ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
Katedra počítačové grafiky a interakce

Procedurální Generování Jeskynních Systémů

Procedural Generation of Cave Systems

Bakalářská práce

Vedoucí práce: Ing. Jaroslav Sloup

Martin Douda

Roztoky u Prahy
19. května 2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Douda** Jméno: **Martin** Osobní číslo: **507407**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Specializace: **Počítačové hry a grafika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Generování jeskynních systémů

Název bakalářské práce anglicky:

Cave Networks Generation

Pokyny pro vypracování:

Seznamte se s principy vzniku jeskyní a souvisejícími fyzikálními a geologickými procesy. Proveďte rešerši literatury pojednávající o metodách generování jeskyní používaných v počítačové grafice [1-4].
Na základě článku [1] naimplementujte generátor jeskynních systémů tvořených chodbami různých průměrů a tvarů (eliptický, klíčový, kaňonovitý). Prostudujte stávající možnosti herního enginu Unity a navrhněte způsob integrace generátoru do prostředí Unity. Generátor rozšiřte o přehledné uživatelské rozhraní umožňující měnit všechny relevantní simulační parametry (např. permeabilita, směr prasklin, úroveň vodní hladiny).
Navrhněte vhodný způsob reprezentace krápníků a strategii pro jejich rozmístění ve vytvořených jeskyních. Dále implementujte zjednodušený model tekoucí vody s cílem vytvořit podzemní jezírka a řeky.
Funkčnost implementace ověřte vytvořením alespoň třech různých typů jeskynních systémů a zhodnoťte jejich věrohodnost porovnáním s fotografiemi podobných reálných jeskyní.

Seznam doporučené literatury:

[1] A. Paris, E. Guérin, A. Peytavie, P. Collon, E. Galin: Synthesizing Geologically Coherent Cave Networks. Computer Graphics Forum, Vol.40, No.7, p.277-287, 2021.
[2] K. Franke, H. Müller: Procedural generation of 3D karst caves with speleothems. Computers & Graphics, Vol.102, p.533-545, Elsevier, 2022.
[3] D.M. Tortelli, M. Walter: Modeling and Rendering the Growth of Speleothems in Real-time. In International Conference on Computer Graphics Theory and Applications, vol.1, pp. 27-35. SCITEPRESS, 2009.
[4] J. Cui, Y.-W. Chow, M. Zhang: Procedural generation of 3D cave models with stalactites and stalagmites. International Journal of Computer Science and Network Security, Vol.11 No.8, August 2011.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jaroslav Sloup Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **08.02.2024** Termín odevzdání bakalářské práce: **24.05.2024**

Platnost zadání bakalářské práce: **21.09.2025**

Ing. Jaroslav Sloup podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.
Roztoky u Prahy, 19. května 2024

Podpis: .....

Poděkování

Děkuji Ing. Jaroslavu Sloupovi za jeho vedení, trpělivost, zpětnou vazbu a důležité připomínky při zpracování této bakalářské práce. Také děkuji své babičce Ing. Evě Doudové za průběžné čtení práce a následnou zpětnou vazbu ohledně struktury textu práce.

Abstrakt

Tato bakalářská práce představuje nástroj pro procedurální generování jeskynních systémů za použití algoritmu pro nejkratší cestu, algoritmu Primitive Sweeping, algoritmu Marching Cubes a dalších algoritmů z počítačové grafiky. Tyto existující algoritmy jsou sekvenčně využity pro vygenerování realisticky vypadajících krasových jeskynních systémů, krápníků a jeskynních jezer. Při generování jsou v úvahu brány geologické podmínky prostředí, jako permeabilita prostředí, natočení krystalové mřížky a vzdálenost od hladiny podzemních vod.

Klíčová slova: počítačová grafika, procedurálně generovaný obsah, jeskynní systémy, speleogeneze, krápníky, jeskynní jezera

This bachelor's thesis introduces a tool for procedural generation of cave systems using algorithms such as the shortest path algorithm, Primitive Sweeping algorithm, Marching Cubes algorithm, and other algorithms from computer graphics. These existing algorithms are sequentially employed to generate realistically looking karst cave systems, stalactites, and cave lakes. During the generation process, geological conditions of the environment are considered, such as the permeability of the medium, orientation of the crystal lattice, and distance from the groundwater level.

Key words: computer graphics, procedural content generation (PCG), cave systems, speleogenesis, speleothems, cave lakes

Obsah

Seznam obrázků

Seznam tabulek

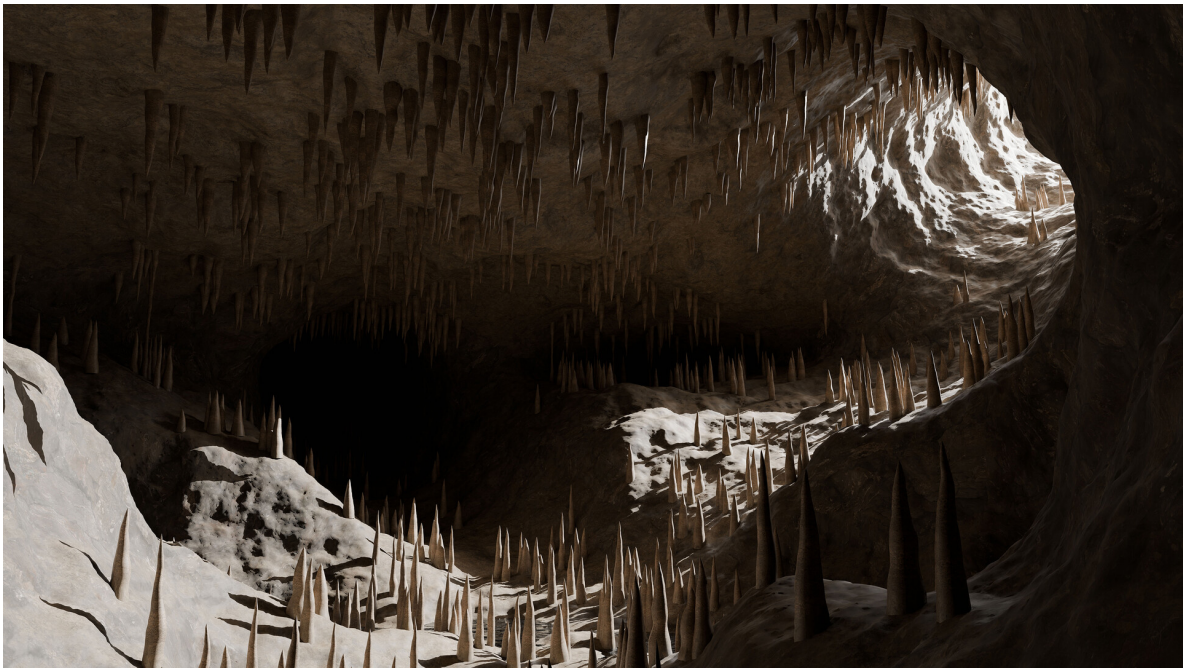
1 Úvod

Jeskynní systémy jsou fascinujícími geologickými útvary, které mají široké využití od průzkumu a dobrodružství až po videohry a filmovou tvorbu. Procedurální generování jeskynních systémů se skládá z mnoha metod pro vytváření těchto útvarů pomocí algoritmů, které jsou založené na snaze se co nejvěrněji přiblížit reálným fyzikálním dějům. Metody pro procedurální generování jsou běžně používány v počítačové grafice pro vytváření textur, 3D modelů a terénů. Mezi hlavní výhody procedurálního generování patří:

- Různorodost: Procedurální generování umožňuje vytvářet velké množství různorodého obsahu s minimálním dodatečným manuálním úsilím. To znamená, že je možné vytvářet nekonečné množství neopakujících se tvarů jeskynních systémů, terénů, budov, stromů a dalších prvků, bez nutnosti vytvářet každý z nich ručně.
- Efektivita: Díky procedurálnímu generování lze vytvářet obsah nesrovnatelně rychleji než manuálně. To je zvláště užitečné v herním průmyslu, kde je potřeba produkovat velké množství obsahu v krátkém čase. Malou změnou vstupních parametrů jsou vývojáři schopni vytvořit pomocí stejného nástroje modely pro různé hry.
- Dynamika: Herní prostředí může být generováno za běhu a měnit se v závislosti na pohybu hráče a různých jiných faktorech. To přináší hráčům chuť prozkoumávat své okolí a často v kombinaci s otevřeným světem zpříjemňuje herní zážitek.
- Úspora paměti: Procedurální generování umožňuje vytvářet rozsáhlého prostředí s minimální spotřebou paměti díky implementaci náhody v počítačích. Pseudonáhodný generátor čísel vygeneruje podle jedné vstupní posloupnosti znaků (seedu) vždy stejnou posloupnost a tedy i svět. Například hráčům tedy pro sdílení svého světa stačí sdílet tento několikamístný řetězec.

Vytváření univerzálních nástrojů pro procedurální generování bývá však obtížné a často balancuje mezi menším množstvím nastavitelných parametrů, pro uživatele příjemnější na používání s menší kontrolou nad výsledkem a velkým množstvím nastavitelných parametrů, což povede k lepším výsledkům za cenu větší složitosti ovládání. Přes velké využití a rostoucí poptávku po procedurálně vygenerovaném terénu, zůstává generování některých struktur pozadu. Za tím často stojí potřeba vysoké úrovně detailu pro přesvědčení lidského oka o realističnosti. Do této kategorie zapadají i jeskynní systémy, jejichž prostory bývají velice specifické, zahrnují spoustu větvení, rozšiřování a zužování tunelů a vysoce detailní složení stěn, což značně generování komplikuje. Existují ale metody, které se tyto často vznikající komplikace pokoušejí řešit. Ukázka z velké části procedurálně vygenerované jeskyně v Blenderu je na *obr.1*.

Tato práce se v první části zaměřuje na shrnutí procesů, které se podílejí na vzniku různých typů jeskynních systémů a na metody používané k jejich simulaci v počítačové grafice. Do větší hloubky potom rozebírá krasové jeskyně, což je nejběžnější jeskyně, kterou si člověk pod slovem jeskyně představí. V druhé části se potom práce věnuje návrhu řešení a implementaci algoritmů pro generování realistických jeskynních systémů s krápníky a jeskynními jezery.



Obrázek 1: **Vygenerovaný jeskynní systém v Blenderu**. Jeskynní systém s krápníky z velké části procedurálně vygenerovaný pomocí nástroje vytvořeného v Blenderu. Marcel Lukac, artstation.com

2 Jeskynní systémy v reálném světě

V této kapitole se budu věnovat vzniku různých typů jeskynních systémů a procesům, které za ním stojí. Dále budu rozebírat krasové jeskyně a s jejich vznikem spojené chemické a mechanické procesy. Rozeberu také vznik jednotlivých typů krápníků. Poslední věcí obsaženou v této kapitole je vznik jeskynních jezer uvnitř krasových jeskyní.

2.1 Typy jeskynních systémů

Jeskyně jsou přírodně vzniklé podzemní dutiny a formace propojené do větších celků. Dle článku [1] existuje pět základních jeskynních typů, které vznikají odlišnými způsoby. Jedná se o krasové jeskyně, mořské jeskyně, lávové tunely, ledovcové jeskyně a eolské jeskyně.

2.1.1 Krasové jeskyně

Jsou to jeskyně, které vznikají krasověním, což je chemický proces, který formuje jeskynní systémy. Dešťová voda, nasává před dopadem na zem ze vzduchu oxid uhličitý a při prosakování půdou se mění na slabou kyselinu. Ta se pak usadí ve výšce hladiny podzemních vod a postupně rozpouští horniny, čímž vznikají rozsáhlé podzemní dutiny. Nejběžnější horninou ve které se krasové jeskyně tvoří je vápenec, pro který je proces znázorněn na *obr.4* a detailně popsán v kapitole 2.2. Příklad krasové jeskyně je na *obr.2a*.

2.1.2 Mořské jeskyně

Mořské jeskyně se vyskytují téměř všude, kde vlny narážejí do pevných břehů. Vznikají z několika důvodů, které jsou přímým důsledkem prudkých nárazů vln. Největší vliv na formování mořských jeskyní mají malé částečky a jiné abrazivní materiály, které vlny nesou. Ty poté v rychlosti narážejí do břehů, kde vytvářejí malé praskliny a napomáhají erozi. Velký vliv má také samotná hydraulická síla vln, která uvěžňuje vodu a vzduch ve vzniklých skalních prasklinách a vlastní vahou je zatlačuje velkou silou dovnitř. Kusy skály se potom odlamují a vytvářejí dutiny, které násobí tento efekt. Vracející se voda poté tyto kusy smývá a znovu je používá k rozšiřování vzniklých prohlubní. Dle [1] největší mořská jeskyně s názvem Riko Riko Cave na pobřeží Nového Zélandu je na *obr.2b*.

(a) Koněpruské jeskyně, krasové jeskyně v České republice. Autor: Lenka Kachlík, konepruske.caves.cz

(b) Riko Riko Cave, mořská jeskyně na Novém Zélandu. Autor: Brook Sabin, stuff.co.nz

Obrázek 2: **Krasová jeskyně a mořská jeskyně**.

2.1.3 Lávové tunely

Lávové tunely, nebo-li pyrodukty [2], jsou jeskyně, které vznikají při sopečných erupcích. Během erupce láva proudí z kráteru, nebo z bočních prasklin sopek. V případě že se svrchní vrstva lávy rychle ochladí a ztuhne na povrchu, zatímco spodní vrstva lávy ještě neztuhla, může spodní vrstva náhle odtéct a zanechat za sebou tunely. Nejčastěji spodní vrstva odtéká po skončení erupce, kdy poklesne zpět hlouběji do nitra Země. Lávové tunely na Havaji jsou na *obr.3a*.

2.1.4 Ledovcové jeskyně

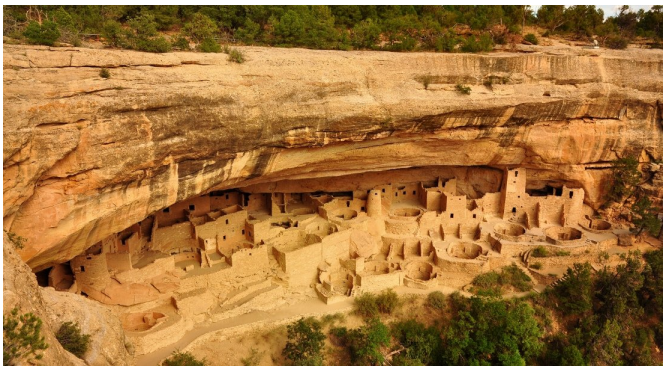
Ledovcové jeskyně vznikají táním ledu, kdy potůčky vody pronikají prasklinami v ledovcích hlouběji a třením led zahřívají. Tím se vytváří větší praskliny, které umožňují pronikání většího množství vody a někdy i proudění vzduchu, který také napomáhá tání. Podle článku [1] se jim často nesprávně říká ledové jeskyně. Ledové jeskyně je ale označení pro jeskyně, které obsahují celoročně větší množství ledu a jedná se většinou o krasové jeskyně, nebo lávové tunely. Ledovcová jeskyně na Islandu jsou na *obr.3b*.

2.1.5 Eolské jeskyně

Eolské jeskyně se vyskytují v pouštních oblastech, nejčastěji v pískovcových horninách. Jsou formovány malými částicemi, které vítr sebere a tře jimi o pískovcové stěny a jen výjimečně nabývají většího objemu. Tento typ jeskyně je známý tím, že byl některými civilizacemi využíván pro obydlí. Eolské jeskyně v Mesa Verde National Park v Coloradu jsou na *obr.3c*.

(a) Nāhuku-Thurston, lávovové tunely na Havaji. Autor: D. Boyle, nps.gov

(b) Kverkfjoll Glacier Caves, ledovcové jeskyně na Islandu. Autor: Daniel González, visitvatnajokull.is



(c) Eolská jeskyně v Mesa Verde National Park v Coloradu. Převzato z: authentikusa.com

Obrázek 3: **Lávový tunel, ledovcová jeskyně a eolská jeskyně**.

2.2 Procesy tvořící krasové jeskyně

Krasové jeskyně budou jediným typem jeskyní, kterým se tato práce bude dále zabývat. Samotné procesy, které se podílejí na tvorbě krasových jeskyní, jsou výsledkem dlouhodobých interakcí mezi různými prvky přírodního prostředí. Hlavními jsou procesy chemické a mechanické, které mají klíčový vliv na formování typických jeskynních struktur. Působení těchto procesů a formování jeskynních struktur v mnoha fázích je sdruženo pod pojmem speleogeneze.

2.2.1 Chemické procesy

Krasovění je hlavní chemický proces, který formuje krasové jeskynní systémy. Dešťová voda se před dopadem na zem mísí se vzdušným oxidem uhličitým a vytváří tak slabou uhličitou kyselinu [15]:

$$H_2O + CO_2 \longrightarrow H_2CO_3$$

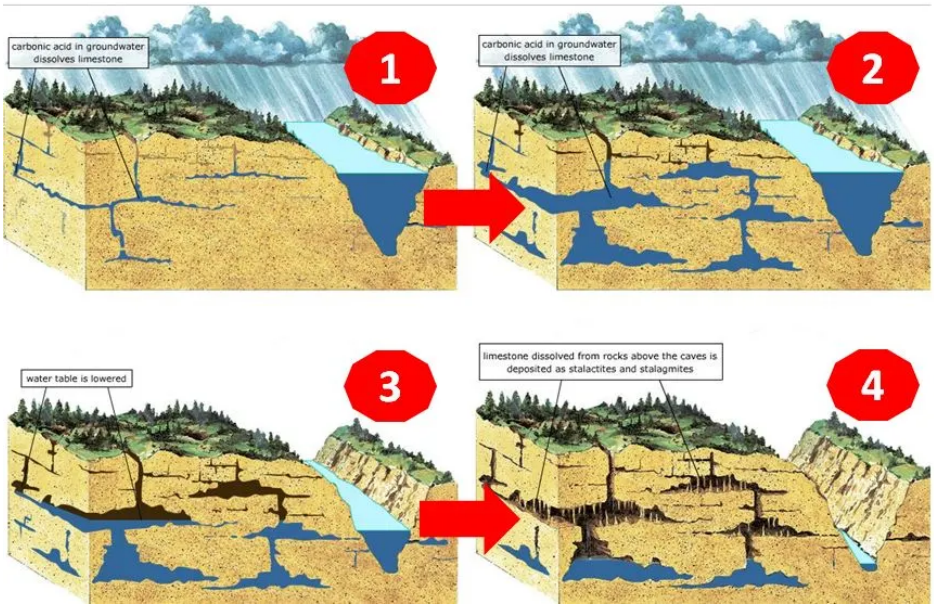
Voda s nízkým obsahem kyseliny uhličité potom po dopadu na zem prosakuje půdou a horninami, načež se usadí ve výšce hladiny podzemních vod (první rámeček na *obr.4*) a postupně rozpouští horniny. Mezi tyto horniny se řadí převážně uhličitany (vápenec, dolomit a mramor) a evapority (sádrovec, anhydrit a halit) [1]. Dále se budeme soustředit na nejběžnější horninu, kterou je vápenec. Když se kyselina uhličitá dostane do styku s uhličitanem vápenatým, ze kterého je vápenec složen až z 80%, chemickou reakcí spolu vytvářejí hydrogenuhličitan vápenatý [15]:

$$CaCO_3 + H_2CO_3 + H_2O \longrightarrow Ca(HCO_3)_2 + H_2O$$

Hydrogenuhličitan vápenatý zůstává rozpuštěný ve vodě a putuje dále skrze horninu, přičemž na svém původním místě zanechává dutiny (druhý rámeček na *obr.4*). Ve chvíli, kdy narazí na nějaký jeskynní strop, přijde do kontaktu se vzduchem a začne odkapávat, spouští zpětnou chemickou reakci. Při té částice hydrogenuhličitanu vápenatého krystalizují a vzniká opět vápenec [15]:

$$Ca(HCO_3)_2 + H_2O \longrightarrow CaCO_3 + CO_2 + H_2O$$

Krystalizace probíhá v závislosti na rychlosti odkapávání buď na stropě, a dává tak za vznik stalaktitům, nebo až po dopadu kapky na zem a vytváří tak stalagmity (většinou se jedná o obojí najednou). Stalaktity, které se propojí se svým stalagmitem, se potom nazývají stalagnáty. Celý tento postupný proces může trvat tisíce až miliony let, v závislosti na místních geologických a povětrnostních podmínkách. Každá kapka vody, která se spustí do jeskyně, může přispět k rozšíření krápníků a vytvoření nových útvarů. Průměrný růst krápníků je pak kolem 0.1 mm za rok, ale za naprosto ideálních podmínek je možný růst až 3 mm za rok.

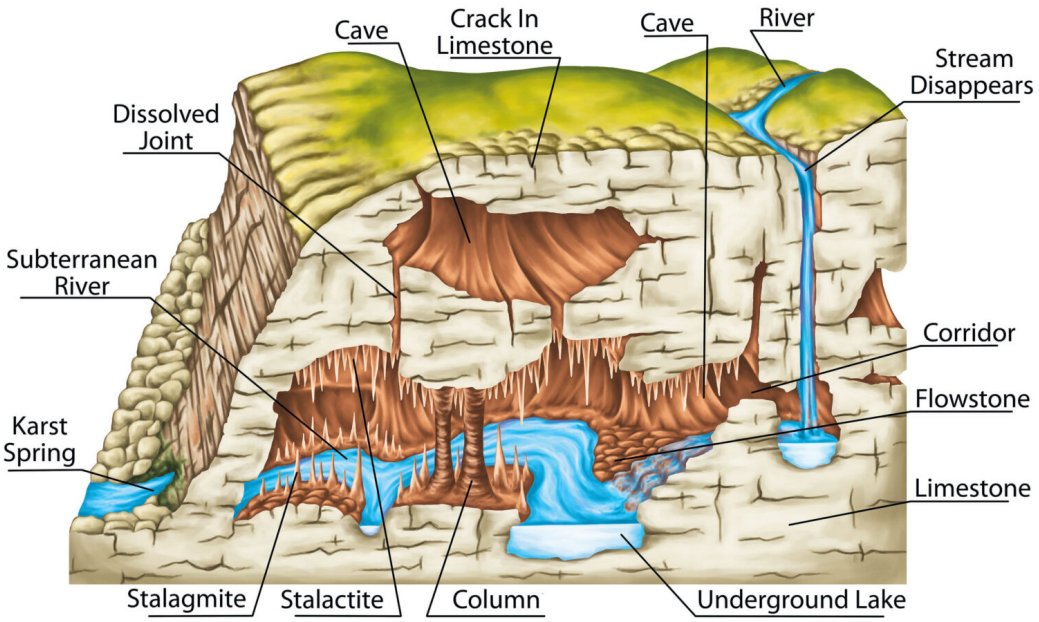


Obrázek 4: **Proces krasovění**. 1) Kyselina uhličitá prosakuje do podzemí a usazuje se ve výšce podzemních vod. 2) Kyselina uhličitá rozpouští vápenec a rozšiřuje tak jeskyně. 3) Vodní hladina podzemních vod klesá a s tím i hladina okolních nádrží. 4) Rozpuštěný vápenec se usazuje, krystalizuje a tvoří stalaktity, stalagmity a stalagnáty. Převzato z: filipinocavedivers.com

2.2.2 Mechanické procesy

Mezi nejdůležitější mechanické procesy, které urychlují vývoj nejen krasových jeskynních systémů, patří eroze a abraze [13]. Jsou to procesy, jejichž efekt je většinou přímo úměrný s velikostí a rychlostí průtoku vody, ať už nepravidelné dešťové, nebo pramenité. Mikroskopické částice vody interagují s povrchem horniny a působí jako jemný brusný prostředek, který odlamuje větší kusy a smývá je s sebou. Pohyby vody nesoucí sedimenty tyto efekty znásobují, neboť voda s pevnými částicemi pracuje jako efektivní abrazivní prostředek, který má schopnost rychleji obrušovat a tvarovat horniny. Abraze, nárazové vlny a obecné proudění vodních toků dále přispívá k erozi, když jsou podemílány nestabilní kusy horniny, které se poté časem sesypávají. Tyto procesy dohromady pomalu odstraňují částice horniny a formují chodby, stěny a kanály jeskynních systémů. Vliv vody je znázorněn na *obr.5*.

Mechanická eroze může být též důsledkem přirozeného procesu zvětrávání hornin, kde se horniny postupně rozpadají v důsledku změn teploty, tlaku a jiných vlivů prostředí. Tření a opakovaná interakce mezi povrchem hornin a prostředím vytvářejí dynamické jeskynní struktury, které mohou být podstatně ovlivněny charakterem hornin, hydrologickými podmínkami a dobou působení erozních faktorů. Tyto procesy, nejenže formují vnitřní strukturu jeskyní, ale také ovlivňují celkový ekosystém a geologickou diverzitu podzemních prostor. Vliv zemětřesení může také hrát klíčovou roli a změnit směr, jímž se vytvářená jeskyně ubírá. Vibrace a náhlé změny tlaku mohou urychlit nebo narušit procesy eroze a tím i tvorbu jeskynních systémů. Ve vážnějších případech mohou zemětřesení způsobit kolaps částí jeskynních prostor, nebo jejich propadání, což může dramaticky změnit jeskynní topografii. Mimo jiné je také katastrofické pro živočichy, kteří v těchto jeskyních žijí (netopýři, mloci, pavouci atd.).



Obrázek 5: **Efekt abraze**. Neustálý průtok vody způsobuje obrušování vápence a dochází tak k tvarování vodního koryta. Koryto je nadále prohlubováno a zanechává nad sebou pukliny, které tvoří jeskyně. Převzato z: nckri.org

2.3 Vznik jeskynních jezer

Pronikání vody do jeskynních systémů a její usazování je základním procesem pro vznik jeskynních jezer. Voda vstupuje do podzemí nejen pronikáním skrze horninu, ale také přímo jeskynními závrty, povrchovými puklinami a štěrbinami. Tento proces, pronikání vody hlouběji ve svrchní části zemského povrchu, se nazývá infiltrace a je zvláště výrazný v oblastech s rozpustnými horninami. Skrze jeskynní otvory mohou mimo jiné protékat potoky, nebo případně i řeky, obvykle v závislosti na množství dešťů v současném období. S množstvím dešťů přímo souvisí výška hladiny podzemních vod, která se také často v průběhu roku mění. Zásadní změna výšky hladiny podzemních vod může některé sekce jeskynních systémů buď úplně vysoušet, nebo zaplavovat (viz známé neštěstí způsobené záplavou jeskyně Tham Luang v Thajsku v roce 2018).

Jakmile voda pronikne do jeskynních systémů, dochází k usazování minerálních sedimentů, které mimo stalaktitů a stalagmitů, přispívají k formování podzemních jezer. Jeskynní jezera pak vznikají, když se voda hromadí v rozsáhlých jeskynních dutinách nebo podzemních prostorách, které jsou formovány různými mechanismy, včetně eroze, rozpouštění hornin nebo kolapsu stropů jeskyní. Usazování minerálních sedimentů zde přispívá při formování jeskynních jezer ve chvíli, kdy se voda vypařuje nebo ztrácí svou rozpouštěcí schopnost. Minerální látky, které byly rozpuštěny ve vodě, se pak usazují na dně jeskynních dutin a postupně tvoří sedimentární vrstvy, které zamezují vodě pronikat hlouběji. Reálné jeskynní jezero je na *obr.6*.

Obrázek 6: **Jeskynní jezero**. Reálné jeskynní jezero blízko města Kanab v americkém státě Utah. Převzato z: etbtravelphotography.com

3 Jeskynní systémy v počítačové grafice

V této kapitole se budu věnovat využívání jeskynních systémů v počítačové grafice. Nejvíce budu rozebírat počítačové hry, které začaly jeskynní systémy preferovat jako své herní prostředí a mají tak zásluhu na rozvoji tohoto oboru. Dále rozeberu aktuálně nejpoužívanější metody pro procedurální generování jeskynních systémů a krápníků v počítačové grafice. Budu se zabývat jejich přednostmi, nedostatky, jejich využití a vzájemnému kombinování jednotlivých metod.

3.1 Historie jeskynních systémů v počítačové grafice

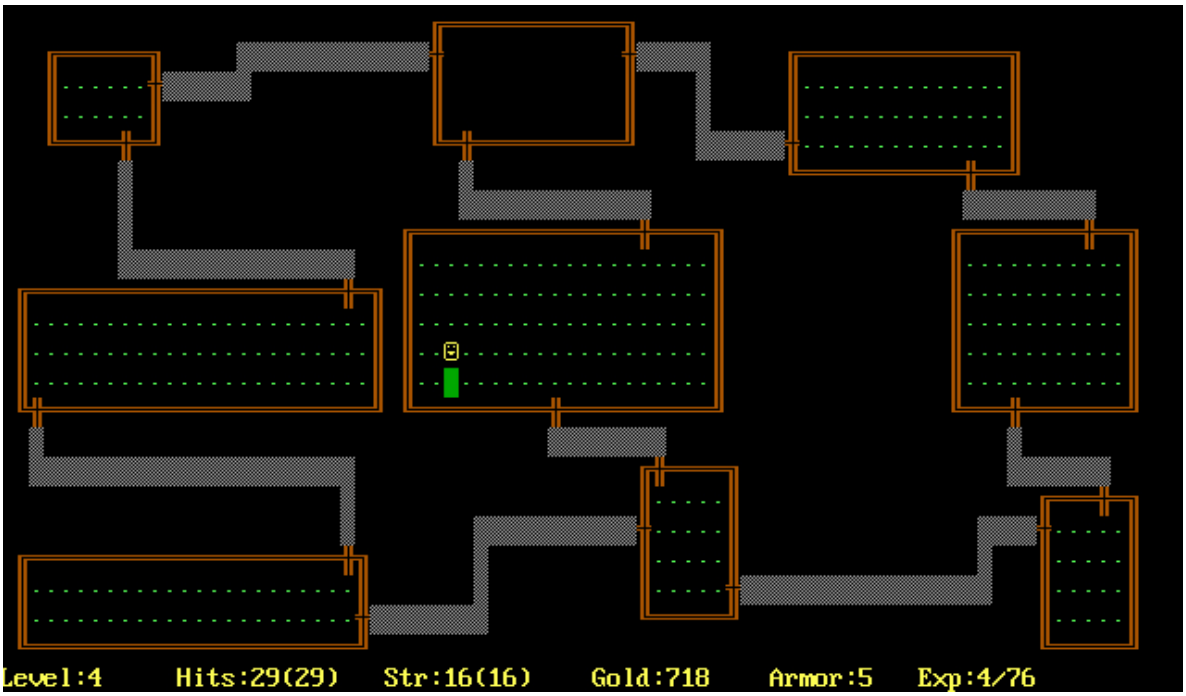
Průzkum počítačem generovaných jeskynních systémů a jejich vizualizace je součástí počítačové grafiky a simulací již několik desetiletí. V oblasti počítačových her se generování jeskyní stalo prominentním již v 70. a 80. letech.

Hra *Colossal Cave Adventure* byla vyvinuta nadšeným jeskyňářem Williamem Crowtherem, který popisoval své zážitky při prozkoumávání Mamutí jeskyně v americkém státě Kentucky a chtěl, aby i ostatní lidé mohli zažívat podobné pocity jako on [6]. Známá délka všech podzemních prostorů této jeskyně sahá dnes k 630 km, což jí řadí na první příčku nejdelších jeskynních systémů světa. Proto se rozhodl vytvořit sekci Mamutí jeskyně v krátké textové rozhodovací hře, čímž v druhé polovině 90. let stvořil nový žánr video her známý jako "textová adventura". Snímek obrazovky ze hry *Colossal Cave Adventure* je na *obr.7*.

```
It is now pitch dark.  If you proceed you will likely fall
into a pit.
>light
Your lamp is now on.
You are in a debris room filled with stuff washed in from the
surface.  A low wide passage with cobbles becomes plugged
with mud and debris here, but an awkward canyon leads
upward and west.  A note on the wall says:
       Magic Word "XYZZY"
>up
You are in an awkward sloping east/west canyon.
>west
You are in a splendid chamber thirty feet high.  The walls
are frozen rivers of orange stone.  An awkward canyon and a
good passage exit from east and west sides of the chamber.
A cheerful little bird is sitting here singing.
>e
You are in an awkward sloping east/west canyon.
>e
You are in a debris room filled with stuff washed in from the
surface.  A low wide passage with cobbles becomes plugged
with mud and debris here, but an awkward canyon leads
upward and west.  A note on the wall says:
       Magic Word "XYZZY"
A three foot black rod with a rusty star on an end lies nearby.
```

Obrázek 7: **Hra Collosal Cave Adventure**. Snímek obrazovky ze hry Collosal Cave Adventure, kde se hráč nachází v jeskyni s cílem nalézt poklad. Převzato z: ostechnix.com

Velkou inspiraci ze hry *Colossal Cave Adventure*, která se stala okamžitým hitem, si vzala hra *Rogue*, která si určitě zaslouží zmínku. Autoři *Rogue* uvedli, že použít pro jejich hru jiné než jeskynní prostředí je ani nenapadlo [6] a rozhodli se zaplnit tuto nově objevenou díru na trhu. Hra využívala procedurální generování k vytváření jeskynních systémů sloužící jako dungeony, kterými se hráč pohyboval. Jedná se o pseudografickou hru, která užívá alfanumerické znaky, jako pomlčky a křížky, pro reprezentaci jeskynních systémů. Hra *Rogue* si také vysloužila po sobě pojmenovaný, dnes velice populární, herní žánr pod názvem "roguelike". Pro tento žánr je typickým procedurálně generovaný obsah, permanentní smrt a důraz na prozkoumávání. Vizuální věrnost byla tehdy ale stále omezená kvůli hardwarovým možnostem. Náhodně vygenerovaná úroveň ze hry *Rogue* je na *obr.8*.



Obrázek 8: **Hra Rogue**. Snímek obrazovky ze hry Rogue, kde se hráč nachází v náhodně vygenerovaném dungeonu a chystá se vstoupit do poslední neprozkoumané místnosti. Převzato z: wikipedia.org

První hry, které přinesly 3D vizualizací něčeho co reprezentovalo jeskynní systémy, byly hry *Myst* a *Doom* v polovině 90. let. Během dalších let, s pokrokem ve výpočetní síle a grafických technologií, pokračovali programátoři a vývojáři her v prozkoumávání a zdokonalování algoritmů pro vytváření a generování realistických jeskynních systémů, což se ale ukazuje být poměrně obtížné. K využití samotného procedurálního generování 3D jeskynních systémů, po dlouhou dobu v žádných videohrách nedocházelo.

3.2 Jeskynní systémy v počítačové grafice nyní

Jeskynní systémy v dnešní počítačové grafice nabízejí vysokou úroveň realismu a komplexnosti. To ale většinou nesouvisí s průlomem ve vytváření samotných jeskynních prostor. Procedurální generování v tomto oboru bývá používané pro dodatečné rozmístění různých běžných objektů, jako jsou krápníky, jezera, kameny, do předpřipravených jeskynních chodeb. Moderní algoritmy pro procedurální generování sice umožňují generovat realistický terén, ale realistické jeskynní systémy s neomezenou vertikalitou, objemem prostorů a dalšími možnostmi zůstávají bez univerzálního řešení.

Existuje pouze hrstka 3D počítačových her, které jeskynní systémy dynamicky kompletně procedurálně generují. Tyto hry však zanedbávají většinu procesů (častěji všechny), které za vznikem jeskyní stojí. Je to z toho důvodu, že jeskynní systémy, které se běžně v přírodě vyskytují, bývají pro hry nevhodné. Typicky jsou buď moc klaustrofobické, úzké, nebo krátké, což není nic co by hráči vyhledávali. Do takových jeskyní je složité vložit jakýkoliv herní obsah a je to jeden z hlavních důvodů, proč se realistické jeskyně ve hrách nevyskytují.

Důsledkem toho se úspěšné hry, které využívají procedurální generování jeskynních systémů, často nezaměřují na realistickou grafiku. Jednou z nejznámějších her, která je na jeskyních založená je hra *Minecraft*, která jeskyně (a řeky) generuje primitivně za pomocí algoritmu Perlinových červů. Dalším příkladem je hra *Deep Rock Galactic*, ta využívá procedurálně generované cesty, které spojují předem připravené místnosti s procedurálně dogenerovaným obsahem. Stejným směrem jako *Deep Rock Galactic* se ostatně vydává většina moderních her a aktuálně se jedná u hráčů o nejúspěšnější a pro vývojáře časově nejméně náročné řešení. Snímek obrazovky ze hry *Deep Rock Galactic* je na *obr.9*.

Obrázek 9: **Hra Deep Rock Galactic**. Procedurálně vygenerovaná jeskyně ve hře Deep Rock Galactic, ve které jsou rozmístěny krápníky, kameny a svítící organismy. Převzato z: accursedfarms.com

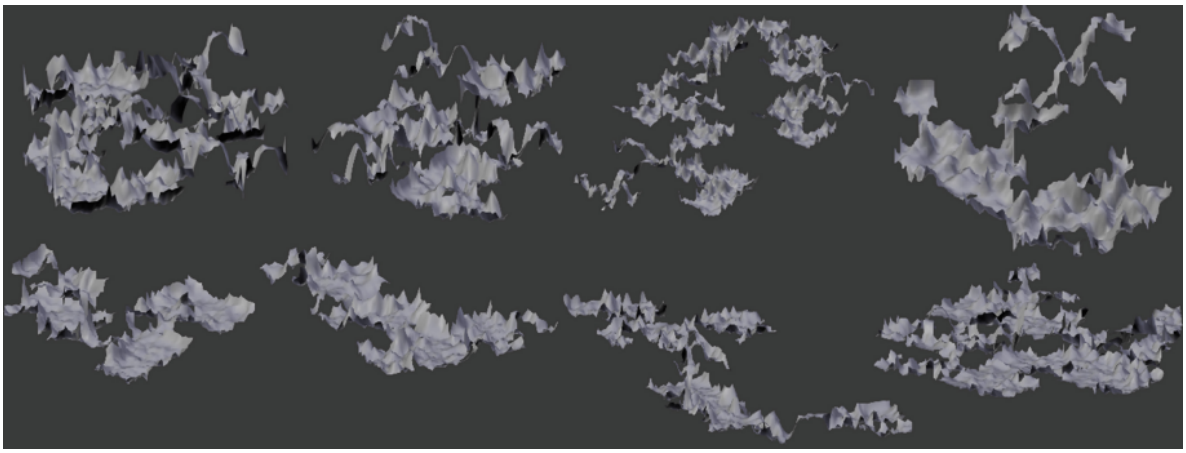
I přes výše jmenované nedostatky se najdou práce, aplikace a výzkumy, které se pokoušejí procedurální generování jeskynních systémů zdokonalovat a posunout toto odvětví počítačové grafiky kupředu. Hlavním důvodem zůstává touha po rychlém vytváření detailních a realistických prostředí bez manuálního modelování. Toho je s moderním hardwarem často možné dosáhnout v reálném čase. Velkou výhodu dává všem novějším výtvorům obecný pokrok ve vykreslování realistických scén v herních enginech. Jedná se například o podporu pro vysoce kvalitní textury nebo dodatečné efekty osvětlení a stínování, což zásadně zvyšuje autenticitu prostředí.

3.3 Metody procedurálního generování jeskynních systémů

V současnosti v počítačové grafice existuje několik hlavních metod pro procedurální generování realistických jeskynních systémů, které převažují nad ostatními. Konkrétně se jedná se o metody pracující s L-systémy, fraktálními algoritmy, fyzikálními simulacemi a algoritmy pro nejlevnější cestu. Všechny tyto metody se často kombinují pro vytvoření komplexnějších a realističtějších jeskynních systémů. Moderní přístupy využívají tyto metody také v kombinaci s umělou inteligencí, která se v budoucnosti jeví jako možné finální řešení tohoto problému. To ale není předmětem této práce.

3.3.1 L-systémy

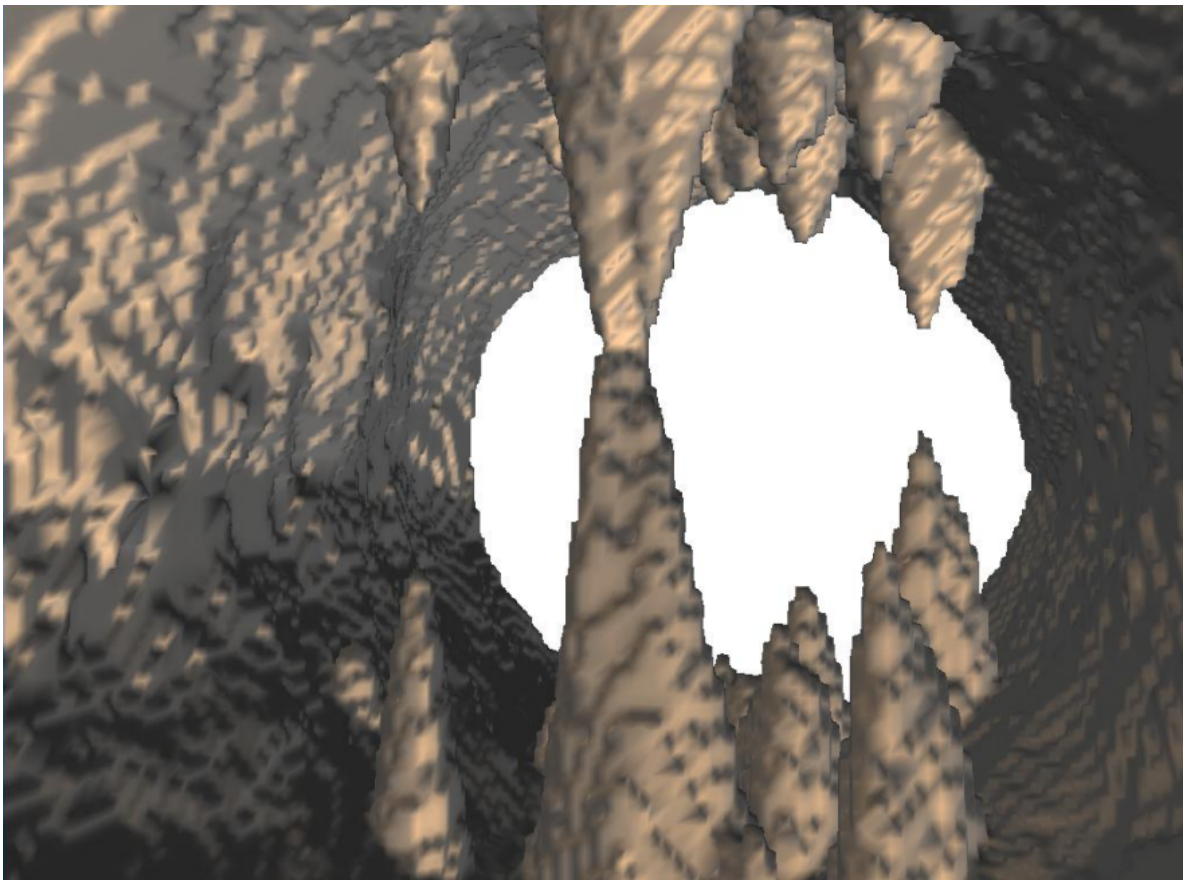
L-systémy jsou formální gramatiky používané pro modelování rostlin nebo struktur. Skládají se z počátečního axiomu a množiny pravidel, která popisují, jak se axiom rozvíjí v závislosti na předchozím stavu. Tento proces se opakuje iterativně, což vede k vytvoření detailních struktur. V kontextu jeskynních systémů mohou být pravidla L-systémů modifikována tak, aby simulovala procesy růstu a tvorby jeskynních prostor, což ukázali autoři článku [3], kde využili L-systémy v kombinaci s celulárními automaty k vytvoření jeskyní a krápníků (*obr.10*). L-systémy umožňují modelování různých typů jeskynních formací a chodeb, a poskytují flexibilitu při tvorbě komplexních jeskynních systémů.



Obrázek 10: **Jeskynní systémy vygenerované pomocí L-systému**. Všechny tyto jeskynní systémy byly vygenerovány pomocí L-systému a celulárního automatu. Převzato z: [3]

3.3.2 Fraktální algoritmy

Fraktální algoritmy, jako například fraktální šum, Perlinův šum, případně Perlinovy červi (algoritmus, používaný hrou Minecraft, pro generování řek a jeskynních systémů), nebo Simplexový šum, který se často využívá k vytváření realistických textur a struktur. Perlinův šum funguje na principu vrstvení různých frekvencí šumu, což vytváří detailní a přirozeně vypadající povrchy. Autor [4] použil Perlinův šum a voxelovou reprezentaci (kterou nakonec algoritmem vyhladil) k procedurálnímu generování jeskyní a krápníků (*obr.11*). Perlinovy červi fungují na principu opakované volby náhodných směrových vektorů a délky kroku. Simplexový šum je novější alternativa k Perlinově šumu, která je v mnohých případech rychlejší a méně náchylná na periodické vzory. Určitými způsoby lze použít i celulární automaty, které mohou simulovat růst struktur podle určitých předem definovaných pravidel, což může vést k vytváření komplexních a rozmanitých terénů. Přestože tyto algoritmy jsou mocné nástroje pro generování terénních útvarů, jejich použití pro vytváření jeskynních systémů může být náročné. Zahrnutí geologických podmínek, jako jsou různé typy hornin, podzemních vod, tektonické pohyby a erozní procesy, je složité a vyžaduje sofistikovanější přístup k algoritmické tvorbě terénu.



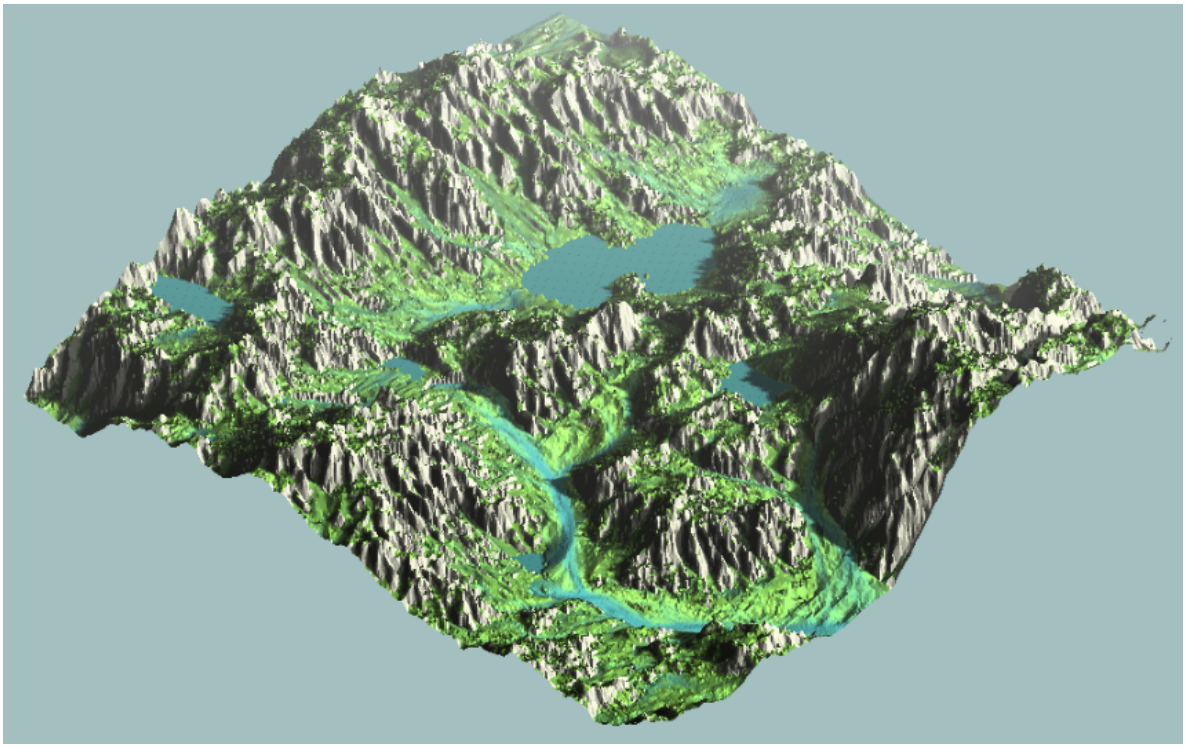
Obrázek 11: **Jeskynní ulička vygenerovaná pomocí Perlinova šumu**. Tato ulička byla vygenerovaná za použití Perlinova šumu, objemového pole a algoritmu podobného Marching Cubes. Krápníky byly taktéž generovány s užitím Perlinova šumu. Převzato z: [4]

3.3.3 Fyzikální simulace

Simulace v jejichž prostředí se detailně zkoumají procesy, které formují skály a jeskynní struktury v reálném světě. Tyto simulace umožňují porozumět přírodním mechanismům, které stojí za tvorbou skalních útvarů, a napodobit je za účelem vytváření realistických virtuálních prostředí. V kontextu jeskynních systémů jsou některé z klíčových fyzikálních procesů, které se v těchto simulacích zahrnují, následující:

- Hydraulická eroze: Simulace zaměřující se na působení vody na skalní povrch a následně na vytváření jeskynních struktur. Například simulace kapek vody, které dopadají na povrch, může modelovat postupné erozní účinky, což detailně rozebírá autor [11]. Eroze pak vede ke vzniku různých tvarů, jako jsou jámy a chodby, typické pro jeskyně vytvořené vodní erozí. Výstup ze simulace hydraulické eroze na terénu je na *obr.12*.

- Větrná eroze: Simulace větrné eroze se potýká s tím, jak vítr ovlivňuje povrch skal a jakým způsobem tvoří charakteristické formace, jako jsou dutiny a reliéfy. Větrná eroze je klíčovým faktorem při formování skalních útvarů v aridních a větrných oblastech, a tak její simulace přispívá k pochopení procesů tvorby jeskynních struktur v různých klimatických podmínkách.



Obrázek 12: **Terén vygenerovaný pomocí fyzikální simulace**. Tento terén byl vygenerován iterativně simulovaním hydraulické eroze. Podobný postup se dá využít i pro jeskyně. Převzato z: nickmcd.me

3.3.4 Algoritmus pro nejlevnější cestu

Metodu Algoritmu pro nejlevnější cesty rozeberu více do hloubky, protože je pro tuto práci nejdůležitější a bude se mu v dalších kapitolách věnovat. Tento postup, který navrhují autoři [12], se skládá v první části z konstrukce kostry jeskyně.

Nejdříve je potřeba diskrétně rozdělit prostor tak, aby bylo možné na něm vytvořit graf ze vzájemně nejbližších bodů. Toho lze docílit například metodou Poissonových koulí, která systematicky náhodně rozmísťuje do prostoru koule tak, že se vzájemně nedotýkají. Vzájemně nejbližší koule se pak dají propojit tak, aby bylo možné středy koulí použít jako uzly v grafu. Dále následuje hledání nejlevnějších cest při průchodu grafu s určitou heuristikou zvolenou tak, aby napodobovala geologické podmínky. Nakonec se ještě provádí prořezávání nevyhovujících cest a rozvětvení pro získání dodatečných slepých cest. Výsledkem je potom kostra jeskynního systému.

V druhé části je za použití získané kostry potřeba vytvořit diskrétní objemovou reprezentaci prostoru. Tu lze získat například algoritmem Primitive Sweeping, který užívá primitiva a prochází postupně prostor po kostře v malých krocích. V místech do kterých v objemové reprezentaci zasahují primitiva, změní uloženou hodnotu. Objemová reprezentace je na závěr využita algoritmem Marching Cubes (nebo jinými alternativy, např. Marching Tetrahedra) pro konstrukci trojúhelníkové sítě.

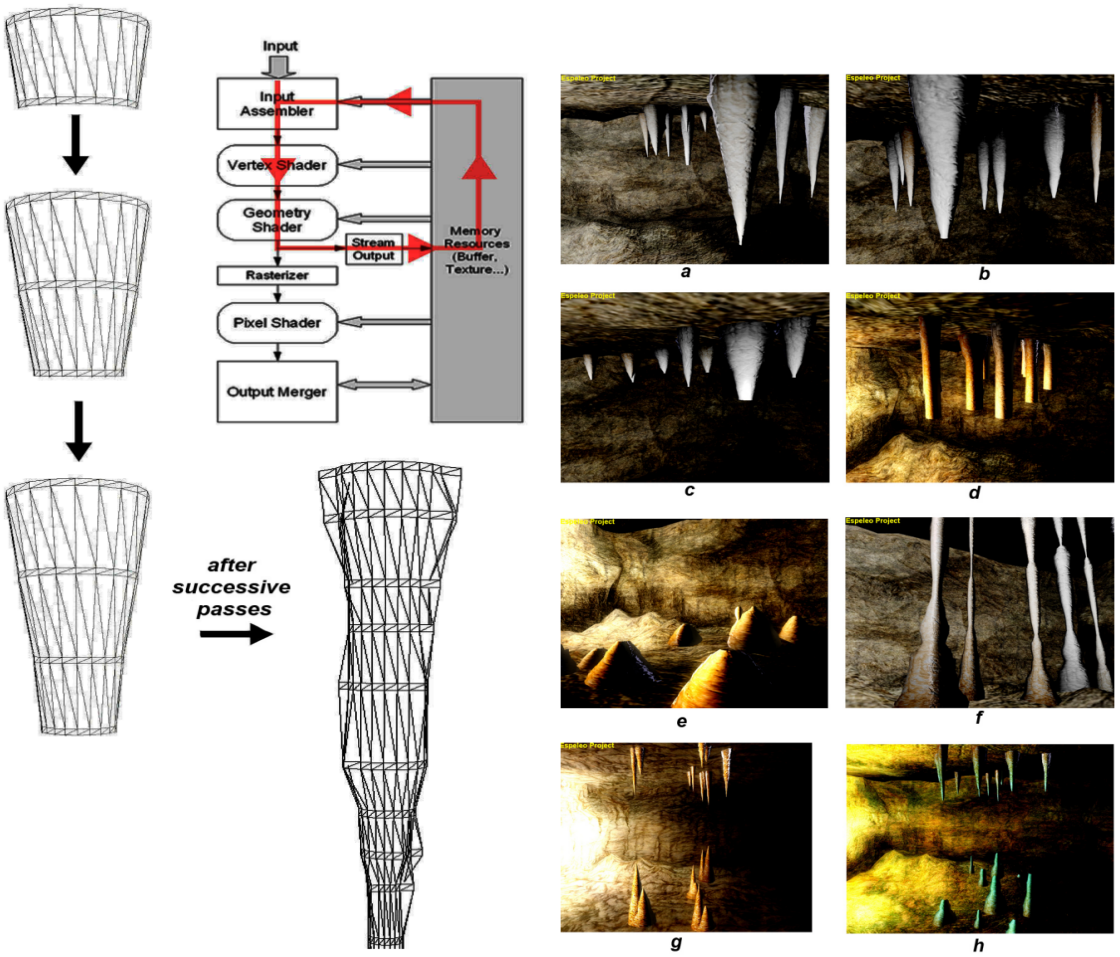
Nakonec zbývá vykreslit a otexturovat trojúhelníkovou síť jeskynního systému v reálném čase, pro což jsou trojúhelníkové sítě dělané (autoři [12] využili metodu Signed Distance Function, která není vhodná pro aplikace běžící v reálném čase). K vykreslení lze použít jakýkoliv vykreslovač, který přijme jako vstup trojúhelníkovou síť. Otexturování je pak možné provést triplanárním mapováním a nanést tak všechny typy textur na povrch jeskynního systému. Výsledek nadcházející implementace je na *obr.13*.

Obrázek 13: **Jeskynní systém vygenerovaný pomocí Algoritmu pro nejlevnější cestu**. Výsledná jeskyně generátoru naimplementovaného v této práci, který využívá metody Algoritmu pro nejlevnější cestu.

3.4 Metody procedurálního generování krápníků

Existuje pouhá hrstka prací, které se věnují procedurálnímu generování krápníků. Implementované metody se pohybují okolo vytvoření kuželovité trojúhelníkové sítě a nějakou formou aplikování posunutí na vrcholy. Autoři [14] kuželovité krápníky sestavují pomocí prstenců, které posílají do geometrického shaderu. Postupně tyto prstence staví nad sebe, propojují a jednotlivé vrcholy posouvají. Vrcholům také přiřazují barvy v závislosti na barvách různých hornin uvnitř jimi vytvořené tabulky. Práce je z roku 2009 a v dnešní době by takovou metodu nad stejnými daty bylo dle mého názoru rychlejší provést na CPU z důvodu pokroků v architektuře procesorů a multi-threadingu (užívání mnoha vláken). Tato metoda je znázorněna na *obr.14*.

Generování krápníků provádí také autor [4], který pro generování krápníků používá objemové pole. Do tohoto pole vloží tvar, který připomíná kužel, a pak ho rozptýlí Perlinovým šumem. Na závěr z objemového pole zkonstruuje trojúhelníkovou síť pomocí algoritmu, který je podobný algoritmu Marching Cubes. Jeho výsledek je na *obr.11* v předchozí sekci.



Obrázek 14: **Krápníky vygenerované pomocí GPU**. Na levém obrázku je naznačen proces generování krápníků metodou propojování prstenců, která probíhá v geometrickém shaderu na GPU. Na pravém obrázku jsou vidět výsledné otexturované a obarvené krápníky vygenerované touto metodou umístěné uvnitř jeskyně. Převzato z: [14]

4 Návrh Řešení a Implementace

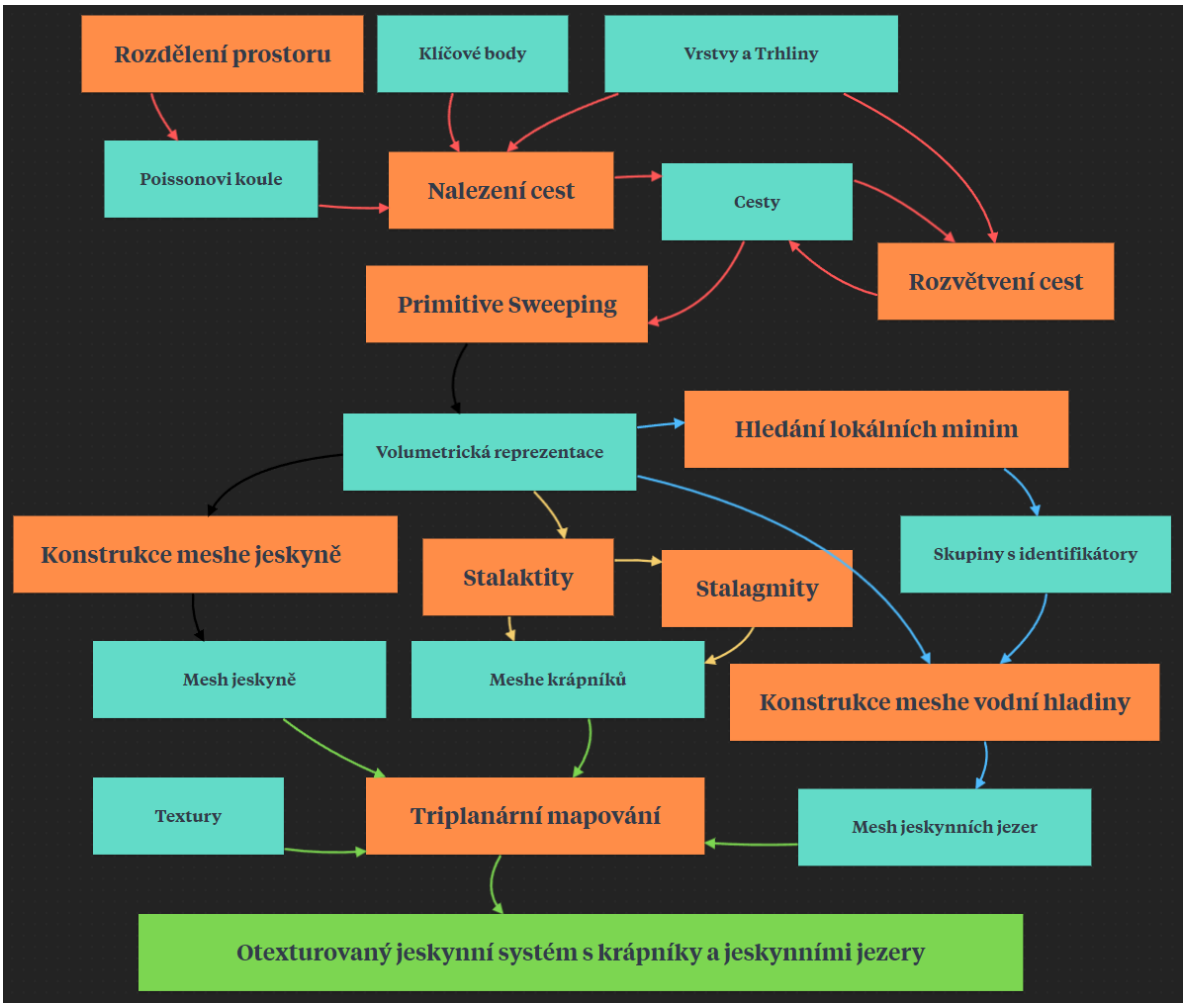
Pro účely této práce jsem se rozhodl provádět implementaci užitím metody Algoritmu pro nejlevnější cestu a budu ho dál detailně rozebírat. V této kapitole se budu věnovat návrhu řešení jednotlivých části Algoritmu pro nejlevnější cestu a popisovat postup při implementaci mého nástroje. Pro samotný nástroj jsem původně zamýšlel vytvořit vlastní vykreslovač (renderer), což se ale ukázalo z důvodu časové náročnosti nerealistické. Rozhodl jsem se tedy implementaci provést v Unity Engine. Unity je multiplatformní herní engine vyvinutý společností Unity Technologies. Unity Engine jsem, oproti alternativám (Unreal Engine 5, Godot), zvolil z důvodu několika let předchozích zkušeností. Navíc Unity má v sobě zabudovaných spoustu vymožeností, které mi značně usnadnily celou implementaci. Např.:

- **Skriptování v C#** - C# je dobře strukturovaný moderní jazyk, který nabízí dobrou čitelnost a srozumitelnost. Je také plně integrovaný s Unity API, což umožňuje snadný přístup k interním funkcím a proměnným.
- **Unity Shader Graf** - Nástroj v Unity, který umožňuje vytvářet a vizualizovat shaderové efekty a materiály bez nutnosti psaní kódu. Poskytuje uživatelsky přívětivé prostředí, kde lze vytvářet složité vizuální efekty pomocí propojování uzlů.
- **Debug systém**
  - Breakpointy - Unity umožňuje propojení s Visual Studio a umísťování breakpoint do jednotlivých skriptů.
  - Logování - Disponuje také velice vyspělým systémem logování a zobrazování jednotlivých hodnot.
  - Profiler - Součástí Unity je také profiler, který poskytuje informace o spotřebě CPU, paměti, grafických zdrojích (GPU), alokaci paměti, počtu vykreslených objektů atd.
  - Gizmos - Jsou v Unity vizuální prvky, které slouží k vizualizaci různých aspektů herního světa nebo komponent. Tyto vizuální prvky jsou viditelné pouze v editoru.
- **Dokumentace** - Jedna z nejsilnějších stránek Unity Enginu je dokumentace. Je velmi obsáhlá a zahrnuje informace a návody pro všechny hlavní funkce, komponenty a rozhraní, které Unity nabízí pro vývoj.

4.1 Generování kostry jeskynního systému

Pro generování jeskynního systému jsem zvolil metodu Algoritmu pro nejlevnější cestu. Jedním z důvodů je kombinace zachování kontroly uživatelem a snaha o napodobení výsledků fyzikálních simulací. Implementace generování kostry je inspirována metodami popsanými v práci [12], kde autoři také zvolili tento postup.

Řešení generování kostry spočívá v následujícím. Z prostoru daného uživatelem získáme konečnou množinu vzorků, které vhodně reprezentují geologické vlastnosti horniny. Poté s pomocí algoritmu A* a heuristiky, skládající se z geologických vlastností vzorků a koeficientů, budeme mezi nimi hledat optimální cesty. Následně provedeme prořezání, při kterém se zbavíme nadměrného množství vzniklých smyček v grafu. Na závěr provedeme rozvětvení, kdy k neprořezaným cestám přidáme dodatečné slepé cesty, které zajistí komplexnější kostru. Tato část řešení je znázorněna červenými šipkami v diagramu na *obr.15*, ke kterému se budu v dalších částech řešení odkazovat. V následujících sekcích je detailně popsán postup, který jsem při generování kostry jeskyně využil.



Obrázek 15: **Diagram řešení**. Zde je celé řešení vyobrazené jako diagram. Oranžové boxy zastávají důležité kroky při generování jeskynního systému, modré boxy reprezentují různá data a zelený box je výsledek. Červené šipky označují sekci generování kostry, černé generování trojúhelníkové sítě jeskyně, žluté generování krápníků, modré generování jeskynních jezer a zelené šipky vykreslování a skládání do výsledného jeskynního systému.

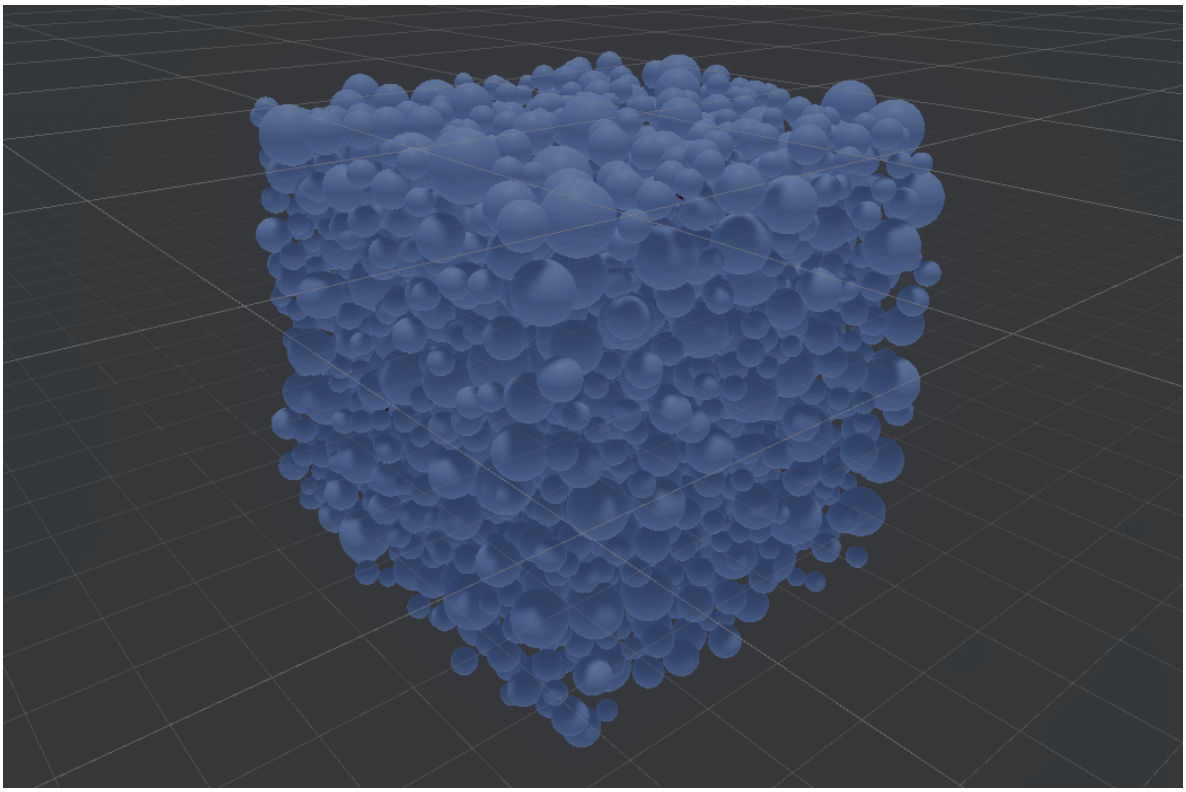
4.1.1 Rozdělení prostoru

Prvním nezbytným krokem je diskrétní rozdělení daného prostoru umísťováním koulí tak, aby se jednotlivé koule nedotýkaly a prostor byl jimi relativně rovnoměrně pokrytý. Metoda užitá v [12] a také metoda, kterou jsem sám použil, se nazývá metoda Distribuce Poissonových Koulí [8].

Budeme generovat koule, které mají náhodný poloměr v definovaném rozmezí. Celý prostor si rozdělíme do 3D pole, ve kterém bude mít jedna buňka rozměry takové, aby její tělesová úhlopříčka měla délku stejnou jako minimální možný poloměr koule. Tím zajistíme, že v jedné buňce 3D pole se může nacházet pouze jedna koule. Do ohraničeného prostoru umístíme jednu kouli doprostřed a přidáme jí do fronty generujících koulí a do 3D pole. Poté opakujeme následující proces, dokud fronta není prázdná: Vybereme náhodný poloměr v rozmezí a sečteme ho s poloměrem generující koule. Výsledek součtu vynásobíme s náhodným směrovým vektorem. Součet pozice generující koule a výsledku násobení je pozice nové koule.

Navrhovanou kouli uznáme za vyhovující v případě, že se nedotýká žádné jiné koule. To zjistíme tak, že vytvoříme opsanou krychli (často nazýváno bounding box) kouli, která má stejný střed jako navrhovaná koule, ale má poloměr větší o $maxRKoule - minRKoule$. Poté ověříme, že se nenachází žádná další koule ve všech buňkách 3D pole, do kterých vytvořená krychle zasahuje. V případě, že navrhovaná koule tuto podmínku splňuje, vložíme ji do fronty generujících koulí, do seznamu všech koulí a do buňky v 3D poli, ve které se nachází její střed. Následně pokračujeme pokusem o generování další koule ze stejné generující koule.

V případě, že navrhovaná koule nevyhovuje podmínkám, zahodíme jí a opět pokračujeme pokusem o generování další koule ze stejné generující. Nepodaří-li se nalézt vyhovující kouli v určitém množství pokusů, přejdeme na další generující kouli ve frontě a pokračujeme dále.



Obrázek 16: **Poissonovy koule**. Diskrétní rozdělení prostoru za použití metody Poissonových koulí, kde se jednotlivé koule nedotýkají. Středy koulí jsou uzly, později používané pro nalezení nejkratších cest.

4.1.2 Spojení sousedních bodů v prostoru

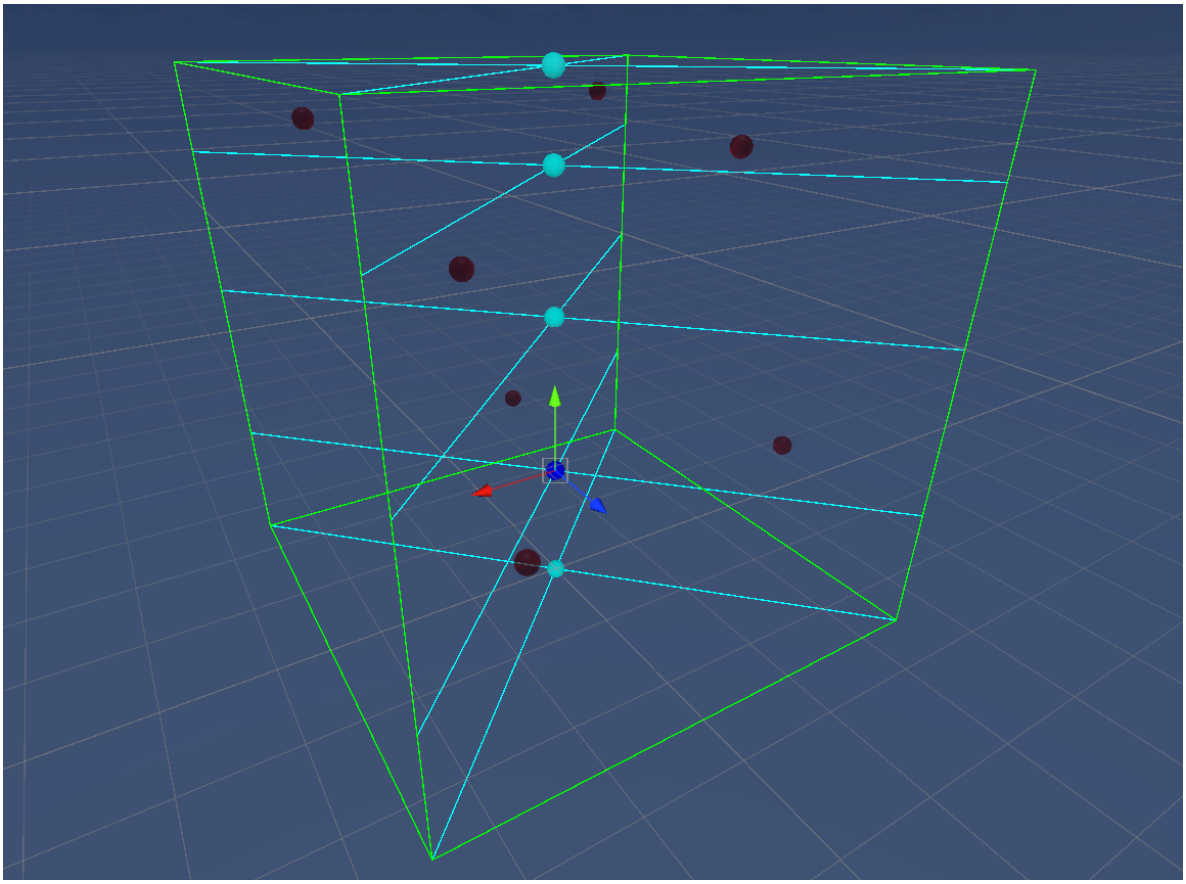
Nyní, když máme prostor rovnoměrně rozdělený, musíme zajistit, aby každá koule měla odkaz na určitý počet svých nejbližších sousedů. Chceme na těchto koulích vytvořit graf pro průchod, který později využijeme při hledání nejlevnějších cest.

Pro každou kouli ze seznamu nejdříve prozkoumáme okolní buňky v určitém poloměru a každou nalezenou kouli vložíme do haldy, která všechny koule seřadí od nejbližších po nejvzdálenější. V případě že halda neobsahuje dostatečné množství koulí, prohledávanou oblast rozšíříme a pokračujeme v prohledávání ve větší vzdálenosti. Nakonec vyjmeme z haldy požadovaný počet nejbližších koulí a do zkoumané koule si na ně uložíme odkazy. Použitím haldy zajistíme řazení se složitostí $n * log(n)$.

4.1.3 Důležité pojmy

V této sekci definuji pojmy, které je nezbytné chápat pro nadcházející kapitoly a budou dále běžně užívány.

- **Klíčové body:** Klíčové body jsou body, kterými bude jeskyně určitě procházet. Uživatel si zvolí libovolný počet klíčových bodů a rozmístí je do prostoru. Jsou to červené body na *obr.17*.
- **Vrstvy - Permeabilita prostředí:** Vrstvy jsou umístěny do určité výšky, kde definují permeabilitu prostředí. Permeabilita vyjadřuje, jak snadné je pro přírodní procesy narušovat horninu, což eventuálně vede ke vzniku jeskyně. Uživatel si může zvolit počet vrstev a umístit je do libovolné výšky. Mezi jednotlivými vrstvami lze různými způsoby interpolovat. Jsou to modré body s horizontálními čárami do kříže na *obr.17*.
- **Trhliny - Natočení krystalové mřížky v hornině:** Předpokládáme, že natočení krystalové mřížky uvnitř horniny je uniformní v celém regionu. Praskliny, které v horninách vznikají, jsou s největší pravděpodobností rovnoběžné se stěnami krystalů. Uživatel si může vybrat normály k těmto stěnám.



Obrázek 17: **Ovládání v Unity editoru**. Červeně jsou vyobrazené klíčové body, které uživatel přesune do míst, kudy chce, aby jeskyně vedla. Modře jsou vyobrazené vrstvy, u kterých si uživatel zvolí výšku a permeabilitu.

4.1.4 Generování cest

Jednotlivé klíčové body potřebujeme propojit mezi sebou a vytvořit tím cesty. K tomu využijeme průchod připraveným grafem.

Pro každou nově tvořenou cestu, zahrnující pár klíčových bodů, najdeme nejbližší kouli v grafu pro oba klíčové body. První označíme jako počáteční bod a druhou jako konečný bod. Nutno podotknout, že samotné klíčové body v grafu neleží, jedná se o samostatné objekty ve scéně. Následně pomocí algoritmu A* budeme chtít tyto body při průchodu grafem propojit.

Algoritmus A* je postavený na minimalizaci součtu cen: $g(a_n)$ cena cesty z počátečního bodu a_0 do aktuálního bodu a_n v grafu a heuristiky $h(a_n)$, neboli cena, kterou bude stát cesta z bodu a_n do konečného bodu a_k (detailnější vysvětlení algoritmu A* je např. zde [16]). Vzorec pro získání celkové ceny $f(a_n)$ je tedy $f(a_n) = g(a_n) + h(a_n)$. Vzorce pro výpočet $g(a_n)$ a $h(a_n)$, které použijeme, budou následující:

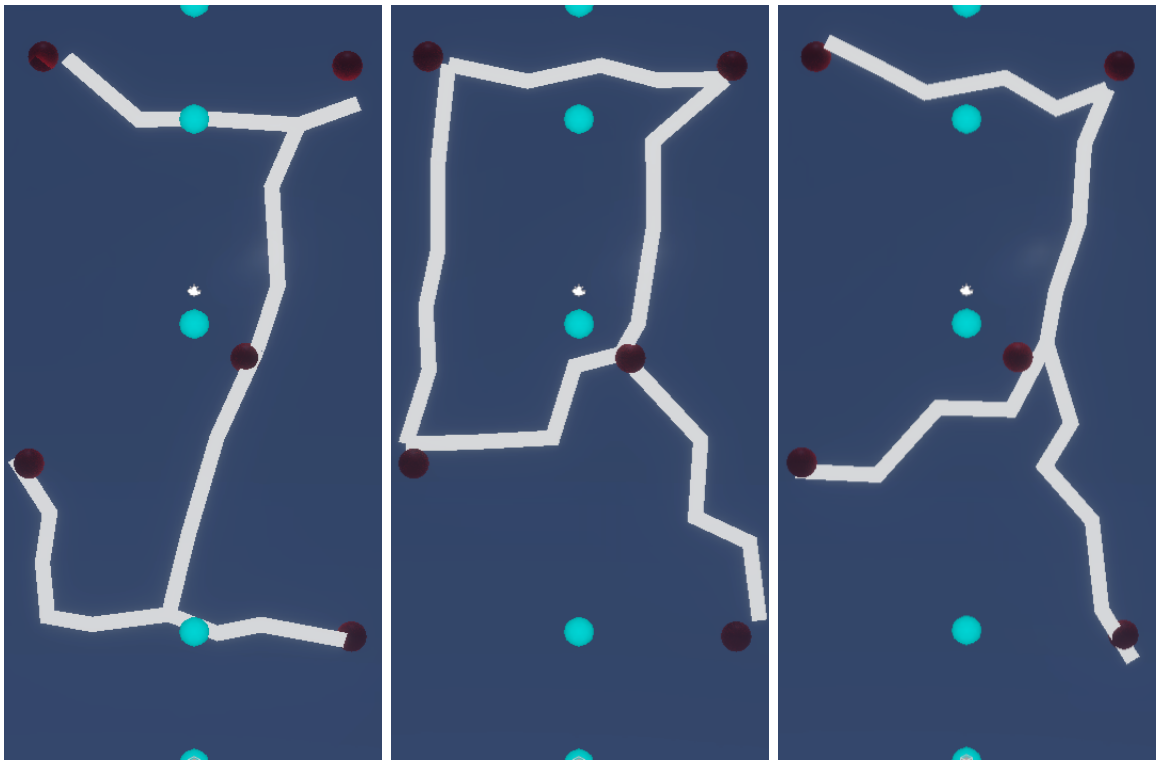
$$g(a_n) = g(a_{n-1}) + ||pozice(a_n) - pozice(a_{n-1})|| \\ *(1 + cenaVrstev(a_n) * p_1 + cenaTrhlin(a_{n-1}, a_n) * p_2)$$

$$h(a_n) = ||pozice(a_n) - pozice(a_k)|| * (1 + cenaVrstev(a_k) * p_1 + cenaTrhlin(a_n, a_k) * p_2)$$

kde p_1 a p_2 jsou nastavitelné parametry (váhy).

Při výpočtu je normalizovaný součet cen vynásoben délkou hrany, což zajistí, konzistentnost ceny v závislosti na uražené vzdálenosti. Toho využívají i autoři [5], kteří se pokoušejí vylepšit metody z práce [12]. Tato shoda vylepšení vzorce pro výpočet ceny je však pouhou náhodou. Implementaci této části jsem provedl a odevzdal jako semestrální projekt několik měsíců před vydáním tohoto článku.

Každou nejlevnější cestu, kterou algoritmus najde, nakonec uložíme do seznamu cest. Celý tento proces hledání cest je vhodný pro paralelizaci, protože hledání jedné cesty je nezávislé na na hledání jiných cest. Tuto paralelizaci jsem provedl pomocí nativní C# knihovny *System.Threading.Tasks*, která poskytuje užitečné funkce, jako například *Parallel.For*. Ukázka vlivu ceny vrstev a trhlin je na *obr.18*, kde jsou všechny klíčové body pro lehčí zobrazení umístěny v jedné rovině.



(a) Maximální cena vrstev, nulová cena trhlin.

(b) Nulová cena vrstev, maximální cena trhlin.

(c) Cena trhlin a vrstev vybalancovaná.

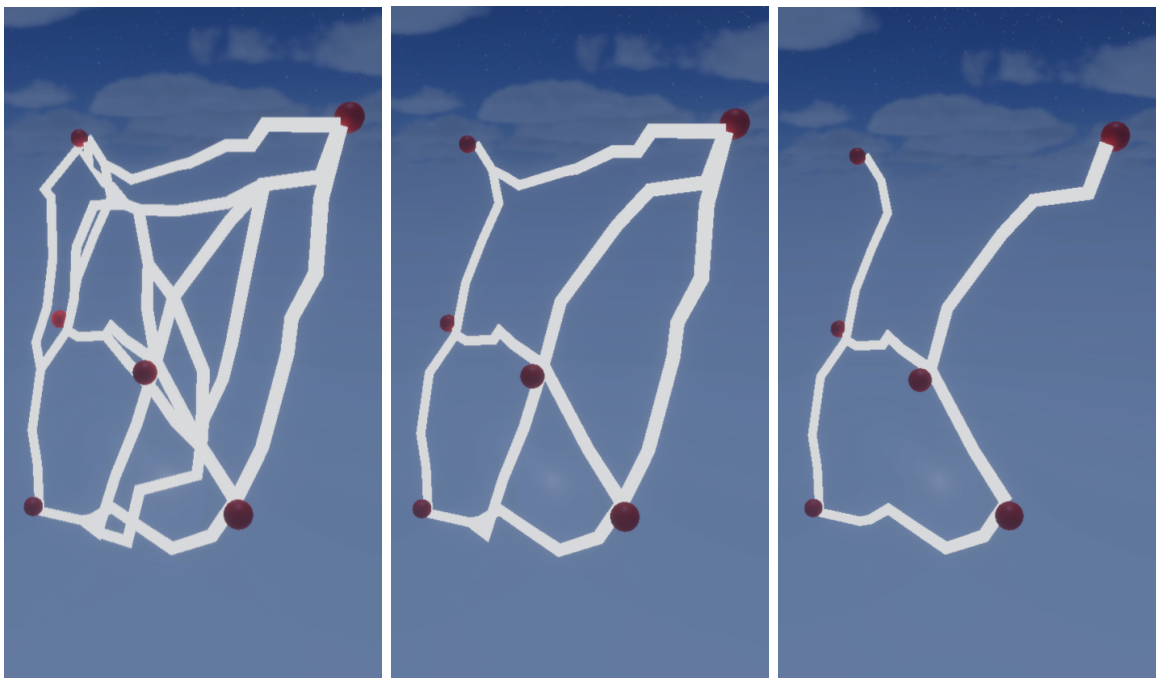
Obrázek 18: **Vliv ceny vrstev a trhlin**. Vliv ceny vrstev a trhlin vyobrazený pomocí ortogonální projekce se všemi klíčovými body v jedné rovině. Mezi generováním cest se mění jen parametry koeficientů cen horizontů a vrstev, všechny ostatní parametry zůstávají stejné. Na všech obrázcích je pět vrstev (modré koule), přičemž druhá a čtvrtá mají nastavenou cenu na 1 a první, druhá a pátá na 0. Na *obr.*(a) se cesty drží levných vrstev, na *obr.*(b) se cesty snaží pohybovat v kolmých směrech a na *obr.*(c) balancují mezi obojím.

4.1.5 Prořezávání cest

Máme propojeny všechny klíčové body cestami, které simulují, kudy by nejspíše vedla jeskyně, pokud by v reálném světě mezi těmito body vznikala. Cest ale může být nežádoucí velké množství. Autoři [12] proto navrhují řešení, které ponechá pouze ty cesty, co budou vyhovovat následující podmínce: Pro každé dvě cesty, pro které platí, že jedna z cest vede z libovolného bodu a do libovolného prostředního bodu p a druhá z p do libovolného bodu b, provedeme následující:

- Pro každou jinou cestu vedoucí z bodu a do bodu b musí platit:
  - $Cena(a,b)^j < Cena(a,p)^j + Cena(p,b)^j$
  - kde $Cena(x,y)$ je celková cena cesty z bodu x do bodu y a j je prořezávací exponent.
- V případě nesplnění podmínky tuto cestu odstraníme ze seznamu cest (prořežeme).

Ukázka vlivu prořezávacího exponentu je na *obr.19*.

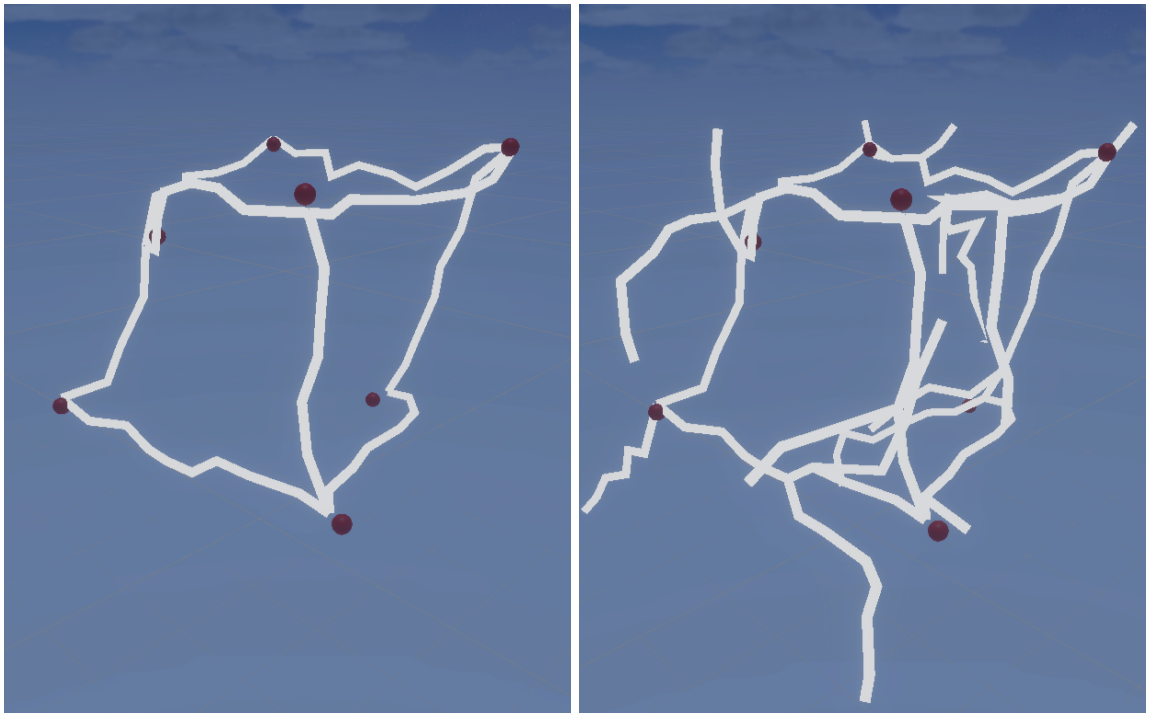


(a) Prořezávací exponent: 0 (b) Prořezávací exponent: 2 (c) Prořezávací exponent: 6

Obrázek 19: **Vliv přořezávacího exponentu**. Vliv prořezávacího exponentu je vyobrazený nad stejným grafem. Na obr. (a) je prořezávací exponent nastaven na nulu, tedy žádné prořezávání neprobíhá a každý klíčový bod je napojený na každý jiný klíčový bod přímou cestou. Na obrázcích (b) a (c) je prořezávací exponent nastaven na dvojku a na šestku a probíhá prořezávání nevyhovujících cest.

4.1.6 Rozvětvení cest

Protože naším cílem je vytvořit komplexní jeskynní systém, přidáme i některé slepé vedlejší uličky. Budeme simulovat jejich vznik v okolí již nalezených cest. Pro každý uzel na každé již nalezené cestě provedeme s danou pravděpodobností rozvětvení. Zvolíme náhodný uzel v grafu v daném poloměru a nalezneme nejlevnější cestu mezi těmito dvěma uzly pomocí algoritmu na generování cesty a přidáme novou cestu do seznamu cest. Podobně lze vytvořit i větve na větvích s neomezeným množstvím iterací. Výsledek rozvětvení je na *obr.20*.



Obrázek 20: **Vizualizace rozvětvení cest**. Na levém obrázku je vygenerovaná kostra stejného jeskynního systému před rozvětvením a na pravém obrázku po rozvětvení. Cesty prochází skrze body v grafu, které jsou ke klíčovým bodům nejblíže. Z toho důvodu se místy klíčové body přímo cest nedotýkají.

4.2 Generování trojúhelníkové sítě

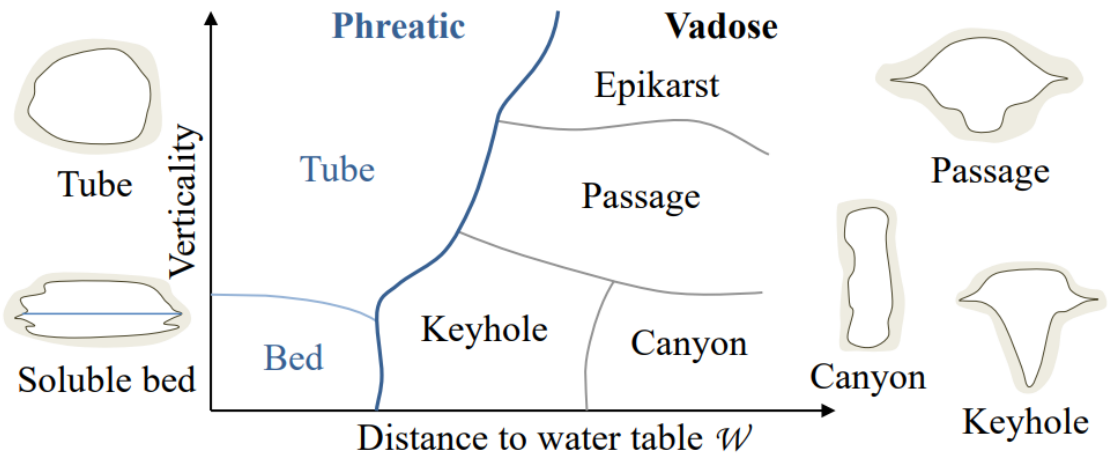
Aktuálně máme seznam nalezených cest, kde každá cesta prochází konečnou množinou bodů z grafu. Cesty jsou reprezentovány seznamem těchto bodů z grafu tak, že hrana je vždy mezi dvěma sousedními body v seznamu.

Navrhované řešení pro generování trojúhelníkové sítě jeskyně je následující. Pro konstrukci trojúhelníkové sítě použijeme algoritmus Marching Cubes, se kterým mám předchozí zkušenosti. Marching Cubes požaduje jako vstup 3D pole hodnot, které leží v rozmezí od 0 do 1. Potřebujeme tedy získat diskrétní objemovou reprezentaci jeskyně z vytvořené kostry. Jako řešení se nabízí užití metody Primitive Sweeping, což je metoda, při které budeme používat primitivum. Primitivum bude mít tvar kolmého průřezu jeskyně a budeme jím posouvat po krátkých krocích po jednotlivých cestách. Tato část řešení je znázorněna černými šipkami v diagramu na *obr.15*. V následujících sekcích je detailně popsána implementace pro generování trojúhelníkové sítě, kterou jsem prováděl.

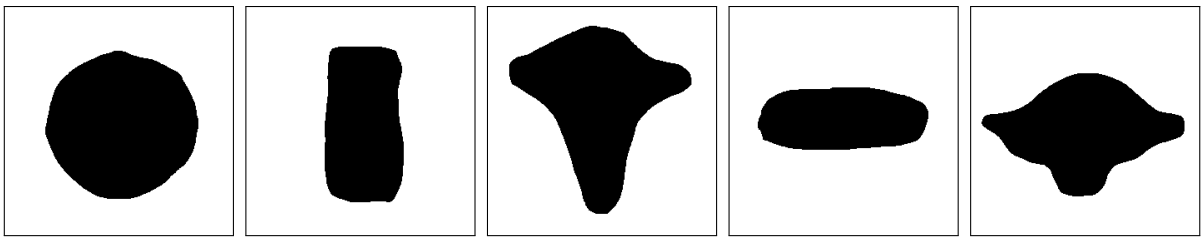
4.2.1 Tvorba primitiv

Pro využití techniky Primitive Sweeping budeme potřebovat primitiva a body na cestách, po kterých budeme primitiva posouvat. U primitiv budeme rozlišovat několik tvarů jeskynních chodeb, ze kterých budeme v každém bodě vybírat v závislosti na dvou podmínkách. Jedná se o strmost tečny grafu v aktuálním bodě a vzdálenost aktuálního bodu od výšky hladiny podzemních vod. Konkrétní tvary, které budeme používat a použili je i autoři [12], jsou: podloží (bed), trubka (tube), klíčová dírka (keyhole), kaňon (canyon) a průchod (passage). Graf pocházející z jejich práce je na *obr.21*.

Samotná primitiva budeme konstruovat pomocí metody Poissonových Disků, což je stejný algoritmus jako algoritmus Poissonových Koulí pouze nad dvojrozměrným prostorem. Tentokrát tedy vytvoříme pouze 2D pole a budeme do něho rozmísťovat analogickým způsobem jako v kroku 4.1.1 kruhy s tím rozdílem, že vždy ještě ověříme, zda střed navrhovaného disku leží uvnitř masky průřezu jeskyně *obr.22*. Nové primitivum zkonstruujeme v každém bodě podle vybrané předlohy znovu, abychom se vyhnuli repetitivnímu vzhledu. Zkonstruovaná primitiva jsou na *obr.23*.

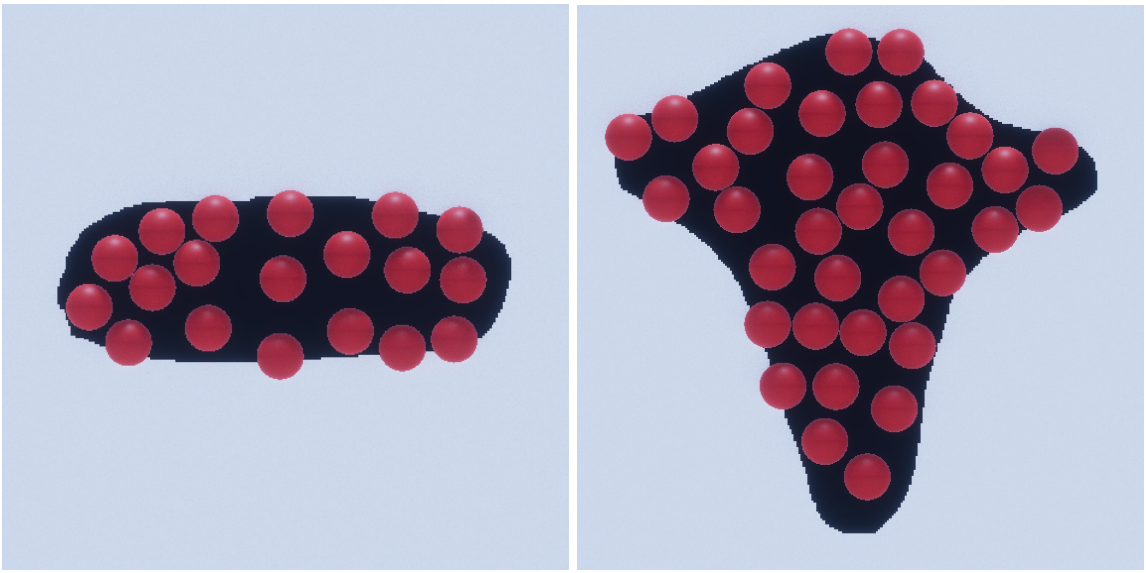


Obrázek 21: **Graf klasifikace typu chodby**. Na tomto obrázku je vidět klasifikace typu chodby v závislosti na strmosti a vzdálenosti od vodní hladiny podzemních vod. V práci je použita aproximace tohoto grafu, který byl vytvořen autory [12] na základě výzkumu.



(a) tunel (b) kaňon (c) klíčová dírka (d) podloží (e) průchod

Obrázek 22: **Masky pro konstruování primitiv**. Masky průřezů jeskyní používané na vytváření primitiv při umisťování disků v algoritmu Poissonovy Disky.



Obrázek 23: **Vizualizace Poissonových disků**. Vizualizace Poissonových disků pomocí koulí umístěných do středů disků pro tvary podloží (nalevo) a klíčová dírka (napravo).

4.2.2 Primitive Sweeping

Každou cestu budeme chtít rozdělit na velkou množinu bodů, které mezi sebou budou mít stejné rozestupy. Spočítáme tedy délku cesty jako: $\sum_{i=1}^{k} ||pozice(a_{i-1}) - pozice(a_i)||$ a určíme vhodný počet bodů tak, aby rozestupy mezi body byly co nejblíže požadovanému rozestupu. Tím získáme relativně rovnoměrně rozmístěné body na cestách a zajistíme, že různá vzdálenost mezi dvěma body nebude mít vliv na finální tvar jeskyně.

Nyní budeme procházet postupně všemi body na všech cestách a budeme v nich, podle masek na *obr.22*, vytvářet odpovídající primitivum. To vždy natočíme tak, aby tečna ke grafu v daném bodě korespondovala se směrem normály primitiva a současně směr nahoru ve scéně se minimálně odchýlil od směru nahoru primitiva. Nakonec provedeme pro střed každého disku v primitiva operaci "Odebírání horniny z terénu"(viz následující krok 4.2.3), což bude upravování hodnot v našem poli v závislosti na množství horniny v okolí. Výsledek Primitive Sweepingu je vidět na *obr.31* v další podkapitole, kde je již vytvořená a otexturovaná trojúhelníková síť.

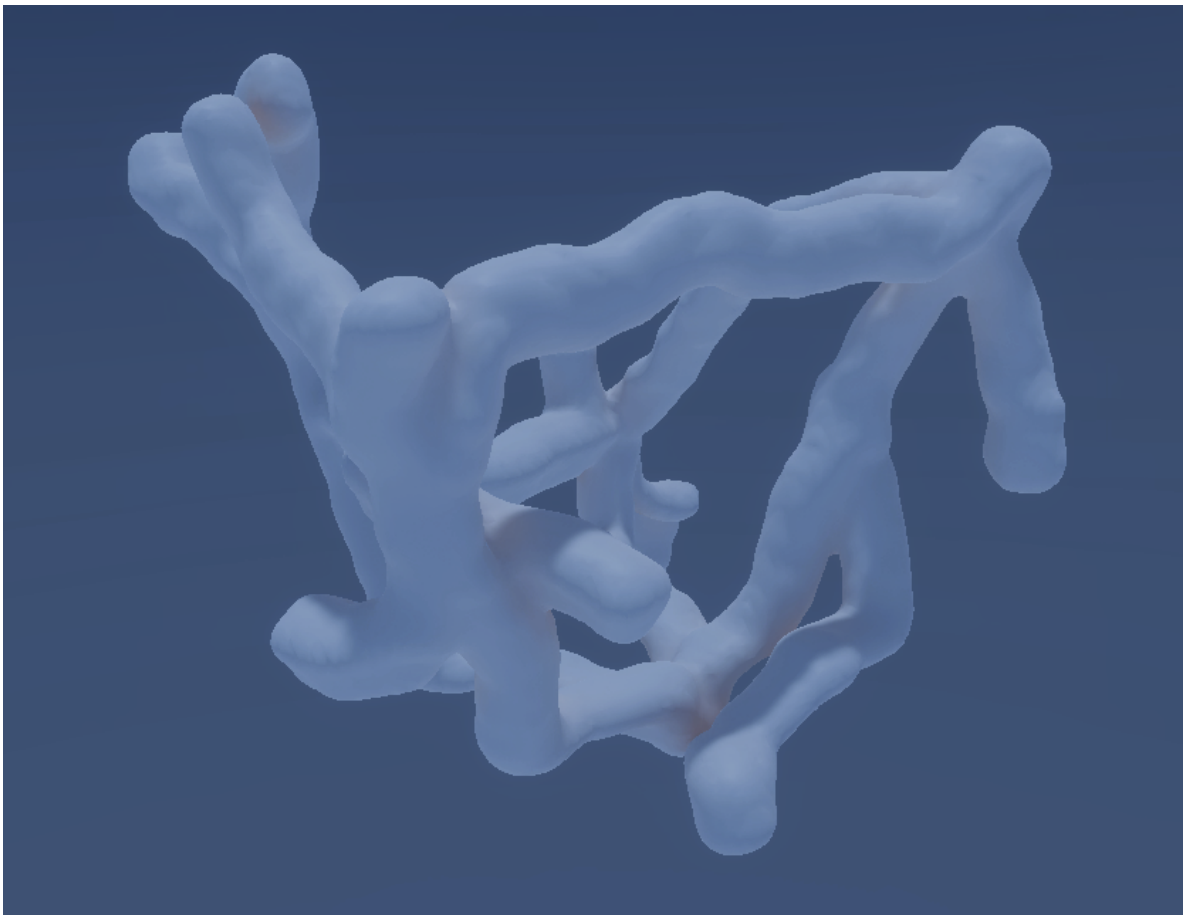
4.2.3 Odebírání horniny z terénu

Vytvoříme 3D pole diskrétně rozmístěných bodů v prostoru, které přesně pokrývá velikost naší oblasti. Každý bod bude mít hodnotu od 0 do 1, přičemž 0 značí, že v okolí tohoto bodu je pouze vzduch, a 1 značí, že v okolí tohoto bodu je pouze hornina. Výchozí hodnota všech bodů bude nastavena na 1.

Odebírání horniny z terénu provedeme úpravou hodnot uvnitř připraveného 3D pole. Pro každý střed disku s_i uvnitř primitiva najdeme všechny body b_j v 3D poli, které se nacházejí v určitém poloměru r a odečteme v nich hodnotu v závislosti na vzdálenosti od středu disku s_i. Tuto odečtenou hodnotu spočítáme jako: $(||s_i - b_j||/r)^p$, kde p je zvolený exponent.

4.2.4 Konstrukce trojúhelníkové sítě

V tuto chvíli máme vytvořenou požadovanou objemovou reprezentaci našeho prostředí ve formě 3D pole s hodnotami od 0 do 1. Ke konstrukci trojúhelníkové sítě použijeme algoritmus Marching Cubes, který je detailně popsaný v práci [9]. Algoritmus Marching Cubes vnímá toto 3D pole jako kostky naskládané na sobě, považuje tedy body v 3D poli za vrcholy kostek. Existuje $2^8 = 256$ různých vzorů pro strukturu kostek, ze kterých vybereme ten správný podle hodnot ve vrcholech. V tuto chvíli algoritmus rozlišuje ve vrcholech jen mezi hodnotami menšími než 0.5 a většími než 0.5. Je to tedy jen osm jedniček a nul (true a false) a vzor najdeme užitím bitové masky. Po nalezení vzoru sestavíme sekci trojúhelníkové sítě, s užitím interpolace po hranách a naplníme vertex buffer (VBO) vrcholy a element buffer (EBO) příslušnými indexy. Zároveň vrcholy vyfiltrujeme pomocí hashmapy tak, aby se neopakovaly ve vertex bufferu a Unity mohlo správně spočítat ve vrcholech normály. Na závěr vytvoříme trojúhelníkovou síť za použití vertex bufferu a index bufferu a Unity dopočítá normály automaticky. Výsledná trojúhelníková síť je na *obr.24*.



Obrázek 24: **Vygenerovaný jeskynní systém bez textur**. Na obrázku je trojúhelníková síť vytvořená algoritmem Marching Cubes za pomoci Primitive Sweeping bez aplikovaných textur.

4.3 Mapování textur a vykreslování

V následujících sekcích se budu věnovat mapování textur, implementaci shaderů, volbě renderovací pipeline a nastavení vykreslování vygenerované trojúhelníkové sítě. Aktuálně máme vygenerovanou trojúhelníková síť jeskynního systému, která se skládá z vrcholů neobsahujících UV (texturovací) souřadnice. To jsou souřadnice, které se v počítačové grafice nejběžněji používají na mapování textur na trojúhelníkovou síť a jsou uloženy v jednotlivých vrcholech. UV souřadnice bychom mohli vytvořit, ale úplně by nám to nepomohlo. Nelze totiž zajistit, aby na sebe textury navazovaly na všech sousedních trojúhelnících zároveň. Nabízí se dvě řešení, triplanární mapování a použití 3D textury.

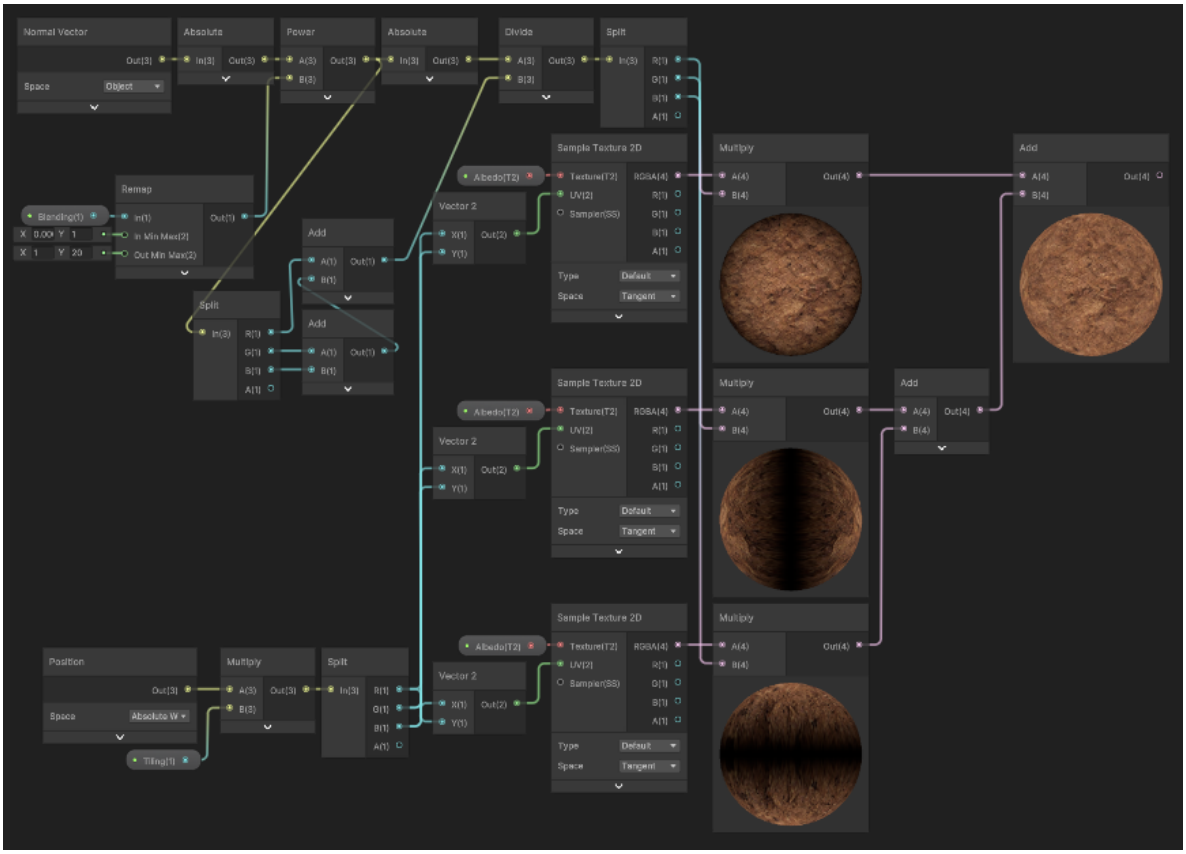
- **Triplanární mapování** - Triplanární mapování umožňuje aplikovat textury na 3D modely s minimální deformací. Jedná se o projekce textur na povrch podél tří různých os (obvykle X, Y a Z) nezávisle na sobě. Projekce jsou následně smíchány v závislosti na normálách povrchu, čímž vytváří plynulou texturu po celém povrchu.
- **3D textury** - Můžeme použít navazující (seamless) 3D texturu a samplovat v ní podle pozice vrcholu v lokální souřadnicové soustavě. Jedná se o elegantní řešení, které nás úplně zbaví veškerých nedostatků, které se objevují v triplanárním mapování. Problémem je zde ale to, že kdyby taková textura měla délku hrany 1k, její velikost by byla $1024^3 * 4\ B \doteq 4\ GB$, což samo o sobě zaplní z poloviny video paměť průměrné grafické karty.

Vhodná 3D textura bohužel není k sehnání a veřejně dostupné modely umělé inteligence ještě nepodporují vytváření takto velkých souborů. Z toho důvodu jsem pro řešení zvolil triplanární mapování. Tato část řešení je v diagramu na *obr.15* znázorněna zelenými šipkami.

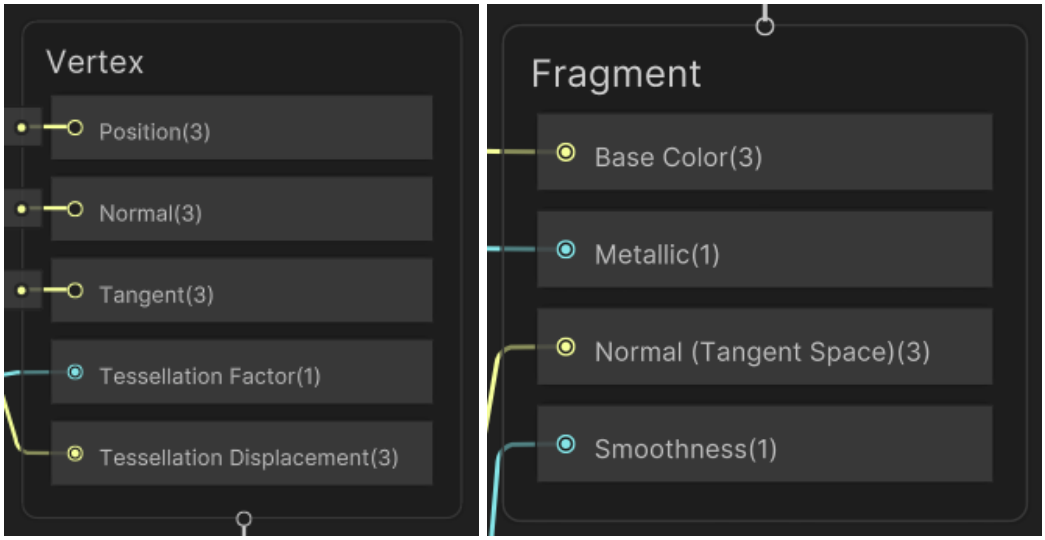
4.3.1 Triplanární mapování

Vybereme vhodný materiál jeskyně, který obsahuje barvu, normálovou mapu, mapu hladkosti (smoothness) a mapu posunutí (displacement). Uvnitř Shader Grafu budeme potom projektovat textury na povrch zvlášť podle os X, Y a Z. Použijeme k tomu absolutní hodnotu normálového vektoru, kterou potom umocníme na hodnotu míchání (blending) a na závěr ho vydělíme jeho délkou, výsledek budu nazývat míchací vektor. Pro samplování v texturách použijeme pozici v lokálním souřadnicovém systému. To jsou tři hodnoty, ze kterých vybereme vždy dvojici a provedeme samplování se zapnutým tilingem (opakování textury). Poté výstupní hodnotu ze sampleru vynásobíme hodnotou, která je uložená v míchacím vektoru na pozici osy souřadnic, kterou jsme nepoužili při samplování a tyto tři hodnoty na závěr sečteme. Implementace celého tohoto shaderu je vyobrazená v Unity Shader Grafu na *obr.25*.

To samé provedeme pro všechny textury, které budeme chtít namapovat a napojíme je na příslušné výstupy v Unity Shader Grafu. Výstupy v Unity Shader Grafu jsou na *obr.26*. Otexturovaný jeskynní systém je pak vidět na *obr.28*.



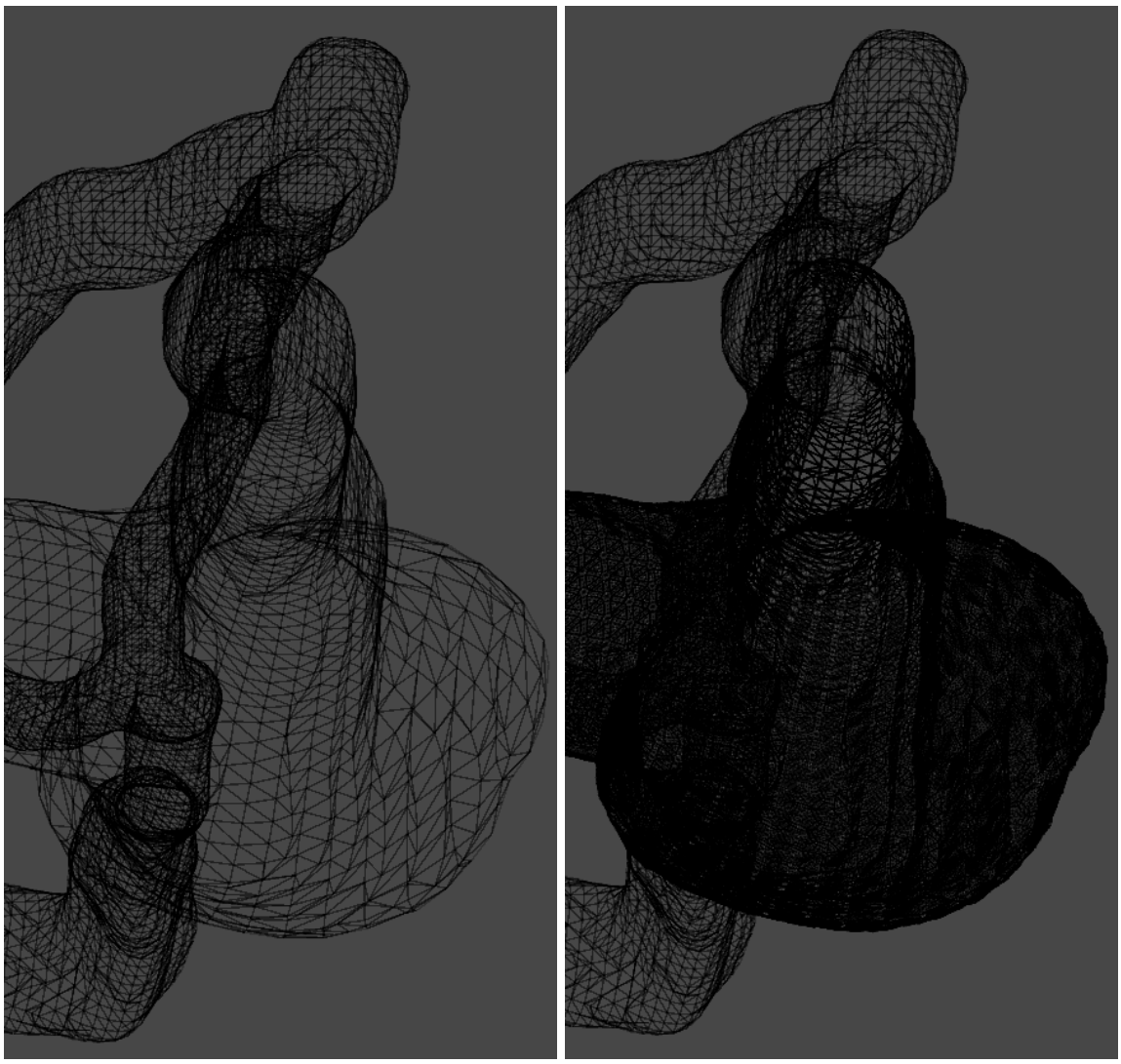
Obrázek 25: **Implementace triplanárního mapování v Unity Shader Grafu**. Na pravé straně je znázorněna projekce textur podél jednotlivých os, a pak jejich následné míchání do jedné výsledné.



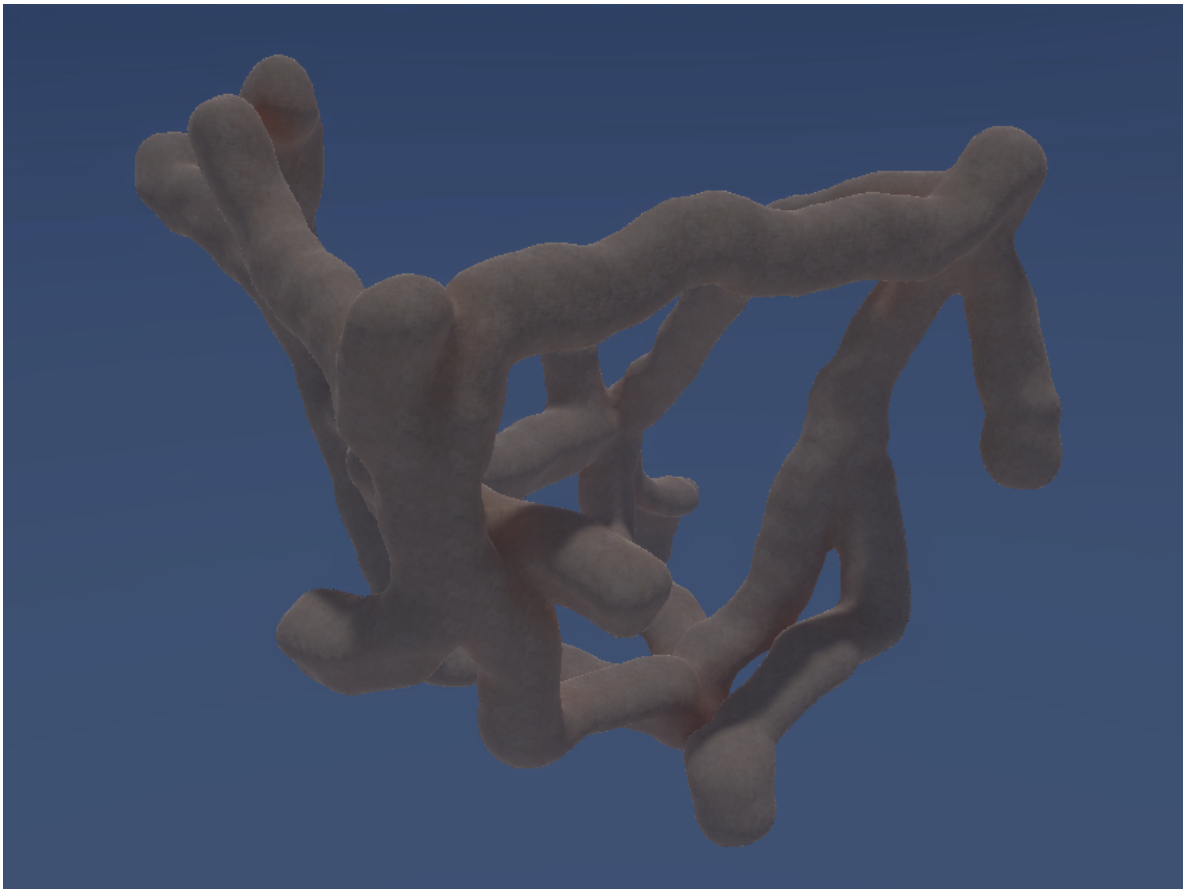
Obrázek 26: **Výstupní parametry z Unity Shader Grafu**. Zde jsou vidět uzly s výstupními parametry v Unity Shader Grafu. Na levém obrázku jsou parametry z vertex shaderu a na pravém z fragment shaderu.

4.3.2 Teselace a posunutí (displacement)

Pro teselaci v Unity Shader Grafu stačí pouze nastavit hodnotu teselačního faktoru (tesselaltion factor). Tím se v Unity aktivuje automatická adaptivní teselace. Pro posunutí použijeme opět triplanární mapování na mapu posunutí a to pak napojíme na teselační posunutí (tessellation displacement). Výstupy související s teselací v Unity Shader Grafu jsou opět na *obr.26* a výsledky automatické adaptivní teselace v Unity jsou na *obr.27*.



Obrázek 27: **Vypnutá a zapnutá teselace**. Na levém obrázku je teselace vypnutá a na pravém je zapnutá. Na pravém obrázku je zřejmé, že se úroveň teseleace mění se vzdáleností kamery od trojúhelníků.



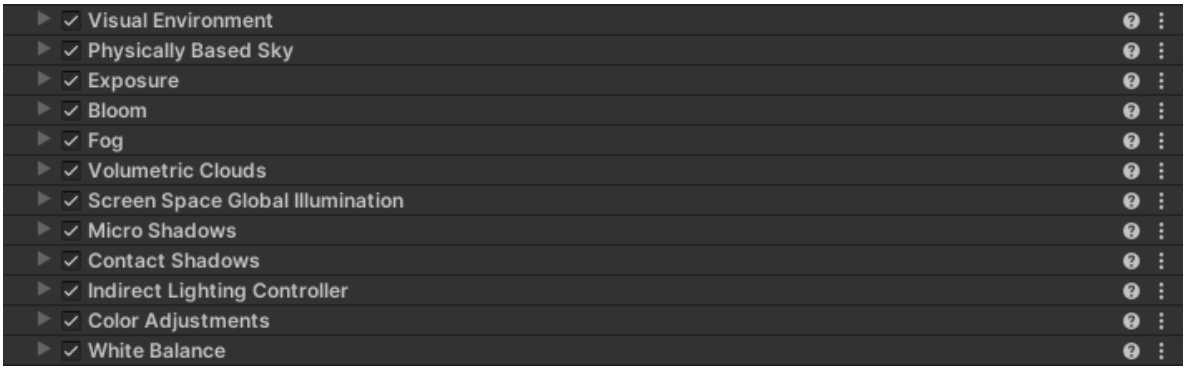
Obrázek 28: **Otexturovaný vygenerovaný jeskynní systém**. Vygenerovaný jeskynní systém, který má na sobě pomocí triplanárního mapování aplikované všechny textury, tedy barvu, normálovou mapu, mapu hladkosti a mapu posunutí.

4.3.3 Nastavení vykreslování v Unity

Uvnitř Unity máme na výběr z několika vykreslovacích pipeline:

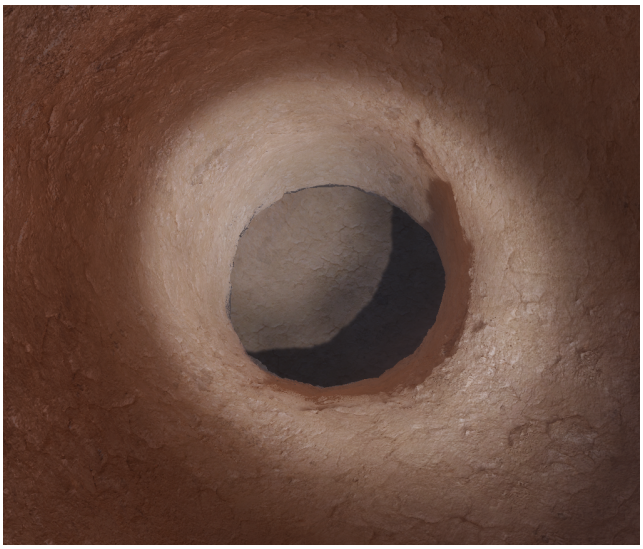
- **Built-in Render Pipeline** (Renderovací pipeline integrovaná v Unity): Základní renderovací pipeline, kterájebude používána v každém novém prázdném projektu. Používá jednoduchý osvětlovací model a techniky pro vykreslování scén.
- **Universal Render Pipeline (URP)**: Dříve známá jako Lightweight Render Pipeline (LWRP). URP je optimalizovaná pro výkon a mobilní zařízení. Nabízí nízkou náročnost na výpočetní výkon a zahrnuje možnosti využití shaderů, které jsou optimalizovány pro moderní grafické karty.
- **High Definition Render Pipeline (HDRP)**: HDRP je zaměřena na vysokou kvalitu vizuálního zpracování a umožňuje vytvářet více realistické a detailní scény. Je vhodná pro projekty, které vyžadují špičkovou grafiku a vizuální efekty. Nabízí rozsáhlé možnosti úprav a nastavení postprocessing efektů, které mohou výrazně ovlivnit vzhled hry nebo aplikace.
- **Custom Render Pipelines** (Vlastní renderovací řetězec): Vedle těchto vestavěných pipeline existuje v Unity také možnost vytvoření vlastní vykreslovací pipeline. To umožňuje vývojářům plnou kontrolu nad vykreslováním scén a shaderů.

Pro co nejrealističtější vzhled jsem zvolil HDRP, která poskytuje opravdu obrovské množství dodatečných nastavení oproti ostatním a to například třeba v Global Volume. Global Volume je v Unity nástroj, který umožňuje aplikovat globální efekty na scénu či herní prostředí, viz *obr.29*. Zde můžeme nastavit různé dodatečné efekty, jako je barevná korekce, mlha, rozostření, ale také složitější efeky, jako je globální nasvícení nebo volumetrické mraky. Tyto efekty mohou zásadně ovlivnit vizuální styl a atmosféru celého prostředí. Pohled do výsledné vygenerované a otexturované jeskyně se všemi efekty je na *obr.30*. Na *obr.31* jsou pak vidět jednotlivé typy průřezů cest.



Obrázek 29: **Použité efekty v Global Volume**. Na obrázku je vidět nastavení Global Volume se spoustou přidaných komponent. Každou lze ještě rozbalit a ladit podle potřeby.

Obrázek 30: **Pohled do vygenerované jeskyně**. Zde je vidět část vygenerovaného jeskynního systému zevnitř nasvícená bodovým světlem, které je záměrně umístěno o pár metrů za kamerou. Uprostřed obrázku se jeskyně rozděluje na tři části. Jedna vede doleva, druhá doprava a třetí kolmo nahoru. Na pravé straně obrázku je patrná teselace a posunutí.



(a) tunel

(b) kaňon

(c) klíčová dírka

(d) podloží

(e) průchod

Obrázek 31: **Vygenerované typy uliček**. Pohled do uliček vygenerovaného jeskynního systému. Jednotlivé tvary odpovídají primitivům z *obr.22*.

4.4 Krápníky

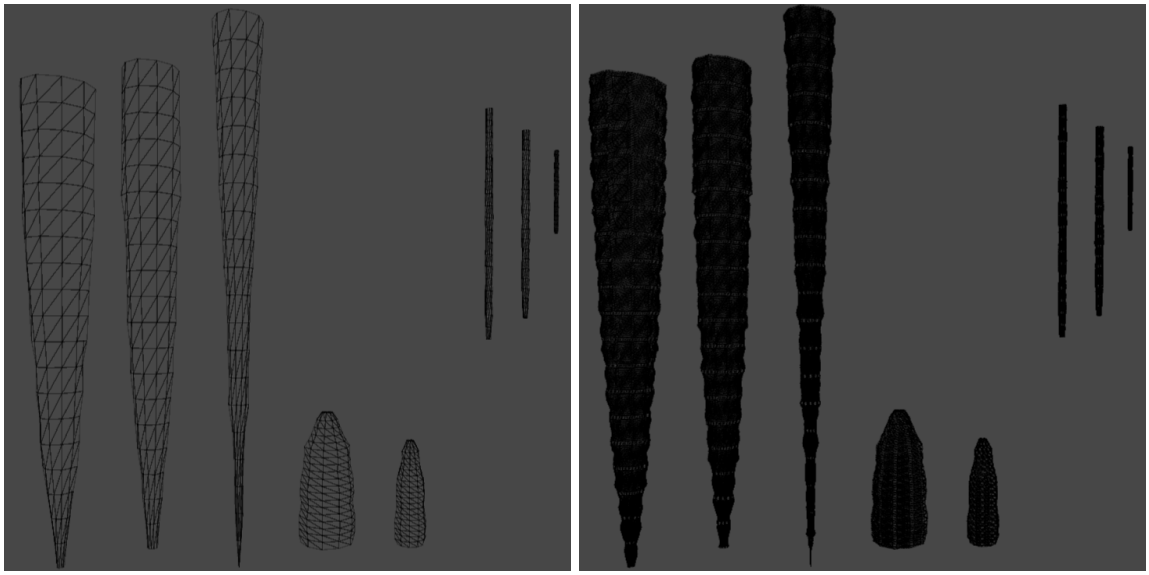
Generování krápníků se bude skládat z nalezení vhodných míst pro uchycení stalaktitů a v závislosti na tom nalezení míst pro umístění stalagmitů. Následně budeme v závislosti na lokálních geologických a fyzikálních podmínkách generovat trojúhelníkovou síť pro jednotlivé krápníky. Tato část řešení je znázorněna žlutými šipkami na *obr.15*.

4.4.1 Stalaktity

Prvním naším cílem je nalézt místa v jeskyni, kde by se stalaktity s největší pravděpodobností vyskytovaly. Dle pozorování vznikají stalaktity na téměř rovných jeskynních stropech, kde je největší šance, že se z vody vytvoří kapky. Nalezení vhodných míst pro stalaktity provedeme z důvodu optimalizace v kroku 8.4 současně s konstrukcí trojúhelníkové sítě. V algoritmu Marching Cubes máme k dispozici všechny trojúhelníky, ze kterých se jeskyně skládá uvnitř smyčky. Je tedy snadné pouze přidat výpočet, který zjistí náklon trojúhelníku a na každém vyhovujícím trojúhelníku poté zvolit body pro uchycení stalaktitu a vložit je do seznamu.

Stalaktity rozdělujeme podle tvaru na dva typy, brčka a kužely. Brčka jsou stalaktity, které vznikaly za většího průtoku vody a rychleji zatímco kužely vznikali za menšího průtoku vody o mnoho pomaleji. Brčka mají tvar úzkého podlouhlého válce a mají tedy přibližně stejný poloměr na opačných koncích. Kužely mají, jak název naznačuje, kuželovitý tvar, který může být lehce prohnutý dovnitř nebo ven a mají na svém povrchu poměrně pravidelné horizontální kruhové výstupy.

Nyní procházíme seznam bodů pro uchycení stalaktitu a s určitou pravděpodobností na každém místě budeme generovat stalaktit. Před generováním jednoho stalaktitu je zvolen jeho typ a v závislosti na tom se pak generuje jeho mřížka. Stavba mřížky probíhá po kruhových vrstvách, které jsou mezi sebou propojeny trojúhelníkovými proužky (triangle strips). Vygenerované krápníky jsou na *obr.37*. Pseudokód generování stalaktitů je uveden na *obr.33*.



Obrázek 32: **Trojúhelníkové sítě vygenerovaných krápníků**. Na levém obrázku jsou výsledné trojúhelníkové sítě krápníků s vypnutou teselací a napravo se zapnutou teselací.

```
1
2  stalaktity_vhodná_místa[]
3
4  For místo In stalaktity_vhodná_místa Do:
5      If Náhoda.číslo < pravděpodobnost_stalaktitu Do:
6          If Náhoda.číslo < pravděpodobnost_kuželu Do:
7              výška_stalaktitu <- průměrná_výška_stalatitu * Náhoda.v_rozmezí
8              poloměr_stalaktitu <- průměrný_poloměr_stalaktitu * Náhoda.v_rozmezí
9              tloušťkový_exponent_stalaktitu <- průměrný_tloušťkový_exponent_stalaktitu * Náhoda.v_rozmezí
10
11             vrcholy[]
12             indexy[]
13             výška_segmentu <- výška_stalaktitu / (počet_bodů_vertikálně - 1)
14
15             For y < počet_bodů_vertikálně Do:
16                 For i < počet_bodů_na_prsteci Do:
17                     úhel <- i / (počet_bodů_na_prsteci * 2 * PI)
18                     úroveň <- 1 - (y/počet_bodů_vertikálně) ^ tloušťkový_exponent_stalaktitu
19                     poloměr <- poloměr_stalaktitu * úroveň
20
21                     vrcholy.přidej(Cos(úhel) * poloměr , -y * výška_segmentu, Sin(úhel) * poloměr)
22
23                 EndFor
24             EndFor
25
26             For y < počet_bodů_vertikálně - 1 Do:
27                 For i < počet_bodů_na_prsteci Do:
28                     indexy.přidej( i+                              y    * počet_bodů_na_prsteci)
29                     indexy.přidej((i+1) % počet_bodů_na_prsteci +  y    * počet_bodů_na_prsteci)
30                     indexy.přidej( i+                             (y+1) * počet_bodů_na_prsteci)
31
32                     indexy.přidej((i+1) % počet_bodů_na_prsteci +  y    * počet_bodů_na_prsteci)
33                     indexy.přidej((i+1) % počet_bodů_na_prsteci + (y+1) * počet_bodů_na_prsteci)
34                     indexy.přidej( i+                             (y+1) * počet_bodů_na_prsteci)
35
36                 EndFor
37             EndFor
38
39             trojúhelníková_síť <- nová_trojúhelníková_síť(místo, vrcholy, indexy)
40         EndIf
41     EndIf
42 EndIf
43
```

Obrázek 33: **Pseudokód generování stalaktitů**. Pseudokód sestavování krápníku kuželovitého tvaru z bodů v prostoru a indexů ukazujících do seznamu těchto bodů. Pole s názvem "stalaktity_vhodná_místa" je vytvořeno při generování trojúhelníkové sítě jeskynního systému. Výsledkem je trojúhelníková síť stalaktitu, kterou je umístěna do scény na vhodné místo.

4.4.2 Stalagmity

Pro generování stalagmitů bohužel nelze použít předchozí metodu, protože stalagmity vždy vznikají přímo pod stalaktity v místech, kam z nich dopadají kapky vody. Z toho důvodu musíme body pro uchycení stalagmitů nalézt jiným způsobem. Jako nejjednodušší se nabízí fyzický raycasting, který je v Unity implementován. Stačí tedy na mřížku jeskyně pouze přidat komponentu Mesh Collider a provést raycasting ze špiček stalaktitu vertikálně dolů.

Kdy se stalagmit začne na daném místě tvořit, je také v úzké souvislosti s náklonem plochy dopadu kapky. Stalagmity nabývají svého tvaru v závislosti na rychlosti dopadu kapky, tedy na výšce, ze které kapky padají. Čím více se kapka rozptyluje po dopadu, tím více stalagmit poroste do šířky. Z toho důvodu většinou bývají stalagmit širší a menší než jejich stalaktity a mají zaoblenější špičku.

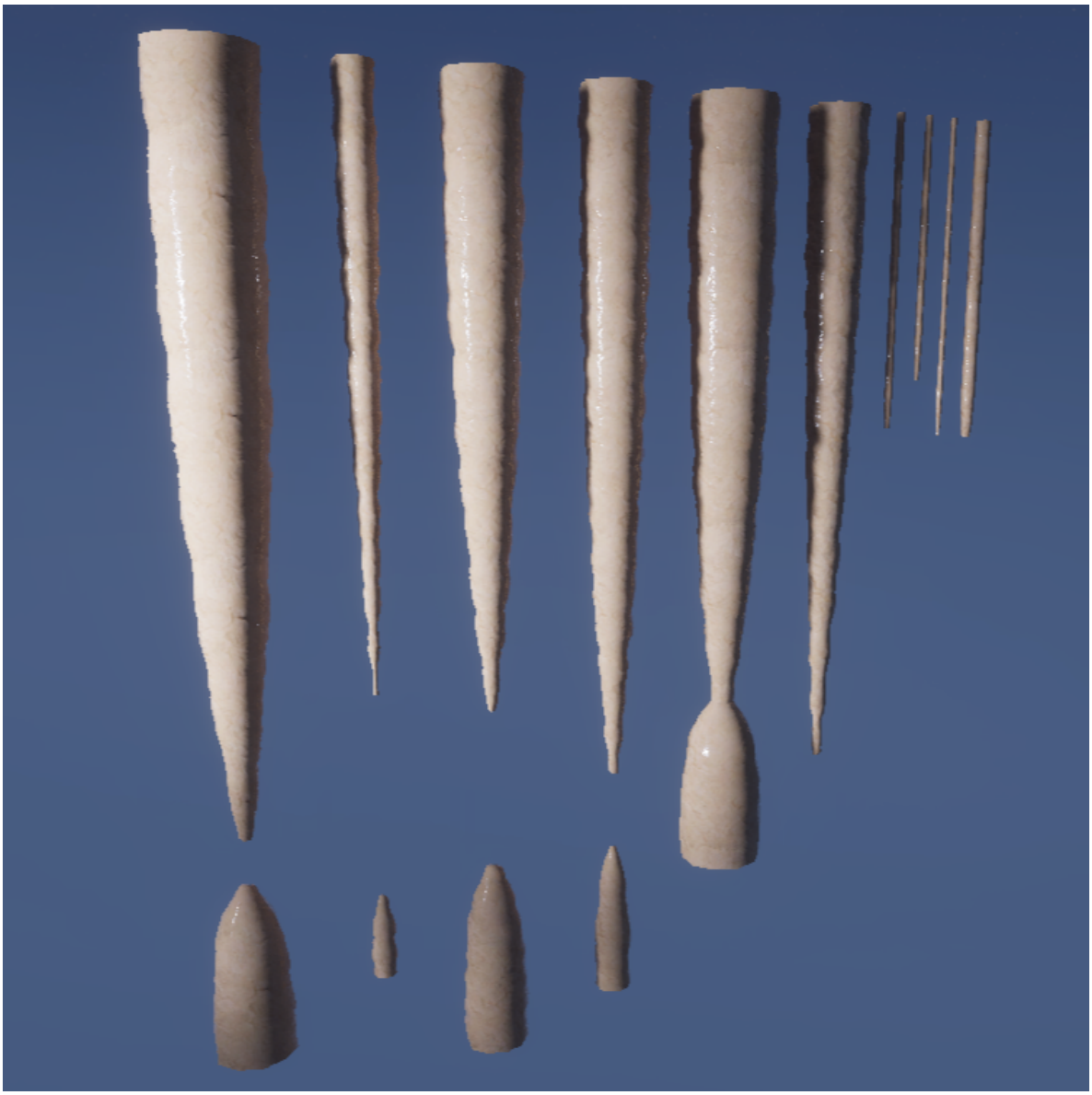
My budeme stalagmit umisťovat pod stalaktity s určitou pravděpodobností v případě, že náklon místa dopadu je dostatečně rovný. Samotný stalagmit vygenerujeme přibližně ve tvaru funkce $-|x|^k$ s koeficientem $k > 2$, který zvolíme v závislosti na rychlosti dopadajících kapek.

4.4.3 Stalagnáty

Stalagnáty vznikají spojením stalaktitu a stalagmitu, což v našem případě již může při generování nastat, jako vedlejší ale přirozený efekt. Tedy pro regulování počtu a tvaru vzniklých stalagnátů postačí pouze regulovat parametry generování stalaktitů a stalagmitů.

4.4.4 Vykreslování krápníků

Pro vykreslování krápníků použijeme opět triplanární mapování s vhodnými texturami. Použijeme tedy stejný shader jako pro vykreslení jeskyně a modifikujeme ho. Pro vytvoření realisticky vypadajících boulí na krapnících použijeme absolutní hodnotu ze sinusoidy s náhodnou amplitudou a frekvencí, kterou ještě vynásobíme s perlinovým šumem. Výsledek je na *obr.34*.



Obrázek 34: **Vygenerované krápníky v řadě**. Jednotlivě vygenerované stalaktity a stalagmity, které jsou umísťovány do jeskynních prostorů. Stalagmity jsou vždy menší než jejich stalaktity. Pod krápníky typu brčka se stalagmity negenerují. Každý z krápníků má náhodnou frekvenci boulí, přičemž u krápníků typu kuželů jsou značně větší.

4.5 Jeskynní jezera

V této sekci se budeme věnovat generování jeskynních jezer s pomocí 3D pole, použitého algoritmem Marching Cubes. Toto pole budeme nazývat objemové pole a budeme na něm provádět průchod grafem. Tímto průchodem grafu zjistíme, které sekce jeskynního systému se propojí, pro zvolenou výšku vodní hladiny, v dané sekci. Nakonec provedeme konstrukci trojúhelníkové sítě algoritmem Marching Squares. Tato sekce je znázorněna na *obr.15* modrými šipkami.

4.5.1 Hledání lokálních minim a výšky vodní hladiny

Prvním krokem je hledání vhodných míst v jeskynním systému, kde by se voda mohla udržet. Jedná se o prohlubně, které budeme nazývat lokální minima. Pro jejich nalezení využijeme objemové pole. Vytvoříme si 3D pole o stejné velikosti, do kterého si budeme ukládat identifikátor skupiny. Identifikátor skupiny bude nezáporné číslo, které budou sdílet všechny body, do kterých se dostaneme z lokálního minima pouze pohybem do stran a nahoru. To znamená že bude existovat jedinečný identifikátor pro každé nalezené lokální minimum. Pseudokód hledání lokálních minim je na *obr.35*.

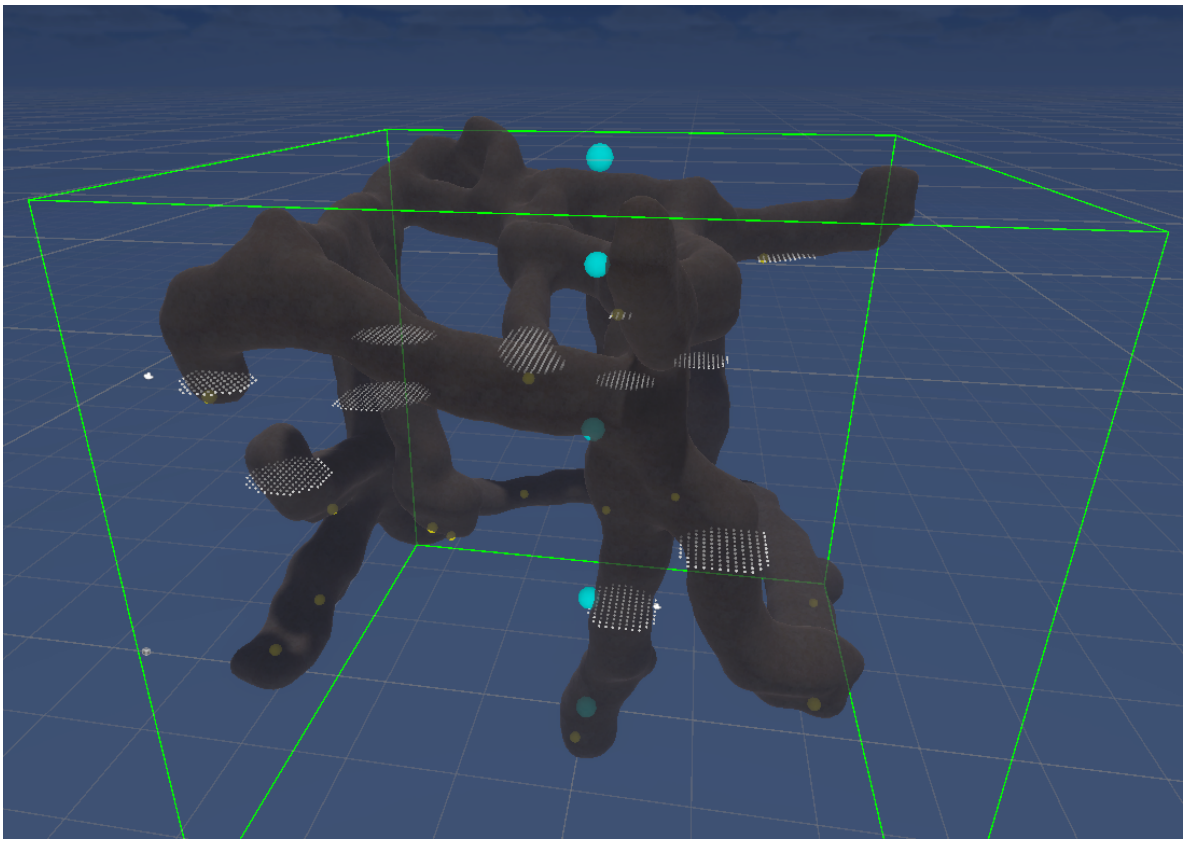
```
1
2  objemové_pole[][][]
3  skupiny[]
4
5  For y,x,z In objemove_pole Do:
6      If objemové_pole[x,y,z] není uvnitř horniny a nemá id Do:
7          nové_id
8          fronta_teď
9          fronta_další
10
11         fronta_teď.enqueue(objemové_pole[x,y,z])
12
13         While fronta_teď není prázdná Do:
14             While fronta_teď není prázdná Do:
15                 prvek <- fronta_teď.dequeue()
16                 prvek_nad <- objemové_pole[prvek.x, prvek.y + 1, prvek.z]
17
18                 If prvek_nad není uvnitř horniny a ještě nemá id Do:
19                     prvek_nad.id <- nové_id
20                     fronta_další.enqueue(prvek_nad)
21                 EndIf
22
23                 For soused In všichni sousedi přímí a diagonální Do:
24                     If soused není uvnitř horniny a ještě nemá id:
25                         soused.id <- nové_id
26                         fronta_teď.enqueue(soused)
27
28                     EndIf
29                 EndFor
30             Endwhile
31             fronta_teď <- fronta_další
32
33         Endwhile
34     EndIf
35 EndFor
36
```

Obrázek 35: **Pseudokód hledání lokálních minim a přiřazování id skupin**. Pseudokód implementace hledání lokálních minim a přiřazování identifikátorů skupin. Jednotlivým skupinám je poté přiřazena výška hladin v závislosti na sousedních skupinách.

Nyní budeme procházet všechny body v objemovém poli odspoda. Pro každý bod, který má hodnotu menší než 0,5 (je mimo horninu) a ještě nemá přidělený žádný identifikátor skupiny, víme, že ve stejném patře se nutně nachází lokální minimum. Vytvoříme nový identifikátor skupiny a provedeme prohledávání do šířky (breadth first search - BFS). Při prohledávání do šířky přidáváme do fronty všechny body, ležící nad a vedle prohledávaného bodu, jejichž hodnota je menší než 0,5. Všem bodům, které navštívíme, přiřadíme nově vytvořený identifikátor skupiny. Při běhu algoritmu si pro každou skupinu ukládáme ještě výšku lokálního minima a výšku nejvyššího dosaženého bodu (lokálního maxima). Nakonec běhu zvolíme v závislosti na výšce lokálního maxima a výšce lokálního minima náhodně výšku vody v dané sekci tak, aby nebyla nižší než výška podzemních vod.

V případě že při prohledávání do šířky narazíme směrem nahoru na bod s určeným identifikátorem skupiny, dohledáme si podle něj hodnotu výšky hladiny v dané sekci. V případě že je hodnota vyšší než výška bodu, na který jsme narazili, přiřadíme aktuální skupině stejnou výšku hladiny. V opačném případě pokračujeme normálně dále beze změny.

Nakonec skončíme se sekcemi označenými identifikátory, které mají přidělenou výšku vodní hladiny a zbývá jen už v jeskynních prostorech v daných místech vytvořit trojúhelníkovou síť vodní hladiny a vykreslit ji. Lokální minima a výška vodní hladiny jednotlivých sekcí jsou znázorněny na *obr.36*.



Obrázek 36: **Znázornění výšky vodní hladiny v sekcích**. Výška vodních hladin v jednotlivých sekcí, které jsou označené identifikátory, je znázorněna bílými koulemi. Žluté koule v jeskynním systému znázorňují nalezená lokální minima.

4.5.2 Konstrukce trojúhelníkové sítě vodní hladiny

Máme zvolené výšky vodních hladin ve všech sekcích jeskyně oddělených identifikátory. Nyní projdeme všechny vrstvy v objemovém poli, které obsahují nějakou vodní hladinu, a budeme zde chtít vytvořit novou trojúhelníkovou síť, která bude reprezentovat vodní hladinu. K tomu využijeme Marching Squares, což je jednodušší verze algoritmu Marching Cubes modifikovaná pro 2D prostor. Postup je zde analogický jako v kroku 4.2.4, jen procházíme vždy objemové pole po vrstvách a výstupem jsou všechny vodní hladiny jako jedna trojúhelníková síť. Ukázka výsledné trojúhelníkové sítě je na *obr.38*.

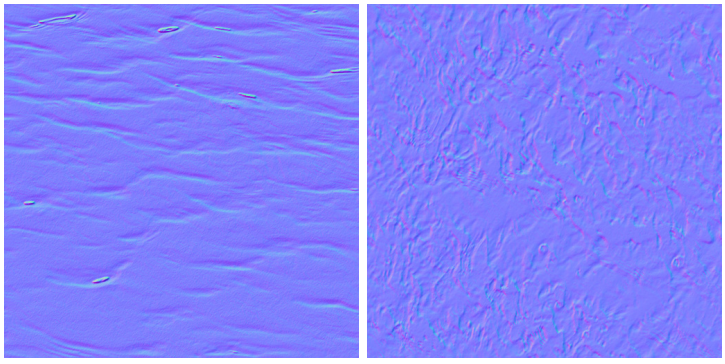
4.5.3 Vykreslování vodní hladiny

Vykreslením realistické vodní hladiny se zabývá, na rozdíl od jeskynních systémů, obrovské množství prací, které ho dovádějí téměř k dokonalosti. Existují články s vysokou úrovní realističnosti vykreslování vodních hladin, které byly napsané před více než dvěma dekádami. Mezi ně se řadí například práce [7], kde autoři shrnují metody pro vykreslování vodních hladin, optimalizaci formou LOD (úroveň detailu) a navrhují novou metodu optimalizace formou projekce vodní hladiny z pohledu kamery.

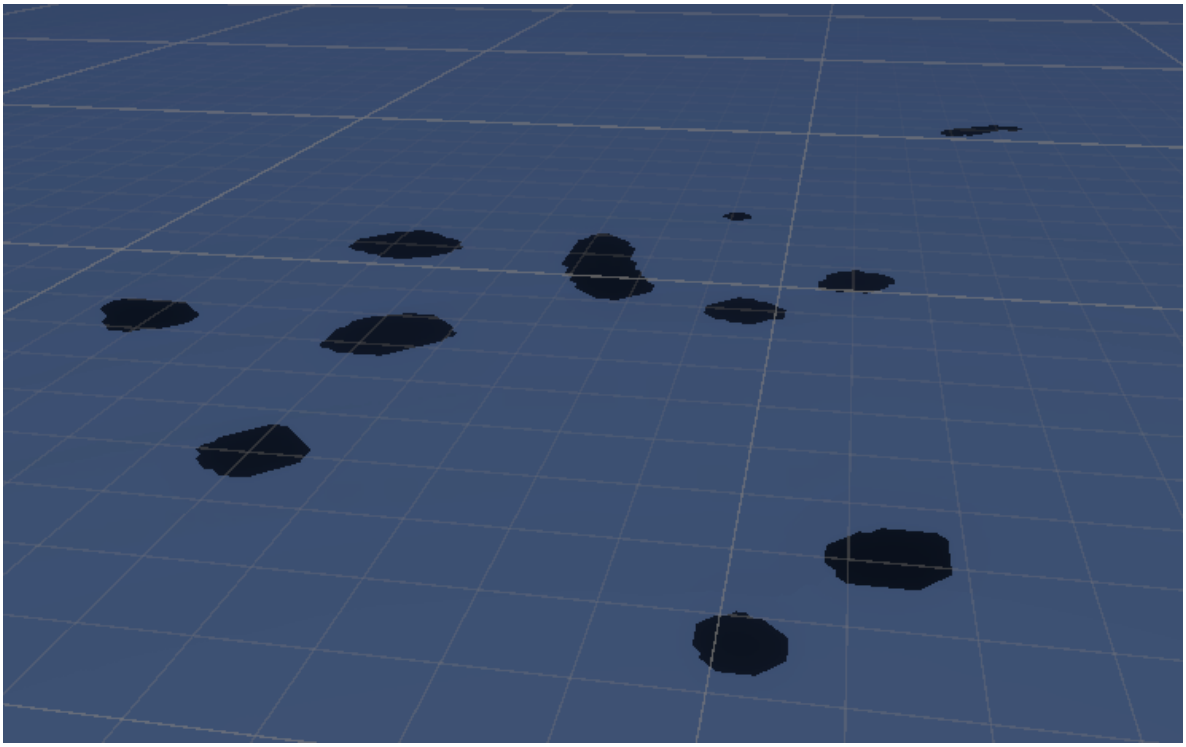
Pro tuto práci jsem se rozhodl implementovat zjednodušenou metoda, která zanedbáváme odrazy vodní hladiny. Toto rozhodnutí padlo z důvodu komplikací a neuspokojivých výsledků popsaných v následující sekci 4.6. První věcí, kterou potřebujeme je způsob, jak namapovat textury na povrch vodní hladiny. To lze vyřešit triviálně za použiti mapování v závislosti na poloze ve scéně, tedy UV souřadnice nepotřebujeme.

Shader pro vykreslování vodní hladiny se skládá ze dvou hlavních poznatků, první je fakt, že průhlednost povrchu vody se mění s úhlem pohledu a druhá je, že čím hlubší voda v daném místě je, tím méně světla z ní pronikne zpět k pozorovateli. Hloubku vody jsme schopni zjistit pomocí hloubkového bufferu (depth buffer), do kterého vyrenderujeme scénu bez vodní hladiny a poté při finálním renderování pro každý fragment odečteme hloubku od příslušné hodnoty v depth bufferu vynásobenou vzdálenou rovinou (z far plane). Úhel pohledu zjistíme pomocí skalárního součinu normály vodní hladiny a pohledového vektoru (view vector). Tyto dvě hodnoty spolu potom vynásobíme a dle výsledku budeme interpolovat mezi dvěma barvami a určovat průhlednost vodní hladiny v daném místě. Pro hlubší vodu zvolíme tmavě modrou barvu a pro mělkou světle modrou.

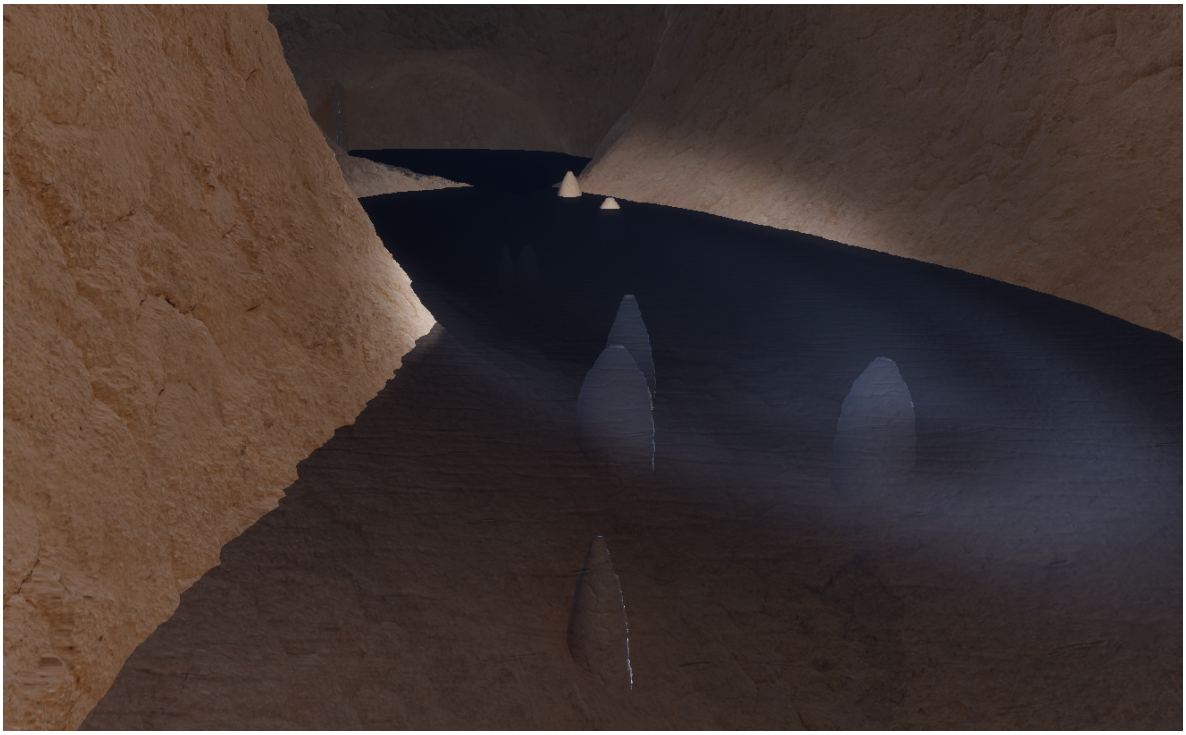
Pro simulování průvanu, nebo větru v jeskynním systému použijeme dvě normálové mapy, kterými budeme pohybovat v opačném směru. Tuto finální normálu využijeme i v předchozích kalkulacích. Výsledná vykreslená vodní hladina je na *obr.39*.



Obrázek 37: **Normálové mapy**. Ukázka dvou normálových map použitých pro vykreslování vodní hladiny.



Obrázek 38: **Trojúhelníková síť jeskynních jezer**. Trojúhelníková síť vytvořených jeskynních jezer po vrstvách pomocí algoritmu Marching Squares. Jednotlivá jezera neleží v jedné rovině.



Obrázek 39: **Výsledná vodní hladina**. Vodní hladina vykreslená s ohledem na úhel pohledu a hloubku vody v daném místě.

4.6 Odraz na vodní hladině

Pro vykreslení odrazu na vodní hladině jsem zvažoval tři způsoby. Konkrétně se jedná o reflection probes, vykreslování zpod hladiny a poslední možnost, která se zdá nejvíce logická, použití v zásadě nové implementace vykreslování vodní hladiny poskytnutou přímo Unity. Ani jeden ze způsobů jsem ale nakonec v implementaci neponechal buď z důvodu neuspokojivých výsledků nebo v souvislosti s Unity Shader Grafem, jehož využívání se ukazuje v tomto ohledu jako nešťastné.

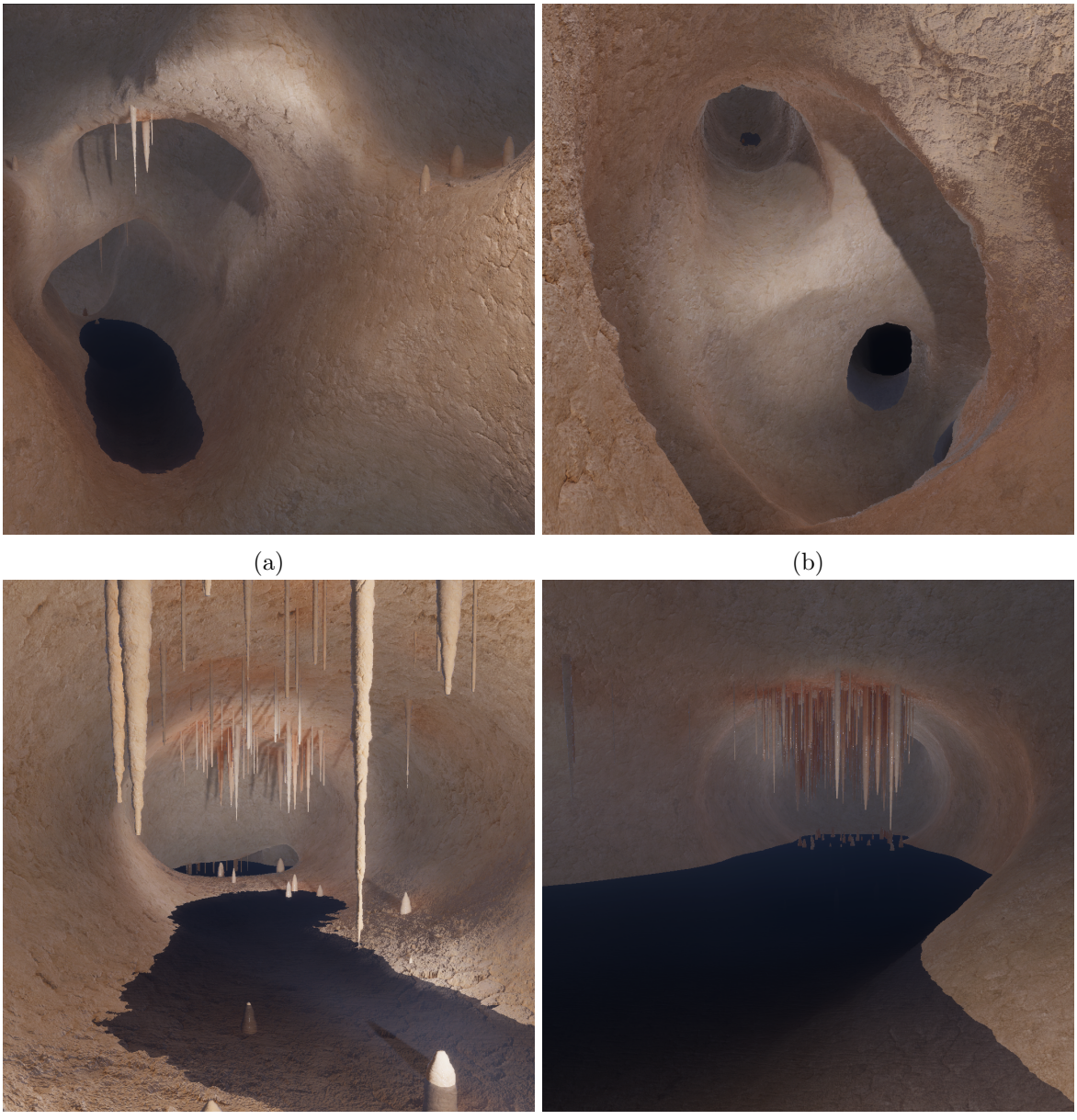
- **Reflection probes** - První způsob je použití reflection probes, které jsou v Unity naimplementovány. Jedná se o body ve scéně, ve kterých jsou vytvořeny, vykreslením okolního prostředí (evironment mapping), kostkové mapy (cube maps). Mezi nimi se pak přepíná v závislosti na vzdálenosti od reflektivního objektu. Je nutné podotknout, že tento způsob je pouze aproximace reálných odrazů a při použití této metody budou na odrazech chyby, které je nutné akceptovat. Tento způsob je ale bohužel nešťastný z důvodu krápníků. Ve chvíli co je vodní hladina obklopena krápníky, musí reflection probe být umístěna mezi ně. Tím ale často zastíní velkou část zorného pole, což úplně zkazí jakékoliv dojmy z odrazu. Přes vyvinuté úsilí tento způsob tedy uznávám za nevhodný.

- **Vykrelování zpod hladiny** - Druhý způsob je umístění kamery zrcadlově pod každou hladinu a vykreslení scény v několika krocích. Pro realizování tohoto způsobu je třeba pro každou úroveň hladiny, která je viditelná na obrazovce, vykreslit scénu, skrze zrcadlově převrácenou kameru, podél této hladiny, do render bufferu. Zároveň je třeba se zbavit všech fragmentů, které se nacházejí pod výškou dané hladiny. Toto může být velice náročné na vykreslování v případě, že je na obrazovce vidět velké množství hladin v různých výškách. Za cenu náročného vykreslování ale tento způsob zajistí dokonale přesné odrazy. Bohužel i tento přístup má problém. Unity Shader Graph, který je použit na všechny shadery v této práci, nepodporuje vyhazování (discard) fragmentů dle podmínky a také přímo nepodporuje vykreslování do renderovacích bufferů. Toto řešení by tedy vyžadovalo všechny shadery přepsat do jazyka HLSL, a poté scénu renderovat na několikrát pomocí skriptů a uniformů. Vzhledem k tomu, že se práce od začátku ubírá směrem Shader Graphu, považuji toto řešení také za nevhodné.

- **Vodní hladina poskytnuta Unity** - Poslední způsob a také ten, který by byl nejjednodušší implementačně a vypadal nejrealističtěji, je použití vykreslování vodní hladiny přímo od vývojářů Unity. Uvnitř Unity lze do scény vložit herní objekt s názvem "Water Surface"(vodní hladina), ve kterém si lze zvolit z možností: moře/jezero, bazén, řeka. Výsledkem je téměř bezchybná realistická vodní hladina. Tento způsob se bohužel opět neobejde bez problému. Tím je tentokrát to, že tento shader vodní hladiny je vázaný na předdefinovanou trojúhelníkovou síť. Aktuálně tedy nejde tento shader aplikovat na vygenerovanou trojúhelníkovou síť vodní hladiny.

5 Výsledky

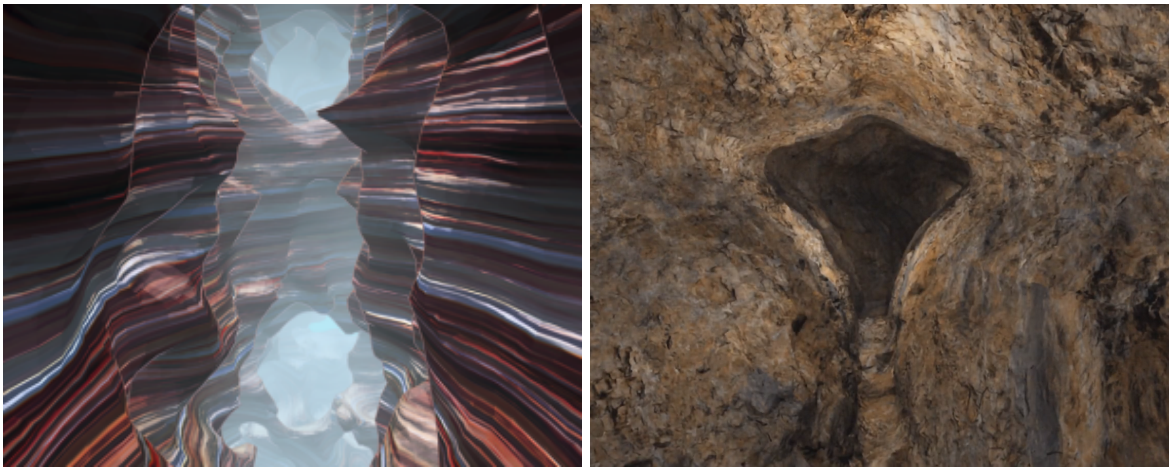
Po propojení všech uvedených metod máme naimplementovaný nástroj, který je schopen vygenerovat jeskynní systém v závislosti na okolním prostředí a na zadaných parametrech uživatelem. V této kapitole se budu věnovat evaluaci výsledků generátoru a jeho rychlosti a výkonnosti. Ukázka rozmanitosti vygenerovaných jeskynních systémů je na *obr.40* a ukázka výsledku generátorů z jiných prací je na *obr.41*.



(a) (b)

(c) (d)

Obrázek 40: **Nástrojem vygenerované jeskynní systémy s různě nastavenými parametry**. Na obrázcích (a) a (b) je generátor nastavený tak, aby generoval velké množství cest, zatímco na (c) a (d) pouze malé množství. Na obrázku (b) mají primitiva menší poloměr, na obrázku (d) velký poloměr a na obrázcích (a) a (c) normální. Na obrázku (d) je navýšeno množství generovaných krápníků a je zvětšena jejich velikost.



(a) Zdroj: [10] (b) Zdroj: [12]

Obrázek 41: **Vygenerované jeskynní systémy z jiných prací**. Na obrázku (a) je rozsáhlý podzemní kaňon s různě obarvenými vrstvami a na obrázku (b) jeskyně ve tvarů klíčové dírky.

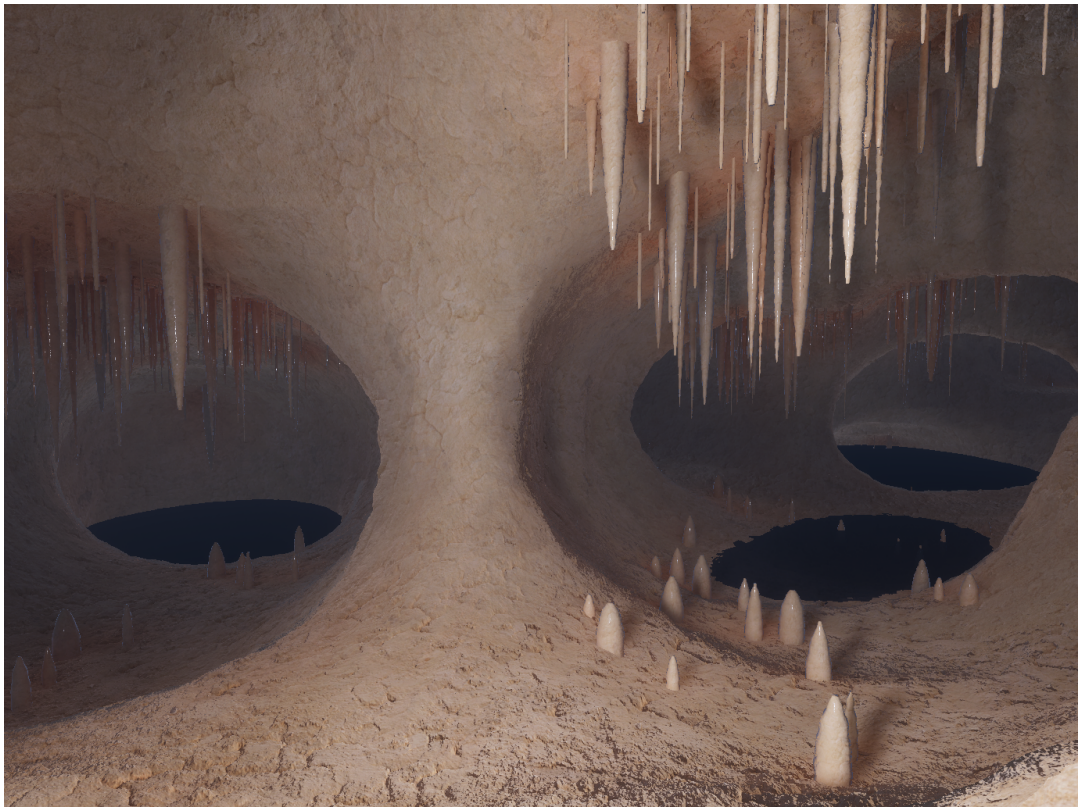
5.1 Porovnání výsledku s realitou

V této sekci budou porovnány výsledně vygenerované jeskynní systémy s jeskynními systémy z reálného světa. Konkrétně jsem zvolil tři pohledy na různé prvky v jeskynním systému. Jedná se o krápníky, jeskynní jezera a pohled na jeskyni jako celek.

5.1.1 Porovnání kompletních jeskynních systémů

Porovnání je zachyceno na *obr.42*. Na obou těchto obrázcích je pohled na vybranou část jeskynního systému krasových jeskyní. Na vygenerovaném obrázku je vidět prostor, který se v daném místě utvořil křížením cest, stejně tak jako na reálném obrázku je vidět něco na způsob křižovatky uvnitř jeskyně. Na obou obrázcích většinu prostoru zabírají stěny jeskyně, pokryté srovnatelnými prasklinami a barvami. Jsou na nich také ve vzdálenosti rozmístěny stalaktity a stalagmity, které se utvořily na rovných plochách.

Největším rozdílem mezi těmito dvěma obrázky je tvar stěn jeskyně, kde v reálném obrázku je spousta ostrých a nepravidelných hran, zatímco ve vygenerovaném jsou pouze hrany tupé a pravidelné, to je bohužel limitace algoritmu Marching Cubes. Na reálném obrázku jsou vidět formace krápníků uchycené na krápníkových základnách, což generátor nepodporuje.



(a)

(b) Autor: Tracy A., taburns25.com

Obrázek 42: **Vygenerovaný a reálný jeskynní systém**. Na horním obrázku je sekce jeskynního systému vygenerovaná implementovaným algoritmem a na dolním obrázku je sekce Koněpruských jeskyň.

5.1.2 Porovnání krápníků v jeskyni

Porovnání vygenerovaných a reálných krápníků je uvedeno na *obr.43*. Na obou obrázcích se nachází velké množství stalaktitů, které nabývají různých tvarů. V obou případech jsou vidět stalaktity kuželovité a brčkovité, přičemž brčkovité v obou obrázcích převládají. Na reálných krápnících je pozorovatelné zvlnění, které lze pozorovat i na vygenerovaných. Další shodou je zbarvení vápence, které je do jisté míry v obou obrázcích stejné.

Hlavní vadou vygenerovaných krápníků je uniformita, která na levém obrázku jasně převažuje. Vygenerované krápníky mají pravidelný tvar a na rozdíl od reálného obrázku nejsou tak nepravidelné. Na reálném obrázku jsou také vidět krápníky, ze kterých trčí značné výrůstky do stran. Největším rozdílem je ale samotná základna krápníků, tedy místo uchycení krápníků, která je masivní a značně rozvinutá v reálném obrázku. Řešení pro generování základny krápníků generátor neobsahuje.

(a) (b) Převzato z: npkrka.hr

Obrázek 43: **Vygenerované a reálné krápníky**. Na levém obrázku jsou stalaktity vygenerované implementovaným algoritmem a na pravém obrázku jsou stalaktity v reálné jeskyni.

5.1.3 Porovnání jeskynních jezer

Porovnání vygenerovaného a reálného jeskynního jezera je znázorněno na *obr.44*. Na těchto dvou obrázcích se nabízí pohled na jeskynní jezera uvnitř krasových jeskynní. V obou obrázcích jsou patrné zvýšení průhlednosti vodní hladiny se snižující se hloubkou, stejně jako pokles průhlednosti se zvyšujícím se úhlem pohledu. Na levém obrázku lze skrze průzračnou vodu vidět základnu blízkých stalagmitů, na rozdíl od vzdálenějších, které jsou maskované díky absolutnímu odrazu. Stejně také na pravém obrázku lze zahlédnout blízké objekty pod vodní hladinou, zatímco ve větší vzdálenosti převládá odraz. Za shodné lze označit slabé vlnění na povrchu vodních hladin, které je lehce vidět díky nasvícení.

Největším rozdílem je samotný odraz vodní hladiny, který implementovaný generátor jeskynních systémů neřeší a značně vylepšuje realističnost pravého obrázku. Na vygenerovaném obrázku je použita tmavší barevná paleta než na reálném, což ale lze vysvětlit rozpuštěnými látkami a nasvícením. Větší rozdíl je velké světlání modré barvy v nejmělčích oblastech na reálném obrázku, které není na vygenerované obrázku k zahlédnutí.

(a) (b) Autor: V. G. Custode, dreamstime.com

Obrázek 44: **Vygenerované a reálné jeskynní jezero**. Na levém obrázku je vodní hladina jeskynního jezera vygenerovaná implementovaným algoritmem a na pravém obrázku je jeskynní jezero v reálné jeskyni.

5.2 Výkon

V této části je provedeno testování doby, za kterou je schopen vytvořený nástroj vygenerovat jednotlivé části jeskynního systému. Pro získání přehledu jsem provedl velké množství testů, ze kterých jsem vybral ty nejzajímavější výsledky. Testování bylo prováděno na stolním počítači s těmito komponenty: CPU - Intel Core i5-12400F, GPU - Nvidia RTX 3060 Ti (8GB VRAM), RAM - 16GB DDR4. Každý z testů má nějakým způsobem unikátně nastavený parametr a má přidělené číslo. Nastavitelných parametrů je příliš velké množství, pro zkoušení všech kombinací, proto budu měnit jen některé vybrané a pouze jeden naráz. Tímto způsobem získáme aspoň nějakou představu, které parametry mají největší vliv na výkon. Finálně vybrané parametry jsou: *Size*, *TerrainEditsPerUnit*, *PrimitiveRadius* a *MarchingCubesScale*. Klíčových bodů zůstává ve všech testech stejný počet (šest) a jsou ponechány ve stejných místech.

Test č.	*Size*	*TerrainEditsPerUnit*	*PrimitiveRadius*	*MarchingCubesScale*
1.	50x50x50	5	2,25	0,5
2.	75x75x75	5	2,25	0,5
3.	100x100x100	5	2,25	0,5
4.	50x50x50	5	2,25	1
5.	50x50x50	5	3	0,5
6.	50x50x50	10	2,25	0,5

Tabulka 1: **Nastavené parametry pro jednotlivé testy**.

Test č.	Koule	Cesty	Síť	Krápníky	Jezera	Celkem
1.	219ms	364ms	1,18s	77ms	21ms	1,86s
2.	508ms	1,41s	1,80s	76ms	42ms	3,83s
3.	1,53s	2,31s	2,79s	97ms	95ms	6,82s
4.	211ms	348ms	263ms	21ms	4ms	847ms
5.	180ms	350ms	4,10s	100ms	36ms	4,68s
6.	193ms	435ms	2,31s	76ms	24ms	3,04s

Tabulka 2: **Doby generování částí jeskyně**. Jednotlivé hodnoty byly naměřeny časovacím kódem, který jsem přidal do nástroje.

Popis jednotlivých testů:

1. Toto nastavení parametrů považuji za ideální, proto se kolem něj pohybují všechny testy.

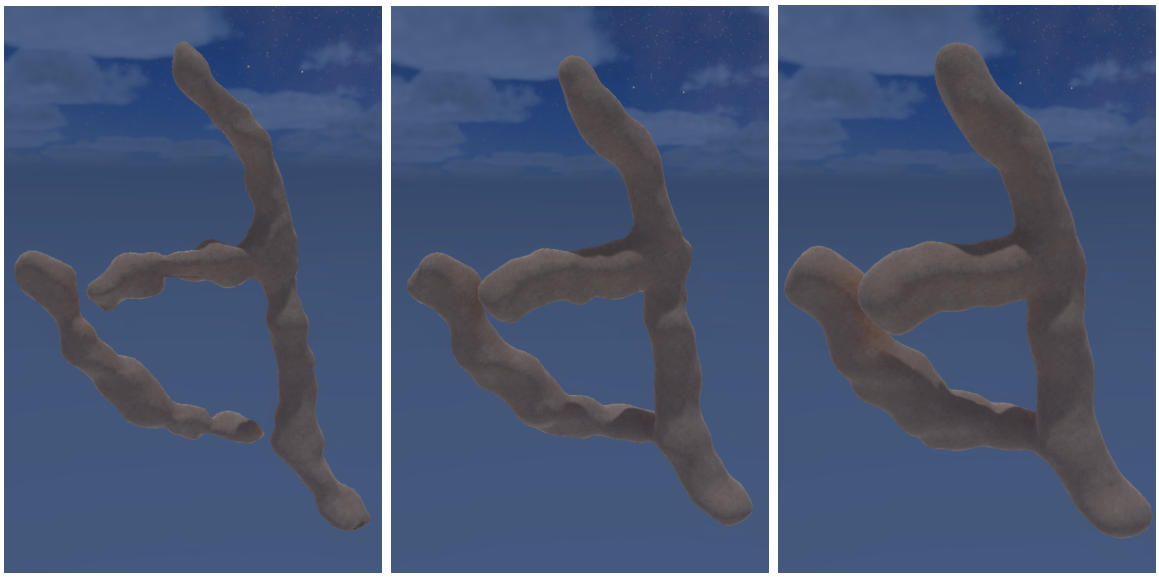
2. Region *Size* je zvětšeno na $75x75x75$, což zvětší prohledávaný prostor cca 3,4krát oproti ideálnímu nastavení. Tomu odpovídá delší rozmísťování koulí a náročnější průchod větším grafem při hledání cest. Generování trojúhelníkové sítě není o tolik složitější, protože výsledná trojúhelníková síť je stejná.

3. Zde je *Size* zvětšeno na $100x100x100$, což je přesně 8krát větší než v prvním testu. Výsledkem je tedy úměrně delší rozmísťování koulí a cest. Při generování trojúhelníkové sítě je nutné projít velké množství prázdného prostoru, proto délka roste.

4. V tomto testu je nastaveno *MarchingCubesScale* na 1, tedy volumetrická reprezentace obsahuje osminu bodů oproti ideálnímu nastavení. Z toho důvodu je zde vidět značný pokles doby konstrukce trojúhelníkové sítě, generování krápníků a generování jezer, které jsou všechny lineárně závislé na množství bodů ve volumetrické reprezentaci.

5. *PrimitiveRadius* je zvětšen z 2,25 na 3, tedy změna pouze o 25%, ale vliv na výkon je velký. Doba generování trojúhelníkové sítě je přibližně čtyřnásobná ve srovnání s prvním testem. Růst této doby je způsoben složitějším rozmísťováním disků v každém kroku, kterých je finálně více. To způsobuje nutnost provádět více operací odebírání horniny z terénu, která má poměrně velký vliv na výkon.

6. *TerrainEditsPerUnit* je zvětšeno na dvojnásobek, což zdvojnásobí množství operací prováděných při Primitive Sweepingu. Z dat vyplývá že doba generování trojúhelníkové sítě je, ve srovnání s prvním testem, také přibližně dvojnásobná a potvrzuje lineární závislost na parametru. Vliv tohoto parametru je znázorněn na *obr.45*.



(a) *TerrainEditsPerUnit* nastaveno na: 1

(b) *TerrainEditsPerUnit* nastaveno na: 2

(c) *TerrainEditsPerUnit* nastaveno na: 5

Obrázek 45: **Ukázka vlivu počtu editací na jednotku délky**. Na těchto obrázcích je měněn pouze parametr *TerrainEditsPerUnit*. Z výsledků je zřejmé, že hodnota 1 znázorněná na obrázku (a) je nedostačující a způsobí nespojitost jeskynního systému podél cest. Hodnota 2 obrázku (b) je přibližné minimum pro zajištění spojitosti jeskynního systému. Hodnota 5 na obrázku (c) je dle mého názoru optimum a nezanechává téměř žádné artefakty.

6 Závěr

Tato práce by se dala rozdělit na tři hlavní části. V první jsem se věnoval typům jeskynních systémů a jejich vzniku v reálném světě. Ve druhé části jsem popisoval to, jak různé aplikace a hry využívaly a využívají jeskynní systémy a souhrnu metod, které se pro jejich generování využívají v praxi. Třetí část se pak věnuje implementaci nástroje v Unity Engine pro procedurální generování jeskynních systémů, krápníků a jezer. Pro shrnutí:

1. První část práce popisuje pět různých typů jeskynních systémů, které jsou rozděleny podle prostředí a způsobů, kterými vznikají. Konkrétně shrnuje mořské jeskyně, lávové tunely, ledovcové jeskyně, eolské jeskyně a nejdůležitější krasové jeskyně, které rozebírá do hloubky včetně procesů, které přispívají k jejich vzniku.

2. Druhá část práce se věnuje jeskynním systémům a jejich generování v počítačové grafice a v počítačových hrách. Uvedena byla hra Colossal Cave Adventure, která odstartovala zájem o jeskyně v počítačových hrách, pak postupně hry Rogue, Minecraft a Deep Rock Galactic. Poté rozebírá hlavní metody procedurálního generování jeskynních systémů v počítačové grafice, mezi ně byly zařazeny L-systémy, fraktální algoritmy, fyzikální simulace a metody užívající algoritmů pro nejlevnější cestu.

3. V třetí, hlavní a poslední části, práce je popsán celý postup při implementaci nástroje pro generování jeskynních systémů, krápníků a jeskynních jezer.

   (a) Pro generování jeskynních systémů byl využit algoritmus pro nejlevnější cestu. První krokem bylo vytvoření kostry jeskyně. Nejdříve byl prostor diskrétně rozdělen pomocí algoritmu Poissonových Koulí a připraven pro průchod grafem. Následně byly vygenerovány nejlevnější cesty algoritmem A*, který s heuristikou graf využil k nalezení nejlevnějších. Heuristika zahrnovala uraženou vzdálenost, cenu vrstev a trhlin. Cesty byly poté prořezány a rozvětveny pro podpoření větší nepravidelnosti. Druhým krokem bylo obalení kostry trojúhelníkovou sítí. Nejprve bylo třeba vytvořit objemovou reprezentaci prostoru, čehož bylo dosaženo za pomoci algoritmu Primitvie Sweeping, který v objemovém poli vytvořil uličky předdefinovaných tvarů v závislosti na prostředí. Na závěr byl použit algoritmus Marching Cubes pro vytvoření samotné trojúhelníkové sítě z objemového pole. Posledním krokem při vytváření jeskyně bylo namapování textur a nastavení vykreslování v Unity. K namapování textur, mezi kterými byla barva, normálová mapa, mapa hladkosti a mapa posunutí, bylo využito triplanární mapování. K vykreslování v Unity byla použita High Definition Render Pipeline a Global Volume.

   (b) Při generování krápníků byla využita data z objemového pole pro nalezení míst pro uchycení stalaktitů a fyzikální raycasting pro nalezení míst pro uchycení stalagmitů. Trojúhelníkové sítě jednotlivých krápníků byly poté vygenerovány jednotlivě podle přiděleného typu. Pro stalaktity byly rozlišeny dva základní typy brčka a kužely, pro stalagmity pouze jeden a to rozpláclý kužel.

   (c) Pro generování jeskynních jezer bylo opět použito objemové pole, skrze které bylo provedeno postupné prohledávání do šířky odspoda. Pomocí prohledávání do šířky byly pak sdruženy jednotlivé sekce jeskyní, které mají své společné lokální minimum. Následně byla zvolena pro sekce výška hladiny tak, aby neporušovala zákony hydrostatiky. Nakonec byla vygenerována trojúhelníková síť hladiny a vykreslena s ohledem na hloubku a úhel pohledu.

Vzniklý nástroj lze použít pro rychlé vygenerování jeskynního systému, dle představ uživatele. Na uživateli je také ponechána volba, ve formě nastavitelných parametrů, jestli preferuje realističnost, nebo aby jeskyně měla nějaký jiný, méně pravděpodobný tvar.

6.1 Možná vylepšení

Existuje mnoho možností, jak tento nástroj rozšířit nebo vylepšit. Do nástroje je možné například přidat zóny, kterými by se dal modifikovat algoritmus Poissonových koulí tak, aby v nich koule negeneroval (nebo generoval jinak). Tím bychom mohli docílit například generování jeskynní pod nějakým údolím, nebo napojení generátoru na jiné nástroje, které třeba generují terén. S napojením na nástroj, který generuje terén na zemském povrchu, souvisí také generování závrtů. Jeskynní systém by se dal na povrch napojit a závrt generovat v závislosti na typu terénu, umístění a naklopení terénu. To také nástroj aktuálně nepodporuje.

Další věcí jsou odrazy vodní hladiny. Pro jejich realizaci by bylo nejspíše nutné přepsat všechny shadery do HLSL, což je stínovací jazyk, který Unity používá. Dále by bylo nejvhodnější použít metodu vykreslování zpod vodní hladiny a scénu tím pádem vykreslovat na vícekrát. To by vyžadovalo jisté optimalizace, jako například test viditelnosti každé vodní hladiny a změnu velikosti vykreslované textury v závislosti na vzdálenosti od kamery.

Problém s vodní hladinou a také jiné problémy by se daly vyřešit ray tracingem (sledováním paprsků). Ray tracing ale aktuálně ještě není vhodný pro hry a většinu aplikací bežících v reálném čase, které pro něj nejsou zásadně optimalizované. To se s velkou jistotou v nadcházejících letech změní a rozšíření grafických karet, co podporují výkonný hardwarový ray tracing vzroste.

Z hlediska optimalizace by se dal nástroj určitě také vyladit. Aktuálně je použito rozdělení na vlákna jen v některých sekcích v procesu generování. Pro podporu více vláken například při konstrukci trojúhelníkové sítě by se dalo objemové pole rozdělit na několik sekcí, nad kterými by jednotlivá vlákna zkonstruovala více samostatných trojúhelníkových sítí. Ty by se nakonec spojily do jedné finální trojúhelníkové sítě jeskynního systému.

Reference

[1] Caves. *National Cave and Karst Research Institute*, 2021. https://nckri.org/caves/types/.

[2] Lava tubes. *National Park Hawai'i*, 3 2021. https://www.nps.gov/havo/learn/nature/lava-tubes.htm.

[3] Izabella Antoniuk and Przemyslaw Rokita. Procedural generation of underground systems with terrain features using schematic maps and l-systems. *Challenges of Modern Technology*, 7:8–15, 09 2016.

[4] Juncheng Cui. Procedural cave generation. Master's thesis, Faculty of Informatics, University of Wollongong, 2011.

[5] Augustin Gouy, Pauline Collon, Vincent Bailly-Comte, Eric Galin, Christophe Antoine, Benoît Thebault, and Philippe Landrein. Karstnsim: A graph-based method for 3d geologically-driven simulation of karst networks. *Journal of Hydrology*, 632:130878, 2024.

[6] Dennis Jerz. Cave gave game: Subterranean space as videogame place. *Electronic Book Review*, 10 2015. https://electronicbookreview.com/essay/cave-gave-game-subterranean-space-as-videogame-place/.

[7] Claes Johanson. Real-time water rendering. 3 2004.

[8] Ares Lagae and Philip Dutré. Poisson sphere distributions. In L. Kobbelt, T. Kuhlen, T. Aach, and R. Westermann, editors, *Vision, Modeling, and Visualization 2006*, pages 373–379, Berlin, November 2006. Akademische Verlagsgesellschaft Aka GmbH.

[9] William E. Lorensen and Harvey E. Cline. *Marching cubes: a high resolution 3D surface construction algorithm*, page 347–353. Association for Computing Machinery, New York, NY, USA, 1998.

[10] Benjamin Mark, Tudor Berechet, Tobias Mahlmann, and Julian Togelius. Procedural generation of 3d caves for games on the gpu. 06 2015.

[11] B. Neidhold, M. Wacker, and Oliver Deussen. Interactive physically based fluid and erosion simulation. 2005.

[12] Axel Paris, Adrien Peytavie, Eric Guérin, Pauline Collon, and Eric Galin. Synthesizing geologically coherent cave networks. *Computer Graphics Forum*, 40, 10 2021.

[13] Audra Philippe, Quinifp Yves, and Rochette Pierre. The genesis of the tennengebirge karst and caves (salzburg, austria). *Journal of Cave and Karst Studies*, 3:155–164, 01 2002.

[14] Daniel Tortelli and Marcelo Walter. Modeling and rendering the growth of speleothems in real-time. *GRAPP 2009 - Proceedings of the 4th International Conference on Computer Graphics Theory and Applications*, pages 27–35, 01 2009.

[15] William White. Chemistry and karst. *Acta Carsologica*, 44, 02 2016.

[16] Yumeng Yan. Research on the a star algorithm for finding shortest path. *Highlights in Science, Engineering and Technology*, 46:154–161, 04 2023.

A Návod k použití

Celá implementace byla provedena v Unity Engine ve verzi editoru 2022.3.13f1. Doporučuji projekt spustit v jakékoliv verzi v rozmezí od uvedené po poslední vydanou verzi editoru 2022. Nedoporučuji spouštět projekt v novějších verzích editoru, tedy 2023 a více. Projekt může fungovat bez problému, ale nemusí.

První věc, kterou je po spuštění editoru dobré udělat, je zapnout Gizmos, které jsou vytvořené pro zlepšení přehledu při ovládání generátoru. Po vygenerování požadované jeskyně doporučuji Gizmos zase vypnout, aby nekazily pohled na jeskyni. Nejdůležitější pro ovládání samotného generátoru je herní objekt s názvem "CaveGenerator". V něm se nastavuje většina parametrů, které mají vliv na generování jeskynního systému. Nastavitelné parametry v "CaveGenerator"jsou vyobrazené na *obr.46* a jednotlivé jsou níže popsány. Navíc je mezi nimi sekce **Debug Menu**, kterou lze použít k vizualizaci vygenerovaných Poissonových koulí a k zobrazení a schování různých trojúhelníkových sítí a objektů ve scéně.

Nejdůležitějšími objekty ve scéně, které slouží k ovládání vzniku jeskyní jsou klíčové body (potomci herního objektu "KeyPoints"), vrstvy (potomci herního objektu "Horizons") a trhliny (potomci herního objekty "Fractures"). Klíčové body je možné libovolně posouvat, duplikovat a mazat. Vrstvy lze také libovolně duplikovat a mazat, pouze je nutné, aby herní objekty "TopHorizon"a "BottomHorizon"zůstaly ve scéně. Ceny jednotlivých vrstev se nastavují uvnitř inspektoru po kliknutí na určitou vrstvu. U trhlin lze pouze upravovat v inspektoru jednotlivé normály směrů trhlin. Je samozřejmě nutné, aby jednotlivé normály nebyly příliš podobné.

Pro vygenerování jeskynního systému slouží šest tlačítek, které jsou vidět dole na *obr.46*. Tlačítko Generate Everything vygeneruje celý jeskynní systém s krápníky a jezery najednou. Má stejnou funkci, jako zmáčknutí všech pěti tlačítek nad ním v řadě po sobě. Těchto pět tlačítek v řadě provádí následující:

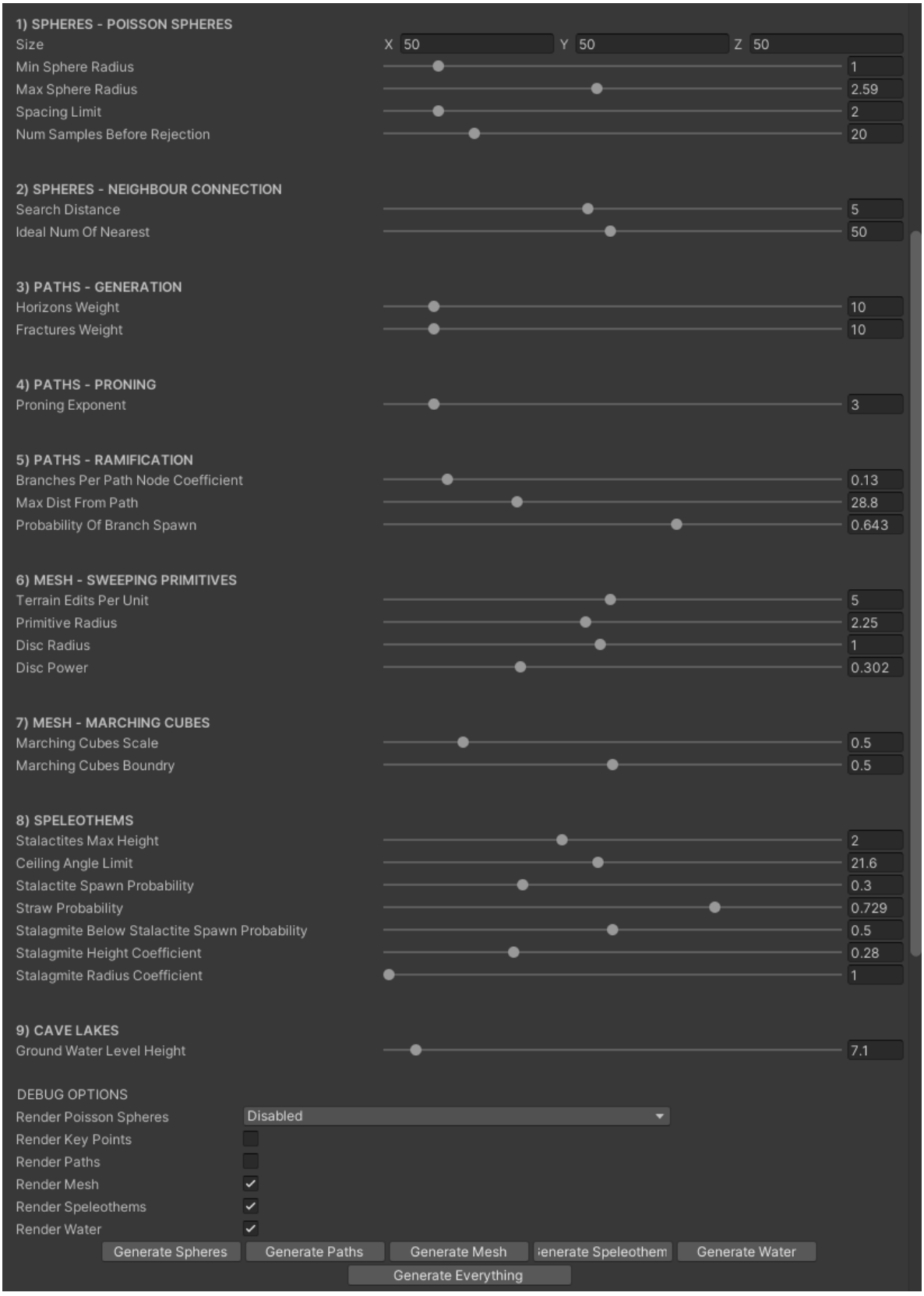
1. **Generate Spheres** - vytvoří nad prostorem nové Poissonovy Koule.
2. **Generate Paths** - najde cesty mezi klíčovými body a provede prořezaní s rozvětvením.
3. **Generate Mesh** - provede Primitive Sweeping a vytvoří trojúhelníkovou síť pomocí Marching Cubes.
4. **Generate Speleothems** - vygeneruje krápníky.
5. **Generate Water** - vygeneruje jeskynní jezera.

Jednotlivá tlačítka lze mačkat samostatně, ale při zmáčknutí jednoho z prvních tří uvedených je nutné, pro správný výsledek, zmáčknout ještě postupně všechny tlačítka směrem doprava.

1. **Spheres - Poisson Spheres**
   - *Size* - velikost regionu, kde se bude jeskynní systém generovat v metrech.
   - *MinSphereRadius* - nejmenší možný poloměr Poissonovy koule.
   - *MaxSphereRadius* - největší možný poloměr Poissonovy koule.
   - *SpacingLimit* - největší možná mezera mezi novou koulí a generující koulí.
   - $NumSamplesBeforeRejection$ - udává po kolika pokusech generátor vzdá snahu o nalezení místa pro novou kouli a přejde na další generující kouli.
2. **Spheres - Neighbour Connection**
   - *SearchDistance* - nejvyšší vzdálenost ve které jsou koule propojovány.

- *IdealNumOfNearest* - požadovaný, ale ne nutně splněný, počet propojených nejbližších koulí v grafu.

3. **Paths - Generation**
   - *HorizonsWeight* - koeficient, který určuje, jak moc se cesty drží horizontů s nízkou cenou.
   - *FracturesWeight* - koeficient, který určuje, jak moc cesty následují směry trhlin.
4. **Paths - Proning**
   - *ProningExponent* - prořezávací exponent, ovládá množství prořezávání.
5. **Paths - Ramification**
   - *BranchesPerPathNodeCoefficient* - počet míst na jedné hraně na cestě, kde existuje šance tvorby větve.
   - *MaxDistFromPath* - maximální vzdálenost konce větve od bodu na cestě, ze které je tvořena.
   - *ProbabilityOfBranchSpawn* - pravděpodobnost vytvoření nové větve.
6. **Mesh - Sweeping Primitives**
   - *TerrainEditsPerUnit* - kolik bude provedeno operací odebírání horniny z terénu na jednotku délky.
   - *PrimitiveRadius* - poloměr primitia, které je používáno při Primitive Sweepingu.
   - *DiscRadius* - poloměr Poissonových disků při tvoření primitiva.
   - *DiscPower* - kolik terénu bude jeden disk odebírat.
7. **Mesh - Marching Cubes**
   - *MarchingCubesScale* - udává, jak velké trojúhleníky vytvořené Marching Cubes budou, menší hodnota znamená větší pole a menší trojúhelníky.
   - *MarchingCubesBoundry* - udává, jakou hodnotu v objemovém poli bude Marching cubes považovat za horninu.
8. **Speleothems**
   - *StalactitesMaxHeight* - nejvyšší možná výška stalaktitu.
   - *CeilingAnglelimit* - nejvyšší možný úhel stropu pro tvorbu stalaktitu.
   - *StalactiteSpawnProbability* - pravděpodobnost, vytvoření stalaktitu na vhodném místě.
   - *StrawProbability* - pravděpodobnost, že stalaktit bude typu brčko.
   - *StalagmiteBelowStalactiteSpawnProbability* - pravděpodobnost, že se pod stalaktitem vytvoří stalagmit.
   - *StalagmiteHeightCoefficient* - udává velikost stalagmitu v závislosti na velikosti stalaktitu.
   - *StalagmiteRadiusCoefficient* - udává poloměr stalagmitu v závislosti na poloměru stalaktitu.
9. **Cave Lakes**
   - *GroundWaterLevel* - definuje výšku, nad kterou se budou generovat jeskynní jezera.



Obrázek 46: **Nastavitelné parametry generátoru**. Nastavitelné parametry pro procedurální generování jeskynního systému v Unity inspektoru.