VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

BLENDER ADD-ON PRO GENEROVÁNÍ HRADŮ

CASTLE GENERATOR BLENDER ADD-ON

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **RICHARD GALL**
AUTHOR

VEDOUCÍ PRÁCE — **TOMÁŠ CHLUBNA, Ing.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Gall Richard**
Program: Informační technologie
Název: **Blender add-on pro generování hradů**
Castle Generator Blender Add-on
Kategorie: Počítačová grafika

Zadání:

1. Naučte se základy práce s 3D modelovacím programem Blender (verze 2.9 a výše).
2. Seznamte se se skriptovacím Blender Python API.
3. Navrhněte add-on, který rozšíří Blender o novou funkcionalitu generování hradů a podobných staveb.
4. Nastudujte možná řešení dané problematiky.
5. Navrhněte uživatelské rozhraní pro výsledný add-on.
6. Add-on implementujte a demonstrujte jeho použití na různých typech dat.
7. Zdokumentujte a zveřejněte výsledný add-on pro použití dalšími uživateli.
8. Vytvořte video reprezentující výsledky vaší práce.

Literatura:

- Blender Python API Documentation https://docs.blender.org/api/current/index.html

- Ebert, David S., et al. Texturing & modeling: a procedural approach. Academic Press, 2014. ISBN 1483297020, 9781483297026

- Lagae, Ares, et al. "A survey of procedural noise functions." *Computer Graphics Forum*. Vol. 29. No. 8. Oxford, UK: Blackwell Publishing Ltd, 2010.

- Fita, Josep Lluis, Gonzalo Besuievsky, and Gustavo Patow. "A perspective on procedural modeling based on structural analysis." *Virtual Archaeology Review* 8.16 (2017): 44-50.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Chlubna Tomáš, Ing.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 1. listopadu 2021

Abstrakt

Práce se zabývá tvorbou addonu pro Blender, který umožňuje procedurální generování modelu hradu. Na rozdíl od existujících řešení nabízí addon možnosti generování celých opevnění, tvrzí věží a bran. A využívá moderních postupů v Blenderu jako jsou geometry nodes. Addon byl zveřejněn a otestován uživateli.

Abstract

The thesis focuses on creation of a Blender addon, that allows procedural castle generation.Unlike other solutions this addon offers generation of complete fortifications such as castle walls, keeps, towers and gates. The addon uses modern procedures im Blender such as geometry nodes. The addon was published online and tested by users

Klíčová slova

Blender addon, procedurální generování, hrady, Geometry nodes, 3D modelování, Blender

Keywords

Blender addon, procedural generation, castles, Geometry nodes, 3D modelling, Blender

Citace

GALL, Richard. *Blender add-on pro generování hradů*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Tomáš Chlubna, Ing.

Blender add-on pro generování hradů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Tomáše Chlubny Ing. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Richard Gall
8. května 2022

Poděkování

Rád bych poděkoval mému vedoucímu, kterým byl Tomáš Chlubna Ing., za odborné rady, zpětnou vazbu a ochotu pomáhat při tvorbě této práce.

Obsah

Kapitola 1

Úvod

V praxi, při tvorbě videoher, animovaných filmů či krátkých animací, může být složité a časově náročné vygenerovat model hradu. Ve snaze usnadnit tento úkol je cílem této práce tvorba addonu pro 3D editor Blender, který umožní procedurální generování hradu s vysokou mírou detailů. Existující nástroje, které částečně tuto možnost poskytují, neumožňují po vytvoření modelu jeho následující editaci a je tedy nutné při změně parametrů vygenerovat daný prvek znovu. Dále je addon schopen generovat nejen rovnou hradní zeď ale také členité komplexní opevnění, včetně detailů jako je cimbuří, podsebití, věže, brány a také samostatné tvrze.

V této práci jsou mimo tvorby addonu rozepsány také metody procedurálního tvoření objektů, textur, ale také architektur hradů a fungování Blenderu. Tyto témata jsou popsány v kapitole 2 a dávají dostatečný teoretický základ pro následnou implementaci. V kapitole 3 je zaznamenán terénní výzkum, který byl prováděn na hradě Helfštýn, během kterého byly získány fotky, pro referenci a zhodnocení finálního výsledku. Kapitola 4 nabízí pohled na základní struktury hradu pro účely implementace, parametry, které má mít uživatel k dispozici, a uživatelské rozhraní. A v kapitole číslo 5 je do detailů rozepsán způsob implementace v Geometry nodes, které Blender nabízí. Výsledný addon je volně dostupný pro uživatele a součástí práce je i zpětná vazba uživatelů.

Kapitola 2

Teorie

Pro procedurální generování modelů hradů a opevnění je nutno nastudovat několik věcí. Mezi první z nich patří tvorba procedurálních modelů a textur, dále je nutno získat informace týkající se struktury a architektury hradů a v neposlední řadě je nutné se seznámit s nástroji ve kterých bude projekt vytvářen. V následujících sekcích jsou popsány metody procedurálního generování, principy šumu, fungování textur a shaderů, architektura hradů a funkce, které nabízí Blender.

2.1 Procedurální generování

V počítačové grafice jsou dva hlavní způsoby vytváření objektů. V první řadě je možné ručně vytvořit model pevným nastavením statických hodnot bodů, hran a stěn. V případě velkého množství podobných objektů je tento způsob ovšem časově náročný a ve většině případů jsou provedené změny geometrie nevratné, je tedy obtížné u takto vytvořených objektů rychle editovat jejich parametry. Alternativou je procedurální generování, jehož cílem je vytvářet objekty na základě matematických pravidel oproti pevnému stanovení jednotlivých hodnot. Ve výsledku je tak možno změnou parametrů měnit výsledný objekt.

Příkladem je generování lesa. Konvenčními metodami by bylo nutné modelovat každý strom individuálně, následkem čehož by výsledné modely neobsahovaly mnoho detailů a byly časově velmi náročné. V případě procedurálního vytváření stromu lze jeho modely vytvářet tolikrát, kolikrát uživatel potřebuje, s různými parametry.

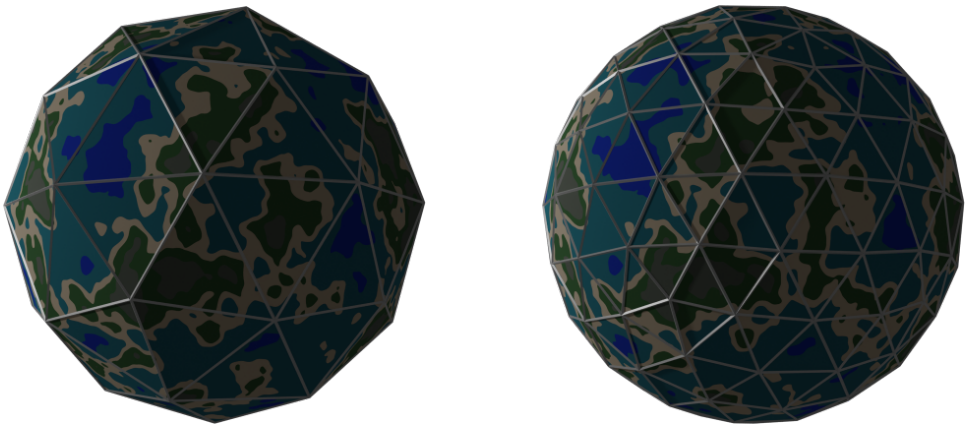
Další výhodou procedurálního generování je také možnost aplikovat nekonečně velké detaily. V dnešní době, kdy výpočetní výkonost stále roste a nová media se předhánějí ve využití co nejlepších speciálních efektů, a reálně vypadajících 3D modelů, je procedurální generování téměř nutností.

V článku „A Survey of Procedural Techniques for City Generation“ [13] jsou popsány metody procedurálního generování, jako jsou fraktály, L-systémy, metody procedurálního šumu a tvorby procedurálních textur. Procedurální techniky se již 30 let využívají v počítačové grafice [13]. Ať už se jedná o přidávání šumu do textur, či generování fotorealistických materiálů jako je dřevo či mramor, nebo v generování modelů rostlin. Procedurální textury lze najít téměř v každém odvětví počítačové grafiky. Procedurální metody dokáží za okamžik vygenerovat to, co by týmu umělců trvalo měsíce, je tedy ekonomicky i časově výhodné je využívat.

Mezi výhody procedurálního generování se řadí také abstrakce. Data geometrie a textur nejsou specifikovány konvenčním způsobem, ale namísto toho jsou detaily abstrahovány

do algoritmu či množiny procedur. Tyto procedury jsou poté zpracovány počítačem a volány, když jsou potřeba. Uživateli tedy stačí pouze obecná znalost parametrů pro vytváření modelů.

Další z výhod je kontrola parametrů, které jsou definovány tak, aby přímo korespondovaly ke specifickému chování v procedurálním generování. Vývojář může nadefinovat co možná nejvíce užitečných nastavení, které umělec může efektivně využít. Příkladem využití je výška stromu, hustota větví a tloušťka kmenu, v procedurálním generování lesa či počet segmentů v procedurálním generování planety (viz obrázek 2.1). Flexibilita a kontrola kterou procedurální generování poskytuje je užitečná k tvorbě umělecky nových a experimentálních objektů vhodným zvolením parametrů.



(a) Planeta s nízkým počtem segmentů (b) Planeta s vyšším počtem segmentů

Obrázek 2.1: Rozdíl uživatelem definované míry detailů

Procedurálním generováním také lze dosáhnout lepšího rozložení úrovně detailu (anglicky „Level of detail“, dále jen LOD). Pro objekty které jsou velmi daleko od kamery, a řádově zabírají jednotky pixelů, není nutné mít velký počet polygonů. Optimalizace tedy lze dosáhnout snížením kvality takového modelu vhodným nastavením parametrů. V případě rekurzivního generování objektů, které je popsáno v následujících kapitolách, je LOD využito k ukončení rekurze.

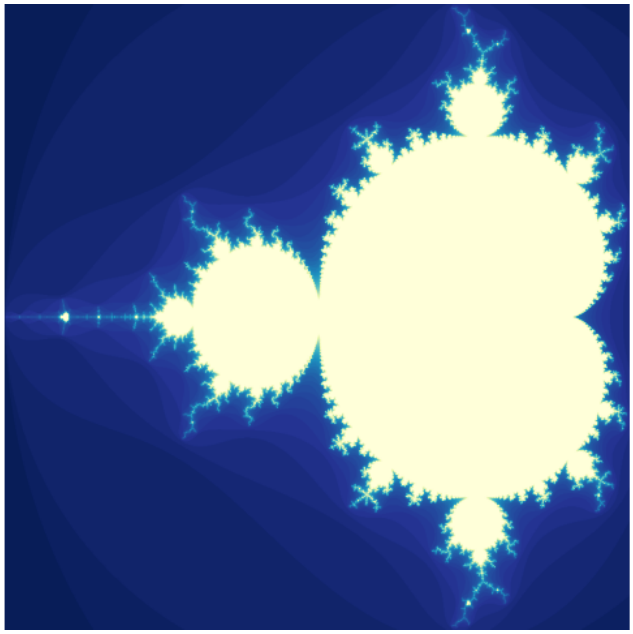
2.1.1 Fraktály

Objekty v přírodě mají velmi komplexní tvary. Při bližším přezkoumaní ovšem lze zjistit, že se často jedná o matematicky jednoduchý proces který se často opakuje v dílčích částech. Jedná se o malé prvky sebe-podobnosti, které se nacházejí jako dílčí části celku. Jako ostatní procedurální metody i fraktály jsou definovány předpisem pro jejich generování. Pro generování fraktálů stačí využít jednoduchého matematického zápisu a rekurzivního volání této metody. Kolikrát bude volána tím větší bude detail samotného fraktálu.

Příkladem je Mandelbrotova množina, která má následující matematický zápis [16]:

$$Z_0 = 0 \tag{2.1}$$
$$Z_{n+1} = Z_n^2 + c \tag{2.2}$$

Kde hodnota Z_n je komplexní číslo, aktuální iterace které je v každé iteraci umocněno na druhou, a komplexní konstanta c, která je v každé iteraci přičtena. Hodnota Z_{n+1} je hodnotou následující iterace. Tato operace bude probíhat po n iterací, kde n je přirozené číslo. Iterace zpravidla končí ve chvíli, kdy dodatečné detaily, získané další iterací, jsou natolik malé, že je v grafické reprezentaci již nelze zobrazit. Na obrázku 2.2 lze vidět grafickou reprezentaci Mandelbrotovy množiny.



Obrázek 2.2: Fraktál - Mandelbrotova množina. Na tomoto obrázku lze zde vidět světlé hodnoty, které konvergují k 0, tmavé hodnoty, které divergují k nekonečnu a modré hodnoty, které konvergují k jednomu bodu, či periodicky nabývají několika hodnot. Tento příklad byl vygenerován z 25 iterací.

2.1.2 L-systémy

Lindenmayerovi systémy (dále jen L-systémy) jsou formální gramatikou vytvořenou biologem A. Lindenmayerem jako matematická teorie pro biologický vývoj.

L-systémy byli původně vyvinuty pro studium replikace bakterii a růstových vzorců jednoduchých organismů jako jsou řasy, jež byly popsány v „Journal of Theoretical Biology“ v roce 1968 [15]. Samotný systém byl vyvinut, a v dnešní době slouží pro generování fraktálu a realistickému modelování rostlin.

L-systémy jsou metodou úpravy komplexních objektů následným přepisováním částí jednoduchého původního objektu množinou přepisovaných pravidel. Komponenty L-systému jsou následující:

- V je abeceda, množina symbolů obsahující proměnné (prvky, které jsou nahrazovány).

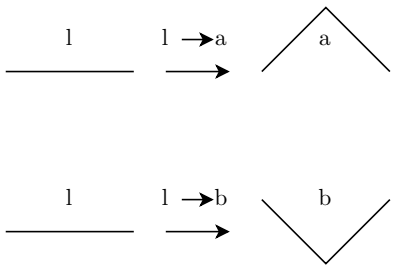
- S je množina symbolů obsahující konstanty (prvky, které zůstanou stejné).
- ω začátek (axiom) jedná se o řetězec symbolů a konstant, které definují počáteční stav systému.
- P je konečná množina přepisovacích pravidel, které určují jakým způsobem budou proměnné nahrazovány kombinací konstant a jiných proměnných. Pravidlo se skládá z dvou řetězců, předchůdce a následníka.

Původní stav neboli axiom ω je přepsán za využití sérií přepisovaných pravidel. Přepisovatelná pravidla jsou aplikována iterativně, což umožňuje libovolně složité objekty definovat jako jednoduchou množinu pravidel.

V praxi lze L-systémů využít ke generování komplexních modelů organických struktur. Parametr hloubky rekurze definuje detail výsledného objektu. Algoritmy pro jejich generování jsou kompaktní, intuitivní a lze jimi pokrýt velké množství přírodních struktur. L-systémy mohou být snadno pozměněny pomocí parametrů a jsou velmi podobné ostatním gramatikám. Příklad využití L-systémů lze nalézt v ukázkách 2.1, 2.3 a 2.4.

$V = a, b, l$		$\omega : a$		$n = 0$	l
		$\omega = a$		$n = 1$	a
P_1 :	l	$\rightarrow$	a	$n = 2$	ab
P_2 :	a	$\rightarrow$	ab	$n = 3$	$abba$
P_3 :	b	$\rightarrow$	ba	$n = 4$	$abbabaab$

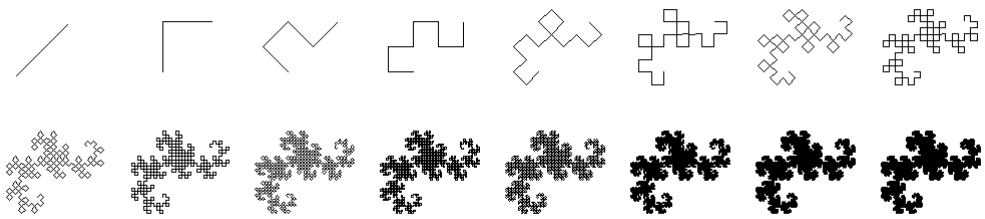
Tabulka 2.1: Ukázka pravidel L-systémů, kde n je číslo iterace



Obrázek 2.3: grafická reprezentace pravidel

Obrázek 2.4: grafická reprezentace iterací pravidel, kde každá přímka bude v další iteraci nahrazena dvěma dalšími, na sebe kolmými přímkami. V případě pravidla a, bude první přímka rotována doleva, v případě pravidla b bude rotována doprava.

Za využití rekurzivní transformace jednotlivých elementů dojde k vytvoření dračí křivky, jak je ukázáno na obrázku 2.5.



Obrázek 2.5: L-systém - Dračí křivka

Jak vidno z obrázku 2.5 i z velmi jednoduchého pravidla lze vygenerovat velmi komplexní obrazec. L-systémy mohou být závislé na kontextu, kde každý symbol bude nahrazen jiným pravidlem na základě okolních symbolů. Dalším dodatkem k L-systémům je využití želví grafiky. Želví grafika je způsob vektorového kreslení v kartézské soustavě souřadnic za využití kurzoru, tzv. želvy, který je ovládán příkazy posunu a otočení. Kurzor za sebou zanechává stopu, kterou lze nastavit parametry štetce. Želví grafiky je využíváno k vykreslení L-systémů a jejich rozšíření o závorky, které uloží pozici a úhel kurzoru. Pomocí této metody lze generovat velké množství přírodních útvarů jako jsou stromy či trávy, viz obrázek 2.6.

Obrázek 2.6: L-systém - Stéblo

2.2 Textury a shadery

Z kapitoly „Building Procedural Textures“ z knihy „Texturing & Modeling: a Procedural Approach“ [7] lze zjistit že vzhled povrchů je možné vytvářet dvěma způsoby a to pomocí procedurálního shaderu, nebo pevně stanovené textury.

Z počátku vývoje shaderů se shadery vypočítávaly převážně z přímého světla pocházejícího ze světelných zdrojů. Na počátku 80. let se vývoj zaměřil na simulaci globálního osvětlení, které pochází z nepřímého světla následky odrazů, lomu světla a rozptylu světla z ostatních povrchů ve scéně. V praxi se často využívá ray-tracing a radiosita.

Z počátku byly metody mapování textur velmi primitivní. V této iteraci každý bod povrchu v 3D prostoru korespondoval k specifickému bodu (u,v) v 2D v parametrovém prostoru. Touto metodou mohla být 2D textura namapována na 3D povrch. Každá hodnota (u,v) takto mohla být namapována na odpovídající pixel v texturovém obrázku.

Takto jednoduchá metoda vzorkování pixelu na jeden bod bohužel povede k chybám způsobeným aliasingem. Aby se tomuto předešlo, je nutné využít pokročilejších mapovacích metod.

Blinn and Newell (1976) [5, 7] přišli s odrazovým mapováním, které simuluje odrazy, podobně jako povrch zrcadla. Reflexní mapování je zjednodušená podoba ray-tracingu. Tato metoda vypočítává směr paprsku R vůči kameře k bodu, který je vykreslován. Vzorcem zapsáno jako:

$$R = 2(N \cdot V)N - V \tag{2.3}$$

Kde N je normála povrchu a V je vektor směřující ke kameře (oba vektory musí být normalizovány). Textura je mapována pomocí úhlů Θ,φ, které reprezentují šířku („latitude“) a délku („longitude“) normalizovaného vektoru $R = (x, y, z)$ ve sférické souřadnicové soustavě (viz. vzorec 2.4 a 2.5).

$$\Theta = tan^{-1}(\frac{x}{y}) \tag{2.4}$$

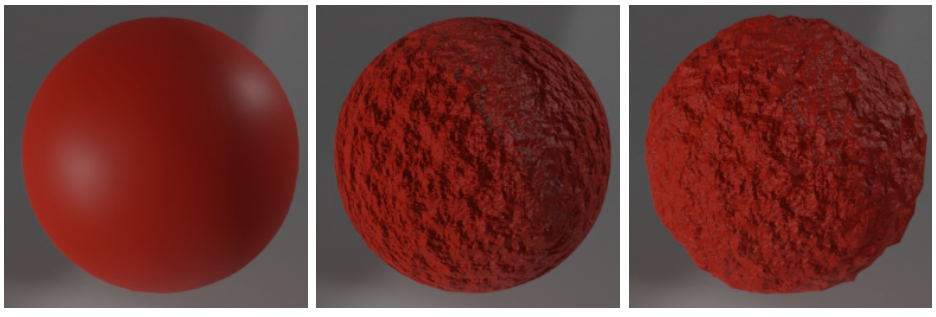
$$\varphi = sin^{-1}(z) \tag{2.5}$$

V případě vhodně zvolené odrazové textury, bude povrch odrážet obraz okolí objektu. Vzhled takového objektu je zdokonalen, je li scéna obehnána koulí s namapovaným prostředím. Pomocí úhlů šířky a délky průniku lze přistoupit k odrazové textuře.

2.2.1 Bump a displacement mapping

V roce 1978 James Blinn přišel s konceptem bump mappingu [4]. Bump mapping umožnil simulaci vzhledu členitého povrchu aniž by měnil geometrii. Bump mapping využívá textury k modifikaci směru normál povrchu. Nedostatkem bump mappingu je že silueta objektu zůstane nezměněná, jelikož dochází pouze ke změně výpočtu shaderů, nikoliv geometrie.

Bump mapping byl v roce 1984 následně rozšířen R. L. Cookem [6]. Toto rozšíření nese název displacement mapping. V tomto rozšíření textura skutečně pohybuje s povrchem objektu a nemění pouze normály. Asi nejpatrnějším rozdílem mezi bump mappingem a displacementem je v siluetách objektů, zatímco bump mapping zachovává siluety objektu původního, displacement mění povrch a s ním i siluety, viz obrázek 2.7

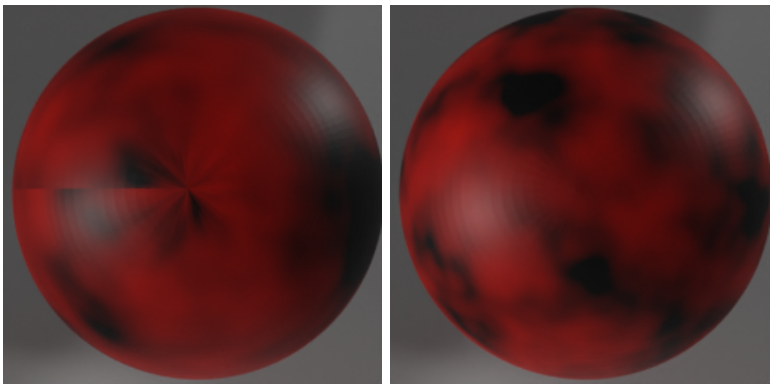


(a) Základní koule (b) bump mapping (c) displacement

Obrázek 2.7: Příklad bumpu a displacementu

2.2.2 Pevné textury

Roku 1985 popsali Peachey a Perlin prostor vyplňující textury zvané pevné (anglicky „solid textures“). Jedná se o alternativu k 2D texturám které jsou mapovány na povrch objektu. Oproti tomu pevné textury jsou vyhodnocovány na základě 3D souřadnic bodu, který je právě texturován (viz obrázek 2.8). Výhodou pevných textur je fakt že nejsou ovlivněny zakřivením prostoru povrchů objektu (například póly UV koule).



(a) povrchová textura (b) pevná textura

Obrázek 2.8: Příklad pevných textur

2.3 Šum

Pro dosažení různých výsledků při procedurálním generování je výhodné využít šumu. Šum je generátor náhodných čísel. Jedná se o náhodný, nestrukturovaný vzor, který je využit kdekoliv, kde je potřeba vysokého detailu, který nicméně nemá zjevně viditelnou strukturu [14]. Náhodné vzorce jsou zpravidla popsány ve frekvenční doméně a zároveň platí, že v prostorové doméně je signál vyhodnocen v každém místě prostoru. Ve frekvenční doméně je signál definován amplitudou a fází pro každou frekvenci. Jelikož u šumu je fáze náhodná a není tedy podstatná, bývá zpravidla signál popsán amplitudami svého spektra.

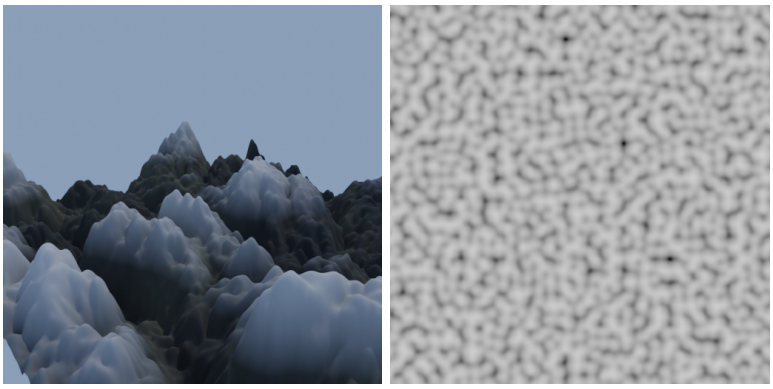
Funkce procedurálního šumu jsou často využívané v počítačové grafice. Jejich hlavní výhodou je schopnost vytvořit komplexní detaily (viz obrázek 2.9) za cenu nízkého využití paměti a jednoduchého zápisu instrukcí. Změnou parametrů tak lze rychle vytvořit velké množství různých vzorů.

Pomocí šumu lze vytvářet procedurální textury mraků, vln, povrchů země a mnoho jiného. Lze jej také využít jako masky při distribuci částic na objekty nebo pro drobný pohyb při animacích, který přidává na pocitu skutečnosti.

Šum je zpravidla tvořen sumou několika dílčích šumů o různé frekvenci, dílčí prvek frekvence se nazývá oktáva. V přírodě je mnoho velikostí detailů. Jako příklad lze využít horské pohoří. Velké detaily jsou velké vrcholy a velká údolí, střední detaily jsou malé pahorky a hřebeny a malé detaily tvoří balvany, kameny apod.

Za účelem vytvoření užitečnější textury, která lépe reflektuje přírodní rozmanitost, je nutné využít kombinaci několika šumu různých oktáv, o rozdílných amplitudách a frekvencích. Variance amplitud a frekvencí je vyjádřena jako hodnota persistence. Persistence umožňuje vyjádření efektu, který mají následující oktávy na předchozí iteraci, tím že defi-

nuje amplitudy mezi oktávami jako zlomek. Persistence je blíže popsána v kapitole Perlinova šumu 2.3.2.



(a) Procedurální hory (b) Procedurální šum

Obrázek 2.9: Ukázky praktického využití šumu v počítačové grafice

2.3.1 Formální definice šumu

Z článku „A Survey of Procedural Noise Functions“ [14] lze vyčíst následující definici: Pro diskrétní náhodný proces $y = N(x)$ je funkce hustoty pravděpodobnosti F_N n-tého řádu definována následovně:

$$F_N(y_1, y_2 \ldots, y_n; x_1, x_2, \ldots, x_n) = P(N(x_1) = y_1, N(x_2) = y_2, \ldots, N(x_N) = y_n) \tag{2.6}$$

Jedná se zároveň o pravděpodobnost P že šum nabývá hodnoty y_k v n místech x_k. První řád pravděpodobnostní funkce je zpravidla nazývána distribucí amplitud neboli histogramem signálu.

Pro n-tý obecný moment je vážený průměr odpovídajících funkcí hustoty pravděpodobnosti n-tého řádu. Moment prvního řádu je definován jako střední hodnota:

$$E[N(x)] = \int y F_N(y) dy \tag{2.7}$$

Moment druhého řádu je očekávaná hodnota šumu na dvou místech a je nazývána autokorelační či autokovarianční:

$$E[N(x_1)N(x_2)] = \int \int y_1 y_2 F_N(y_1, y_2; x_1, x_2) dy_1 dy_2 \tag{2.8}$$

Stacionární funkce je taková funkce, jejíž statistika je invariantní vůči posunu v podčásti souřadnicového systému a náhodné funkce jsou izotropní pakliže její statistika je invariantní vůči rotaci souřadnicového systému. Pro stacionární a izotropickou funkci je autokorelace R redukována na funkci o jedné proměnné.

$$E[N(x_1)N(x_2)] = R(|x_1 - x_2|) \tag{2.9}$$

Šum tedy lze definovat jako: stacionární a normální (Gaussovský) náhodný proces. Kontrola nad amplitudovým spektrem je poskytována buďto přímo či sumou hodnot nezávislých násobených instancí typicky fázově omezeného šumu.

Šum bývá procedurální. Hlavní výhody procedurálních funkcí šumu jsou tyto:

- Funkce procedurálního šumu jsou velmi kompaktní. Zpravidla zabírají několik kilobytů v porovnání s megabyty pro obrázky obsahující šum.
- Procedurální funkce šumu je spojitá, vícerozměrná a není závislá na diskrétně měřených datech. Je schopná generovat šum v libovolném rozlišení.
- Funkce procedurálního šumu vyplňuje celý n-rozměrný prostor. Je tedy schopná vyplnit celý prostor bez zjevných okrajů a bez viditelného opakování.

Funkce procedurálního šumu je parametrizovaná, je tedy schopná vygenerovat velké množství podobných vzorů, na rozdíl od jediného vzoru. Parametr udává amplitudové spektrum funkce, které charakterizuje funkci.

Funkce procedurálního šumu může být vyhodnocena v konstantním čase nezávisle na pozici, a předchozím vyhodnocení. Díky těmto vlastnostem lze efektivně využít grafické karty a vícejádrových procesorů pro paralelní výpočty.

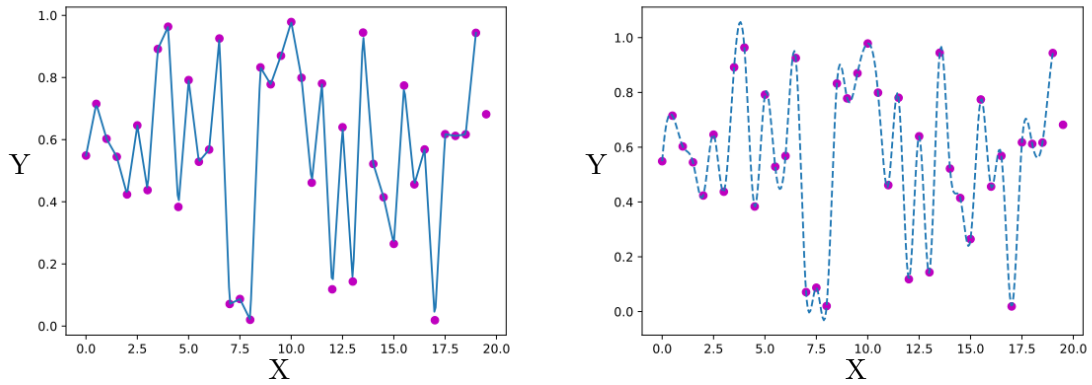
Texturové funkce jsou definovány jako funkce, které generují vzory které lze použít jako základ pro generování textur. Tvorba vizuálně bohatých textur na základě šumu není snadný úkol, jelikož nepředvídatelná a náhodná povaha šumu, zpravidla ztěžuje schopnost předpovídání výsledku. Přesto se jedná o nedocenitelnou položku procedurální tvorby, která umožňuje přiblížit se chaosu reality.

2.3.2 Perlinův Šum

Perlinův šum lze využít ke generování přírodních útvarů jako jsou mraky, a hory či textury jako je kůra stromů či mramor. Výhodou perlinova šumu je mnohem organičtější vzhled, který je způsoben vyhlazenou sekvencí pseudo-náhodných čísel.

Oproti bílému šumu, který v každém čase vrací náhodnou hodnotu z daného rozmezí Perlinuv šum vychází z interpolovaných hodnot. Perlinův šum vyžaduje argument určující v jakém čase bude hodnota vrácena. Toto je užitečné jelikož jej lze vypočítávat paralelně, protože ve stejném čase bude mít vždy stejné hodnoty.

Pro vytvoření Perlinova šumu jsou potřeba dvě komponenty: generátor náhodných čísel a funkce interpolace [17]. Viz obrázek 2.10.

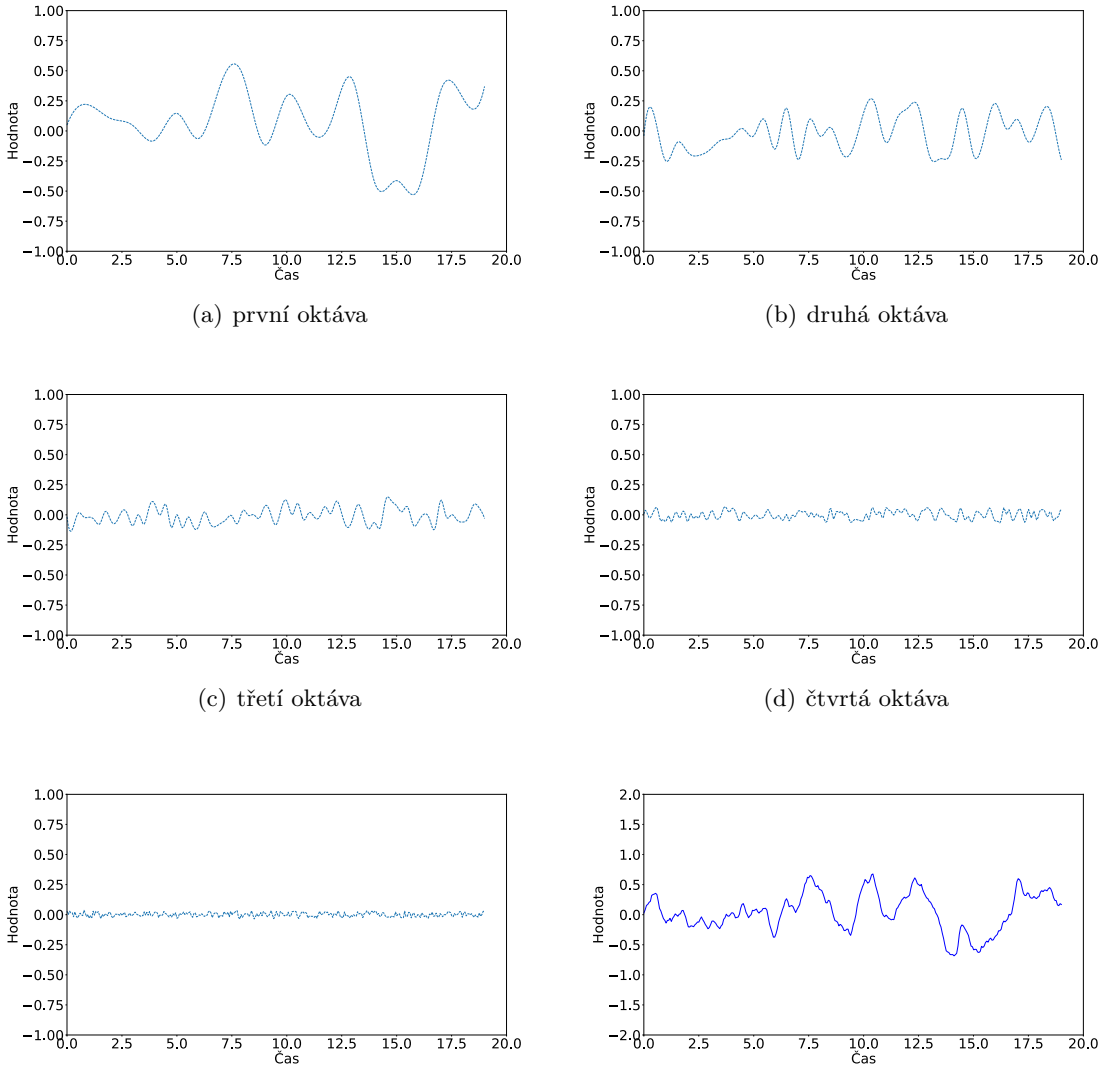


(a) lineární interpolace

(b) kubická interpolace

Obrázek 2.10: Metody interpolace bodů a jejich výsledný tvar

Perlinův šum spočívá ve vytváření několika interpolovaných šumů se zvyšující se frekcencí a snižující se amplitudou, a výpočtu jejich sumy, viz obrázek 2.11.



(a) první oktáva

(b) druhá oktáva

(c) třetí oktáva

(d) čtvrtá oktáva

(e) pátá oktáva

(f) suma pěti oktáv

Obrázek 2.11: Perlinův šum

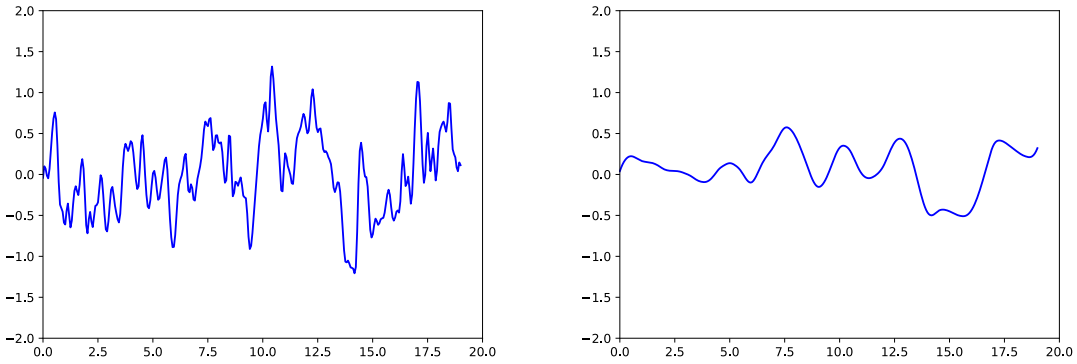
Každá další funkce, která bude k výslednému součtu Perlinova šumu přidána se nazývá oktáva. Volba počtu oktáv záleží na uživateli, nicméně od určité frekvence bude vzdálenost detailů natolik malá, že ji nebude možné zobrazit (méně než pixel). Podobně je na tom i amplituda. V momentě kdy následující oktávy mají natolik malou amplitudu že její změna nebude v hodnotě barvy znatelná je vhodné ukončit přidávání dodatečných oktáv.

Při kombinování jednotlivých oktáv může být složité zvolit správnou hodnotu amplitud a frekvencí. V předchozím příkladu (viz obrázek 2.11) je v každé následující oktávě využito dvojnásobné frekvence a poloviční amplitudy. Ve většině případů je využito právě tohoto vztahu, který se nazývá persistence. Tento pojem poprvé použil Mandelbrot [17]. Persistence určuje jak na sobě závisí následující oktávy kde frekvence f a amplituda a jsou vypočteny

na základě persistence p a iterace i jako:

$$f = \frac{1}{p^i} \tag{2.10}$$

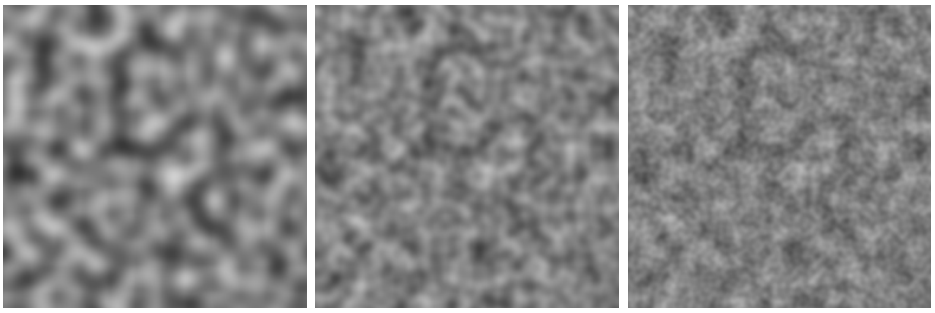
$$a = p^i \tag{2.11}$$



(a) vysoká persistence $p = 0.9$ (b) nízká persistence $p = 0.1$

Obrázek 2.12: Zvolení persistence může změnit tvar šumu. Vysoká míra persistence povede k zubatým tvarům šumu, zatímco nízká míra persistence povede k hladkým rovnoměrným tvarům.

Podobným principem změny je volba počtu oktáv. Příkladem o více rozměrech (2D) může být obrázek 2.13.



(a) jedna oktáva (b) dvě oktávy (c) tři oktávy

Obrázek 2.13: Perlinův šum v 2D a vliv oktáv na výsledný tvar

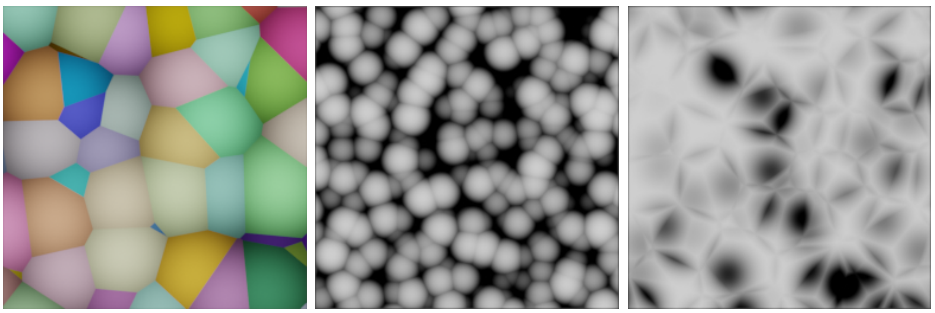
2.3.3 Worleyho textura

Worleyho textura vytváří buněčnou strukturu, založenou na vzdálenosti od příznačných bodů, které jsou náhodně rozmístěny v prostoru. Toho lze využít při generování organické kůže, zmuchlaného papíru, pohoří a kráterů. Implementace Worleyho šumu je velmi podobná řídkému konvolučnímu šumu.

Buněčná struktura, sloužící pro generování procedurálních textur, byla definována Stevnem Worleyem roku 1996 v článku „A Cellular Texture Basis Function“ [19]. V této práci

byl popsán algoritmus, který rozdělí prostor do pole buněk, čímž vytvoří texturu podobnou buňkám v biologii. Tato metoda vznikla jako doplněk k Perlinovu šumu, aby poskytovala texturu podobnou kůži či kůře stromů, viz obrázek 2.14.

Worleyho diagramy předčily své určení v procedurálním generování a byli využity k prostorové analýze, plánování městských částí, geologii, robotice a ekologii.



(a) barva nejbližšího bodu (b) vzdálenost k nejbližšímu bodu (c) vzdálenost k druhému nejbližšímu bodu

Obrázek 2.14: Ukázky Worleyho textury

Worleyho algoritmus dosahuje vysoké míry abstrakce za účely generování buněčných povrchů pomocí množiny parametrů nazývaných Worleyho konstanty, které definují operace algoritmu, přesto ale nabízí velké množství výstupů.

Při implementaci Worleyho šumu nejprve dojde v vygenerování příznačných bodů v prostoru. Následně v každém pixelu dojde k výpočtu vzdálenosti k n-tému nejbližšímu bodu. Následně bude každému pixelu přiřazena hodnota na základě této vzdálenosti. Pro vyšší optimalizaci lze prostor rozdělit do jednotlivých segmentů a vzdálenost k nejbližšímu bodu získávat pouze z blízkých segmentů.

2.4 Procedurální syntéza geometrie

Za účelem tvorby vysoce detailních přírodních modelů jako jsou třeba rostliny byly vytvořeny metody procedurální tvorby geometrie.

2.4.1 Procedurální generování budov a historické rekonstrukce

V práci „Detailed 3D Modelling of Castles“ [10] je za účely generování historických budov využito několika technik zahrnujících pozemní a vzdušnou fotogrammetrii, laserové scany a GPS. Práce také uvádí, že nejefektivnější metodou pro vytváření geometrie hradu je porovnání s půdorysy hradu, vzdušnými fotografiemi nebo laserovými scany. Rozhodujícími vlastnostmi v tvorbě modelu jsou lokace hradu, jeho rozměry, povrch zdí a střech, složitost jeho tvaru, textura a požadovaná úroveň detailu. Pro některé lokace je vhodné využít i informací z GPS pro umístění v globálním souřadnicovém systému za účely integrace digitálního terénního modelu. Při využití laserového scanu je jeho hlavní nevýhodou nedostatek vysoce kvalitních texturových informací, které jsou dodatečně získávány fotoaparátem.

Klíčový a poslední krok v integraci modelů je přenos všech modelů do referenčního systému s využitím kontrolních bodů a redundantních dat v překryvech geometrie. Registrace digitálního terénního modelu lze vytvořit pomocí dat z GPS.

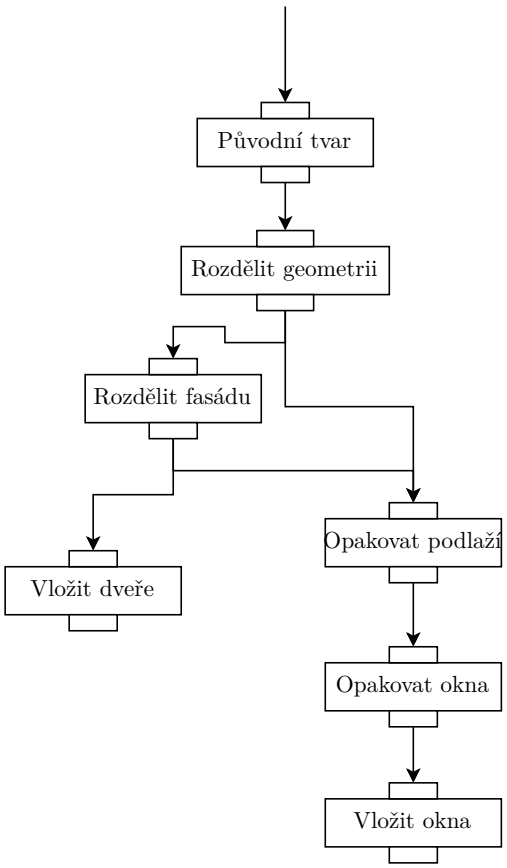
Na práci „A 3D Model of Castle Landenberg (CH) from Combined Photogrammetric Processing of Terrestrial and UAV based Images.“ [18] je zajímavá zejména rekonstrukce a vizualizace hradu za využití Blenderu, který byl použit k mapování textur na vytvořený model. Z výsledných záznamů scanů z „dronu“ byly záznamy převedeny do černobílé barvy využitím pouze zeleného kanálu, který obsahuje nevíce informací.

2.4.2 Generování budov za pomocí gramatik

Z práce „A Perspective on Procedural Modeling Based on Structural Analysis“ [11] je patrné že zatímco nástroje sloužící pro generování moderních budov existují, generování historických budov není ještě plně prozkoumaná oblast a z hlediska zachování záznamů o památných budovách a celkově historické architektuře je výhodné vytvářet tyto nástroje. Jejich využití lze uplatnit jak v plánování struktury měst, modelů do videoher a filmů či jako rozšíření GPS navigačních služeb.

V této práci je také využito procedurálního generování za využití gramatik. Koncept tvarové gramatiky je definován následovně:

Základní pravidlo (v angličtině seed) je následováno iterativní aplikací transformačních pravidel. Každé pravidlo začíná specifickým označením, tzv. předchůdcem, který vybere tvar vstupu (část budovy) a následovně další pravidla, zvané následovníci, mění geometrii a tedy tvar budovy. Aplikováním těchto pravidel je vytvářen graf, na základě kterého je vytvářena konečná geometrie objektu, viz obrázek 2.15.



Obrázek 2.15: Gramatické generování budov

Další experimentování této práce spočívalo ve využití fyzikálních simulací, za účelem testování stability generované geometrie. Využívali podobné mechaniky jako zedníci stavějící katedrály ve středověku. Tedy stavba klenby katedrály bez pojiva, tak, aby tření jednotlivých kamenů zajišťovalo stabilitu. Následně byly provedeny experimenty, ve kterých se měnily parametry klenby, přidávaly se dodatečné stabilizační prvky a testovalo se zda dojde k poškození struktury, bude dosaženo stability či naopak dojde ke kompletnímu kolapsu.

Na základě těchto poznatků lze zkoumat jakým způsobem, v jakém pořadí a jakými metodami bylo dosaženo konstrukcí skutečných historických budov a umožnění hlubšího porozumění těchto metod za účely procedurálního generování budov.

V práci „Procedural Modeling Historical Buildings for Serious Games“ [3] se autoři zaměřují na vytváření realistických modelů historických budov a částí měst. I v této práci je, podobně jako v předchozí, využito gramatik pro tvorbu geometrie. Nejčastějšími pravidly jsou dělení, opakování, rozdělení komponent a vložení, který nahradí část objektu za předmodelovaný objekt.

Jelikož část historické architektury nelze zrekonstruovat následky přestavby, modernizaci či zničení budov lze využít již existujících budov, v okolí, k extrapolaci ostatních architektonických prvků.

Jelikož je tato metoda využívána pro tvorbu modelů videoher je nutné zachovat rozumnou geometrickou složitost modelů, aby bylo možné větší množství modelů vykreslovat v reálném čase. Za těmito účely jsou modely udržovány v nižší grafické složitosti a detaily jsou řešeny pouze texturou.

Následujícím článkem studia byla práce „A Hierarchical 3D Reconstruction Approach for Documenting Complex Heritage Sites“ [9]. Zde je definováno generování historických budov ve třech typech. Prvním z nich je rekonstrukce na základě geometrických výkresů budov, u kterých nelze využít modelování na základě senzorů. Druhým je kombinovaná metoda využití jak senzorů tak geometrických výkresů. Třetí, poslední a nejvíce žádaná metoda je využití senzorů pro většinu geometrie a využití výkresů pro chybějící části budov. Jako několik předchozích metod i tato využívá fotografii k rekonstrukci částí budov a výkresů k částem, které nelze nafotit. Podobně jako několik předchozích prací i tato převážně spočívala ve skenování příznaků grafického modelu a nezabývala se velmi procedurálním generováním.

Ukázkou procedurálního generování v Blenderu je demo „Procedural Building“ [1] viz obrázek 2.16. Toto demo umožnuje vytvořit budovu a měnit parametry počtu pater, šířku a hloubku budovy, tvary jednotlivých oken, dodatečné detaily jako parapety, stínidla a květináče u oken. Na základě tohoto dema lze vytvářet podobnými metodami i hrad.

[1] Dostupné na https://www.blender.org/download/demo-files/

Obrázek 2.16: Procedurální budova z demo souboru „Procedural Building“

2.5 Architektura hradu

V práci „Architectural Modelling of Alternatives for Verification of New Interventions on the Example of the Romanesque Palace at Spiš Castle in Slovakia“ [12] je detailně popsána existující architektura a hypotetická struktura v dnešní době již zříceniny hradu Spiš na Slovensku. V této práci je popsána teoretická rekonstrukce hradu a jeho architektura.

2.5.1 Struktura

Na základě informací z webu „Castles and Manor Houses“ [2] a „History Rebuilt“ [8] se hrad skládá z opevnění, do kterého jsou zasazeny obranné věže, aby lučištníci mohli bránit opevnění.

Na zdi jsou také umístěny ochozy, aby se obránci mohli snadno přemísťovat v rámci hradu. Opevnění je opatřeno cimbuřím, které brání před nepřátelskými projektily a zároveň poskytuje otvory, ze kterých je možné střílet na útočníky, viz obrázky 2.17.

V jádru hradu se nachází tvrz. Jedná se zpravidla o sídlo panovníka. Tvrz je stavěna tak, aby poskytla poslední úroveň obrany, ale zároveň také slouží jako obydlí. Zpravidla se v ní nachází velké sály, užívané při společenských událostech.

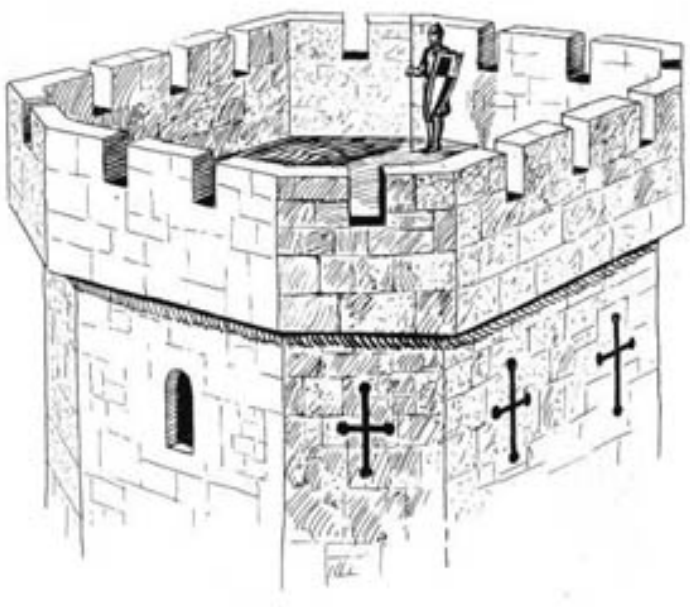
Nádvoří je obehnáno obvodovou zdí (v angličtině „curtain wall“), tyto zdi slouží ke spojení věží, aby umožnili obranu území za hranicemi hradu.

Uvnitř obvodových zdí se často nacházely budovy, které byly strategicky důležité pro fungování hradu v případě obležení.

Obrázek 2.17: Opevnění hradu Helfštýn

2.5.2 Opevnění

Opevnění bývá často opatřené cimbuřím, ozuby cimbuří (v angličtině „merlons“) sloužily k ochraně obránců před šípy, zatímco střílny (v angličtině „crenels“) jim umožňovaly střílet po útočnících. Některé cimbuří jsou opatřeny ještě dodatečnými výklenky, které poskytovaly ještě větší míru obrany. Tyto výklenky byli širší uvnitř hradu a zužovali se směrem ven. Výklenky jsou nejčastěji vertikální, aby nabídli jednomu střelci velké množství úhlů palby. Méně často se pak vyskytují horizontální verze, které sloužily pouze v místech, kde pohyb útočníků byl omezen okolním prostředím. Další variací byla kombinace těchto dvou, kde výklenek měl tvar kříže, které byly nejčastěji využity pro střelce z kuše, viz obrázek 2.18.



Obrázek 2.18: Střílny sloužící pro obranu střelců [2]

[2]Obrázek získán z webu: https://www.castlesandmanorhouses.com[2]

Ve středověku byly poměry mezi ozubím a střílnami 3:1. Počínaje 13. století byli střílny opatřeny dřevěnými závěrkami, které mohly být otevřeny při palbě a při nabíjení opět uzavřeny, viz obrázek 2.19.

Obrázek 2.19: Závěrky[3]

Opevnění bylo také opatřeno širší podstavou, to mělo hned tři důvody. Za prvé ztěžovala možnost prorazit nebo podkopat zeď, díky její velké hmotnosti. Za druhé ztěžovala využití žebříků při obléhání a také bránila proti obléhacím věžím, které mimo výšku, vyžadovaly i délku, potřebnou k překlenutí základny zdi. Za třetí umožňuje obráncům vrhat kameny, které se o základnu zdi roztříštily a zasypaly útočníky šrapnely, viz obrázek 2.20.

[3]Obrázek získán z webu: https://www.castlesandmanorhouses.com[2]

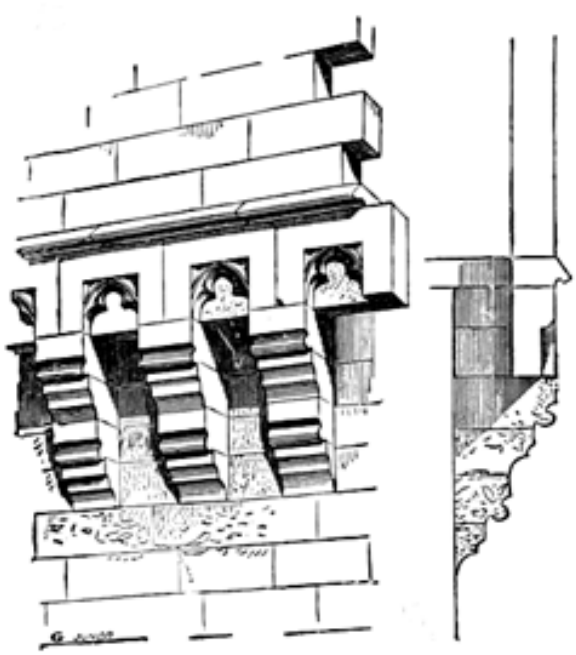
Obrázek 2.20: Podstava [4]

Zdi byly také často opatřeny ochozy, které obráncům umožňovaly hlídkovat, či poskytovaly výhodná místa k ostřelování.

Některá opevnění také měla dřevěný element na ochozech, který umožňoval dodatečnou ochranu. Jelikož byly ze dřeva, v časech bojů je bylo možné rychle sestavit. Zároveň jim díky tomu hrozilo nebezpečí vzplanutí, aby obránci tomuto předcházely byly často pokryté kůží čerstvě poražených zvířat.

Evolucí těchto opevnění bylo podsebití. Jedná se o otvory vespod ochozů, ze kterých lze shazovat kameny na útočníky. Podsebití se nachází ve vysuté pozici. Zajímavostí je že se těmto otvorům přezdívá vraždící díry. Oproti dřívějším dřevěným variantám mělo výhodu ve své permanenci, nehořlavosti, pevnosti a faktu že podlamovalo psychiku útočníků, viz obrázek 2.21

[4]Obrázek získán z webu: https://www.castlesandmanorhouses.com[2]



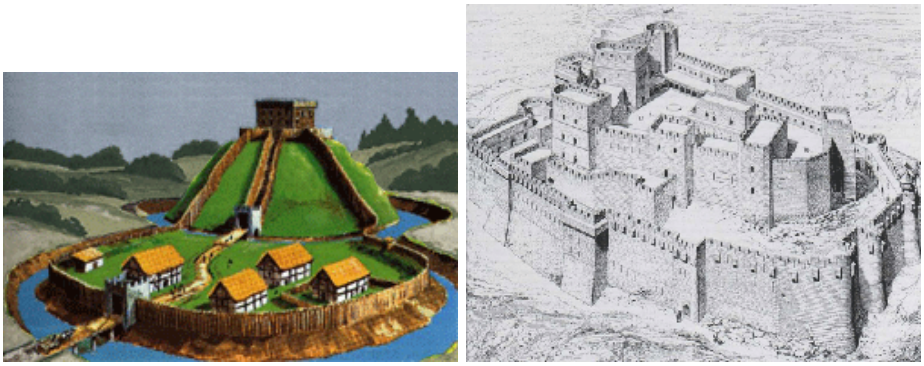
Obrázek 2.21: Podsebití [5]

Častěji se ovšem vyskytuje pouze lokalizovaná verze podsebití, která se nachází nad kritickými místy jako jsou brány. V pozdější letech, během širokého užití střelného prachu, již byly otvory uzavřené. Vysuté elementy opevnění však dále sloužily k okrasným účelům.

2.5.3 Tvrz

Asi nejznámější formou hradu je tvrz na kopci obehnána ochranou zdí a hradním příkopem [2, 8]. Tvrz či věž může být tvořena dřevem, či kamenem a kopec může být přírodní, či člověkem tvořený.

Zdokonalenou formou tvrze je citadela. Citadela je poslední linií obrany, poté co hrad a v něm obsažené budovy padly do rukou nepřítele, viz obrázek 2.22.



(a) Zeď a příkop (b) citadela

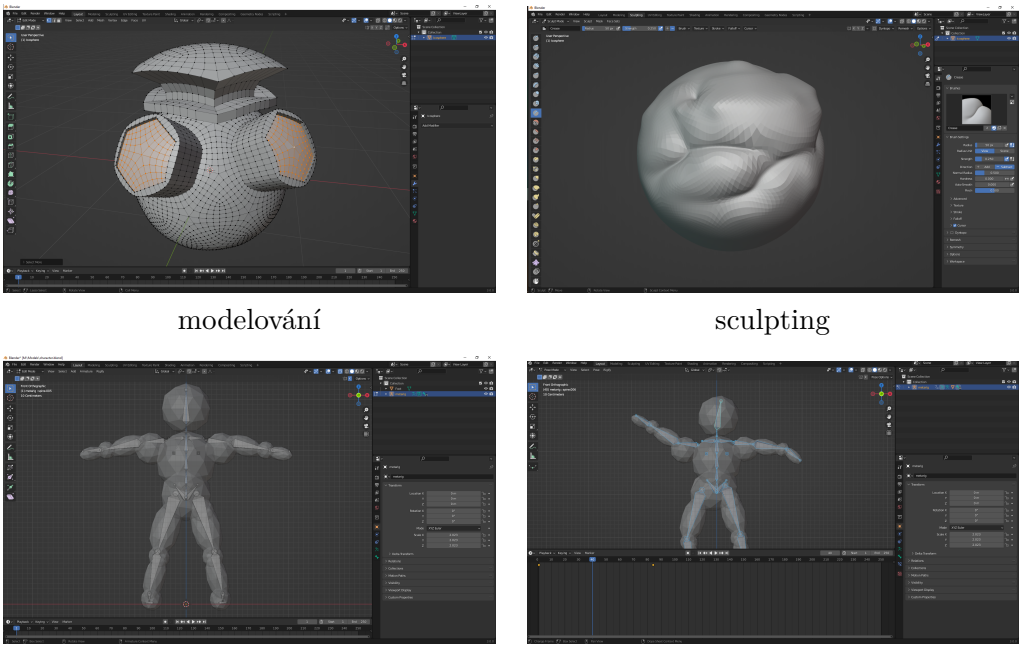
Obrázek 2.22: Hrady [6]

[5] Obrázek získán z webu: https://www.castlesandmanorhouses.com[2]

2.6 Blender

Blender je open-source software sloužící k 3D modelování. Je distribuován pod licencí GNU GPLv2. První verze Blenderu byla vydána v roce 1995. Následně v roce 2002 vznikla Blender Foundation, jako nezávislá veřejná benefiční organizace. Jedná se o nejpoužívanější open-source software pro 3D modelování, který má velmi aktivní komunitu.

Blender poskytuje jak nástroje pro konvenční modelování, jako je 3D modelování, sculpting, rigging a animování tak nástroje pro procedurální generování (viz obrázek 2.23). Asi nejsilnějším nástrojem pro procedurální generování jsou geometry nodes. V nejnovější iteraci Blenderu 3.0 umožňují manipulaci geometrie na základě jednoduchých pravidel. Hlavní nevýhodou geometry nodes je nemožnost implementace procesových smyček, což neumožňuje snadnou tvorbu fraktálů či L-systémů. Nicméně i v současné iteraci geometry nodes lze vytvářet graficky složité a procedurální objekty.



modelování | sculpting

rigging | animování

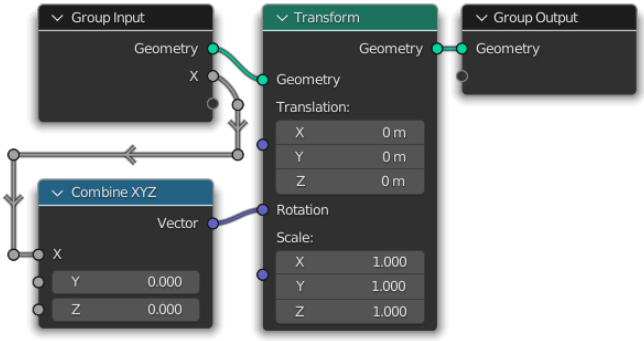
Obrázek 2.23: Využití blenderu

2.6.1 Geometry Nodes

Od verze 3.0 lze v blenderu využít geometry nodes: Fields [1]. Tato možnost umožňuje vytvářet komplexní geometrii na základě tzv. uzlů (anglicky "Nodes").

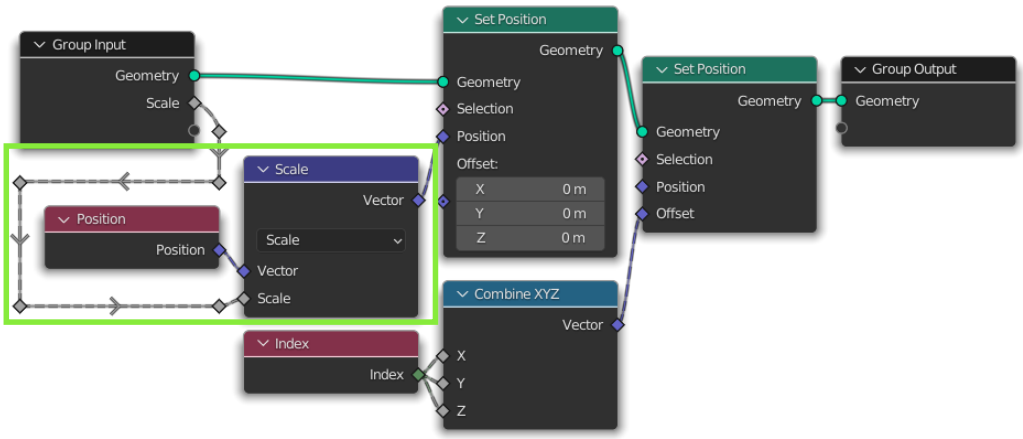
Fields byl koncipován tak, aby umožnil pracovat s geometrii podobným způsobem jakým se pracuje se shadery viz obrázek 2.24.

[6]Obrázky získány z webu: https://www.castlesandmanorhouses.com[2]



Obrázek 2.24: Geometry nodes - Tento příklad má jako vstupní hodnotu X, která bude rotovat vstupní geometrii na ose X.

Field je funkce, množina instrukcí, která dokáže transformovat libovolný počet vstupů do jednoho výstupu. Výstupy fieldu poté mohou být opakovaně vypočítávány s různými vstupy. Často jsou používány v geometry nodes pro výpočty, které mají různý výsledek pro každý element (mesh, vrcholy, strany atd.). V příkladu v obrázku 2.25 je ukázána editace vstupní geometrie pomocí operací vycházejících z hodnot pozic a indexů jednotlivých bodů geometrie.



Obrázek 2.25: Příklad modifikace na základě jednotlivých bodů, kde *Set Position* závisí na vstupu škálování, který závisí na vstupu pozice (viz zvýrazněná část), jednotlivých bodů, a transformuje je do jednotného vektoru za využití jedné instrukce.

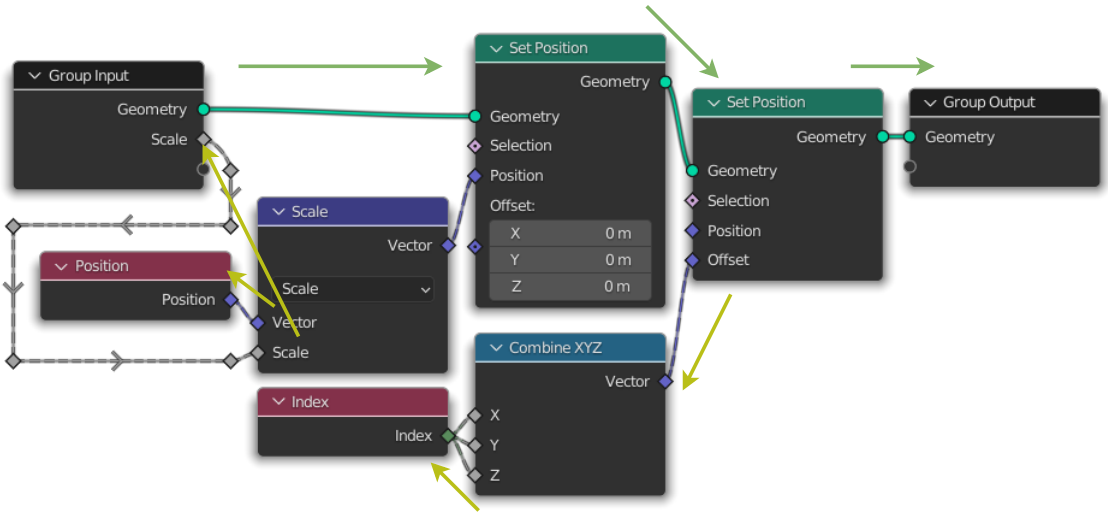
Tvary zdířek jsou využity k určení které zdířky jsou fields a které jsou obyčejná data. Existují tři typy zdířek:

- **Kolečko:** tato zdířka vyžaduje jeden reálný vstup, nepřijímá vstup fields. Pro výstup to znamená, že tato zdířka vždy vydává pouze jeden výstup.
- **Diamant:** Tato zdířka přijímá vstup fields, nebo má na výstupu fields. Konstantní hodnota může být připojena do této zdířky, ale výstup se vždy bude lišit pro každý element.

- **Diamant s tečkou:** Tato zdířka může být field, ale aktuálně se jedná o konstantní hodnotu. Toho lze využít jelikož to umožňuje sledování míst, kde jednotlivé hodnoty jsou vypočítávány, namísto pole s mnoha hodnotami. Také to znamená, že zkoumání zdířek ukáže hodnotu namísto názvu vstupu fields.

V Geometry nodes fields je využito dvou typů toků: datový tok a funkční tok. Datový tok jde zleva doprava a operuje přímo na geometrii, zpravidla je na vstupu geometrie, která je předána uzlu, který má na výstupu novou, upravenou geometrii. V node editoru je značen celou čarou jak lze vidět na obrázku 2.26.

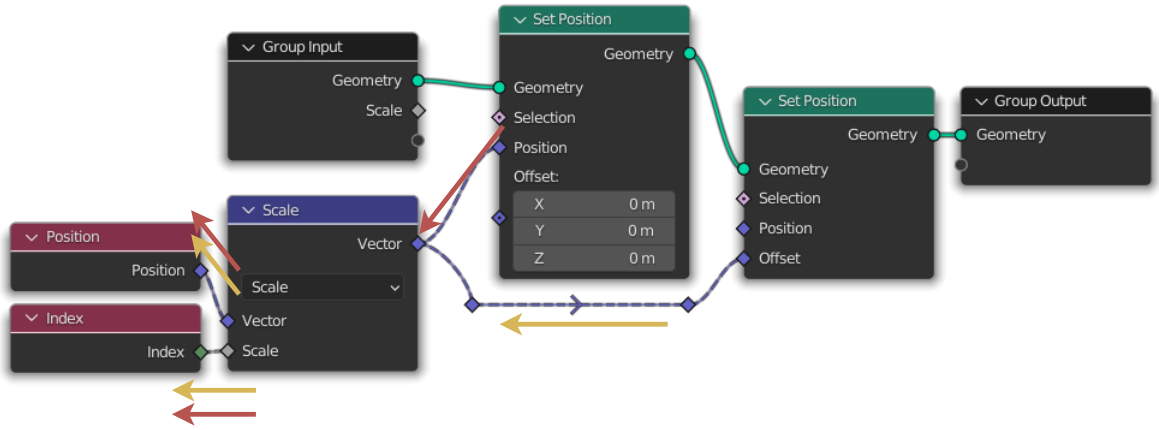
Funkční tok (v originále „Field flow“) je sekvence funkčních uzlů, které jsou předávány geometrickým uzlům a jsou čtený v opačném pořádí jako callback. V node editoru je značen čárkovanou čarou.



Obrázek 2.26: Příklad datových toků, kde zelené šipky ukazují datový tok a žluté šipky ukazují funkční tok.

Vstupní uzly předávají informace vyhodnocujícímu procesu, samy o sobě nic neznamenají slouží k předání hodnoty v kontextu datového toku (např. *Position* a *Index*), viz manuál [1].

Všechny uzly fungují v kontextu datových toků, do kterých jsou připojeny. Kontext se zpravidla skládá z geometrických komponent a atributové domény, která vyhodnocuje, jaká data jsou získávána ze vstupních uzlů. Je proto třeba si uvědomit že stejný strom uzlů, připojený do více různých zdířek nebude vždy dávat stejný výstup, jelikož celý strom uzlů bude pro každý vstup vyhodnocen znova a je možné, že v každém uzlu je jiná geometrie, na které bude operace vyhodnocena. Tato metoda je demonstrovaná na obrázku 2.27.

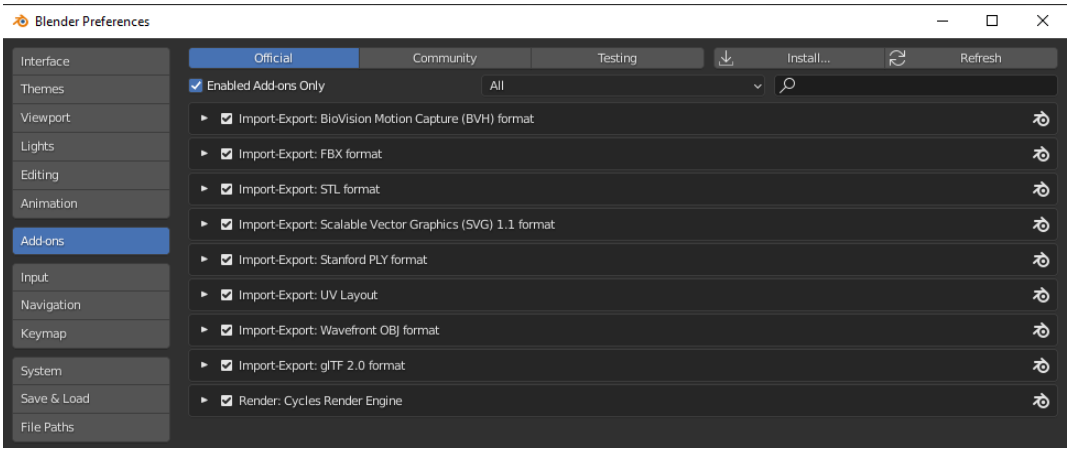


Obrázek 2.27: Příklad rozdílného vyhodnocování stejného stromu v různých kontextech, kde žlutý datový tok může nabývat rozdílných výsledků oproti červenému toku, přestože oba toky využívají stejného stromu.

2.6.2 Addony

Se zvyšující se popularitou Blenderu přibyla také potřeba tvorby dodatečných nástrojů a možnost jejich distribuce.

Addony pro Blender jsou rozděleny do dvou kategorii v závislosti na tom, kdo je vytváří a udržuje. Jsou zde addony oficiální, které jsou součástí Blenderu a lze je přidat v menu: edit → preferences → addons. Toto menu je ukázáno v obrázku 2.28.



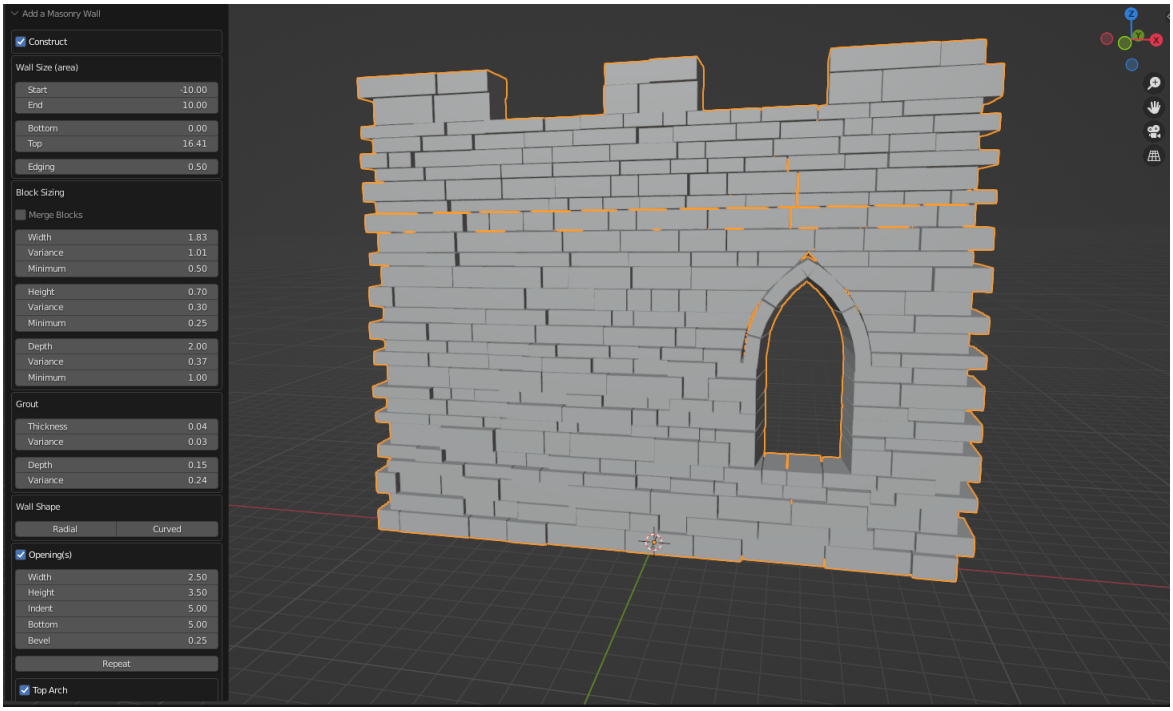
Obrázek 2.28: Menu pro instalaci addonů

Alternativou jsou addony komunitní, které jsou vyvíjeny uživateli. Komunitní addony mohou být zdarma či zpoplatněné. Komunitní Addony je možné získat na webech jako je : Gumroad[7], Blendermarket[8], Blender-addons[9] a další.

Adonny pro blender je možné vytvářet v Pythonu 3 za pomocí knihovny *bpy*, která obsahuje Blender API.

2.6.3 Extra objects Wall

Existuje oficiální addon pro Blender jménem „Add Mesh: Extra Objects“, který mimo jiné umožňuje vygenerovat zeď hradu. Ukázán je v obrázku 2.29. I přesto že tato implementace je kvalitní, má několik nevýhod, které by tato práce měla zdokonalit. Největší nedokonalostí je fakt, že takto vytvořená zeď není plně procedurální a po jejím vytvoření již nelze editovat její parametry. Zejména při tvorbě hradu je často potřeba upravovat parametry a vidět okamžitou změnu a tento způsob implementace vede k zdlouhavému procesu opakovaného vytváření. Tato práce má za cíl vytvořit takovou zeď, která bude po celou dobu tvorby editovatelná, a plně procedurální.



Obrázek 2.29: Wall factory, oficiální addon pro tvorbu hradní zdi.

[7]Gumroad: https://gumroad.com/discover
[8]Blendermarket: https://blendermarket.com/
[9]Blender-addons: https://blender-addons.org/

Kapitola 3

Výzkum v terénu

V rámci přípravy a hledání referencí k vypracování této bakalářské práce byla proveden výzkum na hradě Helfštýn, během kterého byly pořízeny fotografie sloužící k odkazům pro modelování (viz obrázky 3.1).

Helfštýn je zřícenina hradu u Týna nad Bečvou v okrese Přerov. Hrad je od roku 1963 chráněn jako kulturní památka ČR. Jedná se o hrad z počátku 14. století. Sloh hradu je renesanční a později rozšířen o gotický.

Na základě výzkumu se prokázalo, že tento hrad odpovídá struktuře z teoretické části. Dále se mi naskytla příležitost vymodelovat tento hrad jako výstup mé práce a možnost porovnat takto vytvořený výstup vůči realitě. Ve výsledku je možné využít i šablony pro generování tohoto hradu přímo ve vytvořeném addonu. Výzkum pomohl zejména v následujících úlohách:

- Získání referencí pro modelování a porovnávání výsledku.
- Možnosti prohlédnutí hradu zblízka z úhlů, které oficiální fotky často nenabízejí.
- K lepšímu uvědomění skutečných rozměrů jednotlivých částí hradu.
- K ověření části teorie, zabývající se strukturou a architekturou hradů.

Hradní zeď

Věž

Vnitřní opevnění

Tvrz

Obrázek 3.1: Helfštýn

Kapitola 4

Návrh

V rámci návrhu bylo nejprve nutné analyzovat části hradů, které bude třeba implementovat, dále zvážit způsob tvorby a možností, které Blender nabízí. Také bylo třeba navrhnout uživatelské rozhraní. V neposlední řadě bylo nutné zvážit využití metod procedurálního tvoření v 3D grafice. V následujícím textu jsou rozebrány jednotlivé metody, jejich výhody a nevýhody a volby, které byly využity ve výsledném produktu. Addon, který byl takto navrhnut byl pojmenován „CastleGen“ (zkrácená verze pro „Castle generator“).

4.1 Technologie

Blender nabízí možnosti modelování, a na základě Pythonového skriptu, vytváření výsledného modelu. Přestože tento způsob umožňuje procedurálně generovat hrad zadáváním počátečních parametrů, po vytvoření je velmi složité model dále editovat. Alternativou k tomuto řešení jsou Geometry nodes: fields, které se v Blenderu vyskytují od verze 3.0. Implementace řešení pomocí geometry nodes umožňuje editaci parametrů v reálném čase a nabízí způsob implementace ve formě grafického programování. Nevýhodou tohoto přístupu je nižší míra optimalizace, nemožnost debugování a nemožná implementace programových smyček.

Implementace v geometry nodes ovšem nabízí možnosti které addony z dřívějších verzí neumožnují. A to intuitivní a snadnou editaci parametrů, snadnou distribuci vytvořených skupin geometry nodes, a docela přehlednou strukturu vytvořených skupin, kterou je možné uživatelsky editovat i laikem.

Za účely plně procedurální tvorby hradu se tedy využití geometry nodes jevilo jako nejlepší řešení. V rámci této práce také umožnilo získat zkušenosti s relativně novým nástrojem, který stále není příliš rozšířený, a který má potenciál stát se budoucí normou procedurálního generování.

4.2 Vstupy addonu

Cílem práce je navrhnou co nejjednodušší rozhraní addonu, které bude schopen využít i začátečník pracující s Blenderem. Je tedy nutné zredukovat vstupy na nejnižší nutné minimum pro funkčnost.

Nutné vstupy tedy budou křivky, které slouží jako půdorys hradeb. Na této křivce bude následovně vygenerována zeď a pomocí parametrů geometry nodes modifierů bude editována, viz obrázek 4.1. Pro tvrz bude stejným způsobem vyžadována pouze křivka

určující tvar podstavy tvrze a vše ostatní bude nastavitelné v parametrech. V neposlední řadě bude addon nabízet možnosti vytváření hradní brány, pro niž je jediným vstupem pozice objektu, na kterém bude generována.

Samotné skupiny geometry nodes bude nutné před použitím addonu importovat ze souboru. Přestože tento krok není nutný, a bylo by možné importovat komponenty při inicializaci addonu, přidal by tento krok skupiny geometry nodes ke všem souborům, které by byli s nainstalovaným addonem vytvořeny a v případě, že uživatel zrovna nepotřebuje addon na daný projekt využívat, docházelo by k zbytečnému nárůstu velikostí souborů a presety geometry nodes využívané v CastleGenu by se zobrazovali při vkládání uživatelem vytvořených skupin.

(a) Prázdná plocha (b) Vstupní křivka (c) Vygenerovaný hrad

Obrázek 4.1: Proces fungování a výstupů addonu

Vstupy parametrů modifierů geometry nodes by také měly být co nejvíce zredukované, tak, aby nedocházelo k zbytečnému množství parametrů, které by měly malý vliv na výslednou podobu, a zároveň by snižovaly přehlednost a intuitivnost výsledného produktu. Je tedy nutné při volbě toho, které parametry budou uživatelsky nastavitelné, a které budou nastaveny implicitně, zvážit, co opravdu uživatel využije ve výsledném projektu. Dále je také nutné zajistit aby parametry, které lze implicitně vypočítat, byly vypočítány bez uživatelských vstupů, příkladem mohou být hodnoty posunu střech věží a podsebití.

4.3 Uživatelské rozhraní

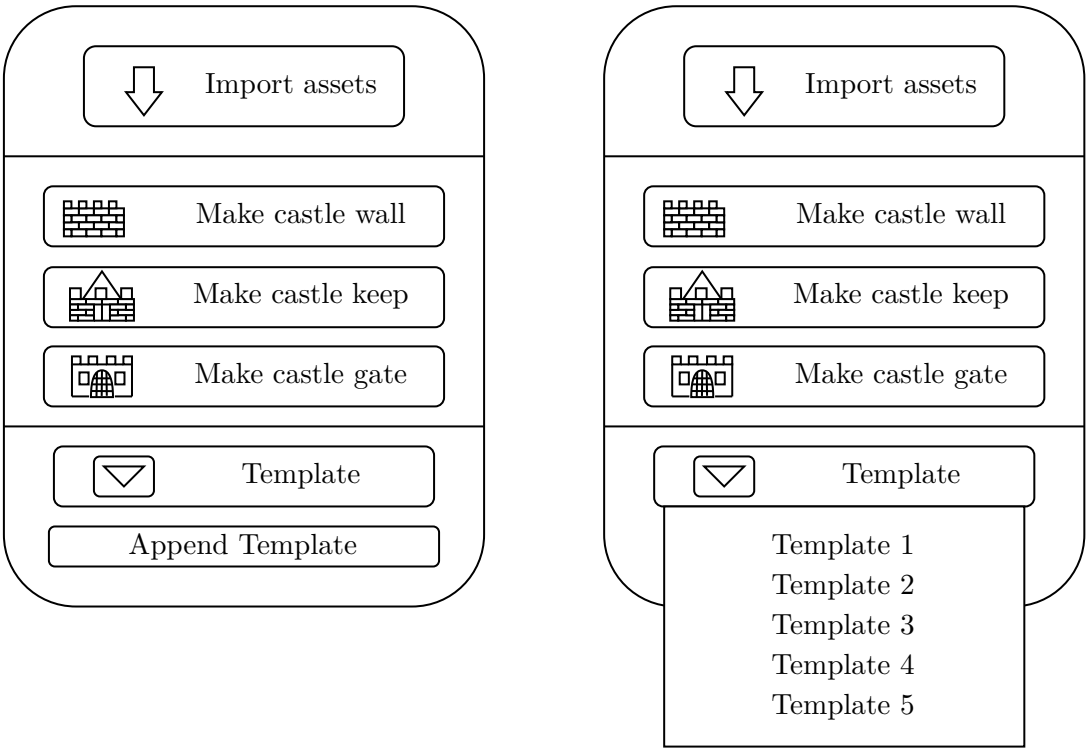
Hlavním cílem tvorby uživatelského rozhraní byla snaha dosáhnout co největší jednoduchosti a intuitivnosti pro využití addonu. Bylo nutné zvážit co všechno bude uživatel mít k dispozici a odstranit ty položky, které lze nastavit implicitně.

Jedním z prvních položek, které je třeba obsáhnout je způsob importování komponent addonu do aktivního projektu v Blenderu. Asi nejsnadnější volbou se stalo tlačítko, na nejvyšší pozici ovládacího panelu, které je po spuštění addonu jediné aktivní (tj. není zašedlé). Jeho stiskem dojde k aktivaci addonu a zpřístupní se i ostatní položky ovládacího panelu.

Dalšími položkami jsou tlačítka pro přidání hradby, tvrze a brány. Jelikož jak pro tvrz tak pro hradbu je vstupním objektem křivka nebylo možné přidávání těchto objektů řešit pomocí pouze jednoho tlačítka, které by na základě vstupního objektu automaticky vygenerovalo žádaný objekt. Bylo tedy nutné vytvořit tlačítka tři. Aby bylo zabráněno zbytečnému zmatení uživatele budou jednotlivé tlačítka zašedlé, nebudou-li vybrány správné vstupní objekty.

V neposlední řadě bylo nutné přidat možnost vložení šablony, na které bude vygenerován hrad. Jelikož ve fázi návrhu nebyl jasný počet šablon a bylo nutné zvážit možnost přidávání nových šablon, bylo vhodné využít rozbalovací nabídky. Pod ní, či vedle ní, by se mělo nacházet tlačítko, které vloží vybranou šablonu do projektu.

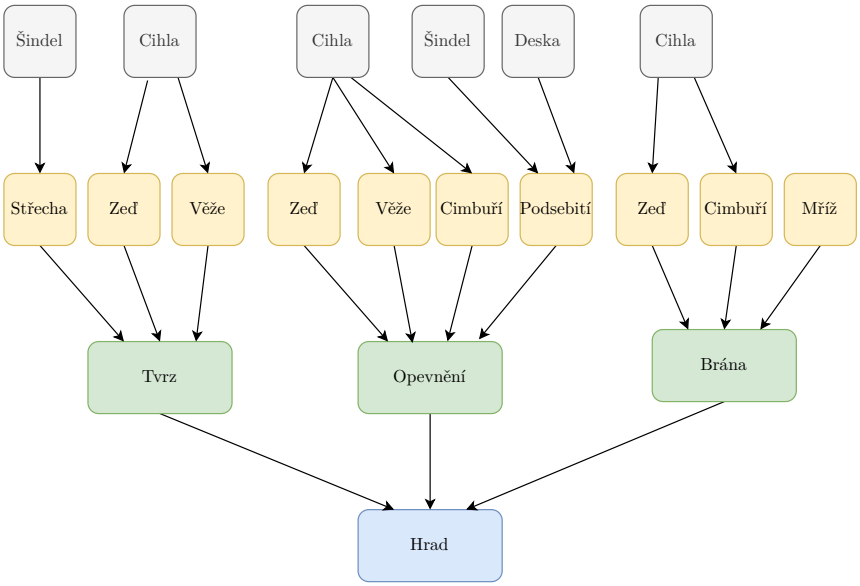
Za těmito účely byl vytvořen mockup na obrázku 4.2.



Obrázek 4.2: Návrh uživatelského rozhraní

4.4 Komponenty hradu

Pro konstrukci hradu je nutné nejprve vytvořit komponenty, ze kterých se hrad bude skládat a rozhodnout, jakým způsobem je kombinovat. Ve výsledné podobě hradu je celkový objekt složen z následujících komponent: cihla, šindel a prkno. Tyto elementy následně tvoří zdi, ochozy, cimbuří, podsebití, střechy, věže, vzpěry. Ve finále z těchto prvků lze vytvářet tvrze, brány a opevnění. Využití komponent lze detailněji popsat diagramem 4.3



Obrázek 4.3: Hierarchie komponent při tvorbě hradu

4.4.1 Cihla

Konstrukce vnitřího materiálu opevnění nebylo nutné modelovat, jelikož jej ve výsledném modelu nelze vidět. Částí, která již viditelná je jsou stěny hradeb, které bývají obehnány kameny, či cihlami. I když náhodně tvarované kameny, které vyplňují rovinu bez překryvu lze pomocí geometry nodes generovat, jejich transformace pomocí křivky je velmi složitá a výpočetně náročná. Ve výsledné implementaci bude tedy zeď generována pouze za pomocí cihel ve tvarech kvádru. Model opevnění tedy bude tvořen pouze vnější zdí.

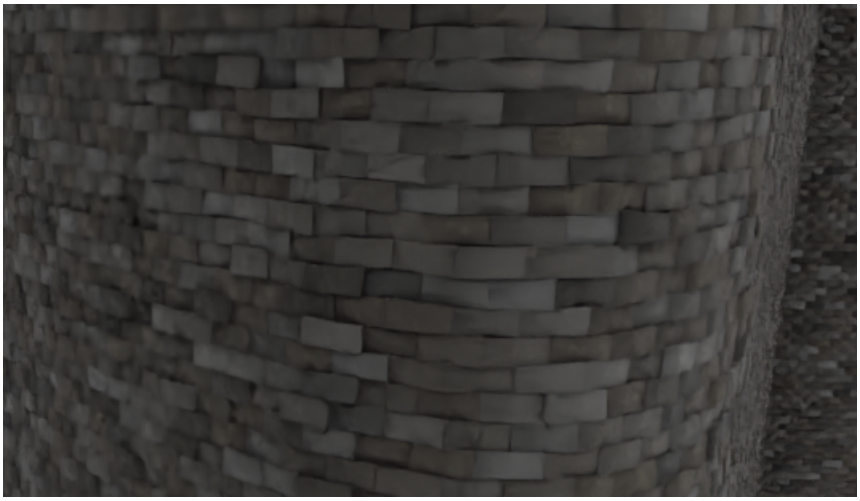
Jelikož cihel bude ve výsledné implementaci generováno několik tisíc je třeba zvážit jak detailní model cihly zvolit (viz obrázek 4.4). Samotnou cihlu lze v nejjednodušší podobě implementovat jako kostku o 8 bodech, která je následně zvětšena na jednotlivých osách. alternativou je přidání *Bevel* modifieru na takto vytvořenou kostku, tato operace ovšem znásobí počet geometrie výsledného modelu a je tedy možné že pro zachování možnosti editace v reálném čase bude tento krok volitelný či zanedbán.

(a) Základní cihla (8 bodů) (b) *Bevel* modifier (96 bodů)

Obrázek 4.4: Detail cihly před a po aplikaci bevel modifieru

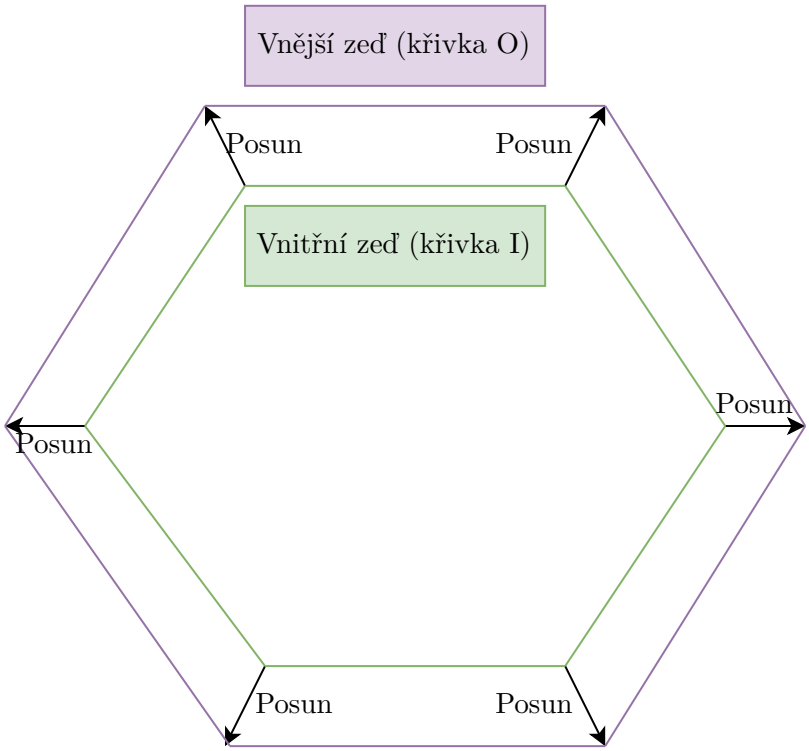
4.4.2 Zeď

Před tvorbou samotné zdi je nejprve nutné rozhodnout zda nejprve bude vygenerována rovná zeď a následně bude modifikována pomocí *curve* modifieru, či zda budou jednotlivé cihly generovány přímo podél křivky. První přístup má výhodu jednodušší implementace. Přesto má velké množství nevýhod, mezi první z nich patří nemožnost pokřivení na všech třech osách, mezi další z nich patří nerealisticky vypadající proporce geometrie v prudce se měnících směrech křivky, viz obrázek 4.5.



Obrázek 4.5: Pokřivení zdi

V případě generování cihel přímo podél křivky lze všem těmto problémům předcházet. V několika oblastech má ovšem také nedostatky. Hlavním z nich je nepřesnost v případě škálování křivky na základě jejích normál.



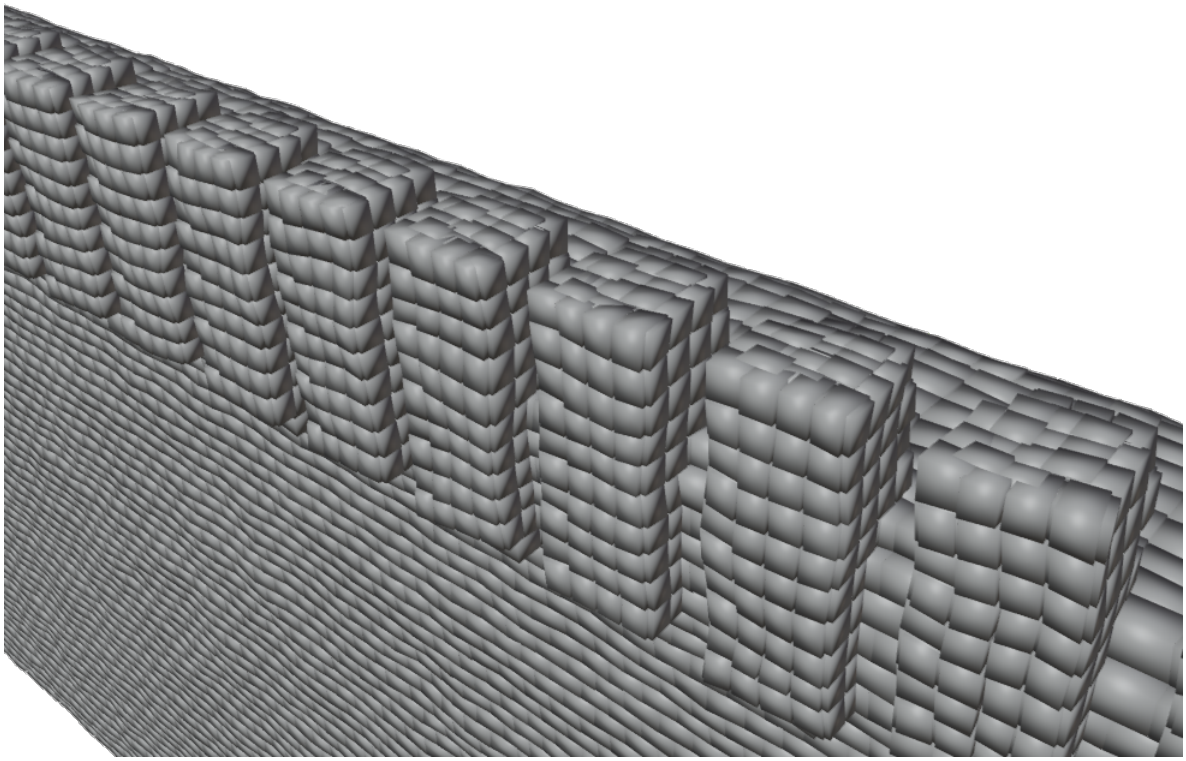
Obrázek 4.6: schéma škálování křivky: Pro opevnění je nutné vytvořit ještě jednu zeď a ochoz. Pro tvorbu sekundární zdi bude třeba zvětšit vstupní křivku I o danou tloušťku zdi. Bude stačit posunout všechny body křivky I ve směru normály škálované tloušťku zdi. Na této výsledné křivce bude inicializována druhá zeď (křivka O).

Pro tvorbu ochozu je nutné vytvořit úsečku, která bude pomocí profilové funkce při převodu křivky na mesh využita k vygenerování hrubé verze ochozu. Pro inicializaci cihel, ze kterých se bude ochoz skládat lze využít dvou různých způsobů umístění. Tím implementačně jednodušším je využití node *Distribute points on faces* s režimem distribuce Poissonovým diskem. Tato distribuce zajistí, že u cihel nedojde k překryvu, zároveň se však objevuje i možnost, že některé části ochozu nebudou obsahovat žádnou cihlu a celková struktura může působit až příliš náhodně a nedokončeně. Alternativním způsobem umístnění cihel je využití bodů, které budou získány při vytváření podstavy ochozu za pomocí profilování křivky. Jak na křivce tak na úsečce lze nastavit vzdálenost mezi body a na základě znalostí rozměrů cihel lze pak snadno implementovat inicializaci a vytvořit tak ochoz tvořený rovnoměrně rozmístěnými cihlami. Hlavní nevýhodou tohoto přístupu je fakt, že nelze jednoduše vytvořit posun cihel u sudých řad, jak tomu bylo při tvorbě zdi, či stavbě reálných hradů. A drobnou nevýhodou je že mimo rotaci a posouvání jednotlivých cihel nahoru či dolů nelze docílit efektu, náhodnosti, který by tak působil realističtěji.

4.4.3 Cimbuří a podsebití

Pro tvorbu cimbuří je třeba vytvořit několik komponent: Vzpěry, ozubí a nižší zídku, na které se ozuby nacházejí. Zde lze poprvé využít již existující node skupin, a to zeď pro tvorbu nižší zídky.

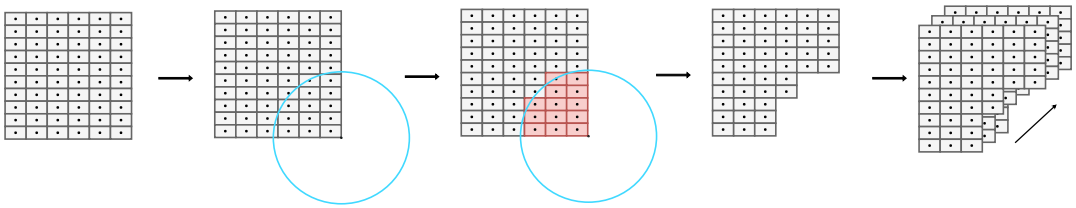
Pro jednotlivé ozubí by opět bylo možné využít node skupiny zdi. V praxi se ovšem ukázalo, že pokud neplatí že délka strany podstavy zdi, která je naprosto rovná, není násobkem rozměru cihly, dojde k zaoblování rohů, a vzhled cimbuří pak nepůsobí dobře. Nutností tedy bylo vytvořit podstavu tvořenou čtyřmi úsečkami, které ovšem nejsou spojeny. Na nich bude vytvořena zeď bez ochozu a vrcholy jednotlivých ozubů budou vytvořeny pomocí mřížky (node *Grid*), jejichž rozměry budou odpovídat rozměrům cihel.



Obrázek 4.7: Cimbuří

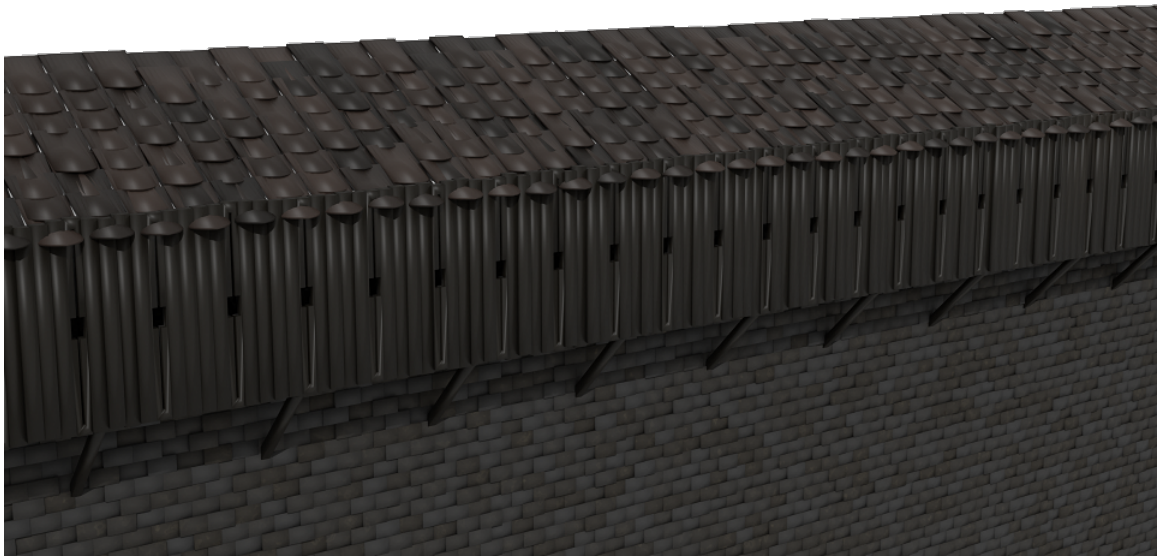
Cimbuří také bude možné odsadit ode zdi o danou vzdálenost. Celá struktura tak bude vytvořena na křivce zvětšené o velikost odsazení. Implementačně bude opět křivka zvětšena na základě hodnot normál křivky, stejným způsobem, kterého bylo využito při tvorbě vnější zdi. Cimbuří také musí být zajištěno vzpěrami nacházejícími se pod ním. Addon bude obsahovat dva druhy vzpěr, a to dřevěné v případě že hradba obsahuje podsebití a kamenné v případě, že hradba obsahuje pouze cimbuří. V případě dřevěné verze se budou vzpěry skládat pouze ze tří kvádrů uspořádaných do tvaru pravoúhlého trojúhelníku. Kamenná verze bude o něco komplexnější. Jelikož v realitě je složitější pracovat s kamenem, který je zavěšený ve vzduchu, je nutné vymyslet realisticky vypadající kamenné vzpěry. Dobře vypadajícím způsobem je vytvoření dvojrozměrné mřížky, kde vzdálenost mezi jednotlivými body je určena parametry cihel. Následně je do rohu, který je nejvíce vzdálen ode zdi (viz obrázek 4.8), umístěn kruh (v implementaci se jedná o kouli ale z teoretického hlediska se stále pracuje ve dvou rozměrech). Následně všechny body, které se nacházejí uvnitř kruhu

budou odstraněny a na zbylých bodech budou vygenerovány cihly. Celá takto vytvořená vzpěra bude následně instancována na rovné křivce, která je vzorkována tloušťkou cihel.



Obrázek 4.8: Schéma vytváření kamenných vzpěr pomocí mřížky cihel, kde cihly, jejichž středy se nachází v nižší vzdálenosti, než je hodnota pro odstranění, od vnějšího rohu mřížky, jsou odstraněny. Dále je celá mřížka instancována vedle sebe.

Další součástí opevnění kterou bylo možné vytvořit, a obohatit tím addon, bylo podsebití, viz obrázek 4.9.



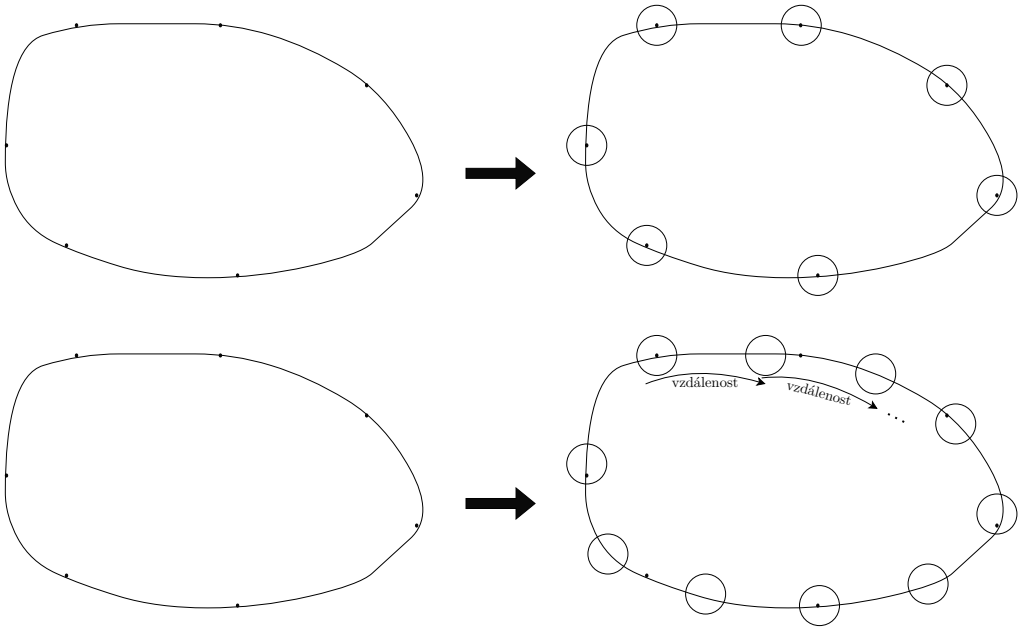
Obrázek 4.9: podsebití

Jedná se o dodatečnou stěnu a stříšky, vytvořené zpravidla ze dřeva, která lze k cimbuří přidat v časech války (viz kapitola Teorie 2). Nejprve je třeba rozdělit podsebití na menší části, a ty navrhnout. Ve výsledku návrh obsahoval tři části: ochoz, který byl tvořen deskami (implementačně se jedná o kvádry), stěnu, která je opět tvořena deskami ovšem každá n-tá deska obsahuje střílnu v křížovém tvaru. Tato střílna je vytvořena pomoci boolovského operátoru, tato metoda funguje z hlediska geometrie dobře, má ovšem nedostatek ve špatné topologii pro případ „shade smooth“ módu shadingu. Dále je nutné přidat střechu, k její tvorbě je nejprve nutno vytvořit koncept šindelů. I když by šindel bylo možné vytvořit procedurálně, jeho implementace by byla příliš složitá a výsledek by vypadal příliš jednoduše.

Z tohoto důvodu je výhodné využít šindelu vytvořeného uživatelem, nebo základního modelu šindelu, který bude obsažen v samotném addonu. Dále je potřebné vytvořit body, na kterých bude šindel inicializován. Toho lze docílit pomocí přímé křivky, která bude inicializována na všech bodech vzorkované vstupní křivky, kde vzorek bude vytvořen na základě rozměrů šindele. V neposlední řadě je třeba navrhnout trámy, které budou podpírat konstrukci. Za těmito účely stačí navzorkovat vstupní křivku na základě vzdálenosti trámů, která bude uživateli přístupná jako parametr. A v každém bodě instancovat kvádr, který bude dosahovat do výšky střechy. A nakonec vytvořit trám, který bude tvořen vstupní křivkou s profilem čtverce, který bude vyzvednut do výšky střechy a dokončí tak podsebití.

4.4.4 Věže

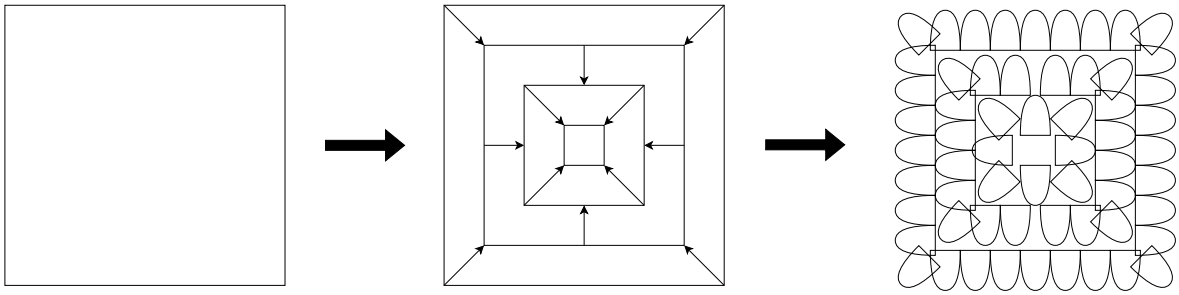
Pro tvorbu věže bude opět možné využít již existujících node skupin. Nejprve bude nutné vytvořit n-úhelník, o daném počtu bodů, které bude možné nastavit jako parametr. Na základě této křivky bude vytvořena zeď s cimbuřím. Jedinou částí, která zbývá je ochoz nahoře věže. Ve výsledném návrhu věže je opět využito již dříve definovaného cimbuří. Nicméně je využito pouze kamenné verze cimbuří, a podsebití je v případě věže zanedbáno. Ve fázi návrhu i implementace bylo zváženo přidání oken do výsledného modelu věže. V tomto případě se naskytly tři možnosti. První z nich byla možnost využití boolevského operátoru a vytváření kvádrových děr v již vytvořené věži. Tato možnost se ovšem prokázala býti neefektivní, jelikož boolovské operátory se prokázaly jako výpočetně velmi náročné operace pro množství geometrie, které bylo generováno. Druhou možností bylo umístnění oken, která by byla ručně vymodelována buď uživatelem, či jako součást addonu. Tato možnost by generovala dobře vypadající a výpočetně zvládnutelné časy, přidala by ovšem další prvek manuálního modelování, které by musel vytvářet uživatel, nebo které by narušovalo procedurální povahu celého projektu. Třetí možností bylo celkové zanedbání této části věže, jelikož tato část není příliš znatelná ve výsledném projektu a sníží se tím počet parametrů a prvků, se kterými by uživatel musel pracovat. V neposlední řadě bylo nutné zvolit body, na kterých budou věže generovány. V tomto případě se naskytaly dvě volby. První z nich bylo generování věží v daném periodickém intervalu. Tato možnost ovšem měla nedostatek ve faktu, že v případě že délka vstupní křivky nebyla násobkem periody, na konci křivky nebyla vygenerována věž a výsledek tak vypadal nerealisticky. Alternativou bylo generování věží na všech kontrolních bodech vstupní křivky, nejen že tato možnost předcházela nedostatkům periodického generování, zároveň umožnila generování věží v místech, kde dochází ke změnám směrů vstupní křivky, a v případě realistických umístění věží hradů lépe reflektuje skutečné lokace věží, viz obrázek 4.10.



Obrázek 4.10: Schéma umístění na křivce. V horním obrázku jsou věže umístěny na kontrolních bodech křivky. V dolním obrázku jsou věže rozmístěny periodicky po délce křivky

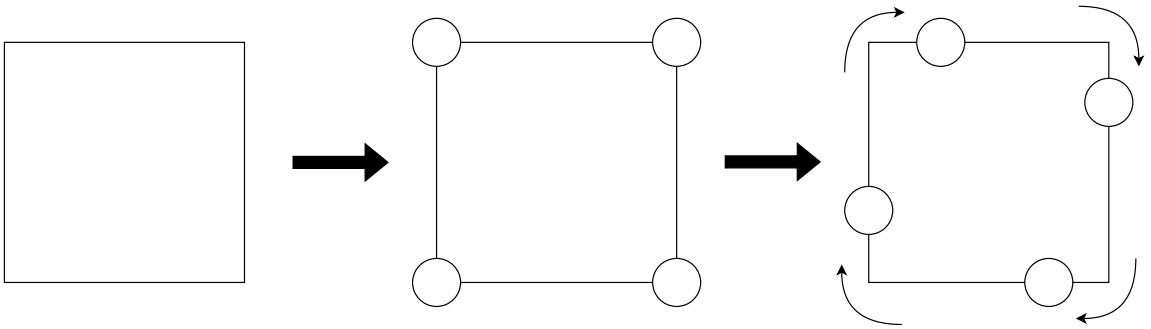
4.4.5 Tvrz

Podobně jako u zdi i u tvrze bude na vstupu křivka, která reprezentuje podstavu tvrze. Tvrz bude mít zpravidla menší rozměry oproti opevnění, nicméně princip tvorby tvrze bude podobný. I u tvrze lze znovu využít dříve vytvořených node skupin. Například okraje tvrze lze vytvořit pomocí node skupiny zdi a cimbuří. Jediným zbývajícím elementem je střecha. Střechu lze vytvořit pomocí škálování vstupní křivky na základě normály křivky a následného vzorkování jednotlivých křivek pomocí parametrů rozměrů šindelu (viz obrázek 4.11). Celek tvrze by ve výsledku bylo možné doplnit ochozem vytvořeným na základě vstupní křivky, nicméně v praxi tato metoda vypadala nerealisticky a ve výsledné podobě addonu od ní bylo upuštěno.



Obrázek 4.11: Schéma střechy. V prvním obrázku se nachází vstupní křivka. Ve druhém obrázku jsou vytvořeny další křivky škálováním vstupní na základě velikosti šindelu. Ve třetím obrázku jsou všechny křivky vzorkovány na základě rozměru šindele a na všech bodech jsou instancovány šindele.

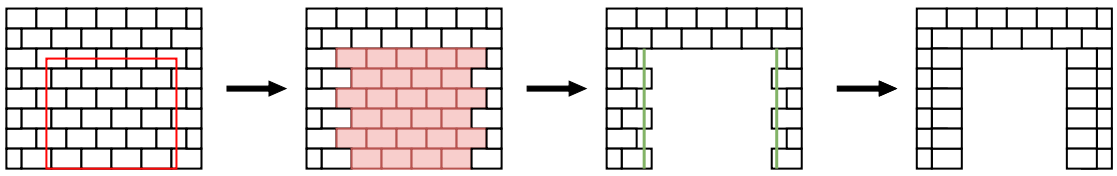
Tvrz je také obehnána věžemi. Oproti opevnění má tvrz zpravidla tvar podobající se n-úhelníku, není tedy nutné generovat věže v rozích budovy. Oproti tomu budou věže generovány v periodické vzdálenosti a uživatel bude mít možnost nastavení hodnoty počtu věží. Jelikož oproti opevnění je vstupní křivka tvrze vždy uzavřená nehrozí chybějící věž na konci křivky. Uživatel také bude mít možnost posunu věže o dané procento délky vstupní křivky, viz obrázek 4.12.



Obrázek 4.12: Schéma posunu věží, kde všechny věže jsou posunuty po křivce v jednom směru o dané procento délky křivky

4.4.6 Brána

Nejobtížnější částí hradu byla brána, nejen že musela obsahovat výseč, která byla obehnána zdí, muselo této výseče být dosaženo bez pomocí booleovských operátorů, které se v praxi prokázaly býti výpočetně velmi náročné. Nejprve bylo nutné vytvořit samotnou stěnu brány. Jelikož bylo nutné zachovat ostré hrany bylo opět použito stejné node skupiny jako při tvorbě ozubí cimbuří. Tedy čtyř nespojitých rovných křivek tvořících obdélník. Opět zde bylo využito node skupiny cimbuří, která vytvoří ochoz a ozubí. Konečně bylo třeba vytvořit výseč v geometrii brány a vytvoření geometrie zdi dané výseče. Jelikož boolovské operace byly výpočetně velmi náročné a v praxi s nimi nebylo možné editovat brány v reálném čase bylo nutné přijít s výpočetně jednodušším řešením. Tím se prokázalo být odstranění všech bodů, na kterých by byly instancovány cihly, v bráně, na základě umístění, uvnitř daného kvádru (viz obrázek 4.13). Tato možnost se prokázala býti funkční v reálném čase, nicméně zredukovala tvorbu brány na mesh o tvaru kvádru, který není vytvářen na základě křivky, jak tomu bylo u tvrze a zejména opevnění. I přes tento fakt je možné umístit a rotovat bránu takovým způsobem že ve výsledném produktu bude tohoto nedostatku zjištěno pouze detailním zkoumáním výsledného produktu.



Obrázek 4.13: Schéma Generování brány

Dále je nutné přidat mříž brány. Velmi efektivním a rychlým způsobem se ukázalo využití mřížky, která bude následně převedena z meshe na křivky, dojde tak k zachování

pouze bodů a hran. Tyto hrany pak pomocí převodu zpět na mesh s profilem nastaveným jako n-úhelník lze vytvořit procedurální mřížku, u které lze nastavit jak rozměry tak počet jednotlivých tyčí.

Kapitola 5

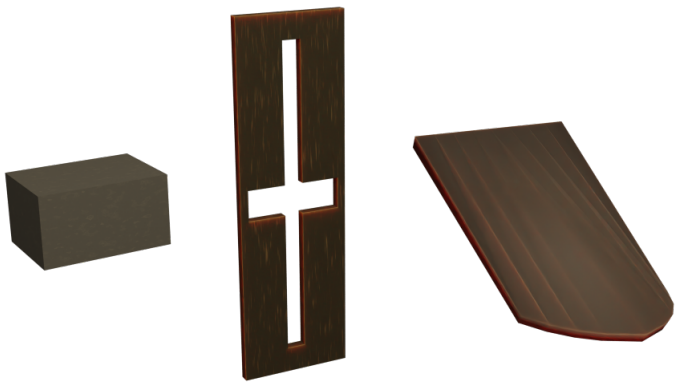
Realizace

K realizaci bylo z velké části využito geometry nodes, které jsou obsaženy přímo v Blenderu. Uživatelské rozhraní je implementováno v Pythonu 3.9, a využívá Blenderové knihovny `bpy`. Výsledný addon byl zveřejněn na webu *Gumroad.com* a je k dispozici uživatelům[1]. V této kapitole jsou obsaženy zjednodušená schémata geometry nodes skupin. Detailní schémata se nachází v přílohách. Ve schématech se vyskytují tři barvy bloků:

- **Zelená**: Vstupní parametry skupin
- **Fialová**: Hodnota field (atributy geometrie)
- **Bílá**: Bloky, které provádějící operace nad geometrii či matematické operace.

5.1 Základní struktury

Jako základ modelu hradů bylo nejprve nutné vytvořit procedurálně generované modely jednotlivých komponent, ze kterých se hrad bude skládat. Těmito komponentami jsou cihla, deska a šindel, viz obrázek 5.1.



Obrázek 5.1: Základní objekty

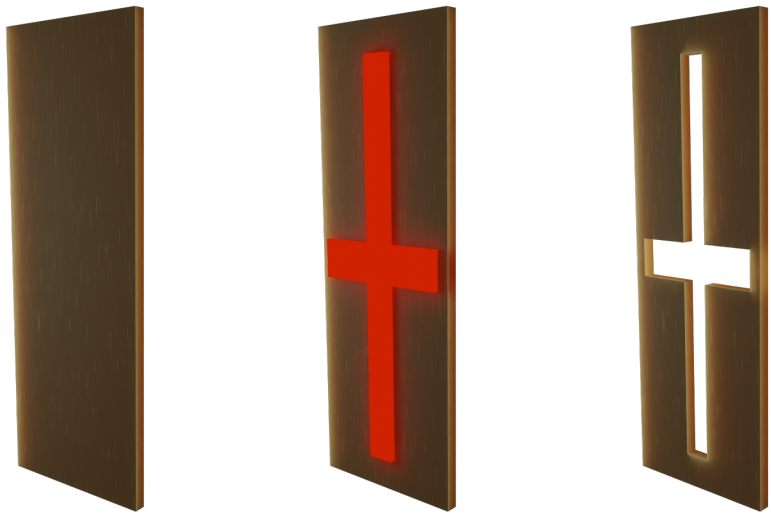
[1] Výslední addon je zdarma k dispozici na adrese: https://richardgall.gumroad.com/l/CastleGen.

5.1.1 Cihla

Pro tvorbu hradu bylo nejprve nutné vytvořit základní stavební kámen, kterým v tomto případě byla cihla. Ve finální verzi je cihla definována jako primitive objekt Blenderu `Cube`, u které jsou nastaveny rozměry na jednotlivých osách pomocí tří proměnných `width`, `height` a `depth`. V dřívějších verzích bylo k dispozici využití rovné křivky, která byla pomocí node `Curve to Mesh` s profilem nastaveným jako `Rectangle` s přidaným `Fillet Curve` pro zaoblení rohů. Tento způsob byl zvolen jelikož Blender stále nenabízí funkci `Bevel` v podobě node. Ve finální verzi ovšem bylo od detailnějšího způsobu implementace cihly odstoupeno, jelikož ve výsledných produktech docházelo k instanci řádově několik stovek tisíců cihel a zvýšený počet geometrie už výrazným způsobem omezoval výkon práce s addonem.

5.1.2 Deska

V rámci podobné jednoduchosti geometrie byly jednotlivé desky podsebití opět implementovány jako `Cube`, u které jsou nastaveny rozměry na jednotlivých osách pomocí tří proměnných `width`, `height` a `depth`. Pro desky, které měly navíc střílny byly v node skupině `CG_Panel with a hole` přidány další dva objekty `Cube`, které měly rozměry vypočteny na základě rozměrů desek vynásobené hodnotou `Hole width` a `Hole height`, které reprezentují podíl velikosti děr. Následně je díry dosaženo node `mesh boolean` s nastavením `difference`, viz obrázek 5.2.



(a) Základní deska (b) Základní deska s kvádry pro rozdíl (c) Deska se střílnou

Obrázek 5.2: Ukázka tvorby desky se střílnou

5.1.3 Šindel

Šindel byla jediná položka, která nebyla vytvořena procedurálně. Jelikož tvorba procedurálním způsobem byla zbytečně komplikovaná a výsledek nevypadal příliš dobře. Byl vytvořen základní model, který je součástí addonu. Aby bylo stále dosaženo dostatečně modularity, má uživatel možnost zadat svůj model šindele jako parametr. Aby bylo možné pracovat s libovolným šindelem byla vytvořena node skupina `CG_Object size` se dvěma vstupy, prvním

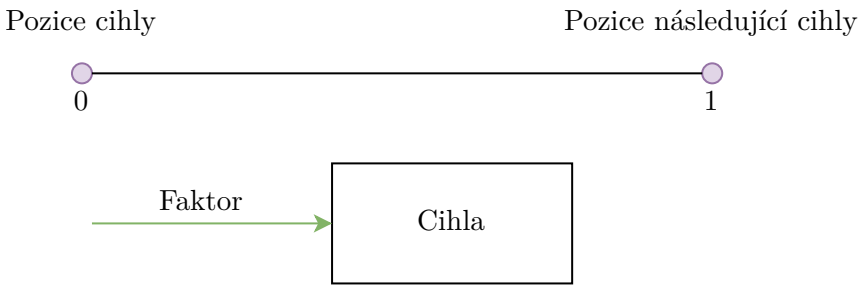
z nich je model šindelu a druhým je osa na které bude měřena velikost šindele. Jelikož nelze určit jaký bude mít šindel tvar jsou rozměry měřeny na základě bounding boxu.

5.2 Vyšší struktury

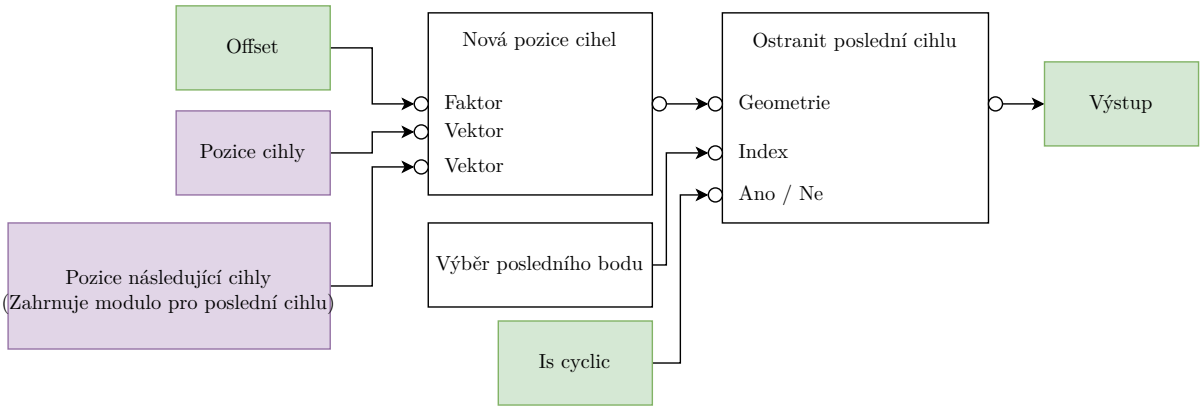
Ze základních struktur jsou následně konstruovány progresivně složitější struktury. Příkladem jsou zdi, ochozy, cimbuří, věže a podsebití.

5.2.1 Zeď

Asi nejvíce využívanou strukturou celého addonu je zeď implementována `CG_Wall`. Tato skupina implementuje jednoduchou zeď, u které jsou, na základě vstupní křivky, instancovány cihly, které jsou rotovány na základě tečen a normál křivky. Následně jsou tyto cihly skládány v řadách na sebe. Každá lichá řada cihel bude odsazena o hodnotu `Brick offset`, viz obrázek 5.3. Node group `CG_Wall` obsahuje `CG_Wall offset`, která implementuje alternativní zeď, která je odsazená o `CG_Brick offset` implementace této skupiny je znázorněna v obrázku 5.4.



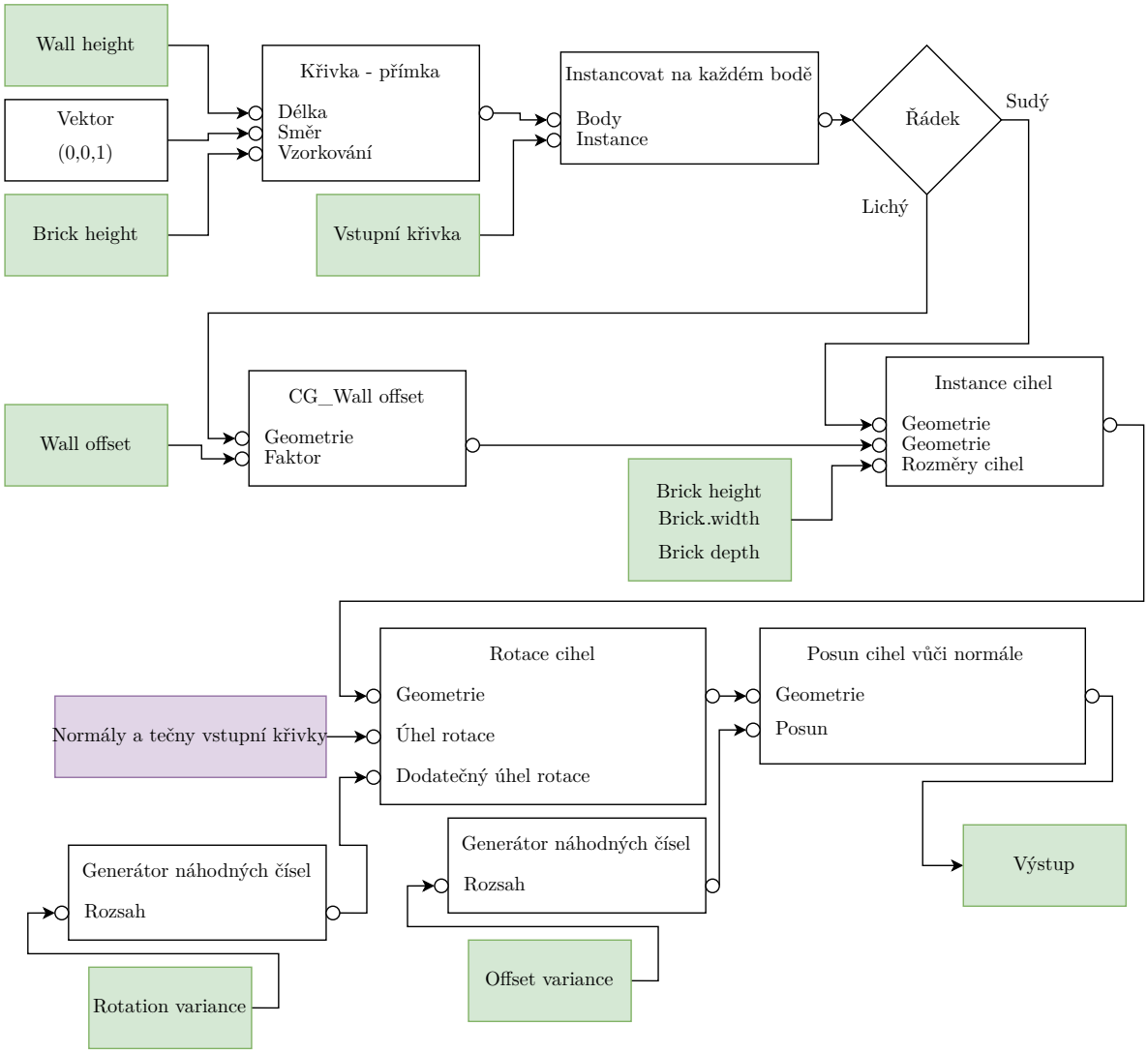
Obrázek 5.3: Odsazení cihly v lichých řadách



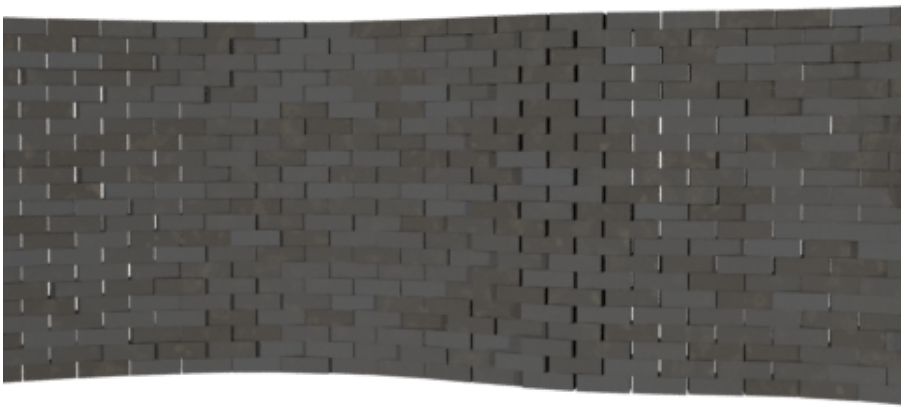
Obrázek 5.4: Schéma implementace CG_wall_offset

Následně je také vytvořena originální zeď bez odsazení. Poté je vytvořena rovná křivka, jejíž počet bodů se odvíjí od vztahu `Wall height` děleno `Brick height`. V tomto případě bude každý sudý bod křivky instancovat řadu původní zdi a každý lichý bod bude

instancovat řadu odsazené zdi. Hodnotu odsazení bude možné ovládat pomocí parametru `Brick offset`. Implementace skupiny `CG_Wall` je znázorněna v schématu 5.5. Výsledný vzhled této struktury lze vidět na obrázku 5.6.



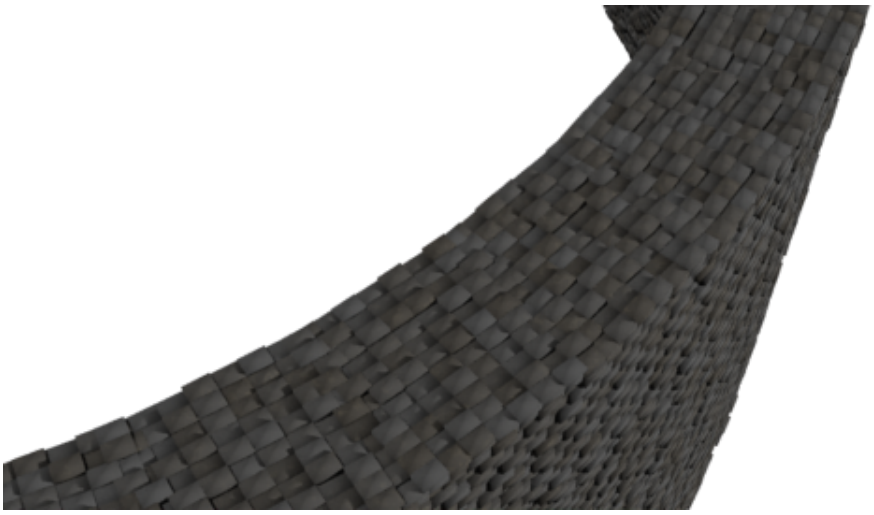
Obrázek 5.5: Schéma implementace CG_Wall



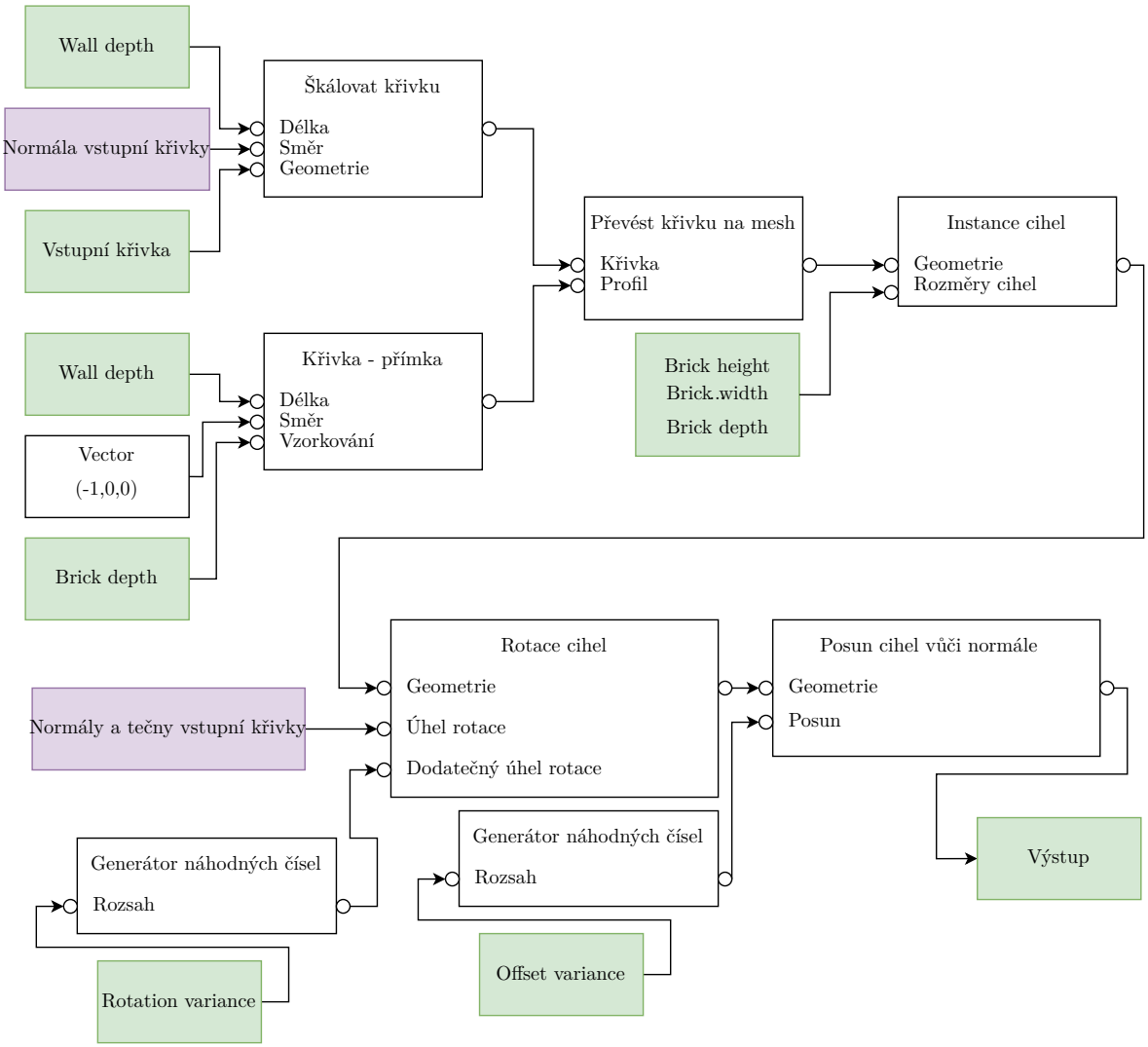
Obrázek 5.6: Render zdi

5.2.2 Ochoz

Další strukturou je `CG_Walkway`. Tato struktura představuje ochoz, který se vyskytuje nahoře, mezi vnější a vnitřní zdí. Jelikož zde nelze snadno instancovat několikrát identickou křivku s daným odsazením, protože se jedná o strukturu složenou s několika postupně se zmenšujících křivek, kde vzdálenost mezi body jednotlivých křivek je pevná hodnota. Je tedy nutné vytvořit alternativní způsob řešení. Schéma implementace ochozu je vyobrazeno na obrázku 5.8. Podoba výsledného ochozu je ukázána na obrázku 5.7.



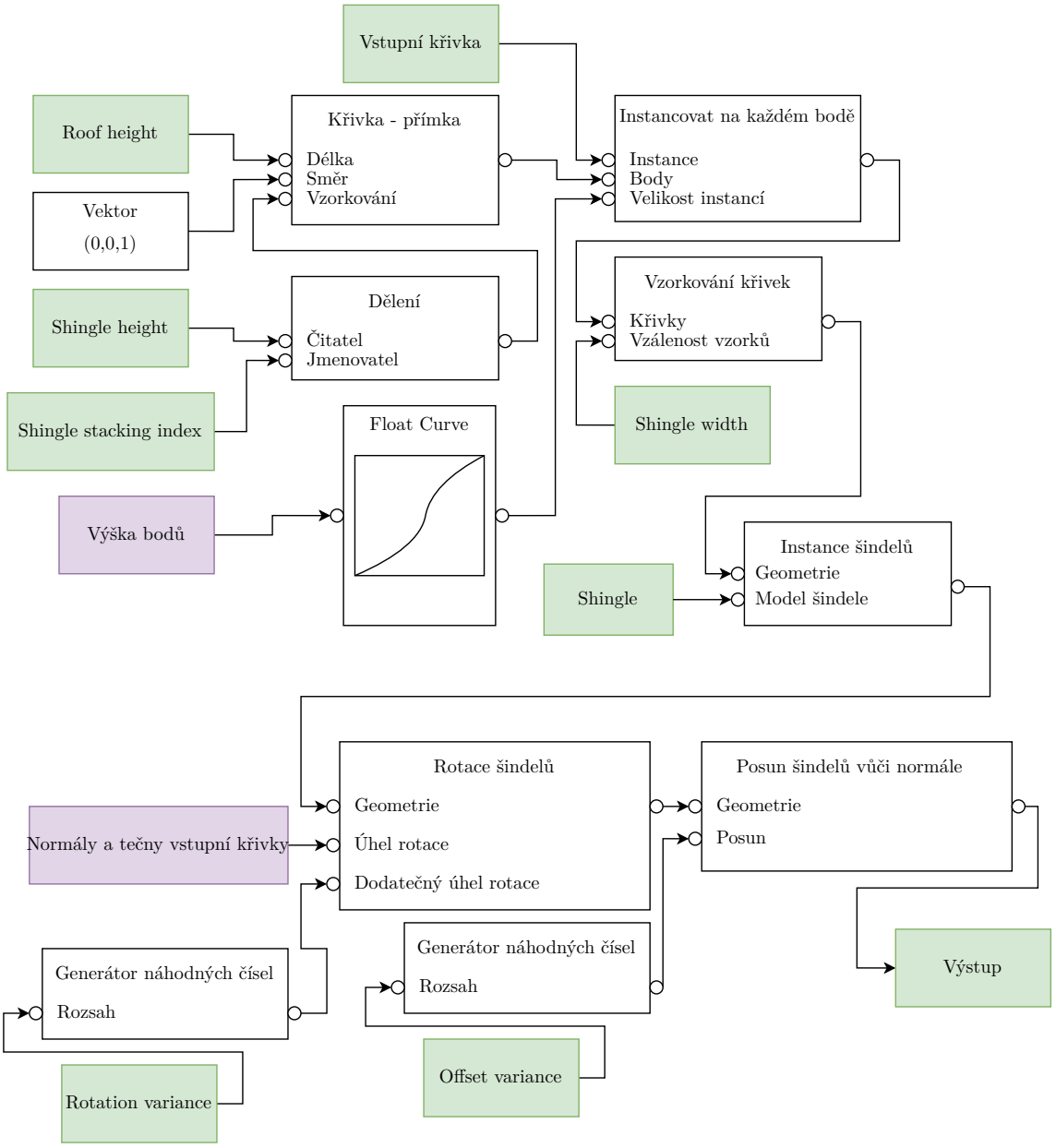
Obrázek 5.7: Render ochozu



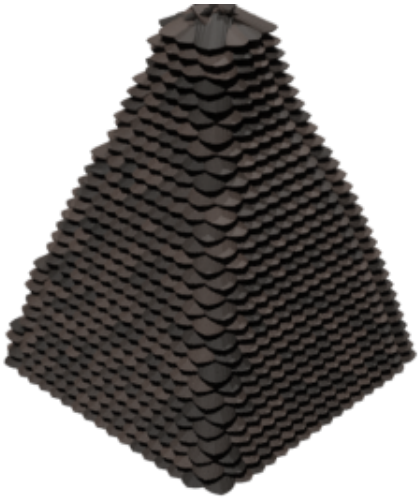
Obrázek 5.8: Schéma implementace CG_Walkway

5.2.3 Střecha

Další strukturou která je několikrát v addonu využita je `CG_Roof`. Tato struktura na základě vstupní křivky generuje vrstevnice, které jsou následně pomocí `Resample Curve` navzorkovány, a na těchto bodech jsou instancovány jednotlivé modely šindelů. Celková struktura skupiny `CG_Roof` je ukázána ve schématu 5.9. Ukázka takto vytvořené střechy je viditelná na obrázku 5.10



Obrázek 5.9: Schéma implementace CG_Roof



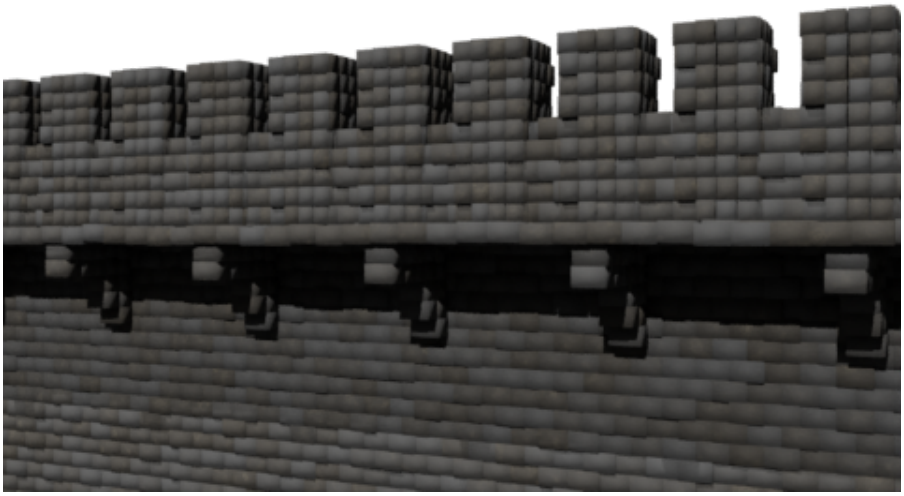
Obrázek 5.10: Render střechy

5.2.4 Cimbuří

Skupinou, která implementuje cimbuří je `CG_Main battlement`. Tuto skupinu tvoří dvě podskupiny: `CG_Battlements`, která vytváří ozuby a spodní stěnu cimbuří a `CG_Supports`, která implementuje vzpěry cimbuří. Ve skupině `CG_Supports` dochází pododbně jako v předchozích skupinách ke vzorkování vstupní křivky na základě hodnoty `Supports distance` a následně je na základě hodnoty `Battlements / Hoardings` vybráno, zda budou na jednotlivé body instancovány dřevěné vzpěry, imlementované skupinou `CG_Wooden support`, nebo kamenné vzpěry implementované skupinou `CG_Stone support`.

Skupina `CG_Battlements` je složena ze dvou hlavních částí: první implementuje cimbuří a druhá generuje podsebití. Zda bude podsebití generováno je opět určeno proměnnou `Battlements / Hoardings`. Část, která implementuje cimbuří se skládá z části spodní zdi, která je generována za pomocí `CG_Main wall`, kde `Wall width` je roven `Battlement offset` a `Wall height` se rovná polovině `Merlons height`.

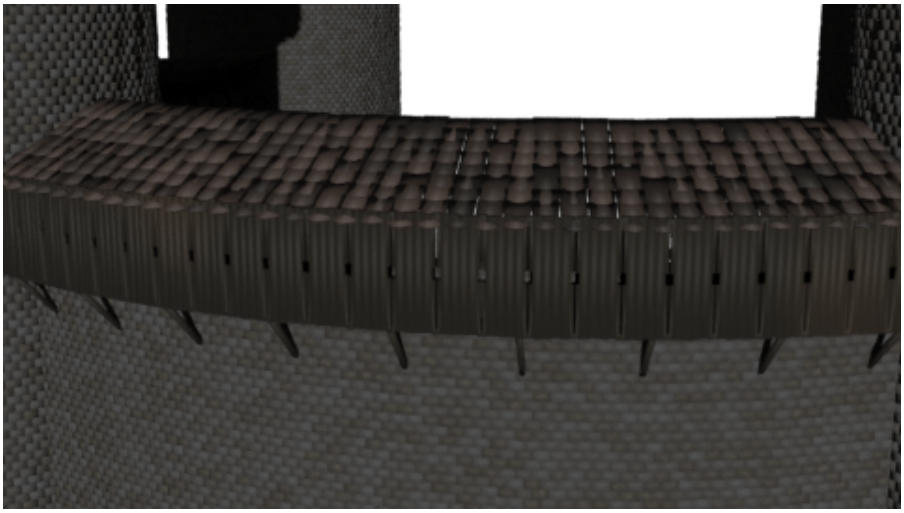
Dále je také nutné přidat spodní podstavu cimbuří jelikož `CG_Main wall` podstavu neobsahuje. Tato podstava je vygenerována za pomocí `CG_Walkway`, s hodnotou `Wall depth` rovné hodnotě `Battlement offset`. V neposlední řadě se cimbuří skáldá z ozubí. Jednotlivé ozuby jsou generovány pomocí `CG_Wall`, která má jako vstupní křivku `Quad++`. Dodatečně je ješte přidána vrchní vrstva cihel která je vytvořena za pomocí node `Grid`, na jejichž bodech jsou instancovány cihly. Ukázka vzhledu cimbuří je na obrázku 5.11.



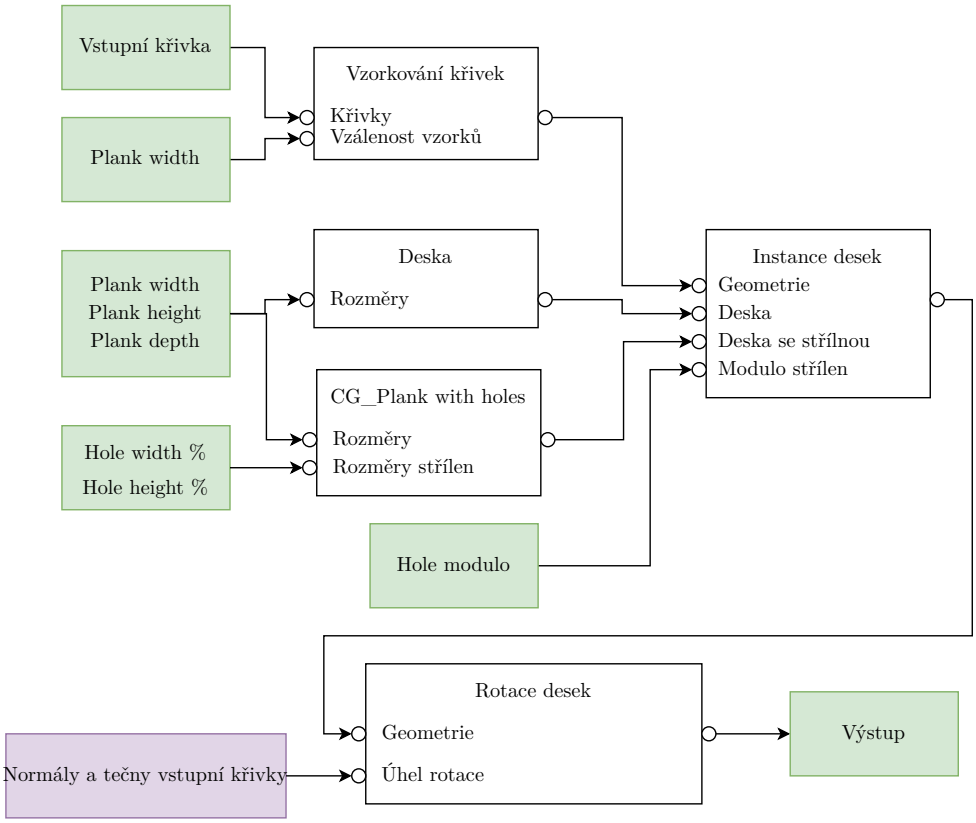
Obrázek 5.11: Render cimbuří

5.2.5 CG_Planks

Další využívanou strukturou je `CG_Planks`. Tato skupina implementuje přední desky podsebití podél zdi. V této skupině je vstupní křivka nejprve navzorkována na základě vstupního parametru `Plank width`. následně jsou na každém n-tém bodě instancovány desky se střílnou, kde n je hodnota obsažena v parametru `Hole modulo`. Na ostatních bodech bude vygenerována základní deska, která je implementačně pouze `Cube`, která má jako rozměry nastavené `Plank width`, `Plank height` a `Plank depth`. Parametry jednotlivých rozměrů střílen lze nastavit za pomocí parametrů `Hole width %` a `Hole depth %`. V případě že je splněna boolovská hodnota pro generování podsebití `Battlements / Hoardings` budou zuby cimbuří generovány s posunem `Battlement offset` rovným nule a hodnota `Battlement offset` bude sloužit k posunu pouze podsebití o danou hodnotu. Vzhled desek lze spatřit na obrázku 5.12. Implementace `CG_Planks` je popsána schématem 5.13.



Obrázek 5.12: Render Podsebití



Obrázek 5.13: Schéma implementace CG_Planks

5.2.6 CG_Tower

Implementace věže se nachází ve skupině `CG_Tower`. Základem věže je n-úhelníková křivka o N bodech, kde N je nastavitelné jako parametr `Tower sides`. Následně je takto vytvořená křivka použita jako vstup do skupiny `CG_Wall`, která vytoří zeď věže, dále je tato křivka předána do skupiny `CG_Main battlements` a `CG_Walkway`, v případě, že hodnota parametru `Roof / Merlons` je nastavena na hodnotu 1. Alternativně je křivka použita jako vstup do skupiny `CG_Roof`, která vytvoří střechu věže.

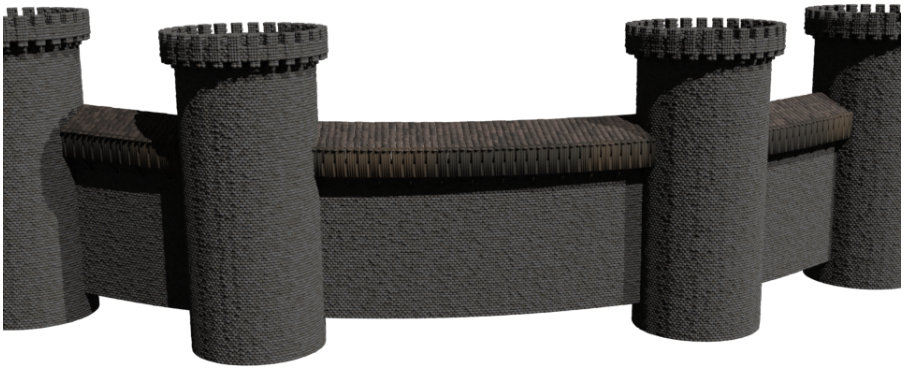
Pro instanci všech věží je využita skupina `CG_Towers`, která na vstupu má křivku, na jejichž bodech budou instancovány věže vytvořené předchozí skupinou. Tyto věže budou následně rotovány okolo osy Z na základě hodnot normál vstupní křivky. Dodatečně lze všechny věže posunou směrem dolů parametrem `Towers down offset`. Tato možnost byla přidána z důvodů stavů, ve kterých části podstav věží zůstávaly ve vzduchu v případě prudce se vlnící vstupní křivky. Tento dodatek zajistí, že věže budou mít podstavu umístěnou pod ostatní geometrii. Ukázka finální podoby věže je na obrázku 5.14.

Obrázek 5.14: Render věže

5.3 Hradní zeď

Hradní zeď je implementována v node skupině `CG_CastleWall`. Detailní popis parametrů této skupiny lze nalézt v příloze práce.

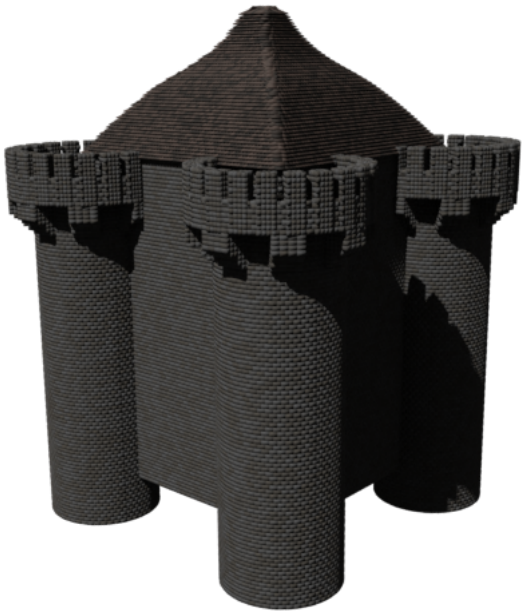
Celková struktura node skupiny `CG_CastleWall` se skládá ze tří částí: Cimbuří definované node skupinou `CG_Main battlements`, zdí definované v `CG_Main wall` a ochozem definovaným `CG_Main walkway`. Dále se struktura skládá z věží definovaných v `CG_Towers`. Render kompletní hradní zdi lze spatřit na obrázku 5.15.



Obrázek 5.15: Render hradní zdi

5.4 Tvrz

Samotná tvrz je implementována třídou `CG_CastleKeep`. Pro implementaci této struktury bylo opět využito `CG_Wall` s pevně nastavenou hodnotou `Is cyclic` na hodnotu 1, jelikož podstava tvrze bude vždy uzavřená křivka. Střecha tvrze je zde opět implementována pomocí skupiny `CG_Roof`, která umožnuje vytvořit střechu na základě vstupní křivky. Dále jsou podél křivky instancovány věže, implementovány skupinou `CG_Tower`. Oproti opevnění jsou zde věže rozmístěny rovnoměrně podél křivky a parametr `Towers count` určuje na kolik bodů je vstupní křivka vzorkována. Jednotlivé věže lze posunout parametrem `Tower offset`, tato funkcionalita je implementována skupinou `CG_Offset on curve`. Hodnoty `Tower offset` nabývají hodnot v intervalu 0 až 1. Na obrázku 5.16 lze vidět vzhled takto vygenerované tvrze.

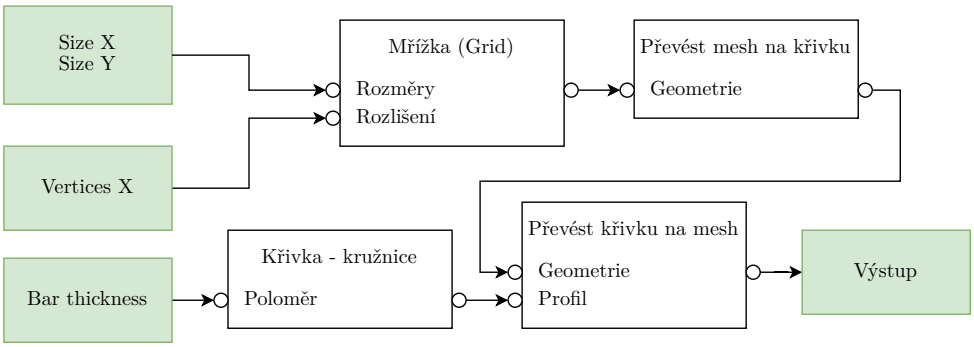


Obrázek 5.16: Render hradní tvrze

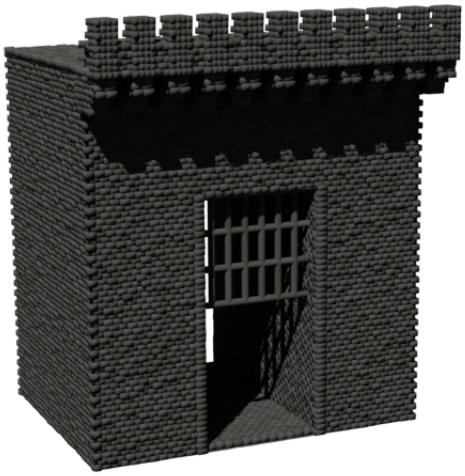
5.5 Brána

Brána je implementována skupinou `CG_CastleGate`. Pro Generování stěn bylo opět využito instance `CG_QuadWall`, na které je vygenerována vnější zdi. Pro zeď tvořící průchod, je využito další instance `CG_QuadWall`, otočené o 90°na ose X. Střecha brány je podobně jako u zdi generována pomocí `CG_Walkway` se vstupem první křivky `CG_Quad++` a šířkou ochozu rovným hloubce brány. Dále jsou ze základního obdélníkového tvaru brány na základě `CG_Box select` vybrány cihly, které se nacházejí v oblasti průchodu a pomocí node `Delete Geometry` jsou body, na kterých později dochází k instancování cihel, odstraněny. Dále je nutné přidat cihly, které tvoří rohy průchodu brány, vygenerované na okrajích kvádru, který odstraňuje cihly průchodu. Následně zbývá pouze vytvořit mříž brány. Ta je

implementována pomocí skupiny `CG_Iron bars`, jejíž implementaci lze nalézt na obrázku 5.17. Na obrázku 5.18 lze spatřit výslednou podobu hradní brány.



Obrázek 5.17: Schéma implementace CG_Iron bars



Obrázek 5.18: Render hradní brány

Kapitola 6

Zhodnocení

6.1 Zpětná vazba

V rámci tvorby práce byl výsledný addon zveřejněn uživatelům. Aby bylo možné zhodnotit jeho úspěšnost byl vytvořen dotazník, který umožní zjistit uživatelskou spokojenost a další vlastnosti o které by bylo možné addon rozšířit.

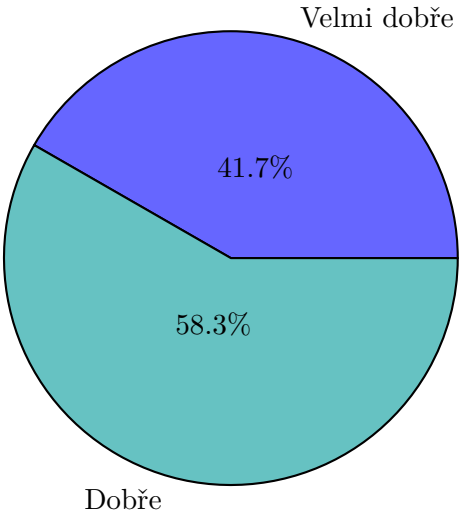
6.1.1 Dotazník

V rámci sbírání zpětné vazby byl také vytvořen dotazník pro uživatele addonu. Dotazník obsahoval 6 otázek a to:

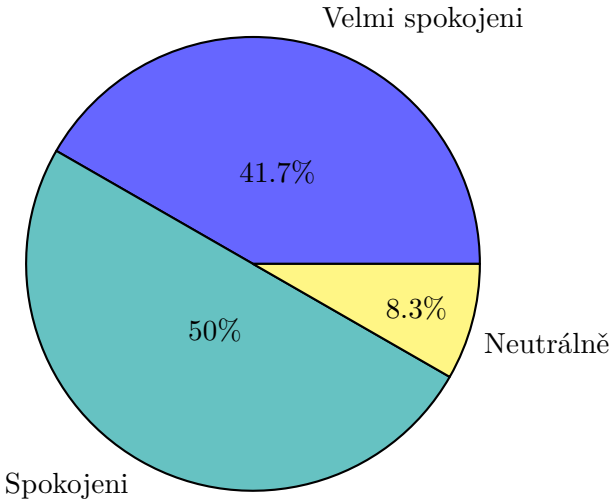
- Jak intuitivní je uživatelské rozhraní? (stupnice od 1 do 5, kde 1 - velmi špatné, 5 - velmi dobré)
- Jak jste spokojeni s funkcionalitou addonu? (stupnice od 1 do 5, kde 1 - velmi nespokojeni, 5 - velmi spokojeni)
- Co by jste přidali k funkcionalitě addonu / co vám v addonu schází? (Textová odpověd)
- Kdyby jste pracovali na projektu tvorby hradu s jakou pravděpodobností využijete tohoto addonu? (stupnice od 1 do 5, kde 1 - nidky, 5 - vždy)
- Jak realistické jsou výstupy addonu (stupnice od 1 do 5, kde 1 - naprosto nerealistické, 5 - velmi realistické)
- Jak intuitivní jsou instrukce pro intalaci a skupiny geometry nodes? (stupnice od 1 do 5, kde 1 - neintuitivní, 5 - intuitivní)

Na základě zpětných vazeb z tohoto dotazníku vzešly následující hodnoty: Dotazník celkem vyplnilo 12 uživatelů. Na jednotlivé otázky odpověděli následovně:

Otázka 1: Jak intuitivní je uživatelské rozhraní?



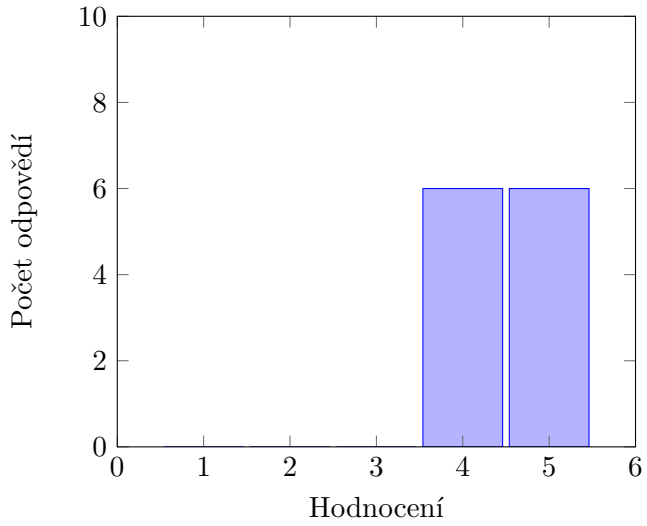
Otázka 2: Jak jste spokojeni s funkcionalitou addonu?



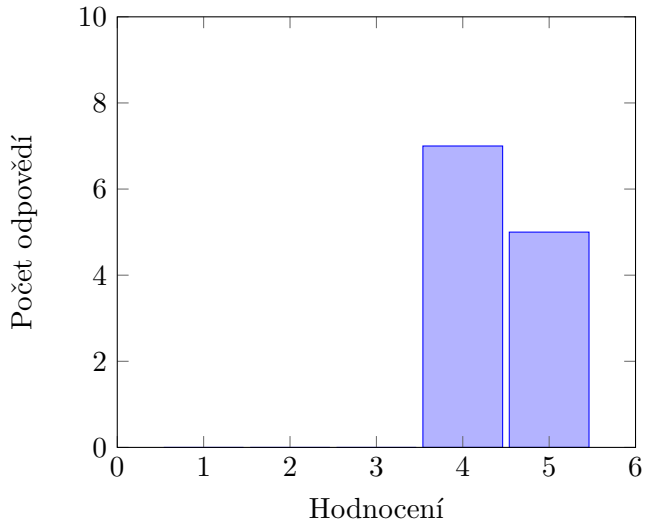
Otázka 3: Co by jste přidali k funkcionalitě addonu / co vám v addonu schází?

V případě třetí otázky se objevilo několik zajímavých návrhů, mezi které patří například přidání roviny, která by vyplňovala vnitřní část hradeb a skládala by se z dlaždic či jiných materiálů. Dalším zajímavým návrhem byla možnost přidání vlastností chátrání hradu, pomocí kterého by bylo možné generovat ruiny. Mezi nedostatky addonu uživatelé zmiňují vysoký počet polygonů ve výsledném produktu a možnost tvorby verze z nižší mírou detailu, nebo možností bakingu detailů do zredukovaného modelu. Objevily se i návrhy na rozšíření nabídky objektů addonu, mezi ně patřili okna či dveře pro tvrze a věže.

Otázka 4: Kdyby jste pracovali na projektu tvorby hradu s jakou pravděpodobností využijete tohoto addonu?



Otázka 5: Jak realistické jsou výstupy addonu?



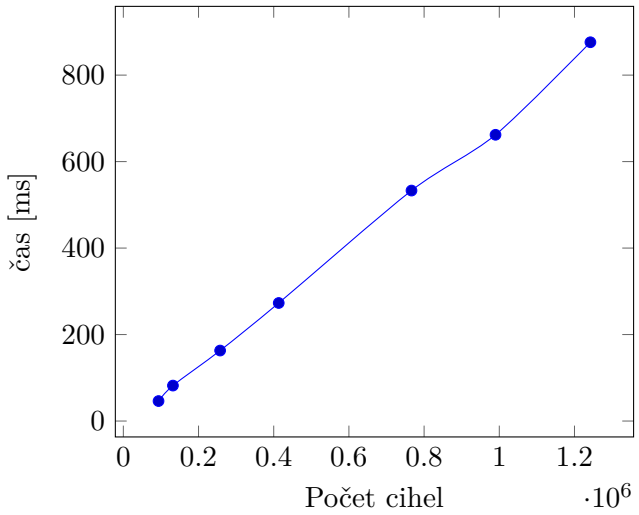
6.2 Výkon

Při měření výkonu bylo využito opevnění se základnímu parametry a jako vstupní křivka byla využita kruhová křivka s postupně se zvyšujícím poloměrem. Jelikož v základním nastavení parametrů jsou generovány pouze cihly, které mají 6 stěn, stačilo následně pouze vydělit celkový počet stěn vygenerovaného objektu hodnotou 6, pro zjištění počtu cihel ve výsledném objektu. V následující tabulce 6.1 je zaznamenána časová hodnota v milisekundách jednotlivých node skupin, změřená pomocí funkce *Node Timings*, kterou Blender nabízí.

Počet cihel	Cimbuří	Zdi	Věže	Realizace instancí	Celkový čas [ms]
93 604	3.4	2.8	13	27	46,2
131 919	11	14	14	43	82
257 454	16	53	15	79	163
413 344	30	88	15	140	273
766 457	61	186	13	273	533
989 974	77	247	14	324	662
1 242 360	94	265	13	504	876

Tabulka 6.1: Tabulka výkonu v závislosti na počtu cihel hradu

Následně lze vytvořit graf závislosti počtu cihel na výsledném čase pro výpočet geometrie. To je vyobrazeno v grafu 6.1. Z grafu je patrné že čas pro vytvoření geometrie roste lineárně v závislosti na počtu cihel.

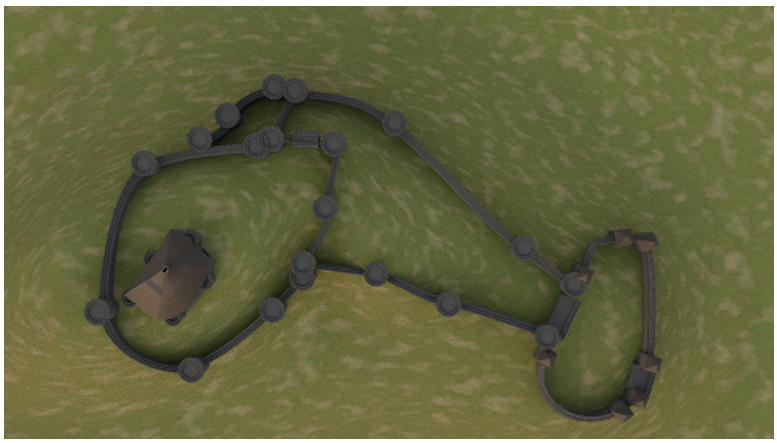


Obrázek 6.1: Graf časů výpočtu v závislosti na počtu cihel

6.3 Porovnání výstupů s realnými fotografiemi

Pravděpodobně nejdůležitější vlastností procedurálně vygenerovaného modelu je jeho vzhled a jak moc odpovídá realitě. K tomuto bylo využito modelu Helfštýnu a jeho skutečných fotek. Následně lze na obrázcích 6.2 lze vidět reálnou fotku, kterou lze porovnat s renderem nacházejícím se na obrázku 6.3.

Obrázek 6.2: Vzdušná fotografie Helfštýna [1]



Obrázek 6.3: Vzdušný render vygenerovaného Helfštýna

Přestože výsledný výstup úplně neodpovídá realitě, jeho vytvoření je otázkou několika minut a v případě že se jedná o například herní model může se tato aproximace jevit dostatečná.

[1] obrázek získán z https://helfstyn.cz/

Kapitola 7

Závěr

Výsledkem této práce je addon, který umožňuje tvorbu hradu na základě křivek, vytvořených uživatelem. V addonu je využito nově přidaných Geometry nodes: fields, které Blender nabízí od verze 3.0. Výsledná implementace umožňuje tvorbu a editaci vytvořeného hradu v reálném čase a díky využití geometry nodes je výsledný produkt velmi detailní. Vygenerované objekty poskytují vysokou míru modularity a u každé struktury lze editovat velké množství parametrů. Tvorba hradů je natolik snadná že ji zvládne i laik. Výhodou a zároveň nevýhodou této práce je míra detailu, který addon generuje. Jelikož hrad je generován jako komplexní objekt složený z jednotlivých primitiv (cihly, desky, šindele), je míra geometrie velmi vysoká, což má výhodu ve vyšší míře detailů nicméně doba výpočtů finální geometrie může být vyšší. Na základě výsledků měření výkonu lze ovšem vidět že vztah počtu cihel vůči času pro generování nabývá lineární složitosti.

Na základě zpětných vazeb dotazníků vyplývá, že výsledný addon je uživatelsky velmi intuitivní a uspokojivě modulární. Uživatelé dále uvádí že by addon využili při tvorbě hradu.

Možným rozšířením do budoucna by byla možnost přidání dalších budov a přidání alternativní možnosti tvorby hradu, který by měl detaily jako cihly řešené pomocí textur, pro lepší výkon v případě využití pro videohry či animace.

Literatura

[1] *Blender 3.0 Reference Manual* [online]. [cit. 2021-04-12]. Dostupné z: https://docs.blender.org/manual/en/latest/index.html.

[2] *Castles and Manor Houses around the World* [online]. [cit. 2021-11-21]. Dostupné z: https://www.castlesandmanorhouses.com/architecture_03_walls.htm.

[3] Besuievsky, G. a Patow, G. Procedural modeling historical buildings for serious games. *Virtual Archaeology Review*. 2013, sv. 4, č. 9, s. 160–166. DOI: 10.4995/var.2013.4268. ISSN 1989-9947. Dostupné z: https://polipapers.upv.es/index.php/var/article/view/4268.

[4] Blinn, J. F. Simulation of wrinkled surfaces. *ACM SIGGRAPH computer graphics*. ACM New York, NY, USA. 1978, sv. 12, č. 3, s. 286–292.

[5] Blinn, J. F. a Newell, M. E. Texture and reflection in computer generated images. *Communications of the ACM*. ACM New York, NY, USA. 1976, sv. 19, č. 10, s. 542–547.

[6] Cook, R. L. Shade trees. In: *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*. 1984, s. 223–231.

[7] Ebert, e. a. *Texturing & modeling: a procedural approach*. Academic Press, 2014. ISBN 1483297020.

[8] Edwards, S. *History Rebuilt* [online]. [cit. 2021-11-21]. Dostupné z: https://historyrebuilt.com/.

[9] El Hakim, S., Beraldin, J.-A., Gonzo, L., Whiting, E., Jemtrud, M. et al. A Hierarchical 3D Reconstruction Approach for Documenting Complex Heritage Sites. *20th Symposium of International Cooperation to Save the World's Cultural Heritage, Torino, Italy*. Leden 2005.

[10] El Hakim, S., Gonzo, L., Voltolini, F., Girardi, S., Rizzi, A. et al. Detailed 3D Modelling of Castles. *International Journal of Architectural Computing*. 2007, sv. 5, č. 2, s. 199–220. DOI: 10.1260/1478-0771.5.2.200. Dostupné z: https://doi.org/10.1260/1478-0771.5.2.200.

[11] Fita, J., Besuievksy, G. a Patow, G. Perspective on procedural modeling based on structural analysis. *Virtual Archaeology Review*. Květen 2017, sv. 8, s. 44. DOI: 10.4995/var.2017.5765.

[12] Gregorová, J., Kalesný, F., Polomová, B. a Vojteková, E. Architectural Modelling of Alternatives for Verification of New Interventions on the Example of the Romanesque Palace at Spiš Castle in Slovakia. *IOP Conference Series: Materials Science and Engineering*. IOP Publishing. oct 2017, sv. 245, s. 082007. DOI: 10.1088/1757-899x/245/8/082007. Dostupné z: https://doi.org/10.1088/1757-899x/245/8/082007.

[13] Kelly, G. a McCabe, H. A survey of procedural techniques for city generation. *ITB Journal*. Citeseer. 2006, sv. 14, č. 3, s. 342–351.

[14] Lagae, A., Lefebvre, S., Cook, R., DeRose, T., Drettakis, G. et al. A Survey of Procedural Noise Functions. *Computer Graphics Forum*. 2010, sv. 29, č. 8, s. 2579–2600. DOI: https://doi.org/10.1111/j.1467-8659.2010.01827.x. Dostupné z: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2010.01827.x.

[15] Lindenmayer, A. Mathematical models for cellular interactions in development I. Filaments with one-sided inputs. *Journal of theoretical biology*. Elsevier. 1968, sv. 18, č. 3, s. 280–299.

[16] Mandelbrot, B. B. Fractal aspects of the iteration of z→ Λz (1-z) for complex Λ and z. *Annals of the New York Academy of Sciences*. Blackwell Publishing Ltd Oxford, UK. 1980, sv. 357, č. 1, s. 249–259.

[17] Perlin, K. An Image Synthesizer. *SIGGRAPH Comput. Graph.* New York, NY, USA: Association for Computing Machinery. jul 1985, sv. 19, č. 3, s. 287–296. DOI: 10.1145/325165.325247. ISSN 0097-8930. Dostupné z: https://doi.org/10.1145/325165.325247.

[18] Püschel, H., Sauerbier, M. a Eisenbeiss, H. A 3D Model of Castle Landenberg (CH) from Combined Photogrammetric Processing of Terrestrial and UAV based Images. In: Chen, J., Jiang, J. a Cho, K., ed. Lemmer: ISPRS, 2008, XXXVII, B6b, s. 93 – 98. DOI: 10.3929/ethz-b-000011989. ISSN 1682-1750. 21st ISPRS Congress; Conference Location: Beijing, China; Conference Date: July 3-11, 2008.

[19] Worley, S. A cellular texture basis function. In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 1996, s. 291–294.

Příloha A

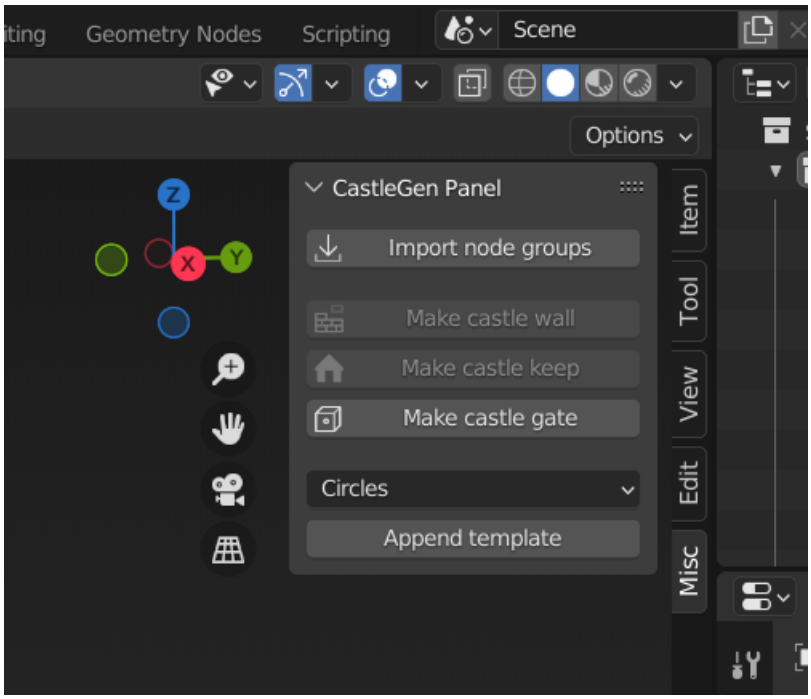
Instalace

Addon je k dispozici na adrese: https://richardgall.gumroad.com/l/CastleGen. Addon je distribuován s cenou „Pay what you want“, je tedy možné jej stáhnout zdarma.

Pro instalaci je nejprve nutné stáhnout addon z výše uvedené adresy, jedná ze o „.zip“ soubor. Následně je třeba otevřít Blender (verze 3.1 a vyšší) a kliknout na Edit →Preferences →Addons, dále kliknout na tlačítko *Install* a navigovat do složky, do které je addon stážený. Následně je třeba vybrat soubor „Castlegen.zip“ a kliknout na tlačítko *Install add-on*. V manažeru addonů se vám zobrazí řádek s názvem CatleGen. Jako poslední krok klikněte na zaškrkávátko v tomto řádku a addon je úspěšně nainstalován.

A.0.1 Použití

Hlavní panel addonu lze najít v záložce Misc, viz obrázek A.1.



Obrázek A.1: Hlavní panel

Nejprve je nutné importovat node groupy a kolekce z addonu do aktuálního souboru. K tomto účelu slouží tlačítko *Import node groups*. Dokud nedojde k importu, budou ostatní tlačítka zašedlé. V případě že k importu již došlo je možné addon používat. Tlačítka *Make castle wall* a *Make castle keep* budou aktivní v případě že všechny vybrané objekty jsou typu křivka. Po stisku těchto tlačítek budou všechny vybrané křivky přetvořeny na hradby či tvrze, na základě vybraného tlačítka. Totéž lze udělat pomocí tlačítka *Make castle gate* s tím rozdílem že všechny označené objekty musí být mesh.

Dodatečně má uživatel možnost využít již vytvořených šablon hradů. Z rozbalovacího seznamu může vybrat jednu možnost a následně tlačítkem *Append template* vložit takto vytvořený hrad do aktuální scény.

Příloha B

Parametry node skupin

CG_CastleWall

- **Wall curve** - vstupní křivka na které bude vygenerována vnější zeď.
- **Wall height** - výška zdi.
- **Wall depth** - tloušťka zdi, tedy vzdálenost mezi vnější a vnitřní zdí.
- **Brick width** - šířka cihly.
- **Brick height** - výška cihly.
- **Brick depth** - hloubka cihly.
- **Brick offset** - posun lichých řádků o hodnotu v intervalu 0 - 1, násobenou šířkou cihly.
- **Is wall cyclic** - nastavení zda je vstupní křivka uzavřená či otevřená.
- **Rotation variance** - maximální rozsah o který budou jednotlivé cihly náhodně rotovány.
- **Offset variance** - maximální rozsah o který budou jednotlivé cihly posouvány vůči normále křivky v bodě, na kterém jsou instancovány.
- **Shingle model** - model šindele.
- **Shingle stacking** - určuje kolik šindelů bude intancováno v rozahu výšky jednoho šindele.
- **Battlement offset** - vzdálenost posuvu cimbuří směrem ven.
- **Battlement support height** - výška vzpěr cimbuří.
- **Battlement support thickeness** - tloušťka vzpěr cimbuří.
- **Battlement support distance** - vzdálenost jednotlivých vzpěr cimbuří.
- **Battlements / Hoardings** - volba zda bude mít cimbuří i podsebití.
- **Hoardings plank width** - šírka jednotlivých desek podsebití.

- **Hoardings plank height** - výška jednotlivých desek podsebití.
- **Hoardings hole modulo** - udává vzdálenost v deskách mezi deskami ze střílnami.
- **Hoardings hole width %** - podíl šířky střílny vůči šířce desky
- **Hoardings hole height %** - podíl výšky střílny vůči výšce desky
- **Hoardings roof depth** - hloubka, do jaké bude generována střecha
- **Hoardings pillars thickness** - tloušťka trámů podsebití
- **Hoardings pillars distance** - vzdálenost jednotlivých trámů podsebití
- **Merlon width** - šířka jednotlivých zubů cimbuří.
- **Merlon height** - výška jednotlivých zubů cimbuří.
- **Merlon depth** - hloubka jednotlivých zubů cimbuří.
- **Crenels width** - Vzdálenost mezi středy jednotlivých ozubů.
- **Tower height** - výška věží.
- **Tower radius** - poloměr věží.
- **Tower wall sides** - počet stran n-úhelníku podstavy věží.
- **Tower roof height** - výška střechy věže.
- **Tower battlement offset** - velikost odsazení cimbuří od strany věže
- **Tower battlement support height** - výška vzpěr cimbuří věží.
- **Tower support thickness** - tloušťka vzpěr cimbuří věže
- **Tower support distance** - vzdálenost mezi jednotlivými vzpěrami cimbuří věže.
- **Tower merlon width** - šířka ozubů cimbuří věže.
- **Tower merlon height** - výška ozubů cimbuří věže.
- **Tower merlon depth** - hloubka ozubů cimbuří věže.
- **Tower crenels width** - vzdálenost mezi jednotlivými ozuby cimbuří věžě.
- **Tower down offset** - posun věže směrem dolů k zajištění, aby se části věže nevznášeli nad nerovným povrchem.

CG_CastleKeep

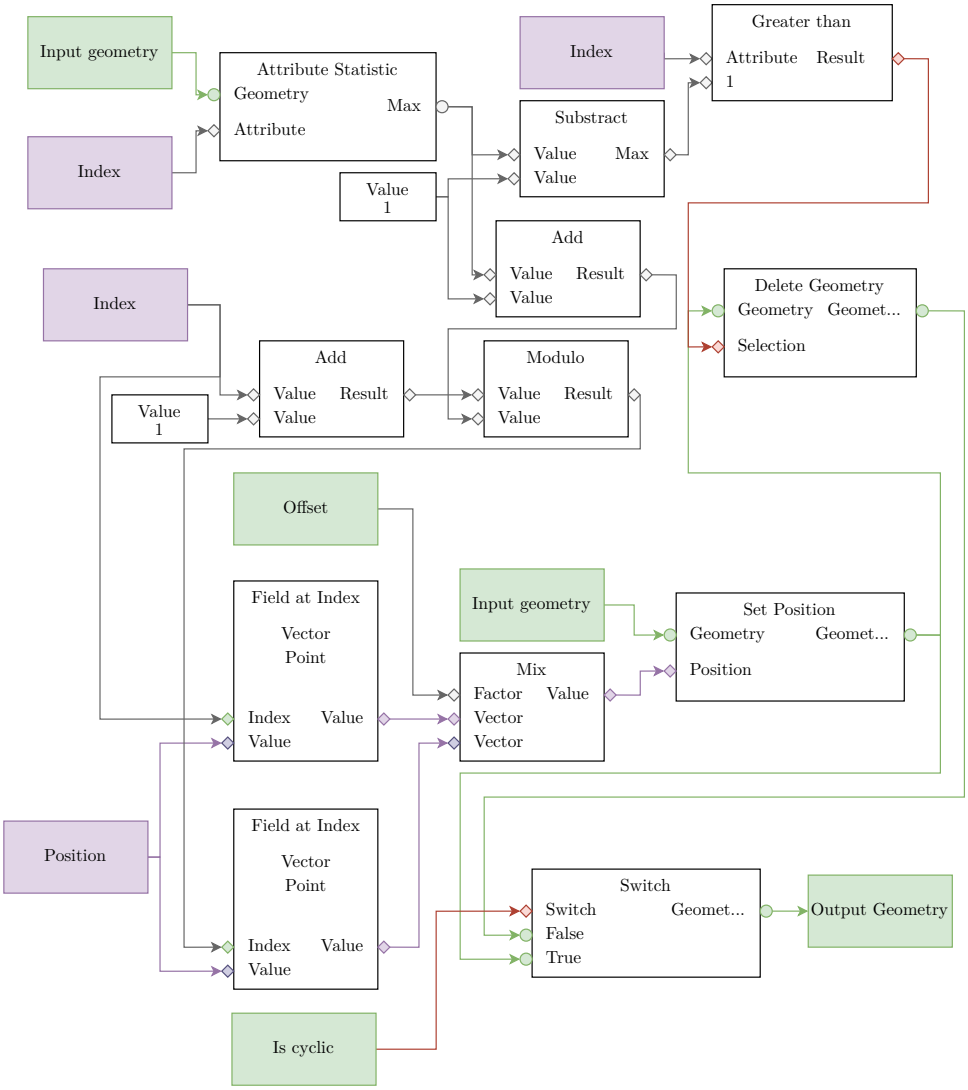
- **Keep Curve** - vstupní křivka, na které bude vygenerována tvrz.
- **Towers count** - počet věží, které budou umístěny po obvodu tvrze.
- **Shingle** - model šindele.
- **Roof height** - výška střechy.
- **Shingles density** - hustota šindelů nad sebou.
- **Rotation variance** - rozsah náhodné rotace elementů.
- **Offset variance** - rozsah náhodného posuvu elementů.
- **Wall height** - výška zdi tvrze.
- **Brick width** - šířka cihly.
- **Brick height** - výška cihly.
- **Brick depth** - hloubka cihly.
- **Offset** - posun lichých řádků o hodnotu v intervalu 0 - 1 násobenou šířkou cihly.
- **Tower radius** - poloměr věží.
- **Tower roof height** - výška střechy věží.
- **Tower sides** - počet bodů n-úhelníku který tvoří podstavu věže.
- **Tower battlement offset** - vnější posuv cimbuří.
- **Tower battlement support height** - výška vzpěr cimbuří.
- **Tower battlement support thickness** - tloušťka vzpěr cimbuří.
- **Tower battlement supports distance** - vzdálenost jednotlivých vzpěr cimbuří.
- **Tower merlon width** - délka ozubí cimbuří.
- **Tower merlon height** - výška ozubí cimbuří.
- **Tower merlon depth** - hloubka ozubí cimbuří.
- **Tower crenels width** - vzdálenost mezi jednotlivými ozuby cimbuří.
- **Tower offset** - posun všech věží po obvodu tvrze.
- **Tower height** - výška věží.
- **Tower roof / merlons** - volba zda věže budou mít střechu či cimbuří.

CG_CastleGate

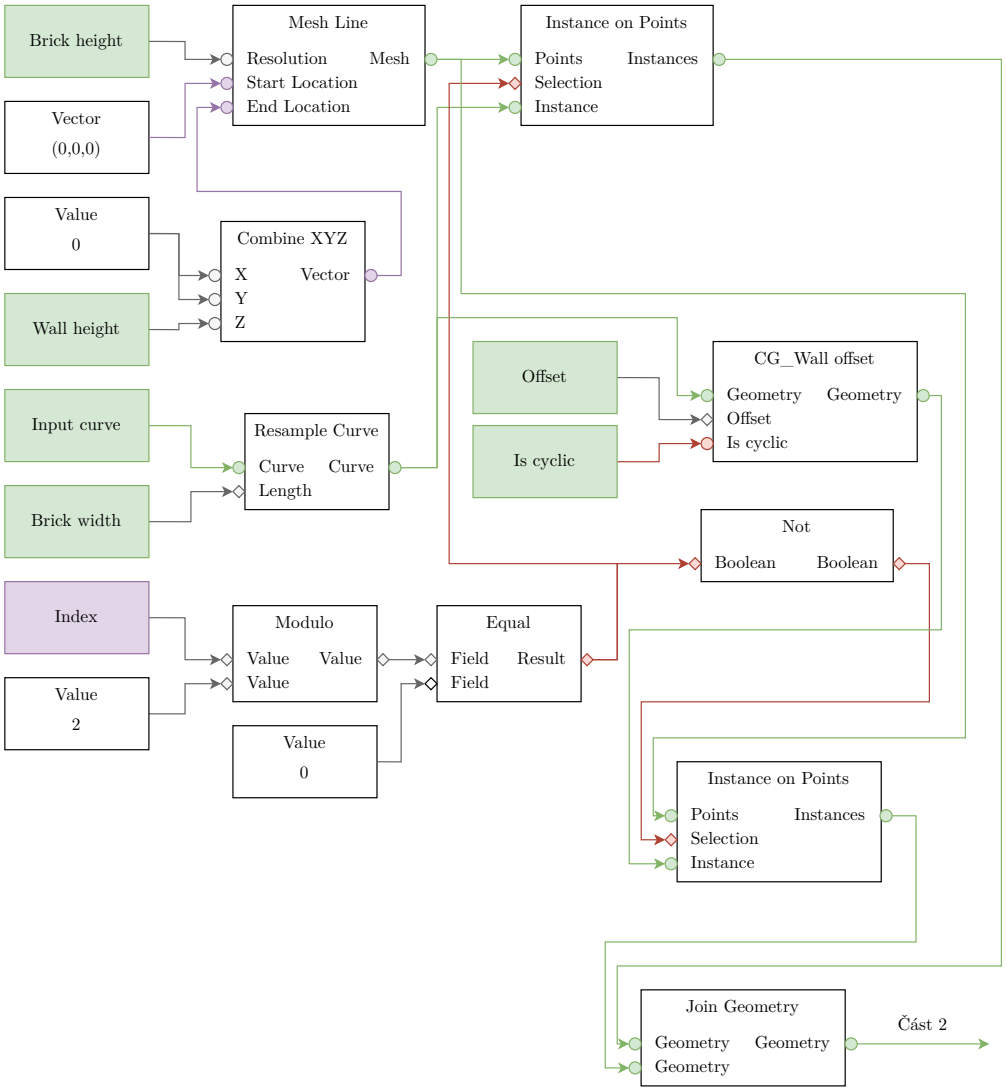
- **Wall depth** - hloubka brány.
- **Wall width** - šířka brány.
- **Wall height** - výška brány
- **Brick width** - šířka cihly.
- **Brick height** - výška cihly.
- **Brick depth** - hloubka cihly.
- **Offset** - posun lichých řádků o hodnotu v intervalu 0 - 1 násobenou šířkou cihly.
- **Rotation variance** - rozsah náhodné rotace elementů.
- **Offset variance** - rozsah náhodného posuvu elementů.
- **Gate height** - výška průchodu brány.
- **Gate width** - šířka průchodu brány.
- **Battlement offset** - odsazení cimbuří od vnější zdi.
- **Battlement support height** - výška vzpěr cimbuří.
- **Support thickness** - tloušťka vzpěr cimbuří.
- **Support distance** - vzdálenost vzpěr cimbuří.
- **Merlon width** - šířka ozubů cimbuří.
- **Merlon height** - výška ozubů cimbuří.
- **Merlon depth** - hloubka ozubů cimbuří.
- **Crenels width** - vzdálenost ozubů cimbuří.
- **Bar desity** - hustota tyčí mříže brány.
- **Bar thickness** - tloušťka tyčí mříže brány.
- **Gate open index** - míra otevření brány 0 - brána je zavřená, 1 - brána je otevřená.

Příloha C

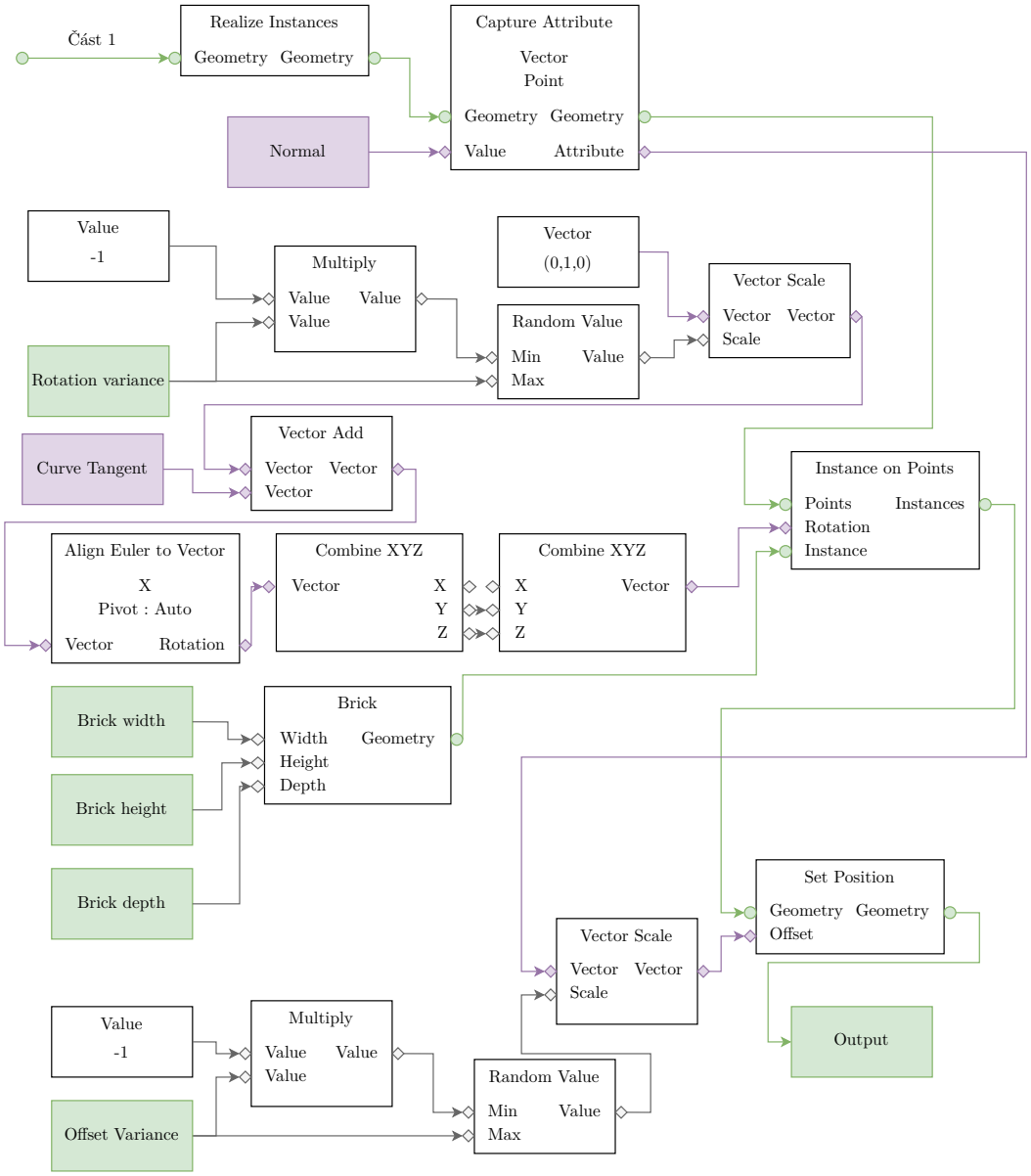
Schémata



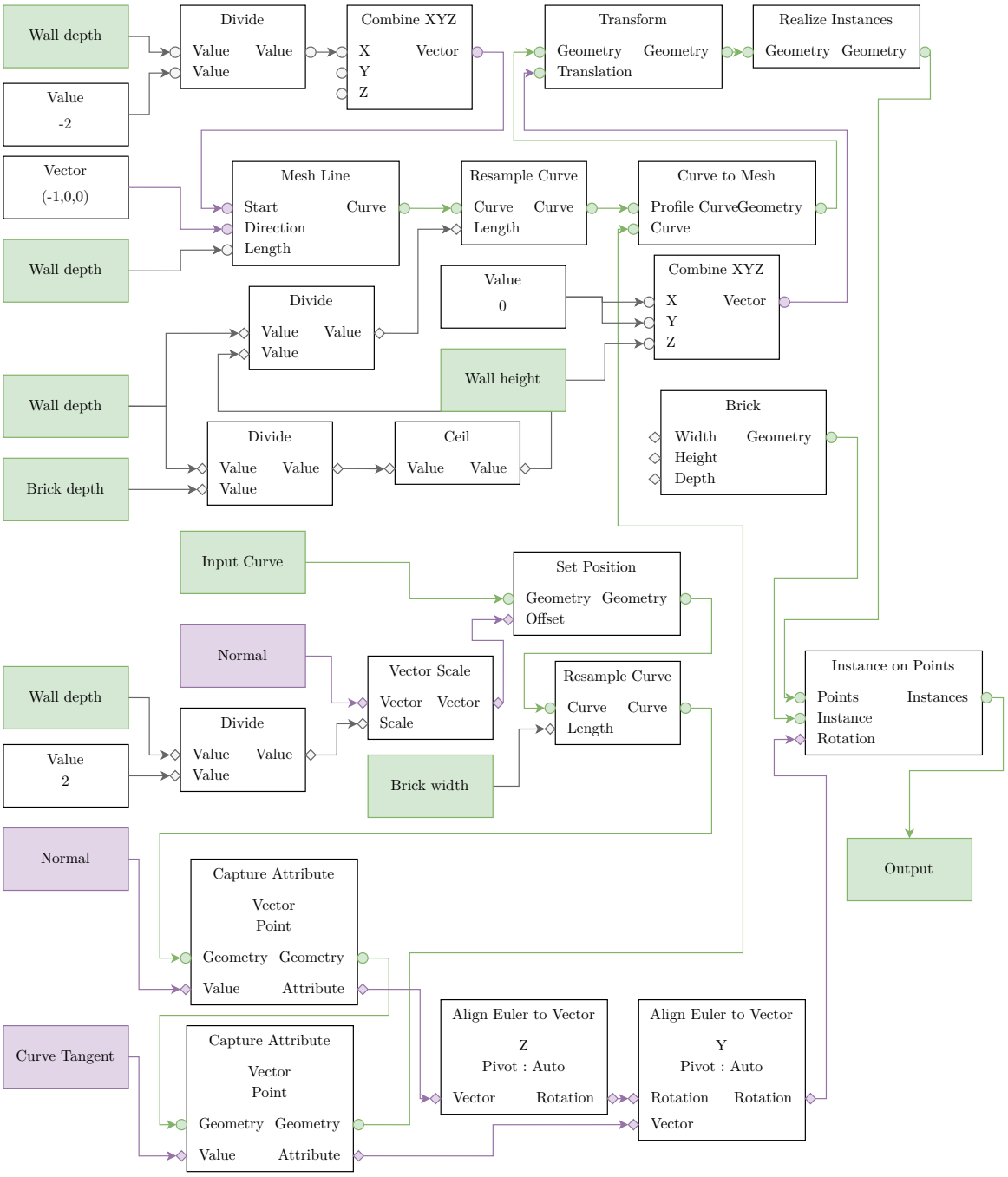
Obrázek C.1: Detailní schéma implementace CG_Wall offset



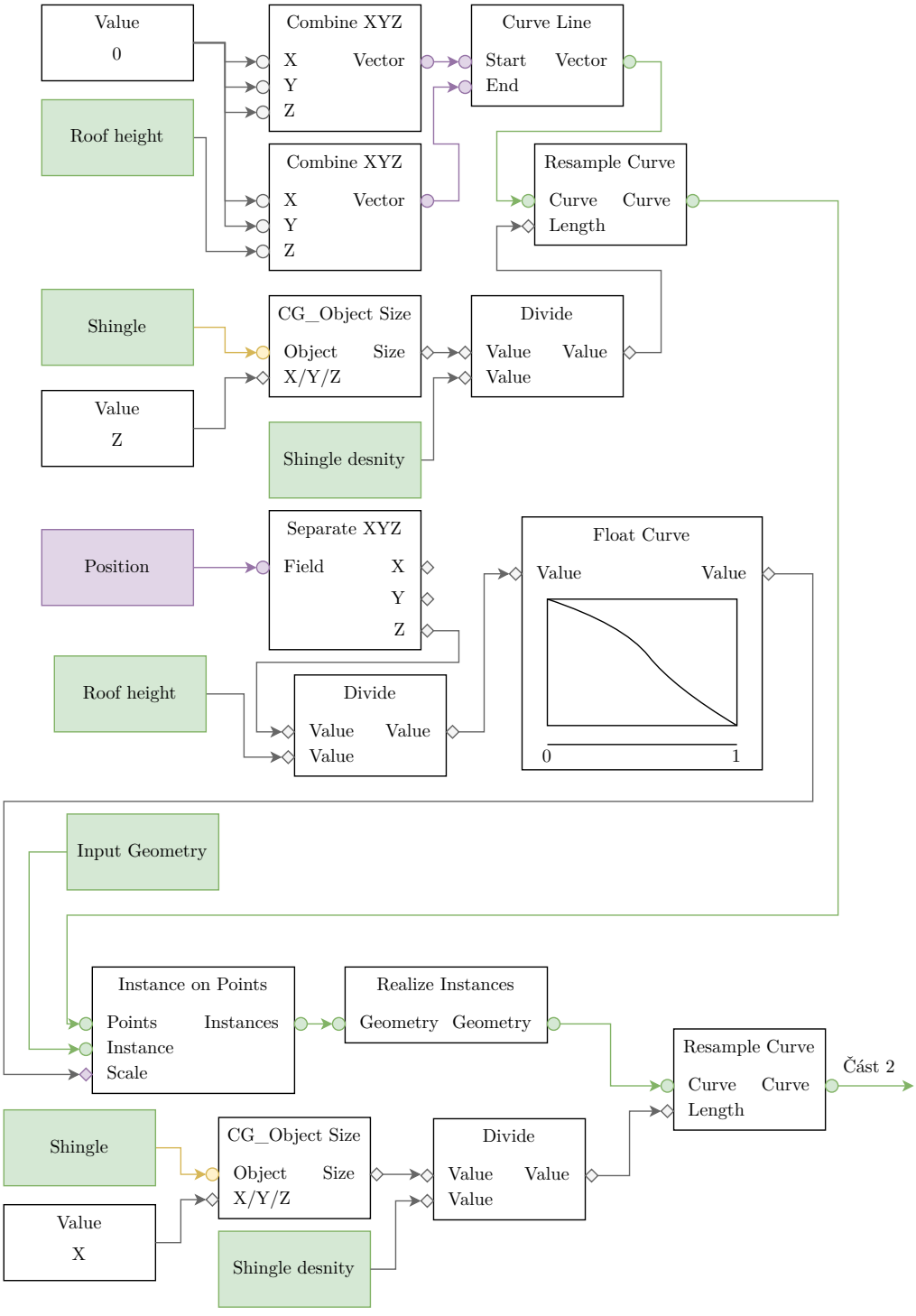
Obrázek C.2: Detailní schéma implementace CG_Wall, 1. část



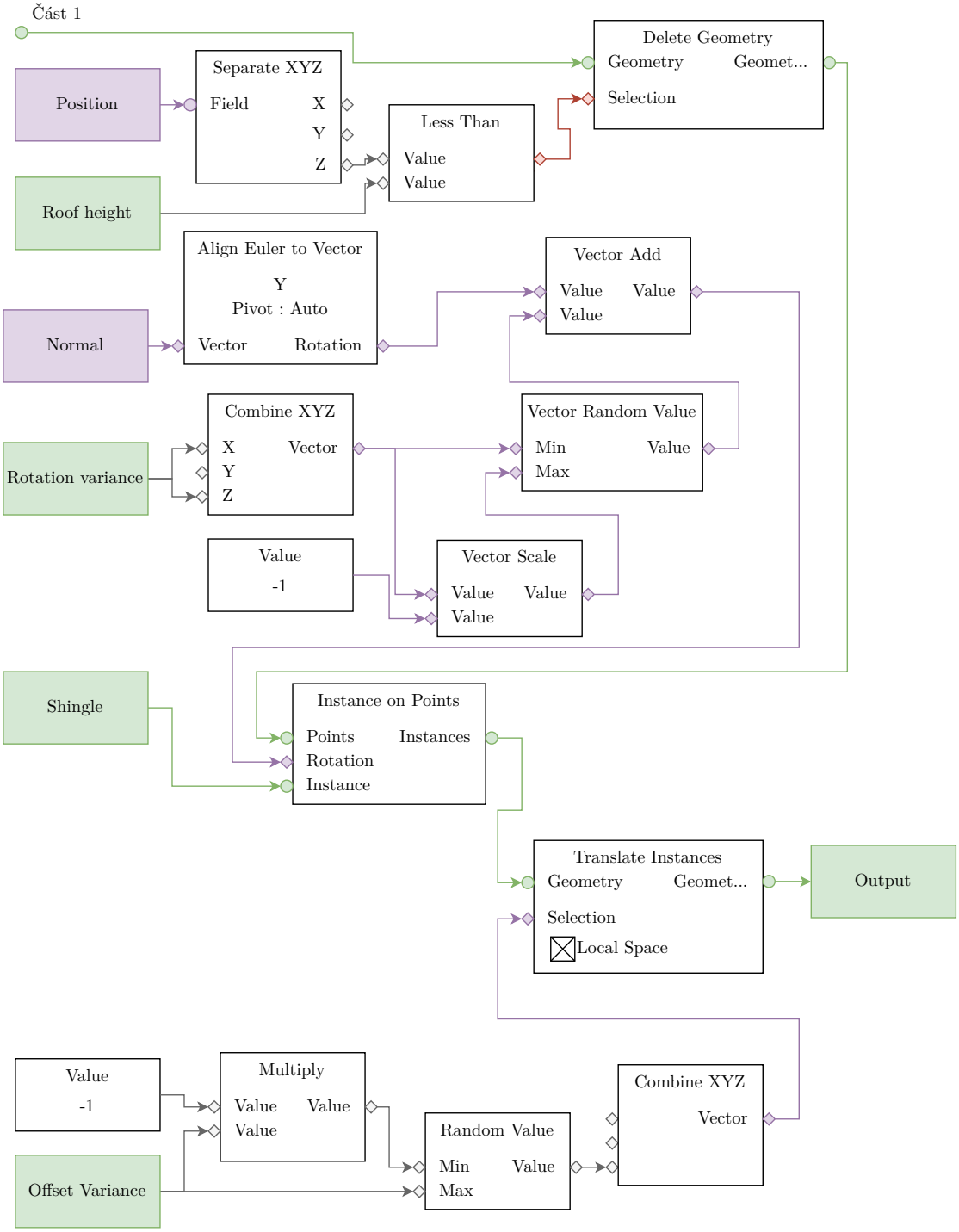
Obrázek C.3: Detailní schéma implementace CG_Wall, 2. část



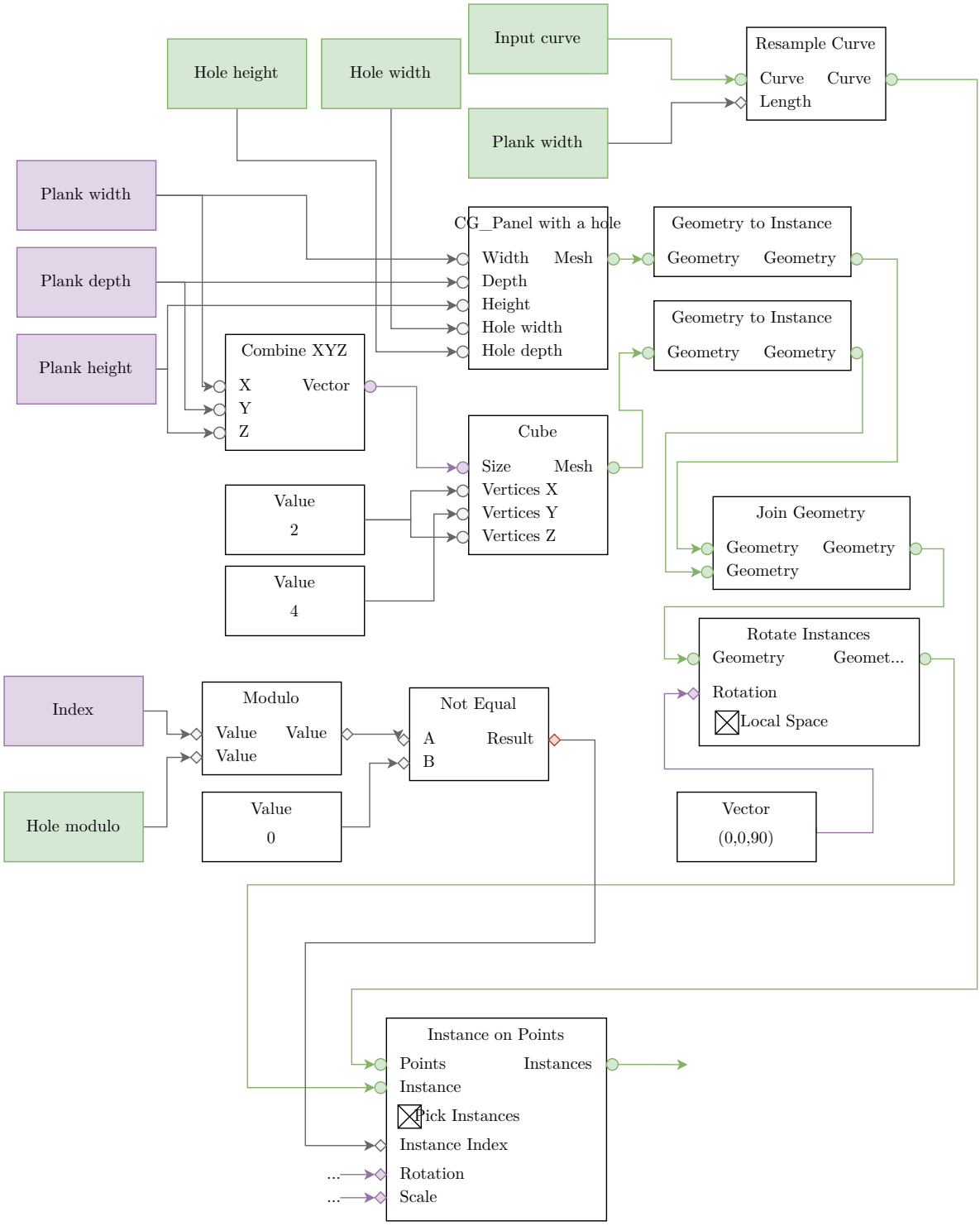
Obrázek C.4: Detailní schéma implementace CG_Walkway



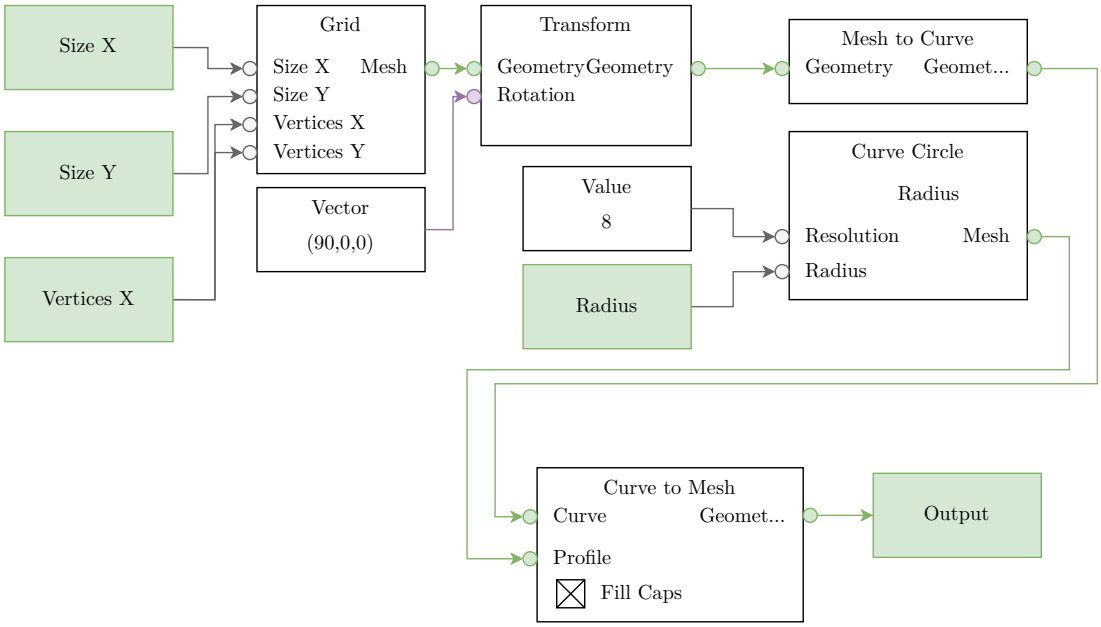
Obrázek C.5: Detailní schéma implementace CG_Roof, 1. část



Obrázek C.6: Detailní schéma implementace CG_Roof, 2. část



Obrázek C.7: Detailní schéma implementace CG_Planks



Obrázek C.8: Detailní schéma implementace CG_Iron bars