou pro tento přístup následující:

- *Vyhledávací algoritmus.* Jedná se o hlavní komponentu tohoto přístupu. Relativně jednoduché evoluční algoritmy jsou často zcela dostačující, ovšem mohou nastat situace kdy je výhodnější využít komplexnější algoritmus.

- *Reprezentace obsahu.* Jedná se o reprezentaci artefaktů, které chceme generovat. Reprezentace obsahu definuje (a tím i limituje), jaký obsah může být vygenerován, a určuje, zda je možné efektivní vyhledávání.

- *Vyhodnocovací funkce.* Tato funkce nám pro každý artefakt vrátí číselné ohodnocení daného artefaktu. Výsledek tohoto ohodnocení nám může udávat například hratelnost vygenerované úrovně. Vytvoření vyhodnocovací funkce, která spolehlivě ohodnotí aspekty daného artefaktu, patří mezi nejsložitější úkoly při implementaci přístupu založeném na vyhledávání.

1.6.2 Metody konstruktivního generování

Metody konstruktivního generování na rozdíl od přístupu založeném na vyhledávání neohodnocují své výstupy, za účelem je následně přegenerovat. Většina těchto metod je zároveň jednoduchá na implementaci. Nejčastější použití mají tyto metody při generování *dungeonů*, ovšem dají se použít i pro generování jiného videoherního obsahu. Pojem *dungeon* definuje *N. Shaker* [12] následovně:

▶ **Definice 1.9** (Dungeon)**.** *Jde o labyrintové prostředí skládající se z navazujících výzev, odměn a hádanek poskládaných smysluplně v čase a prostoru tak, aby vytvářely vysoce strukturovaný videoherní zážitek.*

Dungeony se nemusí ovšem nacházet pouze v počítačových hrách – příkladem může být klasická papírová (nepočítačová) hra na hrdiny s názvem *Dračí doupě* (nebo její původní zahraniční verze *Dungeons & Dragons*). Ve většině počítačových RPG her se *dungeony* skládají z několika místností, které jsou propojeny chodbami. Může se jednat o jeskynní komplexy, lidské stavby a další. Procedurálním generováním *dungeonů* se rozumí generování topologie, geometrie a objektů jinak souvisejících s hraním úrovně tohoto typu. Typicky se metoda generující *dungeony* skládá z následujících tří prvků.

1. *Reprezentativní model.* Abstraktní zjednodušený model reprezentující samotný *dungeon.*
2. *Metoda ke zkonstruování reprezentativního modelu.*
3. *Metoda vytvářející reálnou geometrii dungeonu z jeho reprezentativního modelu.*

Metod konstruktivního generování je mnoho. Následující části textu se věnují dvěma metodám, které jsou nejvíce používány při procedurálním generování dungeonů. První metodou je *metoda dělení prostoru* a druhou *metoda agentem řízeného růstu.*

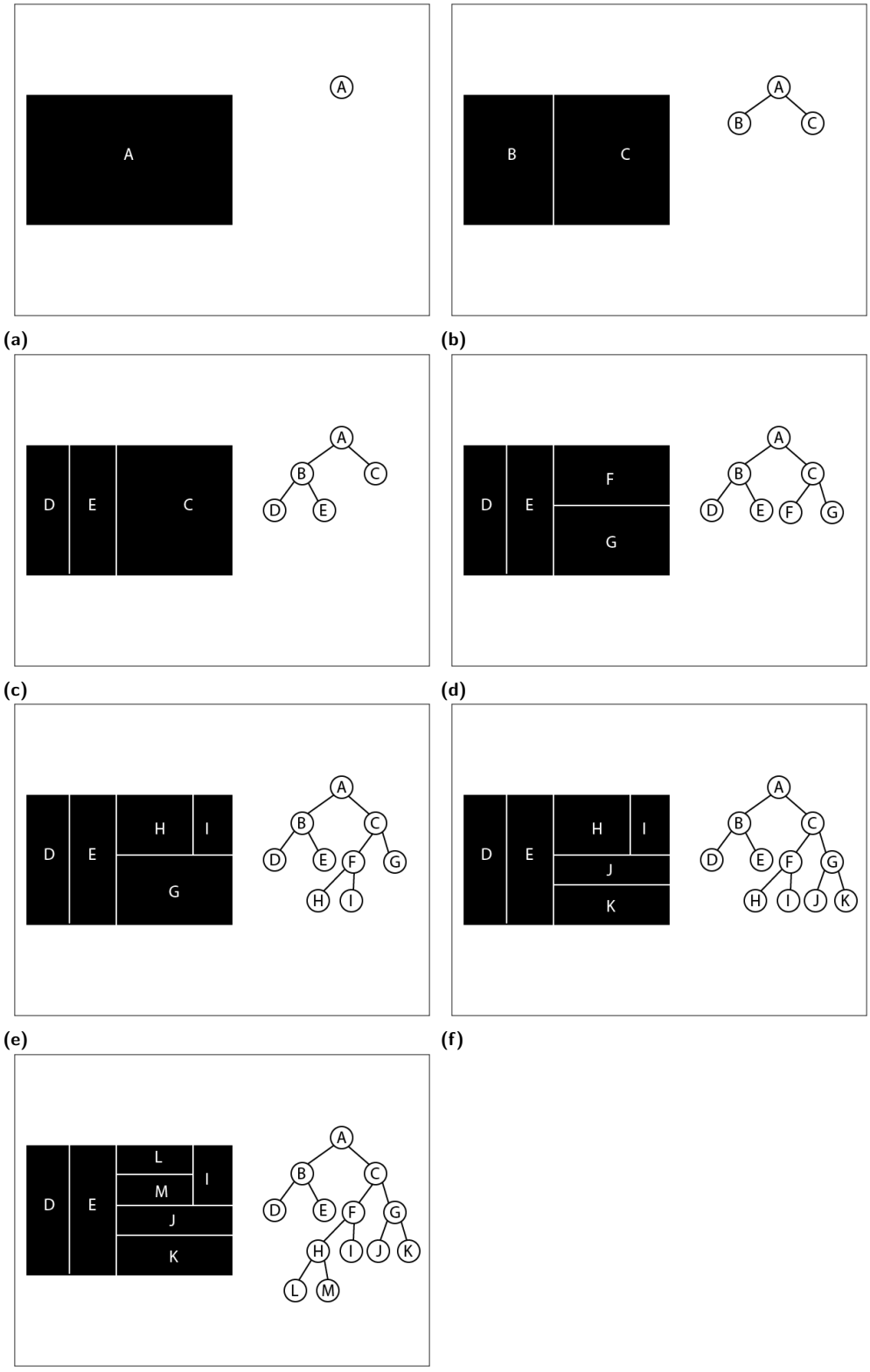
Metoda dělení prostoru

U metody dělení prostoru nám algoritmus rozděluje zadaný prostor na disjunktní podmnožiny tak, aby každá část zadaného prostoru ležela v jedné z těchto podmnožin. Tyto algoritmy často pracují hierarchicky – každá podmnožina zadaného prostoru je algoritmem rekurzivně dále dělena. To umožňuje jednotlivé podmnožiny seřadit v takzvaném *prostor-dělícím stromu.*

Nejpopulárnější metodou pro dělení prostoru je *binární dělení prostoru* (binary space partitioning – BSP), která rekurzivně dělí prostor na dvě podmnožiny. U této metody můžeme prostor reprezentovat pomocí binárního stromu, který nazýváme *BSP strom.* V popisu algoritmu 1 je popsán konkrétní algoritmus používající BSP strom k dělení prostoru za účelem vytvoření dungeonu. Tento algoritmus byl převzat z již zmíněné knihy [12]. Grafická reprezentace algoritmu je znázorněna na obrázku 1.5.

Metoda agentem řízeného růstu

Další metodou konstruktivního generování je *metoda agentem řízeného růstu.* Zjednodušeně se tato metoda dá popsat jako metoda, při které jeden agent postupně kope tunely a tím vytváří chodby a místnosti. Tato metoda se více hodí k vytvoření organického a chaotického prostoru, jako jsou například jeskyně. Vzhled výsledného dungeonu vytvořeného tímto přístupem se liší podle vlastností, které nastavíme agentovi. Pokud agenta implementujeme jako z většiny náhodného, může jeho postup vyústit v chaotickou změť chodeb, které se mohou v mnoha místech protínat a vytvářet tak dungeon připomínající spíše bludiště. To ovšem nemusí být na škodu a možná je tohle právě cíl, kterého chce tvůrce používající tuto metodu dosáhnout. Agenta můžeme zároveň implementovat tak, aby dával větší pozor na to, kam kope, a tím nevytvářel již zmíněné protínající se chodby. To ovšem záleží na preferencích samotných tvůrců.



Obrázek 1.5 Grafická reprezentace Algoritmu dělení prostoru pomocí BSP stromu. Prostor je nejprve rozdělen na podmnožiny B a C. Podmnožina B je následně rozdělena na podmnožiny D a E. Jak podmnožina D, tak podmnožina E jsou již moc malé na další dělení, tudíž zůstanou jako listy v BSP stromu. Algoritmus se přesouvá na podmnožinu C a pokračuje stejně jako u předchozí podmnožiny.

Algoritmus 1 Algoritmus dělení prostoru s využitím BSP stromu

1: Začni s celým prostorem dungeonu (kořen BSP stromu)
2: Rozděl zvolenou oblast podél vertikální nebo horizontální osy na dvě nové oblasti
3: Vyber jednu ze dvou nových oblastí
4: **if** Oblast je větší, než minimální akceptovatelná velikost **then**
5: Jdi na krok 2 (Použij tuto oblast jako oblast určenou k dělení)
6: **end if**
7: Vyber druhou oblast a jdi na krok 4
8: **for** Každou oblast **do**
9: Vytvoř místnost v této oblasti tak, že náhodně vybereš dva body (levý horní a pravý spodní) uvnitř zvolené oblasti
10: **end for**
11: Počínaje nejnižší úrovní BSP stromu, vytvoř chodby spojující jednotlivé místnosti, které jsou potomky stejného rodiče v BSP stromu
12: Opakuj krok 11 dokud nejsou spojeni potomci kořene BSP stromu

1.7 PCG v praxi

Jak už vyplývá z předchozího textu, je mnoho způsobů, jak PCG při vývoji své hry (a nejen hry) využít. Nejběžněji je tento nástroj využit při generování světů, ale generovat mohou vývojáři například také nepřátele, hudbu či dokonce příběh. V této části jsou popsané zajímavé způsoby použití PCG při vývoji počítačové hry.

1.7.1 Procedurálně generované světy

Jak je shrnuto v článku s titulem *The Daggerfall Paradox* [13], ač je PCG mocným nástrojem, vygenerované světy často postrádají duši a jejich jednotlivé části mohou působit na pozorovatele až identicky. To je dáno samozřejmě tím, že tvůrce odevzdává část tvůrčího procesu do rukou počítače. Zmíněný článek tuto zásadní nevýhodu ilustruje na dvou videoherních titulech. Jedním je *The Elder Scrolls II: Daggerfall* a druhým *No Man's Sky*. Ač mezi vydáním obou her uběhlo zhruba dvacet let, tyto hry mají jeden společný charakterový rys – u obou vsadili jejich tvůrci na procedurálně generovaný svět.

Daggerfall se dnes jako druhý díl legendární série již řadí mezi kultovní počítačové hry. Jedná se o jeden z prvních titulů, kde tvůrci nechali počítač vytvořit gigantický procedurálně generovaný svět. Rozloha tohoto světa odpovídala zhruba čtvrtině rozlohy Spojeného království Velké Británie a Severního Irska.

S hrou *No Man's Sky* je to podobné, ač v mnohem větším měřítku - hráč v ní může prozkoumat až 18 kvintilionů (18 000 000 000 000 000 000) unikátních planet a měsíců, které byly vygenerovány pomocí počítačového algoritmu. Procedurálně generovaný obsah je u této hry klíčovou vlastností – závisí na něm celkový pocit ze hraní a celý smysl hry je právě objevování zdánlivě nekonečného procedurálně generovaného vesmíru. Rozhodnutí použít procedurální generování úzce souviselo s rozhodnutím, o jaký typ hry se bude jednat.

No Man's Sky bylo ve své době velmi očekávaným titulem, ovšem po jejím vydání byla tato hra pro většinu hráčů obrovským zklamáním. Na vině byla právě zdánlivá identičnost jednotlivých planet. Autor zmíněného článku [13] připodobňuje důvody neúspěchu této hry právě k *Daggerfallu*, jedné z prvních her, kde bylo PCG použito pro generování otevřeného světa a kde tvůrci narazili na podobné problémy již o dvacet let dříve.

Daggerfall v době svého vydání (1996) také vystupoval z řady právě kvůli do té doby nevídané rozloze herního světa (tento svět je dodnes jedním z největších videoherních světů vůbec). Ač byla ve hře spousta možností co dělat a co objevovat, a ač činy hráče mohly ovlivnit osud království,

Obrázek 1.6 Příklad procedurálně generovaného skřeta ze hry Shadow of War – obrázek z článku *10 Things To Know About Shadow Of War's Nemesis System* [15]

svět této hry rozhodně nevzbuzoval dojem, že by byl vytvořen hráči na míru. Aby vývojáři docílili onoho dojmu realistické rozlohy světa, uchýlili se k podobné, ač mnohem primitivnější strategii, jako vývojáři *No Man's Sky*. K PCG se vývojáři podle článku [13] uchýlili z toho důvodu, aby v hráči vzbudili pocit, že opravdu žije v obrovském fantasy světě. Zároveň bylo jejich cílem nenutit hráče vydat se jedinou předepsanou cestou. Výsledkem nakonec bohužel bylo, že všechna místa, kam hráč přišel, vypadala relativně podobně. Města měla málo charakteristických rysů, podle kterých by šla od sebe oddělit a dungeony byly zřídkakdy zapamatovatelné. A tento zásadní nedostatek bohužel podědil i následovník *Daggerfallu* – o dvacet let mladší *No Man's Sky*.

1.7.2 Sadow of Mordor

Vývojáři hry *Shadow of Mordor* a později jejího pokračování *Shadow of War* vsadili na PCG při generování nepřátel. V článku *7 uses of procedural generation that all developers should study* [14] je popsáno, jak pomocí procedurálního generování vývojáři vytváří vztah mezi hráčem a jednotlivými nepřáteli – skřety – z nichž každý je unikátní, právě díky PCG.

Ve hře je procedurálně generovaný jak vzhled jednotlivých skřetů, tak jejich jméno, nebo dokonce vztah k ostatním skřetům. Skřeti jsou ve hře povýšeni, pokud se jim podaří zabít hráče. V takovém případě si skřeti hráče pamatují při jejich dalším setkání, což pomáhá právě budování vztahu mezi hráčem a nepřítelem. Pokud je naopak skřet přemožen hráčem a v boji zraněn, při příštím setkání k jeho vzhledu přibude například procedurálně generovaná jizva nebo páska přes oko.

Tento systém, pomocí kterého vývojáři budují vztah mezi hráčem, jednotlivými skřety a okolním světem nazvali samotní tvůrci *Nemesis System* – ten je dokonce patentován a je podrobněji popsán v článku *10 Things To Know About Shadow Of War's Nemesis System* [15].

1.7.3 Dungeon Alchemist

Dungeon Alchemist je nástroj, který uživateli umožňuje relativně jednoduše vytvořit libovolný dungeon s pomocí procedurálního generování. Tento nástroj popsal *J. R. Zambrano* ve svém

Obrázek 1.7 Příklad vygenerovaného dungeonu (zde je dungeonem budova) pomocí nástroje Dungeon Alchemist. Obrázek převzat přímo z oficiálního webu tohoto nástroje – *www.dungeonalchemist.com*.

článku *D&D: Map Your Dungeon With An AI – Dungeon Alchemist* [16]. Tato aplikace uživateli vygeneruje dungeon včetně vybaveného interiéru. S objekty, kterými je dungeon vybaven, může uživatel následně libovolně manipulovat, popřípadě je měnit. To, jak výsledný dungeon vypadá, je primárně ovlivněno tím, jakou tématiku si uživatel pro jednotlivé místnosti zvolí – touto tématikou může být například kobka nebo vězení. Dungeon Alchemist ovšem není schopen pracovat bez vstupu od uživatele. Uživatel musí ručně zadat například rozlohu jednotlivých místností nebo již zmíněnou tématiku. Momentální verze zároveň není schopná generovat více pater. Rozdílné je ovšem primárně zaměření tohoto generátoru – zatímco předchozí zmíněné projekty používaly procedurální generování jako určitou mechaniku, která je součástí počítačové hry, Dungeon Alchemist je zamýšlen primárně jako generátor map k již zmíněné papírové hře Dungeons & Dragons. Uživatel si tedy vygeneruje dungeon, ten si následně například vytiskne na papír a s takto vytvořeným dungeonem je připraven hrát. Příklad vytvořeného dungeonu pomocí aplikace Dungeon Alchemist je k vidění na obrázku 1.7.

Kapitola 2
Analýza

Tato kapitola se věnuje analýze jednotlivých metod a technologií, které je možné použít v praktické části práce – je zde vysvětleno, proč byly dané technologie zvoleny, a zároveň jsou popsány jejich výhody a nevýhody. Kapitola je rozdělena do dvou částí – nejprve se věnuje technologiím zaměřeným na tvorbu 3D modelů, ať už se jedná o samotné modelování, či tvorbu materiálů. Ve druhé části jsou pak následně analyzovány technologie, které je možné použít při implementaci algoritmu procedurálního generování.

2.1 Tvorba modelů

Tvorba modelů je klíčovou součástí praktické části práce. Veškeré modely, včetně materiálů, budou vytvořeny ručně, k čemuž je potřeba vybrat sadu odpovídajících nástrojů a metod. V této části jsou popsány nástroje a metody, které je možné použít při tvorbě modelů a je zde zdůvodněno, proč byly dané nástroje a metody zvoleny pro použití v praktické části práce.

2.1.1 Modelování

Modelování bude provedeno v nástroji *Blender*. Ač tomu tak dříve nebylo, po verzi *2.8* začal být Blender vnímán v širší komunitě 3D grafiků jako nástroj schopný konkurovat profesionálním komerčním nástrojům, jakými jsou například *Autodesk 3D Studio MAX*, *Autodesk Maya* nebo *Cinema 4D*.

Ač zmíněné komerční nástroje mohou v některých oblastech překonávat zvolený software, *Blender* má oproti nim jednu klíčovou výhodu – jedná se totiž o svobodný a otevřený software. Od již zmíněné verze *2.8* navíc výrazně stoupla jeho kvalita (kvalita stoupá již roky, ovšem tato verze je v očích mnoha 3D grafiků zásadním zlomem), tudíž se jedná o ideální software pro začínající 3D grafiky. Většina klíčových procesů je navíc v *Blenderu* podobná zmíněným komerčním softwarům. Člověk ovládající *Blender* by tedy měl být snadno schopen přejít na jiný software, který je v daném týmu standardem. Blender má také jako svobodný a otevřený software obrovskou komunitu, která je aktivní na nejrůznějších fórech a webech, a která svou aktivitou velmi přispívá k vývoji samotného softwaru – často se například stává, že plugin vytvořený členem této komunity je v budoucích verzích zahrnut do základního softwaru.

Další výhodou zvoleného softwaru je, že bude zároveň cílovým softwarem, pro který bude v praktické části práce vytvořen plugin, respektive sada pluginů, umožňující procedurální generování budov včetně interiérů. Nebude tedy zbytečně zvýšen počet potřebných nástrojů pro realizaci praktické části této práce.

Blender bude zároveň použit pro UV unwrapping vytvořených modelů. Tento proces, který je podrobněji popsán v první kapitole, patří mezi závěrečné kroky při vytváření jakéhokoliv modelu – tento krok je potřeba provést předtím, než začneme pro daný model vytvářet odpovídající materiály. Ač se UV unwrapping může zdát jako proces související více s texturováním, a tím pádem by někdo mohl očekávat, že se bude provádět v softwaru určenému pro texturování, je standardem, že UV unwrapping provádíme ve stejném softwaru, ve kterém modelujeme. Do texturovacích softwarů se následně importuje model s již provedeným UV unwrappingem. Blender, stejně jako komerční nástroje, navíc disponuje funkcí, která celý proces provádí automaticky, ovšem ta, ač proces značně urychlí, není dokonalá, a je mnohdy lepší provést unwrapping ručně.

2.1.2 Sculpting

Jedním z prvních, ne-li vůbec první software umožňující sculpting, byl *ZBrush*. Od té doby tento software byl, a dodnes je standardem co se týče sculptingu – jak je také řečeno v článku *What is ZBrush: How It Works & What It's Used For* [17].

Největší slabina softwaru *ZBrush* může být pro většinu uživatelů jeho uživatelské rozhraní – začátečníkům může dělat problém se v komplexním rozhraní orientovat. ZBrush je zároveň komerčním nástrojem a začínající 3D grafici mohou tedy raději sáhnout po otevřeném softwaru, jakým je již zmíněný Blender.

Blender krom modelování umožňuje i sculpting. Výhodou je zde již zmíněná otevřenost softwaru. Další výhodou může být, že v případě, kdy uživatel používá Blender pro modelování, nemusí kvůli sculptingu přecházet na jiný software. ZBrush ovšem zůstává mnohem komplexnějším nástrojem, co se týče sculptingu, a jak již bylo řečeno, jedná se o průmyslový standard.

Při realizaci praktické části práce bude při sculptingu použit opět software Blender. Ač je ZBrush mnohem komplexnější nástroj, jeho veškeré výhody by nebylo možné při projektu tohoto rozsahu využít. Vzhledem ke zvolené stylizaci modelů, která je popsána v následujícím textu a vzhledem k tomu, že většina modelů určených ke generování jsou neorganické modely, nebude sculpting při jejich vytváření klíčový. Pokud bude vůbec sculpting použit, bude využit při modelování pouze k přidání pocitu větší organičnosti modelu tam, kde se to hodí.

2.1.3 Materiály

Existuje mnoho softwarů a způsobů, jak vytvořit materiály pro vytvořené 3D modely. Ač dříve byl standardní postup ten, že textury se malovaly přímo na 2D UV mapy modelu, dnes se již texturuje pomocí malování na samotné 3D modely – to ovšem neznamená, že bychom se mohli vyhnout UV unwrappingu. Momentálně jsou v tomto průmyslu nejvíce používané dva softwary – software *Substance Painter* vlastněný společností *Adobe* a software *Mari* vlastněný společností *Foundry*. Rozdíly mezi těmito dvěma nástroji a doporučení, kdy se jaký z nich hodí více, jsou popsány v článku *Substance Painter VS Mari Which is Better?* [18].

Mari je jasnou volbou pro většinu tvůrců, kteří vytvářejí textury pro vizuální efekty filmů. Hlavním důvodem je to, že Mari je vytvořen za účelem co nejsnadnějšího vytváření materiálů a textur s velmi vysokým rozlišením. Vysoké rozlišení může být důležité například při scénách, kdy je vytvořený objekt zabírán z velké blízkosti. Tento software byl vytvořen společností *Weta Digital* aby byl použit při produkci filmů *King Kong* a *Avatar*.

Substance Painter je průmyslovým standardem co se týče vytváření materiálů a textur do počítačových her. To je hlavním důvodem, proč byl tento nástroj zvolen při realizaci praktické části této práce. Mezi hlavní výhody tohoto softwaru patří relativní jednoduchost použití a možnost dosáhnout velmi rychle vysoce kvalitních výsledků. *Substance Painter* byl použit například při tvorbě obou části videohry s názvem *The Last of Us*.

Velkou výhodou *Substance Painteru* je také schopnost vytvářet komplexní materiály relativně snadno. Sám o sobě má širokou nabídku předpřipravených materiálů, které může uživatel

Obrázek 2.1 Snímek obrazovky ze hry The Last of Us Part II – převzat z webových stránek *www.playstation.com*. Při tvorbě materiálů pro tuto hru byl použit software *Substance Painter*.

libovolně použít. Zároveň umožňuje použití takzvaných *smart materiálů*, což jsou v podstatě parametrizovatelné materiály – uživatel může jednoduše nastavit například poničení omítky u zdi. Při vytváření vlastních materiálů lze ještě *Substance Painter* rozšířit o další software s názvem *Substance Designer*, který umožňuje právě navrhování zcela nových materiálů.

2.1.4 Realistické vs. stylizované modely

To, jak modely ve hře vypadají a jak celková hra nakonec působí můžeme rozdělit do dvou kategorií – hry a jejich světy mohou být buď realistické nebo stylizované. Rozdíl mezi realistickými a stylizovanými modely, a mezi technikami použitými pro jeden nebo druhý přístup popsala *Kim Aava* ve svém článku *Realistic vs. Stylized: Technique Overview* [19].

U realistických modelů se snaží jejich tvůrci co nejlépe napodobit reálný život – model nemusí reprezentovat něco, co nutně existuje v reálném světě, ale musí to vypadat tak, že by to součástí našeho světa být mohlo. Hry, které jsme ovšem brali jako realistické před deseti lety nám dnes již moc realistické nepřipadají. Některé takové hry na nás dnes mohou působit dokonce stylizovaně. To můžeme dát samozřejmě za vinu technologickým limitům v minulosti. Autorka článku tento fakt demonstruje na videoherní sérii *Uncharted* (ukázka na obrázku 2.2). Tvář hlavního hrdiny této série v prvním díle postrádala spoustu detailů – tyto detaily si již dnes mohou tvůrci díky postupujícím technologiím dovolit a tak vidíme u nejnovějšího dílu velké zlepšení. Ač byl v roce 2007 první díl *Uncharted* brán jako realisticky vypadající hra, dnes tak působit nemusí.

Při tvorbě stylizovaných světů nehledíme na to, jak reálně modely vypadají. Tvůrce těchto modelů si může libovolně hrát s barvami a tvary, a docílit tak zajímavých výsledků. Ač dříve bylo u stylizovaných světů standardem nízké rozlišení modelů, a u textur používání pouze barevné složky, dnes narazíme i na stylizované světy s vysokým rozlišením, využívající komplikovanější materiály. Ač mohou obě metody působit velmi podobně, díky nulovým restrikcím u tvorby stylizovaných modelů se výsledky výrazně liší. Stylizované modely mohou mít zpravidla méně detailů nebo například jednodušší tvary oproti modelům realistickým. Tento styl můžeme znát například z animovaných filmů. Jak mohou stylizované světy vypadat je vidět na obrázku 2.3.

Obrázek 2.2 Posun v realističnosti u videoherní série *Uncharted* – snímek z článku *Realistic vs. Stylized: Technique Overview* [19].

Autorka článku rozděluje stylizované modely do dvou dalších kategorií – těmito kategoriemi jsou přehnaná stylizace a minimalistická stylizace. Tyto dvě kategorie jsou podrobněji popsány v následující části.

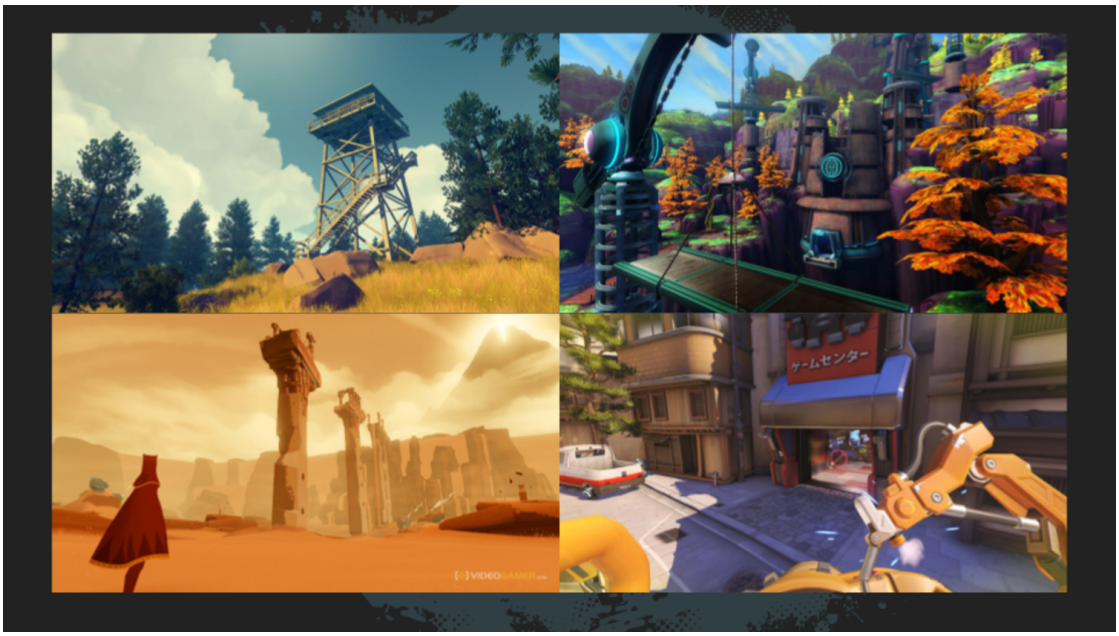
Při přehnané stylizaci se tvůrce zaměřuje převážně na výraznější detaily a tvary objektu který modeluje. Drobné detaily, které by bylo nutné do modelu zahrnout, pokud by měl být výsledek realistický, můžeme při tvorbě modelu metodou přehnané stylizace zanedbat. Detaily, které jsou pro objekt esenciální naopak ještě zdůrazníme. Při tvorbě takových modelů je třeba brát v potaz důležitost daného detailu pro objekt jako celek – je onen detail potřeba, aby pozorovatel například pochopil o jaký objekt se jedná? Pokud je odpověď na tuto otázku záporná, detail může tvůrce vynechat.

Minimalistická stylizace je, jak už z názvu vyplývá, založena na jednoduchosti. Pro tuto stylizaci se často používají jednoduché tvary a barvy - při tvorbě materiálů zpravidla používáme pouze barevnou složku a vynecháváme například normal mapy. Tvůrce modelu metodou minimalistické stylizace zanedbává většinu detailů a modely ponechává co nejjednodušší.

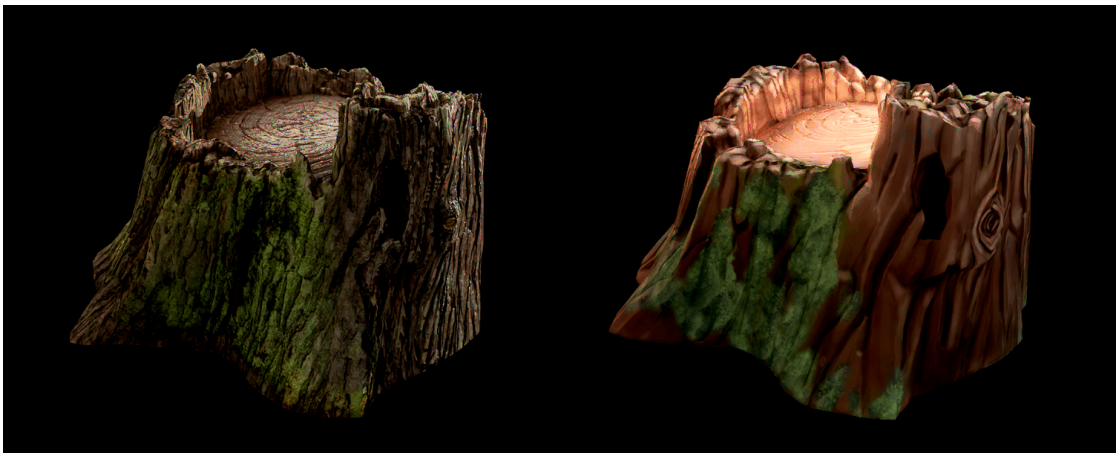
V praktické části této práce budou vytvořeny modely metodou minimalistické stylizace. Výhodou této metody je, že nechává svému tvůrci volnou ruku. Zároveň je vytváření modelů touto metodou nejjednodušší – tvorba realistických modelů by drasticky zvýšila časovou náročnost praktické části této práce. Výsledný plugin bude ovšem fungovat tak, aby mohly být jednotlivé modely jednoduše zaměnitelné, a tudíž mohl uživatel zvolený styl modelů snadno změnit.

2.2 Implementace algoritmu PCG

V této části jsou popsány zvolené technologie a metody, které budou použity při implementaci generátoru v praktické části této práce. Podrobněji se návrhu samotné implementace věnuje následující kapitola.



Obrázek 2.3 Příklad stylizovaných světů – snímek z článku *Realistic vs. Stylized: Technique Overview* [19]. Vlevo jsou příklady minimalistické stylizace, zatímco vpravo jsou ukázky světů vytvořených metodou přehnané stylizace.



Obrázek 2.4 Porovnání realistického a stylizovaného modelu – snímek z článku *Realistic vs. Stylized: Technique Overview* [19].

2.2.1 Použité technologie

Existuje několik různých způsobů, jak přistoupit k samotné implementaci algoritmu PCG. V praktické části práce bude algoritmus implementován jako plugin do softwaru Blender – jak je popsáno v zadání. Další možností by například bylo algoritmus implementovat přímo v rámci jednoho z videoherních enginů, jako je například Unity. Oba přístupy mají své výhody i nevýhody.

Vytvoření pluginu do softwaru Blender umožní uživateli ihned po vygenerování snadno manipulovat s vygenerovaným modelem – ať už na úrovni objektů nebo například jednotlivých vrcholů. Tento přístup se nejvíce hodí pro procedurální generování za účelem návrhu obsahu. Vygenerované modely je ovšem následně nutné exportovat z Blenderu a importovat je do videoherního enginu. Implementace PCG v rámci videoherního enginu znamená, že se uživatel nemusí zabývat exportem a importem vytvořených modelů. Nevýhodou zde ovšem je náročnější manipulace s objekty – například na úrovni jednotlivých vrcholů – nebo manipulace s materiály. Implementace PCG v rámci herního enginu dává největší smysl, pokud chceme aby se jednalo přímo o mechaniku dané hry – tedy aby se nám například generoval svět v reálném čase, během hry.

Algoritmus bude konkrétně implementován jako zásuvný modul do softwaru Blender verze *3.1.2*. Pluginy se do tohoto softwaru implementují v jazyce *Python* verze *3.9.7*. Důležité je, aby byl výsledný nástroj pro generování budov vytvořen tak, že půjde snadno rozšířit o další funkcionality a že bude snadno zapojen do komplexnějších nástrojů generujících celá města.

Další možností, jak procedurálně vytvářet (nejedná se přímo o generování) budovy v softwaru Blender může být použití nástroje zvaného *geometry nodes*. Ač tento nástroj nebude při implementaci budoucího generátoru využit, rozhodně se jedná o zajímavou alternativu, jak vytvářet budovy a jiné objekty s určitým prvkem náhodnosti, a stojí za to ji zmínit. Krásný příklad, čeho je tento nástroj schopen, zveřejnil autor s přezdívkou *sozap* na webu *www.blenderartists.org* [20], a jeden z výsledků je na obrázku 2.5. Jak sám autor v příspěvku zmiňuje, jedná se o přehnané použití *geometry nodes* a tento projekt dělal primárně za účelem studia tohoto nástroje. *Geometry nodes* se v praxi používají v kombinaci s klasickým modelováním – typickým příkladem je například náhodné rozprostření kamenů po zemi.

2.2.2 Použitá metoda generování

K samotnému generování bude použita jedna z *metod konstruktivního generování*. Jak vyplývá z popisu těchto metod v první kapitole, pro generování budov a jejich místností, včetně vybavení, se nejvíce hodí *metoda dělení prostoru*. Tato metoda nám umožní vygenerovat uspořádané místnosti s rozumně a smysluplně rozloženým nábytkem. *Metoda agentem řízeného růstu* se hodí spíše pro generování organických dungeonů, jakými mohou být nejčastěji jeskyně. Podrobnější popis budoucího algoritmu se nachází v následující kapitole.

Obrázek 2.5 Příklad využití *geometry nodes* při tvorbě budov – snímek z příspěvku na webové stránce www.blenderartists.org [20].

Kapitola 3
Návrh

Tato kapitola se věnuje návrhu budoucího pluginu. Nejprve je popsán smysl a cíl samotného pluginu – pro jaké uživatele je tento plugin určen a jaké má být jeho využití. Poté je navržena struktura generovaných budov – konkrétně typy jednotlivých místností, které bude plugin generovat. Následně je vytvořen návrh samotné funkčnosti – jakým způsobem se budovy budou generovat a jaké metody budou při generování použity. Nakonec je navrženo jednoduché uživatelské rozhraní odpovídající požadované funkčnosti.

3.1 Využití pluginu

V předchozích kapitolách byl popsán proces a možné použití procedurálního generování. Zároveň v nich bylo vysvětleno, k jakým účelům se procedurální generování hodí a k jakým nikoliv. Výsledný plugin bude spadat do kategorie nástrojů sloužících při *návrhu obsahu*, který byl popsán v první kapitole (1.4.1). Výsledný plugin má tedy za cíl co nejvíce automatizovat vytváření modelů budov, které budou následně použity při tvorbě videoherních světů, ovšem zároveň počítá se zásahem uživatele do výsledných modelů – například ruční přesunutí určitých modelů na místo, které je podle uživatele vhodnější.

Nástroje sloužící k návrhu obsahu mají několik zásadních výhod, které se odráží na jejich vývoji. Na rozdíl od nástrojů, které generují videoherní světy v reálném čase během hraní, nástroj sloužící pro návrh obsahu nemusí generovat modely v řádu desetin vteřin a běh samotného generování může být tedy delší – i když stále v rozumných mezích. Zároveň lze plugin implementovat více náhodný za cenu toho, že občas vygeneruje nesmysl, který musí uživatel ručně opravit. Náhodnost ovšem nesmí být ani u návrhu obsahu přehnaná, aby ruční zásah byl opravdu spíše výjimkou než pravidlem.

3.2 Struktura budov

Budova, kterou bude plugin generovat, se bude skládat z několika typů místností. Každá místnost bude charakterizována modely, které se do dané místnosti budou generovat, aby byla odlišitelná od ostatních typů. V této části je popsána struktura těchto typů místností a to, jaké modely se do daných typů místností budou generovat.

3.2.1 Místnosti

Jak již bylo zmíněno, vygenerovaná budova bude rozdělena do několika místností. Tyto místnosti mohou být různého typu. Plugin bude schopen generovat následující místnosti:

Typ místnosti	Povinné objekty	Možné objekty
Ložnice	Postel	Skříň
Kuchyň	Krb, kuchyňský stůl	
Chodba		Skříň
Společenská místnost	Stůl se stoličkami	Skříň

Tabulka 3.1 Tabulka vztahů mezi jednotlivými objekty a místnostmi

1. *Společenská místnost*
2. *Ložnice*
3. *Kuchyň*
4. *Chodba*

3.2.2 Objekty

Plugin má za úkol generovat nejen samotnou budovu, ale i vyplnit její interiér nejrůznějšími objekty. U některých předmětů existují omezení, ve kterých místnostech se mohou nacházet – například nedává smysl, aby se na chodbě generovala postel. Naopak by bylo zvláštní, kdyby postel chyběla v ložnici. Plugin bude schopen generovat následující objekty:

1. *Postel*
2. *Stůl se stoličkami*
3. *Skříň*
4. *Krb*
5. *Kuchyňský stůl (obdoba dnešní kuchyňské linky)*

Rozdělení navržených objektů podle místností, ve kterých se mohou nebo musí nacházet, je rozepsáno v tabulce 3.1.

3.3 Funkčnost pluginu

Plugin bude fungovat ve třech základních krocích. Prvním krokem je vytvoření půdorysu budovy. V druhém kroku se instance objektu reprezentující jednotlivá patra rozdělí na místnosti. Ve třetím a posledním kroku se importují jednotlivé modely, které dohromady dávají celou budovu – zároveň se do jednotlivých místností vygenerují veškeré objekty – skříně, postele, stoličky, stoly a další. Tyto tři kroky jsou v této části podrobněji popsány. Poté jsou popsána jednotlivá pravidla, omezující samotné generování a zabraňující, aby plugin generoval nesmyslnou změť pokojů a objektů.

3.3.1 Logika generování

Vytvoření půdorysu

Jak již bylo zmíněno, prvním krokem je generování exteriéru budovy. V tomto kroku jsou pomocí pseudonáhodného algoritmu poskládány předem vytvořené modely zdí tak, aby dohromady dávaly vždy celé patro. Uživatel má možnost ovlivnit počet pater. Před tím, než se mohou začít generovat jednotlivá patra se generuje půdorys budoucí budovy. Algoritmus pracuje se zdmi

o délce 5 metrů. Algoritmus na sebe tyto zdi rozumně navazuje, dokud nevytvoří použitelný půdorys budovy. Při vytváření půdorysu ovšem nesmí překročit žádný z rozměrů délku 20 metrů (budova nesmí být širší nebo hlubší než 20 metrů). Podrobněji tento postup popisuje algoritmus 2. Algoritmus zároveň zaznamenává okrajové souřadnice rektangulární sítě, do které vygenerovaný půdorys spadá – tato síť je později použita při rozdělení půdorysu na stejně velké čtvercové plochy, neboli segmenty.

Algoritmus 2 Algoritmus vytvoření půdorysu budovy

1: Vytvoř zatím prázdný seznam zdí.
2: Alokuj čtyři souřadnice, které reprezentují rohové souřadnice rektangulární sítě, do které vygenerovaný půdorys spadá. Všechny čtyři souřadnice nastav na hodnoty (0, 0).
3: Vytvoř první zeď, která bude počáteční zdí při vytváření půdorysu budovy. Zeď začíná na souřadnicích (0, 0) a končí na souřadnicích (5, 0). Veškeré souřadnice, které zeď protne, a jejichž obě hodnoty jsou násobkem pěti, přidej do seznamu pokrytých souřadnic.
4: Vytvoř novou zeď tak, že vybereš směr, ve kterém novou zeď navážeš na předchozí zeď. Směr určuje, zda budova roste do šířky nebo do hloubky. Možné směry jsou omezeny aktuálním stavem půdorysu, aby nedocházelo k zacyklení nebo vytváření nesmyslných tvarů.
5: **if** Po navázání na předchozí zeď by byl překročen limit 20 metrů u šířky nebo hloubky. **then**
6: Vrať se na krok 4.
7: **else if** Po navázání na předchozí zeď by byl uzavřen půdorys – konec nové zdi by byl na souřadnici (0, 0) **then**
8: Navaž vygenerovanou zeď na zeď předchozí.
9: Veškeré souřadnice, které zeď protne, a jejichž obě hodnoty jsou násobkem pěti, přidej do seznamu pokrytých souřadnic – ten je součástí každé zdi.
10: Půdorys byl úspěšně uzavřen. Algoritmus končí.
11: **else**
12: Navaž vygenerovanou zeď na zeď předchozí.
13: Veškeré souřadnice, které zeď protne, a jejichž obě hodnoty jsou násobkem pěti, přidej do seznamu pokrytých souřadnic – ten je součástí každé zdi.
14: Pokud vygenerovaná zeď přesahuje ven ze sítě, reprezentované čtyřmi souřadnicemi vytvořenými na začátku algoritmu, aktualizuj tyto souřadnice tak, aby do vzniklé sítě patřila i nová zeď.
15: Vrať se na krok 4.
16: **end if**

Po vytvoření půdorysu se ještě testuje, zda vygenerovaný půdorys splňuje požadavky dané budovy – pokud ne, generujeme půdorys znovu. Tyto požadavky jsou podrobněji popsány v sekci *Návrh pravidel*. Pokud půdorys požadavky budovy splňuje, tento krok končí.

Rozdělení pater na místnosti

V druhém kroku, který rozděluje vygenerovaná patra na jednotlivé místnosti, je potřeba provést několik dílčích činností. Kromě samotného rozdělení patra na místnosti a vytvoření zdí, oddělujících tyto místnosti, je potřeba vytvořit v jednotlivých patrech podlahy a zároveň schodiště, vedoucí z jednoho patra do druhého (pokud má budova více pater). Umístění schodiště samozřejmě ovlivňuje jak rozložení místností v patře, kde schodiště začíná, tak i v patře, kde schodiště končí. Z toho důvodu je potřeba brát na schodiště ohled při rozdělování místností – nechceme například, aby nám schodiště vedlo přímo do ložnice. Místnosti, ve kterých se může nacházet schodiště jsou chodba a společenská místnost. V ložnici a kuchyni schodiště nesmí ani začínat, ani končit.

Algoritmus pro rozdělení pater na místnosti nejprve rozdělí půdorys na jednotlivé segmenty o šířce 5 metrů a hloubce 5 metrů. K tomuto rozdělení slouží rektangulární síť segmentů, kterou

nám reprezentují souřadnice alokované při generování půdorysu. Postup při rozdělování půdorysu na jednotlivé segmenty je popsán algoritmem 3.

Algoritmus 3 Algoritmus rozdělení půdorysu na jednotlivé segmenty

```
Vytvoř zatím prázdný seznam segmentů patřících k půdorysu.
Pro každý řádek rektangulární sítě projdi jednotlivé segmenty následovně.
Vyber první segment v řádku a nastav jeho stav na neznámý.
if Stav zvoleného segmentu je neznámý. then
    if Levá hrana vybraného segmentu je na našem půdorysu překryta zdí. then
        Přidej segment do seznamu segmentů patřících k půdorysu.
        if Pravá hrana vybraného segmentu je na našem půdorysu překryta zdí. then
            Stav následujícího segmentu bude udávat, že nepatří k půdorysu.
            Přejdi na další segment a pokračuj krokem 4.
        else
            Stav následujícího segmentu bude udávat, že patří k půdorysu.
            Přejdi na další segment a pokračuj krokem 4.
        end if
    else
        if Pravá hrana vybraného segmentu je na našem půdorysu překryta zdí. then
            Stav následujícího segmentu bude udávat, že patří k půdorysu.
            Přejdi na další segment a pokračuj krokem 4.
        else
            Stav následujícího segmentu bude udávat, že nepatří k půdorysu.
            Přejdi na další segment a pokračuj krokem 4.
        end if
    end if
else if Stav zvoleného segmentu udává že je součástí půdorysu. then
    Přidej segment do seznamu segmentů patřících k půdorysu.
    if Pravá hrana vybraného segmentu je na našem půdorysu překryta zdí. then
        Stav následujícího segmentu bude udávat, že nepatří k půdorysu.
    else
        Stav následujícího segmentu bude udávat, že patří k půdorysu.
        Přejdi na další segment a pokračuj krokem 4.
    end if
else if Stav zvoleného segmentu udává že není součástí půdorysu. then
    if Pravá hrana vybraného segmentu je na našem půdorysu překryta zdí. then
        Stav následujícího segmentu bude udávat, že patří k půdorysu.
        Přejdi na další segment a pokračuj krokem 4.
    else
        Stav následujícího segmentu bude udávat, že nepatří k půdorysu.
        Přejdi na další segment a pokračuj krokem 4.
    end if
end if
```

Po rozdělení půdorysu na segmenty následuje samotné rozdělení na místnosti. Algoritmus po každém rozdělení zadané plochy na polovinu kontroluje, zda je aktuální plocha vhodná pro danou místnost nebo zda je potřeba provést další dělení. Algoritmus samotného rozdělení půdorysu na jednotlivé místnosti je popsán algoritmem 4. Tento algoritmus je založen na *metodě dělení prostoru* s využitím BSP stromu. Tato metoda byla více popsána v první kapitole (1.6.2).

V rámci algoritmu rozdělení půdorysu na místnosti (4) je potřeba umět provést dva kroky. Nejprve potřebujeme zjistit, zda nám nově vzniklá zeď vůbec spadá do půdorysu – tento proces je popsán algoritmem 5. Poté ověřujeme, zda nově vzniklá zeď neprotíná nějakou již existující

Algoritmus 4 Algoritmus rozdělení půdorysu na místnosti

1: Vytvoř kořen BSP stromu – vlož do něj všechny zdi a všechny segmenty, které patří k půdorysu. Tento kořen vyber jako první zpracovávaný vrchol.
2: **if** Právě zpracovávaná oblast již nepotřebuje dále dělit. **then**
3: Algoritmus končí.
4: **end if**
5: Pseudonáhodně vyber jeden ze souřadnicových bodů, který patří k jedné ze zdí právě zpracovávaného vrcholu BSP stromu.
6: Pseudonáhodně vyber druhý bod, který má stejnou x nebo y souřadnici, jako první zvolený bod, a patří k jedné ze zdí právě zpracovávanému vrcholu.
7: **if** Zeď protínající oba body patří do půdorysu a zároveň by vzniklá zeď neprotínala již existující zeď. **then**
8: Vytvoř zeď.
9: Do jednoho potomka právě zpracovávaného vrcholu stromu přidej veškeré zdi a segmenty na jedné straně od nově vytvořené zdi, do druhého potomka přidej veškeré zdi a segmenty na druhé straně od nově vytvořené zdi.
10: Pro oba nově vytvořené potomky opakuj algoritmus od kroku 2.
11: **else**
12: Vrať se na krok 2 a opakuj algoritmus.
13: **end if**

zeď. To je popsáno algoritmem 6.

Algoritmus 5 Algoritmus ověření, zda nová zeď spadá do půdorysu budovy.

1: Vytvoř hranu (budoucí zeď) mezi zadaným bodem a a bodem b. Hrana je reprezentována všemi protnutými souřadnicemi, jejichž x a y souřadnice jsou násobkem pěti.
2: Pro každý bod náležící budoucí zdi ověř, že patří k nějakému segmentu půdorysu.
3: **if** Všechny dvojice sousedních bodů spadají do půdorysu budovy **then**
4: Nová zeď spadá do půdorysu budovy. Algoritmus končí.
5: **else**
6: Nová zeď nespadá do půdorysu budovy. Algoritmus končí.
7: **end if**

Listy vzniklého BSP stromu nám reprezentují jednotlivé místnosti. Po vytvoření tohoto stromu je tedy závěrem každému listu přiřazen typ místnosti.

Generování interiéru

Závěrečným krokem je generování interiéru. Předměty jsou pseudonáhodně rozmístěny do místnosti dle daných pravidel – například zda se daný předmět musí nacházet u zdi nebo naopak uprostřed segmentu. Metoda, která rozmisťuje předměty do místností dostane zadaný počet objektů. Následně pro každý segment dané místnosti rozhoduje, zda se v něm bude nebo nebude nacházet daný předmět. Pokud metoda rozhodne, že se v daném segmentu předmět nacházet bude, zvolí jeho umístění v tomto segmentu dle zadaných pravidel.

3.3.2 Návrh pravidel

To, jakým způsobem se do budov budou generovat místnosti, a do těchto místností jednotlivé předměty, bude omezeno sadou pravidel. Tato pravidla budou určovat například kolikrát se může daný typ místnosti vyskytovat v budově. Jako nejzákladnější pravidla můžeme chápat vztahy

Algoritmus 6 Algoritmus ověření, zda nová zeď neprotíná již existující zeď.

1: Vytvoř hranu (budoucí zeď) mezi zadaným bodem a a bodem b. Hrana je reprezentována všemi protnutými souřadnicemi, jejichž x a y souřadnice jsou násobkem pěti.
2: Pro každý bod na budoucí zdi (vyjma zadaných bodů a a b) zkoumej, zda již nepatří k existující zdi.
3: **if** Žádný ze zkoumaných bodů není součástí existující zdi. **then**
4: Nová zeď neprotíná žádnou existující zeď. Algoritmus končí.
5: **else**
6: Nová zeď protíná existující zeď. Algoritmus končí.
7: **end if**

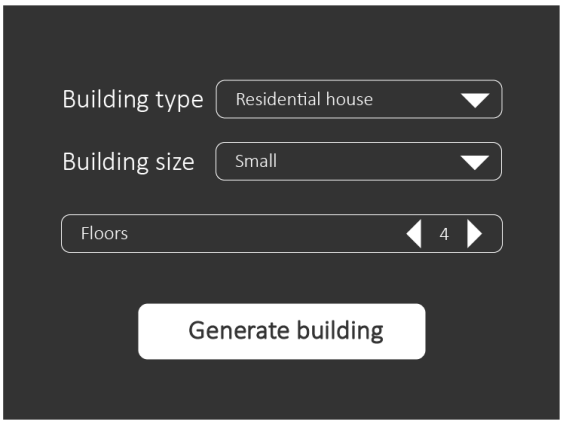
mezi jednotlivými typy místností a mezi objekty, které byly dříve popsány. Následující pravidla tato omezení ještě rozšiřují, aby vygenerované budovy měly rozumnou strukturu. Jednotlivá pravidla znějí následovně:

Pravidla omezující generování

1. V budově se nesmí nacházet více než jedna místnost typu *kuchyň* na patře.
2. Postel se musí vždy nacházet u zdi.
3. Skříň se musí vždy nacházet u zdi, otočená dveřmi do prostoru.
4. Krb se musí vždy nacházet u zdi.
5. Stůl se stoličkami ve společenské místnosti se vždy nachází uprostřed segmentu.
6. Stůl v kuchyni se vždy nachází uprostřed segmentu.
7. Schodiště může začínat a ústit pouze v místnostech typu *chodba* a *společenská místnost*.

3.4 Návrh uživatelského rozhraní pluginu

Vzhledem k tomu, že výsledný plugin je zamýšlen tak, aby mohl být v budoucnu rozšířen v generátor celých měst, není potřeba vytvářet komplikované uživatelské rozhraní. Z toho důvodu bude mít uživatel při používání pluginu pouze tři možnosti, jak vzhled výsledné budovy ovlivnit. Uživatel může zvolit typ budovy, počet pater a velikost budovy. Ač vytvořený plugin bude generovat pouze jediný typ budovy, kterým bude *obytný dům*, tato položka slouží k budoucímu rozšíření o další typy, jakými mohou být například hostince nebo obchody. Uživatel bude dále schopen nastavit, zda má mít budova jedno až čtyři patra a zda má být její rozloha malá, střední nebo velká. Velikost rozlohy chápeme jako počet pokrytých segmentů. Návrh uživatelského rozhraní je vidět na obrázku 3.1.



Obrázek 3.1 Návrh uživatelského rozhraní

Kapitola 4
Realizace

Tato kapitola se skládá ze dvou částí. První část se věnuje vytvoření modelů, které jsou následně použity v samotném pluginu. V druhé části je popsána implementace pluginu. Závěrem jsou předvedeny příklady použití vytvořeného generátoru.

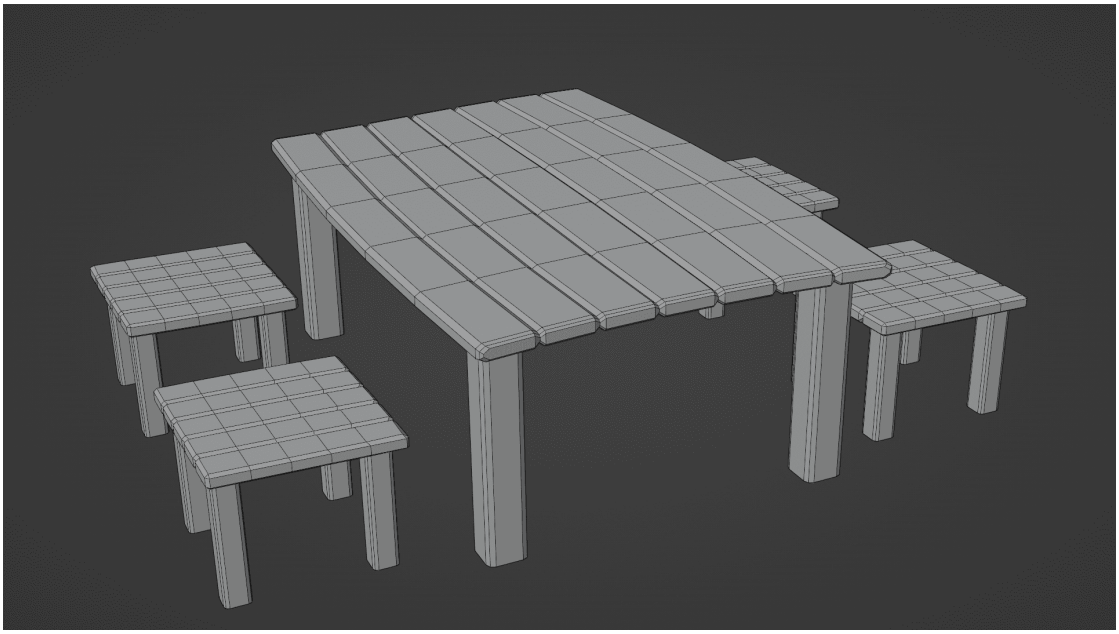
4.1 Tvorba modelů

Jednotlivé 3D modely jsou klíčovou částí pro funkčnost samotného pluginu. Všechny modely jsou vytvořeny ručně, včetně jednotlivých textur. Samotné modelování a UV unwrapping proběhlo v softwaru Blender verze 3.1.2. Materiály jsou vytvořeny pomocí softwaru Substance Painter.

Veškeré modely byly vytvořeny stejným postupem. Modely byly nejprve vymodelovány a následně na nich byl proveden UV unwrapping – tato část procesu se odehrávala v softwaru Blender. Poté byly vytvořené modely exportovány ve formátu *fbx* a importovány do softwaru Substance Painter. V dalším kroku byly pro model vytvořeny textury – při vytváření textur byla použita pouze diffuse (albedo) mapa a ta byla po jejím vytvoření vyexportována ve formátu *png*. Ve finále byla vytvořená textura přiřazena k danému objektu opět v softwaru Blender a takto vytvořený objekt včetně textury byl vyexportován ve formátu *fbx* – jak objekt, tak textura se nachází v jediném souboru. Tyto soubory mohou být následně použity vytvořeným pluginem při generování samotných budov. V následujících částech jsou podrobněji popsány jednotlivé kroky při tvorbě takového modelu na příkladu modelu stolu se stoličkami.

4.1.1 Modelování

Při vytváření modelů byl kladen důraz na nízké rozlišení – výsledkem měly být jednoduché, stylizované modely. Zároveň jsou veškeré modely vytvořeny tak, aby se skládaly pouze z ploch o čtyřech vrcholech. To je důležité z toho důvodu, aby byly veškeré plochy snadno a jednoznačně rozdělitelné na trojúhelníky, které poté vykreslují videoherní enginy. Toto rozdělení probíhá automaticky, zpravidla při exportu modelu z modelovacího nástroje, kterým je v našem případě Blender. Dalším možným přístupem je veškeré plochy vytvořeného modelu rozdělit na trojúhelníky ručně – tím můžeme docílit nižšího rozlišení a přitom stejného množství detailů na modelu. Vzhledem k tomu, že jsou výsledné modely velmi jednoduché by ovšem mezi rozlišením automaticky rozdělených ploch a ručně rozdělených ploch nebyl žádný výrazný rozdíl a proto byl zvolen zmíněný přístup. Model stolu se stoličkami, na kterém bude reprezentován produkční postup tvorby kompletního modelu je k vidění na obrázku 4.1.



Obrázek 4.1 Model stolu se stoličkami

4.1.2 UV unwrapping

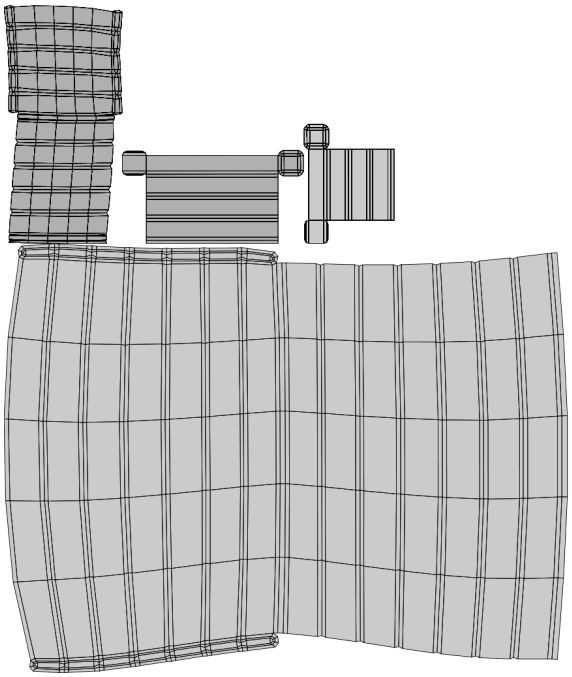
Ač existují nástroje, které automatizují proces UV unwrappingu, u veškerých vytvořených modelů byl UV unwrapping proveden ručně. Důvod je ten, že při UV unwrappingu chceme, aby výsledná UV reprezentace daného modelu měla co nejméně švů – čím více máme na modelu, respektive jeho UV mapě, švů, tím náročnější může být tvorba materiálů – chceme totiž docílit toho, aby švy nebyly na výsledném modelu vidět. Z toho důvodu se jich tedy snažíme vytvářet co nejméně a zároveň vytváříme švy tam, kde budou pravděpodobně při budoucím použití modelu nejméně vidět. Například u stolní desky to může být její spodní část. UV mapa již zmíněného stolu se stoličkami se nachází na obrázku 4.2.

4.1.3 Tvorba materiálů

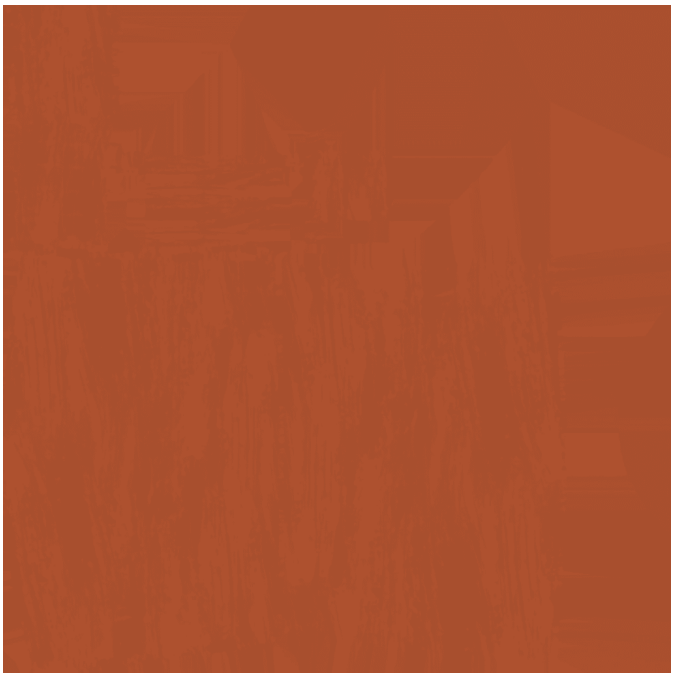
Po vytvoření modelu a provedení UV unwrappingu se výsledný model exportuje ve formátu *fbx* a výsledný soubor se importuje do softwaru Substance Painter. Tento software slouží k vytváření materiálů a jedná se o nástroj pro 3D malování – při vytváření materiálů tedy nemusíme kreslit na UV mapu modelu, ale můžeme kreslit přímo na model. Tento software zároveň disponuje celou řadou již vytvořených materiálů, které je možné použít jako základ při tvorbě materiálu pro daný model. Nabídku dostupných materiálů lze i rozšířit buď vytvářením vlastních materiálů nebo získáním těchto materiálů od třetích stran. Při vytváření materiálů byla využita pouze barevná složka, neboli albedo mapa – tento postup je běžný právě při vytváření stylizovaných modelů. Výsledná textura (respektive albedo mapa) pro model stolu se stoličkami je k vidění na obrázku 4.3. Vytvořená textura je po jejím dokončení exportována ve formátu *png*.

4.1.4 Výsledné modely

Finálním krokem je přiřazení vytvořené textury k danému modelu. Po jejím přiřazení lze exportovat celý model, včetně textury, do jediného souboru ve formátu *fbx* – výhodou je, že plugin



Obrázek 4.2 UV mapa modelu stolu se stoličkami



Obrázek 4.3 Albedo mapa modelu stolu se stoličkami

Název modelu	Počet trojúhelníků	Počet objektů	Popis
Podlaha	340	1	
Sloup	28	1	
Schody	1116	1	
Vnitřní zeď bez dveří	340	1	
Vnitřní zeď s dveřmi	852	2	Zeď, dveře
Venkovní zeď s dvěma okny	940	5	Zeď, čtyři okenice
Venkovní zeď s jedním oknem	574	3	Zeď, dvě okenice
Venkovní zeď bez oken	212	1	Zeď
Venkovní zeď s dveřmi	1416	3	Zeď, dvojité dveře
Postel	936	1	
Skříň	1004	1	
Krb	1100	1	
Stůl s židlemi	4556	5	Stůl, čtyři židle
Kuchyňský stůl	2164	1	

Tabulka 4.1 Tabulka vytvořených modelů

generující budovy pak pracuje pro každý model pouze s jediným souborem a nemusí řešit přiřazování textur. Výsledný render modelu s přiřazenou texturou se nachází na obrázku 4.4. Kompletní seznam vytvořených modelů, včetně počtu trojúhelníků, případně počtu objektů, pokud se model skládá z více objektů, je k vidění v tabulce 4.1.

4.2 Implementace pluginu

Plugin je vytvořen v jazyce Python verze 3.9.2 a je určen pro software Blender – primárně verze 3.1.2, ale měl by být zpětně kompatibilní až do verze 2.8. Plugin pracuje s modely, jejichž tvorba byla popsána dříve. Tyto modely jsou pomocí navržené logiky skládány smysluplně do prostoru tak, aby vytvořily model budovy včetně vybaveného interiéru. V této části je popsána implementace klíčových algoritmů, které provádějí samotné generování budovy. Zároveň je popsána implementace jednoduchého uživatelského rozhraní navrženého v předchozí kapitole.

4.2.1 Implementace procedurálního generování

V této části je popsána implementace logiky procedurálního generování budov včetně jejich interiérů. Je zde popsána implementace klíčových kroků potřebných k generování – návrh těchto kroků a souvisejících algoritmů se nachází v předchozí kapitole.

Celé generování je spouštěno řídící třídou *ProceduralGeneration*, která je potomkem třídy *bpy.types.Operator* – ta je součástí rozhraní pro Blender v jazyce Python a zajišťuje nám chod celého pluginu. V rámci této třídy, konkrétně její metody *execute*, je vytvořena instance třídy *Building* (respektive potomka této třídy, který reprezentuje daný typ budovy), v rámci které se odehrává samotné generování. Generování má na starosti metoda *generate* již zmíněné třídy *Building*. Jakmile je vygenerována struktura budovy, reprezentována několika instancemi různých tříd v jazyce Python, následuje samotný import jednotlivých modelů, který má na starosti metoda *spawn*. Veškerý zdrojový kód vytvořeného pluginu je obsažen v jednom souboru – ač tím zásadně utrpěla přehlednost samotného kódu, je takto vytvářený plugin mnohem jednodušší testovat v Blenderu během vývoje. Při tvorbě pluginu byly dodrženy zásady OOP (objektově orientovaného programování), díky nimž by měl být člověk schopen zasáhnout do specifické funkčnosti pluginu (například generování ložnice), aniž by musel rozumět ostatním funkčnostem pluginu (například vytváření půdorysu). Plugin je tedy vytvořen tak, aby byl v budoucnu snadno rozšiřitelný nebo aby bylo snadno modifikovatelné samotné generování.

Obrázek 4.4 Finální render modelu stolu se stoličkami

Generování půdorysu budovy

Krokem, kterým začíná celkové generování budovy je vytvoření jejího půdorysu. Podrobný popis toho, jak vygenerování půdorysu probíhá se nachází v předchozí kapitole (3.3.1). Generování půdorysu se odehrává v metodě *generateFloorPlan* třídy *Building* – tato metoda se nachází ve výpisu kódu 4.1. První zeď půdorysu začíná na souřadnicích (0, 0) – následně jsou k půdorysu přidávány náhodně generované zdi, které odpovídají několika požadavkům. Náhodnost navazující zdi musí být omezena různými pravidly z toho důvodu, aby nedocházelo k generování nesmyslných, překrývajících se zdí nebo například k zacyklení programu. Tuto validaci zajišťuje metoda *validateWall*. Generování půdorysu končí, jakmile vygenerovaná zeď končí na počátečních souřadnicích (0, 0). Poté, co je půdorys vygenerován, je validován – kontroluje se, že pokrývá odpovídající počet segmentů a pokud tomu tak není, je generován celý znovu.

Rozdělení patra na místnosti

Každé patro je individuálně rozděleno na místnosti a výsledným místnostem je poté přiřazen jejich typ. Od počátku generování je napříč celým pluginem předávána informace, v jakém segmentu se nachází schodiště. Tato informace slouží k tomu, aby se místnosti, ve které se schodiště nachází, přiřadil správný typ. Zároveň je v pluginu zakázáno generovat do segmentu se schodištěm jakékoliv objekty.

Rozdělení patra na místnosti funguje na základě algoritmu dělení prostoru pomocí BSP stromu, který byl popsán v první kapitole (1.6.2). Navržený algoritmus vycházející ze zmíněného algoritmu je podrobně popsán v předchozí kapitole (3.3.1). Algoritmus začíná tak, že vezme plochu celého patra a z ní vytvoří kořen binárního stromu. Následně dělí rekurzivně zadaný prostor vždy na dvě části a ty přiřazuje jako levého a pravého potomka právě procházeného vrcholu. Rekurze se zastaví ve chvíli, kdy je velikost dané místnosti menší než maximální možná velikost daného typu místnosti.

Vrcholy binárního stromu nám reprezentuje jednoduchá třída *Node*, jejíž potomkem je třída *FloorArea*, rozšiřující základní rozhraní binárního stromu o metody specifické právě pro dané použití. Ve výpisu kódu 4.2 se nachází úryvek kódu, který zajišťuje logiku dělení zadaného prostoru

Výpis kódu 4.1 Metoda zajišťující generování půdorysu

```
def generateFloorPlan(self):
    self._floorPlan.clear()
    originalDirection = Direction(1, 0)
    origin = Coords(0, 0)
    direction = copy.deepcopy(originalDirection)

    while True:
        wallLength = SEGMENT_SIZE
        wall = self.createWall(wallLength, direction, origin)

        if self._floorPlan.validateWall(wall, wallLength, direction):
            self._floorPlan.addWall(wall)
            origin = wall.getLastCoords()
            originalDirection = copy.deepcopy(direction)

        direction = self.createNewDirection(originalDirection)

        if origin.getXCoord() == 0 and origin.getYCoord() == 0:
            break
```

na dvě části, pokud je zadaný prostor stále moc velký na to, aby z něj vznikla místnost.

Generování předmětů do místností

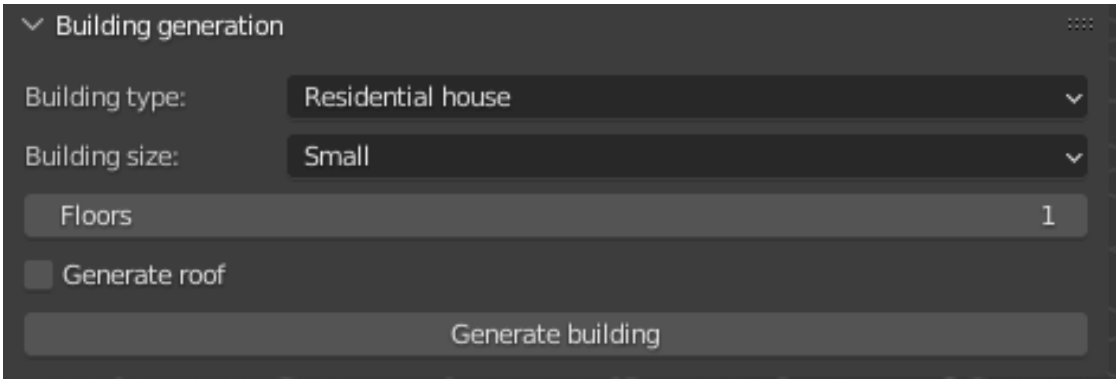
To, jakým způsobem jsou do místnosti rozmístěny předměty, je rozdílné pro každou místnost. Každý typ místnosti je reprezentován svou vlastní třídou – všechny tyto třídy jsou potomkem třídy *Room*. Logika rozmisťování předmětů do místností je implementována poměrně jednoduše. Nejprve je pseudonáhodně zvolen počet objektů, které se budou do místnosti generovat. Tyto objekty jsou pak rozmístěny do místnosti tak, aby splňovaly zadaná pravidla – například aby se nacházely u zdi nebo naopak uprostřed segmentu, v rámci kterého se objekt rozmísťuje. To, jakým způsobem funguje rozmístění objektů je vyobrazeno ve výpisu kódu 4.3. Je zde vidět struktura nejjednodušší možné místnosti, kterou je chodba. V metodě *spawn* této třídy je k vidění logika samotného rozmísťování předmětů segment po segmentu. Metoda *wallDirection*, která je ve zmíněné metodě *spawn* použita, slouží ke zjištění, ve kterých směrech se pro daný segment nachází zdi (pokud takový směr vůbec existuje). Tato metoda je zobrazena ve výpisu kódu 4.4.

4.2.2 Implementace uživatelského rozhraní

Vzhledem k jednoduchosti uživatelského rozhraní nebyla jeho implementace nijak náročná. Celé uživatelské rozhraní nám zajišťují dvě třídy – třída *MyPanel* a třída *BuildingData*. Třída *MyPanel* je potomkem třídy *bpy.types.Panel*, která je součástí rozhraní pro Blender v jazyce Python a zajišťuje nám samotné uspořádání prvků v našem uživatelském rozhraní. Třída *BuildingData* je potomkem třídy *bpy.types.PropertyGroup* – jedná se opět o součást zmíněného rozhraní, ovšem tato třída nám udává samotný obsah uživatelského rozhraní, tedy například to, z jakých možností si uživatel může vybírat. Implementace třídy *BuildingData* je popsána výpisem kódu 4.5. Jsou zde vidět možnosti, které uživatel má při použití uživatelského rozhraní. Výsledné uživatelské rozhraní je vidět na obrázku 4.5. Oproti návrhu přibyla ještě možnost, kdy si uživatel vybere, zda bude generovat střechu – střechy jsou vzhledem k složitosti řešeny pouze rovnou plochou.

Výpis kódu 4.2 Úryvek konstruktoru třídy *FloorArea* provádějící dělení prostoru, případně přiřazování typu místnosti

```
maxSegments = 3
roomType = building.prepareRoom(self.__segments)

if roomType == 'BEDROOM' or roomType == 'KITCHEN':
    maxSegments = 2
if roomType == 'LIVINGROOM':
    maxSegments = 4

if len(self.__segments) > maxSegments:
    while True:
        firstCoords = self.pickRandomCoords()
        secondCoords = self.pickRandomCoords()

        if self.validateNewWallCoords(firstCoords, secondCoords):
            self.addInsideWall(firstCoords, secondCoords)
            random.choice(self.__wallsInside).setEntrance()
            break

    leftWallsOutside, rightWallsOutside = self.splitWalls(firstCoords,
                                                          secondCoords)

    self._left = FloorArea(leftWallsOutside, floorPlan, building, True)
    self._right = FloorArea(rightWallsOutside, floorPlan, building, True)
else:
    self.__room = building.getRoom()
```



Obrázek 4.5 Uživatelské rozhraní

Výpis kódu 4.3 Implementace třídy *Hallway*

```
class Hallway(Room):
    def spawn(self, floor, floorArea, floorObject, context):
        self.clear()
        segments = floorArea.getSegments()

        counter = 0

        closetsCount = random.randint(0, max(1, len(segments) // 2))

        while True:
            for segment in segments:
                spawnCloset = bool(random.getrandbits(1))
                wallsDirections = self.wallDirection(segment,
                                                     floorArea.getWalls())

                if spawnCloset \
                    and closetsCount > 0 \
                    and len(wallsDirections) > 0\
                    and not segment\
                        .getLeftLowerCoord()\
                        .sameCoords(floorArea.getBuilding()
                                             .getStairsSegment()
                                             .getLeftLowerCoord()):
                    if self.spawnCloset(floor,
                                        segment,
                                        floorObject,
                                        wallsDirections,
                                        context):
                        closetsCount -= 1

            if closetsCount == 0 or counter > 50:
                break

    def getType(self):
        return 'HALLWAY'
```

Výpis kódu 4.4 Metoda *wallDirection*

```
def wallDirection(self, segment, walls):
    directions = []

    for wall in walls:
        if wall.containsCoords(segment.getLeftLowerCoord())\
                and wall.containsCoords(segment.getRightLowerCoord())\
                and not wall.isEntrance():
            directions.append('DOWN')

        if wall.containsCoords(segment.getLeftLowerCoord())\
                and wall.containsCoords(segment.getLeftUpperCoord())\
                and not wall.isEntrance():
            directions.append('LEFT')

        if wall.containsCoords(segment.getLeftUpperCoord())\
                and wall.containsCoords(segment.getRightUpperCoord())\
                and not wall.isEntrance():
            directions.append('UP')

        if wall.containsCoords(segment.getRightUpperCoord())\
                and wall.containsCoords(segment.getRightLowerCoord())\
                and not wall.isEntrance():
            directions.append('RIGHT')

    return directions
```

4.2.3 Výsledky

Výsledný plugin je schopen generovat budovy, které mají jedno až čtyři patra a jejichž půdorys může mít buď malou, střední nebo velkou rozlohu. Vzhledem k tomu, že plugin generuje budovy, jejichž půdorys může nabývat různých tvarů – ne jen obdélníkových – budovy nejsou zakončeny tradiční špičatou střechou, ale pouze rovnou plochou. Kvůli složitým tvarům půdorysu by bylo náročné vytvořit zároveň generátor odpovídajících střech a narostla by tím značně náročnost tvorby samotného pluginu. Ukázka výsledků vytvořeného generátoru je k vidění na obrázcích 4.6.

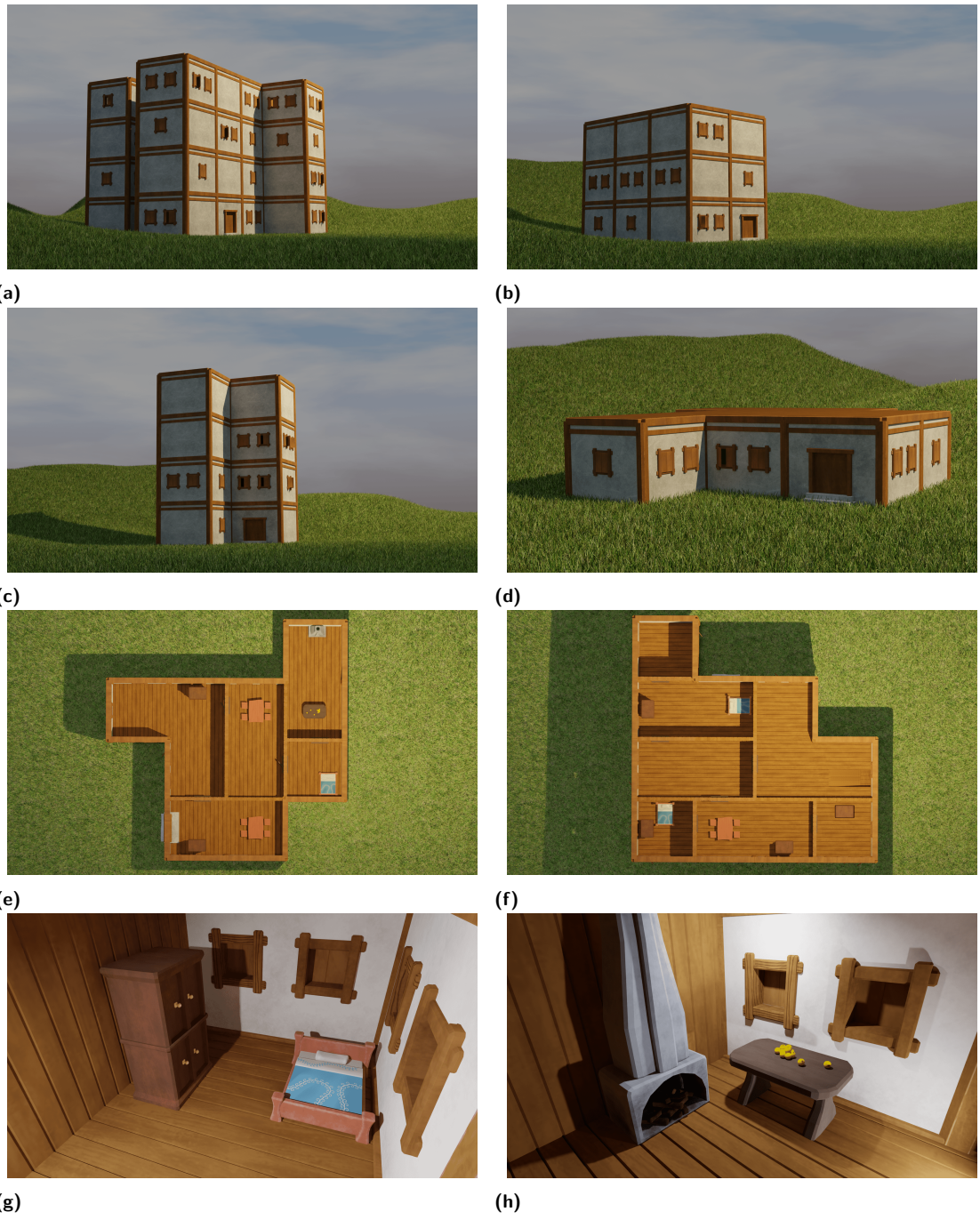
Plugin je zároveň vytvořen tak, aby šlo snadno rozšířit sadu modelů, které se při generování používají, a tím docílit ještě náhodnějších výsledků. Důležité je pouze zachovat rozměry jednotlivých modelů – například postel je dlouhá 2 metry a široká 1 metr a 60 centimetrů. Tento rozměr je používán při detekci kolizí, tudíž by pro rozdílný rozměr bylo potřeba zasáhnout do samotného kódu pluginu.

4.2.4 Instalace pluginu

Instalace pluginu je jednoduchá. V nastavení Blenderu v záložce *Add-ons* stačí kliknout na tlačítko *Install* a vyhledat soubor ve formátu *zip*, který obsahuje vytvořený plugin včetně všech potřebných souborů. Takto nainstalovaný plugin se následně aktivuje zaškrtnutím políčka *Enable Add-on*. Aktivovaný plugin se poté v Blenderu nachází pod záložkou *Tool*.

Výpis kódu 4.5 Implementace uživatelského rozhraní

```
class BuildingData(bpy.types.PropertyGroup):
        generateRoof: bpy.props.BoolProperty(
        name="Generate roof",
        description="Generate roof",
        default=False
    )

    floors: bpy.props.IntProperty(
        name='Floors',
        description='Number of floors',
        default=1, min=1, max=4, step=1
    )

    buildingTypes = [('RESIDENTIAL', "Residential house", '', '', 0)]

    buildingType: bpy.props.EnumProperty(
        items=buildingTypes,
        name='Building type',
        description='Type of the building',
    )

    buildingSizes = [('4', "Small", '', '', 0),
                     ('8', "Medium", '', '', 1),
                     ('12', "Large", '', '', 2)]

    buildingSize: bpy.props.EnumProperty(
        items=buildingSizes,
        name='Building size',
        description='Type of the building',
    )
```



(a) (b)

(c) (d)

(e) (f)

(g) (h)

Obrázek 4.6 Příklady vygenerovaných budov

Závěr

Tato práce měla za cíl prozkoumat produkční postup tvorby videoherních světů a v tomto kontextu možnosti použití procedurálního generování. Následně měl být vytvořen nástroj realizující procedurální generování budov včetně interiérů. Cílem bylo, aby uživatel následně mohl vygenerované modely použít právě při tvorbě videoherního světa. Zmíněný nástroj měl být vytvořen jako plugin do softwaru Blender.

V první kapitole byl popsán jak produkční postup při tvorbě videoherních světů, tak možnosti použití procedurálního generování. Jsou zde popsány možné přístupy použitelné při vytváření 3D modelů, možnosti použití PCG a konkrétní metody, které lze při implementaci PCG využít. Závěrem jsou v této kapitole ukázány projekty, které PCG nějakým způsobem využívají. Druhá kapitola se již věnuje rozboru konkrétních postupů, metod a nástrojů, které bylo možné použít při tvorbě výsledného pluginu. Z popsaných možností jsou zde vybrány ty, které byly následně využity při realizaci praktické části. Zároveň jsou v této kapitole zdůvodněna zmíněná rozhodnutí. V třetí kapitole je navržen samotný plugin – navrženy jsou zde například důležité algoritmy nebo uživatelské rozhraní. Ve čtvrté a poslední kapitole je popsána tvorba samotného pluginu. V této kapitole jsou také předvedeny některé výsledky, které vznikly pomocí vytvořeného nástroje.

Plugin byl vytvořen dle zadání – výsledný nástroj je ovšem realizován jako jediný plugin a ne jako sada pluginů, jak je napsáno v zadání – vzhledem k požadované funkčnosti dává tento přístup větší smysl. Jedná se o velmi komplexní plugin, který zajišťuje veškeré požadované procesy – jak pseudonáhodné vytvoření půdorysu budovy a následné vygenerování jejího exteriéru, tak rozmístění objektů do interiéru budovy. Plugin je navíc navržen tak, aby celý tento proces mohl proběhnout bez jakéhokoliv zásahu uživatele. Rozšíření o další pluginy by se nabízelo v případě, kdy budeme generovat celá města – jeden plugin by pak mohl sloužit ke generování jednotlivých budov a druhý plugin, který by byl prvnímu pluginu nadřazený, by vygenerované budovy rozmisťoval do prostoru.

Dalším krokem by mohlo být rozšíření výsledného pluginu o další modely, typy místností a typy budov, abychom dosáhli větší náhodnosti a rozdílnosti při generování – s tímto rozšířením by bylo rozumné také vylepšit jednotlivé metody, které se starají o samotné generování místností. Zároveň by mohl být vytvořen generátor klasických šikmých střech. Nejzajímavějším rozšířením by byl několikrát zmíněný generátor měst, který by opakovaně generoval budovy a rozmísťoval je do prostoru.

Tato práce měla představit jak nejrůznější metody používané při tvorbě světů do počítačových her, tak metodu procedurálního generování používanou v tomto kontextu. Mělo se jednat o úvodní text, který bude sloužit jako rozcestník pro všechny, kteří by se tomuto tématu chtěli dále věnovat. Cíle práce, ať už se týkají teoretické, či praktické části, byly tedy splněny a tento text se snad pro některé čtenáře stane zmíněným rozcestníkem.

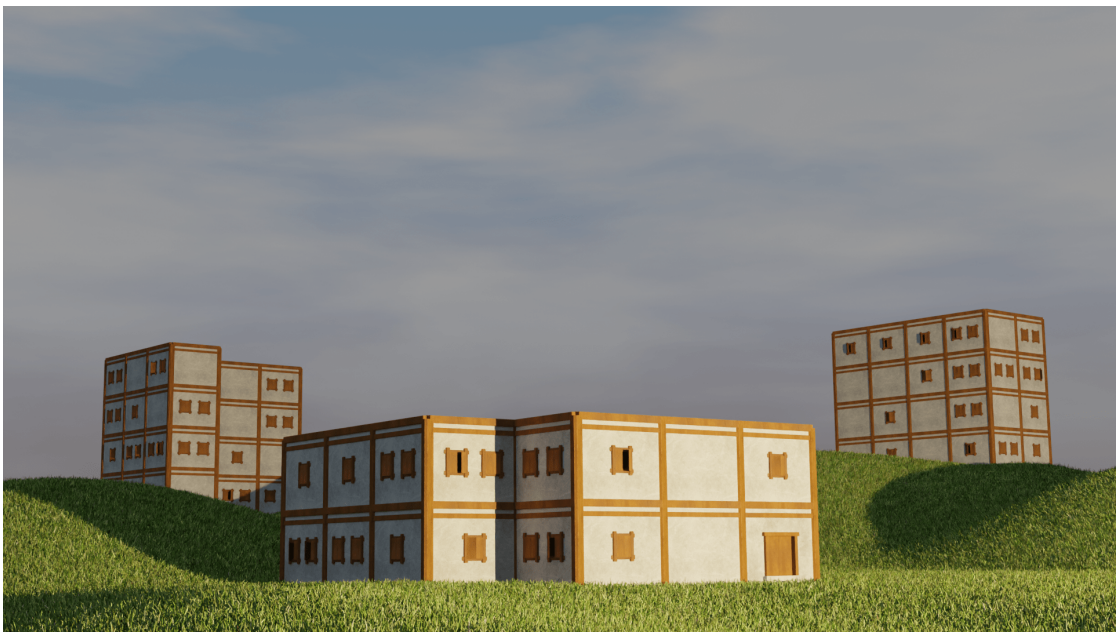
Příloha A

Další výsledky

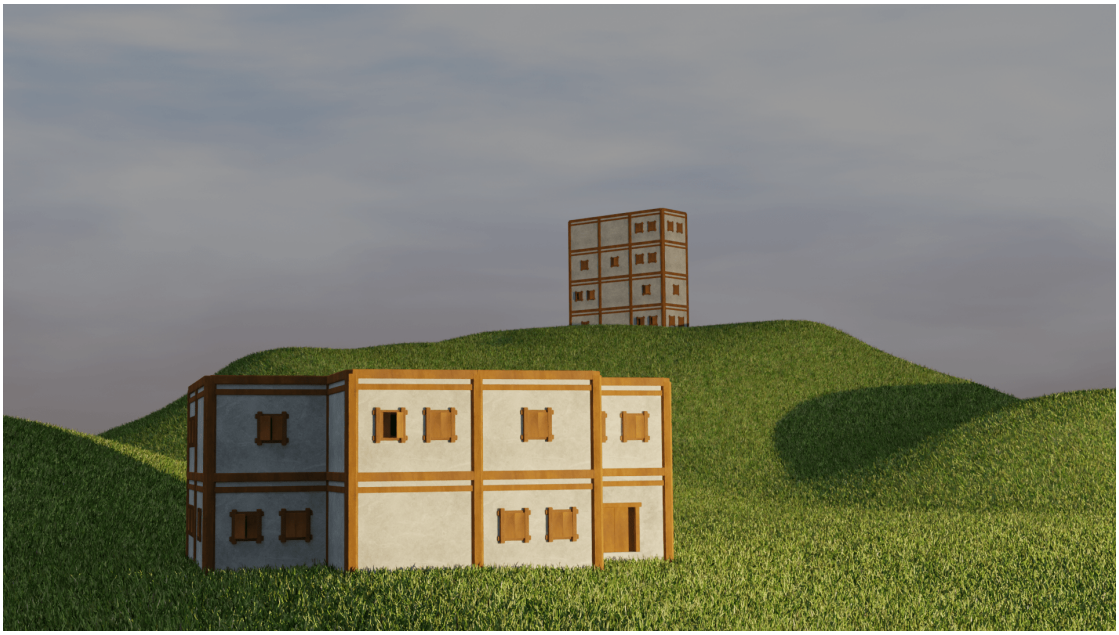
Součástí této přílohy jsou rendery modelů, které nebyly zobrazeny ve čtvrté kapitole. Jedná se pouze o modely objektů, které jsou použity při generování interiérů. Zároveň je uvedeno pár dalších příkladů využití vytvořeného generátoru.

Obrázek A.1 Střední budova se čtyřmi patry

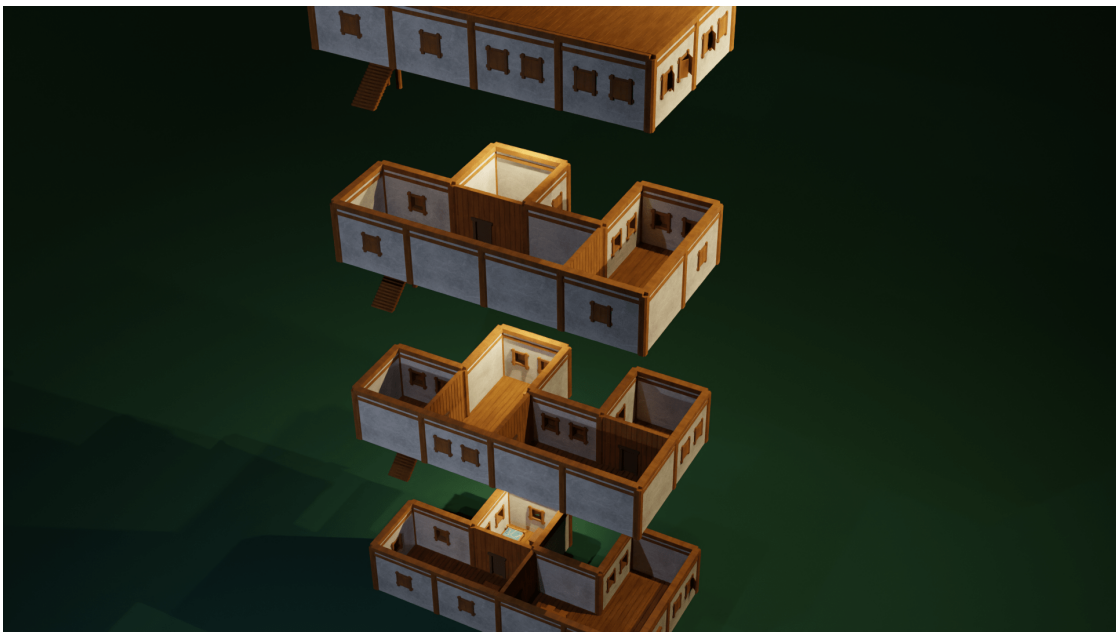
Obrázek A.2 Malá budova se třemi patry



Obrázek A.3 Tři budovy rozmístěné ručně do prostoru

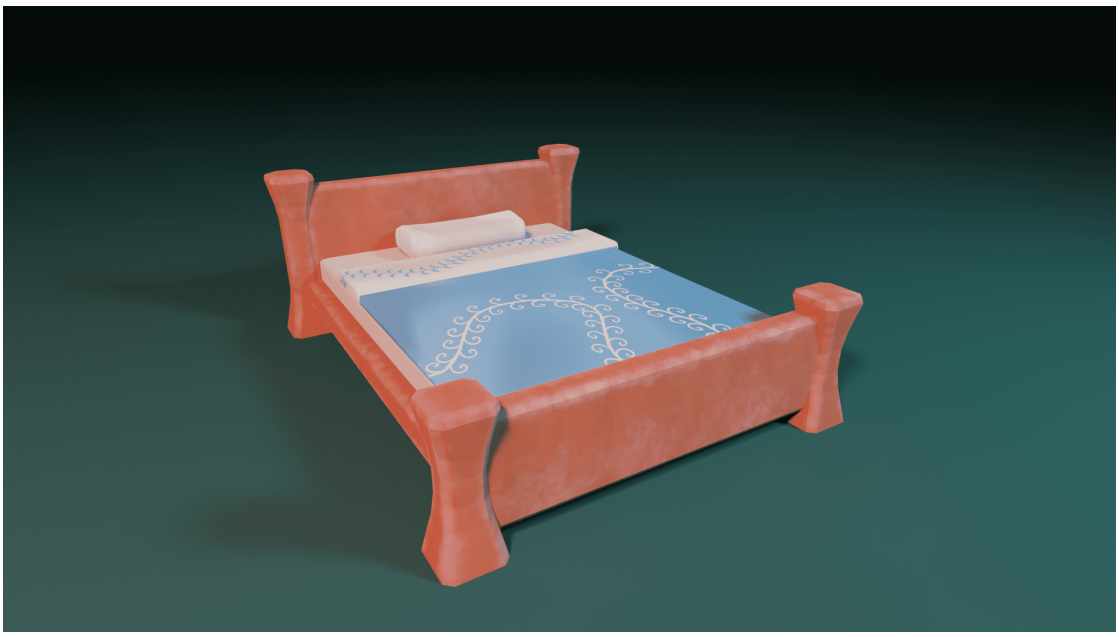


Obrázek A.4 Dvě budovy rozmístěné ručně do prostoru



Obrázek A.5 Průřez malou budovou

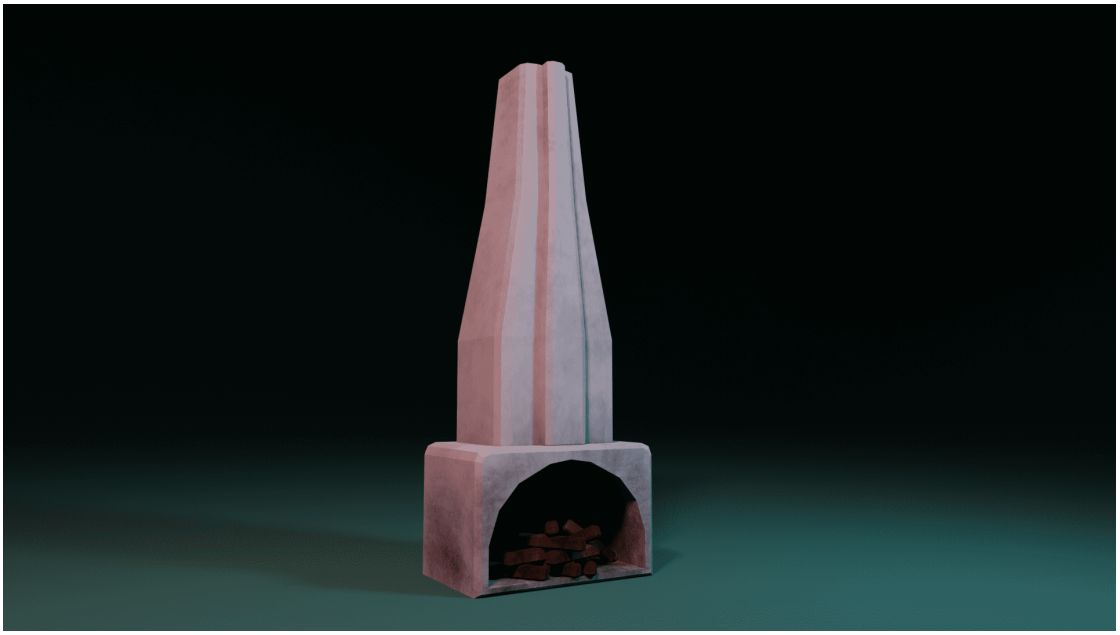
Obrázek A.6 Průřez středně velkou budovou



Obrázek A.7 Vytvořený model postele

Obrázek A.8 Vytvořený model kuchyňského stolu

Obrázek A.9 Vytvořený model skříně



Obrázek A.10 Vytvořený model ohniště

Bibliografie

1. GUEVARRA, Mendoza; CHATTERJEE. *Creating Game Environments in Blender 3D*. Springer, 2020.
2. BYCER, Josh. *Level design vs. environment design*. SUPERJUMP, 2020. Dostupné také z: `https://superjumpmagazine.com/level-design-vs-environmental-design-b8d19992924e`.
3. AHEARN, Luke. *3D Game Environments: Create professional 3D game worlds*. AK Peters/CRC Press, 2017.
4. DUNLOP, Renee. *Production Pipeline Fundamentals for Film and Games*. Routledge, 2014.
5. FLAVELL, Lance. *Beginning blender: open source 3d modeling, animation, and game design*. Apress, 2011.
6. TAYLOR, James. *Modeling vs sculpting: How do you know which to use?* 2016. Dostupné také z: `https://www.methodj.com/modeling-vs-sculpting/`.
7. SANDEN, Henning; JAEGER, Morten. *Concept Sculpting for Film and Games*. flippednormals, [b.r.]. Dostupné také z: `https://flippednormals.com/downloads/concept-sculpting-for-film-and-games/`.
8. KUMAR, Abhishek. *Beginning PBR Texturing: Learn Physically Based Rendering with Allegorithmic's Substance Painter*. Apress, 2020.
9. *THE PBR GUIDE - PART 2*. Adobe, [b.r.]. Dostupné také z: `https://substance3d.adobe.com/tutorials/courses/the-pbr-guide-part-2`.
10. SMITH, Gillian. An Analog History of Procedural Content Generation. In: *FDG*. 2015.
11. SHORT, Tanya; ADAMS, Tarn. *Procedural generation in game design*. CRC Press, 2017.
12. SHAKER, Noor; TOGELIUS, Julian; NELSON, Mark J. *Procedural content generation in games*. Springer, 2016.
13. BROGAN, Jacob. *The paradox of using algorithms to create Infinite Video Game Worlds*. Slate, 2016. Dostupné také z: `https://slate.com/technology/2016/10/the-paradox-of-procedurally-generated-video-games.html`.
14. *7 uses of procedural generation that all developers should study*. Game Developer, 2016. Dostupné také z: `https://www.gamedeveloper.com/design/7-uses-of-procedural-generation-that-all-developers-should-study`.
15. CHRYSOSTOMOU, George. *10 things to know about shadow of war's nemesis system*. 2021. Dostupné také z: `https://screenrant.com/shadow-of-war-nemesis-syustem-facts-rivia/`.

16. ZAMBRANO, J.R. *D&D: Map your dungeon with an AI - Dungeon Alchemist*. 2022. Dostupné také z: https://www.belloflostsouls.net/2022/04/dd-map-your-dungeon-with-an-ai-dungeon-alchemist.html.
17. PETTY, Josh. *What is zbrush: How it works & what it's used for*. 2018. Dostupné také z: https://conceptartempire.com/what-is-zbrush/.
18. *Substance painter vs Mari which is better?* InspirationTuts, 2021. Dostupné také z: https://inspirationtuts.com/substance-painter-vs-mari-which-is-better/.
19. AAVA, Kim. *Realistic vs. stylized: Technique Overview*. 80lv, 2019. Dostupné také z: https://80.lv/articles/realistic-vs-stylized-technique-overview/.
20. SOZAP. *Procedural abandoned house with Geometry Nodes*. 2022. Dostupné také z: https://blenderartists.org/t/procedural-abandoned-house-with-geometry-nodes/1363024.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `plugin` ..... adresář obsahující archiv s pluginem
- `src`
  - `python` ..... zdrojové kódy implementace
  - `blender` ..... adresář obsahující jednotlivé scény, modely a textury
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF