



**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**  
BRNO UNIVERSITY OF TECHNOLOGY

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**  
FACULTY OF INFORMATION TECHNOLOGY

**ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ**  
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

**WEBOVÝ PROHLÍŽEČ PANORAMATICKÝCH SNÍMKŮ**  
WEB-BASED PANORAMA IMAGE VIEWER

**BAKALÁŘSKÁ PRÁCE**  
BACHELOR'S THESIS

**AUTOR PRÁCE**  
AUTHOR

**TOMÁŠ SLUNSKÝ**

**VEDOUCÍ PRÁCE**  
SUPERVISOR

**Doc. Ing. MARTIN ČADÍK, Ph.D.**

**BRNO 2017**

**Vysoké učení technické v Brně - Fakulta informačních technologií**

Ústav počítačové grafiky a multimédií

Akademický rok 2016/2017

**Zadání bakalářské práce**

Řešitel: **Slunský Tomáš**

Obor: Informační technologie

Téma: **Webový prohlížeč panoramatických snímků**  
**Web-Based Panorama Image Viewer**

Kategorie: Web

**Pokyny:**

1. Provedte rešerši existujících projektů pro prezentaci sférických panoramat a videí ve webovém prohlížeči.
2. Seznamte se s technologiemi, které budete potřebovat pro vývoj, zejména s HTML5.
3. Navrhněte architekturu a uživatelské rozhraní prohlížeče sférických panoramat a videí. Prohlížeč bude zobrazovat informace o orientaci a zorném poli kamery a další metadata.
4. Implementujte prohlížeč s využitím technologie HTML5. Soustředte se na efektivní zobrazení sférických videí jak v ekvirektangulárním módu, tak v módu rybího oka.
5. Provedte uživatelské testování.
6. Dosažené výsledky prezentujte formou videa, plakátu, článku, apod.

**Literatura:**

- <http://cadik.posvete.cz/>

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese  
<http://www.fit.vutbr.cz/info/szz/>

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Čadík Martin, doc. Ing., Ph.D., UPGM FIT VUT**

Datum zadání: 1. listopadu 2016

Datum odevzdání: 17. května 2017

**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**  
Fakulta informačních technologií  
Ústav počítačové grafiky a multimédií  
612 66 Brno, Řečketěchova 2  
L.S.



doc. Dr. Ing. Jan Černocký  
vedoucí ústavu

## **Abstrakt**

Do tohoto odstavce bude zapsán výtah (abstrakt) práce v českém (slovenském) jazyce.

## **Abstract**

Do tohoto odstavce bude zapsán výtah (abstrakt) práce v anglickém jazyce.

## **Klíčová slova**

WebGL, HTML5, javascript, panoráma, sférické video, prohlížeč

## **Keywords**

WebGL, HTML5, javascript, panorama, spherical video, viewer

## **Citace**

SLUNSKÝ, Tomáš. *Webový prohlížeč panoramatických snímků*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Čadík Martin.

# Webový prohlížeč panoramatických snímků

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana X... Další informace mi poskytli... Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....  
Tomáš Slunský  
21. dubna 2017

## Poděkování

Rád bych poděkoval Martinu Čadíkovi za jeho cenné rady, vedení při tvorbě bakalářské práce a jeho celkový přístup. Dále bych rád poděkoval své rodině a blízkým za jejich trpělivost, toleranci a pochopení.

# Obsah

<b>1</b>	<b>Úvod</b>	<b>3</b>
<b>2</b>	<b>Webové technologie prohlížeče</b>	<b>4</b>
2.1	HTML5 . . . . .	4
2.1.1	Video . . . . .	4
2.1.2	Plátno . . . . .	8
2.1.3	Vektory . . . . .	9
2.2	GLSL . . . . .	10
2.2.1	Vertext shader . . . . .	10
2.2.2	Fragment shader . . . . .	10
2.2.3	Typy proměnných . . . . .	11
2.3	WebGL . . . . .	12
2.3.1	Vykreslování primitiv . . . . .	12
2.3.2	Vyrovnávací paměť . . . . .	12
2.3.3	Projekce . . . . .	13
2.3.4	Vytvoření a běh programu . . . . .	14
<b>3</b>	<b>Návrh klíčových částí</b>	<b>15</b>
3.1	Geometrie . . . . .	15
3.1.1	Výpočet . . . . .	16
3.1.2	Vertex . . . . .	17
3.1.3	Normály . . . . .	17
3.1.4	Textura . . . . .	17
3.1.5	Indices . . . . .	17
3.2	Korekce textur . . . . .	17
3.3	Projekce . . . . .	17
3.3.1	Modelová matice . . . . .	17
3.3.2	Zobrazovací matice . . . . .	17
3.3.3	Perspektivní matice . . . . .	17
3.4	Metadata . . . . .	17
3.5	Zorný úhel . . . . .	18
3.5.1	Návrh prvku . . . . .	18
3.6	Kompas . . . . .	19
3.7	Blending . . . . .	19

<b>4 Implementace panoramatického prohlížeče</b>	<b>20</b>
4.1 Grafické rozhraní . . . . .	20
4.2 Kompas . . . . .	20
4.2.1 Pozicování . . . . .	20
4.3 Zorný úhel . . . . .	21
<b>5 Testování</b>	<b>22</b>
<b>6 Závěr</b>	<b>23</b>
<b>Literatura</b>	<b>24</b>
<b>Přílohy</b>	<b>25</b>
<b>A Jak pracovat s touto šablonou</b>	<b>26</b>

# Kapitola 1

## Úvod

V současné době se technologie ve všech odvětvích stále posouvá a není tomu jinak ani u prohlížení fotek a videí. V současné době i takto nahrané informace pomocí digitálních zařízení je trend dál posouvat a zvyšovat tak zázitek ze zaznamenané události. Díky tomuto trendu si už nevystačíme s "klasickými" přehrávači, popř. prohlížeči videí, nebo fotografií. Díky sférickým 360 stupňovým kamerám, je dnes možné zaznamenat video o srovnatelné velikosti jako u dnes běžného telefonu, ale s mnohem větším objemem informací, které se ale nedají srozumitelně přehrát v klasických přehrávačích. Ruku v ruce se sdílením takových dát, je velice výhodné takový prohlížeč nabídnout jako webovou verzi, aby i jiní uživatelé mohli svá takto natočená videa přehrávat popř. sdílet. Důkazem toho je rozmach multimedialního obsahu na internetu, kdy samotný jazyk HTML pro prezentaci webových stránek dříve nenabízel přímou podporu videí, což se změnilo příchodem nové verze HTML5. Některé portály jako např. Youtube již dnes začíná nasazovat do svého prohlížení videí jeho rozšířené sférické verze, což tedy potvrzuje tento trend. Tato práce má za cíl takové prohlížení posunout ještě dál.

Následující práce se věnuje implementaci webového prohlížeče panoramatických snímků a videí v různých módech, ve kterých je video interpretováno. Cílem je tedy navrhnout a zrealizovat řešení, aby uživatel po natočení videa dále nepotřeboval k přehrání např. sférického videa v režimu rybího oka specifický program, který by video musel nejprve překonvertovat. To stejné se týká i panoramatických snímků a videí v equirectangulárním zobrazení.

Další inovací v prohlížení videí je přidání některých dat, které v běžném prohlížení nejsou k dispozici. Např. informace o světových stranách ve vztahu k obrázku, či videu popřípadě údaje o velikosti zorného pole v daném kontextu prohlížení až po dodatečná metadata v panoramatických obrázcích, které jsou vhodné např. k tomu, aby autor obrázku mohl popsat jeho zajímavé části popřípadě blíže popsat zachycenou scénu.

Samotná práce je členěna do šesti částí. V kapitole **Webové technologie prohlížeče** se budu věnovat nastínění všech použitých technologií nutných pro pochopení problematiky, se kterými se bude pracovat v následujících kapitolách. O návrhu řešení pojednává kapitola **Návrh klíčových částí**, na kterou navazuje implementační část **Implementace panoramatického prohlížeče**. Otestování správného návrhu a implementace se zabývá kapitola s názvem **Testování**. V poslední části práce **Závěr** jsou prezentovány dosažené výsledky a možnosti dalšího rozšíření.

## Kapitola 2

# Webové technologie prohlížeče

Následující kapitola pojednává o technologiích, které samotný prohlížeč používá a jsou tedy nezbytné v následujících kapitolách, zejména v části **Implementace panoramatického prohlížeče**, kde se s nimi přímo pracuje.

### 2.1 HTML5

Jazyk HTML (HyperText Markup Language) je značkovací jazyk určený pro popis webových stránek. Vychází z univerzálního značkovacího jazyka SGML (Standard Generalized Markup Language). V současné době aktuální verzi jazyka HTML je jeho již pátá verze - HTML5. Nová verze HTML přináší zásadní vylepšení, nové funkce a možnosti, které jsou nezbytné pro návrh a implementaci webového prohlížeče.

Pro návrh samotného programu je nezbytná podpora multimédií – tedy audio a video, v neposlední řadě také plátno canvas sloužící pro práci s grafikou.

#### 2.1.1 Video

Dříve nebylo možné do webových stránek vložit video tak jako dnes. K tomuto účelu bylo využíváno různých zásuvných modulů třetích stran a do webových stránek se video vkládalo např. jako objekt. Nejvíce rozšířeným přehrávačem videí a tedy jakási náhrada za podporu videí, kterou tehdy HTML nemělo, se v širším spektru stal Adobe Flash, který funkci přehrávače plní v menší míře až doposud, avšak je již zastíněn efektivnějším řešením, a to právě HTML5.

Nový prvek v HTML5 vytvořený k tomuto účelu je `<video>`. Byl navržen tak, aby mohl být použit bez detekčních skriptů na stránce. V elementu videa je možné nastavit více souborů s videem a dle podpory si daný prohlížeč vybere jím podporované video. V případě, že by prohlížeč prvek videa nepodporoval, bude jej ignorovat. Nastavení více zdrojů videa s odlišnými kodeky lze pomocí elementu `<source>` uvnitř páru elementů `<video>...</video>`. Jakmile prohlížeč narazí na `<video>`, podívá se, zdali je přítomen atribut `src`, v opačném případě začne procházet jeden element `<source>` po druhém a bude hledat právě takový, který umí přehrát.

Aby bylo možné ovládat video dynamicky pomocí kláves nebo myši, bude nutné využít DOM (Document Object Model). Jedná se o stromovou strukturu dokumentu, kterou si prohlížeč sestavuje po načtení webové stránky. Všechny elementy webové stránky jsou interpretovány v DOM jako objekt. Některé značky se v DOM vytvoří, aniž by byly ve

zdrojovém kódu zapsány. Obecně platí, že pomocí objektového modelu je díky javascriptu možné tuto stromovou strukturu dále upravovat a rozšiřovat.

### 2.1.1.1 Stavy načítání a přehrávání videa

Další velice důležitou částí je načítání videa a jeho ověření, zdali je již video připraveno k přehrání. K ověření dostupnosti videa lze použít jeden ze síťových stavů elementu pomocí stavového atributu `networkState`, popřípadě přímo zjišťovat připravenost videa pomocí atributu `readyState`, vracející jeden z následujících stavů, podle kterých se můžeme dále při přehrávání řídit a přizpůsobit tomu běh programu. Jednotlivé konstanty nabývají hodnot od 0 do 4 a dle hodnot rozlišujeme:

- **HAVE NOTHING** (ekvivalentní hodnotě 0)
  - nastane v situaci, kdy zdrojové video není dostupné, nebo žádná data pro aktuální přehrávanou pozici
  - koresponduje také s návratovou hodnotou síťové metody `networkState()`, když její návratová hodnota je rovna 0, což odpovídá konstantě `NETWORK_EMPTY`
- **HAVE\_METADATA**
  - základní data o zdroji se podařilo získat a zdroj je tedy považován jako dostupný. Zatím ale ještě není dostatek dat, aby bylo možné začít s přehráváním. V tomto stavu jsou k dispozici jen data popisující přehrávány subjekt, jako délka, šířka, dekódování apod. Dochází k vyvolání události `loadedmetadata`
- **HAVE\_CURRENT\_DATA**
  - data pro bezprostřední začátek přehrávání jsou připravena, ale video ještě není načteno dál za tuto pozici. Přehrávání může dostat do stavu `HAVE_METADATA`, nebo následující data videa již nejsou k dispozici.
- **HAVE\_FUTURE\_DATA**
  - data pro přehrávání jsou připravena pro současnou i nadcházející pozici. V případě, že by bylo video dosáhlo takového stavu poprvé, bude vyvolána událost `canplay`.
- **HAVE\_ENOUGH\_DATA**
  - data připravena pro aktuální a nadcházející pozice pro plynulé přehrání. Dochází k vyvolání události `canplaythrough`

Mimo stavy načtení videa jsou tu i stavy indikující, co přesně se právě děje s videem po jeho načtení. Tyto stavy se hodí k dotazování elementu `<video>` pomocí DOMu např. pro tvorbu vlastního specifického ovládání videa, nebo pro zahrnutí interakce s myší.

- **playing** - video se aktuálně přehrává, atribut `paused` je nastaven na `false`
- **waiting** - přehrávání videa je pozastaveno
- **ended** - přehrávání videa doběhlo nakonec
- **canplaythrough** - tato událost říká, že je možné video přehrát až do konce bez nutnosti video zastavit k dalšímu načítání. Tohoto stavu může dosahnut jen v případě, že stavový atribut `readyState` je roven `HAVE_ENOUGH_DATA`.

### 2.1.1.2 Atributy videa

Důležité atributy pro manipulaci s videem a nastavením.

**src** - jedná se o atribut, do kterého se definuje zdrojové adresa videa jako URL, které se má zobrazit. Používá se zejména v situaci, kdy je k dispozici právě jedna verze videa.

**autoplay** - jedná se atribut typu `bool`. V případě, že je `true`, spustí se přehrávání média automaticky jakmile je vše připraveno. Uvedení názvu atributu do elementu samotného je již dostačující informace o tom, co se bude dít s videem. Tedy bude tato hodnota typu `bool` považována jako `true`, jinak `false`.

**poster** - Slouží k vystihnutí obsahu daného videa. Ještě než začne samotné přehrávání, je možné jako první snímek videa zvolit obrázek zadaný cestou. V případě, že tento atribut nebude vyplněn, HTML automaticky zvolí jeden z prvních neprázdných rámciů.

**loop** - jedná se atribut typu `bool`, který začne po dosažení konce videa s přehráváním videa opět od začátku. Tato operace se provádí v nekonečné smyčce.

**width** - šířka přehrávače v pixelech.

**height** - výška přehrávače v pixelech.

**paused** - atribut videa typu `bool`. Indikuje, zdali se video přehrává či nikoliv.

**controls** - jedná se opět o atribut typu `bool`, který říká, aby prohlížeč použil vlastní ovládací prvky pro video. Ovládací prvky se dají také vytvořit a přizpůsobit velice snadno díky DOMu.

**preload** - tímto říkáme, jakou část videa by měl webový prohlížeč načíst ihned po načtení stránky. Tento atribut nabývá jednou ze tří hodnot:

- **none** - nebude načítat nic, výhodné v případě, kdy je potřeba minimalizovat vytížení pásma
- **metadata** - načte pouze metadata daného videa - základní údaje o jeho délce apod.
- **auto** - sám zvolí, co přesně udělá

### 2.1.1.3 Metody videa

Jelikož pomocí DOM je možné upravovat vlastnosti elementu `<video>`, tak jsou tedy nutné metody k ovlivnění jeho atributů, aby bylo možné s přehráváním videa manipulovat.

`play()` - metoda sloužící k nastavení atributu `paused` na `false`, pokud video již skončilo, začne je přehrávat od začátku.

`pause()` - metoda sloužící k nastavení atributu `paused` na `true`, tím dojde k pozastavení videa

`canplaytype()` - metoda sloužící k ověření, zdali webový prohlížeč dokáže video daného typu přehrát. Metoda vrací jednu z následujících hodnot:

- `""` - pokud by metoda vrátila prázdný řetězec, video prohlížeč nedokáže přehrát.
- `maybe` - prohlížeč si není zcela jist, zdali dokáže formát přehrát.
- `probably` - daný formát videa dokáže prohlížeč přehrát s vysokou pravděpodobností.

`load()` - všechny data videa se načtou znovu, čímž dojde ke zrušení všech akcí a současných dat, které byly doposud staženy a načteny. Poté se vše zavolá a načte znovu následujícím způsobem:

1. Nejprve dojde k inicializaci, kdy `readyState` je nastaven na `HAVE NOTHING` a nastaví i příslušný síťový atribut `networkState` na 0, `seeking` je nastaven na `false`, `paused` je nastaven na `true`, vše ostatní je prázdné, nebo nula
2. vybere se zdroj z atributu `src` nebo `<source>`, dále je vyvolána událost `loadstart` a dochází ke stažení metadat videa
3. jakmile jsou metadata stažena, nastaví se zakladní vlastnosti videa - šířka, výška, délka a `readyState` je nastaven na `HAVE_METADATA` a spolu s tím je vyvolána událost `loadedmetadata`
4. Jakmile je `readyState` větší nebo roven `HAVE FUTURE DATA` je vyvolána událost `canplay` a `loadeddata`
5. dojde ke spuštění přehrávání. Je vyvolána událost `play` a `playing`, atribut `paused` nastaven na `false`

## 2.1.2 Plátno

Jak již bylo avizováno výše, plátno, neboli element `<canvas>` slouží v HTML5 pro vykreslení grafů, herní grafiky, obrazů bitmap apod. Díky elementu `<canvas>` lze vykreslovat i náročnější grafické objekty za pomocí WebGL. V HTML5 slouží tedy především k vykreslení 2D prvků pomocí Javascriptu. Plátno je párový element, přesto se mezi párové tágы zdánlivě nic nevykresluje, celý obsah je skryt. Takový obsah se nazývá tzv. *fallback content* a je zobrazen v případě chyby, nebo prohlížečům, které tento element HTML5 nepodporují.

Plátno je bezpochyby nejdůležitější částí pro realizaci celé práce. Bude pro vyobrazení používat především `<video>`, z kterého bude číst data a využije tedy element video jak zdroj. Díky DOM pak dokážeme s objektem snadno manipulovat. Díky elementu `<canvas>` jsme tedy schopni získat kontext Javascriptového API - WebGL a díky němu schopni pracovat na úrovni grafické karty.

### 2.1.2.1 Atributy plátna

Canvas má pouze dva atributy a těmi jsou `width` a `height` pro nastavení šířky a výšky plátna. Je možné nastavit výšku a šířku pomocí kaskádových stylů, avšak tato možnost není doporučována, protože by mohlo dojít k nežádoucí deformaci obsahu. Při změně velikosti bitmapy pomocí atributů dojde pouze ke změně velikosti dané bitmapy, změna kaskádovými styly ale změní velikost obsahu celé bitmapy a tím tedy dojde ke zkreslení.

### 2.1.2.2 Použití a práce s plátnem

Použit plátno lze pomocí rozhraní `HTMLCanvasElement`, které slouží pro přístup k jeho vlastnostem a metodám. Přístup k objektu `HTMLCanvasElement` lze pomocí standardní javascriptové metody `getElementById()` v případě, že je v tágovi `<canvas>` nastaven atribut `id`, nebo lze objekt získat pomocí metody `querySelector("canvas")`.

Po získání objektu plátna lze dále s elementem manipulovat pomocí dvou metod (počet volání dané metody vrátí pokaždé ten stejný objekt):

- `toDataURL([type[, quality]])`
  - vrací URL adresu aktuálního obrázku v elementu `<canvas>`. Metoda může a nemusí mít nějaké argumenty. První argument, v případě že je zadán, rozhoduje jaký bude navrácený typ obrázku. Pokud nezadáme žádný argument, tak jako výchozí formát obrázku bude automaticky zvolen *image/png*. Výsledná adresa URL bude vrácena jako řetězec.
- `getContext(contextId[, ... ])`
  - metoda vrací referenci na daný objekt aplikačního rozhraní, díky kterému bude možné kreslit na plátno. Pokud by kontext pro daný argument nebyl podporován prohlížečem, návratová hodnota bude `null`. Typ aplikačního rozhraní je vybrán na základě argumentu metody:
    - **"2d"** - vrací objektové rozhraní `CanvasRenderingContext2D`, kterí slouží pro kreslení grafických primitiv.
    - **"webgl"** - pokud tuto funkci prohlížeč podporuje, bude navrácen objekt `WebGLRenderingContext`.

### 2.1.3 Vektory

SVG (Scalable Vector Graphics) - škálovatelná vektorová grafika je již zařízený standard. Doposud ale nebylo možné kód `svg` vkládat přímo do HTML. Změna přišla až s HTML5. SVG vychází z jazyka XML (eXtensible Markup Language)<sup>1</sup>. Mezi jeho nesporné výhody patří velikost, přenositelnost a nezávislost na rozlišení aj. Do HTML se vkládá pomocí elementu `<svg>`, který má zpravidla dále zanořené elementy, které reflektují jeho obsah. Mezi takové vnořené elementy patří také tág `<path>`.

#### 2.1.3.1 Element path

Generický element `<path>` slouží ve vektorové grafice k vytvoření křivky, která může být dále vyplněna barvou, nebo může sloužit k vytvoření ořezové cesty apod. Dále může mít nastaveny atributy specifikující vykreslování obrysů, nastavení stylů a tříd, nebo nastavení kalkulace délky cest. Mezi takové atributy patří:

**d** - tento atribut nese informace o cestě, resp. data k vykreslení obrysů, který poté můžeme vyplnit konkrétní barvou, popř. jej využít k dalším podobným účelům viz výše. Data atributu se skládají z následujících částí:

- **M/m (moveto)** její syntaxe: `M x y` nebo `m x y` - realizuje posun na danou dvojici souřadnic (`x`, `y`) bez kreslení. Velké písmeno `M` značí, že souřadnice budou absolutní. Malé písmeno `m` indikuje, že budou souřadnice relativní<sup>2</sup>. Tuto vlastnost je velice výhodné používat pro všechny cesty, jinak by vždy následující čára začínala na konci té předchozí.
- **Z/z (closepath)** - tento příkaz je bez parametrů a slouží k uzavření cest, tedy spojení posledního bodu s prvním bodem vykreslovaných cest.
- **L/l (lineto)** - tento příkaz již kreslí čáru, jeho syntaxe je stejná jako u příkazu `moveto` tj `L/l x,y`
- **H/h (horizontal lineto)** - kreslí horizontální čáru z bodu (`cpx, cpy`) do (`x, cpy`), argument příkazu je pouze souřadnice `x`
- **V/v (vertical lineto)** - kreslí vertikální čáru z bodu (`cpx, cpy`) do (`cpx, y`), argument příkazu je pouze souřadnice `y`
- **A/a (elliptical arc)** - parametry: `(rx ry x-axis-rotation large-arc-flag sweep-flag x y)` kde `rx` značí první poloměr elipsy, `ry` druhý poloměr, `x-axis-rotation` rotace kolem osy x, příznak `large-arc-flag` značí, zdali bude oblouk krátký (hodnota 0) popř. dlouhý (hodnota 1), dále argument `sweep-flag` příznak značící oblouk proti směru/po směru (0/1), `x` a `y` souřadnice koncového bodu

**fill** - vyplní křivku zadánou argumentem `d` libovolnou barvou

---

<sup>1</sup>obecný značkovací jazyk pro serializaci dat

<sup>2</sup>Velikost písmen má stejný význam u všech příkazů

## 2.2 GLSL

**GLSL** (OpenGL Shading Language) - jedná se o jazyk pro psaní shaderů<sup>3</sup>. Jazyk vychází z jazyka C a také proto je mu svojí syntaxí velice podobný. Základní konstrukce každého programu je v zásadě úplně stejná jako v jazyku C, každý program musí obsahovat hlavní funkci `main()` bez jakýkoliv parametrů a návratových hodnot. V hlavní funkci programu jsou vyhrazeny proměnné se speciálním významem, jako např. `gl_Position`, kterou nastavujeme pozici vrcholu ve vertex shaderu, nebo `gl_FragColor` pro nastavení barvy ve fragment shaderu.

### 2.2.1 Vertext shader

Jedná se o program v jazyce GLSL, který je pro prováděn pro každý vstupní vrchol zadáné geometrie.

Každá souřadnice vrcholu je dále přeypočítána do tzv. *clipspace*, což je souřadný systém pro vykreslování do plátna. V případě, že by některé souřadnice byly mimo hranice souřadného systému, budou oříznuty a vykreslovat se nebudou. Hranice *clipspace* jsou definovány pro každou ze souřadnic `x, y, z` intervalom  $<-1, 1>$ . Vertex shader může sám ovlivnit souřadnice vrcholů, textur popř. normálových vektorů a barev. Při zpracování vrcholů není známa topologie zpracovávaného objektu, protože nejsou dostupné informace o sousedících vrcholech.

Poslední operací vertex shaderu před samotným vykreslením je převod do NDC<sup>4</sup>. Rozdílné operační systémy mají rozdílné způsoby reprezentace grafických objektů. Zabránit těmto rozdílným interpretacím v praxi znamená, že se souřadnice `x, y` a `z` vydělí homogenní souřadnici.

### 2.2.2 Fragment shader

Je program - označovaný taktéž jako *pixel shader*, který je volán pro každý pixel vykreslované scény. Jeho úkolem je výpočet barvy každého pixelu a díky tomu lze s takto získanou barvou dále pracovat. Barva je získána jako čtyřsložkový vektor, první tři složky tvoří paletu RGB, poslední čtvrtá složka tvoří  $\alpha$  kanál<sup>5</sup>. V případě vykreslování složitějších objektů umožňuje GLSL automaticky body geometrie interpolovat, tedy automaticky získat pixel dané geometrie, aniž by musel být každý samostatně definován. Ukázka interpolace je demonstrována na obrázku 2.1.

---

<sup>3</sup>Jedná se o programy řídící vykreslování na grafické kartě

<sup>4</sup>Normalized Device Coordinates

<sup>5</sup>Složka pixelu udávající jeho průhlednost

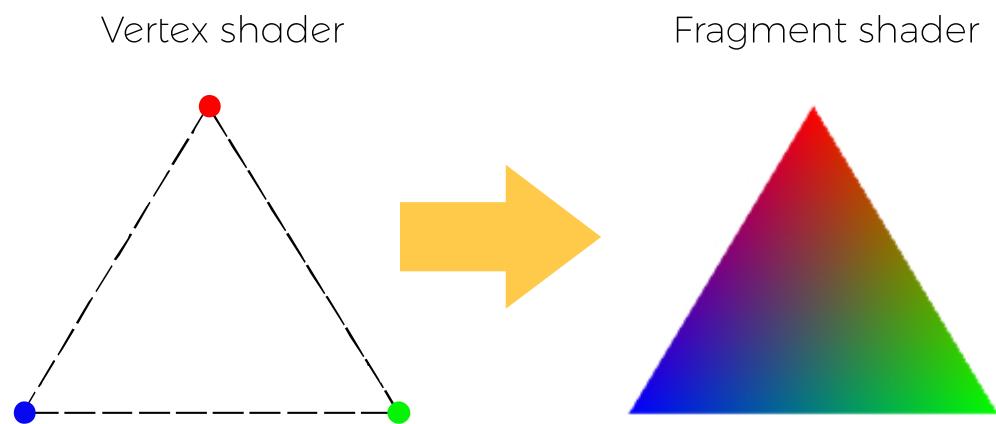
### 2.2.3 Typy proměnných

Předávání hodnot v globálních proměnných může probíhat za pomocí třech datových typu, těmi jsou:

**uniform** - uchovává hodnoty, které jsou konstantní po celou dobu běhu programu. Využívá se např. pro data transformačních matic apod.

**attribute** - jedná se o vstupní data vertex shaderu, která mění svoje hodnoty pro každý vrchol.

**varying** - tento datový typ slouží k předávání hodnot z vertex shaderu do fragment shaderu. Pomocí něj se provádí výše avizovaná interpolace, viz. obr. 2.1



Obrázek 2.1: Ukázka interpolace hodnot mezi vertex a fragment shaderem.

## 2.3 WebGL

WebGL (**W**ebsite **GL**ibrary) je javascriptové aplikační rozhraní, které slouží pro nativní vykreslování 2D a 3D interaktivní grafiky přímo z webového prohlížeče. K samotnému běhu WebGL není potřeba žádných dodatečných knihoven, nebo zásuvných modulů, protože je v tomto ohledu využívána přímo grafická karta. Program WebGL se v zásadě skládá z více jazyků, přičemž každý má svůj vlastní účel a je nutné je používat dohromady. Jedním z nich je javascript, druhý je jazyk GLSL. V jazyce GLSL jsou napsány dva programy - vertex shader a fragment shader. Oba programy kompiluje ovladač grafické karty do kódu, který je poté vykonáván přímo na grafické kartě. Vertex shader i fragment shader mohou být předávány jako řetězce, takže manipulace s nimi je díky tomu značně zjednodušena. Velkou výhodou WebGL je integrace s DOM tudíž od automatické správy pamětí přes zpracování událostí až po jednoduché načítání bitmapy přímo v prostředí prohlížeče apod..

K samotnému vykreslení je použit již výše zmíněný element HTML5 <canvas>.

V současné době je oficiálně vydána verze WebGL 1.0, která je založeno na OpenGL ES 2.0. Doposud se jedná o jedinou stabilní verzi. OpenGL ES<sup>6</sup> je zjednodušená verze OpenGL, která eliminuje většinu vykreslovacích pipeline. Tato verze je především určena pro menší vestavěné systémy. Dalším rozšířením WebGL bude verze 2.0, která se stále nachází ve vývoji. Bude založena na OpenGL ES 3.0 přinášející velká vylepšení v zobrazování 3D grafiky - zejména textur, akceleraci vizuálních efektů, novou verzí jazyka GLSL s plnou podporu desetinných a celočíselných operací apod.

### 2.3.1 Vykreslování primitiv

K vykreslení nějakého grafického prvku jsou potřeba v zásadě jen dvě věci - vrcholy pře-počítané do tzv. *clipspace* a jemu přiřazené odpovídající barvy. Jak pro vrcholy tak pro barvy slouží jiný GLSL shader. K vyobrazení objektu používá WebGL tří základní primitiva - vykreslení samotných bodů, čar nebo trojúhelníku. Vše funguje tak, že zadaný bod, kterému budeme chtít přiřadit barvu, se načte do odpovídající speciální globální proměnné ve vertex shaderu, tam se daná hodnota přepočítá do intervalu  $<-1, 1>$  pro všechny jeho osy a vzniklá hodnota se uloží na grafické kartě. Počet bodu záleží na primitivu, které vykreslujeme. Pokud se tedy jedná o bod, stačí pouze jeden bod ve vertex shaderu, pokud se jedná o čáru tak dvě popř. trojúhelník, tak trojice bodů. GPU<sup>7</sup> poté zjistí, které pixely korespondují s vykreslovaným primitivem a pro každý takový pixel zavolá fragment shader.

### 2.3.2 Vyrovňávací paměť

Buffery resp. vyrovňávací paměti jsou rychle přístupné paměti na grafické kartě. V bufferech jsou uloženy potřebné data k vykreslování grafických objektů jako např. body geometrie, normály, mapování textur apod. Data jsou poté snadno dále předávány GLSL shaderům, které s nimi pracují.

---

<sup>6</sup>OpenGL for Embedded Systems

<sup>7</sup>Graphic Processing Unit - grafický procesor

### 2.3.3 Projekce

Některé frameworky<sup>8</sup> založené na WebGL zaměňují projekci za kameru, ta bohužel jako taková ve WebGL neexistuje. Jedná se pouze o způsob vykreslení grafické informace, ke které jsou využívány právě projekční zobrazení. WebGL podporuje dva základní typy projekce - ortografickou a perspektivní.

Ortografická projekce využívá skutečné velikosti objektu. Scéna je reprezentována jako pravoúhlý hranol.

Perspektivní projekce zkresluje velikost objektů způsobem, jak je tomu v reálném světě. Čím je objekt dál od pozorovatele, tím se bude jevit jako menší, protože jeho pozice je středem této projekce.

Pokud bychom chtěl vykreslit nejaký 3D objekt, bude potřeba mít v bufferech informace o vrcholech, normálách, textuře apod. Tyto data bychom zároveň museli neustále aktualizovat a nahrávat zpět do paměti, aby bylo možné s daným 3D objektem jakkoli manipulovat, posouvat apod. Aby nedocházelo k neustálému přepisování hodnot v paměti na GPU, veškeré změny zrealizujeme transformačními maticemi, kterými se bude geometrie objektu v paměti násobit, aniž by data v paměti byla jakkoli změněna.

#### 2.3.3.1 Homogenní souřadnice

Homogenní souřadnice je tvořena čtyřmi prvky -  $x$ ,  $y$ ,  $z$  a  $w$ . První tři složky tvoří souřadnice euklidovského prostoru, poslední prvek  $w$  je perspektivní složka a spolu se souřadnicemi  $x$ ,  $y$ ,  $z$  tvoří projekční prostor.

#### 2.3.3.2 Model matrix

Jedná se o matici, kterou držíme pro každý vykreslovaný objekt. Stará se o úpravu tvaru, natočení a posuvu objektu.

#### 2.3.3.3 View matrix

Díky této transformační matici realizujeme již avizovanou "roli kamery", kdy simulujeme vzájemná vztah scény a pozorovatele.

#### 2.3.3.4 Perspective matrix

Stará se o perspektivní zkreslení scény. Tato transformace rozhoduje o tom, jak velké zorné pole bude vykresleno a mapováno na obrazovku monitoru.

---

<sup>8</sup>Rámcová softwarová struktura usnadňující řešení dané problematiky

### 2.3.4 Vytvoření a běh programu

Po vytvoření kontextu `webgl` je již možné využívat funkce pro práci s WebGL. Funkce by se daly rozdělit na několik částí, nejprve ty, které pracují s shadery, dále ty které vytváří program, v neposlední řadě funkce pro prací s buffery a vykreslením dat. Všechny funkce WebGL pracují nad vytvořeným kontextem plátna<sup>9</sup>.

`gl.createShader(type)` - vytvoří objekt shaderu, vstupním argumenty `type` mohou být `gl.VERTEX_SHADER` nebo `gl.FRAGMENT_SHADER`

`gl.shaderSource(shader, source)` - asociouje objekt shaderu `shader` se zdrojovým kódem samotného shaderu `source` v textové formě

`gl.compileShader(shader)` - kompiluje GLSL kód na binární data, vstupním parametrem je objekt shaderu `shader`

Následuje práce s programem. Funkce `gl.createProgram()` vytvoří objekt programu. Poté `gl.attachShader(program, shader)` - přiloží daný shader k programu. Funkce `gl.linkProgram(program)`, která slinkuje vše dohromady, `gl.useProgram(program)` - nastaví daný webgl program jako část současného vykreslovacího stavu.

`gl.getAttribLocation(program, attribute)` - vrací referenci na atribut v vertex shaderu

`gl.enableAttribute(attribute)` - aktivuje atribut

`gl.createBuffer()` - vytvoří nový buffer objekt pro data

`gl.bindBuffer(target, buffer)` - nastaví vytvořený buffer `buffer` jako aktivní, parametr `target` specifikuje data s jakými se bude pracovat

`gl.bufferData(target, srcData, usage)` - parametr `target` má stejný význam jako u funkce `bindBuffer`, `srcData` určuje data, která se nahrají do bufferu, poslední parametr `usage` specifikuje způsob využití uložených dat

`gl.vertexAttribPointer(attrib, size, type, normalized, stride, offset)` - attrib prováže atribut programu s aktivním bufferem, `size` určuje, z kolika prvků se skládají data v bufferu, `type` spcecifikuje datový typ dat v bufferu, `normalized` určuje zdali budou hodnoty normalizovány, parametr je typu `bool`, dále `stride` udávající zdali budeme nějaké hodnoty přeskakovat, `offset` odkud budeme data číst.

Samotné vykreslení uložených data v bufferech se realizuje funkcemi:

`gl.drawArrays(mode, first, count)` - vykresluje vstupní data. Parametr `mode` určuje, jaké grafické primitivum budem použito pro vykreslení, `first` určuje odkud se data zažnou upracovávat, `count` specifikuje počet vykreslovaných hodnot

`gl.drawElements(mode, count, type, offset)` - stejně jako u `drawArrays()` určuje vykreslované grafické primitivum, to stejně se týká i parametru `count`, `type` definuje typ dat indexů a `offset` odkud začínáme zpracovávat indexy. HLavním rozdílem vůči funkci `drawArrays()` je ten, že `gl.drawElements(...)` nevykresluje body přímo, ale dostává jako vstupní data indexy na jednotlivé body, které vykresluje funkce `drawArrays()`

---

<sup>9</sup>Vytvořený WebGL kontext budeme označovat jako `gl`

## Kapitola 3

# Návrh klíčových částí

Kapitola nastiňuje, jakým způsobem jsou řešeny nejzásadnější problémy implementace prohlížeče. Snaží se navrhnut ty nejfektivnější způsoby řešení daných problémů a v případě, že tomu tak v konkrétním případě není, snaží se tyto kroky návrhu jasně odůvodnit, proč jsou daným způsobem takto řešeny.

### 3.1 Geometrie

Geometrie je potřebná k mapování textury, a to tak, aby daná textura reflektovala scénu co nejvěrněji. Kdybychom pro texturu nevytvářeli geometrii, na kterou se bude mapovat a jednoduše data namapovali do libovolné 2D roviny, dojde k významnému zkreslení pořizované scény.

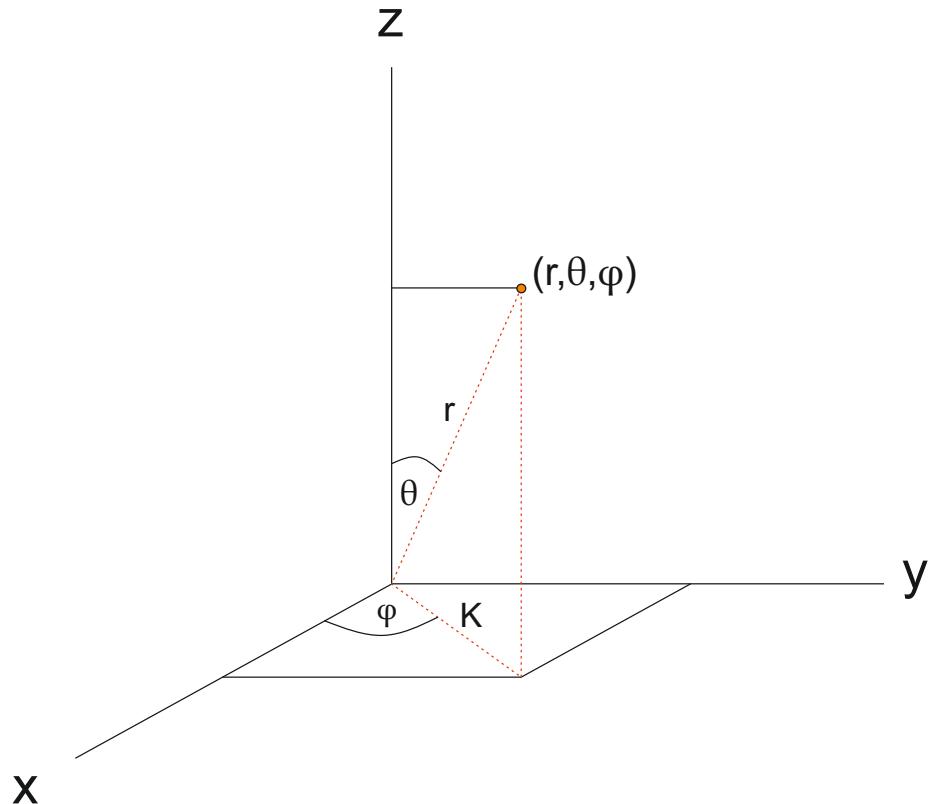
K vyobrazení geometrie budeme využívat klasické pole, kde každý bod bude tvořen třemi rozměry v kartézske soustavě souřadnic, tedy formátu  $[x_1, y_1, z_1, x_2, y_2, z_2, \dots]$ , které utváří jednotlivé body v 3D prostoru. Body pak mezi sebou tvoří prostor, na který budou texturovací data mapovaná. Ve WebGL umíme vykreslit jednotlivé body, čáry – tedy spojnice jednotlivých bodů, nebo trojúhelníky. V našem případě bude potřeba vykreslit texturu jako plochu, na kterou se bude vše mapovat. Pro ten to případ se ve WebGL používá vykreslení pomocí trojúhelníku, jako základní vykreslované primitivum plochy.

Geometrickým tělesem pro mapování bude zvolena koule, protože scéna resp. pořízená data, která budeme zpracovávat jsou zachycena ve sférickém modu a mapováním na kouli je dosažena nejvěrnější reprezentace scény.

Kouli so rozdělíme na poledníky a rovnoběžky. Poledníkem je v tomto případě myšlena spojnice severního a jižního pólu, kde úhel z počátku kartézske soustavy souřadnic mezi jednotlivými poledníky, nabývá hodnot od  $0\text{--}360^\circ$ . Rovnoběžkou je myšlena kružnice svírající úhel s počátkem souřadnic od  $0\text{--}180^\circ$ .

Je třeba tedy nejprve stanovit počet poledníků a rovnoběžek a z takových hodnot bude dále vypočítán úhel. Na základě úhlu budeme schopni přesně definovat bod v trojrozměrné rovině. Čím větší počet rovnoběžek a poledníků, tím bude geometrie věrněji approximovat kouli - zvýší se hustota bodů v bufferech a tím bude i samotné vykreslení náročnější. Proto je ideální kompromis mezi jemností bodů a rychlosti načítání s téměř stejným výsledkem. Tato vlastnost by se nám hodila hlavně tedy v případech, kdy bychom měli vykreslit rozmanitý geometrický objekt se spoustou detailních sekvencí. V našem případě tento rozdíl tedy nepůjde okem rozeznat, takže skrze efektivitu je výhodné spíše ubrat náročnost, čímž dosáhneme pozitivního efektu na samotnou rychlosť programu.

Každá rovnoběžka má po svém obvodu průsečíky s jednotlivými poledníky, tyto průsečíky jsou naše body geometrie, které budou nahrány do bufferů. Jelikož každá rovnoběžka nabývá úhlu od 0 do  $180^\circ$  resp.  $0 - \pi$ , tak tento úhel můžeme podělit počtem rovnoběžek a vynásobit aktuální rovnoběžkou, čímž získáme svíraný úhel s počátkem souřadnic. Provedením této operace pro všechny horizontální obvodové kružnice, vypočítáme všechny body geometrie.



Obrázek 3.1: Výpočet jednotlivých bodů geometrie

### 3.1.1 Výpočet

Z obrázku 3.1

Abychom mohli vytvořit naší geometrii, musíme spočítat body  $[x,y,z]$  pro všechny poledníky a rovnoběžky. Každá vertikální čára od severního pólu k jižnímu (poledník) se protíná s našimi rovnoběžkami. Jelikož poledník nabývá od severního polu k jižnímu  $0^\circ - 180^\circ$ , tak podělením tohoto úhlu počtem rovnoběžek geometrie nám vzniká rovnoměrné rozložení bodů po celém poledníku. Takto máme vyřešen jeden poledník a průnik rovnoběžek, dále je potřeba tuto operaci provést pro všechny poledníky, aby byl model kompletní. Součet úhlů všech poledníku nabývá hodnot od  $0-360^\circ$ , tudíž šířka jednoho poledníku a tedy úhel mezi dvěma vertikálními čarami od počátku soustavy souřadnic X,Y,Z je roven podělení  $360^\circ$  všemi poledníky. Tím dojde k approximaci celé koule, na kterou budeme mapovat naše data.

**3.1.2** Vertex

**3.1.3** Normály

**3.1.4** Textura

**3.1.5** Indices

## **3.2** Korekce textur

### **3.3** Projekce

**3.3.1** Modelová matici

**3.3.2** Zobrazovací matice

**3.3.3** Perspektivní matice

### **3.4** Metadata

## 3.5 Zorný úhel

Zorný úhel je grafický prvek, který udává informace o aktuálním zorném poli zobrazovací matice. Tento úhel se mění na základě přibližování, nebo oddalování scény od pozorovatele.

### 3.5.1 Návrh prvku

K vyobrazení prvku budeme potřebovat vektor svg zobrazující přímý úhel. Reprezentován bude tvarem polokoule, u které se využije vlastnosti tahu. Ve vektorové grafice se jedná o nastavení atributu **stroke-width**, který bude reprezentovat zorný úhel. Důvodem, proč pro vyobrazení samotné není využita samotná křivka vyplněná barvou je ten, že pokud se přenastaví úhel zobrazení, tak se křivka zdeformuje tak, že z ní není zjevný zorný úhel.

Vstupními argumenty je pozice prvku pomocí ve dvojrozměrném prostoru  $(x, y)$  a průměr kruhu. Dále je potřeba přepočítat tyto souřadnice dvojrozměrného prostoru na sférické souřadnice. K samotnému výpočtu dostačuje Pythagorova věta, kdy pomocí bodu  $(x, y)$  jsou známé hodnoty odvesen a je možný spočítat přeponu. Takto odvodíme vztahy pro  $x$  a  $y$

$$x = r \cdot \cos(\theta)$$

$$y = r \cdot \sin(\theta)$$

**3.6 Kompas**

**3.7 Blending**

# Kapitola 4

## Implementace panoramatického prohlížeče

### 4.1 Grafické rozhraní

Veškeré prvky, které nejsou vykreslovány skrze grafickou kartu, se v elementu jak již bylo nastíněno v předcházejících kapitolách, <canvas> nezobrazí z důvodu, že je vše skryto. Jen v případě, že by prohlížeč nepodporoval plátno, zobrazí co je v něm. Realizaci samotných prvků jako je kompas, nebo zorný úhel až po ovládací prvky videa, bude třeba realizovat za pomocí kaskádových stylů, kdy pomocí dynamického pozicování budeme všechny prvky zobrazovat překryvem plátna.

### 4.2 Kompas

Prvek překrývající plátno <canvas>. Vstupními metadaty jsou zadány koordinační body severu, popř. jihu. Díky interakcím s myší - tedy násobení zobrazovací transformační maticí realizujeme pohyb pozorovatele, čímž se mění prostorové úhel, který ovlivňuje odchylku severu, popř. jihu od původní zadанé pozice v metadatech.

Samotná konstrukce kompasu je zasazena do tagu <div id="compass-box">. Funkce, která nahraje data do kontejneru kompasu se nazývá `createCompass()`.

#### 4.2.1 Pozicování

Vytvoření samotného kompasu realizuje funkce `createCompass(width, height)`, argument `width` udává šířku kompasu, argument `height` zase jeho výšku. Pozice kompasu v plátně je realizována pomocí nastavení hodnot `div.style.left` a `div.style.top` v pixelech, kde se bude udávat výsledná hodnota posuvu. Pozici samotnou nastavuje řádek `div.style.position = 'absolute';`. Posun od levého okraje obrazovky `div.style.left` je vypočten tak, že si načteme celkovou šířku

### **4.3 Zorný úhel**

Pravidla [1]. Pravidla [2]. Goniometrie

## **Kapitola 5**

### **Testování**

# **Kapitola 6**

## **Závěr**

Závěrečná kapitola obsahuje zhodnocení dosažených výsledků se zvlášť vyznačeným vlastním přínosem studenta. Povinně se zde objeví i zhodnocení z pohledu dalšího vývoje projektu, student uvede náměty vycházející ze zkušeností s řešeným projektem a uvede rovněž návaznosti na právě dokončené projekty.

# Literatura

- [1] Katka Maria Delventhal, M. K., Alfred Kissner: *Kompendium matematiky: vzorce a pravidla : četné příklady včetně řešení : od základních operací po vyšší matematiku.* Euromedia Group k.s - Knižní klub v edici Universum, 2004, ISBN 80-242-1227-7.
- [2] ODVÁRKO, O.: *Matematika pro gymnázia.* Prometheus, 2008, ISBN 978-80-7196-359-2.

# Přílohy

## Příloha A

# Jak pracovat s touto šablonou

V této kapitole je uveden popis jednotlivých částí šablony, po kterém následuje stručný návod, jak s touto šablonou pracovat.

Jedná se o přechodnou verzi šablony. Nová verze bude zveřejněna do konce roku 2016 a bude navíc obsahovat nové pokyny ke správnému využití šablony, závazné pokyny k vypracování bakalářských a diplomových prací (rekapitulace pokynů, které jsou dostupné na webu) a nezávazná doporučení od vybraných vedoucích. Jediné soubory, které se v nové verzi změní, budou projekt-01-kapitoly-chapters.tex a projekt-30-prilohy-appendices.tex, jejichž obsah každý student vymaže a nahradí vlastním. Šablonu lze tedy bez problémů využít i v současné verzi.

### Popis částí šablony

Po rozbalení šablony naleznete následující soubory a adresáře:

**bib-styles** Styly literatury (viz níže).

**obrazky-figures** Adresář pro Vaše obrázky. Nyní obsahuje placeholder.pdf (tzv. TODO obrázek, který lze použít jako pomůcku při tvorbě technické zprávy), který se s prací neodevzdává. Název adresáře je vhodné zkrátit, aby byl jen ve zvoleném jazyce.

**template-fig** Obrázky šablony (znak VUT).

**fitthesis.cls** Šablona (definice vzhledu).

**Makefile** Makefile pro překlad, počítání normostran, sbalení apod. (viz níže).

**projekt-01-kapitoly-chapters.tex** Soubor pro Váš text (obsah nahraďte).

**projekt-20-literatura-bibliography.bib** Seznam literatury (viz níže).

**projekt-30-prilohy-appendices.tex** Soubor pro přílohy (obsah nahraďte).

**projekt.tex** Hlavní soubor práce – definice formálních částí.

Výchozí styl literatury (czechiso) je od Ing. Martínka, přičemž anglická verze (englishiso) je jeho překladem s drobnými modifikacemi. Oproti normě jsou v něm určité odlišnosti, ale na FIT je dlouhodobě akceptován. Alternativně můžete využít styl od Ing. Radima Loskota

nebo od Ing. Radka Pyšného<sup>1</sup>. Alternativní styly obsahují určitá vylepšení, ale zatím nebyly řádně otestovány větším množstvím uživatelů. Lze je považovat za beta verze pro zájemce, kteří svoji práci chtějí mít dokonalou do detailů a neváhají si nastudovat detaily správného formátování citací, aby si mohli ověřit, že je vysázený výsledek v pořádku.

Makefile kromě překladu do PDF nabízí i další funkce:

- přejmenování souborů (viz níže),
- počítání normostran,
- spuštění vlny pro doplnění nezlomitelných mezer,
- sbalení výsledku pro odeslání vedoucímu ke kontrole (zkontrolujte, zda sbalí všechny Vámi přidané soubory, a případně doplňte).

Nezapomeňte, že vlna neřeší všechny nezlomitelné mezery. Vždy je třeba manuální kontrola, zda na konci řádku nezůstalo něco nevhodného – viz Internetová jazyková příručka<sup>2</sup>.

**A.0.0.0.1 Pozor na číslování stránek!** Pokud má obsah 2 strany a na 2. jsou jen „Přílohy“ a „Seznam příloh“ (ale žádná příloha tam není), z nějakého důvodu se posune číslování stránek o 1 (obsah „nesedí“). Stejný efekt má, když je na 2. či 3. stránce obsahu jen „Literatura“ a je možné, že tohoto problému lze dosáhnout i jinak. Řešení je několik (od úpravy obsahu, přes nastavení počítadla až po sofistikovanější metody). **Před odevzdáním proto vždy překontrolujte číslování stran!**

## Doporučený postup práce se šablonou

1. **Zkontrolujte, zda máte aktuální verzi šablony.** Máte-li šablonu z předchozího roku, na stránkách fakulty již může být novější verze šablony s aktualizovanými informacemi, opravenými chybami apod.
2. **Zvolte si jazyk**, ve kterém budete psát svoji technickou zprávu (česky, slovensky nebo anglicky) a svoji volbu konzultujte s vedoucím práce (nebyla-li dohodnuta předem). Pokud Vámi zvoleným jazykem technické zprávy není čeština, nastavte příslušný parametr šablony v souboru projekt.tex (např.: `documentclass[english]{fitthesis}`) a přeložte prohlášení a poděkování do angličtiny či slovenštiny.
3. **Přejmenujte soubory.** Po rozbalení je v šabloně soubor projekt.tex. Pokud jej přeložíte, vznikne PDF s technickou zprávou pojmenované projekt.pdf. Když vedoucímu více studentů pošle projekt.pdf ke kontrole, musí je pracně přejmenovávat. Proto je vždy vhodné tento soubor přejmenovat tak, aby obsahoval Váš login a (případně zkrácené) téma práce. Vyhnete se však použití mezer, diakritiky a speciálních znaků. Vhodný název tedy může být např.: „xlogin00-Cistení-a-extrakce-textu.tex“. K přejmenování můžete využít i přiložený Makefile:

```
make rename NAME=xlogin00-Cistení-a-extrakce-textu
```

---

<sup>1</sup>BP Ing. Radka Pyšného <http://www.fit.vutbr.cz/study/DP/BP.php?id=7848>

<sup>2</sup>Internetová jazyková příručka <http://prirucka.ujc.cas.cz/?id=880>

4. Vyplňte požadované položky v souboru, který byl původně pojmenován projekt.tex, tedy typ, rok (odevzdání), název práce, svoje jméno, ústav (dle zadání), tituly a jméno vedoucího, abstrakt, klíčová slova a další formální náležitosti.
5. Nahraďte obsah souborů s kapitolami práce, literaturou a přílohami obsahem své technické zprávy. Jednotlivé přílohy či kapitoly práce může být výhodné uložit do samostatných souborů – rozhodnete-li se pro toto řešení, je doporučeno zachovat konvenci pro názvy souborů, přičemž za číslem bude následovat název kapitoly.
6. Nepotřebujete-li přílohy, zakomentujte příslušnou část v projekt.tex a příslušný soubor vyprázdněte či smažte. Nesnažte se prosím vymyslet nějakou neúčelnou přílohu jen proto, aby daný soubor bylo čím naplnit. Vhodnou přílohou může být obsah přiloženého paměťového média.
7. Nascanované zadání uložte do souboru zadani.pdf a povolte jeho vložení do práce parametrem šablony v projekt.tex (`documentclass[zadani]{fitthesis}`).
8. Nechcete-li odkazy tisknout barevně (tedy červený obsah – bez konzultace s vedoucím nedoporučuji), budete pro tisk vytvářet druhé PDF s tím, že nastavíte parametr šablony pro tisk: (`documentclass[zadani,print]{fitthesis}`). Barevné logo se nesmí tisknout černobíle!
9. Vzor desek, do kterých bude práce vyvázána, si vygenerujte v informačním systému fakulty u zadání. Pro disertační práci lze zapnout parametrem v šabloně (více naleznete v souboru fitthesis.cls).
10. Nezapomeňte, že zdrojové soubory i (obě verze) PDF musíte odevzdat na CD či jiném médiu přiloženém k technické zprávě.

## Pokyny pro oboustranný tisk

- Zapíná se parametrem šablony: `\documentclass[twoside]{fitthesis}`
- Po vytisknutí oboustranného listu zkонтrolujte, zda je při prosvícení sazební obrazec na obou stranách na stejně pozici. Méně kvalitní tiskárny s duplexní jednotkou mají často posun o 1–3 mm. Toto může být u některých tiskáren řešitelné tak, že vytisknete nejprve liché stránky, pak je dáte do stejného zásobníku a vytisknete sudé.
- Za titulním listem, obsahem, literaturou, úvodním listem příloh, seznamem příloh a případnými dalšími seznamy je třeba nechat volnou stránku, aby následující část začínala na liché stránce (`\cleardoublepage`).
- Konečný výsledek je nutné pečlivě překontrolovat.

## Užitečné nástroje

Následující seznam není výčtem všech využitelných nástrojů. Máte-li vyzkoušený osvědčený nástroj, neváhejte jej využít. Pokud však nevíte, který nástroj si zvolit, můžete zvážit některý z následujících:

**MiKTeX** L<sup>A</sup>T<sub>E</sub>X pro Windows – distribuce s jednoduchou instalací a vynikající automatizací stahování balíčků.

**TeXstudio** Přenositelné opensource GUI pro L<sup>A</sup>T<sub>E</sub>X. Ctrl+klik umožňuje přepínat mezi zdrojovým textem a PDF. Má integrovanou kontrolu pravopisu, zvýraznění syntaxe apod. Pro jeho využití je nejprve potřeba nainstalovat MikTeX.

**JabRef** Pěkný a jednoduchý program v Java pro správu souborů s bibliografií (literaturou). Není potřeba se nic učit – poskytuje jednoduché okno a formulář pro editaci položek.

**InkScape** Přenositelný opensource editor vektorové grafiky (SVG i PDF). Vynikající nástroj pro tvorbu obrázků do odborného textu. Jeho ovládnutí je obtížnější, ale výsledky stojí za to.

**GIT** Vynikající pro týmovou spolupráci na projektech, ale může výrazně pomoci i jednomu autorovi. Umožňuje jednoduché verzování, zálohování a přenášení mezi více počítači.

**Overleaf** Online nástroj pro L<sup>A</sup>T<sub>E</sub>X. Přímo zobrazuje náhled a umožňuje jednoduchou spolupráci (vedoucí může průběžně sledovat psaní práce), vyhledávání ve zdrojovém textu kliknutím do PDF, kontrolu pravopisu apod. Zdarma jej však lze využít pouze s určitými omezeními (někomu stačí na disertaci, jiný na ně může narazit i při psaní bakalářské práce) a pro dlouhé texty je pomalejší.

## Užitečné balíčky pro L<sup>A</sup>T<sub>E</sub>X

Studenti při sazbě textu často řeší stejné problémy. Některé z nich lze vyřešit následujícími balíčky pro L<sup>A</sup>T<sub>E</sub>X:

- `amsmath` – rozšířené možnosti sazby rovnic,
- `float`, `afterpage`, `placeins` – úprava umístění obrázků,
- `fancyvrb`, `alltt` – úpravy vlastností prostředí Verbatim,
- `makecell` – rozšíření možností tabulek,
- `pdflscape`, `rotating` – natočení stránky o 90 stupňů (pro obrázek či tabulku),
- `hyphenat` – úpravy dělení slov,
- `picture`, `epic`, `eepic` – přímé kreslení obrázků.

Některé balíčky jsou využity přímo v šabloně (v dolní části souboru fitthesis.cls). Nahlednutí do jejich dokumentace může být rovněž užitečné.

Sloupec tabulky zarovnaný vlevo s pevnou šířkou je v šabloně definovaný „L“ (používá se jako „p“).