FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	S10W: Vizualizace plně procedurálního dynamického terénu se zaměřením na generování biomů
Student:	Tomáš Bubeníček
Vedoucí:	Ing. Radek Richtr
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

S10W je projekt realizace 3D světa jako aplikace pro systém SAGE. Úkolem této práce je vizualizace plně procedurálního dynamického terénu se zaměřením na vykreslování rozsáhlých dynamických scén s přirozeným prolínáním oblastí a to na úrovni jednoduché sady objektů (stromy, keře, tráva, ...) i biomů (nížina, oceán, vrchovina, hory, lesy, ...). V práci využijte výsledky předchozích prací projektu.

1) Proveďte analýzu současného stavu API SAGE2 a možnosti zobrazení procedurálních scén na zařízení SAGE

2) Analyzujte problematiku procedurálního zobrazení přírodních scén s přihlédnutím k přechozím pracem projektu.

3) Definujte na základě požadavků zadavatele soubory pravidel (modely) prostředí, podle kterých se bude scéna rozvíjet na úrovní biomů i objektů

4) Navrhnete model řídicí generování biomů, vizualizace biomů a prototyp aplikace.

5) Implementujte prototyp vizualizace scény dle zvolených modelů.

6) Prototyp podrobte vhodným testům.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 12. února 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

S10W: Vizualizace plně procedurálního dynamického terénu se zaměřením na generování biomů

Tomáš Bubeníček

Vedoucí práce: Ing. Radek Richtr

14. května 2018

Poděkování

Tímto chci poděkovat mému vedoucímu práce Ing. Radku Richtrovi za odborné vedení, rady a čas, který mi při řešení bakalářské práce věnoval. Chci také poděkovat mým rodičům a sourozencům za jejich stálou a vytrvalou podporu. Zvláštní dík patří mé matce Kateřině a mé sestře Anežce za kontrolu pravopisných chyb a překlepů. V neposlední řadě chci také poděkovat participantům testu použitelnosti za jejich čas a názory.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

Praha dne 14. května 2018 ……

České vysoké učení technické v Praze

Fakulta informačních technologií

Odkaz na tuto práci

BUBENÍČEK, Tomáš. *S10W: Vizualizace plně procedurálního dynamického terénu se zaměřením na generování biomů*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018. Dostupný také z WWW: ⟨`https://gitlab.fit.cvut.cz/bubentom/SAGE2-ProcViewer`⟩.

Abstrakt

Tato práce si klade za cíl prozkoumat možnosti vizualizace procedurálně generovaného terénu na velkoplošných zobrazovacích zařízeních a zaměřuje se při tom na různorodost terénu. Analyzuje různá velkoplošná zařízení, způsoby generování terénu a poté popisuje návrh a implementaci prototypu vizualizace terénu generovaného pomocí Perlinova šumu pro velkoplošná zařízení běžící pod systémem SAGE2. Terén je rozdělen do desíti různých biomů a uživateli je umožněna pohyb kamery nad tímto terénem.

Klíčová slova Návrh a implementace aplikace, testování aplikace, 3D vizualizace terénu, procedurální generování, šum, Perlinův šum, biomy, three.js, SAGE2

Abstract

This thesis explores the possibilities of generating and visualising of procedural terrain on large scale displays, and focuses on the terrain diversity. It analyses a set of large scale display systems and methods of generating terrain. The thesis describes a design proposal and implementation of a prototype visualisation of an infinite terrain generated using Perlin noise for large scale displays running the SAGE2 system. The terrain is comprised of ten different biomes and the control of the camera is given to the user.

Keywords Application modeling and implemetntation, application testing, 3D terrain visualisation, procedural generation, noise, Perlin noise, biomes, three.js, SAGE2

Obsah

Seznam obrázků

Úvod

Technologie zobrazovacích systémů s vysokým rozlišením se prudce vyvíjí. Mobilní zařízení mají často vyšší rozlišení než pouze pár let staré stolní monitory, na trh se dostávají cenově dostupné 4K televizory, a dokonce i samotný multimédiální obsah je nyní dostupný i ve vyšších rozlišeních.

Velkoplošné zobrazovací systémy složené z několika těchto relativně dostupných zobrazovacích ploch mohou být využity k prezentaci detailnějších dat, než by bylo možné při použití obyčejných velkoplošných projektorů.

Jednou z možností využití těchto zařízení bývá zobrazování již existujících naměřených dat v širším kontextu, kde je díky velikosti displeje možno snáze vidět souvislosti mezi jednotlivými datovými body. Stejně tak mohou být tato zařízení využívána pro zobrazování člověkem vytvořených dat ve formě obrázků či modelů. Jako poslední druh dat, která můžeme zobrazovat, jsou data, která nebyla předem naměřena ani nebyla vytvořena člověkem: procedurálně generovaná data. Tato data mají oproti ostatním typům dat výhodu, že jsou reprezentována algoritmem, který zabírá mnohonásobně méně paměti než data, která byla předem naměřena či vytvořena.

Tato práce je zaměřena na analýzu současných velkoplošných systémů, shrnuje využívané algoritmy pro procedurální generování terénu a obsahuje návrh a implementaci aplikace pro systém SAGE2, která vizualizuje terén generovaný pomocí jednoho z předem popsaných algoritmů. Předcházející práce týkající se tohoto tématu neměly za cíl vizuální prezentaci, která by mohla zaujmout

neodborné publikum, a proto nebyly vhodné k předvádění zařízení tohoto typu veřejnosti. Tato práce se naopak zaměřuje na samotný vzhled terénu generovaného výsledným prototypem.

Práce má dvě části: teoretickou a praktickou. V teoretické části je bude popsána obecná problematika velkoplošných systémů a specifická implementace jednoho takového systému — SAGE2. Budou rozebrány možnosti tvorby aplikací běžících pod tímto systémem a možnosti procedurální generace terénu reprezentovaného v datech. V praktické části bude popsán návrh systému generování a následné vizualizace terénu na systému SAGE2. Tento návrh bude implementován prototypem popsaným v kapitole 4 a v následující kapitole bude tento prototyp podroben vhodným testům.

Tato práce z části navazuje na práce bývalých studentů bakalářského studia FIT ČVUT, specificky na tyto práce:

- ***S10W: Vizualizace plně procedurálního dynamického terénu na velkoplošné obrazovce*** [1] od Michala Vomastka, zabývající se procedurálním generováním jednoduchého terénu na systému SAGE2.
- ***10K SAGE2 Dashboard: Sada widgetů pro velkoplošnou obrazovku*** [2], jejímž autorem je Aleh Kuchynski, a zabývá se analýzou systému SAGE2 a možnostmi API tohoto systému.
- ***10K SAGE2 Dashboard: Sada widgetů pro velkoplošnou obrazovku*** [3] od Radka Meduny, zabývající se analýzou systému SAGE2 a návrhem aplikací pod daný systém.

Cíl práce

Práce je rozdělena do teoretické a praktické části. Cílem teoretické části práce je shrnutí znalostí potřebných pro vytvoření praktické části práce. Jelikož se práce týká dvou relativně odlišných témat (velkoplošné displeje a procedurální generování), je třeba prozkoumat jak možnosti různých velkoplošných systémů a systému SAGE2, na který je praktická část zaměřena, tak analyzovat vlastnosti terénu, které chceme reprezentovat, a způsoby, kterými je možné terén s těmito vlastnostmi generovat.

Cílem praktické části práce je samotný návrh a implementace prototypu aplikace pro systém SAGE2, který bude vizualizovat jeden ze způsobů generování terénu popsaných v teoretické části, a jeho následné otestování.

Kapitola 1

Velkoplošné zobrazovací systémy a systém SAGE2

V této kapitole je popsána analýza zobrazovacích systémů využívajících velké panely displejů a jsou zhodnoceny výhody a nevýhody těchto systémů. Dále je zde také podrobněji rozebrán systém SAGE2 a způsob využití SAGE2 API (aplikačního rozhraní) pro vytvoření aplikace běžící na daném velkoplošném systému.

1.1 Software pro management velkoplošných systémů

Tato sekce obsahuje shrnutí existujícího software pro správu zobrazovaného obsahu na videostěnách a dalších velkoplošných zobrazovacích systémech. Tyto systémy se navzájem v mnoha ohledech liší, ať již jde o účel (od neinteraktivních digitálních cedulí – tzv. digital signage, interaktivních kiosků až po systémy podporující víceuživatelskou kolaboraci) či využité technologie (cenově dostupný hardware versus specializovaná technika pouze pro určitou platformu či multiplatformní).

Níže následuje shrnutí vybraných systémů pro správu těchto velkoplošných systémů a obsahu na nich zobrazovaného:

- **Microsoft Windows OS** – Zjevnou možností je přímé využití grafického rozhraní operačního systému jednoho počítače bez specializovaného systému pro správu více monitorů. Tento systém je pro mnoho využití plně dostačující a umožňuje nativní zobrazování aplikací běžících na standardní instalaci desktopového operačního systému. Toto řešení má však mnoho nedostatků. Systém není možno snadno využít pro kolaboraci, ovládání je ve výchozím stavu omezeno na ovládání periferiemi připojenými přímo k hlavnímu stroji, při pádu systému chybí možnost využití záložního zobrazovacího serveru a systém nelze připojit k většímu množství zobrazovacích ploch bez využití specializovaného hardware v podobě různých embedded zařízení.

- **Matrox**[1] – Placený systém pro videostěny využívající specializovaného hardware od firmy Matrox, který obsahuje podporu zobrazování a interagování s externími aplikacemi běžícími pod OS Windows. Tento systém také podporuje embedded zařízení běžící pod systémem Window Embedded.

- **Hiperwall**[2] – Placený systém řešený na zakázku, využívá snadno dostupného hardware – malých PC klientů propojených po síti k hlavnímu serveru. Není určený pro interaktivní a kolaborativní obsah, ale specializuje se na zobrazování videa a animací. Umožňuje propojení více oddělených stěn a redundantního grafického serveru pro případ poruchy.

- **Navori QL**[3] – Placený systém podobný systému Hiperwall určený pro zobrazování streamovaného videa a digital signage, avšak v mnoha ohledech rozšířenější. Klientem může být nejen PC ale také chytrá televize, podporuje nestandardní rozložení displejů a omezenou interaktivitu s HTML5 obsahem pomocí dotykové obrazovky či periferií napojených ke klientovi, avšak nepodporuje kolaborativní interaktivitu.

- **Polywall**[4] – Placený systém s free trial verzí využívající snadno dostupný hardware s klientskými počítači a servery bežícími pod OS Windows. Podporuje management více stěn, umožňuje automatizaci zobrazování

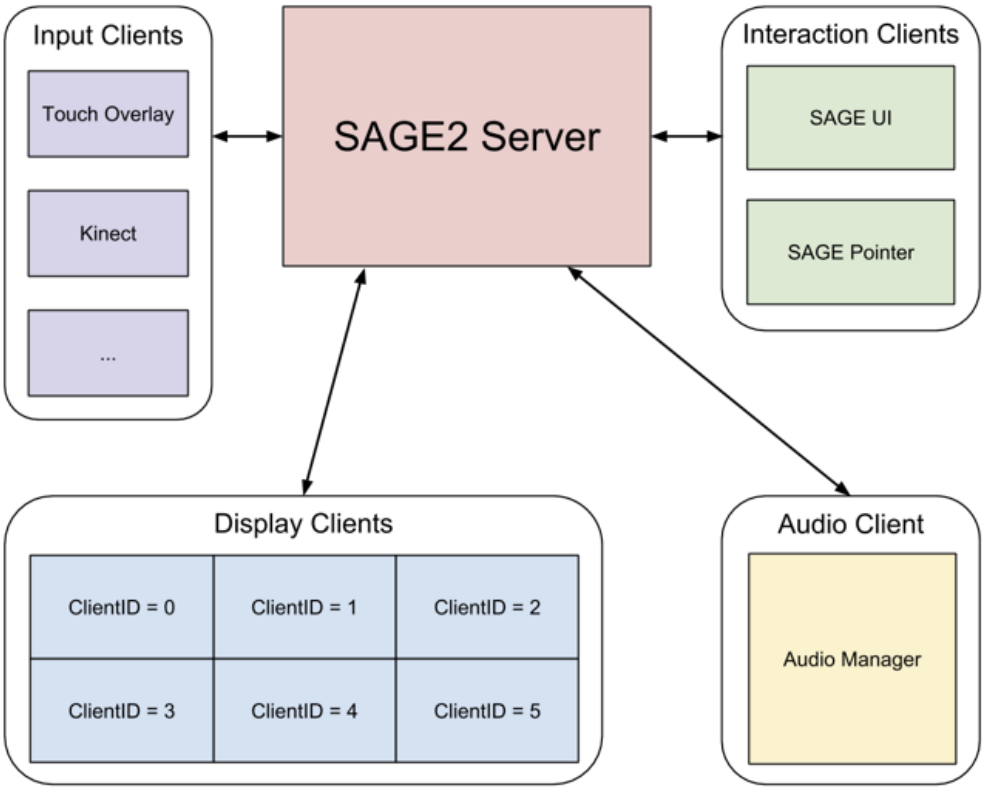
[1] `http://www.matrox.com/graphics/en/solutions/video_display_wall/`
[2] `http://www.hiperwall.com/Why-Hiperwall.aspx`
[3] `https://www.navori.com/digital-signage-software/`
[4] `https://www.polywall.net/polywall`

obsahu, zobrazování obsahu kolaborativně s omezením přístupu uživatele pouze na určité části stěn, ale neumožňuje zobrazovat plně interaktivní aplikace běžící na stěně.

- **Userful**[5] – Placený systém s free trial verzí, využívá webové technologie a standardně dostupný hardware, podporuje interaktivitu nad desktopovými aplikacemi běžícími na stěně. Umožňuje libovolné natočení jednotlivých obrazovek.
- **PiWall**[6] – Volně dostupný systém pod GPL licencí, určený pro malé jednodeskové počítače Raspberry Pi, kde každý stroj ovládá výstup jednoho displeje. Je určen pro přehrávání videa na velkém množství obrazovek (až 300 propojených displejů). Není zaměřen na interaktivní videostěny.
- **SAGE2**[7] – Open source systém dostupný zdarma, vyvíjený v akademickém prostředí, zaměřený na lokální i vzdálené kolaborativní užití. Nabízí možnost propojení více stěn v různých lokacích. Nevyžaduje specializovaný hardware, klient i server napsaný v JavaScriptu, systém je tedy multiplatformní. Podporuje zobrazování interaktivního HTML5 obsahu.

Práce se dále bude zabývat právě posledním zmíněným systémem SAGE2, který je pro potřeby praktické části dostačující: podporuje interaktivní aplikace pomocí API založeného na moderních webových technologiích, umožňuje využití již existujících JavaScriptových knihoven v projektu, je volně dostupný společně se zdrojovým kódem a vyvíjený v akademickém prostředí. Samotný systém je také dostatečně rozšířený, s více jak osmdesáti stěnami v 53 zemích světa[4] včetně stěny dostupné v laboratoři SAGElab, na které je možné projekt testovat.

[5] `https://www.userful.com/`
[6] `http://www.piwall.co.uk/`
[7] `http://sage2.sagecommons.org/`



Obrázek 1.1: Architektura systému SAGE2 [5]

1.2 Struktura systému SAGE2

V této sekci je podán stručný popis jednotlivých částí systému SAGE2 a jejich funkcionality. Tyto části mohou běžet jak centralizovaně na jednom stroji, tak i distribuovaně na více síťově propojených strojích. Na obrázku 1.1 je znázorněno schéma zapojení jednotlivých částí systému.

Hlavní částí systému je **serverová aplikace** (*SAGE2 Server*) napsaná v JavaScriptu běžícím pod systémem *node.js*, která obstarává komunikaci se všemi dalšími částmi aplikace a stará se o rozložení obsahu na stěně. Tato část aplikace je jedinou částí systému, kterou je třeba nainstalovat, pro zprovoznění stěny, veškeré ostatní části jsou pouhými „tenkými klienty“, které poskytuje server skrze webový prohlížeč.

Pro zobrazování obsahu na samotnou stěnu se využívají **zobrazovací klienti** (*Display clients*), kterých může být do serveru zapojeno více najednou. Každý z těchto klientů obstarává zobrazování části stěny určené serverem, jako celek se starají o zobrazování celé stěny či více stěn. Jelikož veškerý kód aplikací postavených nad systémem SAGE2 běží odděleně na každém zobrazovacím klientovi zvlášť, vývojář dané aplikace musí dbát na to, aby byl stav aplikací mezi jednotlivými klienty stejný, k čemuž může využít funkcionality vestavěné do SAGE2 API. Z uživatelského hlediska je tedy vytvořena iluze jedné aplikace roztažené na více monitorů, ačkoliv pro každého zobrazovacího klienta jde o oddělenou aplikaci. Pokud stav aplikací stejný není, mohou různé monitory ve stejném okamžiku pro stejnou aplikaci zobrazovat jiný obsah, což tuto iluzi jedné aplikace ničí.

Další částí systému je **klient pro interakci se stěnou** (*SAGE2 UI*). Tento klient dovoluje uživateli manipulovat s obsahem zobrazovaným na stěně. Umožňuje spouštět nové aplikace, manipulovat s okny jednotlivých aplikací a ovládat ukazatel na stěně (*SAGE2 Pointer*), s jehož pomocí je možno manipulovat se samotným obsahem interaktivních aplikací. Jelikož veškeré klientské části SAGE2 jsou dostupné skrze webový prohlížeč a nevyžadují instalaci, uživatel může pro interakci se stěnou využít jakékoliv zařízení s webovým prohlížečem (např. mobilní telefon, tablet, notebook či desktopový počítač).

Systém také obsahuje klienty, které nejsou pro čisté zobrazování obsahu na stěně zcela nutné, ale které dodávají užitečné funkcionality: **audio klient** (*Audio client*), určený jako hlavní výstupní bod pro zvuk vydávaný běžícími aplikacemi, a klienty pro různé další periferní zařízení či jiné vstupy dat do systému (*Input clients*).

Weboví klienti poskytovaní SAGE2 serverem jsou psány pomocí HTML5 a JavaScriptu, takže jde o multiplatformní systémy schopné běhu na jakémkoliv operačním systému. Přesto však vývojáři systému doporučují pro nejlepší běh SAGE2 webový prohlížeč Google Chrome či Electron [5].

1.3 Analýza SAGE2 API

V této sekci je rozebráno využití SAGE2 API pro tvorbu aplikace spustitelné na velkoplošné stěně běžící pod systémem SAGE2. API je popsáno v textech [6, 5]. Každá taková aplikace se skládá z JavaScriptového objektu dědícího z třídy `SAGE2_App`, dodatečných JavaScriptových knihoven, souboru `instructions.json` popisujícího aplikaci a obrázku reprezentující ikonu aplikace.

Hlavní objekt každé aplikace vytvořené pro SAGE2 přetěžuje řadu metod z rodičovské třídy, konkrétně metody:

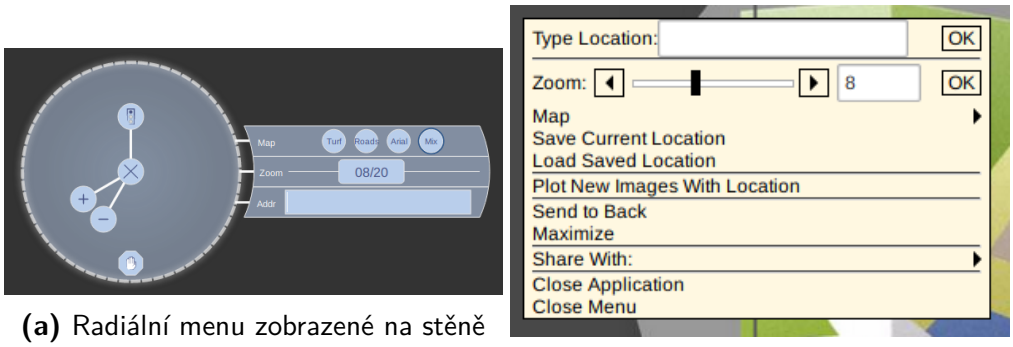
- **init**, starající se o inicializaci aplikace po spuštění,
- **load**, načtení pozměněného stavu,
- **draw**, volané před vykreslením aplikace na stěnu,
- **resize**, volané při změně velikosti okna aplikace,
- **move**, volané při přesunutí aplikace na ploše stěny,
- **event**, obstarávající události vyvolané kursorem, klávesnicí, či stiskem tlačítka v uživatelském rozhraní, a
- **quit**, volané při ukončení aplikace.

Soubor `instructions.json` popisuje strukturu celé aplikace systému SAGE2, určuje soubor, který obsahuje hlavní třídu programu, další soubory, obsahující využívané knihovny, soubor s ikonou a další metadata popisující aplikaci.

Samotná aplikace může zobrazovat data pomocí veškerých systémů dostupných v jazyce JavaScript v prohlížeči, ať už se jedná o přímou modifikaci HTML pomocí DOM JavaScript API, využití SVG s knihovnou *snap.svg*[8] (distribuované společně s SAGE2), či pomocí HTML5 canvas objektu manipulovaného pomocí knihovny přidané k aplikaci.

Dosud popisovaný systém se strukturou tolik neliší od jiných JavaScriptových aplikací, avšak SAGE2 API poskytuje také systémy umožňující synchronizaci

[8] http://snapsvg.io/



(a) Radiální menu zobrazené na stěně

(b) Nabídka v klientu pro interakci

Obrázek 1.2: Ovládací prvky poskytované v SAGE2 API

stavu instancí aplikace běžících na jednotlivých displejích. Stav je synchronizován pomocí metody `broadcast`, která může být použita ke komunikaci mezi jednotlivými displeji. Jeden z displejů je vždy označen pomocí proměnné `isMaster` jako hlavní, čímž se může předejít kolizím v datech vysílaných displeji.

V neposlední řadě také API poskytuje možnost tvorby jednoduchého uživatelského rozhraní: pomocí pravého tlačítka kurzoru na stěně se zobrazí malé radiální menu a stiskem pravého tlačítka na reprezentaci aplikace přímo v klientu pro interakci se stěnou se zobrazí menu druhé. První z těchto dvou nabídek se nastavuje pomocí objektu `this.controls` a jeho výstup je zpracovávaný v metodě `event`, druhý pomocí přetížené metody `getContextEntries()`, která vrací seznam položek v menu a funkcí, které tyto položky obsluhují. Ukázku těchto ovládacích prvků je možno nalézt na obrázku 1.2.

1.4 Analýza existujících aplikací na SAGE2

Pro systém SAGE2 již bylo vytvořeno velké množství aplikací, některé z nich dostupných na SAGE2 App Repository[9]. Tato sekce se zabývá analýzou vybraných aplikací z tohoto repozitáře a dalších aplikací vytvořených v průběhu předchozích prací, které se zabývají systémem SAGE2.

[9] http://apps.sagecommons.org/

1.4.1 Aplikace pro využití SAGE2 jakožto dashboard

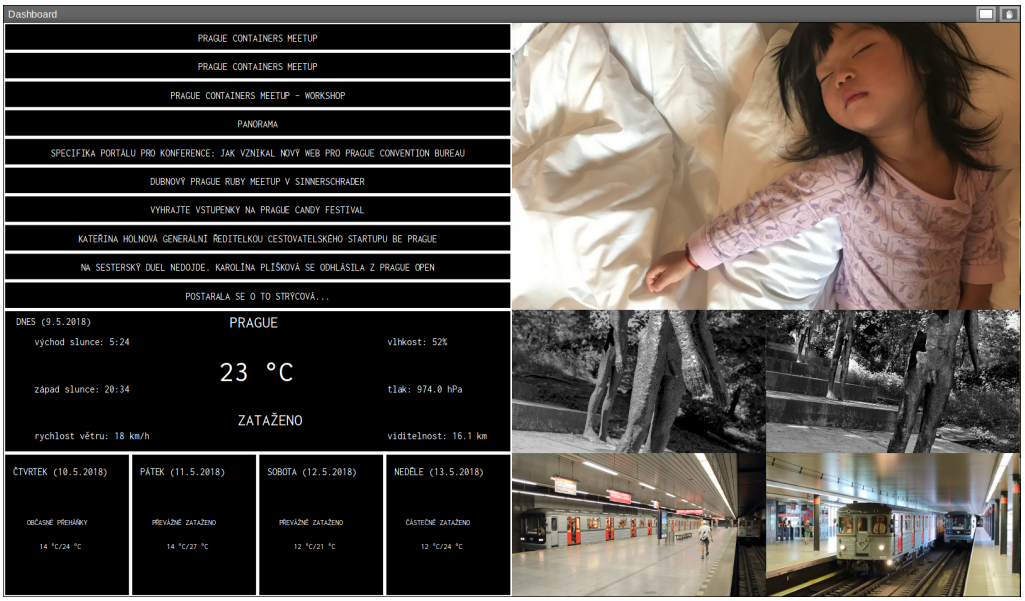
První analyzovanou aplikací je aplikace *Dashboard* od Radka Meduny[3] (náhled aplikace obr. 1.3), jenž je autorem bakalářské práce pod systémem SAGE2, která byla obhájena v roce 2018. Práce Radka Meduny má s touto prací pouze částečný překryv – zabývá se vizualizací dat na systému SAGE2, ale jde o vizualizaci již existujících 2D dat se zaměřením především na interakci uživatele s aplikací. Jeho aplikace shromažďuje více různých widgetů do jednoho dashboardu, který nabízí souhrn užitečných informací o nějakém městě. Každý implementovaný widget zobrazuje určitý druh informací relevantní k danému městu: počasí, novinky a projekci obrázků ze služby Flickr. Aplikace je stavěna pomocí JavaScriptu a SAGE2 API za využití knihovny *d3.js*[10] pro generování zobrazovaného 2D HTML obsahu. Tato knihovna je již zahrnuta v SAGE2 API, což velmi usnadňuje tvorbu aplikací.

Práce obsahuje návrh rozšiřitelného systému widgetů, které mohou být pro aplikaci vytvořeny a které synchronizují stav mezi jednotlivými zobrazovacími klienty. Samotná aplikace tuto synchronizaci řeší například při zobrazování náhodných obrázků z daného města. Systém SAGE2 umožňuje zvětšovat a zmenšovat okno aplikace, ale to způsobuje problémy se samotným rozložením obsahu na obrazovce. Například velikost textu byla autorem nastavena pevně, což znamená, že při otevření aplikace na zdi s vyšším rozlišením je text prakticky vždy příliš malý pro pohodlnou čitelnost.

Aleh Kuchynski[2] je autorem podobného systému jako dříve popsaný Dashboard od Radka Meduny. Při své analýze došel k názoru, že většina již existujících widgetů pro systém SAGE2 je tvořena jako nedělitelné a na sobě nezávislé aplikace, které nejsou určené pro více účelů najednou, a vytvořil proto místo jednotné aplikace několik samostatných aplikací, které jsou na sobě zcela nezávislé. Na analýzu byla vybrána aplikace *DigitalClock* (náhled na obrázku 1.4), realizující digitální hodiny s alarmem.

Aplikace je psaná také pomocí *d3.js* a HTML, avšak oproti předchozí analyzované aplikaci správně reaguje na zvětšování a zmenšování okna. Aplikace umožňuje limitovanou interakci, jako například změnu časové zóny či nastavení alarmu, avšak toto nastavení je dostupné pouze skrze SAGE2 Cursor a nelze

[10] `https://d3js.org/`



Obrázek 1.3: Náhled aplikace *Dashboard*

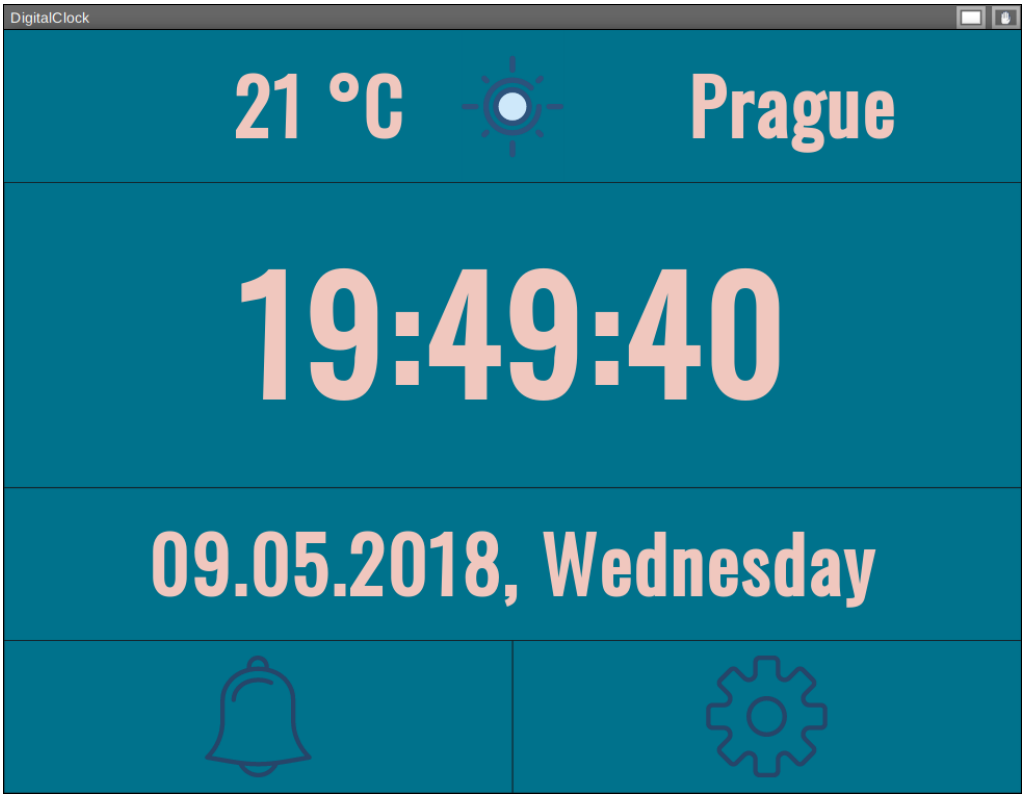
jej upravit přímo v klientu pro interakci se stěnou. Nastavení se také neukládá, takže pokud uživatel provede nějakou změnu a aplikaci restartuje, veškeré změny se ztratí.

1.4.2 Vizualizace s pomocí knihovny *three.js*

Dalším druhem analyzovaných aplikací jsou aplikace zobrazující vizualizaci 3D dat. K analýze byla zvolena aplikace *Car in Three.js*[11] (náhled na obrázku 1.5), která zobrazuje model automobilu a umožňuje uživatli měnit pozorovací úhel. Aplikace, jak již název napovídá, využívá JavaScriptovou knihovnu *three.js*[12] pro zjednodušení práce s grafickou kartou a zobrazování 3D dat.

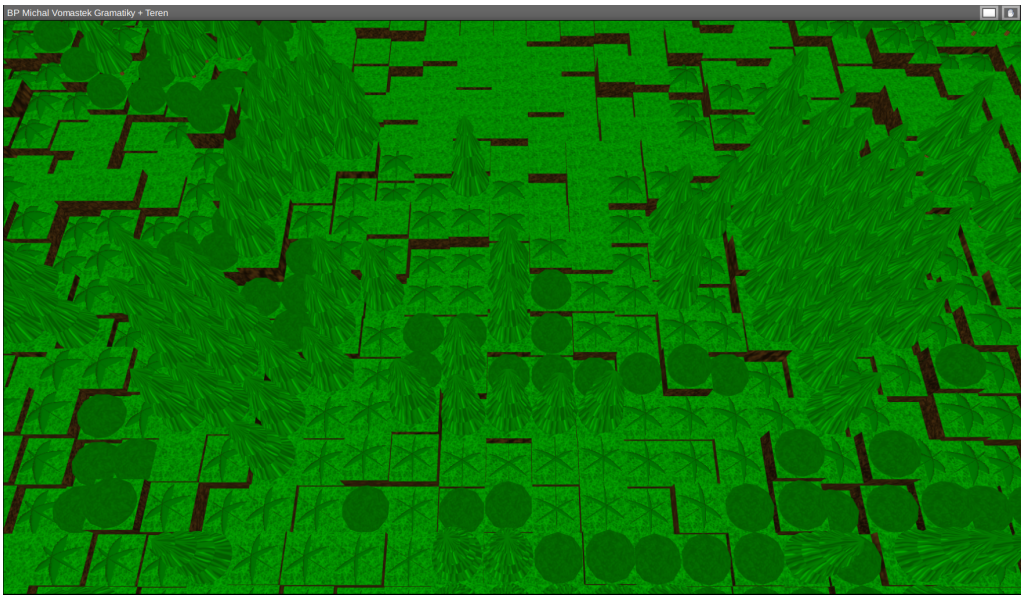
Tato knihovna využívá systém WebGL pro vykreslování dat na grafické kartě a velmi zjednodušuje interakci s tímto systémem. Poskytuje graf scény, velké množství předdefinovaných materiálů (grafických shaderů), které mohou být rozšiřovány, umožňuje vytvoření vlastních materiálů pomocí jazyka GLSL a také načítání objektů z různých datových formátů.

[11] `http://apps.sagecommons.org/a/-KWY_QULT_T6VOGkYETZ`
[12] `https://threejs.org/`



Obrázek 1.4: Náhled aplikace *DigitalClock*

Obrázek 1.5: Náhled aplikace *Car in Three.js*



Obrázek 1.6: Náhled aplikace Michala Vomastka

1.4.3 Vizualizace procedurálního terénu na SAGE2

Jako poslední byla analyzována aplikace od Michala Vomastka[1] (náhled na obrázku 1.6), která má velmi podobné zaměření jako tato práce. Pro větší překryv tématu je tedy jeho práci věnováno více prostoru. Jedná se o aplikaci zobrazující procedurálně generovaný terén na SAGE2. Terén je složený z různě vysoko položených krychlí, na kterých jsou položeny povrchové objekty (strom, keř, tráva…). Aplikace je psaná v čistém JavaScriptu a WebGL bez využití jakékoliv knihovny pro zjednodušení vizualizace 3D dat. To implementaci aplikace výrazně zesložiťuje, jelikož muselo být implementováno také např. načítání jednotlivých modelů, což bylo v předchozí analyzované aplikaci obstaráváno knihovnou *three.js*.

Generování terénu je implementováno dvěma různými způsoby: pomocí pravděpodobnostního modelu a pomocí gramatik. Pravděpodobnostní model jako vstup bere 8-okolí generovaného prvku a postupně ho zpracovává pravidly s dvěma vstupy. Tato pravidla mají jako výstup pravděpodobnostní hodnoty možných generovaných prvků (například pro vstup dvou stromů je pravděpodobnost vygenerování stromu 86%, planiny 7% a křoví 7%). Opakovaným voláním těchto pravidel (a vyhodnocení pravděpodobností) na všechny prvky

8-okolí a jejich výstupy se získá prvek, který bude umístěn na právě generovanou pozici. Díky tomu, že samotná pravidla jsou relativně jednoduchá (mají pouze dva vstupy, u kterých nezáleží na pořadí), je počet pravidel nutných k pokrytí všech možností relativně nízký ($\frac{g+g^2}{2}$, kde g je počet generovaných prvků).

Při generování pomocí gramatik se pro každý prvek vyhodnocuje pravidlo pouze jednou, avšak počet pravidel je mnohonásobně větší. Pravidla v tomo případě na vstupu přijímají tři sousedící prvky, u kterých záleží na pořadí, což zanamená, že pro pokrytí všech možností na vstupu je nutno mít mnohonásobně více pravidel (g^3).

V obou dvou těchto případech byla pravidla definována v souborech přiložených ke generátoru, což umožňuje uživateli před instalací na SAGE2 tato pravidla pozměňovat a tím získat jiné výsledky. U modelu gramatik je však velmi obtížné vytvořit sadu správných pravidel pro generování obsahu, jelikož například i v samotné implementaci aplikace může docházet k situacím, kdy se v určitých směrech generují lesy méně často než ve směrech jiných. K tomu dochází nejspíše z toho důvodu, že ve výstupu některých pravidel jsou pravděpodobnosti generování určitých prvků nastaveny na příliš nízkou či příliš vysokou hodnotu.

Protože je samotné vyhodnocování pravděpodobností, které jsou výstupem pravidel, založené na náhodě, v aplikaci je také řešena synchronizace mezi jednotlivými klienty systému SAGE2. Obsah je generován pouze na master klientovi, který ostatním klientům předává již vygenerované pozice grafických prvků.

Vizualizace samotné aplikace je psána efektivně pomocí WebGL, samotné grafické prvky jsou jednoduché a nejsou tedy náročné na vykreslování, a aplikace konzistentně udržuje počet snímků za vteřinu na hodnotách vyšších než 30 FPS. Vizuální stránka je ale subjektivně nepěkná, což je způsobeno monotónním vzhledem – tráva je stále stejně zelená, stromy vypadají všechny totožně, výškově se terén moc neliší a kamera umožňuje sledovat terén pouze z pevně daného úhlu.

1.5 Shrnutí

V této kapitole bylo popsáno osm různých systémů pro management velkoplošných zařízení. Detailněji byl poté popsán systém SAGE2 a jeho API, jelikož pod tímto systémem bude implementován prototyp aplikace popsaný v prakitcké části práce. Dále byly rozebrány čtyři různé aplikace vytvořené pro tento systém, dvě z pro vizualizaci 3D dat. Největší pozornost byla věnována aplikaci pro vizualizaci procedurálního terénu, která se svým tématem nejvíce přibližuje tématu této práce.

Kapitola 2

Možnosti procedurálního generování

Tato kapitola se zabývá procedurálním generováním vizuálního obsahu, jeho současným využitím a algoritmy, které jsou ke generování často využívány. Zaměřuje se obzvlášť na generování terénu, proto se také pokusí popsat vlastnosti terénu, který má být generován.

Za procedurální generování obsahu je podle [7] považováno algoritmické generování obsahu bez vlivu nebo s minimálním vlivem uživatele. V samotné definici není omezeno, o jaký obsah se specificky jedná, může jít jak o audiovizuální stránku určitého díla, tak i například o pravidla ve videohře. Co však pod procedurální generování nespadá, je samotný engine či umělá inteligence nehráčských postav v počítačových hrách. Ta spadá spíše pod pojem AI. Je nutno také zmínit, že algoritmy pro procedurální generování mohou i nemusí být deterministické, není nutné, aby generovaný obsah byl vždy stejný.

Procedurální generování je často využíváno pro vytváření náhodného obsahu v počítačových hrách, např. hra *Elite* z roku 1984 mohla pomocí procedurálního generování obsahovat stovky rozličných hvězdných systémů, které celkově zabíraly pouhé kilobajty paměti, či hra *.kkrieger*, která s velikostí méně než 100 kB obsahuje graficky velmi vyspělou 3D střílečku. Avšak procedurální generování najde využití i ve filmech, pro které byl vyvinut například algoritmus pro generování Perlinova šumu.

2.1 Vlastnosti terénu

Pro popis procedurálního generování terénu je nejprve nutno upřesnit, co je považováno za terén a jaké má vlastnosti. Terén je pojem, který je definován jako:

> „*Část zemního povrchu (pevniny) tvořená terénním reliéfem, pokrytým objekty, jako např. porostem, vodstvem, komunikacemi, stavbami, technickými zařízeními.*“[8]

Generování lze tedy obecně rozdělit do dvou částí, a to generování výškového reliéfu a generování objektů, které na reliéfu leží. Tento terén lze klasifikovat podle reliéfu (např. rovina, pahorkatina, hornatina…) či podle biologického rozložení do různých biomů (např. poušť, les, tundra…).

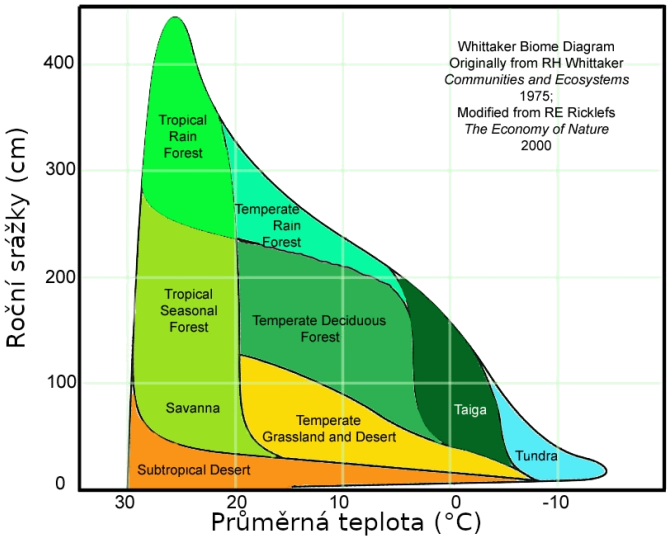
Způsobů, jakým je skutečný zemský terén klasifikován do biomů, je mnoho, ale jedno z známějších dělení biomů (vytvořeno Robertem Whittakerem) klasifikuje terén do biomů podle dvou abiotických vlastností terénu: ročních srážek a průměrné teploty. Pokud je tedy předem známá mapa teplot a vlhkostí, je snadné klasifikovat oblasti na mapě do určitých biomů. Tento způsob rozdělení terénu do biomů je popsán v [9]. Graf tohoto dělení je možno vidět na obrázku 2.1.

2.2 Způsoby procedurálního generování

Níže jsou popsány možné způsoby procedurálního generování terénu a obsahu na něm. Jednou ze snadno rozpoznatelných vlastností přírodních útvarů je soběpodobnost, kde části objektu jsou podobné jiným částem stejného, zvětšeného či zmenšeného objektu. Níže popsané algoritmy se soběpodobnost snaží různými způsoby napodobovat.

2.2.1 Diamond-square algoritmus

Tato metoda generování terénu (popsaná v [7]) probíhá na mřížce s pevně definovanou velikostí. Algoritmus začne nastavením rohových hodnot mřížky (ať již náhodně nebo s vstupem uživatele) a poté pokračuje opakováním kroků „diamond step“ a „square step“.



Obrázek 2.1: Whittakerův diagram biomů [10]

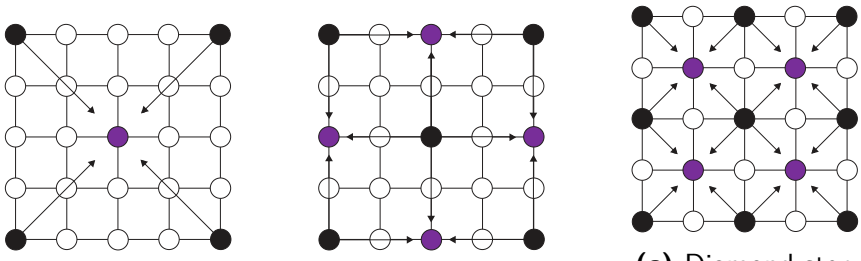
Při „**diamond step**“ se zprůměruje hodnota každé nejmenší čtveřice již vyhodnocených bodů ležících ve čtverci a s přičtenou náhodnou hodnotou je výsledná hodnota zapsána do bodu uprostřed tohoto čtverce.

Při „**square step**“ se provede krok velmi podobný předchozímu, ale využijí se nejmenší čtveřice bodů ležících v „diamantu“, tedy čtverci otočeném o 45°. Tyto kroky se opakují do té doby, dokud není mřížka zaplněna, a při každém opakování se rozptyl možných náhodných hodnot snižuje, aby docházelo k stále jemnějším variacím v terénu. Tato metoda je relativně snadná na implementaci, avšak jelikož je terén tvořen iterací, nelze snadno tento proces paralelizovat na grafické kartě. Jednotlivé kroky diamond-square algoritmu jsou znázorněny na obrázku 2.2.

2.2.2 Metody generování šumem

Další možností pro generování realisticky vypadajícího terénu je využití funkcí generujících šum. Ken Perlin[1] popsal šumovou funkci `Noise()` jako

[1]Ken Perlin poprvé popsal využití šumu pro 3D grafiku v textu *An image synthesizer* v roce 1985



(a) Diamond step **(b)** Square step **(c)** Diamond step podruhé

Obrázek 2.2: Jednotlivé kroky algoritmu Diamond-square [7]

funkci s trojrozměrným vektorem na vstupu, pro kterou platí následující tři vlastnosti:

- **statistická invariance vůči rotaci** (funkce má stejné statistické vlastnosti nezávisle na rotaci),
- **úzká pásmová propustnost dané frekvence** (funkce má viditelné vlastnosti pouze v určitém rozmezí frekvence),
- **statistická invariance vůči translaci** (funkce má stejné statistické vlastnosti nezávisle na posunu).[11]

Formálnější definici šumu lze najít v *A Survey of Procedural Noise Functions*[12], kde po analýze vlastností několika již existujících funkcí šumu je šum definován takto:

> „*A noise is a stationary and normal random process. Control of the power spectrum is provided, either directly, or through the summation of a number of independent scaled instances of (typically band-limited) noise.*[2]“[12]

Dále je v textu definován pojem *funkce procedurálního šumu* jakožto procedurální techniky pro simulování a vyhodnocování šumu. C++ knihovna *libnoise*[13] pro generování terénu využívá ve své dokumentaci přísnější definici funkce koherentního šumu. Za funkci generující koherentní šum považuje jakýkoliv algoritmus, pro který platí tyto tři pravidla:

[2]Překlad autor: „Šum je statický a normální náhodný proces. Poskytuje způsob k ovlivnění výkonového spektra, buď přímo, či pomocí součtu několika nezávislých škálovaných instancí (typicky pásmově omezeného) šumu.“

- Pro stejnou vstupní hodnotu vždy získáme stejnou výstupní hodnotu (jde o deterministický algoritmus).
- Malá změna ve vstupní hodnotě způsobí malou změnu v hodnotě výstupní.
- Velká změna ve vstupní hodnotě způsobí náhodnou změnu v hodnotě výstupní.[13]

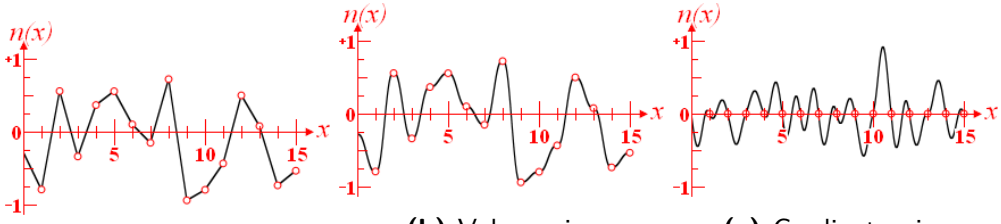
Jak je tedy vidět, definic je více a každá se soustředí na jiné vlastnosti, které šum vykazuje.

Detaily různých metod generování procedurálních šumů jsou popsány v knize *Texturing & modeling: a procedural approach*[14] v kapitole Making noises, kde je popsána metoda generování šumů pomocí mřížky. Při generování těchto šumů se nejprve pseudonáhodně určí vlastnosti bodů ležících na určité mřížce (například hodnota bodů ležících na mřížce všech celých čísel) a body, které na mřížce neleží, se vypočítají pomocí interpolace hodnot okolních bodů ležících na mřížce. Těmto šumům se obecně říka *lattice noises*.

Nejjednodušší formou *lattice noise* je *value noise*, pro jehož generování jsou na body ležící na celých číslech umístěny pseudonáhodně generované hodnoty a pro ostatní body jsou hodnoty získány interpolací z nejbližších celočíselných hodnot. To však způsobuje problémy s artefakty, jelikož takto definovaný šum umisťuje lokální maxima a minima do viditelné mřížky.

Proto se často také využívá *gradient noise*, který problém s těmito artefakty řeší. Tento šum na bodech ležících na celých číslech nabývá nuly a místo toho je náhodně generován vektor gradientu funkce v daném bodu – „svah“, který funkce na daném místě má. Interpolací těchto vektorů je poté možno získat funkci s menším množstvím artefaktů, která nemá lokální maxima a minima omezena pouze na celočíselné hodnoty.

Jednou z implementací *gradient noise* je 3D funkce *Perlinova šumu*, kterou vyvinul Ken Perlin v roce 1983 pro generování textur ve filmu *Tron* a je popsána v textu [16]. Na pozice všech celých čísel se dosadí nulová hodnota a určí se náhodný gradient vector, který udává směr a úhel „svahu“ funkce v daném bodě. Hodnoty ostatních bodů jsou vypočítány interpolací hodnot, které by měly na daném místě „svahy“ nejbližších celočíselných bodů, kdyby pokračovaly na



(a) Value noise s lineární interpolací

(b) Value noise s interpolací pomocí polynomu pátého stupně

(c) Gradient noise s interpolací pomocí polynomu pátého stupně

Obrázek 2.3: Znázornění různých 1D šumů [15]

místo právě vyhodnocovaného bodu. Interpolace se provádí pomocí polynomu pátého stupně $f(t) = 6t^5 - 15t^4 + 10t^3$, čímž se získá funkce, která má první i druhou derivaci spojitou.

V jazyce OpenGL Shading Language (GLSL) pro grafické karty je k dispozici funkce *noise* generující procedurální šum, avšak dle dokumentace [17] není definován přesný postup pro implementaci dané funkce, což může mít za následek různý výstup pro implementace od různých výrobců. Tato funkce se navíc nenachází ve variantě jazyka určené pro webové aplikace a mobilní zařízení GLSL ES.

Šum sám o sobě nemá vlastnost soběpodobnosti. Pro zajištění této vlastnosti je možno využít metody popsané v [18]: využije se součet několika instancí šumu s vždy dvojnásobnou frekvencí a postupně se zmenšující amplitudou. Tyto jednotlivé instance sečtené do výsledného šumu jsou nazývány *oktávami* a vytvářejí ve výstupu soběpodobnost.

Takto generovaný soběpodobný šum lze využít pro generování textur jak pro povrch objektů, tak jako výškovou mapu samotného terénního reliéfu. Úpravou frekvencí, amplitudy či úpravou výstupu pomocí absolutní hodnoty či simulováním turbulence lze výstup tohoto šumu ovlivňovat a získat různorodé výškové mapy pro různý druh materiálu či terénu.

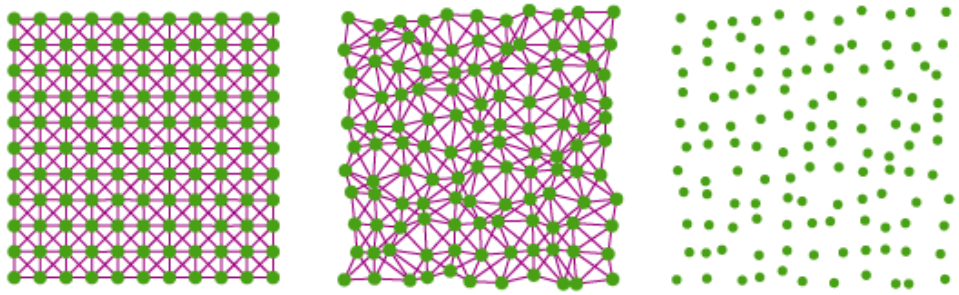
2.2.3 L-systémy

Výše byly popsány způsoby generování reliéfu terénu. Pro generování objektů ležících na povrchu terénu a jejich pozic je však využití těchto technik omezené.

Obrázek 2.4: Rostlina vygenerovaná pomocí L-systému [19]

Pro generování samotných objektů se proto využívají jiné algoritmy, například pomocí Lindenmayerových systemů, neboli *L-systémů*, které poprvé popsal Aristid Lindenmayer v roce 1968. Způsob využití těchto systémů v počítačové grafice byl poté popsán v [19].

L-systémy jsou velmi podobné deterministickým bezkontextovým gramatikám, kde je definována abeceda, počáteční stav a skupina pravidel pro přepisování jednotlivých symbolů abecedy na jiný symbol či skupinu symbolů. Pomocí těchto přepisovacích pravidel se z počátečního stavu vygeneruje řetězec znaků, které reprezentují generovaný model. Při vykreslení se využívá abstrakce „želvy“, která má definované souřadnice a úhel směru. Tato želva má na vstupu vygenerovaný řetězec znaků a určité znaky využívá jako příkazy pro pohyb, rotaci a pro kreslení úseček. Tímto způsobem lze generovat například rostliny, pro které je počátečním stavem kořen, který má pravidla rozvětvení do stonků, ty se poté mohou dělit na další stonky, listy či květy (ukázka na obrázku 2.4).



(a) Mřížka pružin ve výchozím stavu

(b) Mřížka pružin po proběhlé simulaci

(c) Distribuce objektů zobrazená bez pružin

Obrázek 2.5: Náhodné rozmístění objektů pomocí mřížky pružin [20]

2.2.4 Náhodné rozmísťování objektů

Pro vygenerování náhodných pozic, kam se objekty umístí, lze využít prosté náhodně generované souřadnice, což však může způsobit problémy. Objekty mohou být nepřirozeně blízko sobě či položené na nevhodný terén (např. stromy na příliš strmém skalnatém terénu), a proto se často využívá jiných algoritmů. Podle [20] lze například nejprve položit objekty na čtvercovou mřížku, každý objekt propojit se svými sousedy „pružinami“ s náhodně nastavenou výchozí délkou a simulovat vliv těchto pružin na objekty. Postup tohoto algoritmu je ilustrován na obrázku 2.5. Tento systém ale vyžaduje předem pevně daný počet objektů, které se navzájem ovlivňují, a samotná simulace je výpočetně relativně náročná.

Další z možností je využití lokálních minim či maxim funkce Perlinova šumu. Takto generované pozice nejsou sice ovlivňovány pozicemi okolních prvků přímo, avšak z vlastností Perlinova šumu víme, že díky pravidelnému intervalu jednotlivých gradientů nebude mezi dvěmi celočíselnými body více než jedno lokální maximum a jedno lokální minimum.

2.2.5 Teleologické metody generování

Systémy procedurálního generování popisované v předchozích sekcích jsou založené na snaze vytvořit útvar již existující v přírodě, ale samy o sobě neberou v potaz způsob, jakým jsou tyto útvary v přírodě vytvářeny. Alan H. Barr v textu [21] popsal vlastnosti technik *teleologického modelování*, kde místo

reprezentování pouze současného tvaru objektu je objekt reprezentován jako sada cílů, kterých se snaží dosáhnout, a popis tvaru a chování objektu jako funkce v čase. Poté lze pomocí inverzní dynamiky vypočítat nutné děje, aby byly cíle splněny. Tímto způsobem se tedy zjednodušeně simulují přírodní jevy, které ovlivňují tvorbu reálného terénu. Tyto postupy jsou výpočetně podstatně náročnější, jelikož se nesnaží popisovat pouze stav generovaného objektu, ale i postup, kterým se terén do daného stavu dostal. Jako ukázky algoritmů teleologického modelování lze zmínit algoritmy simulující umělý život, či algoritmy eroze výškové mapy.

2.3 Shrnutí

V této kapitole bylo popsáno využití procedurálního generování pro tvorbu terénu. Nejprve byl definován pojem procedurálního generování, byl popsán terén a některé jeho vlastnosti, které je možno modelovat. Také obsahuje popis několika algoritmů pro generování terénního reliéfu, objektů ležících na něm a způsobů umisťování těchto objektů na daný reliéf.

Kapitola 3

Návrh aplikace

Tato kapitola se zaměřuje na návrh aplikace pro vizualizaci procedurálního terénu pod systémem SAGE2. Nejprve jsou popsány funkční a nefunkční požadavky, poté návrh architektury aplikace, jejího uživatelského prostředí a také modelu, podle kterého bude terén generován.

3.1 Požadavky

Pro aplikaci byly definovány tyto funkční požadavky:

- Aplikace umožní uživateli přibližování a oddalování kamery, její posun a otáčení kolem generovaného terénu.
- Model generuje terénní reliéf s rozličnými biomy, na které umísťuje předem vytvořené objekty.
- Terén generovaný objektem je dynamický – uživateli je umožněno ovlivnit výšku terénu.
- Aplikace uživateli poskytne seznam zajímavých generovaných lokací, které je možno zobrazit.

Nefunkční požadavky byly poté definovány takto:

- Aplikace bude psaná v jazyce JavaScript, jelikož jde o jediný jazyk podporovaný v SAGE2 API.

- Vykreslování bude po 95% času běhu aplikace udržovat počet snímků za vteřinu nad 30 FPS.
- Modularita systému. Aplikace by měla být rozdělená na smysluplné části, aby bylo možné některé části nahradit jinou funkcionalitou – například aby bylo snadno možné změnit vizualizaci z modelu procedurálně generovaného terénu do modelu skutečných satelitních snímků zemského povrchu.
- Optimalizace pro laboratoř SAGElab. Aplikace bude vyvíjena obecně pro jakoukoliv stěnu běžící pod systémem SAGE2 s dostatečným výkonem, avšak speciální péče bude věnována stěně SAGElab, pod kterou bude tato aplikace testována.

3.2 Aplikace

V následující sekci budou popsány technologie využité aplikací, uživatelské rozhraní aplikace a architektura aplikace. Jako název aplikace bylo zvoleno jméno *ProcViewer*.

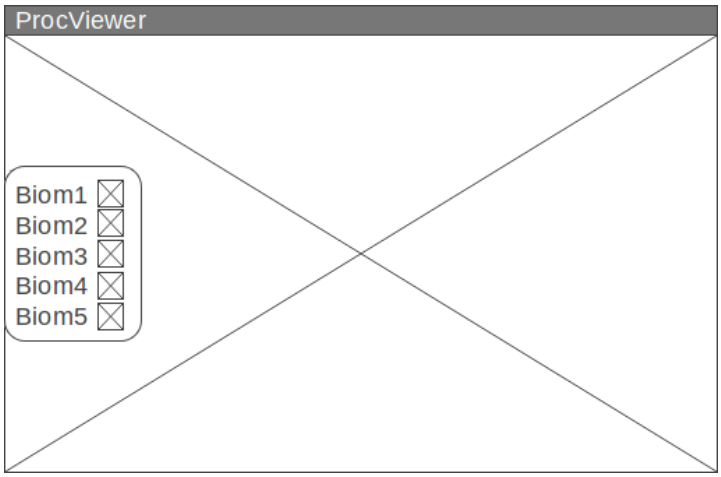
3.2.1 Technologie

Protože aplikace je určena pro systém SAGE2, bude psána v jazyce JavaScript za užití SAGE2 API – vývoj v jiném jazyce systém neumožňuje. Jako systém pro vizualizaci 3D dat byla zvolena knihovna *three.js*, která výrazně zjednodušuje práci s grafickou kartou při zobrazování scény.

Jelikož terén je generován deterministickým způsobem, není nutné, aby mezi sebou jednotliví zobrazovací klienti synchronizovali data o generování. Na každém klientu je terén generován nezávisle na ostatních klientech.

3.2.2 Uživatelské rozhraní a interakce

Prostor okna samotné aplikace bude celý využit pro vizualizaci scény, která se zobrazí po načtení všech objektů nutných pro vykreslení na obrazovku. Část okna vizualizace bude také obsahovat poloprůhlednou legendu popisující jednotlivé vizualizované biomy. Wireframe aplikace je možno vidět na obrázku 3.1.



Obrázek 3.1: Wireframe uživatelského rozhraní vizualizace terénu

Samotná uživatelská interakce bude probíhat SAGE2 kurzorem, který umožní rotaci tažením, přibližování a oddalování kolečkem myši a pohyb po scéně dvojklikem. Dodatečné ovládání nalezne uživatel na klávesnici, kde mu bude umožněn pohyb po scéně šipkami a zapnutí automatického otáčení scény mezerníkem. Klávesami *PageUp* a *PageDown* bude umožněna deformace terénu. V administraci po stisku pravého tlačítka myši najde také uživatel seznam zajímavých lokací vytvořených generátorem.

3.2.3 Architektura aplikace

Samotná aplikace je vícevrstvá, rozdělená do těchto tří vrstev:

- **Vizualizační část**, která se stará o zobrazování scény a uživatelskou interakci. Tato vrstva si vyžádá od generátoru objektů jednotlivé kusy nekonečné scény a bude je zobrazovat. Jediná část aplikace využívající SAGE2 API.

- **Generátor objektů**, který tvoří jednotlivé objekty scény. Vyžádá si od procedurálního generátoru jednotlivé textury určité části nekonečné scény (texturu výškové mapy, povrchu terénu a rozložení porostu), z těchto textur složí výškový reliéf, který zaplní stromy či jiným porostem a přidá hladinu oceánu.

- **Procedurální generátor**, který tvoří procedurální texturu terénu, výškovou mapu a pozice jednotlivých objektů na mapě. Tato data jsou generována na grafické kartě.

Rozdělení aplikace do několika vrstev umožňuje také změnit systém generování terénu s minimálními změnami ve vizualizační části – například místo vizualizace procedurálně generovaného terénu lze vizualizovat satelitní snímky zemského povrchu či jiný předdefinovaný terén.

3.2.3.1 Interface generátoru objektů

Jelikož generátor objektů může být nahrazen procedurálním generátorem či generátorem jiným, je vhodné navrhnout rozhraní, kterým bude vizualizační část s generátorem objektů komunikovat. Proto má generátor pevně dané vlastnosti a metody, které musí splňovat. Interface této třídy je složen z těchto metod a konstantních vlastností:

- `konstruktor(world, chunkSize, resrcPath, renderer)`: Konstruktor generátoru objektů. Parametry popsány níže:
  - `world`: Objekt třídy `map2`, který obsahuje všechny v tuto chvíli vygenerované části světa. Do tohoto objektu se zapisují na souřadnice objekty obsahující vygenerovaný terén včetně dodatečných dat užitečných při zobrazování či zpracování.
  - `chunkSize`: Velikost generovaných částí světa v prostoru.
  - `resrcPath`: Cesta ke kořenové složce aplikace, kde lze najít dodatečné soubory nutné k běhu.
  - `renderer`: WebGLRenderer objekt využívaný k renderování scény. Toto je nutné, pokud generátor generuje textury přímo na grafické kartě místo na CPU.
- `checkReady()`: Funkce pro kontrolu, zda je generátor připravený k generování obsahu. Po samotné inicializaci může docházet k asynchronnímu načítání dat, generátor nemusí být připraven okamžitě.
- `ready`: Boolean hodnota připravenosti, aktualizována pouze po volání `checkReady`.

- `createChunk(x,y)`: Vytvoří část světa na daných souřadnicích.
- `deleteChunk(x,y)`: Smaže danou část světa a dealokuje prostředky této části z grafické karty.
- `deleteAllChunks()`: Smaže celý aktuálně vygenerovaný svět, používané na vyčištění renderované scény.
- `deleteAll()`: Dealokuje veškeré prostředky z grafické karty a ukončí generátor.
- `locations`: Generátor s sebou přináší seznam zajímavých lokací, které jsou uživateli k dispozici, vyžadována pouze hodnota „Default“.
- `zoomMin`, `zoomMax` a `zoomlevel`: Generátor může poskytovat více úrovní přiblížení, tyto hodnoty určují minimum, maximum a současný stav tohoto přiblížení.
- `allowsModification`: Boolean informující o možnostech dynamického upravování terénu
- `drawCircle(position, radius, height, strength = 1)`: Pokud generátor podporuje úpravy terénu, zvýší výšku terénu v kruhu na dané pozici s poloměrem radius o danou hodnotu.
- `deleteModifications()`: Metoda pro smazání všech modifikací terénu.
- `initOverlay()`: Generátor objektů může vyžadovat zobrazení určitých dat jako vrstvu zobrazenou přes samotnou vizualizaci terénu, například pro zobrazení zdroje dat či pro zobrazení doplňkové legendy k mapovým datům. Tato funkce vytvoří scénu, která se bude vykreslovat jako horní vrstva.
- `resizeOverlay()`: Volaný při změně velikosti okna, upraví obsah overlay vrstvy generátoru.
- `overlayScene` a `overlayCamera`: *three.js* scéna a kamera pro vykreslení overlay vrstvy.

Je třeba si povšimnout, že ne všechny vlastnosti generátoru musí být nutně implementovány, aby byl generátor validní. Například místo více úrovní

přiblížení stačí nastavit všechny tři vlastnosti v generátoru na stejnou hodnotu. Stejně tak dynamický terén nemusí být implementován, stačí, pokud je hodnota `allowsModification` nastavena na `false`. Implementace generátoru v této práci se nebude soustředit na více úrovní přiblížení, zato implementuje způsob dynamické modifikace terénu.

3.3 Generování terénu

Jako základ pro generování terénu byl zvolena metoda generování pomocí šumu. Tento způsob byl vybrán z důvodu jeho determinismu. Scéna bude vždy generována stejně, a proto při manipulaci s terénem je možné starý terén odstranit z paměti a případně ho vygenerovat znovu s jistotou, že bude stejný.

Pro samotné generování šumu bude využit algoritmus vylepšeného Perlinova šumu implementovaný v GLSL, popsaný Ianem McEwanem v [22]. Tento algoritmus byl zvolen, jelikož se jedná o relativně jednoduchou a výpočetně nenáročnou implementaci šumu a oproti funkci *noise* v jazyce GLSL bude mít vždy stejný výstup nezávisle na hardware.

Pro generování terénu se nejprve použije šum s velmi nízkou frekvencí. Nízké hodnoty budou indikovat oceán, vysoké hodnoty horské oblasti a ostatní hodnoty budou určovat nížinu, v které se nacházejí ostatní biomy. Do těchto tří typů terénu, mezi kterými se hodnota interpoluje, je dělen samotný výškový reliéf. V tomto kroce je interpolace nutná, aby se zabránilo ostrým hranám ve výšce terénu.

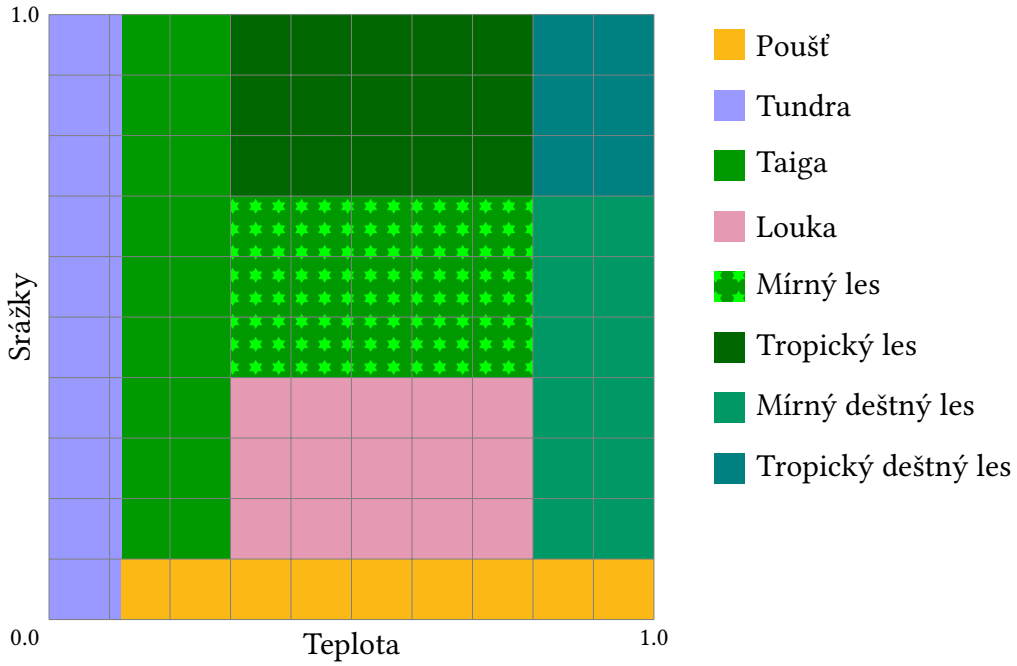
Tyto tři typy reliéfu jsou definovány takto:

- Oceán je definován s pevnou výškou nula.
- Nížiny jsou definovány jako Perlinův šum s několika oktávami.
- Horské oblasti jsou Perlinův šum upravený pomocí absolutní hodnoty, čímž se ve výstupu vytvoří přirozeně vypadající horské hřebeny.

Pro generování biomů v nížinách bude využit systém podobný skutečnému rozdělení biomů popsanému Robertem Whittakerem (popsáno v 2.1). Pomocí šumu bude generována mapa srážek a mapa průměrných teplot, a poté se podle

tabulky známých biomů určí, do kterého biomu terén patří. Pro zjednodušení je Whittakerova tabulka biomů, která je původně trojúhelníková (znázorněna na obrázku 2.1), upravena tak, aby zabírala celý čtverec hodnot mezi nulou a jedničkou. Výsledná tabulka je na obrázku 3.2.

Samotná textura povrchu se generuje velmi podobně jako výšková mapa: nejprve se rozdělí svět do oceánů, nížin a horských oblastí, poté se interpolují barvy definované pro tyto tři oblasti. Oceán má předdefinovanou pískovou barvu (jedná se o dno oceánu, hladina bude reprezentována dodatečným objektem), hory mají barvu šedého kamene, který je od určité výšky pokryt sněhem. Pro nížiny se barva odvodí z hodnoty srážek a teploty pro dané místo. Tyto hodnoty se použijí jako souřadnice v textuře, která určuje barvu povrchu. Samotná textura je velmi podobná tabulce v obrázku 3.2, ale reprezentování barev v textuře umožňuje snadné upravování barevných přechodů mezi biomy a dokonce umožňuje více barev povrchu uvnitř jednoho biomu (například jiná barva písku podle průměrné teploty pouště).



Obrázek 3.2: Návrh rozložení biomů v závislosti na teplotě a srážkách

Tabulka 3.1: Seznam generovaných biomů a frekvence porostu

Id	Název	Porost	Frekvence
0	Oceán	není	není
1	Hory	mrtvý strom	6
2	Poušť	kaktus	4
3	Tundra	borovice	10
4	Taiga	smrk	35
5	Louka	bříza	17
6	Mírný les	buk	37
7	Tropický les	stromové patro	40
8	Mírný deštný les	sekvoj	39
9	Tropický deštný les	husté stromové patro	42

Generování pozic stromů a jiných objektů bude provedeno pomocí Perlinova šumu. Objekty jsou umisťovány na lokální maxima funkce šumu, kde každý biom má svou funkci s odlišnou frekvencí. Tímto lze zajistit to, že např. na poušti se bude nacházet menší množství kaktusů, než kolik je stromů v tropickém deštném lese. Tyto pozice budou vykreslovány na grafické kartě jako obrázek s černou hodnotou pro místa bez porostu a různě barevnými pixely podle typu porostu na dané pozici. Tento obrázek bude zpracováván v JavaScriptu. Rozlišení obrázku může být výrazně nižší než rozlišení samotné textury povrchu, což výrazně zrychlí proces umisťování objektů.

Návrh zahrnuje generování biomů, ke každému biomu náleží jeden druh stromu s určitou frekvencí. Seznam generovaných biomů včetně stromů a frekvencí porostu lze najít v tabulce 3.1.

Samotné 3D modely stromů náležících k biomům budou konvertovány do `json`, který je podporován knihovnou *three.js*. Tato knihovna sice podporuje i další formáty pomocí různých rozšíření, v minimální distribuci však umožňuje import pouze tohoto formátu, do kterého lze modely exportovat pomocí pluginu do open source grafického programu Blender.

3.3.1 Dynamičnost terénu

Dynamický terén je taková reprezentace terénu, která umožňuje jeho změnu za běhu aplikace. Uživateli tedy bude za běhu umožněno modifikovat výšku

terénu způsobem popsaným v sekci 3.2.2. Pro řešení tohoto problému bude využito multitexturování – využití několika textur na jednom materiálu.

V tomto případě bude stále využita pro výškovou mapu procedurálně generovaná textura ve vysokém rozlišení, která bude uložena pouze na grafické kartě, ale přidá se textura nižšího rozlišení, která bude sdílená mezi pamětí počítače a grafické karty. Tuto texturu bude uživatel moci upravovat a změny se projeví na výsledné scéně. Kvůli menšímu rozlišení této textury také není problém danou texturu mít uloženou i poté, co část scény, kterou textura reprezentuje, není zobrazována.

3.4 Shrnutí

V této kapitole byla navržen prototyp třívrstvé aplikace pro vizualizaci procedurálně generovaného terénu. Byla navržena vrstva vizualizační, která se stará o zobrazení terénu, vrstva procedurálního generátoru, která generuje textury terénu a vrstva generátoru objektů, který z vygenerovaných textur tvoří objekt, který je předán na vizualizaci. Byl popsán interface mezi vizualizační částí a generátorem objektů, který může být využit i jinými implementacemi pro vizualizaci jiných vstupních dat. Byla také detailněji popsána pravidla generování samotného terénu včetně biomů, které se v něm mohou nacházet.

Kapitola 4

Realizace

Následující kapitola se věnuje implementaci aplikace podle návrhu a je rozdělena do dvou sekcí – sekce popisující implementaci jednotlivých vrstev aplikace dle návrhu a sekce popisující jednotlivé grafické prvky využívané v aplikaci včetně ukázek těchto grafických prvků.

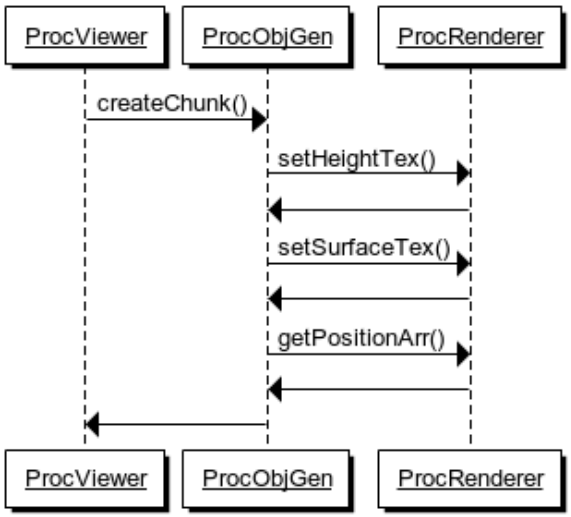
4.1 Implementace jednotlivých vrstev

V této sekci budou popsány jednotlivé implementace vrstev z návrhu v předcházející kapitole a poznatky, ke kterým se při implementaci došlo. Aplikace je podle návrhu složená ze tří vrstev, které mezi sebou interagují při generování jednotlivých částí světa. Na obrázku 4.1 lze nalézt sekvenční diagram průběhu generování jedné části světa.

4.1.1 Vizualizační část

Hlavní částí aplikace je samotná vizualizační část. Jedná se o třídu `ProcViewer` (pojmenovaná stejně jako samotná aplikace), rozšiřující základní třídu `SAGE2_App`. Stará se o zobrazování grafické části aplikace a o uživatelskou interakci.

Three.js scéna, která je zobrazována touto částí aplikace obsahuje pod sebou několik objektů: kameru, směrové světlo, mřížku umožňující pohyb po scéně a skupinu `this.group`, pod kterou se generují jednotlivé části nekonečného



Obrázek 4.1: Sekvenční diagram generování části světa

terénu. S touto skupinou objektů je možno manipulovat, čímž se docílí pohybu kamery po scéně.

Pro ovládání kamery se využívá *three.js* knihovna *OrbitControls*, která umožňuje kamerou otáčet kolem středu scény a kameru přibližovat či oddalovat. Tato knihovna také umožňuje nastavit jednoduché limity, aby například kamera nemohla zajet příliš hluboko pod horizont scény nebo aby se kamera příliš neoddálila.

Pro reprezentaci nekonečného světa interně sdílí vizualizační část s generátorem objektů dvojrozměrnou mapu[1] `this.world`, do které se ukládají jednotlivé vygenerované části světa. Před každým vykreslením scény se zavolá funkce `this.regenerateChunks`, která zkontroluje, zda je nutné vygenerovat nové části scény (vzdálenost generování je dána hodnotou `this.viewDistance`) a vyžádá si od generátoru objektů jejich vytvoření. Stejně tak zkontroluje, které části scény maximální vzdálenost přesahují a předá tyto části generátoru ke smazání.

[1] Jelikož s JavaScript objektem `Map` není snadné pracovat ve více dimenzích, byla pro tuto reprezentaci využita knihovna `Map2`, kterou lze nalézt na `https://github.com/josephg/map2`

Do scény vložena mřížka, pomocí které lze dvojklikem určit nové umístění kamery. Toto bylo zvoleno, jelikož systém SAGE2 API umožňuje pro interakci s aplikacemi využívat pouze levé tlačítko myši (pravé tlačítko zobrazuje radiální menu a prostřední umožňuje pohyb okna aplikace).

Při testování v průběhu vývoje (popsáno v sekci 5.1) bylo zjištěno, že ačkoliv samotné generování je deterministické a na každém klientovi probíhá stejně, kvůli různému vytížení jednotlivých klientů může nastat desynchronizace vizualizačních částí, kdy na všech klientech není kamera umístěna na stejné místo. Proto klient, který je SAGE2 API označen jako master, v pravidelném intervalu předává pozici své kamery ostatním klientům.

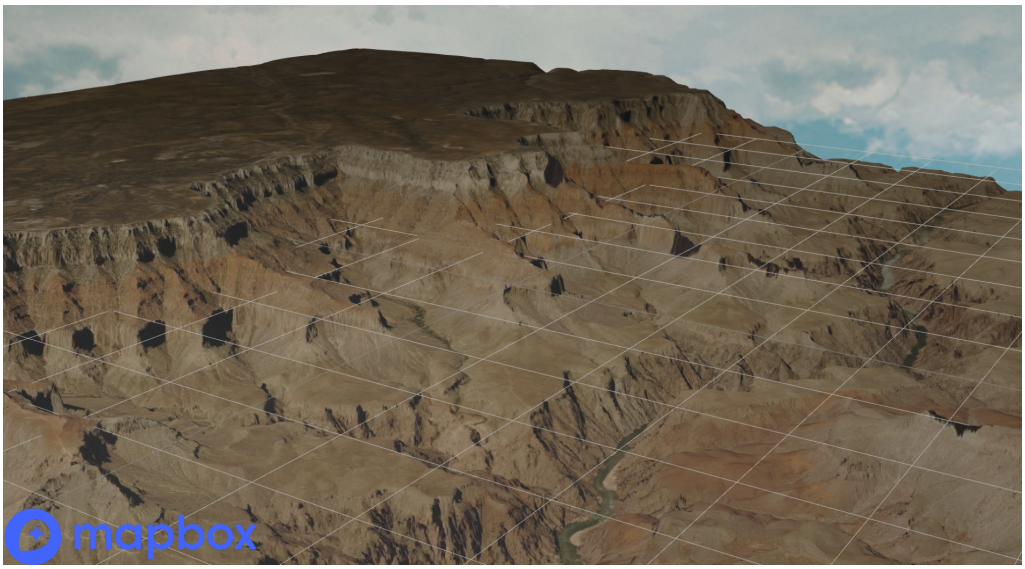
4.1.2 Generátor objektů

Generátor objektů byl implementován třídou `ProcObjGen` a skládá z jednotlivých již existujících zdrojů (vygenerovaných textur, modelů stromů) části scény. Stará se o veškeré načtené i generované modely ve scéně: vytváří je, určuje jejich správnou pozici, případně je dealokuje z paměti grafické karty.

Při implementaci vizualizační vrstvy byl vytvořen také menší ukázkový generátor objektů, který místo dat získaných pomocí procedurálního generátoru získává skutečná zemská výšková a satelitní data ze služby MapBox[2]. Při této implementaci bylo zjištěno, že textury výškové mapy samotného terénu musí mít okraj na sousedících částech terénu stejný. V případě, že se okraj výškových map liší, může dojít na kraji mezi těmito dvěma texturami k změně výšky, což způsobuje viditelnou „trhlinu“ v terénu. Pro data z třetích stran je tedy nutné tyto textury předzpracovat, ale při generování vlastních dat na toto lze myslet již při vývoji. Ukázku vizualizace dat pomocí tohoto generátoru objektů lze najít na obrázku 4.2.

Samotná implementace generátoru pracujícího s procedurálním terénem žádá procedurální generátor o texturu povrchu (*color map*) a výškovou texturu (*displacement map*), ze kterých poté vytvoří `THREE.Mesh` objekt, reprezentující objekt samotného terénu. Jelikož ale dle návrhu tato implementace generátoru používá displacement mapy dvě (procedurálně generovanou a uživatelem modifikovatelnou), je nutné upravit shader, který se pro renderování využívá.

[2] `https://www.mapbox.com/`



Obrázek 4.2: Ukázka prototypu vizualizace pomocí generátoru objektů se zdrojem skutečných dat ze služby MapBox

```
material.onBeforeCompile = function ( shader ) {
  shader.uniforms.tex_modify = {value: modifyTexture};
  shader.vertexShader = "uniform sampler2D tex_modify;\n" +
                        shader.vertexShader;
  shader.vertexShader = shader.vertexShader.replace(
    `#include <displacementmap_vertex>`,
    shaders.modifyTextureDisplacement
   );
  materialShader = shader;
};
```

Obrázek 4.3: Úprava výchozího *three.js* materiálu vlastním GLSL kódem

Tohoto lze v knihovně *three.js* dosáhnout úpravou materiálu využívaného objektem. V tomto případě chceme přidat do shaderu nový uniform a upravit, jakým způsobem se provádí displacement mapping. Ukázku tohoto řešení může čtenář nalézt na obrázku 4.3.

Pro samotné kladení objektů na povrch vyžádá generátor objektů od procedurálního generátoru texturu porostu, která je načtena do `Uint8Array` pro optimalizaci iterace přes pole. Poté se projde celé toto pole a na místa, která nemají nulovou hodnotu, se vytvoří objekty stromů. Protože v JavaScriptu nemá aplikace přímý přístup do výškové mapy, nelze nastavit výšku stromů napevno.

Obrázek 4.4: Šum generovaný algoritmem vylepšeného Perlinova šumu

Je tedy upraven shader materiálu stromů, a to tak, že načítá z výškové mapy vertikální posun, který se na každém bodu modelu provede.

Jako poslední krok se vypočítá výška terénu uprostřed části, která se generuje. Ta je poté uloženo společně s vygenerovaným objektem pro využití vizualizační částí aplikace. Tato výška je nutná, aby vizualizace stále ukazovala povrch terénu i v případech, že jsou generovány vysoké hory.

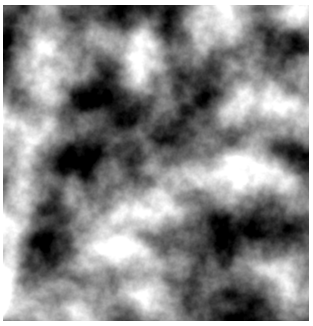
4.1.3 Procedurální generátor

Procedurální generátor je realizován třídou `ProcRenderer`, která pomocí grafické karty vykresluje textury tvořené metodou Perlinova šumu. Tyto textury jsou pomocí metody *Render To Texture* vykreslovány přímo do paměti grafické karty, čímž se podstatně zrychluje vykreslování daného obsahu.[3] Jelikož WebGL neumožňuje sdílení textur mezi několika vykreslovacími kontexty, je nutné procedurálnímu generátoru předat v konstruktoru renderer, který se využívá pro vykreslování samotné scény.

Pro generování Perlinova šumu se podle návrhu využívá implementace vylepšeného Perlinova šumu v jazyce GLSL vytvořená Ianem McEwanem[22], jejíž výstup je možné vidět na obrázku 4.4. Nutno podotknout, že tato funkce ve výchozím stavu generuje hodnoty mezi -1.0 a 1.0, takže v obrázku je výstup posunutý do prostoru 0.0 až 1.0.

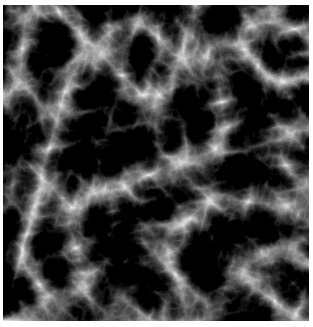
[3] V průběhu implementace procedurálního generátoru nebyla tato metoda nejprve využita, po implementaci se čas samotného generování terénu snížil na 1/40 původního času.

```
float noiseOctaves(vec2 st,float freq){
  float retval = 0.0;
  float amplitude = 1.0;
  for (int i = 0; i < 16; i++){
    retval += snoise(st* freq)*amplitude;
    freq *= 2.0;
    amplitude /= 2.0;
  }
  return retval;
}
```



Obrázek 4.5: GLSL ukázka sčítání několika oktáv Perlinova šumu

```
float noiseMountains(vec2 st, float freq){
  float retval = 0.0;
  float amplitude = 1.0;
  for (int i = 0; i < 16; i++){
    retval += abs(snoise(st* freq)*amplitude);
    freq *= 2.0;
    amplitude /= 2.0;
  }
  return retval;
}
```

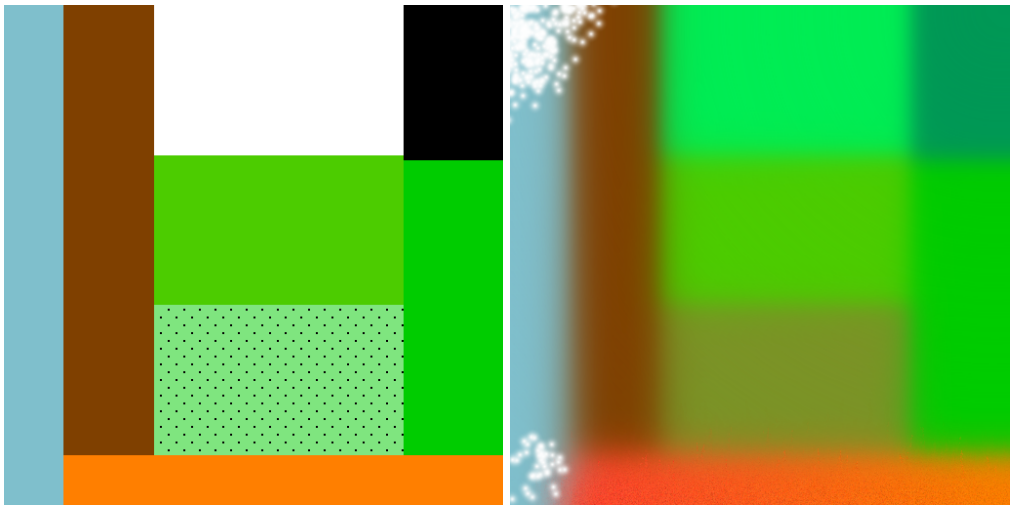


Obrázek 4.6: GLSL ukázka sčítání několika oktáv Perlinova šumu s absolutní hodnotou

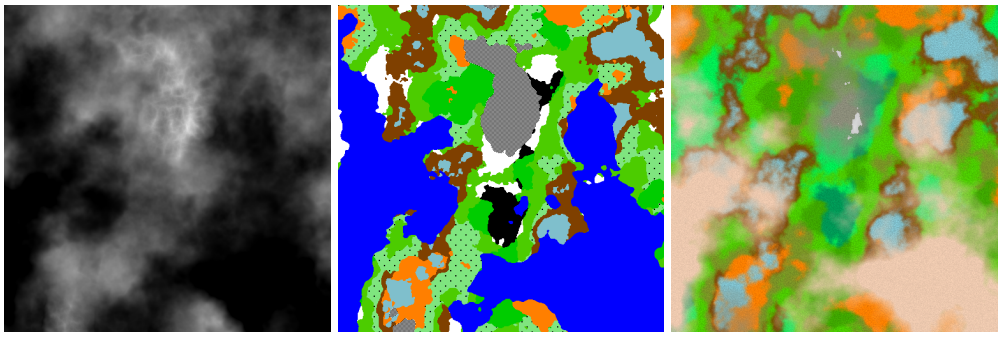
Sčítáním několika těchto šumů s postupně zvyšující se frekvencí a snižující se amplitudou vznikne soběpodobný šum. Pro samotné generování tohoto šumu s více oktávami (kroky s různou frekvencí a amplitudou) lze využít kód v obrázku 4.5.

Pro tvorbu reliéfu horského terénu s ostrými hřebeny je poté nutno v každé oktávě využít funkci absolutní hodnoty. Tímto vznikne terén viditelný na obrázku 4.6. Ve výchozím stavu je však ostrá hrana na hodnotách velmi blízkých nule, výstup byl tedy upraven, aby se netvořila hluboká údolí, ale ostré hřebeny, pomocí vzorce `color = vec3(-noiseMountains(st,1.)+1.);`,

Pro generování celkové výškové mapy je tedy nejprve využit šum s několika oktávami a nízkou frekvencí, pomocí kterého se interpoluje mezi třemi různými funkcemi pro generování oceánu, nížin a horské oblasti. Pro nížiny se poté vygeneruje šum reprezentující srážky a šum reprezentující teplotu dané oblasti. Tyto dvě hodnoty se mohou využít při rozdělování terénu do jednotlivých

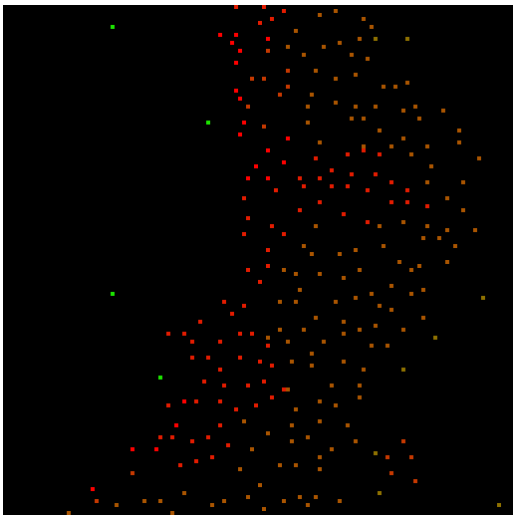


Obrázek 4.7: Rozdělení biomů pomocí podmínek a textura pro určování barvy povrchu



Obrázek 4.8: Tři různé reprezentace terénu: Výšková mapa, mapa biomů a povrchová textura

biomů i při určení barvy samotného povrchu. Při rozdělení do biomů jsou využity podmínky definované tabulkou v návrhu a pro určení textury povrchu se tyto hodnoty využijí jako souřadnice v textuře biomů. Na obrázkcích 4.7 vlevo je zobrazeno rozdělení biomů pomocí podmínek, vpravo zase texturu pro určování barvy terénu. V obou dvou případech jde o tabulku, která reflektuje obrázek 3.2 z návrhu: čím výše určitý pixel je, tím je počet srážek na daném místě vyšší, a čím více vpravo daný pixel je, tím je průměrná teplota na daném místě vyšší. Ukázkový výstup těchto tří generovaných textur terénu lze najít na obrázcích 4.8.



Obrázek 4.9: Výstup generátoru pozic

Pro generování pozic je využita menší textura, jelikož výstup bude poté zpracováván sekvenčně JavaScriptem v samotném generátoru objektů. Využije se opět Perlinův šum, tentokrát však bez oktáv. Pro každý biom je dle návrhu využit Perlinův šum s frekvencí dle návrhu a do lokálních maxim se uloží typ objektu, který se na daném místě nachází. Ukázku znázorňující možné generované pozice naleznete na obrázku 4.9. Barvy jednotlivých pixelů na obrázku poté značí různé grafické prvky, které jsou na dané místo určeny.

4.2 Grafické prvky

Při tvorbě projektu bylo nutno vytvořit větší množství 3D modelů, konkrétně modely pro stromy rostoucí v jednotlivých biomech. Tyto grafické prvky byly vytvořeny specificky pro využití touto implementací vizualizace procedurálně generované scény a jsou zobrazeny na obrázku 4.10.

Dalším grafickým prvkem je textura oblohy, k čemuž byla využita textura mraků vytvořená Pieterem Verhoevenem, který tuto texturu uvolnil pod licencí Creative Commons CC-BY 3.0 umožňující užití za podmínky uvedení autora.[4]

[4] http://creativecommons.org/licenses/by/3.0/

(a) Mrtvý strom (hory) **(b)** Kaktus (poušt) **(c)** Borovice (tundra)

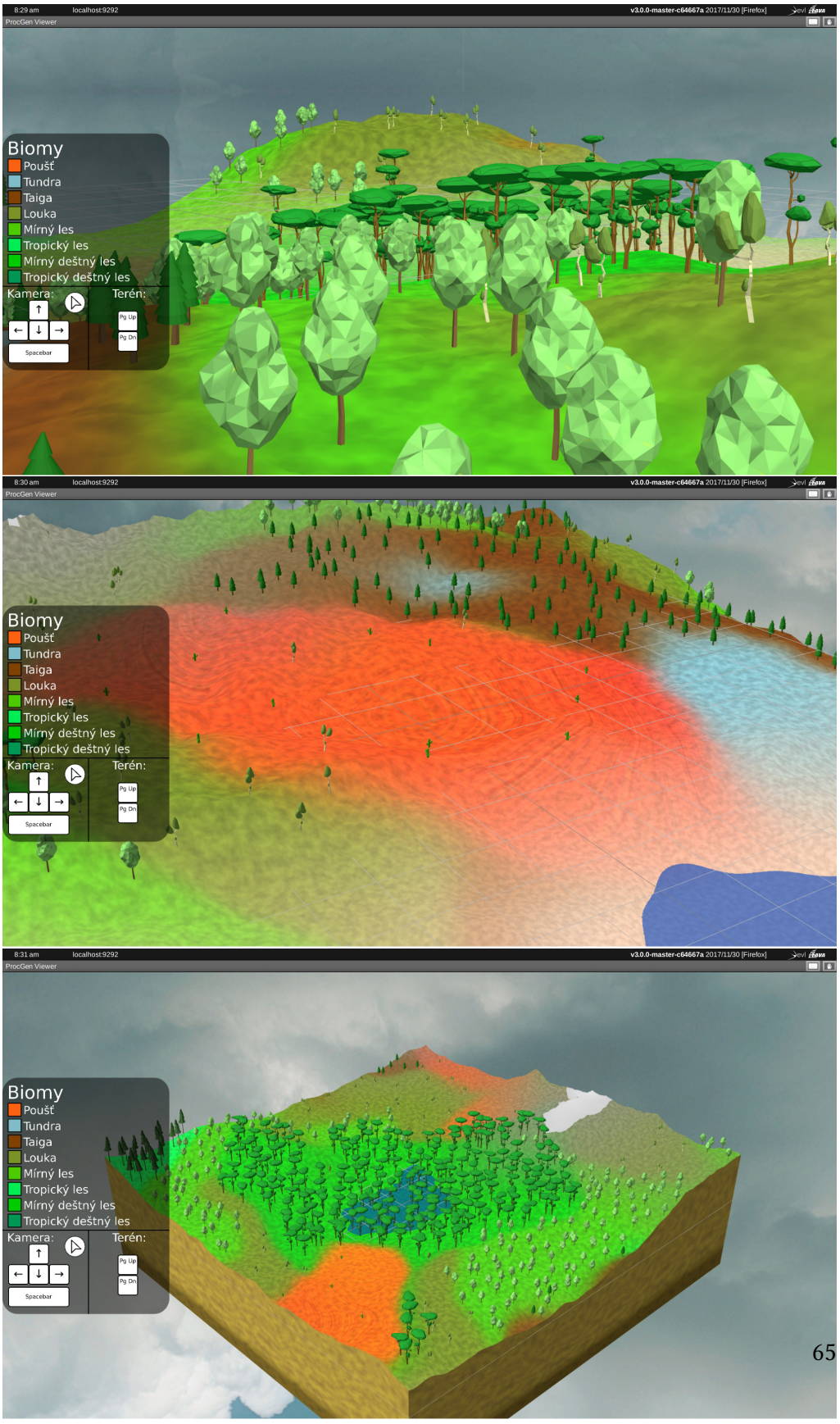
(d) Smrk (taiga) **(e)** Bříza (louka) **(f)** Buk (mírný les)

(g) Stromové Patro (tropický les) **(h)** Sekvoj (mírný deštný les) **(i)** Husté stromové patro (tropický deštný les)

Obrázek 4.10: Různé grafické prvky aplikace

4.3 Shrnutí

Tato kapitola popisuje konkrétní implementaci tří jednotlivých vrstev dle návrhu a ukazuje grafické prvky vytvořené pro danou implementaci. Na obrázcích 4.11 je možno vidět ukázky terénu generovaného touto aplikací.



Obrázek 4.11: Ukázky výsledné implementace prototypu dle návrhu

Kapitola 5

Testování

Důležitou částí vývoje aplikace je kromě samotného návrhu a implementace také testování funkcionality a použitelnosti. Implementovanému prototypu aplikace byla testována funkcionalita během vývoje jak lokálně na počítači, na kterém byla samotná aplikace vyvíjena, tak v laboratoři SAGElab, kde bylo kromě testování funkcionality provedeno také uživatelské testování použitelnosti.

5.1 Testování v průběhu vývoje

V průběhu vývoje byla testována funkcionalita jednotlivých vrstev aplikace. Testování vizualizační části bylo provedeno společně s implementací ukázkového generátoru reálných dat ze služby MapBox. Při tomto testování bylo zjištěno, že blokující generování objektů ve vlákně, ve kterém se obsluhuje uživatelský vstup, způsobuje, že může docházet k desynchronizaci pozice kamery mezi jednotlivými zobrazovacími klienty. Řešení tohoto problému je poté popsáno v sekci 4.1.1.

Pro testování procedurálního generátoru byla vytvořen jednoduchá `html` stránka, která vizualizuje generované textury. Vrstva generátoru objektů, která je položená mezi vizualizační část a procedurální generátor, byla testována jako poslední v celkovém integračním testování, jelikož funkcionalita této vrstvy je zcela závislá na výstupu z procedurálního generátoru a na vizualizaci pomocí vizualizační části.

5.2 Uživatelské testování

Pohled vývojáře na interakci s implementovanou aplikací se může výrazně lišit od skutečných zkušeností, které si uživatelé od interakce s aplikací odnesou. Uživatelské testování použitelnosti se zaměřuje na zkušenosti samotných uživatelů pracujících s testovanou aplikací, a proto poskytuje přehled problémů, s kterými se uživatelé v aplikaci mohou setkat.

Laboratoř SAGElab také mimo spolupracuje s laboratoří pro testování použitelnosti, kde bylo možno provést testování použitelnosti prototypu aplikace. Tato laboratoř je rozdělena do dvou místností: menší je určena jako prostor pro testování a je vybavena počítačem, na kterém běží testovaná aplikace, a systémem kamer a nástrojů, které umožňují testování zaznamenávat aniž by narušili pohodlí testera. Větší místnost je poté určena pro pozorovatele, kteří mohou na systému SAGE2 samotné testování sledovat. Jelikož se systém SAGE2 nachází pouze v místnosti pro pozorovatele, testování bylo provedeno pouze v této místnosti.

Dle [23] je uživatelské testování popsatelné těmito pěti charakteristikami:

1. Hlavním cílem je vylepšit použitelnost produktu.
2. Participující testeři reprezentují skutečné uživatele.
3. Participanti konají reálnou činnost.
4. Pozorovatel sleduje a zaznamenává to, co participanti dělají a říkají.
5. Pozorovatel poté analyzuje data, diagnostikuje reálné problémy a navrhne změny řešící tyto problémy.

5.2.1 Příprava k testování

Před samotným testováním byl autorem připraven scénář, který bude vykonáván participujícími testery, a dotazník, zaměřený na subjektivní pocit z ovládání vizualizace a vzhledu generovaného terénu. Byly také stanoveny uživatelské persony – jelikož užití aplikace není nijak specializované, jako personu je možno brát jakéhokoliv uživatele systému SAGE2 bez žádných omezení.

Dále byl proveden heuristický průchod založený na deseti obecných principech Jacoba Nielsena pro design interakcí, které jsou popsány v [24]. Tyto principy jsou shrnuty níže:

- viditelnost stavu systému,
- shoda mezi systémem a skutečným světem,
- svoboda uživatele a jeho kontrola nad systémem,
- konzistence se standardy,
- prevence chyb,
- rozpoznání místo zapamatování,
- flexibilní a efektivní užití,
- estetický a minimalistický design,
- pomozte uživatelům rozpoznat a diagnostikovat chyby,
- nápověda a dokumentace.

V rámci heuristického průchodu se kontrolují nedostatky všech částí aplikace. Pro roli experta, který tento průchod provádí, byl zvolen student ČVUT FIT, který měl k dispozici testovaný artefakt (prototyp aplikace). Jelikož jediné stavy, které aplikace umožňuje uživateli měnit, je pozice kamery a změna výšky terénu, nalezl expert shodu se skoro všemi výše popsanými principy. Jako jediný princip, který testovaná aplikace nedodržovala v době testování, byla nedostatečná nápověda a dokumentace aplikace, jelikož aplikace neukazovala zřetelně možnost využití kláves *PageUp* a *PageDown* pro změnu výšky terénu. Po tomto průchodu byla tato informace tedy přidána do overlay samotného generátoru objektů.

5.2.2 Průběh uživatelského testování

Uživatelské testování proběhlo 10. května 2018 v laboratoři SAGElab. Testu se zůčastnili tři participanti, všichni studenti FIT ČVUT, a v roli moderátora a pozorovatele byl samotný autor práce. Participanti prošli krátký scénář a byl

jim ponechán volný prostor na prohlédnutí si samotného generovaného terénu. Poté jim byla dána možnost vyplnit krátký dotazník.

Všechny úkoly byly splněny, ale s některou funkcionalitou prototypu měli účastníci testu potíže. Z toho můžeme usoudit, že vytvořený prototyp je zcela použitelný. Ve výsledném dotazníku se participanti testu mohli vyjádřit své subjektivní pocity z grafické stránky samotné aplikace a shodli se na tom, že vizualizace je po grafické stránce pěkná.

Pro funkcionalitu přibližování a oddalování pomocí kolečka myši došli participanti nezávisle na sobě k závěru, že je ovládání prohozené. Stejně tak bylo pro uživatele obtížné přijít na to, že pro změnu posuv kamery je možné využít nejen kurzorové klávesy, ale také dvojklik myši. Tuto funkcionalitu očekávali uživatelé pod tažení pravým tlačítkem myši.

Na základě analýzy interpretovaných dat a doporučení od testerů byly provedeny změny uživatelského rozhraní. Byla upravena funkcionalita kolečka myši a přidáno ovládání posunu kamery pomocí pravého tlačítka myši.

Závěr

Cílem této práce bylo provést analýzu systému SAGE2, jeho API a možností zobrazování procedurálně generovaných scén, a poté navrhnout a implementovat aplikaci vizualizující procedurálně generovanou scénu.

Analýza současného stavu SAGE2 byla provedena společně s analýzou systémů podobného typu v kapitole 1, kde se v sekci 1.4.3 přihlédlo k předchozí práci projektu. Problematika procedurálního zobrazení přírodních scén byla hlouběji analyzována v kapitole 2, kde bylo popsáno několik různých metod pro generování obsahu.

V kapitole 3 je popsán návrh prototypu aplikace pro vizualizaci generovaného terenu, včetně definice pravidel prostředí, podle kterých se scéna rozvíjí. V následující kapitole lze najít samotný popis implementace prototypu aplikace, navržené podle modelu popsaného v návrhu.

V kapitole 5 je popsán postup testování aplikace, kde kromě testování funkcionality bylo také provedeno uživatelské testování v laboratoři SAGElab, ve kterém bylo zjištěno, že aplikace je použitelná a subjektivně se participantům testu líbí.

V budoucnu by bylo vhodné pokračovat ve vývoji a rozšíření modelu generování jednotlivých biomů. Například rozšířit generování na generování samotných objektů, které jsou umisťovány na povrch světa. V současné implementaci jde o předem vytvořené stromy, což by bylo možno nahradit stromy generovanými pomocí L-systémů.

Bibliografie

1. VOMASTEK, Michal. *S10W: Vizualizace plně procedurálního dynamického terénu na velkoplošné obrazovce*. Praha, 2017. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií.
2. KUCHINSKI, Aleh. *10K SAGE2 Dashboard: Sada widgetů pro velkoplošnou obrazovku*. Praha, 2017. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií.
3. MEDUNA, Radek. *10K SAGE2 Dashboard: Sada widgetů pro velkoplošnou obrazovku*. Praha, 2018. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií.
4. EVL. *SAGE2* [online]. 2017 [cit. 2018-04-01]. Dostupné z: `http://sage2.sagecommons.org`.
5. EVL. *SAGE2 Developer Documentation* [online]. 2017-2018 [cit. 2018-04-01]. Dostupné z: `http://sage2.sagecommons.org/download/1309/`.
6. EVL. *SAGE2 Application API* [online]. 2017-2018 [cit. 2018-04-01]. Dostupné z: `https://bitbucket.org/sage2/sage2/wiki/SAGE2%20Application%20API`.
7. SHAKER, Noor; TOGELIUS, Julian; NELSON, Mark J. *Procedural Content Generation in Games (Computational Synthesis and Creative Systems)* [online]. Springer, 2016 [cit. 2018-04-01]. ISBN 3319427148. Dostupné z: `http://pcgbook.com/`.

8. VUGTK. *Terminologický slovník zeměměřičství a katastru nemovitostí* [online] [cit. 2018-04-01]. Dostupné z: http://www.vugtk.cz/slovnik/4403_teren.

9. ANGELIDES, Marios C.; AGIUS, Harry. *Handbook of Digital Games*. Wiley-IEEE Press, 2014. ISBN 1118328035. Dostupné také z: https://www.amazon.com/Handbook-Digital-Games-Marios-Angelides/dp/1118328035?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=1118328035.

10. MARIETTA COLLEGE DEPARTMENT OF BIOLOGY AND ENVIRONMENTAL SCIENCE. *Biomes of the World* [online] [cit. 2018-05-07]. Dostupné z: https://w3.marietta.edu/~biol/biomes/biome_main.htm.

11. PERLIN, Ken. An image synthesizer. *ACM SIGGRAPH Computer Graphics* [online]. 1985 [cit. 2018-05-07]. Dostupné z DOI: 10.1145/325165.325247.

12. LAGAE, A.; LEFEBVRE, S. et al. A Survey of Procedural Noise Functions. *Computer Graphics Forum* [online]. 2010 [cit. 2018-05-07]. Dostupné z DOI: 10.1111/j.1467-8659.2010.01827.x.

13. BEVINS, Jason. *libnoise: Glossary* [online]. 2003-2005 [cit. 2018-04-10]. Dostupné z: http://libnoise.sourceforge.net/glossary/index.html.

14. EBERT, David S. *Texturing & modeling: a procedural approach*. Morgan Kaufmann, 2003.

15. BEVINS, Jason. *Generating coherent noise?* [online]. 2003-2005 [cit. 2018-04-10]. Dostupné z: http://libnoise.sourceforge.net/noisegen/index.html.

16. GUSTAVSON, Stefan. *Simplex noise demystified* [online]. 2005 [cit.]. Dostupné z: http://staffwww.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf.

17. KHRONOS GROUP. *OpenGL® and OpenGL® ES Reference Pages* [online]. 2014 [cit. 2018-05-06]. Dostupné z: https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/noise.xhtml.

18. VIVO, Patricio Gonzalez; LOWE, Jen. *Fractal Brownian Motion* [online]. 2015 [cit. 2018-05-09]. Dostupné z: https://thebookofshaders.com/13/.

19. PRUSINKIEWICZ, Przemyslaw. Graphical applications of L-systems. In: *Proceedings of graphics interface*. 1986, sv. 86, s. 247–253. Č. 86.

20. WEST, Mick. *Random Scattering: Creating Realistic Landscapes* [online]. 2008-08-06 [cit. 2018-04-10]. Dostupné z: https://www.gamasutra.com/view/feature/1648/random_scattering_creating_.php?print=1.

21. BARR, Alan H. Making Them Move. In: BADLER, Norman I.; BARSKY, Brian A.; ZELTZER, David (ed.). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1991, kap. Teleological Modeling, s. 315–321. ISBN 1-55860-106-6. Dostupné také z: http://dl.acm.org/citation.cfm?id=111154.111171.

22. MCEWAN, Ian; SHEETS, David; RICHARDSON, Mark; GUSTAVSON, Stefan. Efficient Computational Noise in GLSL [online]. 2012 [cit. 2018-04-15]. Dostupné z DOI: 10.1080/2151237x.2012.649621.

23. DUMAS, Joseph S; REDISH, Janice. *A practical guide to usability testing* [online]. Intellect books, 1999 [cit. 2018-05-10]. Dostupné z: http://www.jedbrubaker.com/wp-content/uploads/2013/03/Dumas-99.pdf.

24. NIELSEN, Jakob. *10 Usability Heuristics for User Interface Design* [online]. 1995 [cit. 2018-05-10]. Dostupné z: https://www.nngroup.com/articles/ten-usability-heuristics/.

Seznam použitých zkratek

GLSL OpenGL Shading Language

API Application programming interface, aplikační rozhraní

ČVUT České vysoké technické učení v Praze

FIT Fakulta informačních technologií

FPS Frames per second, počet snímků za vteřinu

SAGE Scalable Amplified Group Environment

HTML HyperText Markup Language

UI User interface, uživatelské rozhraní

OS Operační systém

Dodatek A

Uživatelská příručka

Tato příloha popisuje nasazení a provoz aplikace pro vizualizaci procedurálně generované scény. Pro instalaci samotného systému SAGE2 při testování této aplikace byla využita instalace pomocí systému Docker[1], umožňuje automatizaci, kterou se zjednodušuje samotná instalaci a aktualizace systému.

A.1 Nasazení a spuštění aplikace na SAGE2

Pro nasazení aplikace je možno využít předem zabalený soubor `ProcViewer.zip` tímto postupem:

1. Otevřete webový prohlížeč na webové stránce SAGE2 UI (při instalaci pomocí docker se tato stránka nachází na `https://localhost:9090/index.html`) a přihlašte se jako host.

2. Přetáhněte pomocí myši soubor `ProcViewer.zip` z přiloženého nosiče do okna prohlížeče.

3. Resetujte server (při instalaci pomocí docker zavolejte v terminálu `docker stop sage2; docker start sage2;`).

4. Ve webovém prohlížeči klikněte na App Launcher a vyberte aplikaci *ProcViewer*.

[1] `https://bitbucket.org/sage2/sage2/wiki/Install%20(Docker)`

A.2 Ovládání aplikace

V této sekci je popsáno ovládání aplikace pomocí SAGE2 UI klienta pro interakci se stěnou. Aplikace umožňuje následující interakci:

- **Otáčení kamery**: Kamerou lze otáčet pomocí *SAGE2 Pointer* kliknutím a táhnutím.
- **Automatické otáčení kamery**: Pohyb lze zapnou stisknutím klávesy mezerníku.
- **Přibližování a oddalování**: Kameru lze přiblížit či oddálit pomocí SAGE2 Pointer kolečkem myši nebo pomocí nabídky pod pravým tlačítkem myši.
- **Pohyb kamery**: Posouvání kamery po scéně lze docílit pomocí kurzorových kláves či pomocí dvojkliku SAGE2 Pointeru na mřížku ve scéně.
- **Navigace na předem vybraná místa**: V nabídce pravého menu pod SAGE2 UI lze nalézt předem vybraná zajímavá místa ve scéně. Zvolením tohoto místa se kamera na danou pozici přesune.
- **Modifikace terénu**: Pomocí kláves *PageUp* a *PageDown* lze deformovat právě zaměřený terén.

Dodatek B

Podklady pro uživatelské testování použitelnosti

Výstupní dotazník testování prototypu aplikace				
Prosím o vyplnění dotazníku podle vašeho osobního názoru.				
Ovládání				
1	Je otáčení kamery uživatelsky přívětivé?	Ano ☐	Ne ☐	Nevím ☐
2	Je přiblížení kamery uživatelsky přívětivé?	Ano ☐	Ne ☐	Nevím ☐
3	Je posuv scény uživatelsky přívětivý?	Ano ☐	Ne ☐	Nevím ☐
4	Je modifikace terénu uživatelsky přívětivá?	Ano ☐	Ne ☐	Nevím ☐
Grafická stránka				
5	Je terénní reliéf dostatečně rozmanitý?	Ano ☐	Ne ☐	Nevím ☐
6	Jsou jednotlivé biomy rozmanité?	Ano ☐	Ne ☐	Nevím ☐
7	Je rozdělení terénu do biomů smysluplné?	Ano ☐	Ne ☐	Nevím ☐
8	Je vizalizace po grafické stránce pěkná?	Ano ☐	Ne ☐	Nevím ☐

Výsledky výstupního dotazníku

Otázky	Počet odpovědí		
	Ano	Ne	Nevím
Je otáčení kamery uživatelsky přívětivé?	3	0	0
Je přiblížení kamery uživatelsky přívětivé?	0	2	1
Je posuv scény uživatelsky přívětivý?	1	2	0
Je modifikace terénu uživatelsky přívětivá?	2	1	0
Je terénní reliéf dostatečně rozmanitý?	3	0	0
Jsou jednotlivé biomy rozmanité?	2	0	1
Je rozdělení terénu do biomů smysluplné?	1	2	0
Je vizalizace po grafické stránce pěkná?	3	0	0

Dodatek C

Obsah přiloženého nosiče

- `src` ..... Zdrojové kódy prototypů
  - `ProcViewer` .. Zdrojový kód vizualizace procedurálně generované scény
  - `MountainViewer` ..... Zdrojový kód vizualizace reálné scény
  - `ProcViewer.zip` ..... Vizualizace procedurálně generované scény připravená pro nasazení na SAGE2
  - `MountainViewer.zip` Vizualizace reálné scény připravená pro nasazení na SAGE2
- `text` ..... Text práce
  - `thesis` ..... Zdrojové soubory textu pro XeLaTeX
  - `BP_Bubenicek_Tomas_2018.pdf` ..... Text práce ve formátu PDF