VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2020 Zbyněk Drápela

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

INTERAKTIVNÍ WEBOVÉ APLIKACE PRO PODPORU VÝUKY 2D POČÍTAČOVÉ GRAFIKY
WEB APPLICATIONS SUPPORTING EDUCATION OF 2D COMPUTER GRAPHICS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
Zbyněk Drápela

VEDOUCÍ PRÁCE
SUPERVISOR
doc. Mgr. Pavel Rajmic, Ph.D.

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Telekomunikační a informační systémy**

Ústav telekomunikací

Student: Zbyněk Drápela ***ID:*** 195482
Ročník: 3 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Interaktivní webové aplikace pro podporu výuky 2D počítačové grafiky

POKYNY PRO VYPRACOVÁNÍ:

V jazyce JavaScript vytvořte čtyři webové aplikace, které budou sloužit pro interaktivní podporu výuky v oblasti dvojrozměrné počítačové grafiky.
Aplikace budou tématicky zaměřeny na:
1) zvýšení stupně Bézierovy křivky beze změny jejího tvaru,
2) subdivision Bézierovy křivky,
3) Catmull-Clark subdivision,
4) generování procedurálních textur.
Zaměřte se především na názornou podobu a funkčnost pro potřebu výuky. Funkcionalitu všech aplikací podrobně konzultujte s vedoucím práce.

DOPORUČENÁ LITERATURA:

[1] Beneš, B.; Sochor, J.; Felkel, P.; Žára, J.: Moderní počítačová grafika. Computer Press, Brno, 2005.

[2] Piegl, L.,; Tiller, W.: The NURBS Book. Druhé vydání. Springer, 1997

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 8.6.2020

Vedoucí práce: doc. Mgr. Pavel Rajmic, Ph.D.

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

ABSTRAKT

Bakalářská práce se zabývá dvojrozměrnou počítačovou grafikou. Konkrétně Bézierovými křivkami, zvýšením jejich stupně beze změny jejich tvaru a provedení subdivision. Dále pak algoritmem Catmull–Clark subdivision a procedurálními texturami a jejich generováním. Cílem práce bylo popsat teoretický základ těchto oblastí a následně pomocí JavaScriptu a HTML vytvořit čtyři webové applety, které budou sloužit jako pomůcka při výuce počítačové grafiky.

KLÍČOVÁ SLOVA

počítačová 2D grafika, Bézierova křivka, subdivision, Catmull–Clark subdivision, procedurální textura, Perlinův šum, generování textur, applet, JavaScript

ABSTRACT

The bachelor thesis deals with two-dimensional computer graphics. Specifically, Bézier curves, increasing their degree without changing their shape and performing subdivision. Furthermore, the Catmull–Clark subdivision algorithm and synthesis of procedural textures. The work aimed to describe the theoretical basis of these areas and to create four web applets using JavaScript and HTML, which will be used as an aid in teaching computer graphics.

KEYWORDS

2D computer graphics, Bézier curve, subdivision, Catmull–Clark subdivision, procedural texture, Perlin noise, texture synthesis, applet, JavaScript

DRÁPELA, Zbyněk. *Interaktivní webové aplikace pro podporu výuky 2D počítačové grafiky*. Brno, Rok, 40 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: doc. Mgr. Pavel Rajmic, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Interaktivní webové aplikace pro podporu výuky 2D počítačové grafiky“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu doc. Mgr. Pavlovi Rajmicovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Úvod

Tato práce se věnuje dvojrozměrné počítačové grafice, a především Bézierovým křivkám, Catmull–Clark subdivision a texturám. Počítačová grafika je důležitou oblastí informatiky, která používá počítače k tvorbě grafických objektů. V současnosti je velmi důležitá vektorová grafika, která umožňuje tvorbu objektů pomocí vektorů a narozdíl od rastrové grafiky umožňuje tvořit objekty které mohou být zobrazovány v různých velikostech bez ztráty kvality. Typicky se vektorová grafika používá například v počítačových fontech, PC hrách, v reklamních studiích i na webových stránkách.

Cílem této práce je usnadnění výuky dvojrozměrné grafiky pomocí webových aplikací, tzv. appletů[1], které budou sloužit jako pomůcka při výuce a umožní studentům snadnější pochopení této problematiky, a to jak při kontaktní výuce, tak při samostudiu. Applety jsou navrhovány s důrazem na jejich názornost a funkčnost jak na klasických zařízeních typu PC ovládanými myší, tak na zařízeních s dotykovými obrazovkami. Tyto applety jsou zaměřeny na Bézierovy křivky a operace s nimi (zvýšení stupně Bézierovy křivky beze změny jejího tvaru a její subdivision), dále na Catmull–Clark subdivision a poslední z nich na generování procedurálních textur.

Teoretická část se věnuje křivkám, jejich vlastnostem, popisu, modelování a Bézierovým křivkám, jež jsou jedním z významných typů křivek, a dále popisu Catmull–Clark subdivision pro dvourozměrné objekty. Druhá část teorie popisuje textury a funkci generování textur v počítačové grafice. Také jsou zde popsány některé základní funkce pro generování textur a základní textury.

Praktická část se zabývá nejprve volbou softwarového řešení a použitými programovacími jazyky, především s ohledem na jejich aktuálnost a následnou dlouhou životnost appletů. Následně je zde probrána volba vývojového prostředí pro realizaci appletů. Zde je také popsán návrh všech čtyř zpracovávaných appletů včetně vizualizace jejich grafického prostředí. V poslední části se pojednává o implementaci appletů.

[1] Applet je softwarová komponenta, která běží v kontextu jiného programu (typicky webového prohlížeče). Dříve byl applet chápán jako Java applet, ale v dnešní době se více prosazují applety psané v jiných programovacích jazycích, např. v JavaScriptu.

1 Křivky

Křivky se v počítačové grafice a jejich aplikaci používají na mnoha místech. Můžeme se s nimi setkat při modelování ve dvou dimenzích, v počítačových fontech, při určování dráhy v počítačových animacích nebo definici šablonování. Záběr aplikace křivek je velmi široký.

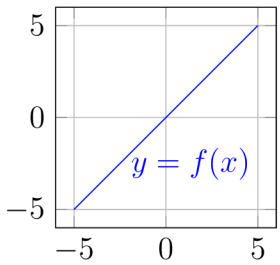
Paul de Casteljau používal od roku 1959 matematický model křivek, který mu umožňoval v jeho práci u firmy Citroën jednoduché vyhodnocení výpočtů. V šedesátých letech vedl Pierre Bézier vývoj systému UNISURF pro návrh křivek a ploch u firmy Renault. Metody pro tvarování křivek se v průběhu času zdokonalovaly a nyní jsou již silným nástrojem, který je stále zdokonalován a rozvíjen. Nachází široké uplatnění v průmyslovém designu [1].

V této kapitole jsou popsány nejčastěji používané křivky používané pro modelování v počítačové grafice.

1.1 Křivky a jejich vlastnosti

Křivky v počítači bývají běžně reprezentovány jako soustava parametrů rovnice a poté je vygenerováno jejich zobrazení.

Křivka může být zadána **explicitně**, například spojitá funkce může být zadána jako $y = f(x)$. Bývá orientována ve směru rostoucího x z definičního oboru a odpovídá jen jedna funkční hodnota y. Tuto křivku můžete vidět na obrázku 1.1.



Obr. 1.1: Explicitně zadaná křivka $y = f(x)$ v rozsahu $\langle -5, 5 \rangle$.

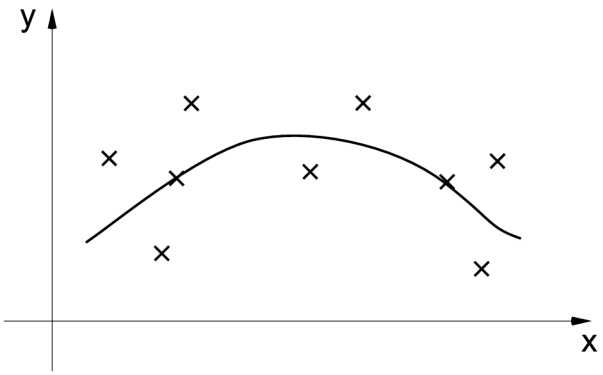
Implicitně může být křivka zadána tvarem $F(x, y) = 0$. Toto je ve srovnání s ostatními možnosti zadání obtížně zobrazitelné, neboť v obecnějších případech neumožňuje postupný výpočet křivky. Má ale svůj význam například při testování oblastí vymezených implicitně zadanou křivkou, nebo při výpočtu průsečíku paprsku s křivkou.

Nejčastěji se však používá tvar **parametrický**, který křivku (pokud ji chápeme fyzikálně) zobrazuje jako dráhu pohybujícího se bodu, jehož souřadnice jsou funkcemi parametru t (času). Můžeme ji například zapsat jako $q(t) = [x(t), y(t)]$ [2]. Parametr t nejčastěji volíme z rozsahu $\langle 0, 1 \rangle$ (počátek a konec časového úseku). Výhodou tohoto zápisu je, že můžeme vyjádřit i průběhy, kdy křivka bodem prochází stejnými body vícekrát (v různých okamžicích), může se křížit, nebo i uzavřít [1].

1.2 Modelování křivek

Základním druhem parametrických křivek, které se používají v počítačové grafice, jsou **polynomiální křivky**. Jejich výhody jsou, že je můžeme snadno vyčíslit a jsou snadno diferencovatelné. Polynomiální křivky můžeme složit z křivek po částech polynomiálních, tedy z jednotlivých segmentů polynomiálních křivek. Nejčastěji se používají křivky třetího stupně, takzvané kubiky. Jejich výpočet nebývá náročný a poskytují dostatečnou škálu tvarů. Je u nich také možné zajist spojitost C^2, kterou často požadují CAD systémy.

Tyto křivky jsou často modelovány tak, že je definováno několik řídicích bodů a matematický aparát z jejich polohy určí průběh křivky. Řídicí body mohou být interpretovány dvěma způsoby – *interpolací* a *aproximací*. Pokud křivku generujeme interpolací, musí probíhat danými body. Při aproximaci je sice její tvar body dán, ale probíhat jimi nemusí, jak je vidět na obrázku 1.2 [1]. Pro nás jsou zajímavější aproximační křivky, o kterých pojednává kapitola 1.3.



Obr. 1.2: Aproximační křivka [1].

Většinou jsou u křivek požadovány tyto vlastnosti:

- Vlastnost konvexní obálky, tedy zda celá křivka leží v konvexní obálce všech svých řídicích bodů, nebo zda část křivky leží v konvexní obálce některých řídicích bodů.

- Lokalita změn – zda se při změně polohy řídicího bodu mění jen část křivky, a ne celá křivka.
- Křivka by měla procházet krajními body svého řídicího polygonu.
- Invariance k lineárním transformacím a projekcím (např. posunutí nebo otočení), jež zaručuje že po transformování řídicího polygonu dosáhneme následným generováním stejného výsledku, jako bychom transformovali každý bod vygenerovaný z křivky [2].

1.3 Aproximační křivky

1.3.1 Bézierovy křivky

Bézierovy křivky jsou nejspíše nejpopulárnějšími aproximačními křivkami. Jméno dostaly po francouzském inženýru Pierru Bézierovi. Používají se pro modelování ve dvou rozměrech, ale také pro definici trojrozměrných objektů. Velmi často se používají k definici písem (fontů).

Obecné Bézierovy křivky n-tého stupně jsou určeny $n+1$ body P_i řídicího polygonu. Jsou definovány vztahem:

$$Q(t) = \sum_{i=0}^{n} P_i B_i^n(t) \tag{1.1}$$

Kde B_i^n jsou Bernsteinovy polynomy n-tého stupně

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}; t \in \langle 0, 1 \rangle; i = 0, 1, \ldots n. \tag{1.2}$$

Při změně polohy řídicího bodu dojde ke změně tvaru celé Bézierovy křivky. Proto se také křivky často dělí na segmenty nižšího stupně, které se postupně navazují [1].

1.3.2 Bézierovy kubiky

Nejčastějším případem Bézierových křivek jsou kubiky. Jedná se o speciální případ Bézierovy křivky, která je dána čtyřmi body. Určuje ji vztah [1]:

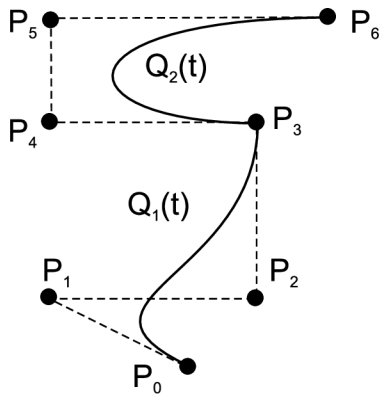
$$Q(t) = \sum_{i=0}^{3} P_i B_i(t) \tag{1.3}$$

Počítačové fonty bývají většinou modelovány právě Bézierovými kubikami [3].

1.3.3 Stupeň Bézierovy křivky

Stupeň Bézierovy křivky je závislý na tom, kolika řídicími body je tvar křivky určen. Křivky n-tého stupně jsou určeny $n+1$ body polygonu P_i. Na obrázku 1.3 je křivka

$Q_1(t)$ určená čtyřmi $(3+1)$ body P_0, P_1, P_2, P_3 je tedy křivka třetího stupně. Druhá křivka $Q_2(t)$ určená body P_3, P_4, P_5, P_6 je taktéž křivka třetího stupně.



Obr. 1.3: Dvě Bézierovy křivky 3. stupně[1].

Pohybem řídicích bodů Bézierovy křivky je možné dosáhnout různých tvarů křivky. Může se ale stát, že není možné docílit požadovaného tvaru křivky s daným počtem řídicích bodů. V tom případě je možné zvýšit stupeň křivky, a přitom zachovat její tvar. Krajní body zůstanou na svých místech a z ostatních řídicích bodů jsou vypočteny nové.

Chceme-li vypočíst lineární (konvexní) kombinaci bodů, tedy z $n+1$ bodů $P_0, P_1, \ldots, P_n$ vypočíst $n+2$ bodů $P'_0, P'_1, \ldots, P'_n, P'_{n+1}$, použijeme k tomu vztah [14]:

$$P'_i = \frac{i}{n+1} P_{i-1} + \frac{n+1-i}{n+1} P_i, \quad i = 0, \ldots, n+1 \tag{1.4}$$

1.3.4 Subdivision pro Bézierovy křivky

Někdy je třeba křivku rozdělit na menší na sobě nezávislé části a při tom zachovat její tvar. Tato operace se nazývá subdivision. Při dělení křivky pomocí subdivision na více menších přibývá počet řídicích bodů, avšak křivka zůstává nezměněná (nebo alespoň velmi podobná původní křivce). Této vlastnosti se používá mimo jiné při vykreslování křivky [2].

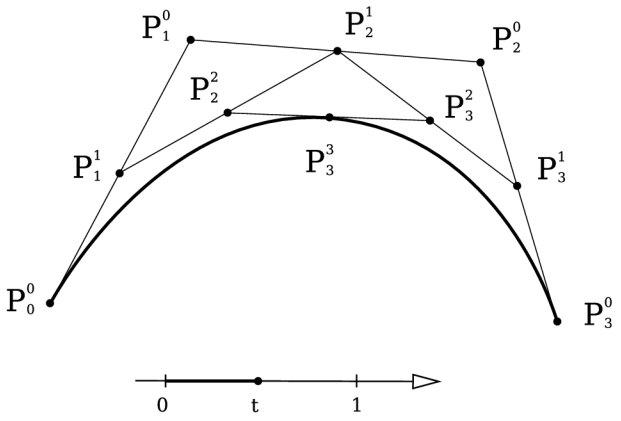
Při subdivision je bod dělení křivky $t = 0{,}5$, tedy polovina jejího průběhu. Pokud se tento bod nachází jinde, nazýváme toto subsection.

Aby bylo tohoto dosaženo, používá se velmi často algoritmu de Casteljau, který můžeme v rekurentním vztahu zapsat takto:

$$P_i^j(t) = (1-t)P_i^{j-1} + tP_{i+1}^{j-1} \tag{1.5}$$

V rovnici 1.5 značí i pořadí bodu v daném cyklu a pro $i = 1, 2, \ldots, n$ a j značí číslo cyklu a nabývá hodnot $j = i, i + 1, \ldots, n$.

Jak je patrné z obrázku 1.4, tento algoritmus spočívá v dělení úseček řídícího polygonu v požadovaném poměru. Při každém dělení se počet řídicích bodů zmenší o jedna. V momentě, kdy nám zůstane jediný bod, jsme dosáhli hledaného bodu na křivce. V případě který je na obrázku je to bod P_3^3.



Obr. 1.4: Hledání bodu na křivce algoritmem de Casteljau.

1.4 Catmull–Clark subdivision

Catmull–Clark subdivision algoritmus je pojmenovaný po svých autorech Edwinovi Catmullovi a Jimovi Clarkovi. Tento algoritmus pracuje s jakýmkoli n-úhelníkem a vždy produkuje čtyřúhelníky.

1.4.1 Popis algoritmu

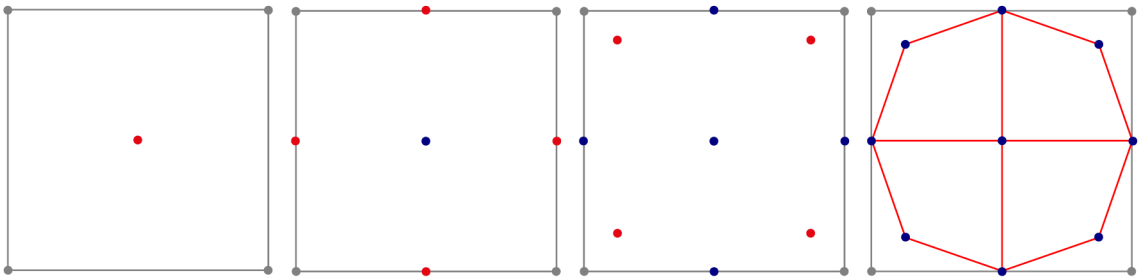
Výpočet tohoto algoritmu je poněkud složitější. Jeho postupné kroky jsou vidět na obrázku 1.5, který znázorňuje jak se algoritmus zachová, pokud ho použijeme na čtverec. Šedá znázorňuje původní tvar a body, červeně jsou body nebo čáry vytvořené při aktuálním kroku algoritmu a modrá znázorňuje předchozí kroky algoritmu, které jsou už součástí jeho výstupu. Pravidla algoritmu jsou:

1. Pro každý polygon v původní mřížce je pomocí aproximace vypočten nový bod, který se nachází ve středu polygonu.
2. Pro každou hranu v původní kontrolní mřížce je vypočten nový bod jako průměr dvou koncových bodů hrany a dvou středových bodů vypočtených v předchozím kroku.

3. Dále jsou staré původní vrcholy posunuty podle vzorce:

$$q = \frac{F}{n} + \frac{2E}{n} + \frac{P(n-3)}{n} \tag{1.6}$$

   kde P je původní bod mřížky, F je průměr všech bodů vypočtených v prvním kroku dotýkajících se P. E je průměr všech n středových bodů hran dotýkajících se bodu P.
4. Nakonec se každý bod z 1. kroku spojí s body vytvořenými ve 2. kroku a každý nový vrchol vytvořený ve 3. kroku se spojí s body z 2. kroku.



Obr. 1.5: Catmull–Clark algoritmus. Grafické znázornění popsaných kroků.

Po prvním dělení jsou již všechny nově vytvořené stěny čtyřúhelníky a počet výjimečných vrcholů nadále zůstává konstantní [12, 13].

1.4.2 Využití algoritmu

Každý, kdo viděl nějaký film z produkce studia Pixar, se nejspíše už setkal s Catmull–Clark algoritmem. Byl totiž použitý při tvorbě takových hitů, jako Hledá se Nemo nebo Úžasňákovi [15].

1.4.3 Autoři algoritmu

První ze spoluautorů toho algoritmu, Edwin Catmull, je spoluzakladatelem společnosti Pixar [15]. James Henry Clark, tedy druhý ze spoluautorů, založil několik technologických firem v Silicon Valley zabývajících se převážně počítačovou 3D grafikou [16].

2 Textury

Textura je velmi důležitá pro vnímání struktury, barvy a kvality objektu. Je to popis vlastností povrchu. Textura je pravidelný nebo nepravidelný vzorek a je spjata s materiálem, který by měl povrch jednoznačně popisovat. Pro zjednodušení mnoha operací se materiál a textura oddělují a aplikují ve dvou krocích [1]. Jednotlivý element struktury se nazývá *texel* (pojem vznikl spojením slov **tex**ture a **el**ement) [2].

Aplikací textury dosáhneme velkého zvýšení vizuální kvality objektu s velmi malými náklady, a proto je textura intenzivně používána typicky v časově kritických aplikacích.

Většinou je efektivnější použít jednoduchou geometrii objektu a složité textury, než definovat složité geometrické detaily. Pokud jsou objekty zobrazeny z velké vzdálenosti, nebo jen krátce, nebývá rozdíl často patrný [1]. Typicky je tento přístup využit v počítačových hrách.

2.1 Rozdělení textur

Textura se snaží popsat mnoho vlastností předmětu. Mezi ně patří barva, množství odraženého světla, průhlednost, zvrásnění a hypertextura (to co je nad povrchem tělesa). Můžeme také kombinovat několik textur na jednom objektu, což se nazývá multitexturing [2].

Textury dělíme podle způsobu jejich definice na *rastrové* a *procedurální* textury. Rastrové textury používají předem připravený rastrový obrázek. Záleží u nich na detailnosti textury. Procedurální textury jsou vyjádřeny pomocí matematické funkce a vždy se tak přizpůsobí velikosti renderovaného obrazu.

Dalším druhem rozdělení je podle počtu dimenzí:

- jednorozměrné – generování opakujících se podélných vzorů, například přerušovaných čar a křivek,
- dvojrozměrné – povrch těles,
- třírozměrné – definují texturu v prostoru a simulují tak materiál tělesa,
- čtyřrozměrné – animace textur [4].

2.2 Procedurální textury

Zavoláním programu procedurální textury získáme hodnotu textury v určeném bodě. Nejsou příliš prostorově náročné a jsou používány především na opakující se vzory, jako je třeba zeď z cihel.

Zajímavé jsou především textury založené na fraktálních šumových funkcích, které syntetizují šum. Ideální funkce pro generování šumu by měla mít frekvenčně neomezené spektrum, jelikož takový šum obsahuje informace na mnoha úrovních a může tak být libovolně zvětšován a zmenšován. Takováto funkce ale bude mít vysoké výpočetní nároky.

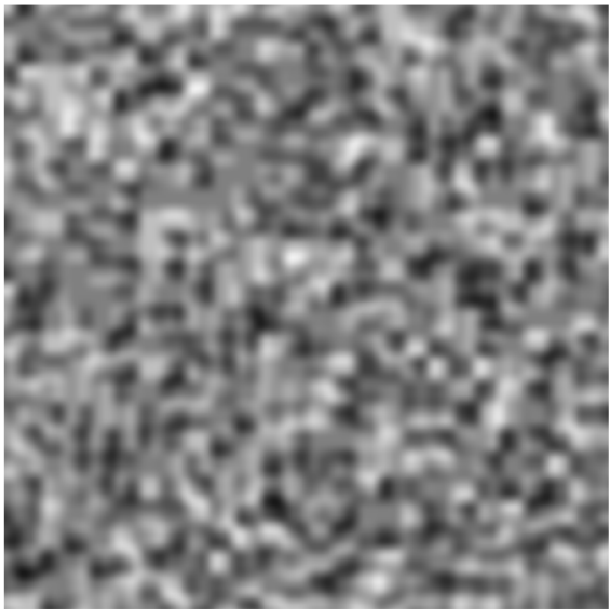
2.2.1 Pseudonáhodnost a opakovatelnost

Základem generování šumu, jenž tvoří základ textur, je pole pseudonáhodných čísel. Toto pole obsahuje předem vygenerovaná čísla, kerá jsou ale vždy stejná. Pokud bychom generovali vždy nové pole, nikdy bychom nedostali texturu, která vypadá tak jak požadujeme.

Díky stálosti výchozího pole čísel můžeme v kombinaci s vhodným výběrem dalších parametrů opakovaně dosahovat stejných výsledných textur. Tedy například cihlová zeď bude vždy vypadat jako cihlová zeď. Tato vlastnost se nazývá opakovatelnost [1].

2.3 Perlinova funkce

Většinou si při generování šumu vystačíme s *Perlinovou funkcí*. Ta je výpočetně rychlá, splňuje požadavky na invariantnost k posunutí a otáčení, je spojitá a opakovatelná. Má také omezené frekvenční spektrum, díky čemuž je výpočet funkce velmi



Obr. 2.1: Šum generovaný Perlinovou funkcí `noise`.

rychlý [4].

Kód funkce `noise` v JavaScriptu můžete vidět níže. Popis jejího kódu naleznete v kapitole 2.5.1 a používané funkce jsou popsány v kapitole 2.5.2. Výstup této šumové funkce můžete vidět na obrázku 2.1.

Výpis 2.1: Perlinova šumová funkce `noise` v JavaScriptu.

```
function noise(x, y, z) {
    let X = Math.floor(x) & 255,
        Y = Math.floor(y) & 255,
        Z~= Math.floor(z) & 255;
    x -= Math.floor(x);
    y -= Math.floor(y);
    z~-= Math.floor(z);
    let u~= fade(x),
        v~= fade(y),
        w = fade(z);
    let A = p[X] + Y, AA = p[A] + Z, AB = p[A + 1] + Z,
        B = p[X + 1] + Y, BA = p[B] + Z, BB = p[B + 1] + Z;

    return scale(lerp(w, lerp(v, lerp(u, grad(p[AA], x, y, z),
        grad(p[BA], x - 1, y, z)),
        lerp(u, grad(p[AB], x, y - 1, z),
            grad(p[BB], x - 1, y - 1, z))),
        lerp(v, lerp(u, grad(p[AA + 1], x, y, z~- 1),
            grad(p[BA + 1], x - 1, y, z~- 1)),
            lerp(u, grad(p[AB + 1], x, y - 1, z~- 1),
            grad(p[BB + 1], x - 1, y - 1, z~- 1)))));
};
```

2.4 Manipulace s Perlinovou funkcí

Perlinova funkce tvoří výpočetní základ pro vytváření textur. Můžeme ale používat různé metody k manipulaci s ní, které nám rozšiřují možnosti použití funkce.

Pokud násobíme argument základní šumové funkce konstantou, můžeme tak regulovat šířku pásma šumu. Používáme tzv. oktávy, tedy konstanty 2, 4, 8, 16, či vyšší.

Dále můžeme pomocí persistence regulovat, jak rychle amplituda klesá mezi každou oktávou. Při nízké persistenci je struktura vygenerovaná z funkce jemnější a při vysoké persistenci má naopak mnohem více detailů.

Také je možné pomocí turbulence tvořit realističtější textury, které připomínají například dřevo nebo mramor [5]. Těmito manipulacemi se podrobněji zabývá kapitola 2.5.3.

2.5 Generování textur

Pokud chceme generovat textury, máme dvě nejzákladnějších možností jak toho docílit. První možností je použití už hotových programů, které nám generování umožní. Většinou je jejich součástí i modelování 3D objektů a nanášení textur na ně. Tyto programy, jejichž zástupcem je například Blender, obsahují i širokou škálu předpřipravených textur, které můžeme použít.

Druhá možnost je o řád složitější. Můžeme využít knihovnu pro tvorbu textur, kterou už někdo vytvořil v našem oblíbeném programovacím jazyce. Nejznámější Perlinův šum napsal právě jeho autor Ken Perlin v jazyce Java. Nyní je však již přeložený nebo přepsaný do mnoha dalších programovacích jazyků [17].

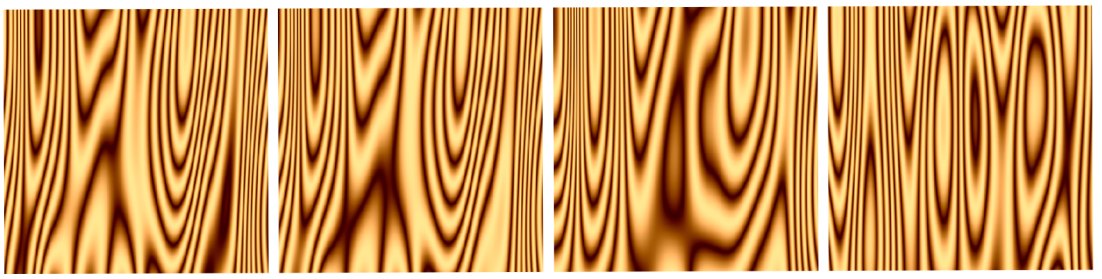
2.5.1 Generování Perlinova šumu ve 2D

Základem generování Perlinova šumu je pole pseudonáhodných čísel, která jsou ale vždy stejná. Využití tohoto pole je diskutováno v kapitole 2.2.1. Toto pole je vhodné inicializovat v programu pouze jednou a následně už vždy používat toto pole. Jinak bychom generování dělali zbytečně náročnější a pomalejší.

Samotný šum je generován funkcí `noise`, jejíž kód můžete vidět zde: 2.1. Funkce má v našem případě tři vstupní parametry x, y, z. Základní funkce pro generování Perlinova šumu totiž generuje 3D šum, ze kterého si ale snadno můžeme vybrat šum ve 2D.

Pomocí prvních dvou proměnných při generování 2D textur určujeme rozměry generovaného obrazu x a y. Pokud jsou tyto rozměry odlišné od rozměru plátna, kde je textura vykreslována, dochází k roztažení či smrštění textury v jednom či druhé směru. Perlinův šum ale můžeme generovat v libovolném počtu dimenzí.

Jelikož nás zajímá pouze 2D šum, třetí parametr z určuje hloubku ve které se nachází pro nás zajímavý obraz. Pokud bychom to přirovnali ke dřevěné kostce, první dva rozměry by určovaly, jak bude kostka vysoká a široká a třetí určí, ve které části ji příčně rozřízneme. Struktura se tedy mění podle umístění tohoto řezu. Při malé změně umístění řezu je struktura podobná, při velké změně bude zcela odlišná, jak je vidět na obrázku 2.2, který znázorňuje generovanou strukturu dřeva pro různé volby parametru z. Tento parametr můžeme jednoduše použít pro doladění struktury, nebo například pro animaci textury.



Obr. 2.2: Struktura dřeva – vlivu parametru z pro hodnoty $0{,}74, 0{,}69, 0{,}52, 0{,}19$.

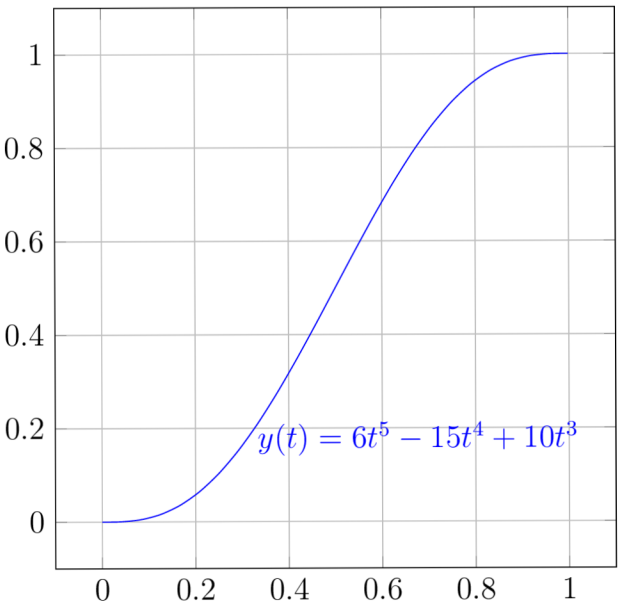
Výstupem z této funkce je desetinné číslo v rozsahu $\langle 0, 1\rangle$. Pokud tedy chceme šum zobrazit například v barevném systému RGB, je nutné provést přepočet. V tomto případě to znamená číslo vynásobit 255 [18].

2.5.2 Základní funkce pro generování šumu

Pro generování šumu je používáno několik funkcí.

Funkce `floor` pro hledání největšího celého čísla které je menší nebo rovno danému desetinnému číslu. Ta se běžně nachází v knihovně `Math`, nebo obdobné (dle programovacího jazyka).

Funkce `fade` počítá křivky slábnutí, kterou můžeme vidět na obrázku 2.3. To zajistí plynulejší a přirozenější přechod mezi jednotlivými body šumu.



Obr. 2.3: Křivka slábnutí.

Pomocí funkce `lerp(a, b, t)` (zkratka z **l**inear int**erp**olation) je prováděna lineární interpolace mezi jednotlivými body pomocí následujícího vzorce. Proměnné a a b jsou hodnoty bodů které jsou interpolovány pro parametr t z rozsahu $\langle 0, 1\rangle$.

$$a + t \cdot (b - a) \tag{2.1}$$

Dále funkce `grad` počítá skalární součin náhodně vybraného gradientního vektoru a osmi vektorů pozice [19].

Běžně nám šumová funkce vrací hodnoty z intervalu $\langle -1, 1\rangle$. Pro použití v grafice je pro nás vhodnější rozsah $\langle 0, 1\rangle$ a k tomu je zde funkce `scale`, která tohoto docílí pomocí vzorce [18]:

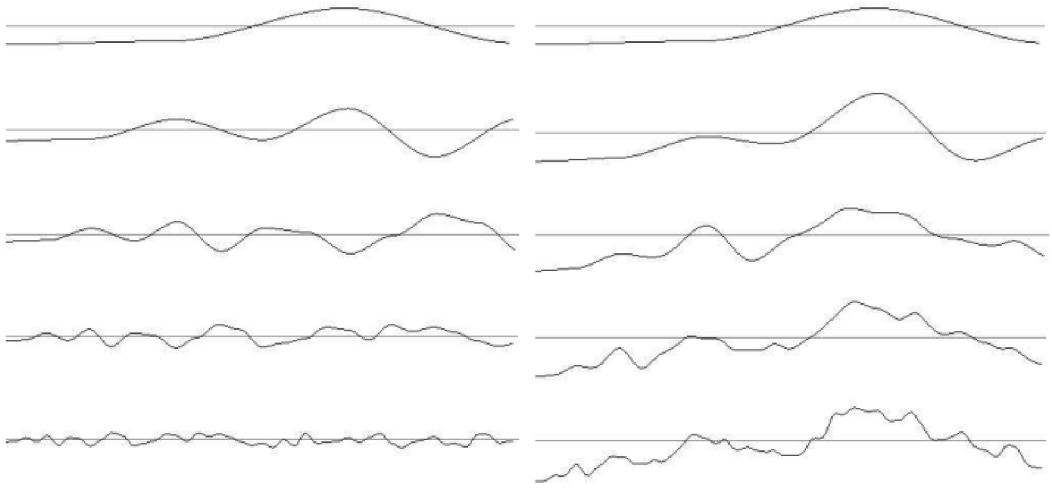
$$\frac{1+n}{2} \tag{2.2}$$

Tento rozsah už můžeme snadno přepočíst na rozsah námi používaného barevného systému, například 8bitový RGB s rozsahem $\langle 0, 255\rangle$.

2.5.3 Skládání šumových funkcí

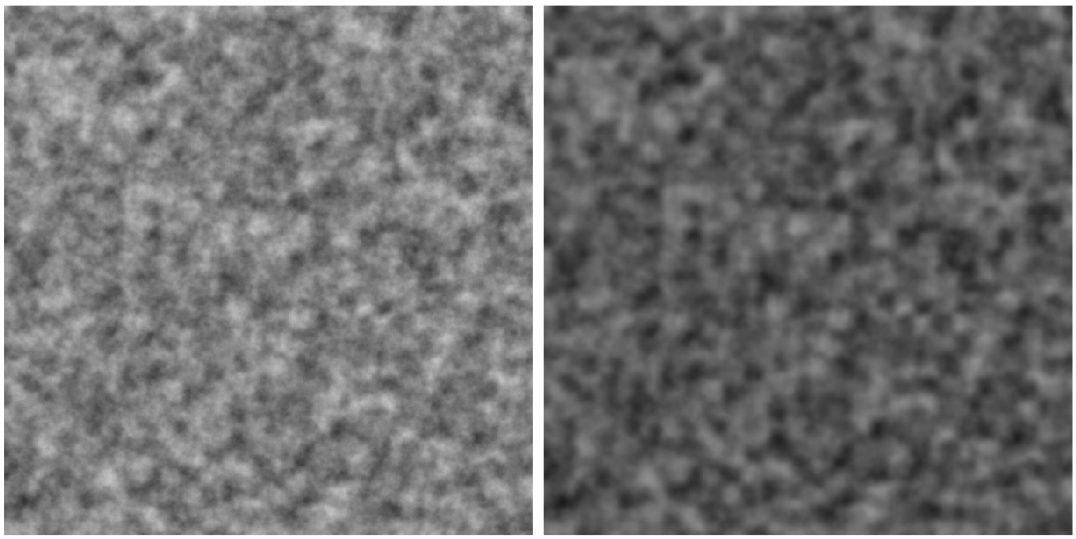
Na funkci Perlinova šumu je založen alrogitmus skládání šumových funkcí. Jedná se o součet šumových funkcí `noise(x, y, z)`, kdy každá má změněnou amplitudu a frekvenci. Příspěvky součtu jsou nazývány oktávy a jako persistence je označována rychlost klesání vlivu na výsledný součet.

Můžeme používat funkci `turbulence(x, y, z, octaves)`, která sečítá šumové funkce dle zvoleného počtu oktáv, nebo také funkci `snoise(x, y, z, persistence, octaves)` která navíc bere v úvahu persistenci. Průběh skládání je vidět na obrázku 2.4, kdy v levém sloupci jsou jednotlivé šumové oktávy a v pravém sloupci



Obr. 2.4: Skládání šumových funkcí [1].

jsou vidět jejich postupné součty. Nejnižší oktávy udávají základní charakter výsledné funkce a ty vyšší jí dodávají detaily [1]. Na obrázku 2.5 je vidět výstup funkcí `turbulence` a `snoise` s nastaveným parametrem persistence na 0,5. S funkcí `snoise` jsme schopní dosahovat stejných výsledků jako s funkcemi `turbulence` nebo `noise`, pokud správně nastavíme parametry.



Obr. 2.5: Textury generované funkcemi `turbulence` a `snoise`.

2.5.4 Funkce pro generování textur

Pomocí výše popsaných funkcí `noise`, `turbulence` a `snoise` můžeme generovat šum a skládat jej. Pro dosahování požadovaných textur je ale nutné s nimi dále pracovat. Níže jsou popsány základní funkce pro jejich generování.

Mramor

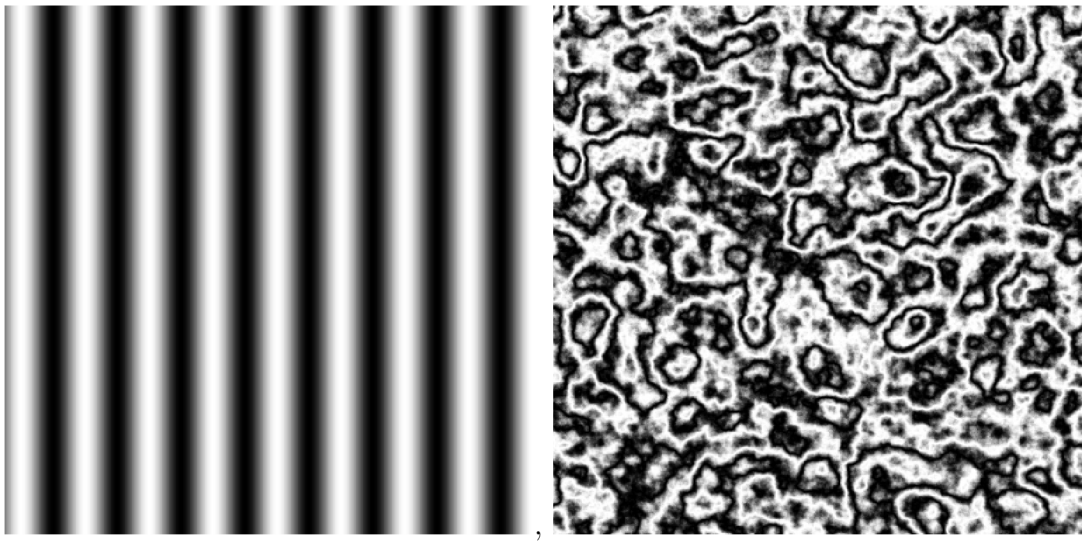
Textura mramoru používá jako základ texturu sinusu, kterou můžete vidět na obrázku 2.6 vlevo. V našem algoritmu je toho docíleno použitím funkce `ramp`, která používá vzorec:

$$\frac{1}{2} \times (1 + \sin x) \tag{2.3}$$

Vzorec funkce `marble` je následující:

$$ramp(influence * snoise(x, y, z, persistence, octaves)) \tag{2.4}$$

Používá skládaný šum `snoise`, který je násobený proměnnou $influence$ pro nastavení síly jejího vlivu. Celá tato konstrukce následně prochází funkcí `ramp`, která jí vloží texturu sinusu [1]. Výsledná textura je vidět na obrázku 2.6 vpravo.



Obr. 2.6: Textura založená na funkci sinus (vlevo), textura mramoru (vpravo).

Dřevo

Textura dřeva je také založena na textuře sinusu. V našem případě je využívána funkce cosinus, což je ovšem rozdíl pouze v posunutí začátku pruhů textury a je jedno, kterou použijeme.

Oproti mramoru je textura dřeva hladší a nechceme do ní vnášet ostrost, kterou přináší skládání šumových funkcí. Proto využíváme jako základ šumovou funkci `noise`, tedy čistý Perlinův šum.

Struktura dřeva by měla být barevná, ideálně v odstínech hnědé. V našem případě tedy bereme výstup z černobílého šumu a k jednotlivým barevným kanálům RGB přičítáme nebo od nich odečítáme hodnoty. Konkrétně k červenému kanálu je připočtena hodnota 54, modrému je odečítáno 108 a zelenému je odečítáno 23.

Výsledkem je struktura vypadající jako struktura dřeva smrku nebo borovice. Vidět je na obrázku 2.2.

Mraky

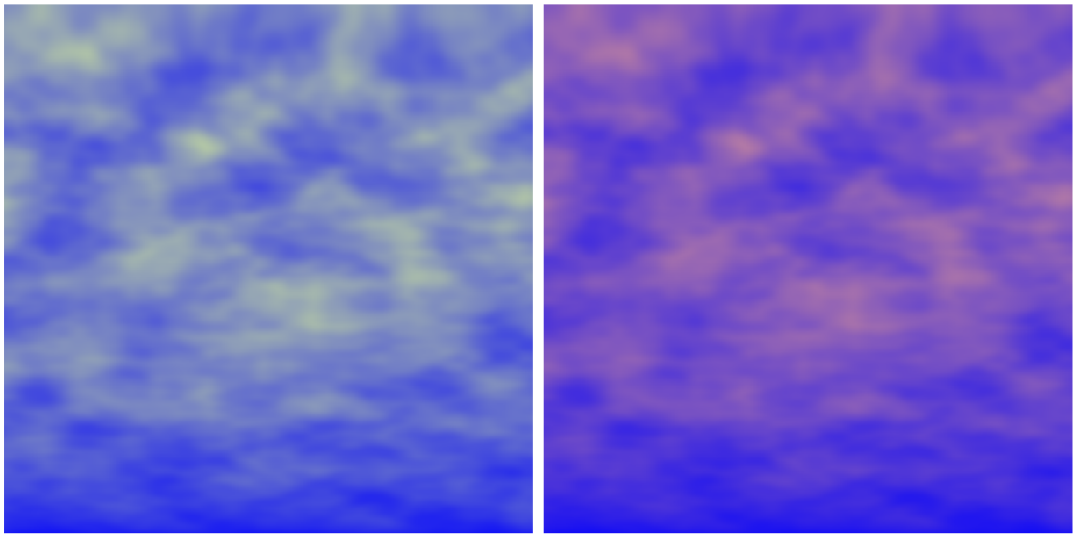
Mraky jsou v našem případě generovány funkcí `turbulence`, jejíž výstup je pro tuto aplikaci vhodný. Počtem oktáv v této součtové funkci můžeme regulovat ostrost okrajů mraků.

Pokud by nám stačil pouze pohled na oblohu pokrytou mraky ze země přímo nahoru, můžeme tento odstavec přeskočit a pokračovat rovnou na nastavení vhodných barev. Jelikož bylo cílem textury simulovat pohled lehce nad horizont, použijeme ještě funkci `bias(a, b)`, kterou můžeme regulovat roztažení mraků na obloze. Pro-

měnná a je hodnota osy která má být roztažena (v našem případě chceme roztáhnout osu y) a proměnná b s rozsahem $\langle 0, 1\rangle$ ovládá roztažení osy a rozprostření textury na ní [18]. Její vzorec je:

$$a^{\frac{\ln(b)}{\ln(0,5)}} \tag{2.5}$$

Dále už je jen třeba si pohrát s podkladem, u kterého chceme modrou barvu, a s barvou mraků, kterou chceme bílou, případně do červena pro generování červánků.



Obr. 2.7: Textura oblohy s bílými mraky (vlevo) a s červánky (pravo).

3 Návrh a implementace appletů

Praktická část je rozdělená na dvě hlavní části. V první se řeší softwarová část práce, tedy volba jazyků a knihoven použitých k implementaci jednotlivých appletů a také návrh grafických uživatelských rozhraní.

Applety mají fungovat jako jednoduchá webová stránka, kterou je možné zobrazit jak na běžném počítači, tak na mobilních zařízeních. Jejich ovládání má být jednoduché a snadno pochopitelné. Práce je proto řešená cestou jednoduchého grafického uživatelského rozhraní bez zbytečné grafiky okolo, která by mohla způsobovat špatnou interpretaci appletů v některých kombinacích operačních systémů a webových prohlížečů.

3.1 Software

V práci bylo nutné zvolit aktuální softwarové řešení, které bude fungovat i dlouhé roky po naprogramování s minimální nutností údržby.

Pro samotnou webovou stránku byl zvolen značkovací jazyk HTML ve své nejaktuálnější verzi 5. V něm byly použity styly CSS pro úpravu rozhraní, avšak jen v minimální verzi kvůli zachování jednoduchosti.

Pro samotné applety byl zvolen jazyk JavaScript, který je aktuálně jeden z nejpoužívanějších jazyků pro webová řešení aplikací. Pro účely grafického znázornění appletů byla zvolena knihovna JSXGraph, která umožňuje velmi jednoduše vytvářet matematické applety a je stále udržována a vylepšována.

3.1.1 HTML5

HTML5 neboli Hypertext Markup Language ve své páté verzi je značkovací jazyk používaný pro tvorbu webových stránek a jejich publikací na internetu. Jazyk HTML byl vyvinut z univerzálního značkovacího jazyka SGML pro použití na www stránkách.

Jeho pátá verze je nejnovější standard, který se od čtvrté verze liší novými zápisy značek, které jsou kratší a rychlejší. U jazyka byl kladen důraz na jednoduchost a efektivnost zápisu, jak pro ulehčení programátorům, zvýšení přehlednosti, tak pro zrychlení načítání stránek. Také se liší podporou přímého přehrávání multimédií ve webovém prohlížeči a podporou pro offline aplikace, které fungují bez připojení k internetu [6].

HTML5 funguje ve všech významných aktuálně používaných webových prohlížečích, jako je Google Chrome, Apple Safari, Microsoft Edge, Mozilla Firefox a dalších [7].

3.1.2 JavaScript

Dříve byl na podobné applety používán programovací jazyk Java. JavaScript je i přes podobné jméno velmi odlišným jazykem.

JavaScript je skriptovací jazyk, který je multiplatformní a objektově orientovaný. Svou syntaxí je podobný jazykům C++ a Java a patří do stejné rodiny jazyků. Používá se velmi často na webových stránkách, obvykle pro ovládací prvky a další interaktivní prvky.

Nejprve se spouštěl vždy na straně klienta, ale později byla zahrnuta podpora i na straně serveru. V současné době ho podporují všechny významné webové prohlížeče a patří mezi nejvíce používané programovací jazyky na webu [8].

JavaScript vs. Java

Hlavní rozdíl mezi JavaScriptem a Javou je, že Java musí být před spuštěním zkompilována. Její kód se musí vždy přeložit do kódu čitelného pouze pro virtuální stroj (JVM). Pro programování v Javě musíme mít stažený vývojářský balíček (JDK). Oproti tomu JavaScript můžeme psát v jakémkoli textovém editoru a nepotřebujeme k tomu nic stahovat. Kód se totiž spouští v čitelné podobě přímo v klientském webovém prohlížeči. Díky tomu je také JavaScript rychlejší. JavaScript je také jednodušší se naučit.

Jazyky se vyprofilovaly každý poněkud jiným směrem. Java je nyní používána především pro programování mobilních aplikací (Android) nebo desktopových aplikací a z webu postupně téměř vymizela. JavaScript je naproti tomu velmi využíván na webu a zaujal místo předchozích Java aplikací a Flashe. V kombinaci s HTML5 a CSS tvoří základ většiny moderních webů [9].

3.1.3 JSXGraph

Pro vizualizaci appletů byla zvolena knihovna JSXGraph, která je určena pro interaktivní vizualizace geometrických obrazců, matematických funkcí a jiných dat ve webovém prohlížeči. Není závislá na žádné jiné knihovně a je vcelku malá (asi 160 kB). Je také dobře optimalizována pro výkon. Podporuje multi-touch události a dobře funguje na zařízeních s dotykovými obrazovkami, kde umožňuje jak zadávání bodů dotykem, tak přesouvání bodů a přibližování či oddalování dvěma prsty.

Oproti jiným knihovnám tato umožňuje velmi jednoduché použití a množství připravených funkcí, které zjednodušují aplikaci při tvorbě interaktivních appletů. Spolu s dobře zpracovanou Wiki [10] plnou ukázek použití se pak jednalo o dobrou volbu pro tuto práci.

Tato knihovna je stále vyvíjena a udržována na Lehrstuhl für Mathematik und ihre Didaktik University of Bayreuth v Německu. Je dostupná k využití zdarma a je distribuována pod licencemi GNU LGPL a MIT [11].

3.1.4 HTML5 canvas

Speciálním prvkem HTML5 je canvas. Tento prvek slouží k dynamickému vykreslování grafiky pomocí JavaScriptu a HTML5 skrze Canvas API. Může být také použit pro vykreslení animací, herní grafiky, nebo pro grafickou vizualizaci dat [20].

Samotný HTML prvek canvas je oblast definovaná v HTML kódu pomocí tagu `<canvas>`, většinou doplněným o jeho ID a rozměry. Původně byl představen společností Apple pro využití v systému OS X. Později byl implementován také Firefoxem a dalšími webovými prohlížeči. Nyní je už jeho podpora rozšířena napříč všemi moderními prohlížeči [21].

Canvas API je zaměřeno především na 2D grafiku a je vhodné pro vykreslení textur generovaných appletem Generování procedurálních textur 3.5.

3.1.5 Vývojové prostředí WebStorm

Pro vývoj appletu bylo použito vývojové prostředí WebStorm od společnosti JetBrains, která jej poskytuje ve studentské licenci zdarma. Prostředí umožňuje jednoduché psaní appletů za použití HTML a JavaScriptu včetně knihovny JSXGraph. Umí také napovídat funkce a prvky z této knihovny. Možnosti debugování jsou pro potřeby vývoje více než dostatečné a prostředí umí zobrazit applet ve všech významných webových prohlížečích pro možnosti testování kompatibility.

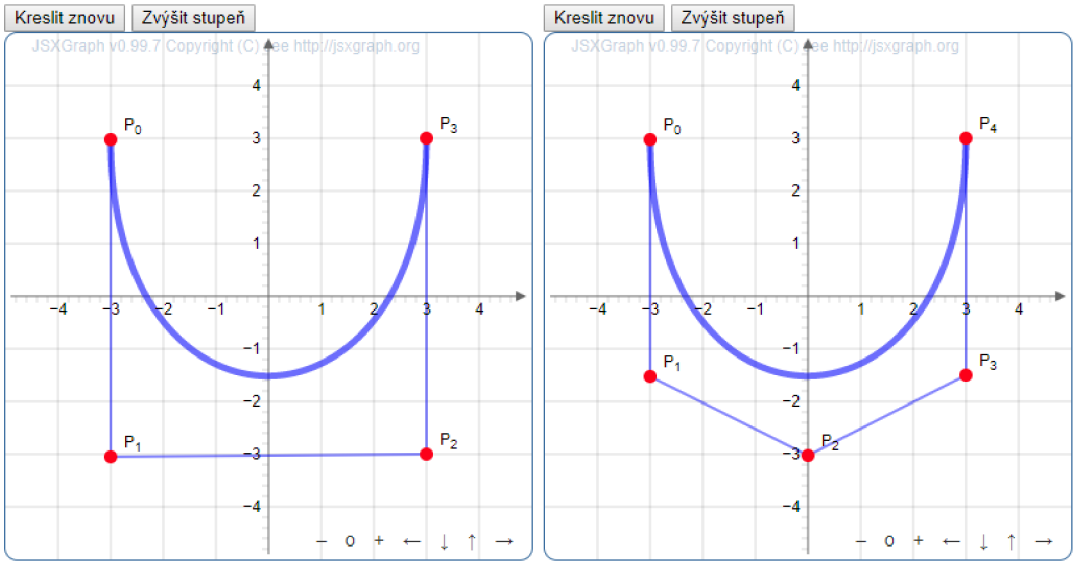
3.2 Applet Zvýšení stupně Bézierovy křivky beze změny jejího tvaru

Applet umožňuje zadání čtyř řídicích bodů, podle kterých je následně vykreslena Bézierova křivka. Teorie k Bézierovým křivkám je vysvětlena v kapitole 1.3.1.

3.2.1 Návrh grafického uživatelského rozhraní

Webová stránka s appletem se má skládat z několika částí. První část má obsahovat název appletu v horní části obrazovky. Dále zde mají být dvě tlačítka pro ovládání appletu. První umožňuje po kliknutí na „Kreslit znovu“ smazat všechny body a křivky v appletu a začít znovu. Druhé tlačítko po kliknutí na „Zvýšit stupeň“ provede výpočet pozice řídicích bodů při jejich počtu o jedna větším než je současný

a znovu ho vykreslí. Řídicí body jsou popsány jako P_0 až P_i, kde i zastupuje pořadí řídicího bodu. Body by měly být pojmenovávány pouze v tomto appletu, jelikož je zde možné je vcelku logicky pojmenovat. Následující část zobrazuje „tabuli“, s osou X a Y, a ovládacími prvky pro přiblížení, oddálení a posun po ose. Na ní je možné klikáním zadávat body a vykreslit křivku. Veškeré grafické znázornění se odehrává zde. Nejníže na obrazovce by měla být stručně vysvětlena teorie a výpočty, které se v appletu odehrávají. Návrh appletu je zobrazený na obrázku 3.1.



Obr. 3.1: Applet Zvýšení stupně Bézierovy křivky beze změny jejího tvaru.

3.2.2 Provedení appletu

Kód appletu se skládá z několika funkcí v JavaScriptu s využitím knihovny JSXGraph. Tou první funkcí je `getMouseCoords` která hlídá pozici ukazatele myši a následně pomocí funkcí `down` obsluhuje interakci myší (konkrétně stisk tlačítka myši) a podle nich vykresluje body a křivky na tabuli. Dále pak funkce `clean` smaže veškeré objekty z tabule po kliknutí na tlačítko „Kreslit znovu“. Poslední funkce v appletu `zvysitStupen` 3.1 vždy po kliknutí na tlačítko „Zvýšit stupeň“ přepočítá pozici bodů pro jejich počet o jedna vyšší a vykreslí je na tabuli. Funkce zobrazující původní Bézierovu křivku zůstává pořád stejná a je vykreslena podle prvotních řídicích bodů. Při výpočtu pozic více bodů vznikají kvůli zaokrouhlování drobné nepřesnosti, ale ani při velmi velkém počtu bodů nejsou nijak velké a v appletu nejsou příliš viditelné. Funkčnosti appletu nijak nebrání.

Níže je vložen kód funkce `zvysitStupen` 3.1, která si zaslouží bližší popis. Na začátku funkce je pozastaveno vykreslování objektů na tabuli, a to především kvůli

lepšímu výkonu a také aby applet uživateli během překreslování neproblikával. Dále jsou z tabule smazány objekty `p`, které představují body (points) a objekty `l`, které představují čáry (lines). Proměnné `p` a `l` jsou globální proměnné deklarované dříve. Následně je deklarována ještě proměnná `q` a skrze ni je pomocí metody push na řádku 6 na tabuli přidán první bod, který není zvýšením stupně křivky nijak ovlivněn. V cyklu `for` který začíná na řádku 8 jsou postupně vypočteny body po zvýšení stupně Bézierovy křivky a na řádku 17 jsou také přidány na tabuli. Na řádku 19 jsou následně přidány segmenty čar spojující jednotlivé body. Dále je ještě zanesen poslední bod a vykreslen poslední segment čáry. Na závěr jsou všechny body z proměnné `q` překopírovány do proměnné `p` a program je tak připraven na další zvyšování stupně křivek.

Výpis 3.1: Ukázka funkce pro výpočet zvýšení stupně Bézierovy křivky.

```
var zvysitStupen = function () {                                    1
        board.suspendUpdate();                                      2
        board.removeObject(p);      board.removeObject(l);          3
        var q = [];                                                 4
        q.push(board.create('point', [p[0].X(), p[0].Y()],          5
          {fixed:true, name:'P_0'}));                               6
        for (var i = 0; i < p.length - 1; i++) {                    7
            var t = 1 - ((i + 1) / p.length);                       8
                                                                    9
            var vzdalenost_x = p[i + 1].X() - p[i].X();             10
            var vzdalenost_y = p[i].Y() - p[i + 1].Y();             11
                                                                    12
            var pos_x = p[i].X() + vzdalenost_x * t;                13
            var pos_y = p[i].Y() - vzdalenost_y * t;                14
                                                                    15
            q.push(board.create('point', [pos_x, pos_y],            16
              {fixed:true, name:'P_{' + (i+1) + '}'}));             17
            l.push(board.create('segment',[q[q.length-2],           18
              q[q.length-1]],{strokeOpacity:0.5}));                 19
        }                                                           20
        q.push(board.create('point', [p[p.length - 1].X(),          21
          p[p.length - 1].Y()], {fixed:true, name:'P_{'             22
            + (i+1) + '}'}));                                       23
        l.push(board.create('segment',[q[q.length-2],               24
          q[q.length-1]],{strokeOpacity:0.5}));                     25
        p = [];     p = q.slice();                                  26
        board.unsuspendUpdate();                                    27
};                                                                  28
```

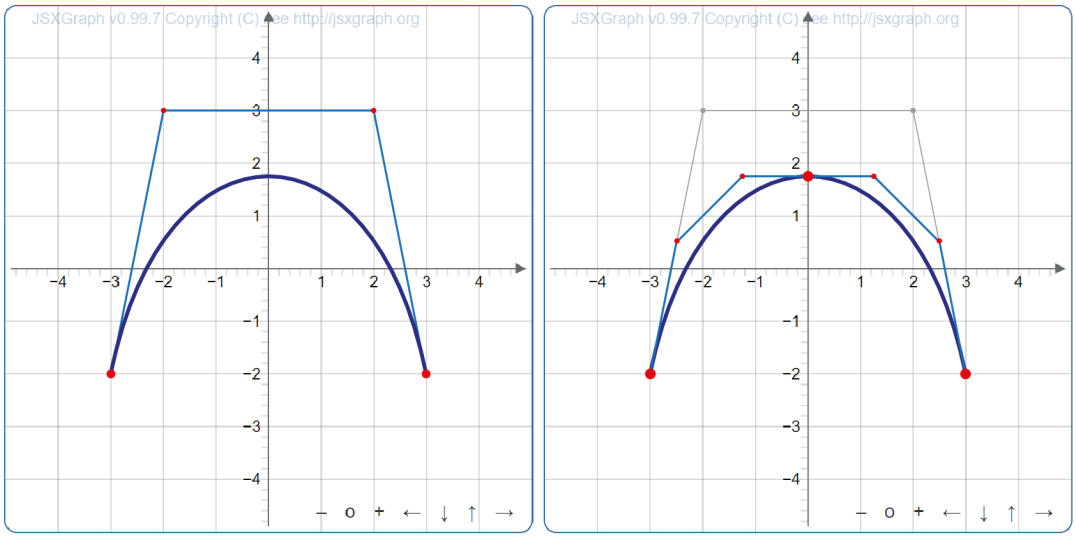
3.3 Applet Subsection Bézierovy křivky

Applet má umožňovat zadat čtyři řídicí body Bézierovy křivky a poté ji vykreslit. Následně má provést subdivision vykreslené Bézierovy křivky. Teorie k subdivision a subsection a jejich rozdíly jsou vysvětleny v kapitole 1.3.4.

3.3.1 Návrh grafického uživatelského rozhraní

Webová stránka s appletem má být složená z několika částí, podobně jako předchozí applet. Nahoře tedy název appletu, níže dvě ovládací tlačítka – první pro smazání bodů a kreslení křivky znovu, druhé pro provedení subsection. Následuje opět část zobrazující tabuli s osami X a Y a ovládacími prvky, na které se křivka vykresluje.

V původním návrhu appletu se počítalo pouze s provedením subdivision. Na obrázku 3.2 je vidět tento původní návrh appletu, který postrádá ovládací prvky pro volbu t a s. Vlevo je zadána základní křivka a napravo je již vidět applet po provedení prvního kroku subsection. Křivka je tedy již rozdělena na dvě a jsou zobrazeny nové řídící body. Původně se také počítalo se zašednutím předchozích řídicích bodů a čar, které je spojují.

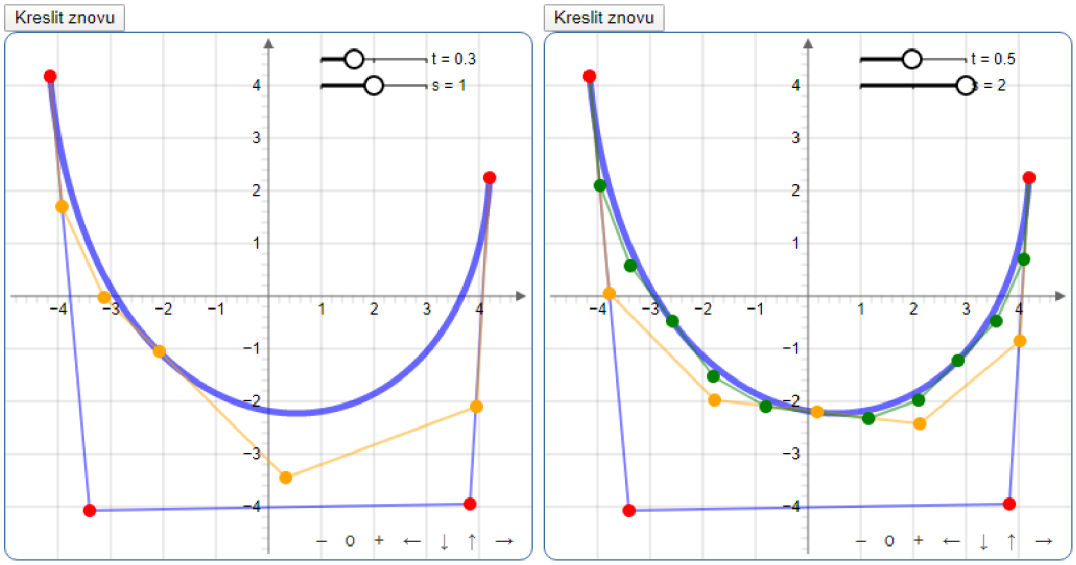


Obr. 3.2: Applet Subdivision Bézierovy křivky.

V průběhu vytváření appletu byl ale návrh přepracován a byly přidány dva ovládací prvky do pravé horní části appletu. První pro volbu času průběhu funkce t k provedení subsection a druhý pro volbu zobrazení počtu kroků subsection s. První krok dělení je znázorněn oranžovými body a čárami, druhý zelenými.

Na obrázku 3.3 vlevo můžete vidět jeden krok subsection provedený v bodě $t = 0{,}3$. Vpravo můžete na stejné křivce vidět dva kroky subsection provedené v bodě

$t = 0{,}5$. Podobně jako u předchozího appletu je zde tlačítko „Kreslit znovu“ které umožňuje smazat všechny body a křivky v appletu a začít znovu.



Obr. 3.3: Applet Subdivision Bézierovy křivky.

Body již v tomto appletu nejsou pojmenovávány, jelikož není možné jim přiřadit jméno, které by stručně a jednoznačně znázornilo jejich pořadí a příslušnost ke konkrétní křivce a jejich pořadí.

3.3.2 Provedení appletu

Kód appletu tvořen funkcemi v JavaScriptu s využitím knihovny JSXGraph. Opět je použita funkce `getMouseCoords` pro obsloužení interakcí myší a v návaznosti na ně vykresluje objekty. Zde je také kromě funkce `down` použita i funkce `up` jež se aktivuje až po uvolnění stisknutí tlačítka myši. To je vhodné pro vykreslení bodů až poté co se uživatel rozhodne, v jakém místě chce provést subsection, což snižuje náročnost appletu. Také funkce `clean` pro smazání všech objektů po kliknutí na tlačítko z předchozího appletu zde našla uplatnění. Ta je zde také doplněna funkcí `cleanM` pro smazání objektů vzniklých provedením subsection. Funkce `provestSubsection1` a `provestSubsection2`, jsou volány dle volby stupně provedení subsection a vykreslují jejich body a čáry.

Důležitou součástí je funkce `deCasteljau`, jejíž kód můžete vidět níže 3.2, která algoritmem de Casteljau vypočte novou pozici řídicích bodů po provedení subsection ve zvoleném čase. Vstupními přoměnnými funkce jsou `vstPoints`, tedy původní body křivky, případně část křivky určená k dalšímu stupni dělení, a čas ve kterém

má být křivka rozdělena `vstT`. Ty jsou následně přepsány do lokálních proměnných `points` (cyklem `for`) a `t` (přímo u deklarace proměnné). Následuje cyklus `while`, kde už probíhá samotný výpočet pomocí algoritmu de Casteljau. Ten prochází všechny body a pomocí středových bodů `midpoints` počítá nové řídicí body. Na závěr je ještě z pole bodů vyjmut středový bod mezi 2. a 3. řídicím bodem pomocí `returnPoints.splice(1, 1)`, jelikož pro potřeby subsection není zajímavý a body jsou vraceny v proměnné `returnPoints`.

Výpis 3.2: Ukázka funkce pro výpočet subsection algoritmem de Casteljau.

```
function deCasteljau(vstPoints, vstT) {
    var a, b, i,
        t = vstT,
        midpoints = [],
        returnPoints = [],
        points = [];
    for (i = 0; i < vstPoints.length; i++) {
        points[i] = [vstPoints[i].X(), vstPoints[i].Y()]
    }
    while (points.length > 1) {
        for (i = 0; i < points.length - 1; ++i) {
            a = points[i];
            b = points[i + 1];
            midpoints.push([
                a[0] + ((b[0] - a[0]) * t),
                a[1] + ((b[1] - a[1]) * t)
            ]);
            returnPoints.push(midpoints[i]);
        }
        points = midpoints;
        midpoints = [];
    }
    returnPoints.splice(1, 1);
    return returnPoints;
}
```

3.4 Applet Catmull–Clark subdivision

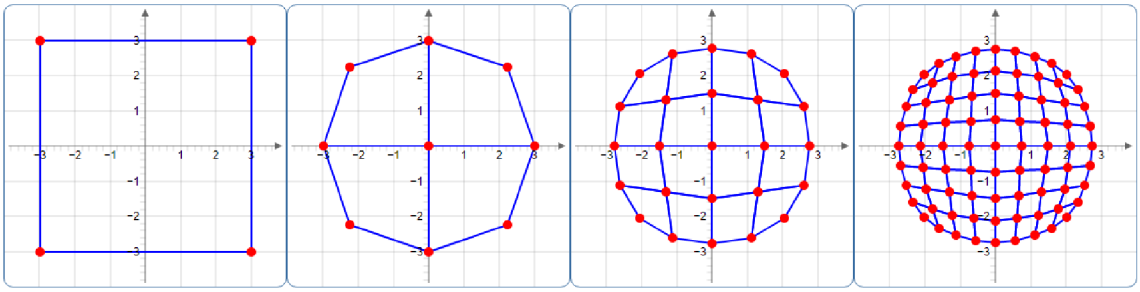
Applet by měl názorně zobrazovat Catmull–Clark subdivision dle teorie vysvětlené v kapitole 1.4. Pro lepší pochopení a názornost je applet provedený pouze ve 2D.

3.4.1 Návrh grafického uživatelského rozhraní

Základ webové stránky s appletem má být stejný jako u předchozích appletů. Změna je ale v počtu tabulí zobrazujících grafiku. Ty mají být čtyři a dle možností zobrazovacího zařízení (monitoru PC či displeje mobilního zařízení) buď zobrazeny vedle sebe, případně pod sebou.

Na první tabuli mají být v základu zobrazeny čtyři body ve tvaru čtverce zarovnaného se středem os X a Y. S nimi má být možné hýbat a různě je přesouvat. Další tabule mají zobrazovat následující kroky Catmull–Clark subdivision a zde mají být řídicí body zamčené. Celkově se tedy zobrazí základní rozložení a tři kroky Catmull–Clark subdivision.

Při vytváření appletu se ovládací prvky pro přiblížení v pravém spodním rohu projevily jako nadbytečné, a proto byly skryty. Stejně tak jakékoli jiné ovládací prvky, kromě možnosti posouvat výchozí body v prvním poli. Tento applet je tedy maximálně jednoduchý. Finální návrh uživatelského rozhraní je vidět na obrázku 3.4.



Obr. 3.4: Applet Catmull–Clark subdivision, finální návrh.

3.4.2 Provedení appletu

Kód appletu je opět tvořen JavaScriptem spolu s knihovnou JSXGraph. V appletu se pracuje s globálními proměnnými pro jednotlivé vykreslované skupiny bodů a čar. Pro obsluhu uživatelských interakcí je zde použita pouze funkce `up`, tedy uvolnění stisknutého tlačítka myši, jelikož má být applet překreslen až poté co si uživatel zvolí vstupní pozici bodů čtyřúhelníku.

Dále je applet složen z funkce `provedHlavni`, která se stará o výpočet jednotlivých kroků subdivision a jejich výpočet a funkce `smazHlavni`, jež se stará o smazání vykreslených bodů a čar, aby bylo možné je vykreslit znovu na nových pozicích.

Funkce `provedSubdivision`, `provedSubdivision2` a `provedSubdivision3` provádí každá samostatný krok subdivision. Vstupním parametrem těchto funkcí je vždy

matice bodů z předchozího kroku subdivision. Není tedy použita jedna univerzální funkce, která by zvládla vypočítat Catmull–Clark subdivision pro libovolně velké pole, ale pracuje se s několika funkcemi. Bohužel pak dochází k drobné chybě při zaokrouhlování, která se při třetím kroku subdivision projevovala na pozici bodů v appletu a bylo nutné použít korekci pro věrné znázornění subdivision.

V těchto funkcích jsou používány funkce pro výpočet třech typů bodů (označovaných také jako vrcholy). Funkce `vypocetF` tedy počítá novou pozici F, funkce `vypocetE` pozici E a funkce `vypocetV` počítá novou pozici vrcholů q. Použitý algoritmus je vysvětlený v kapitole 1.4.

Dále jsou použity funkce pro manipulaci s polem a výpočet hodnot `prumerBodu`, `pocetBodu`, `soucetBodu`, `prevodPole1D2D` a `rozdelMaticiNa4`, jejichž názvy napovídají, co funkce s polem nebo body dělají. K funkcím `vykresliBody`, `smazBody`, `vykresliCary` a `smazCary` už není nutné nic dodávat.

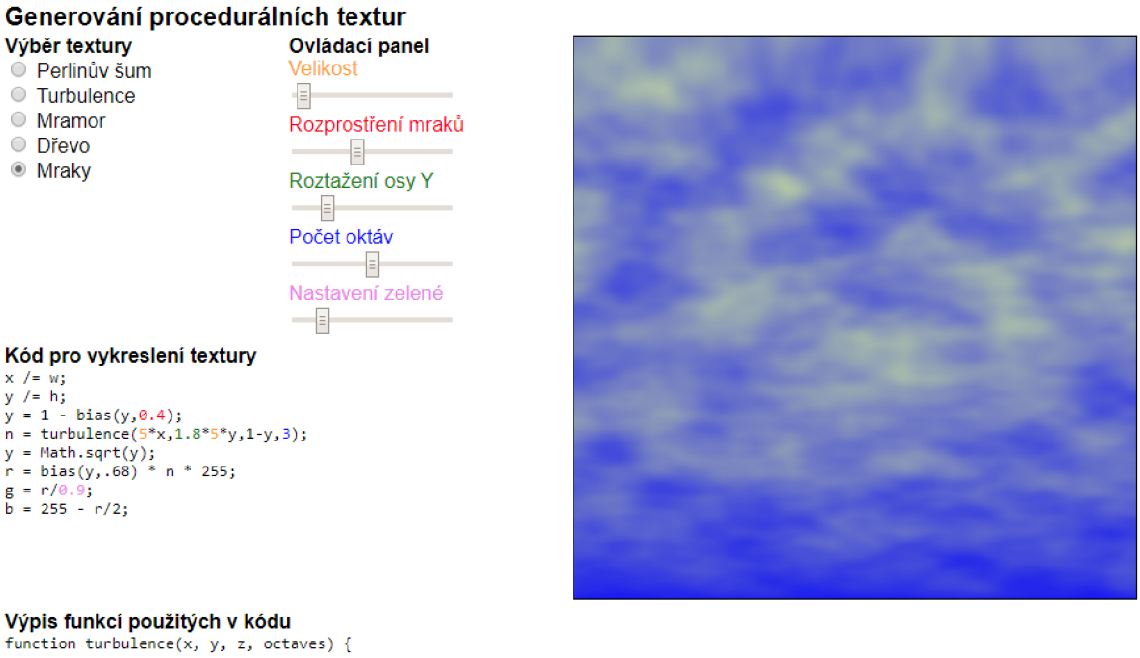
3.5 Applet Generování procedurálních textur

Tento applet bude zásadně odlišný od předchozích. Je propracovanější, ovládací prostředí je mnohem složitější a umožňuje více uživatelských voleb. Generování procedurálních textur je totiž odlišná část počítačové grafiky. Teorie k appletu je vysvětlena v kapitole 2.2.

3.5.1 Návrh grafického uživatelského rozhraní

Základ webové stránky má být opět stejný, tedy nahoře název, dále samotný applet a pod ním základní teorie. Rozhraní samotného appletu ale má být naprosto odlišné. Má se skládat z výběru textury, ovládacího panelu s posuvníky a prostoru kde se samotná textura vykreslí. Ovládací panel se má měnit dle požadavků konkrétní zvolené textury.

Během tvorby appletu bylo provedeno ještě několik změn a vyplepšení. Především byl pro větší názornost přidán blok, kde je vypisován kód v JavaScriptu použitý pro generování textury, který se při každé změně parametrů mění. V něm jsou barevně odlišeny hodnoty, které jsou ovládány pomocí posuvníků v ovládacím panelu. Popisy posuvníků jsou zobrazeny stejnou barvou a jejich propojení je tak velmi názorné. Jelikož jsou v kódu pro generování textur používány funkce, jejichž vstupní parametry a především jejich účel nemusí být na první pohled zřejmý, bylo přidáno také jejich vypisování. Výsledný návrh je vidět na obrázku 3.5.



Obr. 3.5: Applet Generování procedurálních textur, finální návrh, zkráceno.

3.5.2 Provedení appletu

Implementace tohota appletu byla velmi odlišná od ostatních. To je dáno už tím, že procedurální textury jsou úplně odlišná část počítačové grafiky. Stránka je stejně jako předchozí applety vytvořena pomocí HTML5 a především HTML5 elementu „canvas“. Knihovna z minulých appletů zde uplatnění nenašla. Kód je psaný v JavaScriptu s využitím Canvas API.

Uživatelské prostředí je zde o něco složitější a je složené z HTML elementů „radio“, pomocí kterých je vybíraná konkrétní textura a dále z elementů „range“, kterými jsou vybírány hodnoty podle kterých je následně textura vykreslována. Tyto elementy jsou pomocí JavaScriptu dynamicky skrývány a jsou nastavovány jejich parametry podle požadavků konkrétní textury. K tomu slouží funkce `displaySlider` a `setSlider`.

Byly zde také ve větší míře využity CSS styly pro uspořádání jednotlivých prvků uživatelského prostředí a pro dynamickou změnu jejich pozice na základě velikosti zobrazovacího zařízení.

Základem generování textur je třída `PerlinNoise`, které obsahuje předem vygenerované pseudonáhodné pole a také funkce pro generování šumu. K vykreslení textury do elementu canvas jsou používány funkce `drawViaCallback` a `doPixelLoop`. Ty využívají funkci `callback` jejíž kód se mění podle vybrané textury a zadaných parametrů. Jednotlivé textury používají pro generování další funkce, jako jsou

`turbulence`, `ramp`, `marble` nebo `bias`. Při tvorbě appletu byl jako základ použit kód, který vytvořil Kas Thomas [18]. Ten byl následně výrazně rozšířen a upraven.

K výběru textury slouží funkce `setChosenTexture`. Ta na základě vybrané textury zvolí, jak se má nastavit uživatelského prostředí. Níže je vidět kód pro nastavení uživatelské prostředí pro první texturu.

Výpis 3.3: Ukázka funkce pro nastavení UI pro první texturu.

```
function texture1Set() {
        setSlider(slider1,0,100,1,50, "Velikost", "orange");
        setSlider(slider2,0,1,.1,.5, "Osa X", "red");
        setSlider(slider3,0,1,.1,.5, "Osa Y", "green");
        setSlider(slider4,0,1,.1,.5, "Osa Z", "blue");
        displaySlider(slider5, false);
        extractUsedCode("PerlinNoise.noise = "
        +replaceAll(PerlinNoise.noise.toString(),"this.","")
        + "<hr>fade = " + PerlinNoise.fade.toString()
        + "<hr>scale = " + PerlinNoise.scale.toString()
        + "<hr>lerp = " + PerlinNoise.lerp.toString()
        + "<hr>grad = " + PerlinNoise.grad.toString());
    }
```

Vykreslení zvolené textury obsluhuje funkce `drawChosenTexture`. Níže je vidět funkce pro vykreslení první textury, která je touto funkcí volána.

Výpis 3.4: Ukázka funkce pro vykreslení první textury.

```

    function texture1Draw() {
        const kod = "x = w;\n" +
            "y = h;\n" +
            "n = PerlinNoise.noise(slider1 * x * slider2 ," +
            "slider1 * y * slider3 , slider4);\n" +
            "r = g = b = 255 * n;";

        extractCodeCanvas(kod);
        drawCodeCanvas(kod);
    }
```

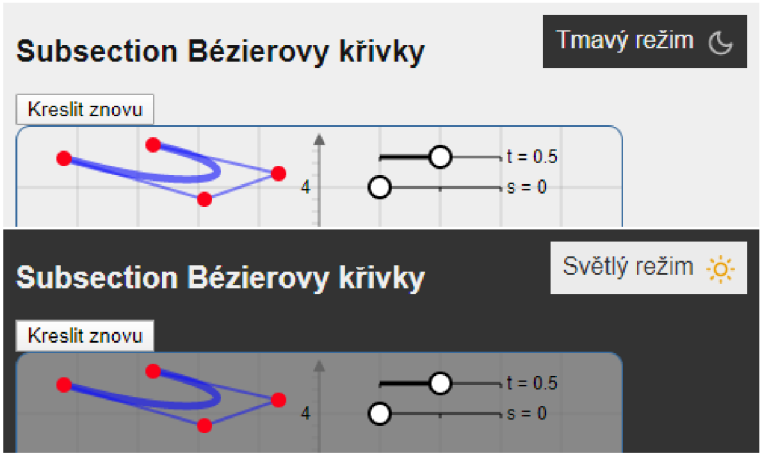
Důležitá je také funkce `drawCodeCanvas`, která připraví kód k vykreslení a nahradí zástupné texty za čísla vybraná posouvání jednotlivých range sliderů.

K vypisování použitého kódu je používána funkce `extractCodeCanvas`. Pro větší názornost je zde také funkce `extractUsedCode`, která vypisuje funkce použité při generování jednotlivých textur.

3.6 Tmavý režim

Velkou módou současných aplikací je možnost používat je v tmavém režimu. Ten přijde vhod především v noci, kdy bílé plochy v aplikaci jasně září a nejsou pro oči přívětivé.

Jelikož tyto applety obsahují velkou bílou plochu na pozadí, je implementace tmavého režimu více než vhodná. Režim appletu se mění automaticky dle uživatelem nastavaného režimu v systému či v prohlížeči. Stále je zde ale možnost ho přepnout pomocí tlačítka v pravém horním rohu.



Obr. 3.6: Světlý a tmavý režim appletu.

Závěr

V rámci teoretické části byla popsána problematika týkající se počítačové grafiky. Především jsou popsány křivky, jejich vlastnosti, jejich modelování a dále již teorie Bézierových křivek a kubik. Také jsou zde popsány algoritmy subdivision pro Bézierovy křivky a pro Catmull–Clark subdivision. Dále byla popsána také teorie textur, a především procedurálních textur. Následně byly rozebrány základní funkce pro generování šumu a textur založených na Perlinově šumu a bylo popsáno generování textur mramoru, dřeva a mraků.

Praktická část práce se zabývá nejprve volbou softwarového řešení appletů, výhodami a důvody volby jednotlivých programovacích jazyků a knihoven. Následují návrhy samotných appletů a popisy jejich funkcionality. Applety také byly implementovány a jsou dostupné k použití. Vysvětlena je i funkce podstatných částí jejich kódu.

Zadání bakalářské práce tedy bylo splněno a applety můžou plnit svoji funkci při podpoře výuky počítačové grafiky.

Literatura

[1] ŽÁRA, Jiří, Bedřich BENEŠ, Jiří SOCHOR a Petr FELKEL. *Moderní počítačová grafika.* 2., přepracované a rozšířené vydání. Brno: Computer Press, 2005. ISBN 80-251-0454-0.

[2] RAJMIC, Pavel, Jiří SCHIMMEL. *Moderní počítačová grafika.* Brno: Vysoké učení technické v Brně, 2013. ISBN: 978-80-214-4906-0.

[3] LEKNEROVÁ, Kristýna. *Tvorba kaligrafického písma* [online]. Brno, 2016 [cit. 2019-12-16]. Bakalářská práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce Pavel Matula. Dostupné z: <`https://is.muni.cz/th/gumcc/`>

[4] PROCHÁZKA, David, Tomáš KOUBEK a Jana DANNHOFEROVÁ. *Programování grafických aplikací s využitím OpenGL a OpenCV: Textury* [online]. 2009 [cit. 2019-12-16]. Dostupné z: <`https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=23680`>

[5] KUŘEOVÁ, Pavlína. *Procedurální generování textur* [online]. Hradec Králové, 2016 [cit. 2019-12-16]. Bakalářská práce. Univerzita Hradec Králové, Fakulta informatiky a managementu. Vedoucí práce Karel Petránek. Dostupné z: <`https://theses.cz/id/whxotv/`>

[6] *HTML standard* [online]. [cit. 2019-12-16]. Dostupné z: <`https://html.spec.whatwg.org/multipage/introduction.html`>

[7] *W3Counter: Global Web Stats - November 2019* [online]. 2019 [cit. 2019-12-16]. Dostupné z: <`https://www.w3counter.com/globalstats.php?year=2019&month=11`>

[8] *Usage statistics of JavaScript as client-side programming language on websites* [online]. [cit. 2019-12-16]. Dostupné z: <`https://w3techs.com/technologies/details/cp-javascript`>

[9] MARTIN, Shifa. Hacker Noon. *Java vs JavaScript: Here's What You Need To Know* [online]. 2019 [cit. 2019-12-16]. Dostupné z: <`https://hackernoon.com/java-vs-javascript-heres-what-you-need-to-know-9b1b708394e2`>

[10] *JSXGraph Wiki* [online]. [cit. 2019-12-16]. Dostupné z: <`http://jsxgraph.uni-bayreuth.de/wiki/index.php/Main_Page`>

[11] *Github - jsxgraph/jsxgraph: JSXGraph is a cross-browser library for interactive geometry, function plotting, charting, and data visualization in a web browser.* [online]. [cit. 2019-12-16]. Dostupné z: <https://github.com/jsxgraph/jsxgraph>

[12] CATMULL, E. a J. CLARK. *Recursively generated B-spline surfaces on arbitrary topological meshes. Computer-Aided Design* [online]. 1978, 10(6), 350-355 [cit. 2019-12-16]. DOI: 10.1016/0010-4485(78)90110-0. ISSN 00104485. Dostupné z: <https://linkinghub.elsevier.com/retrieve/pii/0010448578901100>

[13] OTOČKA, Dávid. *Subdivision Surfaces* [online]. Brno, 2007 [cit. 2019-12-16]. Bakalářská práce. Vysoké učení technické v Brně. Fakulta informačních technologií. Ústav počítačové grafiky a multimédií. Vedoucí práce Stanislav Sumec. Dostupné z: <http://hdl.handle.net/11012/56350>.

[14] FERIN, Gerald. *Curves and surfaces for computer-aided geometric design.* 4th ed. Cambridge (Massachuttes): Academic Press, 1996. ISBN 978-0-12-249054-5.

[15] *3-Dimensional Smoothing: Catmull-Clark Subdivision. Algosome Software Design* [online]. [cit. 2020-06-06]. Dostupné z: <https://www.algosome.com/articles/catmull-clark-subdivision-algorithm.html>

[16] *James H. Clark — co-founder of Silicon Graphics, Netscape. Stanford School of Engineering* [online]. [cit. 2020-06-06]. Dostupné z: <https://engineering.stanford.edu/about/heroes/james-clark>

[17] *Perlin noise. Rosetta Code* [online]. [cit. 2020-06-07]. Dostupné z: <https://rosettacode.org/wiki/Perlin_noise>

[18] *THOMAS, Kas. Perlin Noise in JavaScript. Blogorrhea* [online]. 2011 [cit. 2020-06-07]. Dostupné z: <http://asserttrue.blogspot.com/2011/12/perlin-noise-in-javascript_31.html>

[19] *BIAGIOLI, Adrian. Understanding Perlin Noise. Adrian's soapbox* [online]. [cit. 2020-06-07]. Dostupné z: <https://adrianb.io/2014/08/09/perlinnoise.html>

[20] *Canvas API. MDN Web Docs* [online]. 2020 [cit. 2020-06-07]. Dostupné z: <https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API>

[21] *The canvas element. HTML Standard* [online]. 2020 [cit. 2020-06-07]. Dostupné z: <https://html.spec.whatwg.org/multipage/canvas.html>

A Obsah přílohy

V přiloženém ZIP souboru jsou uloženy veškeré soubory nutné ke spuštění vytvořených appletů. Jedná se o běžné HTML soubory, které je možné přímo otevřít ve webovém prohlížeči a applety používat. Applety jsou kromě přílohy také dostupné na adrese `http://bp.drapela.cz/` (k datu 8. 6. 2020).

Applety byly testovány na PC s operačním systémem Windows 10 verze 2004 na běžně používaných webových prohlížečích v jejich aktuálních verzích k datu 6. 6. 2020. Konkrétně Google Chrome verze 83.0.4103.97 a Microsoft Edge verze 44.19041.1.0.

- `/` ..... kořenový adresář přiloženého ZIP souboru
  - `index.html` ..... rozcestník vytvořených appletů
  - `bezierStupen.html` ... applet Zvýšení stupně Bézierovy křivky beze změny jejího tvaru
  - `subsection.html` ..... applet Subsection Bézierovy křivky
  - `catmullClark.html` ..... applet Catmull–Clark subdivision
  - `proceduralniTextury.html` ..... applet Generování procedurálních textur
  - `appletStyle.css` ..... CSS styly appletů
  - `jsxgraphcore.js` ..... JavaScript knihovna JSXGraph ve verzi 0.99.7
  - `jsxgraph.css` ..... CSS styly knihovny JSXGraph
  - `darkMode.js` ..... JavaScript kód tmavého režimu
  - `darkModeStyle.css` ..... CSS styly tmavého režimu