

Střední průmyslová škola a Vyšší odborná škola, Liberec, příspěvková organizace

GeogebraLite

Ročníková práce

Autor **Jakub Havlas**

Obor **Informační technologie**

Vedoucí práce **Ing. Jan Šimůnek**

Školní rok **2024/2025**

Počet stran **17**

Počet slov **2318**

Anotace

Moje práce se zabývá vytváření grafů z matematických funkcí. Práce vychází z aplikace Geogebry, která umožňuje právě vypsání grafů. Cílem bude vytvořit knihovnu v react a po té ji demonstrovat na prototypové aplikaci.

Summary

This work …

Čestné prohlášení

Prohlašuji, že jsem předkládanou maturitní práci vypracoval sám a uvedl jsem veškerou použitou literaturu a bibliografické citace.

V Liberci dne

Jakub Havlas

Obsah

[Úvod 1](#_Toc195816960)

[1 Úvod do svg 2](#_Toc195816961)

[1.1 Obecný způsob 2](#_Toc195816962)

[1.1 Beziérový způsob 2](#_Toc195816963)

[1.2 Cíle knihovny 3](#_Toc195816964)

[1.3 Library Controller 3](#_Toc195816965)

[1.3.1 ParseExpression 3](#_Toc195816966)

[1.3.2 UseDebounce 3](#_Toc195816967)

[1.4 General 4](#_Toc195816968)

[1.4.1 useMemo 4](#_Toc195816969)

[1.5 Evaluator 4](#_Toc195816970)

[2 Implementace 5](#_Toc195816971)

[2.1 Asymptote-detection tool 5](#_Toc195816972)

[2.1.1 Asymptoty 5](#_Toc195816973)

[2.1.2 Intervaly 5](#_Toc195816974)

[3 Optimalizace 6](#_Toc195816975)

[3.1 SmartStep 6](#_Toc195816976)

[3.2 Cache 6](#_Toc195816977)

[3.2.1 Session storage 6](#_Toc195816978)

[3.2.2 Indexed Databese 7](#_Toc195816979)

[3.2.3 Zjevení - AppData + useMemo 7](#_Toc195816980)

[4 Výzvy 8](#_Toc195816981)

[Závěr 9](#_Toc195816982)

[Seznam zkratek a odborných výrazů 10](#_Toc195816983)

[Seznam obrázků 11](#_Toc195816984)

[Použité zdroje 12](#_Toc195816985)

[A. Seznam přiložených souborů I](#_Toc195816986)

Úvod

Tuto práci jsem si vybral, protože mě baví matematika chtěl bych si vyzkoušet, jak se propojují tyto dva obory. Myslím, že tato práce je právě zlatý střed mezi programovací a matematickou komplexitou.

Nastavte **název** dokumentu a **autora** v nabídce Soubor/Informace.

Pro vkládání zdrojů použijte Reference/Spravovat prameny.

# Úvod do svg[[1]](#footnote-1)

Jako způsob vykreslování grafů jsem si zvolil formát programaticky vykreslovat svg. Svg se pro tuto práci hodí hlavně proto, že to není rastrový obrázek ale vektorový. To znamená, že namísto piselů a barev, můžu jen napsat, body, vektory a křivky a tento formát už přepočítá na pixel pro monitor sám. Výhodou tohoto způsobu je také, že nemusím řešit jakkoukoli ostrost, protože jak už bylo řečeno dříve, theto formát si to vypočítá sám.

## Obecný způsob

Prvním způsobem, který jsem vymyslel je obecný způsob. To znamená, že udělám obecnou funkci pro každou matematickou funkci, která vypočíta x bodů na viewport. Výhodami tohoto způsobu jsou, že setačí pouze jedna funkce na celou aplikaci. Na druhou stranu nevýhoda je, že musím řešit zoom a detail.

## Beziérový způsob

Druhý způsob, který jsem měl na výběr, využívá Bézierovy křivky. Tyto křivky se zapisují například takto:

d="M 100 350 Q 225 50 350 350"

V uvedené citaci si můžeme všimnout písmene Q, které značí kvadratickou Bézierovu křivku (quadratic Bézier curve). Za tímto písmenem následují dvě sady souřadnic. Poslední bod [350, 350] je koncový bod, tedy místo, kde cesta skončí. První bod [225, 50] je tzv. control point (řídicí bod), který určuje zakřivení dráhy.

Bézierovy křivky fungují tak, že místo aby cesta šla přímo mezi dvěma body, je její průběh ovlivněn tímto řídicím bodem. V případě kvadratické Bézierovy křivky je zakřivení definováno jedním řídicím bodem. Výsledná křivka je tedy hladká a elegantně zakřivená trajektorie, která nezačíná ani nekončí v řídicím bodě, ale je jím pouze "tvarována".

Tento způsob zápisu je užitečný například při kreslení hladkých oblouků, zakřivených linií nebo ozdobných prvků ve vektorové grafice. Díky přesnému řízení tvaru křivky pomocí souřadnic je možné vytvářet vizuálně atraktivní a plynulé tvary, které jsou často využívány v SVG grafice či v animačních nástrojích. Návrh knihovny

Po konzultaci s Vedoucím práce, jsme se rozhodli, že nejlepší bude, když začnu obecným způsobem. Myslím, že toto rozhodnutí bylo správné, protože by beziérové křivky nevypadali tak hezky a také by byli dále nad úroveň mé práce. Mým cílem bylo vytvořit knihovnu tak aby si uživatel mohl vybrat využít buď library controller, jako jednodučí volbu anebo General, jako složitější, ale flexibilnější.

## Cíle knihovny

Na začátku jsem nevěděl jak tuto práci mám pojmout, nevěděl jsem jestli knihovna bych měl exportovat z knihovny jen funkci na vytvoření křivky nebo i grafu. Ale na konec jsem se rozhodl pro obojí. Library controller, bude obsahovat veškeré potřebné věci, takže budoucí uživatel jej bude moci hned používat, zatímco funkce General bude mít výstup jen křivky. Tato funkce sice bude méně používaná, ale bude dovolovat uživateli jakoukoli modifikaci.

## Library Controller

Library controller je nejdůležitější součástí knihovny. Tato komponenta řídí vše ostatní. Vlastně je to takový mozek celé knihovny. Tato komponenta má na vstupu tři parametry. Reqs[], viewbox. Reqs je typ který obsahuje expression: string a nepovinný parametr color?: string.

První jhe viewbox naparsouván do useState. Po té začne ta nejdůležitější část, for loop, který porojíždí reqs. První to využije funkci ParseExpression, ke které se dostaneme za chvíli. Po té to prohledá AppData, jestli tato funkce není už v cache. Pokud ano, přidá to parametr do funkce General.

### ParseExpression

Tato funkce, znormalizovala expression string do expression array string. Probléme totiž byl, že Array Evaluator musí dosstat validní array. Tato funkce proto musí být připravena na implicitní násobení, závorky v závorkách a detekovat funkce a seskupit je. Tato funkce využívá Regex. A byla napsána umělou inteligencí.

### UseDebounce

Custom hook useDebounce slouží k odkládání spuštění funkce. Toto využití se až skvěle hodí jako řešení pro optimalizaci v mé aplikaci. Potýkal jsem se totiž s problémy po předání zoomu, že po jakkémkoli dotknutí kolečka myši se aplikace pokoušel re-renderovat veškeré komponenty. Takže při scrollu to až 5 krát vyrendererovalo celý graf.

Toto bylo zbytečné a hodně to zpomalovalo celou aplikaci a tím i user experience.

## General

Komponenta General je druhou nejdůležitější částí celé práce. Tato funkce byla asi nesložitější část práce a trvalo mi nejdéle ji vymyslet, napsat a ladit. Problém vždycky je, že tato komponenta je výpočetně náročná a vždy vypočítá aspoň 1000 bodů. V průběhu tvorby tato komponenta začala využívat useMemo který výrazně napomohl k výkonu. Další výzvou se kterou jsem se poměrně dlouho potýkal a v tomto momentě se ještě stále potýkám je hledání asymptot. Tento algoritmus byť potřebný, vyžadoval další výpočetní kapacitu. Vlastně celá tato práce by se dal popsat balancováním výkonu a dalších funkcí.

### useMemo

Hook useMemo se v tomto případě používá k optimalizaci výpočtu vykreslovaného grafu na základě zadaného výrazu, pohledu (viewBox) a případně uložených předchozích dat (storedExpression). Výpočet grafu (ať už plný nebo částečný) je poměrně náročná operace – zahrnuje iteraci, výpočty a tvorbu SVG cest. Proto se díky useMemo výsledek znovu spočítá pouze tehdy, pokud se některý z klíčových vstupů změní.

To je důležité pro výkon a plynulost aplikace, protože zamezí opětovnému vykreslování, pokud se data nezměnila – například při přiblížení grafu nebo úpravě jiné části UI. Kromě toho je zajímavé i to, že kód chytře detekuje, zda může využít existující předpočítaná data (částečný update), nebo zda je potřeba spočítat celý graf znovu – čímž kombinuje memoizaci s diferencovaným výpočtem.

## Evaluator

K vyhodnocování matematických výrazů, které mohou obsahovat proměnnou

𝑥, funkce (například sin, log) i matematické konstanty (𝜋, 𝑒, aj.), je použita funkce evaluator. Tato funkce je napsaná tak, že si poradí s jakýmkoli matematickým výrazem, pokud je na vstupu ve validním tvaru. Tato funkce umí řešit, tákladní operace (+, - ,…) dále matematické funkce jako sin, tan, pak ještě závorky a to i závorky abolutní hodnoty. Jako poslední funkce evaluatoru je, že dokáže řešit výrazy se známími konstanty, jako pi, e.

Co ale Evaluator neumí, je jakkékoli zjednodušování výrazů. To znamená, že vypočítá každý výraz, tak jak je napsaný do vstupu. Toto může taky vést k budoucím optimalizačním problémům.

Achylova pata této funkce je, že je drahá. Tato funkce obsahuje několik forloopů a ještě recursivní volání v závorkách. Proto bylo cílem většiny optimalizací spouštět tuto funkci co nejméně.

# Implementace

V této kapitole jsou popsány klíčové technické prvky systému, které zajišťují efektivní vykreslování matematických funkcí a interaktivitu při práci s grafy.

## Asymptote-detection tool

Myslím, že nejdelší dobu mi zabralo a ještě zabere. Asymptote-detection tool. Tato implementace navržená pro hledání asymptot byla vymyšlena v 11 večer po dvou hodinách programování (stejně jako je právě teď psaná tato práce). A podle toho také vypadá. Vlastně tato implementace funguje docela jednoduše. Když je y zahranicemi vertikálního viewbox a na další iterací je za hranicemi znovu ale na druhé straně, je prohlášena tato část za asymptotu.

Toto řešení ihned přinášé dva nové problémy. Co když tam bude tak strmý polynom, který ve dvou bodech přeskočí celý graf? aplikace se rozbije

Nepracuje aplikace s náhodou? Co když další bod nebude nad hranicí na druhé straně? Aplikace se rozbije

První problémem jsem se zas tolik netrápil. Ono takový polynom, to by musel být asi 1000x^3 a tam si myslím, že by to detekovalo nějaké body. Ale Strach a zármutek mi spíš dělal ten druhý problém.

Tento problém jsem se rozhodl vyřešit SmartStepem. O kterém ssi povíme v další kapitolce. Jiné řešení než spoléhat na náhodu mě nenapadalo. Chce to asi prostě spočítat hodně bodů na velkém viewportu, říkal jsem si. A to bude znamenat, že budu muset ještě dynamicky přiřazovat výpočet bodů.

### Asymptoty

Tato část se věnuje mechanismu detekce asymptot a jiných nespojitostí v grafu. Systém rozděluje vykreslování do segmentů, aby se předešlo nevhodnému spojení částí funkcí přes asymptoty.

### Intervaly

Tohle ještě neexistuje´

Např.: „sqrt((x+1)/(x-1))

(Ono tohle nevypadá, že funguje ani v geogebře, to tam asi nemaj dobře pošéfený)

# Optimalizace

## SmartStep

Při prvním hledáním optimalizací mě napadlo, že bych se měl snažit. Ale zmenšit počet bodů by vedlo k menšímu detailu. Pro tento problém jsem vyřešil řešení označené jako „SmartStep“ zkráceně „SS“. Tato řešení bylo naplánované tak, že když se bude funkce hodně měnit, tak se zvětší detail, a když ne, tak se krok zvětší a tím se ušetří výkon.

Toto řešení ale mělo svoje mouchy. První bylo potřeba zjistit, jak spočítat větší měnění funkce? Za prvé jsem si řekl, že mě zajímá pouze y, takže potřebu najít jenom části, kdy y hodně klesá nebo stoupá. K tomuto úkolu vypadá, že se hodí jednoduše spočítat delta y a to porovnávat s určitým prahem. V tomto řešení jsem viděl problém v tom, že by neřešilo lineární funkce. Kdyby totiž někdo vložil y = 100x + 1 do vstupu, tak by se to pravděpodobně dostalo do prahu. A tím by byli lineární funkce zbytečně složité.

Útěchu jsem našel ve vektorové matematice. Ta mi totiž dovoluje řešit relativní úhly, a to se skvěle hodilo pro hledání křivek. Ale jak už to tak bývá, tak toto řešení taky není ideální. Je patrně výpočetně náročnější a také neřešilo úplně ideálně můj problém-

Konečným řešením se pak už jen zdálo zkombinovat obě řešení, ale v té době už jsem svůj problém zkonzultoval s vedoucím práce. Společně jsme došli k závěru, že dělat y bude stačit, a že hold, lineární funkce bude náročnější.

Tohle se ukázalo být jako na hovno, ale delta y v pohodě stačilo

## Cache

Mým druhým optimalizačním nápadem byla cacheování dat. Toto byl dobrý nápad, a dokonce i dával v mém řešení smysl. Ale v té době jsem ještě nevěděl jak náročná jeho implementace bude. Ve skutečnosti, v době psaní tohoto textu, ještě nefunguje tato implementace úplně správně. Nyní vám popíši můj dosavadní postup.

### Session storage

Absolutně prvním nápadem, který mi proběhl hlavou je local nebo session storage. Říkal, jsem si, že to bude ideální, jednoduché řešení. Teď ale vím, že to tak není. Session storage by se měl totiž používat na ukládaní malých dat cookies atd. proto jeho velikost je maximálně 5 MB. Další nevýhodou je, že do local storage a session storage se dá uložit jen řetězec znaku (sstring). To mělo za důsledek to, že jsem musel mít ještě parse funkce a unpack funkci. Takže toto řešení jsem prohlásil za nemožné a zbytečné.

### Indexed Databese

Jako dalším řešením mého problému se zdála být na první pohled Indexed Database. Ta je také v prohlížeči a dovoluje aplikaci ukládat složitější data. Rozhodl jsem se začít s dokumentací.

V dokumentaci jsem se dočetl, že IndexedDB Je celá asynchroní, to by znamenalo, že bych si také musel zavést asynchroní programování, což si myslím že není nejlepší řešení.

Na druhou stranu jsem přemýšlel, že asynchroní programování by také mohlo pomoct výkonu.

### Zjevení - AppData + useMemo

Dosavadní jsem odprezentoval panu Stehlíkovi. Pan učitel mi doporučil, že i nemyslí, že je to dobrý nápad. Ať radši využiji useMemo a prostě si data uložím do proměnné.

Začal bych popsání hooku useMemo. Tento hook funguje tak, že zacachejuje výsledek funkce/komponenty takzavaně „Memoizuje“. Po té, když je potřeba re-render, tak se podívá jestli se nezměnili vstupní parametry uvedené v poli jako druhý parametr háčku. Když se parametry nezměnili, tak vrátí memoizovanou hodntou.

Když jsem četl tuto definici, tak jsem byl ohromen, jaká velká pomoc by to mohla být. Ihned jsem se vydal de jej.. implementace, tento hoko jakoby byl stvořen právě pro mou aplikaci.

A nyní popíši AppData, je to pouze proměnná. Ke které jsem si vytvořil funkce jako na manipulaci si touto proměnou. Mohl jsem sice manipulovat s proměnou rovnou, ale moje vnitřní já promluvilo. Myslím, že hodiny webů o view, input a data modelech přišli vhod, protože jsem je využil.

# Výzvy

V této kapitole budu psát o pronlémech které naskytly v průběhu práce a řešení.

Obsah obrázku řada/pruh, Paralelní

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obrázek :umělecké dílo, y=tanx, podle mého programu :D

Závěr

Tak jsem se dostal až na konec.

Seznam zkratek a odborných výrazů

HTML

HyperText Markup Language – značkovací jazyk používaný pro tvorbu webových stránek.

Seznam obrázků

[Obrázek 1 :umělecké dílo, y=tanx, podle mého programu :D 8](#_Toc195818665)

Použité zdroje

1. **Stehlík, Michal.** *Návod k maturitním pracím 2020.* Liberec : Albatros, 2020.

2. **W3Schools.** SVG Path. *W3Schools.* [Online] [Citace: 16. 4 2025.] https://www.w3schools.com/graphics/svg\_path.asp.

3. **Mozilla Foundation.** Paths – SVG from scratch. *MDN Web Docs.* [Online] Mozilla Foundation. [Citace: 16. 04 2025.] https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorials/SVG\_from\_scratch/Paths.

4. **REICHL, Jaroslav.** Funkce – přehled učiva. *Matematika – Jaroslav Reichl.* [Online] 2013. [Citace: 16. 4 2025.] http://www.jreichl.com/matematika/vyuka/texty/fce\_prehled.pdf.

5. **MDN Web Docs.** IndexedDB API. *MDN Web Docs.* [Online] Mozilla. [Citace: 17. 4 2025.] https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB\_API.

6. **React.** useMemo - React. *React.* [Online] Meta Platforms. Inc. [Citace: 17. 4 2025.] https://react.dev/reference/react/useMemo.

1. Seznam přiložených souborů

Na přiloženém datovém nosiči se nacházejí následující soubory a složky:

* **MP2010-Novák-Jan-L4-Tepelné\_čerpadlo.docx** – editovatelná verze dokumentace maturitní práce
* **MP2010-Novák-Jan-L4-Tepelné\_čerpadlo.pdf** – tisknutelná verze dokumentace maturitní práce
* **Výkresy** – kompletní výkresová dokumentace
* **Aplikace** – zdrojové kódy

1. Svg scalable vektor graphics [↑](#footnote-ref-1)