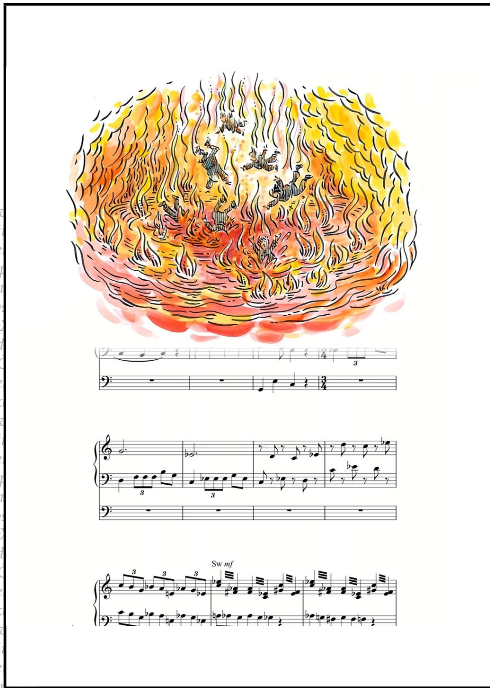
CSTUG



ZPRAVODAJ

Československého sdružení uživatelů TEXu

1–4

2024

ISSN 1211-6661
Tištěná verze

ISSN 1213-8185
Online verze

Ročník 34

OBSAH

Zpravodaj Československého sdružení uživatelů TeXu je vydáván v tištěné podobě a distribuován zdarma členům sdružení. Vydaná čísla Zpravodaje v elektronické podobě (PDF) jsou bezodkladně veřejně vystavena na webové adrese `https://www.cstug.cz/`.

Své příspěvky do Zpravodaje můžete zasílat v elektronické podobě, nejlépe jako jeden archivní soubor (`.zip, .arj, .tar.gz`), na e-mailovou adresu `bulletin@cstug.cz`. Nezapomeňte přiložit všechny soubory, které dokument načítá (s výjimkou standardních součástí TeX Live).

ISSN 1211-6661 (tištěná verze)
ISSN 1213-8185 (online verze)

CSTUG na konferenci TUG 2024

Vít Starý Novotný, Michal Hoftich, Jaromír Kuben

Od čtvrtka 18. července do neděle 21. července se v Praze s podporou sdružení CSTUG konala konference TUG 2024. S organizací konference pomohl předseda Petr Sojka, jeho syn Ondřej a členové výboru CSTUGu Michal Hoftich a Jaromír Kuben. Konference se také aktivně zúčastnil Vítek Starý Novotný. Dorazilo téměř 60 účastníků, kromě zástupců evropských států a USA také početná skupina z Indie, kam se konference příští rok přesune.

Ve čtvrtek ještě před oficiálním zahájením konference proběhl v prostorách MFF UK tutoriál věnovaný tvorbě značkovaných PDF dokumentů. Tutoriál moderovala Ulrike Fischer, členka týmu LaTeX, který se tímto projektem již delší dobu zabývá [1]. Účastníci se dozvěděli, které z často užívaných balíčků už podporují vytváření dobře značkovaných dokumentů a které jsou na seznamu čekatelů o doplnění této schopnosti.

V pátek dopoledne ukázal Martin Ruckert v přednášce nazvané *Profiling TeX input files* [2] nástroje `texprof` a `texprofile`, pomocí kterých je možné měřit rychlost kódu v TeXu. Překlad článku z přednášky začíná na straně 44.

Stejný den odpoledne ukázal Vítek Starý Novotný v přednášce nazvané *Markdown themes in practice* [3; 4], jak je možné generovat testové otázky na základě souborů ve formátu YAML a Markdown, vizte Obrázek 1.

Téhož dne večer ukázali Oliver Kopp, Marcel Krüger a Marei Peischl na workshopu nazvaném *Creation of LaTeX documents using a cloud-based pipeline* [5], jak je možné vytvářet (La)TeXové dokumenty v cloudu. Překlad článku z workshopu začíná na straně 59.

V sobotu ráno představil Michal Hoftich v přednášce nazvané *Web page to PDF conversion with Rmodepdf* [6; 7] způsob, jakým je možné v LuaLaTeXu načítat a sázet webové stránky, vizte Obrázek 2. Překlad článku z přednášky začíná na straně 28.

Během nedělního programu, který vedl Petr Sojka, představil Ondřej Sojka novinky z projektu univerzálních vzorů dělení slov v přednášce nazvané *Expanding hyphenation patterns across Slavic languages* [8; 9], vizte Obrázek 3. Pro pokrytí většího množství slovanských jazyků se Ondřej chystá použít mezireprezentaci textu v mezinárodní fonetické abecedě IPA.

Záznam všech 31 přednášek lze nalézt na kanálu YouTube TUGu [10]. Program konference zakončil varhanní koncert barokních skladeb v kostele U Salvátora.

DOI: 10.5300/2024-1-4/1

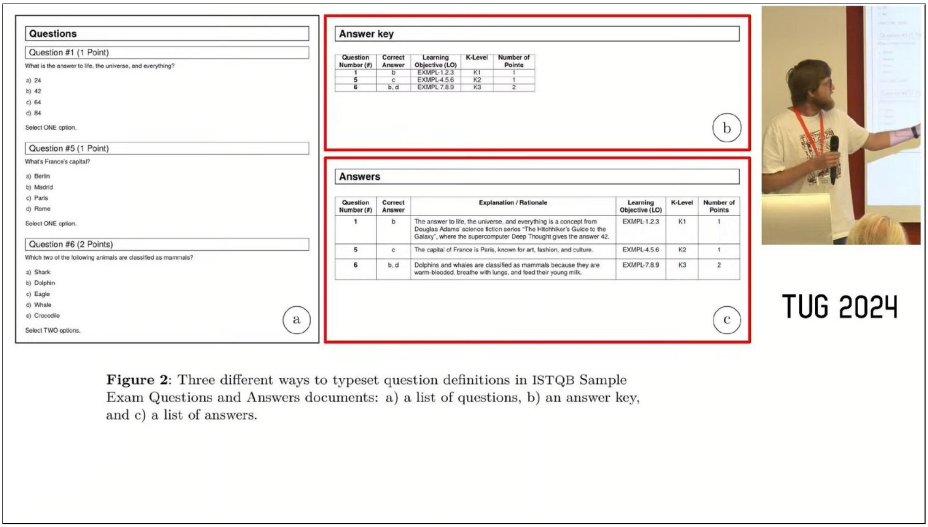
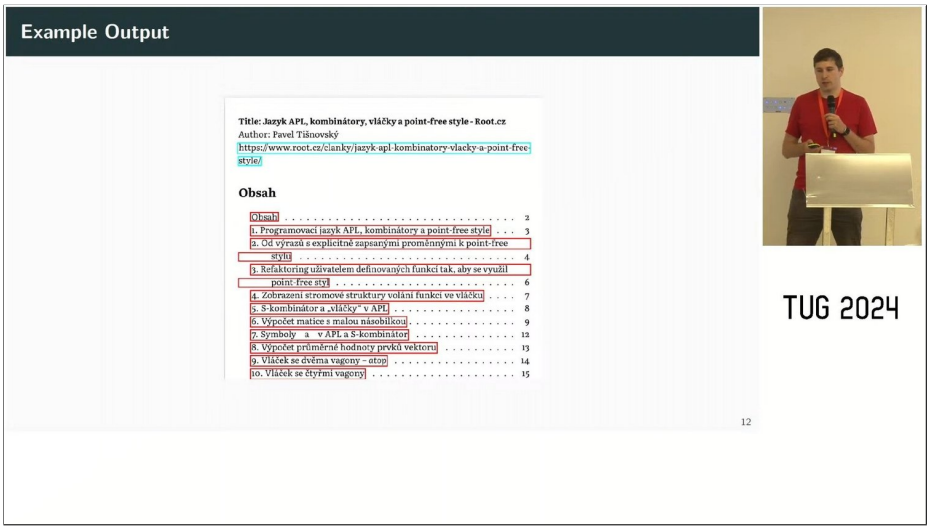
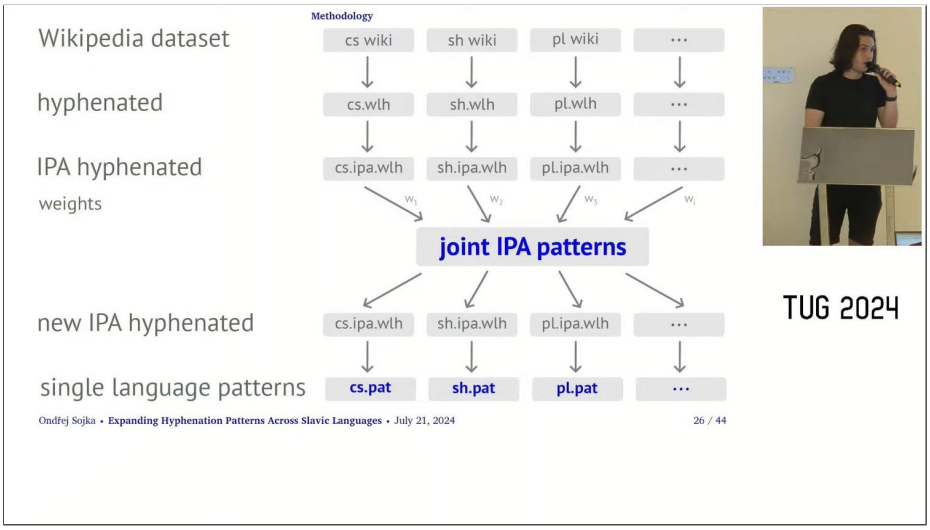


Figure 2: Three different ways to typeset question definitions in ISTQB Sample Exam Questions and Answers documents: a) a list of questions, b) an answer key, and c) a list of answers.

Obrázek 1: Vítek Starý Novotný přednáší o generování testových otázek na základě souborů ve formátech YAML a Markdown.



Obrázek 2: Michal Hoftich přednáší o sazbě webových stránek pomocí svého programu Rmodepdf.



Obrázek 3: Ondřej Sojka přednáší o přípravě panslovanských vzorů dělení slov pomocí mezinárodní fonetické abecedy IPA.

Obrázek 4: Skupinová fotografie účastníků konference.

Odkazy

1. MITTELBACH, Frank; FISCHER, Ulrike; ROWLEY, Chris. *LaTeX Tagged PDF Feasibility Evaluation* [online]. 2022-04. [cit. 2024-11-25]. Dostupné z: https://www.latex-project.org/publications/2020-tagged-pdf-feasibility.pdf.
2. RUCKERT, Martin. Profiling TeX input files. *TUGboat*. 2024, **45**(2), 203–210. Dostupné z DOI: 10.47397/tb/45-2/tb140ruckert-profiler.
3. STARÝ NOVOTNÝ, Vít. *Markdown themes in practice* [prezentační slajdy] [online]. 2024-07-19. [cit. 2024-11-18]. Dostupné z: https://www.tug.org/tug2024/slides/starynovotny-markdown-themes.pdf.
4. STARÝ NOVOTNÝ, Vít. Markdown themes in practice. *TUGboat*. 2024, **45**(2), 211–220. Dostupné z DOI: 10.47397/tb/45-2/tb140starynovotny-markdown-themes.
5. PEISCHL, Marei; KRÜGER, Marcel; KOPP, Oliver. Creation of LaTeX documents using a cloud-based pipeline. *TUGboat*. 2024, **45**(2), 227–233. Dostupné z DOI: 10.47397/tb/45-2/tb140peischl-pipeline.
6. HOFTICH, Michal. *Web page to PDF conversion with Rmodepdf: Leveraging LuaLaTeX for e-book reader-friendly documents* [prezentační slajdy] [online]. 2024-07-16. [cit. 2024-11-18]. Dostupné z: https://www.tug.org/tug2024/slides/hoftich-rmodepdf.pdf.
7. HOFTICH, Michal. Web page to PDF conversion with Rmodepdf: Leveraging LuaLaTeX for e-book reader-friendly documents. *TUGboat*. 2024, **45**(2), 246–252. Dostupné z DOI: 10.47397/tb/45-2/tb140hoftich-rmodepdf.
8. SOJKA, Ondřej. *Expanding hyphenation patterns across Slavic languages* [prezentační slajdy] [online]. 2024-07-21. [cit. 2024-11-18]. Dostupné z: https://www.tug.org/tug2024/slides/sojka-slavic.pdf.
9. SOJKA, Ondřej. Expanding hyphenation patterns across Slavic languages. *TUGboat*. 2024, **45**(2), 268–270. Dostupné z DOI: 10.47397/tb/45-2/tb140sojka-slavic.
10. *TUG 2024 Lectures* [online]. [cit. 2024-11-25]. Dostupné z: https://www.youtube.com/playlist?list=PLLt9mKFAx-FZCy6aqYbXSNyaAkr8Nv14X.

Summary: CSTUG at the TUG 2024 Conference

This article reports on the participation of CSTUG members at TUG 2024 in Prague.

Vít Starý Novotný, witiko@mail.muni.cz
Michal Hoftich, michal.h21@gmail.com
Jaromír Kuben, jaromir.kuben@unob.cz

Příprava videozáznamu české premiéry multimediálního díla „Fantasia Apocalyptica“

Vít Starý Novotný

Dne 11. října 2019 se při příležitosti oslav 25. výročí založení Fakulty informatiky Masarykovy univerzity uskutečnila česká premiéra multimediálního díla „Fantasia Apocalyptica“ za osobní účasti autora, Donalda Knutha. Z představení byl pořízen záznam, který je dostupný na kanálu YouTube Fakulty informatiky.

V tomto článku popisuji přípravu záznamu od akvizice a zpracování surových audiovizuálních dat přes přípravu replik panelů doprovázejících představení po spojení jednotlivých částí do výsledného záznamu. S výjimkou střihu úvodního slova a závěru, navýšení rozlišení videa a zúžení dynamického rozsahu zvuku proběhlo zpracování pomocí svobodných nástrojů jako TEX, Audacity, Aegisub, FFmpeg, MLT, PDFtk a ImageMagick. V článku ukazuji užití těchto nástrojů.

Klíčová slova: audiovizuální záznam, zpracování videa, titulky, zpracování zvuku, TEX, Audacity, Aegisub, ASS, FFmpeg, ImageMagick, MLT, PDFtk, waifu2x

Dne 11. října 2019 se v jezuitském kostele Nanebevzetí Panny Marie uskutečnila česká premiéra multimediálního díla „Fantasia Apocalyptica“ [1] za osobní účasti autora, Donalda Knutha. Z představení byl pořízen záznam, který je veřejně dostupný na kanálu YouTube Fakulty informatiky (FI MU) [2]. Předchozí články ve Zpravodaji CSTUGu popisují průběh představení [3] a přednášek Donalda Knutha při příležitosti návštěvy Brna [4]. V tomto článku popisuji přípravu záznamu představení.

Nejprve v sekci 1 popisuji akvizici a zpracování surových audiovizuálních dat. Následně v sekci 2 na straně 9 popisuji přípravu videozáznamu úvodního slova a závěru představení. Dále v sekci 3 na straně 14 popisuji přípravu panelů doprovázejících představení. Nakonec v sekci 4 na straně 20 popisuji spojení jednotlivých částí do výsledného záznamu. V sekci 5 na straně 23 popisuji promítání hotového videozáznamu na FI MU 18. prosince 2019.

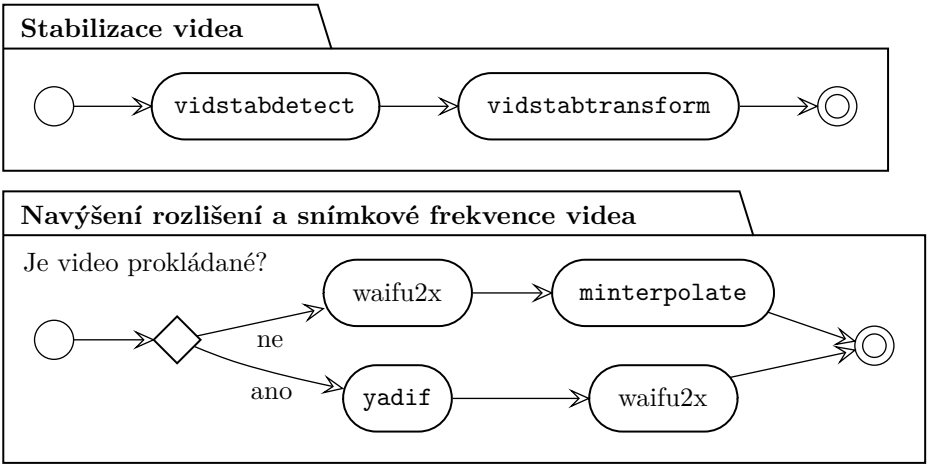
1 Akvizice a zpracování videa a zvuku

Video a zvuk představení zaznamenali a předzpracovali Pavel Šiler, Petr „Tudy“ Holubář a studenti laboratoře LEMMA [5]. Poté jsem u videa sjednotil formát dat a do zvuku jsem přimíchal zvuky okolí a úvodní slovo Jiřího Zlatušky.

1.1 Záznam videa

Videozáznam představení pořídil Pavel Šiler na kameru Sony PXW-X70. Video zahrnuje přednášky Donalda Knutha na FI MU [6; 7; 8, adresář `staticka/`], záběry kostela a varhan a úvodní slovo Jiřího Zlatušky.

DOI: 10.5300/2024-1-4/7



Obrázek 1: Diagram aktivit zpracování videa pomocí nástrojů FFmpeg a waifu2x.

1.2 Zpracování videa

Ačkoliv jsem video obdržel v rozlišení 1920×1080 px a s frekvencí 25 snímků za sekundu, od výsledného videozáznamu se očekávalo alespoň dvojnásobné rozlišení 4K a frekvence 50 snímků za sekundu. Záznamy přednášek jsem obdržel v neprokládaném (tzv. „progressive“) formátu, zatímco záběry kostela, varhan a úvodní slovo byly v prokládaném (tzv. „interlaced“) formátu. Přednášky a úvodní slovo byly dále natáčeny ze stativu, na rozdíl od záběrů kostela a varhan, které byly pořízeny z ruky. Cílem zpracování videa bylo nejen navýšení rozlišení a snímkové frekvence, ale také sjednocení formátu videa a stabilizace záběrů z ruky.

Video jsem zpracoval svobodným nástrojem FFmpeg [9] a otevřeným nástrojem waifu2x [10], vizte obrázky 1–4. Nejprve jsem stabilizoval záběry z ruky filtry `vidstabdetect` [11] a `vidstabtransform` [12] nástroje FFmpeg. Potom jsem navýšil rozlišení a snímkovou frekvenci následujícím způsobem:[1]

- U neprokládaných videí jsem nejprve navýšil rozlišení nástrojem waifu2x. Následně jsem navýšil snímkovou frekvenci filtrem `minterpolate` [13] nástroje FFmpeg, který chybějící snímky interpoluje podle okolních snímků.
- U prokládaných videí jsem nejprve navýšil snímkovou frekvenci filtrem `yadif` [14], který chybějící půlsnímky interpoluje podle okolních půlsnímků a řádků. Nakonec jsem navýšil rozlišení nástrojem waifu2x.

[1] Navýšené rozlišení a především snímková frekvence jsou jen iluze. Proto upravené video tvoří jen ca 7 minut ze 104minutového videozáznamu. Zbylých 97 minut tvoří animované panely, které mají nativní rozlišení 4K a snímkovou frekvenci 50 snímků za sekundu.

1.3 Záznam zvuku

Zvuk představení byl zaznamenán šesti mikrofony:

- Dva mikrofony Zoom H1 [15] byly umístěny na řečnickém pultu a nad panely.
- Dva mikrofony Oktava MK012 [16, soubor `ORTF-MK012_nearReversed.flac`] byly zapojeny do rekordéru Zoom H4, umístěny na kůru a namířeny do kostela.
- Mikrofon Roland R26 [17] a mikrofon kamery Sony PXW-X70 [18] byly umístěny na kůru a namířeny na varhany.

Zvuk předzpracoval Petr „Tudy“ Holubář [16]. Hlavním vstupem jeho mixu byl prostorový záznam z mikrofonů Oktava MK012. Do prostorového záznamu Petr pro konkrétnost lehce přimíchal přímý záznam varhan z mikrofonu Roland R26, který frekvenčně ošetřil tak, aby se neprojevily nežádoucí fázové posuny ve spodním pásmu. Pro zúžení dynamického rozsahu použil Petr digitální VST modul SSL G-Master Buss Compressor [19] od firmy Waves Audio.

1.4 Zpracování zvuku

Zvuk jsem zpracoval svobodným programem Audacity [20; 21], vizte Obrázek 5. Výsledný mix zahrnuje svobodný zvuk zvonu kostnické katedrály [22], předzpracovaný zvuk představení od Petra Holubáře a úvodní slovo Jiřího Zlatušky podkreslené zvuky okolí z mikrofonů Oktava MK012.

2 Příprava videozáznamu úvodního slova a závěru

Videozáznam úvodního slova a závěru jsem připravil se Šimonem Mačejovským.

2.1 Střih úvodního slova a závěru

Videozáznamy úvodního slova a závěru [23] připravil Šimon Mačejovský nástrojem Sony Vegas 15. Videozáznam obsahuje záběry z přednášek Donalda Knutha na FI MU, záběry kostela a varhan, úvodní slovo Jiřího Zlatušky a fotografie od Martiny Morávkové [24].

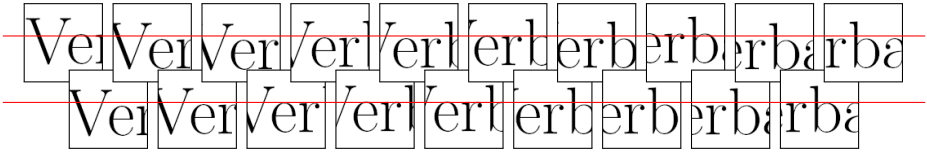
2.2 Titulky úvodního slova

Úvodní slovo Jiřího Zlatušky jsem opatřil titulky ve formátu ASS[2] [28], které jsem připravil ve svobodném nástroji Aegisub [29], vizte Obrázek 6.

[2] Pro vypálení titulků do obrazu videa slouží filtr `ass` nástroje FFmpeg:

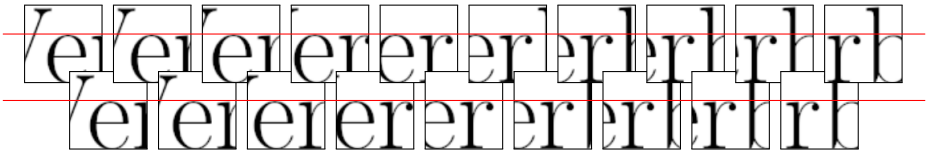
```
$ ffmpeg -i vstupní-video.mp4 -vf ass=vstupní-titulky.ass video-s-titulky.mp4
```

Filtr `ass` [25] implementuje softwarová knihovna `libass` [26], která ve výchozím nastavení nepoužívá kerningovou tabulku použitého písma [27]. Pro použití kerningové tabulky je třeba do sekce `[Script Info]` souboru `vstupní-titulky.ass` přidat řádek `Kerning: yes`.



(a) Při záznamu videa z ruky může vzniknout záznam s roztřeseným obrazem.

```
$ ffmpeg -i vstupní-video.mp4 -vf vidstabdetect -f null -
$ ffmpeg -i vstupní-video.mp4 -vf vidstabtransform výstupní-video.mp4[3]
```



(b) Filtry `vidstabdetect` a `vidstabtransform` izolují a odstraňují rychlé pohyby kamery. Dále dochází k přiblížení obrazu tak, aby nebyly viditelné okraje záběru.

Obrázek 2: Stabilizace videa pomocí filtrů nástroje FFmpeg.

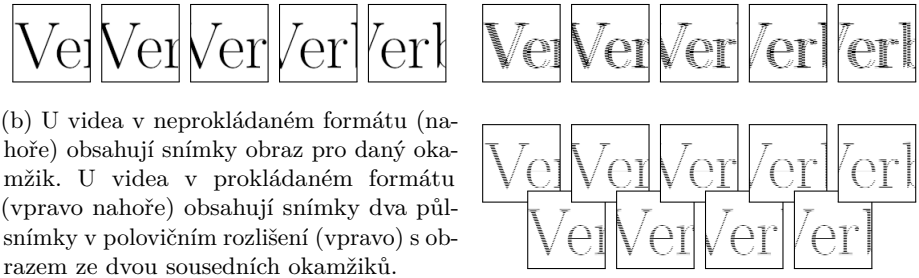
(a) Obraz zachycený digitální kamerou může mít nedostatečné rozlišení.

```
$ ffmpeg -i vstupní-video.mp4 -f image2 obrázky-snímků/%06d.png
$ find obrázky-snímků/ -name '*.png' | sort > obrázky-snímků.txt
$ th waifu2x.lua -l obrázky-snímků.txt -model_dir models/photo -tta 1 \
>                -o zvětšené-obrázky/%06d.png
$ ffmpeg -framerate 25 zvětšené-obrázky/%06d.png výstupní-video.mp4
```

(b) Nástroj waifu2x dokáže zvýšit rozlišení obrazu pomocí umělých neuronových sítí [30].

Obrázek 3: Navýšení rozlišení videa pomocí nástroje waifu2x.

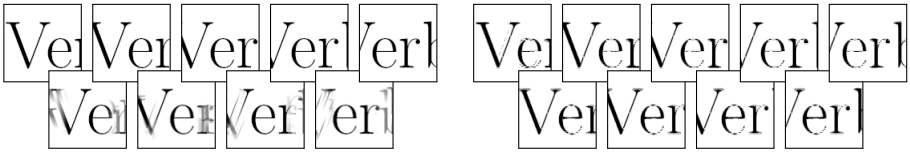
(a) Při záznamu videa na digitální kameru dochází k zachycení mnoha snímků za sekundu.



(b) U videa v neprokládaném formátu (nahoře) obsahují snímky obraz pro daný okamžik. U videa v prokládaném formátu (vpravo nahoře) obsahují snímky dva půlsnímky v polovičním rozlišení (vpravo) s obrazem ze dvou sousedních okamžiků.

```
$ ffmpeg -i vstupní-video.mp4 \
>        -vf minterpolate     \
>          výstupní-video.mp4
```

```
$ ffmpeg -i vstupní-video.mp4 \
>        -vf yadif=send_field \
>          výstupní-video.mp4
```



(c) Filtr `minterpolate` dokáže u neprokládaných videí odhadnout obraz v mezilehlých okamžicích podle okolních snímků.[4]

(d) Filtr `yadif` dokáže u prokládaných videí zdvojnásobit rozlišení půlsnímků podle daného půlsnímku a jeho sousedů.

Obrázek 4: Navýšení snímkové frekvence videa pomocí filtrů nástroje FFmpeg.

[3] Všechny příkazy nástroje FFmpeg používají navíc následující výstupní parametry:

`-c:v libx265 -crf 17 -preset placebo -pix_fmt yuv420p -c:a aac -b:a 384k -ar 44.1k`

Tyto parametry zajišťují vizuálně a auditivně bezztrátovou kompresi videa [31] a zvuku [32].

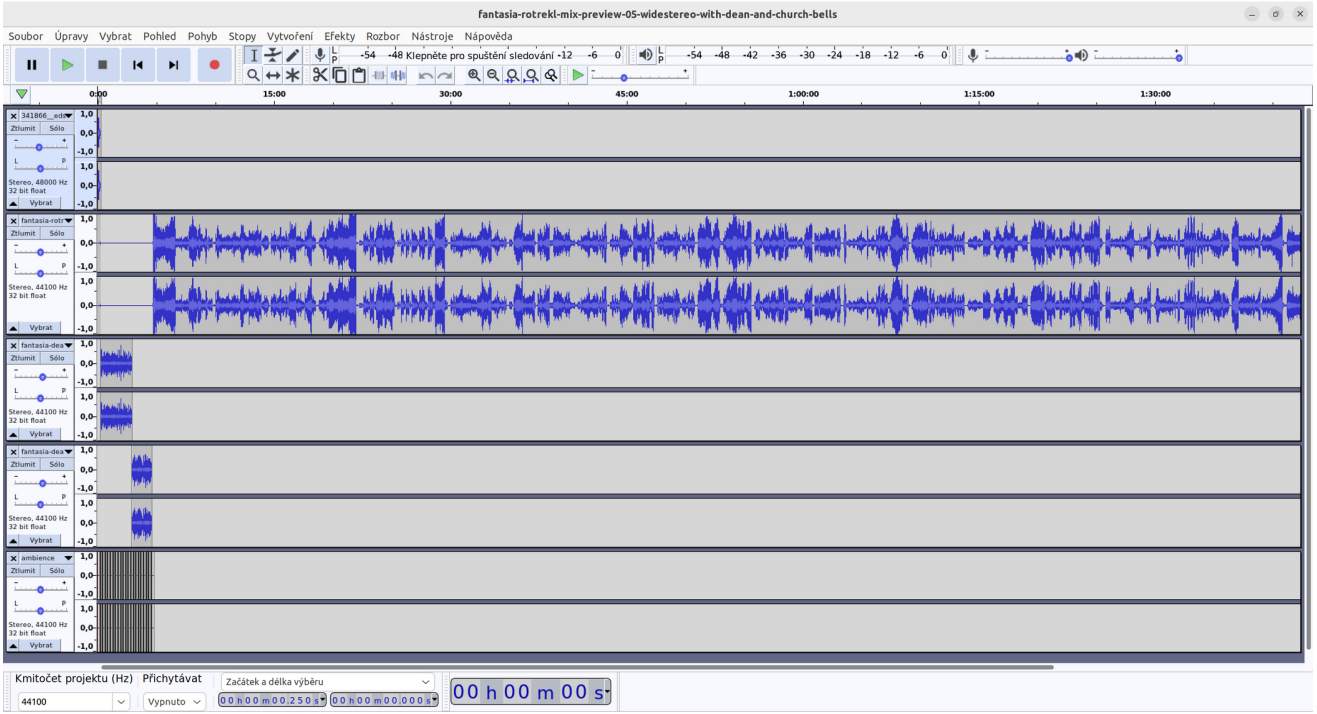
[4] Při rychlém pohybu kamery nebo zabíraného objektu mohou při použití filtru `minterpolate` vznikat vizuální artefakty, především na okrajích snímků a objektů. Při zpracování přednášek Donalda Knutha na FI MU jsem proto ze vstupního videa nejprve vytvořil obrázky snímků:

```
$ ffmpeg -i vstupní-video.mp4 -vf minterpolate obrázky-snímků/%06d.png
```

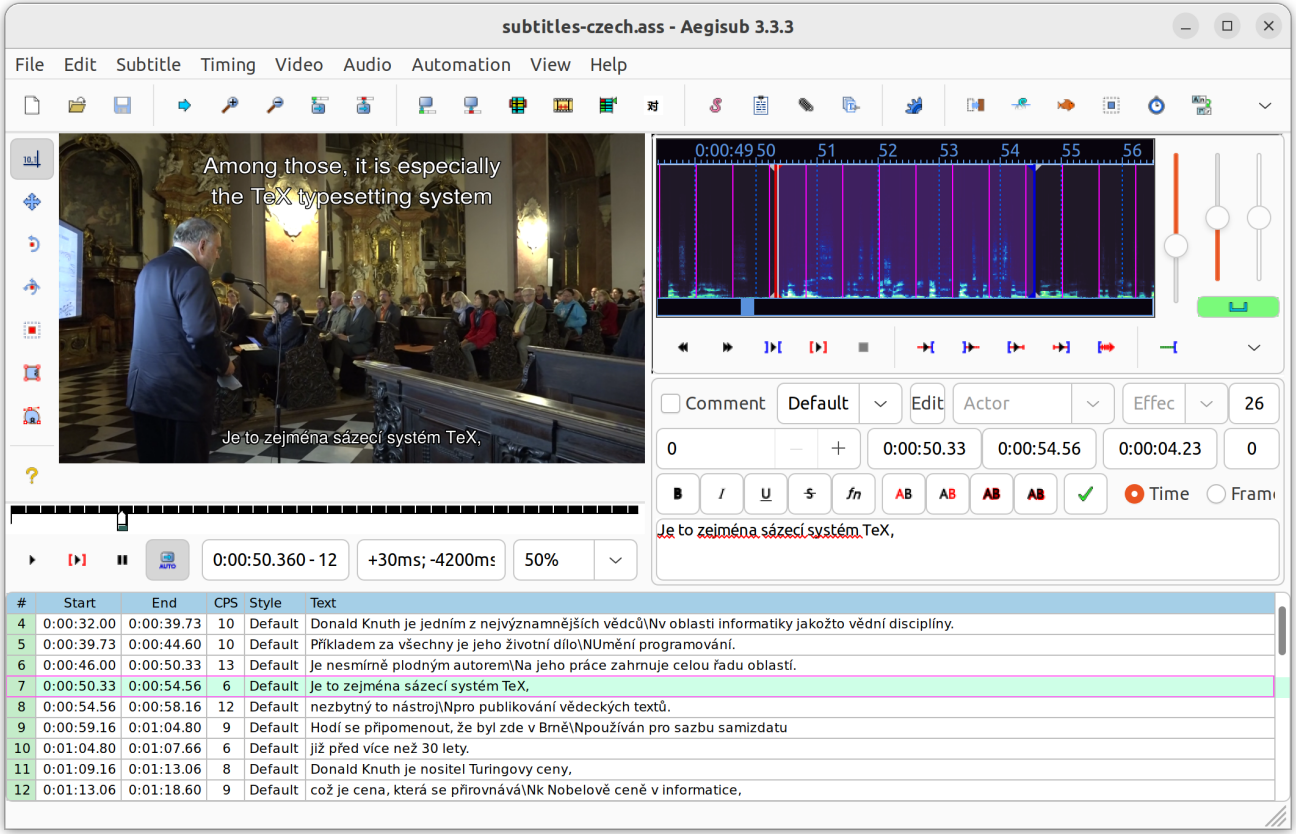
Potom jsem obrázky snímků ručně prošel a smazal jsem ty, na kterých byly viditelné artefakty na osobě Donalda Knutha. Nakonec jsem z obrázků snímků vytvořil výsledné video:

```
$ ffmpeg -framerate 50 obrázky-snímků/%06d.png výstupní-video.mp4
```

Video mělo proměnlivou frekvenci 25–50 snímků za sekundu podle počtu smazaných obrázků.



Obrázek 5: Zpracování zvuku ve svobodném programu Audacity. Výsledný mix zahrnuje zvuk zvonu (nahoře), předzpracovaný zvuk představení od Petra „Tudy“ Holubáře (druhý odshora) a úvodní slovo Jiřího Zlatušky (uprostřed) podkreslené zvuky okolí (dole).



Obrázek 6: Příprava titulků úvodního slova ve svobodném programu Aegisub.

3 Příprava panelů doprovázejících varhanní oratorium

Představení doprovázela projekce na dvou panelech: První panel zobrazoval text Janovy Apokalypsy v novozákonní řečtině a anglickém překladu a druhý panel zobrazoval ilustrace Duane Bibbyho.[5]

Ve videozáznamu jsme panely oproti představení obohatili takto:

1. Do panelu s textem jsem přidal český překlad Janovy Apokalypsy.
2. Navýšil jsem rozlišení ilustrací od Duane Bibbyho.
3. Změny textu a ilustrací na prvních dvou panelech jsem ozvláštnil animací.
4. Tomáš Szaniszlo připravil třetí panel, který zobrazuje noty pro varhany.

V této sekci popisuji proces přípravy panelů použitých ve videozáznamu.

3.1 Příprava panelu s trojjazyčným textem Zjevení Janova

Pro řecký a anglický text jsem použil PDF dokument [34] z kanadské premiéry, který zobrazuje nejprve řecký text na světle žlutém poli a pod ním anglický text na modrém poli, vizte Obrázek 7a.

Aby řecký a anglický text tvořily horní dvě barvy české trikolory, nejprve jsem PDF dokument dekomprimoval svobodným nástrojem PDFtk [35]. Následně jsem pomocí POSIXového editoru `sed` upravil barvu textu a pozadí v dekomprimovaném dokumentu tak, aby byl anglický text na červeném poli, vizte Obrázek 7b.

Pro český text jsem se svolením České biblické společnosti použil český ekumenický překlad. Pomocí TEXu, písma Slabo od písmolijny Tiro Typeworks a nástroje ImageMagick [36] jsem vytvořil dokument, který vzhledově odpovídá řeckému a anglickému textu a který zobrazuje text na modrém poli, vizte Obrázek 7c.

3.2 Příprava panelu s ilustracemi Duanea Bibbyho

Ilustrace Duanea Bibbyho [37] jsem nejprve extrahoval z dokumentu PDF pomocí svobodného nástroje pdfimages. Poté jsem navýšil jejich rozlišení pomocí nástroje waifu2x podobně jako na Obrázku 3:

```
$ pdfimages -png fant-screen1-art.pdf ilustrace/ilustrace
$ find ilustrace/ -name '*.png' | sort > ilustrace.txt
$ th waifu2x.lua -l ilustrace.txt -model_dir models/anime_style_art_rgb \
>                -noise_level 3 -tta 1 -o zvětšené-ilustrace/%06d.png
```

3.3 Animace panelů s textem a ilustracemi

Nejprve jsem pro všechny následující páry obrázků z panelů s textem a ilustracemi vygeneroval nástrojem FFmpeg videa s animacemi přechodu, vizte Obrázek 9.

Posléze jsem obrázky z panelů s textem a ilustracemi synchronizoval se zvukem představení ve svobodném programu Flowblade [38] pro digitální střih videa, vizte Obrázek 8. Projekt Flowblade jsem exportoval do souboru XML ve formátu

[5] Ilustrace Duane Bibbyho vyšly také v knize *Fantasia Apocalyptica Illustrated* [33], kterou je možné zakoupit např. na českém e-shopu `bookshop.cz`.

ΤΟΙΣ ΔΕ ΔΕΙΛΟΙΣ ΚΑΙ ΑΠΙΣΤΟΙΣ ΚΑΙ ΕΒΔΕΛΥΓΜΕΝΟΙΣ ΚΑΙ ΦΟΝΕΥΣΙ ΚΑΙ ΠΟΡΝΟΙΣ ΚΑΙ ΦΑΡΜΑΚΟΙΣ ΚΑΙ ΕΙΔΩΛΟΛΑΤΡΑΙΣ ΚΑΙ ΠΑΣΙ ΤΟΙΣ ΨΕΥΔΕΣΙ ΤΟ ΜΕΡΟΣ ΑΥΤΩΝ ΕΝ ΤΗ ΛΙΜΝΗ ΤΗ ΚΑΙΟΜΕΝΗ ΕΝ ΠΥΡΙ ΚΑΙ ΘΕΙΩ Ο ΕΣΤΙΝ Ο ΘΑΝΑΤΟΣ Ο ΔΕΥΤΕΡΟΣ

But cowards, traitors, perverts, murderers, the immoral, those who practice magic, those who worship idols, and all liars—the place for them is the lake burning with fire and sulfur, which is the second death."

(a) Pro vrchní část panelu jsem použil řecký a anglický text z kanadské premiéry. [34]

```
$ pdftk dokument.pdf output dekomprimovaný-dokument.pdf uncompress
$ sed -i dekomprimovaný-dokument.pdf \
> -e's/\.686 \.933 \.933 RG \.686 \.933 \.933 rg/1 .890 .902 RG 1 .890 .902 rg/' \
> -e's/\.627 \.322 \.176 RG \.627 \.322 \.176 rg/.184 .31 .31 RG .184 .31 .31 rg/'
```

But cowards, traitors, perverts, murderers, the immoral, those who practice magic, those who worship idols, and all liars—the place for them is the lake burning with fire and sulfur, which is the second death."

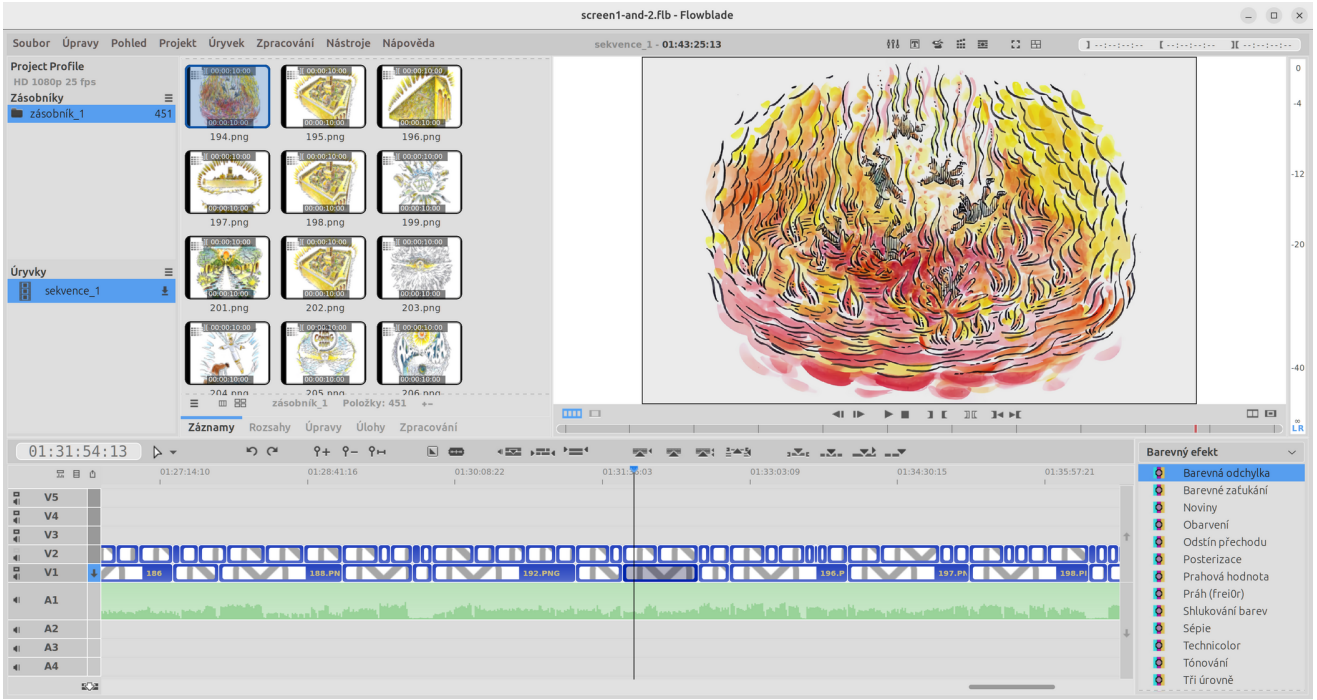
(b) PDF dokument s řeckým a anglickým textem jsem dekomprimoval nástrojem PDFtk a následně jsem ho upravil POSIXovým editorem **sed** tak, aby byl anglický text byl na červeném poli a spolu s řeckým textem tak tvořil horní dvě barvy české trikolory.

```
$ parallel -- convert -density 600 {} -resize '1920x1200!' sRGB {.}.png ::: *.pdf
$ ROZMAZEJ='-resize 50% -filter Gaussian -define filter:sigma=12 -resize 200%'
$ convert pozadí.png \( \( zářící-text.png $ROZMAZEJ \) základní-text.png \
>                       -composite \) -composite výsledný-překrytý-text.png
```

Avšak zbabělci, nevěrní, nečistí, vrahové, cizoložníci, zaklínači, modláři a všichni lháři najdou svůj úděl v jezeře, kde hoří oheň a síra. To je ta druhá smrt."

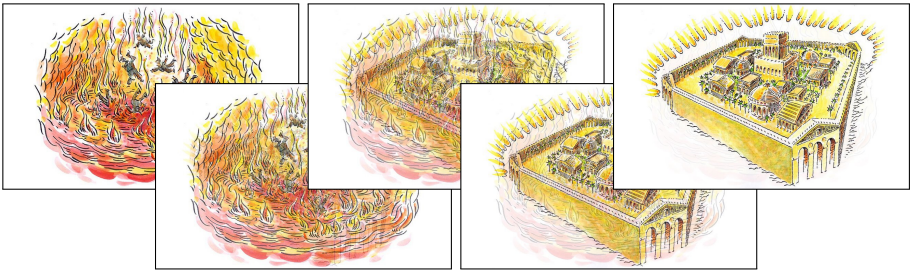
(c) Pro spodní část panelu jsem pomocí TEXu vytvořil samostatný dokument s českým ekumenickým překladem Janovy Apokalypsy. Pro efekt záře jsem vytvořil tři verze dokumentu s pozadím, zářícím textem a základním textem. Dokumenty jsem rasterizoval na bitmapy, zářící text jsem rozmazal a bitmapy jsem překryl, vše nástrojem ImageMagick.

Obrázek 7: Příprava panelu s trojjazyčným textem Zjevení Janova.



Obrázek 8: Synchronizace panelů s textem a ilustracemi se zvukem představení ve svobodném programu Flowblade.

```
$ ffmpeg -loop 1 -framerate 50 -i první-vstupní-obrázek.png              \
>        -loop 1 -framerate 50 -i druhý-vstupní-obrázek.png              \
>        -filter_complex "
>          [0:v]trim=start=0:end=$D[6] [prvni];
>          [1:v]trim=start=0:end=$D[druhy];
>          [prvni]format=yuva420p,fade=out:st=0:d=$D:alpha=1[stmivacka];
>          [druhy]format=yuva420p,fade=in:st=0:d=$D:alpha=1[roztmivacka];
>          [stmivacka][roztmivacka]overlay[prechod]
>        " -map '[prechod]' -r 50 výstupní-video.mp4
```



Obrázek 9: Animace přechodu mezi dvěma obrázky pomocí nástroje FFmpeg.

používaném svobodnou softwarovou knihovnou MLT [39] pro střih videa. Z tohoto souboru jsem pomocí následujícího programu v jazyce Python 3 extrahoval soubory se seznamem obrázků a dobou jejich trvání vyčíslenou v snímcích videa:

```
import xml.etree.ElementTree as ET
from pathlib import Path

obrazky = {}
strom = ET.parse('vstupní-projekt.xml')
for producer in strom.findall('.//producer'):
    obrazek = Path(producer.find('property[@name="resource"]').text)
    producer_id = producer.attrib['id']
    entry = strom.find(f'.//playlist/entry[@producer="{producer_id}"]')
    trvani = int(entry.attrib['out']) - int(entry.attrib['in']) + 1
    obrazky[obrazek] = trvani

with open('časování-ilustrací.txt', 'wt') as f1, \
        open('časování-řeckého-a-anglického-textu.txt', 'wt') as f2, \
        open('časování-českého-textu.txt', 'wt') as f3:
```

[6] Proměnná `$D` značí délku animace v sekundách. Použil jsem hodnoty `D=0.52` pro krátké přechody o délce 520 milisekund a `D=1` pro dlouhé přechody o délce jedné sekundy. Při frekvenci 50 snímků za sekundu trvají krátké přechody 26 snímků a dlouhé přechody 50 snímků.

```
    for obrazek, trvani in sorted(obrazky.items()):
        if obrazek.parent.name == 'ilustrace':
            print(f'{obrazek.name}\t{trvani}', file=f1)
        elif obrazek.parent.name == 'řecký-a-anglický-text':
            print(f'{obrazek.name}\t{trvani}', file=f2)
            print(f'{obrazek.name}\t{trvani}', file=f3)
```

Jeden řádek výstupního souboru `časování-ilustrací.txt` je např. `194.png 1119`. To znamená, že obrázek `ilustrace/194.png` s ilustrací verše 21:8 trvá 1 119 snímků videa, což je při frekvenci 50 snímků za sekundu přibližně 22 sekund.

Pro každý obrázek jsem vybral animaci podle jeho doby trvání:

- Po obrázcích kratších než 2 sekundy následuje okamžitě další obrázek.
- Po obrázcích dlouhých 2–5 sekund následuje krátká animace přechodu (520 ms).
- Po obrázcích delších než 5 sekund následuje dlouhá animace přechodu (1 s).

Např. po obrázku `ilustrace/194.png` následuje dlouhá animace přechodu.

Dále jsem pro každý obrázek nástrojem FFmpeg vygeneroval video, jehož délka odpovídala době trvání obrázku po odečtení doby trvání animace. Např. video pro obrázek `ilustrace/194.png` trvá 1 119 snímků videa po odečtení jedné sekundy pro dlouhou animaci přechodu:

```
$ ffmpeg -loop 1 -framerate 50 -i ilustrace/194.png              \
>        -t $(bc -l <<< '1119 / 50 - 1')                         \
>        -r 50 výstupní-video-pro-verš-21-08.mp4
```

Nakonec jsem videa obrázků a videa animací spojil nástrojem FFmpeg do tří animovaných videí s řeckým a anglickým textem, českým textem a ilustracemi [40].

3.4 Příprava panelu s notami pro varhany

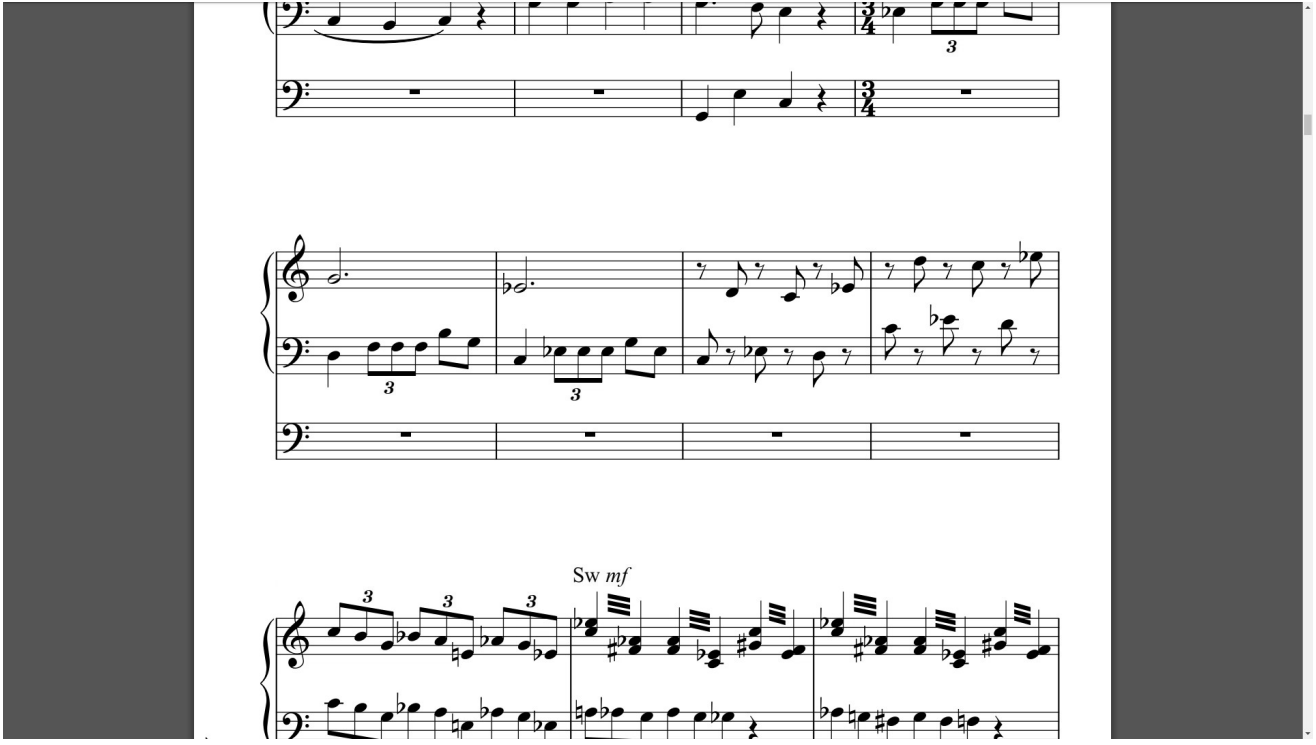
Pro každou z 22 kapitol si Tomáš Szaniszlo na notebooku otevřel digitalizované noty pro varhany [1, sekce „Music scores to download“] v prohlížeči PDF dokumentů a začal zaznamenávat obrazovku počítače. Následně si Tomáš pustil zvuk představení a pomocí trackpadu posouval noty tak, aby zobrazená část odpovídala zvuku. Výsledkem bylo 22 videí not pro varhany, vizte Obrázek 10.

Tato videa jsem nástrojem FFmpeg seřízl z 1920 px na 1344 px, aby zmizely postranní šedivé okraje:

```
$ ffmpeg -i vstupní-video.mp4                                    \
>        -vf 'crop=1344:1076:280:4,
>             pad=w=1920:h=1076:x=ow-iw/2:y=oh-ih/2:color=white' \
>          výstupní-video.mp4
```

Dále jsem do horní části videí doplnil bílý obdélník o výšce 62 px, který na začátku videa překrývá horní šedivý okraj a pak se změní na poloprůhledný obdélník, který přechází od plné neprůhlednosti ve své horní části po plnou průhlednost ve své spodní části a tvoří tak měkký předěl mezi notami a ilustracemi:

```
$ ffmpeg -i vstupní-video.mp4 \
```



Obrázek 10: Videozáznam not pro varhany od Tomáše Szaniszla.

```
>        -vf "drawbox=0:0:1920:62:white:fill:enable='lte(n,$N[7])',
>             drawbox=0:0:1920:1:white@1.0000:enable='gt(n,$N)',
>             drawbox=0:1:1920:1:white@0.9838:enable='gt(n,$N)',
>             drawbox=0:2:1920:1:white@0.9677:enable='gt(n,$N)',
>             ...
>             drawbox=0:61:1920:1:white@0.016:enable='gt(n,$N)'" \
>          výstupní-video.mp4
```

Poté jsem videa sestříhal a synchronizoval se zvukem představení v programu Flowblade podobně jako v předchozí sekci. Výsledné video jsem vykreslil nástrojem Melt z knihovny MLT:

```
$ melt vstupní-projekt.xml -consumer avformat:výstupní-video.mp4 \
>      vcodec=libx265 crf=17 preset=veryslow pix_fmt=yuv420p     \
>      acodec=aac ab=384k ar=44100
```

Pro srovnání surových videí not s výsledným videem vizte obrázky 10 a 12.

4 Spojení jednotlivých částí do výsledného videa

Nejprve jsem videa panelů s textem, ilustracemi a notami pro varhany spojil podle diagramu na Obrázku 11:

```
$ ffmpeg -f lavfi -i color=c=white:s=3840x2160:rate=50 \
>        -i vstupní-video-s-řeckým-a-anglickým-textem.mp4 \
>        -i vstupní-video-s-českým-textem.mp4 \
>        -i vstupní-video-s-ilustracemi.mp4 \
>        -i vstupní-video-s-notami-pro-varhany.mp4 \
>        -filter_complex '
>            [0:v]null[pozadi]; [1:v]null[recky-a-anglicky-text];
>            [2:v]crop=1920:960:0:63[cesky-text];
>            [3:v]pad=1920:1084:ow-iw/2:oh-ih/2:white[ilustrace];
>            [4:v]crop=1344:1076:280:4,
>                 pad=1920:1076:ow-iw/2:oh-ih/2:white[noty];
>            [pozadi][recky-a-anglicky-text]overlay=0:160[kolaz1];
>            [kolaz1][cesky-text]overlay=0:1360[kolaz2];
>            [kolaz2][noty]overlay=1920:1084[kolaz3];
>            [kolaz3][ilustrace]overlay=1920:0[kolaz]' \
>        -map '[kolaz]' -t 01:43:25 -r 50 výstupní-video.mp4
```

Potom jsem k výsledku připojil videozáznam úvodního slova a závěru [40]. Obrázek 12 ukazuje výsledné video.

[7] Proměnná `$N` značí snímek, po kterém se obdélník změní na poloprůhledný, a liší se pro jednotlivá videa not. Např. ve videu pro 21. kapitolu jsem nastavil `N=2700`. Při frekvenci 50 snímků za sekundu tedy dojde ke změně po 54 sekundách, kdy již Tomáš posunul noty tak, že horní šedivý okraj není vidět.

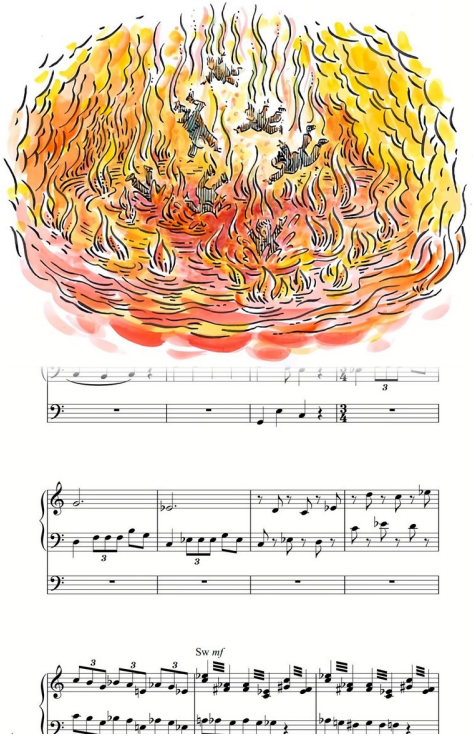
Obrázek 11: Diagram pro spojení jednotlivých panelů do výsledného videa.

21:08

ΤΟΙΣ ΔΕ ΔΕΙΛΟΙΣ ΚΑΙ ΑΠΙΣΤΟΙΣ ΚΑΙ ΕΒΔΕΛΥΓΜΕΝΟΙΣ ΚΑΙ ΦΟΝΕΥΣΙ ΚΑΙ ΠΟΡΝΟΙΣ ΚΑΙ ΦΑΡΜΑΚΟΙΣ ΚΑΙ ΕΙΔΩΛΟΛΑΤΡΑΙΣ ΚΑΙ ΠΑΣΙ ΤΟΙΣ ΨΕΥΔΕΣΙ ΤΟ ΜΕΡΟΣ ΑΥΤΩΝ ΕΝ ΤΗ ΛΙΜΝΗ ΤΗ ΚΑΙΟΜΕΝΗ ΕΝ ΠΥΡΙ ΚΑΙ ΘΕΙΩ Ο ΕΣΤΙΝ Ο ΘΑΝΑΤΟΣ Ο ΔΕΥΤΕΡΟΣ

But cowards, traitors, perverts, murderers, the immoral, those who practice magic, those who worship idols, and all liars—the place for them is the lake burning with fire and sulfur, which is the second death."

Avšak zbabělci, nevěrní, nečistí, vrahové, cizoložníci, zaklínači, modláři a všichni lháři najdou svůj úděl v jezeře, kde hoří oheň a síra. To je ta druhá smrt."



Obrázek 12: Výsledné video.

5 Závěr

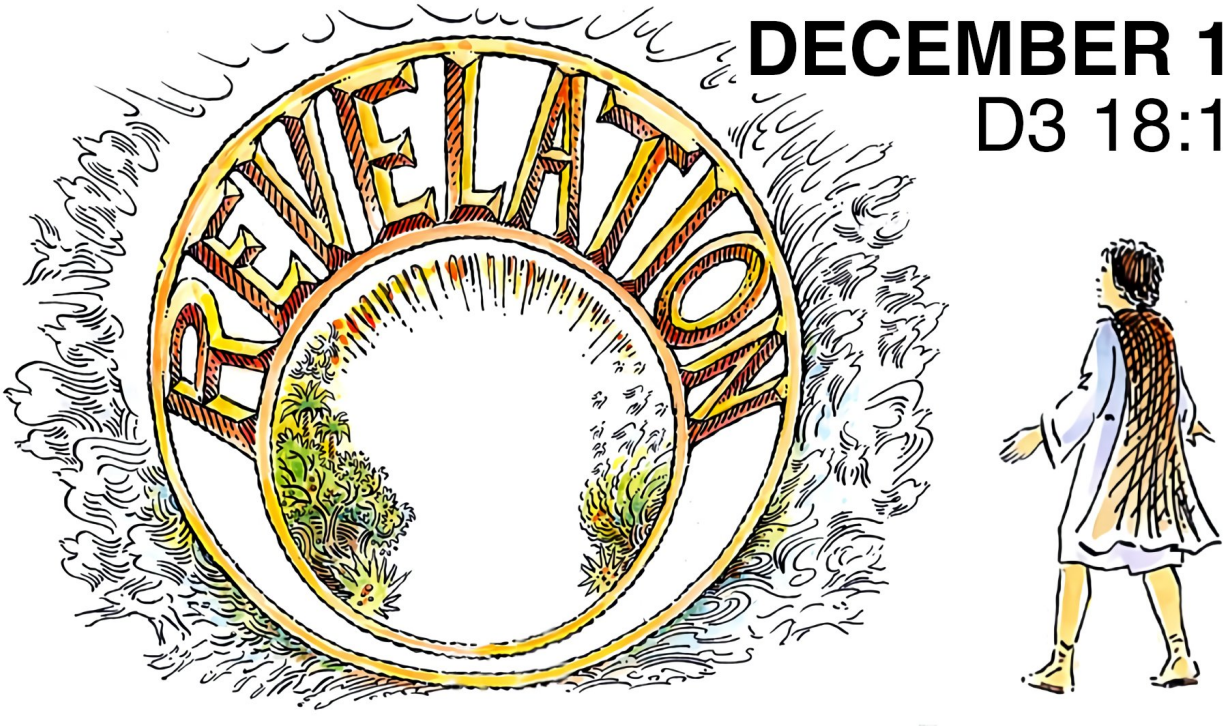
Dne 18. prosince 2019 se v posluchárně D3 na FI MU konalo promítání videozáznamu představení, vizte Obrázek 13.

Před začátkem promítání spustil Tomáš Szaniszlo přes piezoelektrické reproduktory integrované na základních deskách počítačů v počítačové hale digitalizovanou verzi hudby, která při představení doprovází verš 2:1 Zjevení. Zároveň zobrazil na monitorech všech počítačů v hale zprávu: „Přijďte na promítání v D3 o 18:18“ [41]. Překvapené studující následně dovedl do posluchárny autor článku, přestrojený za proroka Jana.

Videozáznam byl zveřejněn na YouTube kanálu FI MU 1. ledna 2020 [2].

Odkazy

1. KNUTH, Donald. *Fantasia Apocalyptica* [online]. [cit. 2023-09-10]. Dostupné z: `https://www-cs-faculty.stanford.edu/~knuth/fant.html`.
2. FAKULTA INFORMATIKY MASARYKOVY UNIVERZITY. *Fantasia Apocalyptica: Česká premiéra* [online]. YouTube, 2020-01-01 [cit. 2023-09-10]. Dostupné z: `https://youtu.be/wk7dEKMPP68`.
3. LUPTÁK, Dávid. Fantasia Apocalyptica: Česká premiéra. *Zpravodaj CSTUGu*. 2019, roč. 29, č. 1–4, s. 11–18. ISSN 1211-6661. Dostupné z DOI: `10.5300/2019-1-4/11`.
4. SZANISZLO, Tomáš. Dva bloky otázok a odpovedí od Donalda Knutha na FI MU. *Zpravodaj CSTUGu*. 2020, roč. 30, č. 1–2, s. 64–97. ISSN 1211-6661. Dostupné z DOI: `10.5300/2020-1-2/64`.
5. *Laboratoř elektronických multimediálních aplikací (LEMMA) – Filmový festival FI MU: Celouniverzitní spolek* [online]. [cit. 2024-04-08]. Dostupné z: `https://www.muni.cz/studenti/studentske-spolky/lemma-filmovy-festival-fi-mu`.
6. ŠILER, Pavel. *Knuth Q&A Session 1 (video, Sony PXW-X70)* [online]. 2019-10-08. [cit. 2024-04-08]. Dostupné z: `ssh://anxur.fi.muni.cz/export/fi-graphics/2019-10-Turing-week/2019-10-08-video-Knuth-QA1-Sony-Siler/`.
7. ŠILER, Pavel. *Knuth Q&A Session 2 (video, Sony PXW-X70)* [online]. 2019-10-09. [cit. 2024-04-08]. Dostupné z: `ssh://anxur.fi.muni.cz/export/fi-graphics/2019-10-Turing-week/2019-10-09-video-Knuth-QA2-Sony-Siler/`.
8. ŠILER, Pavel. *Fantasia Apocalyptica (video, Sony PXW-X70)* [online]. 2019-10-11. [cit. 2024-04-08]. Dostupné z: `ssh://anxur.fi.muni.cz/export/fi-graphics/2019-10-Turing-week/2019-10-11-video-Fantasia-apocalyptica-Sony-Siler/`.



Obrázek 13: Plakát zvoucí studující a zaměstnance FI MU na promítání videozáznamu přestavení.

9. *FFmpeg: A complete, cross-platform solution to record, convert and stream audio and video* [online]. [cit. 2024-04-11]. Dostupné z: https://ffmpeg.org/.
10. NAGADOMI. *waifu2x: Image Super-Resolution for Anime-Style Art* [online]. [cit. 2019-11-21]. Dostupné z: https://github.com/nagadomi/waifu2x.
11. *FFmpeg Filters Documentation: vidstabdetect* [online]. [cit. 2024-04-08]. Dostupné z: https://ffmpeg.org/ffmpeg-filters.html#vidstabdetect.
12. *FFmpeg Filters Documentation: vidstabtransform* [online]. 2024-04-07. [cit. 2024-04-08]. Dostupné z: https://ffmpeg.org/ffmpeg-filters.html#vidstabtransform.
13. *FFmpeg Filters Documentation: minterpolate* [online]. [cit. 2024-04-08]. Dostupné z: https://ffmpeg.org/ffmpeg-filters.html#minterpolate.
14. *FFmpeg Filters Documentation: yadif* [online]. [cit. 2024-04-08]. Dostupné z: https://ffmpeg.org/ffmpeg-filters.html#yadif-1.
15. SOJKA, Petr. *Fantasia Apocalyptica (zvuk, Zoom H1)* [online]. 2019-10-11. [cit. 2024-04-08]. Dostupné z: ssh://anxur.fi.muni.cz/export/fi-graphics/2019-10-Turing-week/2019-10-11-audio-Fantasia-apocalyptica-ZoomH1-TS/.
16. HOLUBÁŘ, Petr. *Fantasia Apocalyptica (zvuk, mix)* [online]. 2019-10-11. [cit. 2024-04-08]. Dostupné z: ssh://anxur.fi.muni.cz/export/fi-graphics/2019-10-Turing-week/2019-10-11-audio-Fantasia-apocalyptica-Mix-Tudio/.
17. SOJKA, Petr. *Fantasia Apocalyptica (zvuk, Roland R26)* [online]. 2019-10-11. [cit. 2024-04-08]. Dostupné z: ssh://anxur.fi.muni.cz/export/fi-graphics/2019-10-Turing-week/2019-10-11-audio-Fantasia-apocalyptica-RolandR26zLemma-organ/.
18. ŠILER, Pavel. *Fantasia Apocalyptica (zvuk, Sony PXW-X70)* [online]. 2019-10-11. [cit. 2024-04-08]. Dostupné z: ssh://anxur.fi.muni.cz/export/fi-graphics/2019-10-Turing-week/2019-10-11-audio-Fantasia-apocalyptica-Sony-Siler/.
19. *SSL G-Master Buss Compressor: "Glue" tracks into a smooth, cohesive mix* [online]. [cit. 2024-04-29]. Dostupné z: https://www.waves.com/plugins/ssl-g-master-buss-compressor.
20. STARÝ NOVOTNÝ, Vít. *Fantasia Apocalyptica (zvuk, mix)* [online]. 2019-11-19. [cit. 2024-04-08]. Dostupné z: ssh://anxur.fi.muni.cz/export/fi-graphics/2019-10-Turing-week/2019-11-19-audio-Fantasia-apocalyptica-Mix-Novotny/.
21. STARÝ NOVOTNÝ, Vít. *Fantasia Apocalyptica (zvuk, mix)* [online]. 2019-12-09. [cit. 2024-04-08]. Dostupné z: ssh://anxur.fi.muni.cz/export/fi-graphics/2019-10-Turing-week/2019-12-09-audio-Fantasia-apocalyptica-Mix-Novotny/.

22. EDSWARD. *ChurchBellKonstanz.wav: Church bells of the Minster in Constance, Germany* [online]. 2016-04-01. [cit. 2024-03-15]. Dostupné z: https://freesound.org/people/edsward/sounds/341866/.
23. MAČEJOVSKÝ, Šimon. *Úvodní slovo a závěr (video)* [online]. 2019-12-24. [cit. 2024-05-02]. Dostupné z: ssh://anxur.fi.muni.cz/export/fi-graphics/2019-10-Turing-week/2019-12-24-video-strih-4k-Macejovsky/.
24. MORÁVKOVÁ, Martina. *Knuth Q&A Session 1 (fotografie)* [online]. 2019-10-08. [cit. 2024-05-02]. Dostupné z: ssh://anxur.fi.muni.cz/export/fi-graphics/2019-10-08-foto-dek1-Mor%C3%A1vkov%C3%A1/.
25. *FFmpeg Filters Documentation: ass* [online]. [cit. 2024-05-02]. Dostupné z: https://ffmpeg.org/ffmpeg-filters.html#ass.
26. *libass* [online]. [cit. 2024-05-02]. Dostupné z: https://github.com/libass/libass.
27. STARÝ NOVOTNÝ, Vít. *Enable font kerning by default* [online]. 2020-01-02. [cit. 2024-05-02]. Dostupné z: https://github.com/Aegisub/Aegisub/issues/164.
28. STARÝ NOVOTNÝ, Vít. *Úvodní slovo a závěr (titulky)* [online]. 2019-12-12. [cit. 2024-05-02]. Dostupné z: ssh://anxur.fi.muni.cz/export/fi-graphics/2019-10-Turing-week/2019-12-12-titulky-Novotny/.
29. *Aegisub: Advanced Subtitle Editor* [online]. [cit. 2024-05-02]. Dostupné z: https://aegisub.org/.
30. DONG, Chao; LOY, Chen Change; HE, Kaiming; TANG, Xiaoou. Image Super-Resolution Using Deep Convolutional Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2016, roč. 38, č. 2, s. 295–307. Dostupné z DOI: 10.1109/TPAMI.2015.2439281.
31. *FFmpeg: H.265/HEVC Video Encoding Guide* [online]. [cit. 2024-10-03]. Dostupné z: https://trac.ffmpeg.org/wiki/Encode/H.265.
32. *FFmpeg: AAC Audio Encoding Guide* [online]. [cit. 2024-10-31]. Dostupné z: https://trac.ffmpeg.org/wiki/Encode/AAC.
33. KNUTH, Donald; BIBBY, Duane. *Fantasia Apocalyptica Illustrated*. Center for the Study of Language a Information, Stanford, 2018. ISBN 978--1684000449.
34. MORLAND, Isaac. *Fantasia Apocalyptica* [online]. 2018. [cit. 2024-08-05]. Dostupné z: https://www-cs-faculty.stanford.edu/~knuth/fant-screen2-texts.pdf.
35. *PDFtk: The PDF Toolkit* [online]. [cit. 2024-08-26]. Dostupné z: https://www.pdflabs.com/tools/pdftk-the-pdf-toolkit/.
36. *ImageMagick: Mastering Digital Image Alchemy* [online]. [cit. 2024-10-03]. Dostupné z: https://imagemagick.org/.
37. BIBBY, Duane R.; KNUTH, Donald E. *Fantasia Apocalyptica Illustrated*. Stanford, California: Center for the Study of Language a Information,

2018. ISBN 978-1684000449. Dostupné také z: https://www-cs-faculty.stanford.edu/~knuth/fant-screen1-art.pdf.
38. *Flowblade: Free & Libre Video Editor* [online]. [cit. 2024-10-03]. Dostupné z: https://jliljebl.github.io/flowblade/.
39. *MLT Multimedia Framework* [online]. [cit. 2024-10-03]. Dostupné z: https://www.mltframework.org/.
40. *FFmpeg: Concatenating media files* [online]. [cit. 2024-10-03]. Dostupné z: https://trac.ffmpeg.org/wiki/Concatenate.
41. SZANISZLO, Tomáš. *Knuth: Fantasia Apocalyptica D3 call* [online]. 2019-12-18. [cit. 2024-10-23]. Dostupné z: https://gitlab.fi.muni.cz/unix/beeplay/-/commit/68c59a75.

Summary: Recording the Czech premiere of the "Fantasia Apocalyptica" multimedia work

On October 11, 2019, the Czech premiere of the "Fantasia Apocalyptica" multimedia work was held for the 25th anniversary of Masaryk University's Faculty of Informatics, featuring its author, Donald Knuth. A video recording of the performance was taken and published at the YouTube channel of the Faculty of Informatics.

In this article, I describe how the recording was prepared from processing the raw footage through replicating the panels that accompanied the performance to the final composition. Apart from super-resolution imaging, the editing of the intro and the wrap-up, and the compression of sound, only free open-source tools like TEX, Audacity, Aegisub, FFmpeg, MLT, PDFtk, and ImageMagick were used. In the article, I describe how readers can use these tools for their own recordings.

Keywords: footage acquisition, audio-visual production, subtitles, TEX, Audacity, Aegisub, ASS, FFmpeg, ImageMagick, MLT, PDFtk, waifu2x

Vít Starý Novotný, witiko@mail.muni.cz

Responzivní design a automatická sazba s LuaLaTeXem

Michal Hoftich

Tento článek představuje využití metod responzivního designu a pokročilých vlastností LuaLaTeXu pro automatickou sazbu dokumentů určených pro různé cílové výstupy, jak tištěné, tak elektronické, například mobilní telefony, tablety nebo čtečky e-knih.

Konkrétně se zaměřuje na využití LuaLaTeXu pro automatizovanou sazbu s pomocí balíčků *Responsive* [1] pro nastavení velikosti písma a řádkování podle velikosti stránky, *Luavlna* [2] pro zamezení výskytu jednopísmenných předložek na koncích řádků, *Lua-widow-control* [3] pro omezení osamocených řádků na koncích a začátcích stránek a *Linebreaker* [4], který brání přetečení řádků.

Klíčová slova: automatická sazba, responzivní design, LuaLaTeX

Úvod

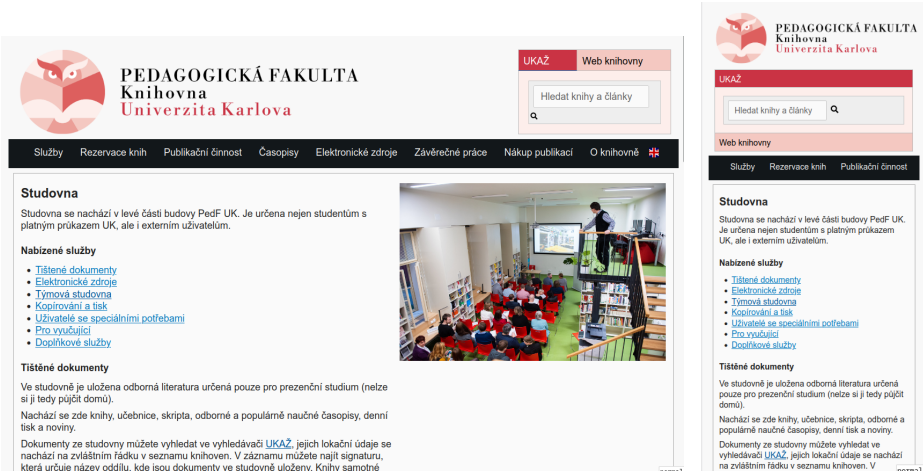
Před časem jsem si pořídil čtečku elektronických knih, ale i přesto většinu textů čtu z obrazovky svého PC, protože pochází z webových zdrojů. Napadlo mě tedy, že bych si delší články ukládal pro pozdější přečtení na čtečce. K tomuto účelu samozřejmě existuje řada aplikací, ale já jsem se rozhodl vytvořit si vlastní, kterou si přizpůsobím přesně svým potřebám a preferencím. Další motivací je možnost naučit se něco nového a vytvořit balíčky, které budou užitečné i pro další uživatele TeXu.

Mým cílem je, aby řešení bylo co nejvíce automatizované, abych například nemusel řešit přetečené řádky nebo jiné chyby, které by vyžadovaly ruční zásah. Díky možnostem LuaLaTeXu je takové řešení již v současnosti možné, jak si ukážeme v následujícím textu.

Responzivní design

Jedním z problémů, který je třeba řešit, je nastavení správné velikosti písma z hlediska čitelnosti. Výchozí velikost písma v LaTeXu je 10 bodů bez ohledu na velikost stránky. To je vhodná velikost písma pro stránku formátu A5. Pro formát A4 by měla být velikost písma větší, pro menší displeje čteček a mobilních telefonů může být naopak menší. Stejně tak můžeme změnit řádkování, které také ovlivňuje čitelnost textu v závislosti na velikosti písma a stránky.

 doi: 10.5300/2024-1-4/28



(a) Ukázka stránky na velkém monitoru

(b) Ukázka stránky na malém displeji

Obrázek 1: Ukázka zobrazení webové stránky s využitím responzivního designu

Podobný problém řeší webové prohlížeče, které musí zobrazit text jak na velkých monitorech PC, tak na menších displejích notebooků, tabletů a mobilních telefonů. Řešení, které používají, se nazývá *responzivní design*.

Responzivní design je způsob návrhu webových stránek, který umožňuje pružné a dynamické přizpůsobení vzhledu a uspořádání obsahu stránky různým zobrazovacím zařízením. Jedním z klíčových prvků responzivního designu je flexibilní struktura, která umožňuje přizpůsobit velikost prvků na stránce zobrazovacímu zařízení.

Dalším důležitým prvkem jsou *media queries*. Ty umožňují definovat pravidla, která se aplikují na základě vlastností zobrazovacího zařízení, jako jsou například šířka a výška obrazovky nebo typ výstupu (papír, displej). Díky těmto pravidlům může být stejný kód stránky dobře zobrazen jak na velkém monitoru, tak na mobilních zařízeních nebo při tisku. Ukázku reálného využití můžete vidět na Obrázku 1.

Balíček *Responsive* [1] se těmito zásadami inspiruje. Jeho hlavní funkcí je nastavení velikosti písma podle velikosti stránky a přibližného počtu znaků, které by se měly vejít na stránku. Dále nastavuje typografickou stupnici (ovlivňuje velikost písma například u nadpisů nebo poznámek pod čarou), písmovou osnovu a podporuje jednoduchou verzi media queries.

Nastavení balíčku *Responsive*

Responsive automaticky nastavuje velikost písma, řádkování a typografickou stupnici na začátku dokumentu. Výchozí hodnoty můžeme změnit pomocí voleb balíčku (`\usepackage[<volby>]{responsive}` nebo příkazem `\ResponsiveSetup{<volby>}`. Příkaz `\ResponsiveSetup` můžeme využít i přímo v textu dokumentu, například pro lokální změny nastavení písma.

Balíček *Responsive* nabízí následující volby:

noautomatic – zabrání nastavení velikosti písma a řádkování automaticky na začátku dokumentu.
characters – počet znaků při automatickém nastavení velikosti písma.
scale – typografická stupnice použitá pro velikosti řezů písma.
lineratio – poměr využitý při výpočtu řádkování.

Základní velikost písma

Velikost písma můžeme nastavit pomocí příkazu `\setsizes{<počet znaků na řádek>}`. *Responsive* se snaží nastavit velikost písma tak, aby na řádku byl v průměru požadovaný počet znaků. Skutečný počet znaků ovšem záleží na použitém textu, neboť při použití proporcionálních fontů má každé písmeno jinou šířku. Ve skutečnosti může být počet znaků zobrazených na řádce mírně vyšší.

Pokud v příkazu `\setsizes` neuvedeme počet znaků, použije se hodnota volby `characters`. Následující příklad využívá právě nastavení této volby. Obrázek 2 ukazuje, jak se stejný text může v daném rámci zobrazit rozdílně v závislosti na nastavených volbách.

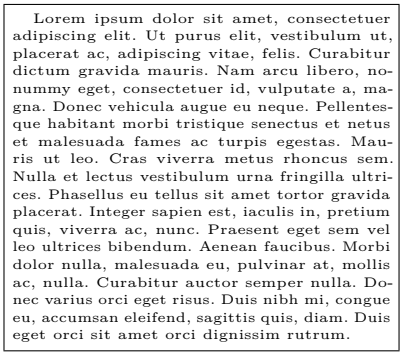
```
\begin{minipage}{5cm}
\ResponsiveSetup{characters=55}
\setsizes{}
\lipsum[1]
\end{minipage}}
```

Řádkování

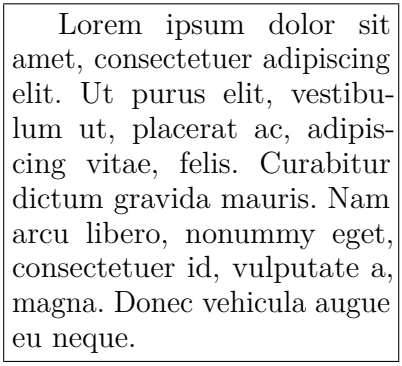
V základním nastavení LaTeXu je řádkování nastaveno na velikost písma násobenou hodnotou 1,2. Pro jiná písma a velikosti stránky může být vhodná jiná velikost řádkování. Stejně tak mohou být vhodné rozdílné hodnoty pro tištěnou a elektronickou verzi dokumentu.

Inspiroval jsem se článkem Edoarda Cavazza [5] o čitelnosti a přidal podporu pro nastavení řádkování na základě poměru výšky malých písmen a proměnné `lineratio`. Ten se získá následujícím výpočtem:

$$\text{řádkování} = \frac{1\text{ex}}{\text{lineratio}/100}$$

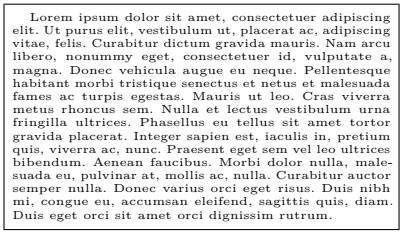


(a) `characters=55`

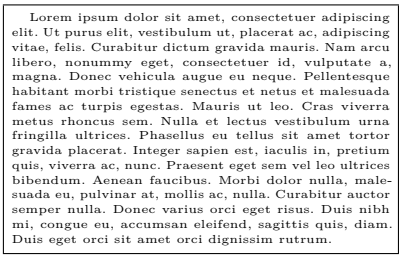


(b) `characters=25, lineratio=38`

Obrázek 2: Rozdíl velikosti písma v závislosti na počtu znaků na řádku



(a) lineratio=38



(b) lineratio=34

Obrázek 3: Změna řádkování změnou hodnoty `lineratio`

Jaký vliv mají změny hodnoty `lineratio`, můžete pozorovat na Obrázku 3. Volba její optimální hodnoty závisí na použitém písmu i velikosti stránky. Pro dosažení maximální čitelnosti výstupu je vhodné porovnat výstup při použití různých hodnot.

Typografická stupnice

Typografická stupnice je soubor předem stanovených velikostí písma, které jsou používány pro vytváření jednotného vizuálního stylu dokumentu nebo webové stránky. Tyto velikosti jsou obvykle vyjádřeny v bodové jednotce a postupně se zvětšují nebo zmenšují o konkrétní interval, který leží na stupnici.

Typografická stupnice může obsahovat velikosti pro nadpisy, poznámky pod čarou a běžný text. Správné používání typografické stupnice pomáhá vytvořit

vizuální hierarchii, která zlepšuje čitelnost a estetický dojem textu. Více informací o typografických stupnicích naleznete v článku Spencera Mortensena [6].

V LaTeXu je typografická stupnice dostupná pomocí příkazů jako `\large`, `\Large`, `\LARGE`, `\huge`, nebo třeba `\scriptsize`. Každý z těchto příkazů je od předešlé velikosti vzdálený o jeden interval. Výchozí stupnice v balíčku *Responsive* je nejbližší stupnici použité v LaTeXu. V Mortensenově článku se nazývá *tetratonická*. Balíček nabízí i další stupnice popsané v článku, například *golden*, založenou na zlatém řezu. Efekt použití stupnice je zobrazen na Obrázku 4.

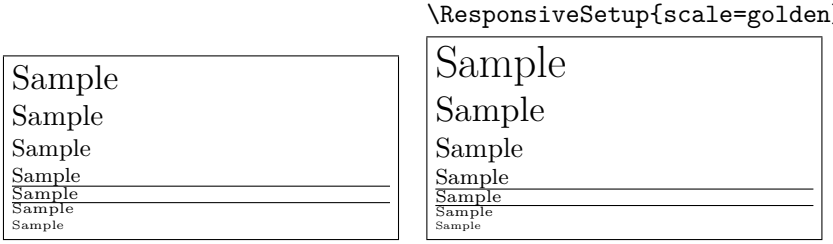
Můžeme však definovat i vlastní stupnici. Například následující kód definuje stupnici, na které se velikost textu zdvojnásobí (`ratio=2`) za tři kroky (`number=3`). Při definici vlastních voleb stupnice je třeba zadat hodnotu `scale=none`. Příkaz `\setsizes` poté redefinuje velikosti písem.

```
\ResponsiveSetup{ratio=2, number=3, scale=none}
\setsizes{}
```

Media queries

Media queries jsou technikou, která umožňuje webovým vývojářům dynamicky přizpůsobovat vzhled a chování webových stránek na základě různých vlastností zařízení, jako jsou například šířka a výška obrazovky, orientace zařízení, podpora barev a mnoho dalších. Díky těmto podmínkám lze vytvářet responzivní a flexibilní webové stránky, které se dokážou automaticky přizpůsobovat různým typům a velikostem zařízení, na kterých jsou zobrazovány.

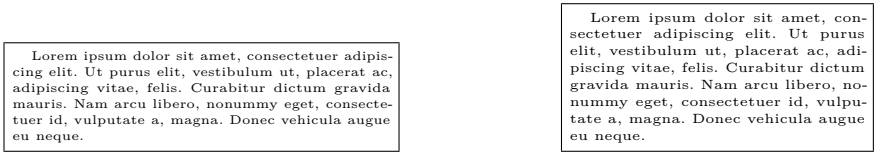
Jak může být tato technika užitečná pro autory LaTeXových balíčků? Mohli by například nastavit velikost písma, řádkování a dalších prvků pro určité rozměry stránky. Poté, co uživatel zvolí velikost stránky podle velikosti zařízení, pro které chce dokument zkompilovat, nastaví se tyto prvky automaticky. Autor balíčku může například definovat, že pokud je šířka textového řádku menší než určitý



(a) Výchozí stupnice (b) Stupnice založená na zlatém řezu

Obrázek 4: Ukázka typografických stupnic (výchozí velikost písma je zvýrazněna linkami)

```
\mediaquery{max-textwidth=4cm}
{\ResponsiveSetup{characters=45}}
{\ResponsiveSetup{characters=60}}
```



(a) Šířka textu 5 cm (b) Šířka textu 3,9 cm

Obrázek 5: Příklad Media Query, která zobrazí méně znaků na řádku, pokud je šířka textu menší než 4 cm

rozměr, zobrazí se na něm méně znaků než na řádcích delších. Výsledek je zobrazen na Obrázku 5.

Media query můžeme deklarovat pomocí příkazu `\mediaquery`, který očekává tři parametry – první je seznam testů, další parametr poté očekává kód, který se vykoná, pokud se testy vyhodnotí jako pravdivé, poslední pak kód spuštěný při nesplněné podmínce. Kód může obsahovat příkaz `\ResponsiveSetup`, ale i libovolné jiné příkazy. Například nastavení velikosti textového bloku, záhlaví a zápatí pomocí balíčku *Geometry*.

Můžeme testovat následující vlastnosti stránky: `paperwidth` a `paperheight` pro rozměry stránky, `textwidth` a `textheight` pro rozměry textu, `orientation` pro orientaci textu a `twocolumn` pro detekci použití dvou sloupců textu v dokumentu.

Testy pro rozměry textu a stránky podporují také předpony `max-` a `min-`. Pomocí nich můžeme testovat, zda daný rozměr je menší nebo větší než zadaná hodnota.

Například následující příkaz změní barvu textu na modrou za podmínky, že dokument má orientaci stránky na šířku, šířka textu je menší než 20 cm a používají se dva sloupce.

```
\mediaquery{orientation=landscape,
max-textwidth=20cm,
twocolumn=true}{\color{blue}}{}
```

Automatická sazba

LuaTEX nám umožňuje automaticky řešit některé problémy, které bylo v minulosti třeba řešit ručním zásahem do dokumentu. Díky tomu, že proces sazby lze ovlivňovat pomocí callbacků v jazyce Lua, můžeme například omezit výskyt

Text s krátkými souhláskami a samohláskami i dalšími jevy z nabídky možností (v textu možnými).
Co třeba í znaky š diakritikou?
Různé možnosti [v závorkách <i jiných znacích
Podpora iniciál a titulů: M. J. Hegel, Ing. Běháková, Ph.D., Ž. Zíbrt, Ch. Borner.
Podpora jednotek: 100,5 MN·s, 100,5 kJ, 200 µA, −1 dag, 1 MB.
Uvnitř matematiky by mělo být zpracování vypnuté: $k \in \mathbb{N}$.

Obrázek 6: Ukázka užití balíčku Luavlna

takových typografických chyb, jako jsou vdovy a sirotci, jednopísmenné předložky na koncích řádků nebo špatně zalomené odstavce. V této sekci si ukážeme několik balíčků, které tyto chyby řeší.

Balíček *Luavlna*

Podle českých a slovenských typografických zvyklostí by se na koncích řádků neměly vyskytovat jednopísmenné předložky nebo spojky. Tomu se dá v TEXu zabránit vložením znaku ~ mezi předložku a následující slovo. Pro zjednodušení této činnosti existuje několik nástrojů, například program *Vlna* Petra Olšáka nebo balíček *Encxvlna* vytvořený také Petrem Olšákem a Zdeňkem Wagnerem.

Balíček *Luavlna* [2] využívá možnosti LuaTEXu pro úpravu plně zpracovaných uzlů textu před zalomením do odstavců. V tomto stadiu jsou již všechna makra expandována, nezlomitelná mezera tedy může být snadno vložena na všechna místa, kde by se měla nacházet.

Luavlna zamezuje zalomení řádků v následujících situacích:

- za jednoznakovými předložkami a spojkami
- u iniciál
- u akademických titulů
- mezi čísly a jednotkami

Ukázku použití balíčku můžete vidět na Obrázku 6. Volby se zadávají pouze pomocí voleb balíčku: `\usepackage[<volby>]{luavlna}`. Nezlomitelné mezery jsou zvýrazněné růžově díky použití volby `debug`. Balíček nabízí řadu dalších voleb, například pro zakázání vkládání nezlomitelných mezer v určitých situacích:

noprocess – nespouštět zpracování dokumentu automaticky
noinitials – nezpracovávat iniciály
nounits – nezpracovávat SI jednotky
nopredegrees – nezpracovávat tituly před jménem

nosufdegrees – nezpracovávat tituly za jménem

Nastavení jednotlivých jazyků se provádí pomocí těchto příkazů:

- `\singlechars{jazyk}{písmena}` – seznam písmen, u kterých se potlačuje zalamování řádků.
- `\enablesplithyphens{jazyk}` – nastaví podporu zalamování spojovníků pro daný jazyk.
- `\preventsinglelang{jazyk}` – nastaví pravidla pro daný jazyk pro celý dokument.

Jazyk lze vybrat pomocí balíčků *Babel* nebo *Polyglossia*. Následující příklad ukazuje, že v českém textu se vloží nezlomitelná mezera, po změně jazyka na angličtinu pomocí `\selectlanguage{english}` se zpracování mezer vypne.

```
A příklad česky.
\selectlanguage{english}
A thing.
```

A příklad česky. A thing.

Zpracování celého dokumentu podle českých pravidel bez ohledu na aktuálně zvolený jazyk lze dosáhnout pomocí příkazu `\preventsinglelang{czech}`.

```
\preventsinglelang{czech}
Příklad v češtině.
\selectlanguage{english}
A thing.
```

Příklad v češtině. A thing.

Balíček *Linebreaker*

Balíček *Linebreaker* [4] brání přetečení textu boxů a odstavců. Příklad jeho použití můžete vidět na Obrázku 7, kde zamezí přetečení několika řádků při sazbě do úzkého sloupce.

Linebreaker využívá callback LuaTEXu `linebreak_filter`, který řídí řádkový zlom. *Linebreaker* nahrazuje výchozí funkci pro řádkový zlom upravenou verzí, která detekuje přetečení nebo podtečení lámaného textu. Při detekci tohoto problému jej sází znovu s většími hodnotami `\tolerance` a `\emergencystretch`, dokud se přetečení nepotlačí nebo dokud se nedosáhne maximální hranice `\tolerance`.

Výhodou je, že tyto změny `\tolerance` a `\emergencystretch` jsou lokální pouze pro aktuálně lámaný odstavec, zbytek textu nijak neovlivní.

Balíček *Linebreaker* můžeme konfigurovat zadáním voleb balíčku `\usepackage[<volby>]{linebreaker}` nebo později i v těle dokumentu s využitím příkazu `\linebreakersetup{<volby>}`:

maxcycles – počet pokusů o přesázení odstavce.
maxemergencystretch – maximální hodnota `\emergencystretch`.
maxtolerance – maximální hodnota `tolerance`.

```
\linebreakersetup{
maxtolerance = 90,          % výchozí 8189
maxemergencystretch = 1em, % výchozí 3em
maxcycles = 4               % výchozí 30
}
```

Balíček *lua-widow-control*

Vdovy a sirotci (souhrně také parchanty) jsou typografickou chybou, která vzniká při zalomení stránek. Osamocený řádek odstavce se někdy ocitne na začátku nebo konci stránky. Sirotek je osamocený řádek na začátku stránky, vdova pak na konci. V TEXu existují dvě penalty, `\clubpenalty` a `\widowpenalty`, které při nastavení vdovy a sirotky potlačují, ovšem za cenu rozšíření vertikálních mezer. Balíček *Lua-widow-control* [3] používá jiný přístup. Každý odstavec sází dvakrát – jednou s výchozími parametry, druhý o jeden řádek delší. To může mít vliv na rychlost kompilace dokumentu, v praxi by ale měl být malý. V případě, že se při zalomení stránky detekuje parchant, vymění se předposlední odstavec na stránce za jeho verzi s delším řádkem, čímž dojde k posunutí problematického řádku.

Balíček *Lua-widow-control* lze nastavit pomocí několika voleb. Ty je možné zadat jako čárkou oddělené volby v příkazu `\usepackage[<volby>]{lua-widow-control}`, nebo pomocí příkazu `\lwcsetup{<volby>}`.

The example document given below creates two pages by using Lua code alone. You will learn how to access TeX's boxes and counters from the Lua side, shipout a page into the PDF file, create horizontal and vertical boxes (hbox and vbox), create new nodes and manipulate the nodes links structure.

(a) Bez balíčku *Linebreaker*

The example document given below creates two pages by using Lua code alone. You will learn how to access TeX's boxes and counters from the Lua side, shipout a page into the PDF file, create horizontal and vertical boxes (hbox and vbox), create new nodes and manipulate the nodes links structure.

(b) S balíčkem *Linebreaker*

Obrázek 7: Příklad využití balíčku *Linebreaker*

Následující volby umožňují změnit metodu použitou pro zamezení parchantů:

default – nepřidává žádné vertikální mezery, používá pouze prodloužení odstavců. Tato metoda by měla odstranit 95 % parchantů, může ovšem vést k příliš vzdušným odstavcům.

strict – přísnější nastavení, které dbá, aby změny zalomení odstavců byly co nejméně postřehnutelné. Odstraní ovšem pouze třetinu parchantů.

balanced – nejdříve přidává vertikální mezery, pokud to nestačí, prodlouží odstavce. Tato metoda odstraní až 90 % parchantů.

Tento výčet je pouze malým úvodem k balíčku *Lua-widow-control*. Podrobný popis naleznete v článku Maxe Chernoffa [7], který vyšel v loňském Zpravodaji.

Odkazy

1. HOFTICH, Michal. *The Responsive package* [online]. 2023. Ver. 0.1 [cit. 2023-03-30]. Dostupné z: `https://www.ctan.org/pkg/responsive`.
2. HOFTICH, Michal. *The Luavlna package: Prevent line breaks after single letter words, units, or academic titles* [online]. 2023. 0.1k [cit. 2023-03-30]. Dostupné z: `https://ctan.org/pkg/luavlna`.
3. CHERNOFF, Max. *The Lua-widow-control package: Automatically remove widows and orphans from any document* [online]. 2022. Ver. 3.0.0 [cit. 2023-03-30]. Dostupné z: `https://ctan.org/pkg/lua-widow-control`.
4. HOFTICH, Michal. *The Linebreaker package: Prevent overflow boxes with LuaLaTeX* [online]. 2023. Ver. 0.1c [cit. 2023-03-30]. Dostupné z: `https://ctan.org/pkg/linebreaker`.
5. CAVAZZA, Edoardo. Modern CSS Techniques To Improve Legibility. *Smashing Magazine* [online]. 2020 [cit. 2023-03-30]. Dostupné z: `https://www.smashingmagazine.com/2020/07/css-techniques-legibility/`.
6. MORTENSEN, Spencer. *The Typographic Scale* [online]. 2011. [cit. 2023-03-30]. Dostupné z: `https://spencermortensen.com/articles/typographic-scale/`.
7. CHERNOFF, Max. Automatically Removing Widows and Orphans with lua-widow-control. *Zpravodaj Československého sdružení uživatelů TeXu*. 2022, roč. 32, č. 1, s. 49–76. Dostupné z DOI: `10.5300/2022-1-4/49`.

Summary: Responsive Design and Automatic Typesetting with LuaLaTeX

This article focuses on the use of responsive design techniques to display web pages on devices with different display sizes, such as mobile phones, tablets, large monitors and printers. These methods allow optimizing the readability of

a document on all devices by using different font sizes, individual page elements, and margins.

We present how similar functionality can be achieved using LaTeX. Specifically, it focuses on the use of LuaLaTeX for automated typesetting, using packages *Responsive* for setting font size and line spacing according to page size, *Luavlna* to prevent the occurrence of single-letter prepositions at line breaks, *Lua-widow-control* to reduce orphan lines at page breaks and page starts, and *Linebreaker* to prevent line overflow.

With these methods, a single source document can be used for different outputs, such as print versions, e-book readers, and web pages, and achieve optimal document display on all devices.

Keywords: automatic typesetting, responsive design, LuaLaTeX

Michal Hoftich, michal.h21@gmail.com

Overleaf Community Edition: Postup inštalácie a rozdiely oproti Overleaf Professional

Michal Stano, Vít Starý Novotný

Webový editor Overleaf umožňuje pohodlné písanie dokumentov pomocou LaTeXu. Veľkou výhodou Overleafu je, že umožňuje spoluprácu viacerých používateľov v reálnom čase. V tomto článku popíšeme postup inštalácie bezplatnej verzie Overleaf Community Edition na vlastné zariadenie a porovnáme ho s platenou verziou Overleaf Professional.

Kľúčové slová: Overleaf, LaTeX, Docker

Úvod

Vstup do sveta TeXu bol pre prvého autora tohto článku náhly a chaotický. V druhom semestri štúdia na Fakulte informatiky Masarykovej univerzity si zapísal predmet *Počítačové sítě – cvičení*. Po každom cvičení dostali študenti za úlohu vypracovať protokol do šablóny v TeXu, k čomu im bol odporučený webový editor *Overleaf* [1]. Keďže prvý autor o TeXu vtedy ešte nič nevedel a namiesto prečítania hocijakej príručky, manuálu alebo knihy sa rovno vrhol do práce, daný dokument dva dni štýlom pokus–omyl lepil dohromady z kusov kódu z rôznych fór, napr. StackExchange [2], so značnou pomocou ChatGPT.

Prvého autora neskôr používanie TeXu začalo baviť, ale kvôli jeho štýlu písania „napíšem, skompilujem a uvidím“ stále preferoval Overleaf oproti manuálnej kompilácii. Preto ho neskôr prekvapilo, keď Overleaf zrazu jeho dokument skompilovať nechcel. Používal totižto bezplatnú verziu *Overleaf Free*. Nevedel, či bol jeho projekt len priveľký, alebo sa musel kompilovať niekoľkokrát, ale podarilo sa mu prekonať časový limit kompilácie.

Overleafu sa vzdať nechcel, a tak rýchlo hľadal spôsob, ako časový limit a kompilačnú kapacitu Overleafu navýšiť. Okrem platenej verzie *Premium* vo variantoch *Standard* a *Professional* (ďalej len *Overleaf Pro*), bola ešte možnosť stiahnuť si software *Overleaf Toolkit* a Overleaf prevádzkovať lokálne na vlastnom zariadení alebo serveri. Overleaf Toolkit je dostupný vo dvoch verziách, *Community Edition* a *Enterprise*. Community Edition (ďalej len *Overleaf CE*) je dostupná zadarmo, ale bez profesionálnej podpory, zatiaľ čo Enterprise je platená, určená najmä pre veľké (napr. firemné) serveri a s profesionálnou podporou. Zvolil si preto Overleaf CE, keďže Overleaf chcel len na vlastné použitie. Proces inštalácie vyzeral jednoducho, a tak sa do toho pustil.

doi: 10.5300/2024-1-4/39

Príprava systému

Pred samotnou inštaláciou je potrebné si pár vecí pripraviť. Keďže prvž autor použil Linux Mint, ktorý vychádza z Ubuntu a teda Debianu, popíšeme postup prispôsobený pre Debianové distribúcie Linuxu. Ak niektorý príkaz zlyhá s výstupom „permission denied“, treba pred daný príkaz napísať `sudo`.

Na inštaláciu a fungovanie Overleafu je potrebné mať nainštalované *Bash* a *Docker*. Neskôr pri inštalácií TEXu je potrebný aj *Perl*.

Docker môžeme nainštalovať nasledujúcimi príkazmi:

```
$ apt update
$ apt install docker docker.io docker-compose-v2
```

Ak by neskôr počas inštalácie Docker písal „cannot connect to the Docker daemon“, je potrebné zadať nasledujúci príkaz:

```
$ systemctl start docker
```

Bash a Perl sa na Linuxe väčšinou nachádzajú natívne.

Inštalácia Overleafu CE

Overleaf CE nainštalujeme postupom vysvetleným v oddieli *Installation* súboru `README.md` v Git repozitári Overleaf-toolkit [3].

Najprv si do vybratého priečinku stiahneme súbory z vyššie spomenutého úložiska a po stiahnutí vstúpime do vytvoreného priečinku:

```
$ git clone https://github.com/overleaf/toolkit ./overleaf-toolkit
$ cd overleaf-toolkit/
```

Každý príkaz odteraz vychádza z toho, že pracujeme v tomto priečinku.

Konfiguráciu vytvoríme príkazom `$ bin/init`, ktorý v priečinku `config` vytvorí súbory `overleaf.rc`, `variables.env` a `version`. V prípade, že chceme povoliť prístup z iných zariadení, napríklad v prípade používania serveru na hosťovanie Overleafu, v súbore `overleaf.rc` je treba pomocou textového editoru priamo v termináli, napr. Vim alebo Nano, hodnotu `SHARELATEX_LISTEN_IP` prestaviť zo 127.0.0.1 na 0.0.0.0 alebo na IP adresu konkrétneho sieťového rozhrania.

Ostáva Overleaf spustiť jedným z dvoch nasledujúcich príkazov:

```
$ bin/up
$ bin/start
```

Tieto príkazy sú vo výsledku rovnaké, majú len jeden menší rozdiel, ktorý hneď popíšeme. Oba však na pozadí odkazujú na sériu skriptov, ktoré pre nás automaticky spustia celý Overleaf systém, t. j. postarajú sa o spustenie `Docker Compose`, ktorý spravuje 3 docker kontajnery: *redis*, *mongo* a *sharelatex*.

Pri použití príkazu `$ bin/up` sa terminál, alebo jeho inštancia, na ktorej bol príkaz spustený, prepojí so záznamom výstupu. Preto je odporúčané prvýkrát použiť tento príkaz, keďže je priamo v termináli vidieť, čo sa so systémom deje. Nevýhodou tohto príkazu je zablokovanie daného terminálu pre ďalšie použitie až do zastavenia procesu. Vypnúť kontajnery (a odblokovať daný terminál) sa dá len pomocou klávesovej skratky `CTRL + C`.

Príkaz `$ bin/start` spustí kontajnery, na terminál vypíše stav jednotlivých kontajnerov (napr.: ✓ `Container sharelatex Started`), ale inak nechá terminál tak. Vypnúť sa dá príkazom `$ bin/stop`.

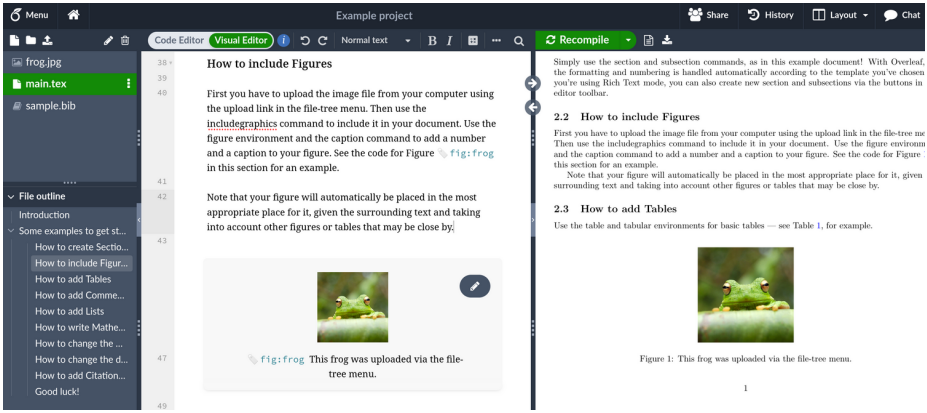
Ak sa služba spustila bez problémov, t. j. výpis príkazu `$ bin/up` neobsahuje chyby, alebo všetky kontajnery majú vo výpise príkazu `$ bin/start` status „Started“ alebo „Healthy“, Overleaf funguje. Pred začiatkom práce môžeme ešte aktualizovať distribúciu TEX Live [4] v kontajneri sharelatex následne:

```
$ bin/shell
# tlmgr update --self --all
```

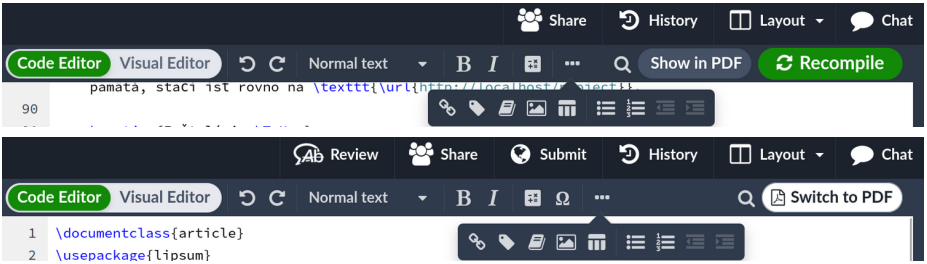
Účet administrátora, prihlásenie a prvý projekt

Na vytvorenie prvého projektu stačí vytvoriť účet administrátora a prihlásiť sa.

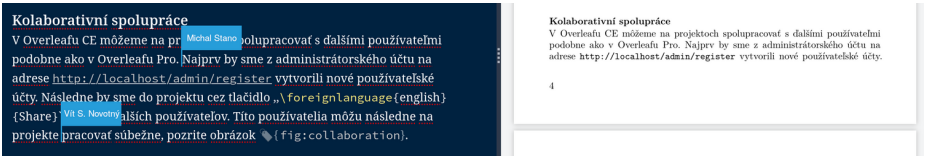
Vo webovom prehliadači na adrese `http://localhost/launchpad` vyplníme do kolóniek „email“ a „heslo“ prihlasovacie údaje administrátora a následne sa na adrese `http://localhost/login` prihlásime pomocou týchto prihlasovacích údajov. Po prihlásení budeme automaticky presmerovaní na adresu `http://localhost/project` so zoznamom všetkých projektov. Na obrázku 1 je zobrazený ukážkový projekt, ktorý je distribuovaný spolu s Overleafom CE.



Obr. 1: Ukážkový projekt v Overleafe CE



Obr. 2: Rozhranie Overleafu CE (hore) a Overleafu Pro (dole)



Obr. 3: Spolupráca dvoch používateľov v Overleafe CE

Porovnanie Overleafu CE a Overleafu Pro

Neskôr počas semestru sa prvý autor dozvedel, že ako študent Fakulty informatiky Masarykovej univerzity má možnosť aktivovať si zadarmo Overleaf Pro. Po aktivácii účtu sa rozhodol, že Overleaf Pro a Overleaf CE porovná.

Používateľské rozhranie

Veľké rozdiely medzi Overleafmi CE a Pro sme nezaregistrovali až na pár zmien v užívateľskom rozhraní, pozrite obrázok 2.

V Overleaf Pro sú pridané tlačidlá „Publish“, ktorými je možné projekt zverejniť na preprintových archívoch, alebo ho ponúknuť na vydanie partnerským vydavateľstvám, a „Review“, ktoré umožňuje zobraziť komentáre a navrhované zmeny od korektora a spoluautorov. Taktiež je v hornej lište pridané tlačidlo Ω, ktoré otvorí knižnicu symbolov a špeciálnych znakov. V Overleafe CE tieto funkcie nie sú z viacerých dôvodov. „Publish“ a „Review“ nie sú zahrnuté kvôli tomu, že sme izolovaní od serveru Overleaf. Špeciálne znaky sú súčasťou výhod verzie Premium a teda sú nedostupné v Overleafe CE ako aj vo verzii Free.

Až na pár drobností je aj samotný textový editor rovnaký. Jediné, čo by prvý autor tvorcom vytkol je, že Overleaf CE pri príkaze `\cite{}` alebo `\includesvg{}` nedopĺňa kľúč. Zaujímavé je, že po `\includegraphics{}` ich dopĺňa.

Kolaboratívna spolupráca
V Overleafe CE môžeme na projektoch spolupracovať s ďalšími používateľmi podobne ako v Overleafe Pro. Najprv by sme z administrátorského účtu na adrese `http://localhost/admin/register` vytvorili nové používateľské účty. Následne by sme do projektu cez tlačidlo „Share“ pozvali ďalších používateľov. Títo používatelia potom môžu na projekte pracovať súbežne, pozrite obrázok 3.

Záver

Webový editor Overleaf [1] umožňuje pohodlné písanie dokumentov pomocou LaTeXu. V tomto článku sme ukázali, ako môžeme nainštalovať bezplatnú self-hosted verziu Overleaf CE na vlastný laptop alebo firemný server, vytvárať projekty a spolupracovať na nich s ďalšími používateľmi. Sme presvedčení, že pre mnohých užívateľov je Overleaf CE zaujímavou alternatívou oproti cloudovým verziám Overleafu, ktorá umožňuje používať na sadzbu LaTeXových dokumentov vlastný hardware a vyhnúť sa obmedzeniam verzie Overleaf Free.

Referencie

1. NOVOTNÝ, Vít. Overleaf: Kolaborativní webový editor LaTeXu. *Zpravodaj CSTUGu*. 2021, roč. 31, č. 1–4, s. 3–8. ISSN 1211-6661. Dostupné z DOI: 10.5300/2021-1-4/3.
2. *TeX – LaTeX Stack Exchange* [online]. [cit. 2024-04-08]. Dostupné z: `https://tex.stackexchange.com/`.
3. OVERLEAF TEAM. *Overleaf github page* [online]. [cit. 2024-02-26]. Dostupné z: `https://github.com/overleaf/overleaf`.
4. *TeX Live* [online]. [cit. 2024-02-26]. Dostupné z: `http://tug.org/texlive/`.

Summary: Overleaf Community Edition: Installation and Comparison with Overleaf Professional

The Overleaf web editor enables convenient writing of documents using LaTeX. A significant advantage of Overleaf is that it enables collaboration among multiple users in real-time. In this article, we describe the process of installing the free version of Overleaf Community Edition on your own device and compare it with the paid version of Overleaf Professional.

Keywords: Overleaf, LaTeX, Docker

Michal Stano, astakus@mail.muni.cz
Vít Starý Novotný, witiko@mail.muni.cz

Profilování TEXových dokumentů

Martin Ruckert

Profiler je nástroj pro analýzu rychlosti kódu. Profiler může poskytovat informace o rychlosti jednotlivých řádků kódu nebo i celých procedur. Tyto informace jsou užitečné při optimalizaci kódu pro dosažení maximální rychlosti.

Ačkoliv jsou profilery běžně dostupné, doposud neexistoval profiler pro programátory, kteří připravují TEXová makra. V tomto článku představuji `texprof` a `texprofile`: dva programy, které společně tvoří profiler pro TEXové dokumenty.

Klíčová slova: TEX, profilování, `texprof`, `texprofile`

1 Kdo potřebuje profiler?

TEXový profiler je nástroj pro programátory, kteří píšou makra pro TEX. To však neznamená, že autor, který příležitostně napíše nějaké makro v TEXu, by měl tento nástroj používat nebo že ho vůbec potřebuje. Optimalizace maker pro rychlost by měla být prováděna pouze tehdy, pokud se makro používá *velmi* často.

Abychom získali lepší představu, co znamená „*velmi* často“, provedeme následující úvahu: Za rozumných předpokladů[1] vede jedna sekunda CPU času k produkci 28 mg CO_2. Opět za rozumných předpokladů[2] způsobí jízda 200 km emise 28 kg CO_2. To znamená, že bychom museli ušetřit miliony sekund CPU času, aby to mělo nějaký podstatný vliv na naši uhlíkovou stopu nebo rozpočet.

Příležitostní autoři TEXových maker mají lepší způsoby, jak trávit čas, než optimalizovat výkon. Existují ale oblíbené makrobalíčky s miliony uživatelů, jejichž makra se spouští mnohokrát za den. Pokud jsme autorem takového balíčku, možná nás zajímá, zda existují příležitosti k optimalizaci, kde tyto příležitosti v kódu hledat a jak velkého zrychlení můžeme docílit.

Možná ještě důležitější je, že profiler nám může říci, kde optimalizaci nehledat a zda změny v kódu přinesly požadovaný efekt. Obecně platí, že bychom nikdy neměli optimalizovat kód pro rychlost bez použití profileru.

2 Jak funguje TEXový profiler?

2.1 Mapování příkazů na vstupní soubory a řádky

TEXový stroj je interpretr, který čte vstupní soubory a vykonává vestavěné příkazy TEXu, jako je například vytvoření horizontálního boxu, zvýšení hodnoty registru

Z anglického originálu [1] přeložil se svolením autora a vydavatele Vít Starý Novotný.

[1] Spotřeba našeho počítače je 200 wattů a na kWh elektřiny připadá emise 500 g CO_2.

[2] Spotřeba našeho vozu je 6 l paliva na 100 km a na litr paliva připadá emise 2370 g CO_2.

 DOI: 10.5300/2024-1-4/44

čítače nebo přidání znaku do aktuálního odstavce. TEXový profiler, nazvaný `texprof`, je TEXový stroj s rozšířeními, která umožňují mapovat každý příkaz na vstupní soubor a na řádek v tomto souboru. K tomuto účelu využívá `texprof` datové struktury, které si každý TEXový stroj (dále jen TEX, pozn. překl.) udržuje, aby mohl zobrazovat užitečná hlášení v případě chyby. Pokud je příkaz součástí formátového souboru, název vstupního souboru a číslo řádku nejsou známy. Dále se podíváme, jak se použití formátového souboru můžeme vyhnout.

I když se vyhneme použití formátového souboru, mnoho příkazů nepochází přímo ze vstupního souboru, ale z výsledku použití maker. Když definujeme makro, TEX uloží příkazy z těla makra do tabulky. Když makro použijeme, TEX načte z tabulky příkazy a vloží je na vstup. Jelikož jsou název vstupního souboru a číslo řádku známy během definice makra, může `texprof` tyto informace také uložit do tabulky a načíst je spolu s příkazy při použití makra. Stejný mechanismus používá `texprof` také v případě, že načítá vstup, například při hledání klíčového slova, a zjistí, že část příkazů musí vrátit na vstup pro pozdější zpracování. Tyto příkazy si s sebou také nesou informace o názvu vstupního souboru a čísle řádku.

Existuje několik vzácných případů, na které se výše uvedený mechanismus nevztahuje. TEX např. vkládá složené závorky kolem výstupní rutiny, aby zajistil, že příkazy provedené ve výstupní rutině nezapříčiní neočekávané globální změny. Z pohledu `texprof`u tyto složené závorky pochází z fiktivního řádku nula ve fiktivním vstupním souboru nazvaném „systém“.

Existují i další fiktivní čísla řádků ve fiktivním vstupním souboru „systém“, která `texprof` používá. TEX vyvolává systémové procedury, jako je rozdělení odstavce na řádky nebo zapsání stránky do souboru DVI, které mohou být časově náročné. Bylo by zavádějící, kdybychom čas strávený v těchto rutinách přiřadili příkazu, který byl proveden na konci odstavce nebo způsobil zapsání stránky do souboru DVI. Proto `texprof` přiřazuje tyto časy k fiktivnímu souboru „systém“ a používá číslo řádku k označení odpovídající systémové procedury TEXu.

2.2 Mapování doby provádění příkazů

Většina příkazů TEXu se vykonává v proceduře `main_control` TEXu. V této proceduře najdeme smyčku, která čte příkazy a poté je vykonává pomocí kódu za návěštím `big_switch` (dále jen „za `big_switch`em“, pozn. překl.). Na začátku každé iterace této smyčky zaznamená `texprof` čas z hardwarových hodin pomocí nízkoúrovňové rutiny operačního systému.

Zaznamenaný čas je začátkem nového časového intervalu a zároveň koncem předchozího časového intervalu. Po zaznamenání počátečního času `texprof` pokračuje v běžném zpracování TEXu a načte další příkaz ze vstupu. Jakmile je příkaz znám, `texprof` zaznamená příkaz, vstupní soubor, číslo řádku (a také makro, ze kterého příkaz pochází; zaměříme se teď ale pouze na příkazy, makra

budou vysvětlena později). Poté pokračuje běžné zpracování a příkaz je vykonán pomocí `big_switch`e, který skončí návratem na začátek smyčky. Tam `texprof` zaznamená koncový čas, vypočítá rozdíl a uloží příkaz, vstupní soubor, číslo řádku a časový rozdíl do velkého pole.

Občas je příkaz v `big_switch`i nahrazen jiným příkazem a následně se `big_switch` použije znovu k jeho vykonání. `texprof` tuto výměnu ignoruje a zaznamená celý čas společně s příkazem, vstupním souborem a číslem řádku, které byly získány na začátku aktuální iterace. To do měření zavádí určitou nepřesnost, ale nezpůsobuje to výrazné chyby.

Mnohem větší vliv na přiřazení času souboru a řádku má existence hlavní smyčky TEXu za návěštím `main_loop`. Kód za návěštím `big_switch` do této smyčky přejde vždy, když narazí na znak, a zůstane v ní, dokud čte příkazy typické pro běžný text: znaky, mezery, kerning, ligatury, změny písma a další podobné věci. Celý čas strávený v této smyčce je poté zaznamenán s příkazem, vstupním souborem a řádkem, které tuto smyčku zahájily. Čas potřebný ke zpracování celého odstavce tak může být přiřazen k prvnímu písmenu tohoto odstavce.

Rozhodnutí nepřiřazovat čas přesněji je v tomto případě odůvodněno následujícími úvahami: Za prvé, takový odstavec je obvykle zpracován pouze jednou a doba zpracování obvykle nepředstavuje významnou část celkového běhu programu. Za druhé, při použití profileru nás obvykle nezajímá čas strávený nad písmeny, mezerami a dalšími částmi běžného textu. Koneckonců žádný autor neoptimalizuje text pro rychlost jeho zpracování. Toto rozhodnutí také výrazně snižuje počet časových intervalů, které musí `texprof` zaznamenávat.

Procedura `main_control` končí vykonáním příkazu `stop`, který zároveň ukončí zaznamenávání časových intervalů. Jako součást závěrečné procedury TEXu `texprof` zapíše všechna sesbíraná data do binárního souboru. Název tohoto souboru je získán připojením přípony `.tprof` k názvu vstupního souboru z vestavěného příkazu `\jobname`. Další zpracování sesbíraných dat neprovádí `texprof`, ale druhý program nazvaný `texprofile`. Jeho použití bude ilustrováno dále.

2.3 Problém měření času

Operační systémy obvykle poskytují několik hodin, ze kterých lze vybírat. `texprof` používá POSIXovou funkci `clock_gettime`, která měří CPU čas aktuálního vlákna v nanosekundách. Ačkoliv nám čas v nanosekundách může připadat jako velmi přesný, skutečná přesnost je spíše v řádu stovek mikrosekund. To má několik důvodů, které si nyní popíšeme.

Moderní počítače provádí mnoho procesů současně a sdílí dostupná CPU mezi prováděnými procesy, přičemž čas potřebný k přepnutí procesů zahrnuje zneplatnění obsahu keše. Instrukce pro načtení dat z hlavní paměti může trvat podstatně déle, pokud se daná paměťová oblast již nenachází v keši. Tento dodatečný čas je poté přiřazen k příkazu TEXu, který se zrovna vykonává.

Moderní CPU také používají techniku zvanou frekvenční škálování pro snížení spotřeby energie. Pokud jen píšeme text v editoru, může náš notebook běžet s frekvencí nižší než 1 GHz. Krátce poté, co spustíte TEX, si operační systém může všimnout, že je potřeba vykonat hodně práce, a může zvýšit frekvenci až na 4 GHz. Všechny příkazy, které se vykonají po této změně, zaberou méně času než stejné příkazy před touto změnou.

V posledních letech začali výrobci kombinovat na jednom čipu několik výkonnostních jader s několika jádry úspornými. Zatímco výkonnostní jádra používají sofistikované superskalární pipeline pro vykonání více instrukcí v jednom cyklu, úsporná jádra používají jednoduché sekvenční provádění instrukcí, které sice spotřebuje podstatně méně energie a produkuje mnohem méně tepla, ale může potřebovat více cyklů na jednu instrukci.

Pokud se operační systém rozhodne přesunout `texprof` z úsporného jádra na výkonnostní jádro uprostřed běhu, bude to mít drastický vliv na časy příkazů. Operační systém ho také může přesunout zpět na úsporné jádro, pokud `texprof` začne provádět vstupně-výstupní operace a musí čekat na disk.

Existuje několik možností, jak tyto efekty zmírnit, například výpočtem průměru přes více běhů nebo použitím syntetických časů. Žádná z těchto metod však není momentálně implementována.

2.4 Mapování výpočetních časů na makra

Když TEX narazí na aktivní znak nebo řídící sekvenci, ví, že musí vykonat primitivní příkaz nebo makro. U makra vyhledá seznam příkazů, které tvoří tělo makra, a tento seznam vloží na svůj zásobník vstupů. Když TEX potřebuje další příkaz ze vstupu, vezme ho z nejvyššího seznamu na zásobníku. Jakmile TEX dosáhne konce tohoto seznamu, odstraní ho ze zásobníku a pokračuje ve čtení příkazů ze seznamu pod ním.

Zásobník vstupů se nepoužívá pouze při použití maker, ale také pro implicitní volání jiných rutin TEXu: Když je připravena nová stránka, konstruktor stránek vloží na zásobník vstupů výstupní rutinu. Na začátku odstavce systém vloží na zásobník příkazy z vestavěného příkazu `\everypar`. Při použití vestavěného příkazu `\input` se na zásobník vloží celé soubory. Je také běžné, že TEX načítá vstup, například kvůli kontrole klíčového slova, a vloží nepoužité tokeny zpět na zásobník, pokud klíčové slovo nebylo nalezeno.

Většinu informací, které `texprof` potřebuje ke sledování maker, lze nalézt na zásobníku vstupů TEXu, ale ne všechny. Zásobník vstupů totiž neobsahuje informace o správné úrovni zanoření maker. Důvodem je chytrá optimalizace, kterou TEX používá na svém zásobníku vstupů pro podporu koncové rekurze: Před vložením těla nového makra na zásobník TEX opakovaně kontroluje, zda je nejvyšší seznam na zásobníku již prázdný. Pokud ano, odstraní prázdný seznam ze zásobníku. Teprve poté, co jsou všechny prázdné seznamy odstraněny, je tělo nového makra vloženo na zásobník.

Díky této optimalizaci může smyčka, která je implementována jako rekurzivní makro provádějící samo sebe jako poslední akci těla makra, běžet bez přetečení zásobníku. Bohužel tato technika odstraní makro ze zásobníku vstupů, zatímco jeho poslední podmakro stále běží, a nové makro se tak vloží na nižší úroveň zanoření, než je jeho skutečná úroveň zanoření.

Proto `texprof` přidává informace o skutečné aktuální úrovni zanoření maker do zásobníku vstupů TEXu a udržuje podpůrný zásobník, který obsahuje informace o všech makrech až po skutečnou úroveň zanoření: název makra, vstupní soubor a číslo řádku, kde bylo makro definováno. Tento podpůrný zásobník poskytuje informace o skutečném začátku a konci provádění makra, které jsou zaznamenávány společně s časovými údaji o vykonaných příkazech.

3 Analýza profilovacích dat

Surová data, která `texprof` zapisuje do výstupního souboru, jsou jen dlouhý seznam tisíců záznamů ve formátu „příkaz, vstupní soubor, řádek, čas“ prokládaných záznamy, které odrážejí změny na zásobníku maker. Extrahování užitečných informací z těchto dat je úkolem samostatného programu nazvaného `texprofile`.

Pro vysvětlení použití `texprof`u a `texprofile`ru je nejlepší použít skutečné příklady. Pro první příklad jsem hledal velký dokument zaměřený na text. Hledáním na internetu jsem našel TEXovou verzi Bible [2]. Několika změnami jsem zajistil, že používá formát plain TEX, takže využívá pouze omezené množství maker. Většina maker pochází z plain TEXu, ale jsou zde také uživatelsky definovaná makra.

Spuštěním příkazu `texprof -prof bible` vznikne soubor `bible.tprof` o velikosti 17 MB. Po spuštění příkazu `tprof bible` bez dalších voleb se zobrazí následující souhrn:

Celkový čas měření:	728,92 ms
Celkový počet vzorků:	2 157 642
Průměrný čas na vzorek:	337,00 ns
Celkový počet souborů:	69
Celkový počet maker:	1 097
Maximální hloubka zásobníku:	7

Pomocí dalších voleb můžeme specifikovat, které datové tabulky má `texprofile` zobrazit a jak by měl informace zobrazovat.

3.1 Deset nejpomalejších řádků

Při použití volby `-T` projde `texprofile` data, sečte časy pro každou kombinaci vstupního souboru a řádku, seřadí výsledky a zobrazí deset nejpomalejších řádků.

soubor	řádek	procenta	čas [ms]	počet	průměr	soubor
systém	shipout	17,68 %	156,05	1 130	138,09 µs	systém
5	29	17,63 %	155,65	54 649	2,85 µs	`bible.tex`
systém	linebrk	15,21 %	134,26	25 777	5,21 µs	systém
systém	buildpg	1,69 %	14,89	55 190	269,00 ns	systém
5	56	0,86 %	7,61	4 750	1,60 µs	`bible.tex`
5	15	0,62 %	5,43	6 183	878,00 ns	`bible.tex`
3	555	0,47 %	4,17	8 549	487,00 ns	`plain.tex`
3	1204	0,28 %	2,44	3 390	719,00 ns	`plain.tex`
3	1201	0,26 %	2,33	2 260	1,03 µs	`plain.tex`
3	1203	0,25 %	2,20	2 258	973,00 ns	`plain.tex`

Tabulka 1: Výstup příkazu `texprofile -T bible`

Výstup příkazu `texprofile -T bible` je zobrazen v Tabulce 1.

První řádek je přiřazen k fiktivnímu souboru „systém“. Záznam ukazuje kumulativní čas důležité systémové rutiny, přičemž číslo řádku identifikuje konkrétní rutinu. Zde jde o zapsání stránky do souboru DVI (shipout), o něco níže pak vidíme rozdělení odstavce na řádky (linebrk) a rozdělení dokumentu na stránky (buildpg). Uvedené časy nezávisí na použití maker, ale pouze na velikosti dokumentu.

Na druhém řádku tabulky vidíme, že řádek 29 v souboru `bible.tex` je zodpovědný za 17,63 % celkové doby běhu programu, a je tedy vhodným kandidátem pro optimalizaci, o kterou se pokusíme v následující sekci. Řádek samotný je poměrně rychlý, v průměru zabere pouze 2,85 µs, ale je používán velmi často: 54 649krát.

To, že zbývajících šest řádků přispívá k celkové době běhu programu méně než 1 %, znamená, že nemusíme uvažovat o jejich optimalizaci.

3.2 Optimalizace makra `\Verse`

Řádek 29 v souboru `bible.tex` definuje makro `\Verse`, vizte Ukázku 1. Makro se používá k vysázení čísla verše zvýšeným textem, vizte Obrázek 1. Toto makro můžeme rychlostně optimalizovat následně: Předpona `\global` není potřeba, protože makro je používáno pouze na globální úrovni. Klíčové slovo `by` je volitelné a můžeme ho vynechat. Každý literál jako `1` je v těle makra uložený jako posloupnost znaků. Tyto znaky jsou při každém vykonání makra opětovně načteny a převedeny na celé číslo. Efektivnější je za tímto účelem použít registry TEXu. Použití matematického módu je zbytečně nákladné pro sazbu zvýšeného textu.

Optimalizovanou verzi makra v novém souboru `bible-opt.tex` můžete vidět v Ukázce 2. Používá dva registry pro potřebné konstanty a používá navíc makro `\leavevmode`, protože vestavěný příkaz `\raise` nelze použít ve vertikálním módu.

```
\def\Verse{\global\advance\vcount by 1${}^{\the\vcount}$}
```

Ukázka 1: Původní definice makra `\Verse`

```
\newcount\1 \1=1 \newdimen\3 \3=3.6pt
\def\Verse{\advance\vcount\1\leavevmode\raise\3\hbox{\sevenrm\the\vcount}}
```

Ukázka 2: Optimalizovaná definice makra `\Verse`

[17]And Moses and Aaron took these men which are expressed by their names: [18]And they assembled all the congregation together on the first day of the second month, and they declared their pedigrees after their families, by the house of their fathers, according to the number of the names, from twenty years old and upward, by their polls.
[19]As the LORD commanded Moses, so he numbered them in the wilderness of Sinai.
[20]And the children of Reuben, Israel's eldest son, by their generations, after their families, by the house of their fathers, according to the number of the names, by their polls, every male from twenty years old and upward, all that were able to go forth to war; [21]Those that were num-

Obrázek 1: Příklady použití makra `\Verse`

soubor	řádek	procenta	čas [ms]	počet	průměr	soubor
systém	shipout	18,35 %	156,29	1 130	138,31 µs	systém
systém	linebrk	15,64 %	133,23	25 777	5,17 µs	systém
5	29	12,85 %	109,48	60 839	1,80 µs	`bible-opt.tex`
3	666	1,95 %	16,61	55 847	297,00 ns	`plain.tex`
systém	buildpg	1,74 %	14,85	55 190	269,00 ns	systém
5	55	0,78 %	6,67	3 552	1,88 µs	`bible-opt.tex`
5	15	0,63 %	5,39	6 183	871,00 ns	`bible-opt.tex`
3	555	0,49 %	4,18	8 549	489,00 ns	`plain.tex`
3	1204	0,29 %	2,43	3 390	716,00 ns	`plain.tex`
3	1201	0,27 %	2,29	2 260	1,01 µs	`plain.tex`

Tabulka 2: Výstup příkazu `texprofile -T bible-opt`

Deset nejpomalejších řádků po optimalizaci je zobrazeno v Tabulce 2. Řádek 29 v `bible.tex` klesl z druhého místa se 17,63 % na třetí místo s pouhými 12,85 %.[3] Ale to není vše: nově se na čtvrtém místě objevil řádek 666 v `plain.tex` s podílem 1,95 %. Celkově jsme dosáhli zrychlení téměř o 3 %, z 17,63 % na 14,80 %.

Pokud chceme zjistit, co způsobilo zvýšené využití plain TEXu, můžeme se podívat na graf volání, který nám může poskytnout další informace.

3.3 Graf volání

Graf volání nám poskytuje informace na vyšší úrovni abstrakce než pouhé sledování deseti nejpomalejších řádků. Zvažme situaci, kdy by odlišné rozložení kódu rozdělilo makro `\Verse` na řádku 29 v souboru `bible.tex`, které jsme analyzovali, do deseti různých řádků. V takovém případě bychom měli deset záznamů s přibližně 1,8 % každý a žádný z nich by se neprobojoval na začátek seznamu.

Pokud chceme být nezávislí na konkrétním rozložení kódu, můžeme se místo řádků zaměřit na makra. Makra poskytují společné jméno pro posloupnost příkazů a obvykle svým názvem označují úlohu, kterou mají příkazy vykonat. Pro splnění úlohy makro obvykle volá další podmakra, která mohou volat další podmakra. Během řetězce volání maker může makro dokonce volat sebe samo, čímž vytvoří rekurzivní smyčku, při které se stane svým vlastním předkem, jak uvidíme níže.

Když se tedy podíváme na běh TEXového programu z pohledu maker, chceme vědět, kolik času jsme strávili v určitém makru včetně všech podmaker, protože to je celkový čas strávený plněním úlohy z názvu makra. Tomuto času říkáme kumulativní čas makra. Dále nás zajímá, jak se kumulativní čas rozděluje na čas, který používá samotné makro, a na čas, který používají jeho podmakra. Všechny tyto informace zjistíme z grafu volání.

Tabulka 3 ukazuje čtyři makra, která zabírají největší procento celkové doby běhu programu. Vidíme, že na prvním místě je makro `\Verse`; vykonávání tohoto makra zabere 174,51 ms, téměř čtvrtinu celkové doby běhu programu. Avšak během 31 011 volání tohoto makra strávíme přímo v makru `\Verse` pouze 101,50 ms, tedy 58,16 %, zatímco zbývajících 73,01 ms strávíme v makru `\leavevmode`.

[3] Změna počtu použití řádku 29 z 54 649 v souboru `bible.tex` na 60 839 v souboru `bible-opt.tex` má následující vysvětlení: Program `texprofile` pro každý spuštěný příkaz zkontroluje, zda pochází z jiného souboru či řádku než příkaz předchozí. Pokud ano, `texprofile` navýší počet pro nový řádek. Několik příkazů z téhož řádku a souboru se tedy počítá pouze jednou.

Když ale TEX v souboru `bible.tex` vstoupí do matematického módu, přejde také z vertikálního do horizontálního módu, což může vyvolat volání dalších maker jako `\everypar` nebo `\output`. V těchto případech se řádek započítá dvakrát. Pokud byl však TEX již v horizontálním módu při vstupu do matematického módu, řádek se započítá pouze jednou. Také když TEX v souboru `bible-opt.tex` volá příkaz `\leavevmode`, vznikne odbočka z `\Verse` k `\leavevmode` (definovanému v souboru `plain.tex`) a pak zpět do `\Verse`, takže řádek se také započítá dvakrát.

Přesnějšího počtu řádků bylo možné dosáhnout sledováním úrovně zanoření na zásobníku maker, aby se rozlišilo, zda vstup na řádek probíhá v rámci volání makra, mimo volání makra, nebo při návratu z volání, kde se úroveň zanoření zmenšuje. Úroveň zanoření se však může zmenšit i v případě, že po m návratech následuje n volání s $m > n$, což nám úkol ztěžuje.

čas	smyčka	procenta	počet/celkem	podmakro
`\Verse`				
174,51 ms		24,64 %	*	`\Verse`
101,50 ms		58,16 %	31 011	`\Verse`
73,01 ms		41,84 %	31 011/31 011	`\leavevmode`
`\output`				
121,22 ms		17,11 %	*	`\output`
1,15 ms		0,95 %	1 130	`\output`
120,07 ms		99,05 %	1 130/1 130	`\plainoutput`
`\plainoutput`				
120,07 ms		16,95 %	*	`\plainoutput`
106,71 ms		88,87 %	1 130	`\plainoutput`
6,99 ms		5,82 %	1 130/1 130	`\makeheadline`
2,76 ms		2,30 %	1 129/1 130	`\pagebody`
2,75 ms		2,29 %	1 130/1 130	`\makefootline`
859,36 µs		0,72 %	1 130/1 130	`\advancepageno`
`\leavevmode`				
73,01 ms		10,31 %	*	`\leavevmode`
20,09 ms		27,52 %	31 011	`\leavevmode`
52,92 ms		72,48 %	495/1 130	`\output`

Tabulka 3: Výstup příkazu `texprofile -G bible-opt`

Z poslední skupiny záznamů v Tabulce 3 vidíme, jak se čas strávený v makru `\leavevmode` rozdělil: Přibližně jedna čtvrtina byla strávena v makru samotném, zatímco zbývající tři čtvrtiny způsobily 495 volání výstupní rutiny `\output`. Celkový počet volání této rutiny je 1 130, což odpovídá počtu stran dokumentu.

Ze záznamů pro registr `\output` a pro makro `\plainoutput` vidíme, že registr `\output` pouze volá makro `\plainoutput`, které vykonává většinu práce a deleguje pouze malou část úloh na vhodně pojmenovaná podmakra.

3.4 Emulace pdfTEXu

Náš druhý příklad je samotný `texprof` – přesněji řečeno, jeho dokumentace. `texprof` je rozšířením TEXu, a protože je TEX implementován jako „literární“ program, je `texprof` implementován jeho rozšířením. Tento literární program lze zpracovat a získat tak jak soubor `texprof.c`, ze kterého může překladač vytvořit spustitelný program, tak soubor `texprof.tex`, ze kterého může TEX nebo pdfTEX vytvořit krásně vysázený dokument.

Vytvoření souboru PDF pomocí pdfTEXu je výrazně pomalejší (2 327 ms) než vytvoření souboru DVI pomocí TEXu (273 ms). Formát PDF je samozřejmě mnohem složitější formát než DVI, což vysvětluje část rozdílů, ale i při deaktivaci vytváření souboru PDF zůstává při použití pdfTEXu značný rozdíl v čase (1 602 ms). Samotný příkaz `texprof -prof texprof` nestačí k nalezení zdroje zpomalení, protože `texprof` vytváří soubor DVI a běží, s přihlédnutím k režii profilování, přibližně stejnou rychlostí (443 ms) jako TEX. Rozdíl v rychlosti je zjevně způsoben makry, která se používají pouze v pdfTEXu.

```
\def\pdftexversion{140}
\def\pdfoutput{1}
\def\pdfdest#1fith{}
\def\pdfendlink{}
\def\pdfannotlink#1goto num#2\Blue#3\Black\pdfendlink{#3}
\def\pdfoutline goto#1 #2 #3{}
\def\pdfcatalog#1{}
```

Ukázka 3: Soubor `fakepdf.tex`, který implementuje zástupná makra pro vestavěné příkazy pdfTEXu

Musíme tedy přinutit `texprof` předstírat, že je pdfTEX. Toho lze dosáhnout tak, že zpracujeme soubor `fakepdf.tex` z Ukázky 3 před zpracováním souboru `texprof.tex`. Příkaz `texprof -prof -jobname=texprof \input fakepdf.tex \input texprof.tex` zabere očekávaných 1 771 ms.

Deset nejpomalejších řádků v Tabulce 4 pak odhaluje, že téměř polovina doby běhu je způsobena čtyřmi řádky (156–159) v souboru `cwebacromac.tex`. Tyto řádky, které vidíte v Ukázce 4, obsahují makra `\addtoks`, `\poptoks` a `\maketoks`.

soubor	řádek	procenta	čas [ms]	počet	průměr	soubor
9	156	20,29 %	522,50	225 137	2,32 µs	`cweb…ac.tex`*
9	157	12,86 %	331,08	140 088	2,36 µs	`cweb…ac.tex`
9	158	9,13 %	235,03	140 938	1,67 µs	`cweb…ac.tex`
9	159	5,88 %	151,27	115 319	1,31 µs	`cweb…ac.tex`
9	172	3,68 %	94,64	15 759	6,00 µs	`cweb…ac.tex`
9	173	3,18 %	81,95	36 954	2,22 µs	`cweb…ac.tex`
systém	shipout	3,16 %	81,27	775	104,87 µs	systém
systém	linebrk	3,14 %	80,93	27 370	2,96 µs	systém
9	152	2,16 %	55,61	67 026	829,00 ns	`cweb…ac.tex`
9	166	1,86 %	47,95	13 042	3,68 µs	`cweb…ac.tex`

*Celé jméno vstupního souboru je `cwebacromac.tex`.

Tabulka 4: Výstup příkazu `texprofile` -T texprof

Pro informace o účelu těchto řádků můžeme konzultovat graf volání. Tabulka 5 ukazuje tři makra, která zabírají největší procento doby běhu. Podívejme se na tato makra jedno po druhém.

3.5 Rekurzivní makra

Na vrcholu je makro `\pdfnote` následované číslem vstupního souboru 7 a číslem řádku 152 v hranatých závorkách. Tyto dodatečné informace jsou poskytovány při použití volby `-i` programu `texprofile` a jsou nezbytné k rozlišení dvou maker, která mají stejný název, jak uvidíme níže. Makro `\pdfnote` vytváří odkazy na různé části dokumentace, je voláno 8 473krát a každé volání vede k volání podmakra `\maketoks`, které je zodpovědné za téměř veškerý čas makra `\pdfnote`.

Každé volání makra `\maketoks` dále deleguje většinu své práce podmakru `\next`. Pokud se podíváme na informace o souboru a řádku pro makra `\maketoks` a `\next`, zjistíme, že obě makra jsou definována ve stejném souboru a na stejném řádku 159. V Ukázce 4 vidíme na řádku 159 definici makra `\maketoks` a na řádku 164 příkaz `\let`, který přiřazuje makru `\next` stejný význam, jaký má makro `\maketoks`. Volání `\next` v řádku 170 ukončuje definici makra `\maketoks`. Ve skutečnosti tedy makro `\maketoks` volá rekurzivně sebe samo. Rekurzivní makro tohoto typu, kde rekurzivní volání probíhá na samotném konci makra, se nazývá koncová rekurze a je optimalizováno tak, aby se při jeho běhu nezvyšoval zásobník vstupů, jak jsme si již vysvětlili v Sekci 2.4.

Makro `\next` rozděluje práci mezi podmakra `\addtokens`, `\poptoks` a několik volání podmakra `\makenote`. 9 271 volání podmakra `\next` v rámci makra `\maketoks` nakonec končí ve 9 271 voláních podmakra `\next` tak, jak je předefinováno na řádku 174, když je nalezena koncová tečka.

čas	smyčka	procenta	počet/celkem	podmakro
`\pdfnote` [7,152]				
1,30 s		61,17 %	*	`\pdfnote` [7,152]
26,54 ms		2,04 %	8 473	`\pdfnote` [7,152]
1,21 s		93,24 %	8 473/9 271	`\maketoks` [7,159]
46,59 ms		3,58 %	24 230/28 824	`\pdflink` [7,24]
14,67 ms		1,13 %	4 507/4 507	`\[` [5,334]
99,89 µs		0,01 %	80/80	`\ETs` [5,177]
54,34 µs		0,00 %	57/57	`\ET` [5,176]
404,00 ns		0,00 %	1/3	`\glob` [4,166]
`\maketoks` [7,159]				
1,24 s		58,35 %	*	`\maketoks` [7,159]
4,67 ms		0,38 %	9 271	`\maketoks` [7,159]
1,21 s		97,76 %	9 271/130 811	`\next` [7,159]
14,59 ms		1,17 %	9 271/140 082	`\poptoks` [7,157]
8,51 ms		0,69 %	9 271/225 136	`\addtokens` [7,156]
`\next` [7,159]				
1,21 s		57,05 %	*	`\next` [7,159]
53,94 ms		4,44 %	130 811	`\next` [7,159]
501,23 ms		41,29 %	142 326/225 136	`\addtokens` [7,156]
462,00 ms		38,06 %	130 811/140 082	`\poptoks` [7,157]
182,12 ms		15,00 %	28 737/28 737	`\makenote` [7,172]
12,19 ms		1,00 %	9 271/9 271	`\next` [7,174]
2,53 ms		0,21 %	1 456/2 254	`\makenote` [7,154]
0,00 ns	1,19 s	0,00 %	121 540/130 811	`\next` [7,159]

Tabulka 5: Výstup příkazu `texprofile -G -i texprof`

```
156 \def\addtokens#1#2{\edef\addtoks{\noexpand#1={\the#1#2}}\addtoks}
157 \def\poptoks#1#2|ENDTOKS|{\let\first=#1\toksD={#1}%
158   \ifcat\noexpand\first0\countB= #1\else\countB=0\fi\toksA={#2}}
159 \def\maketoks{\expandafter\poptoks\the\toksA|ENDTOKS|%
160   \ifnum\countB> 9 \countB=0 \fi
161   \ifnum\countB< 0
162     \ifnum0=\countC\else\makenote\fi
163     \ifx\first.\let\next=\maketoksdone\else
164        \let\next=\maketoks
165        \addtokens\toksB{\the\toksD}
166        \ifx\first,\addtokens\toksB{\space}\fi
167     \fi
168   \else \addtokens\toksC{\the\toksD}%
        \global\countC=1\let\next=\maketoks
169   \fi
170   \next
171 }
```

Ukázka 4: Řádky 156 až 171 v souboru `cwebacromac.tex`

Protože makro `\next` volá samo sebe, je současně svým vlastním potomkem i předkem. To má důsledky pro přiřazení kumulativních časů, jak je zobrazeno v grafu volání. První řádek v tabulce pro makro `\next` ukazuje celkový čas strávený v tomto makru, který činí 1,21 sekundy. Následující řádky rozdělují tyto 1,21 sekundy, což znamená, že časy uvedené v prvním sloupci by měly dohromady činit 1,21 sekundy a procentuální hodnoty by měly dohromady činit 100 %.

Pokud by program `texprofile` jednoduše určil pro každé podmakro začátek a konec a sečetl časové rozdíly, hodnoty pro makro `\next` jako potomka sebe samého by činily 1,19 sekundy, jak je uvedeno v sloupci „smyčka“. Když se však potomek makra `\next` vrátí, je tento čas již zahrnut na předchozích řádcích.

Proto program `texprofile` udržuje pro každého potomka dva akumulátory pro uplynulý čas: Pro čas zobrazený ve sloupci „smyčka“ `texprofile` sečte časové rozdíly pozorované při návratu z podmakra. Pro čas zobrazený ve sloupci „čas“ `texprofile` navíc odečte všechny časové rozdíly, které již byly přičteny k samotnému makru nebo k některému z ostatních podmaker, protože tyto rozdíly již byly zohledněny.

V jednoduché smyčce, jako je tato, je celý čas v makru `\next` jako potomkovi již zohledněn v makru `\next` jako rodiči. Proto sloupec „čas“ ukazuje 0,00 ns. U složitějších rekurzivních smyček tomu tak nemusí být vždy.

Tato sekce na příkladu souboru `cwebacromac.tex` demonstruje způsob, jakým `texprofile` pracuje s rekurzivními makry, ale nezodpovídá uspokojivě otázku, proč jsou makra `\addtoks`, `\poptoks` a `\maketoks` tak pomalá. Tuto otázku se chystá autor zodpovědět v článku „Efficient input file processing with TEX“, který má vyjít v čísle 46:1 časopisu *TUGboat*. (Pozn. red.)

4 Volby a vestavěné příkazy

Volby `texprofu` jsou stejné jako u ostatních TEXových strojů. Jedinou výjimkou je volba `-prof`, která zapne profilování již od začátku dokumentu. Pokud chceme profilovat jen vybrané části souboru, můžeme použít vestavěné příkazy `\profileon` a `\profileoff`.

Volby `texprofileru` se řídí obecným pravidlem, že volby, které vybírají datové tabulky, používají velká písmena, zatímco volby, které mění způsob zobrazení dat, používají malá písmena. Již jsme viděli volbu `-T` pro tabulku deseti nejpomalejších řádků, volbu `-G` pro graf volání a volbu `-i`, která doplňuje makra o čísla vstupních souborů a řádků.

Můžeme zobrazit kumulativní časy podle vstupních souborů volbou `-F`, podle vstupních řádků s volbou `-L` nebo podle TEXových příkazů s volbou `-C`. Volba `-R` pak zobrazí surové časy pro každý příkaz, který byl profilován. Výsledná tabulka může být velká, ale je užitečná, pokud bylo profilování zapnuto pouze na krátkou dobu nebo pokud chceme tabulku dále zpracovávat.

Pokud je tabulka určena pro další zpracování, volba `-m` upřednostňuje strojovou čitelnost před čitelností pro lidi. Zatímco časy jsou standardně zobrazovány v odpovídajících jednotkách (sekundy: `s`, milisekundy: `ms`, mikrosekundy: `us` a nanosekundy: `ns`), volba `-m` zobrazí všechny časy v nanosekundách bez jednotek.

Volba `-p`n potlačí v tabulkách řádky, které spadají pod n procent. Volba `-t`n upraví volbu `-T` tak, aby zobrazila n nejpomalejších řádků. Volba `-s` upraví volbu `-R` tak, aby do tabulky zahrnula informace o změnách na zásobníku maker.

5 Nedostatky, dočasná řešení a budoucí práce

Aktuální verze `texprofu` a `texprofileru` jsou dostupné na adrese `https://github.com/ruckertm/HINT`. Od první prezentace na konferenci TUG 2024 v Praze jsem provedl u obou programů několik vylepšení. Nejvýznamnější z nich je, že formátové soubory nyní obsahují informace o souborech a číslech řádků pro všechny zdrojové soubory použité při vytváření formátu. Připsání doby běhu neznámému souboru by proto mělo být nyní jen výjimečnou situací.

Metoda uvedená v Sekci 3.4, která umožňuje, aby `texprof` expandoval makra jako pdfTEX, je dočasné řešení. Jak je vidět v Ukázce 3, zástupné definice pro pdfTEXová primitiva odpovídají konkrétnímu použití těchto primitiv v daných souborech. Vestavěné příkazy jako `\pdfannotlink` mají složitou syntaxi, kterou je snadné implementovat jako vestavěné příkazy TEXového stroje, ale velmi obtížně pomocí TEXových maker. Zde je potřeba další práce, ať už ve zjednodušení implementace maker se žádanou syntaxí, nebo v přidání nové volby pro `texprofu`, která by zpřístupnila potřebné zástupné definice jako vestavěné příkazy.

Odkazy

1. RUCKERT, Martin. Profiling TEX Input Files. *TUGboat*. 2024, **45**(2), 203–210. Dostupné z DOI: 10.47397/tb/45-2/tb140ruckert-profiler.
2. ALLRED, Sean. *Various translations of the Bible, typeset in TEX* [online]. [cit. 2024-08-23]. Dostupné z: https://github.com/vermiculus/bible.

Summary: Profiling TEX Input Files

A profiler is a tool used by programmers to analyze the run time behavior of the code they write. The profiler can map the CPU time of a program to specific files and lines, or it can map the time to individual procedures. This information is necessary if a programmer wants to optimize the code for speed.

No such tool was so far available to programmers who write macro packages for TEX. This paper presents `texprof` and `texprofile`, two programs working together as a profiler for TEX input files.

Keywords: TEX, profiling, `texprof`, `texprofile`

Martin Ruckert
Hochschule München
Lothstrasse 64
80336 München
Germany
martin.ruckert@hm.edu

Příprava (LA)TEXových dokumentů v cloudu

Marei Peischl, Marcel Krüger a Oliver Kopp

Používání webových služeb pro kolaborativní přípravu dokumentů v (LA)TEXu je dnes běžné, ale tyto nástroje se zaměřují na psaní dokumentů, nikoliv na tvorbu šablon nebo balíčků. Používání serverů pro průběžnou integraci a doručení je běžné ve vývoji softwaru, ale může být snadno přizpůsobeno pro TEX a přátele.

Tento článek ukazuje, jak začít s používáním automatizace na webových službách jako GitHub, Forgejo nebo GitLab, a to jak pro autory dokumentů, tak pro vývojáře (LA)TEXových balíčků.

Klíčová slova: průběžná integrace a doručení, GitHub, Forgejo, GitLab, docker

1 Úvod

Všechno se dnes přesouvá do cloudu nebo je tam již dostupné. (LA)TEX je v cloudu již přibližně 10 let a dnes je zcela běžné používat pro přípravu dokumentů webový editor (jako Overleaf [2; 3], pozn. red.), zatímco kompilace na vlastním počítači se stala „záležitostí geeků“. Existuje však také třetí varianta pro kompilaci (LA)TEXových dokumentů, kterou můžeme využít také ke zlepšení procesu vývoje balíčků a obecně ke zvýšení stability (LA)TEXu: osvojení metod DevOps, jako je průběžná integrace a doručení (tzv. CI/CD) s použitím automatizovaných postupů.

Pracovně můžeme definovat postup CI/CD (anglicky „workflow“, pozn. překl.) jako soubor kroků, které vedou ke kompilaci (LA)TEXového dokumentu do formátu PDF na nějaké online službě, která má přístup ke zdrojovým souborům.

2 Proč průběžná integrace?

Zaběhnuté pracovní postupy nás často odrazují od změn v našem způsobu práce. Aby tedy mělo smysl číst tento článek, natož pak integrovat tyto mechanismy do projektů, musí pro to existovat dobré důvody.

Raní uživatelé postupů CI/CD v ekosystému TEXu se snažili držet krok se současným stavem ve světě open source softwaru a otevřít dveře přispěvatelům do vývojového procesu. Například projekt LATEX interaguje s uživateli prostřednictvím projektů na webové službě GitHub [4]. Přijímá zde také příspěvky v podobě softwarových záplat, ze kterých může těžit celá (LA)TEXová komunita.

Výhody těchto metod však sahají mnohem dál. Zaměříme se na několik vybraných aspektů, protože vyčerpávající popis by vydal na samostatný článek.

Z anglického originálu [1] přeložil a rozšířil se svolením autorů a vydavatele Vít S. Novotný.

DOI: 10.5300/2024-1-4/59

2.1 U mě to funguje?!

Někdy se mi na mém počítači podaří úspěšně zkompilovat dokument, ale mému vedoucímu na jeho počítači už ne. Existuje mnoho důvodů, proč může kompilace TEXového dokumentu na jednom systému uspět a na jiném selhat. Spuštění externího průběžného integračního procesu nejenže ilustruje veškeré potřebné kroky pro přechod od zdrojových souborů k výstupu ve formátu PDF, ale také pomůže zjistit, zda jsou problémy specifické pro konkrétní počítač, nebo zda jsou obecné.

2.2 Zpětná kompatibilita a regresní testování

S použitím postupů CI/CD můžeme spustit kompilaci na různých distribucích TEXu nebo jejich verzích. Díky tomu můžeme otestovat, zda nějaký balíček způsobuje problémy ještě před tím, než např. nainstalujeme nejnovější verzi MiKTEXu na svém počítači. To nám může ušetřit mnoho času, protože návrat k předchozí verzi je často komplikovaný.[1]

Tuto metodu můžeme také využít ke kontrole zpětné kompatibility, například pokud jeden z přispěvatelů používá stabilní verzi Debianu, která obsahuje pouze zastaralou verzi TEXové distribuce. Jak již bylo řečeno, tým projektu LATEX tyto techniky již mnoho let používá a dokonce poskytuje funkce pro regresní testování v rámci jejich systému l3build [5; 6] pro sestavování (LA)TEXových balíčků.

Vývojáři balíčků mohou postupy CI/CD používat jako obecné rozhraní pro regresní testování. To pomáhá předcházet některým chybám, které by jinak byly zveřejněny a nalezeny až samotnými uživateli. Dále je možné CI/CD postupy použít pro detekci nekonzistencí v kódu. Jeden z autorů např. zjistil, že značné množství balíčků a souborů v distribuci TEX Live neuvádí v kódu číslo verze.

3 Struktura článku

Cílem tohoto článku je seznámit čtenáře se základy nastavení postupů CI/CD na webových službách GitHub, Forgejo a GitLab. Článek je zaměřen především na dvě skupiny uživatelů:

1. autory, kteří se zaměřují na sazbu obsahu, a jejich spolupracovníky a
2. vývojáře balíčků a šablon, jejichž práce tvoří podloží pro práci autorů.

Vývojáři samozřejmě mohou využít postupy určené pro autory, například pro sazbu dokumentace balíčku.

Vzhledem k tomu, že všechny služby pokryté tímto článkem jsou založeny na systému správy verzí git, očekáváme, že projekt bude gitový repozitář. Pro čtenáře, kteří dosud git nepoužívají, odkazujeme doplňující informace z repozitáře tohoto článku [7]. S jejich využitím lze git používat, aniž by si to uživatel vůbec uvědomoval, protože je integrován do automatického ukládání v textovém editoru.

[1] To je další problém… Ale to už je jiný příběh, který by také vystačil na samostatný článek.

Tabulka 1: Ukázkové repozitáře zveřejněné spolu s tímto článkem. Názvy repozitářů mají strukturu ⟨*typ postupu*⟩`_`⟨*úkol*⟩ s přidáním koncovky `_minimal` v případě, že příklad nepoužívá předpřipravený dockerový obraz, ale zahrnuje metody instalace balíčků na základě souboru závislostí, jak popisujeme v sekci 6.

Všechny zde uvedené varianty jsou připraveny pro alespoň tři zde uvedené webové služby: GitHub, GitLab a Forgejo.

Vzhledem k tomu, že adresy repozitářů jsou poměrně dlouhé, zveřejnili jsme je v repozitáři tohoto článku [7] na adrese `https://tug.org/l/peischl-cicd2024`.

Název	Služby	Dokument	l3build	Testy	Docker
`latex`		✔			✔
`latex_minimal`		✔			
`latex_testing`		✔		✔	✔
`latex_testing_minimal`		✔		✔	
`l3build`			✔	✔	✔
`l3build_minimal`			✔	✔	

Protože projekt LaTeX používá službu GitHub, začneme podrobným vysvětlením postupů GitHub Actions a poté vytvoříme odpovídající postupy na dalších službách. Všechny postupy CI/CD jsou k dispozici v ukázkových repozitářích uvedených v Tabulce 1. Všechny ukázky kódu v tomto článku jsou označeny ikonami, aby nedošlo k záměně, protože služby budeme v článku střídat pro ilustraci rozdílů: Ikona označuje, že ukázka pochází z postupu GitHub Actions, zatímco ikona označuje postup GitLab CI.

4 Kompilace dokumentu pomocí průběžné integrace

4.1 První kroky s GitHub Actions

Nejprve založíme repozitář na službě GitHub. Repozitář nemusí být veřejný. Ke kompilaci (La)TeXového dokumentu v „čistém“ prostředí potřebujeme následující:

1. Připravit prostředí a nainstalovat TeXovou distribuci, např. TeX Live.
2. Spustit na dokumentu TeXový stroj s příslušným formátem, např. LaTeXem.

Kromě elementárních příkazů poskytuje služba GitHub předpřipravené „akce“, které provádí více elementárních příkazů v jednom kroku. Pro nás je zajímavá např. akce `latex-action` [8], která zkompiluje LaTeXový dokument pomocí příkazu `latexmk` uvnitř kontejneru s instalací TeX Live. Tato a další akce však obvykle omezují možnosti konfigurace, aby bylo rozhraní co nejjednodušší.

V článku se budeme zabývat dvěma přístupy: Použitím dockerového obrazu s plnou instalací TeX Live v této sekci a s minimální instalací TeX Live v Sekci 6.

Konfigurace postupu CI/CD se provádí vytvořením souboru YAML v adresáři `.github/workflows/`. Název souboru můžeme zvolit libovolný, např. `ahoj-světe.yml`. Následující ukázka kódu zobrazuje minimální konfiguraci:

```yaml
name: Ukázkový postup GitHub Actions
on: [push]
jobs:
  ukázková-úloha:
    runs-on: ubuntu-latest
    steps:
      - run: echo "Ahoj světe! 🦁🦆"
```

Níže uvádíme význam jednotlivých sekcí souboru YAML:

name: Název postupu, který bude zobrazený ve webovém rozhraní GitHubu. Tento název je důležitý, pokud projekt obsahuje více postupů.

on: Podmínky, za kterých bude postup spuštěn. V naší ukázce se úlohy spustí při každém `push`i, tj. kdykoliv aktualizujeme obsah repozitáře. Kromě reakce na změny obsahu je možné spouštět úlohy pravidelně v určitý čas.
Pro všechny možnosti vizte dokumentaci [9]. Výchozí nastavení se liší pro každou platformu (jako Windows, Linux a Mac OS, pozn. překl.) a může být specifické pro konkrétní výpočetní uzel, vizte níže.

jobs: Seznam úloh, které se mají postupně spustit. Je například běžné mít jednu úlohu pro kompilaci dokumentu a druhou pro zveřejnění výsledného souboru PDF. V naší ukázce je jen jedna úloha nazvaná `ukázková-úloha`.

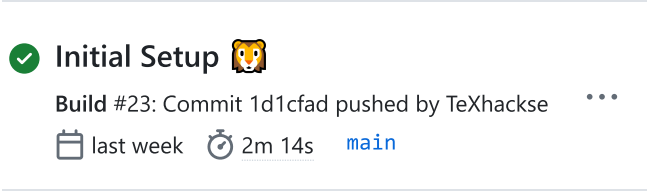
runs-on: Nastavení výpočetního uzlu (anglicky „runner“, pozn. překl.). Výpočetní uzly jsou servery, které vykonávají úlohy definované v sekci `jobs`. Nemusí jít o stejné servery, na kterém jsou hostovány gitové repozitáře. V naší ukázce označuje `ubuntu-latest` vestavěný výpočetní uzel poskytovaný službou GitHub, který používá linuxovou distribuci Ubuntu. Výpočetní uzel obsahuje program NodeJS a některé další nástroje pro usnadnění práce s předdefinovanými akcemi. Úplný seznam vestavěných uzlů najdeme v dokumentaci [10].

steps: Kroky, tj. elementární příkazy a akce, které má postup skutečně provést. V elementárních příkazech je možné přímo zadávat kód pro příkazovou řádku operačního systému a používat Unicode. Náš elementární příkaz vypíše text na standardní výstup a měl by proběhnout bez problémů.

Na GitHubu jsou postupy pro nové repozitáře automaticky povoleny. Při použití vestavěného výpočetního uzlu tak, jak to děláme zde, nám pro spuštění postupu stačí do repozitáře přidat soubor YAML. Následně se postup spustí při všech následujících změnách obsahu repozitáře.

Aktuální stav postupu, tj. který krok právě běží nebo zda již byl dokončen, lze vždy zkontrolovat na ⟨*adrese repozitáře*⟩`/actions`. Například stav postupu

pro první z našich ukázkových repozitářů najdete na adrese `https://github.com/islandoftex/tug2024-workflow-github/actions`. Vypadá nějak takto:



Postup proběhl úspěšně a vypsal text na příkazovou řádku operačního systému. Nyní přejdeme k dalšímu kroku: kompilaci LaTeXového dokumentu.

4.1.1 GitHub Actions s LaTeXem

GitHub Actions jsou postaveny na kontejnerizovaném softwaru, který je spouštěný pomocí dockeru. Naštěstí někteří z nás žijí na Ostrově TeXu (anglicky „Island of TeX“, zkráceně IoT, pozn. překl.) a udržují dockerové obrazy, které zde můžeme využít.[2] Příprava těchto obrazů byla popsána v předchozím článku [12].

V této sekci upravíme ukázkový postup z předchozí strany tak, aby zkompiloval LaTeXový dokument pomocí obrazu od IoT. První část postupu zůstane prozatím stejná, změny nastávají až po direktivě `runs-on`:

```
5     runs-on: ubuntu-latest
6     container:
7       image: texlive/texlive:latest
8     steps:
9       - name: Stáhni repozitář
10        uses: actions/checkout@v4
11      - name: Zkompiluj LaTeXové dokumenty
12        run: "latexmk --lualatex"
```

container: Dockerový obraz, uvnitř kterého se spouští veškeré kroky. V našem příkladu jsme zvolili obraz `texlive/texlive:latest` s plnou instalací TeX Live a některými doplňujícími nástroji [13].

steps: První krok vypadá odlišně od našeho předchozího příkladu. Tento má název „Stáhni repozitář“ a místo elementárního příkazu používá akci `actions/checkout@v4` z tagu `v4` gitového repozitáře na adrese `https://github.com/actions/checkout/`. Tato akce se stará o autentizaci a některé interní záležitosti, takže se nemusíme těmito detaily zabývat. Veškeré další kroky proběhnou už v kořenovém adresáři staženého repozitáře.

[2] Velké díky ostatním obyvatelům Ostrova!

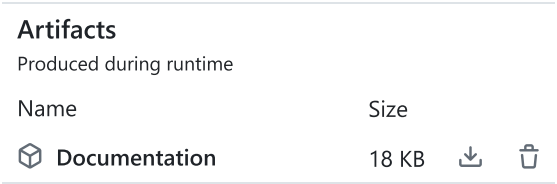
Druhý krok má název „Zkompiluj LaTeXové dokumenty“ a spouští elementární příkaz stejně jako naše předchozí ukázka. Tentokrát spustíme příkaz `latexmk` [15] s volbou `--lualatex`, která, jak možná tušíte, vynutí použití stroje LuaTeX s formátem LaTeX. Ve výchozím nastavení `latexmk` zpracuje všechny soubory s koncovkou `.tex` v kořenovém adresáři repozitáře. Díky tomu v direktivě `run:` nemusíme uvádět jména řídicích souborů našich LaTeXových dokumentů. Postačí, když veškeré soubory s koncovkou `.tex`, které nemají být zkompilovány, umístíme do podadresářů.

4.1.2 Kde jsou soubory PDF?

Pokud byl postup úspěšný, uvidíme sice zelenou ikonu, ale soubory PDF bychom hledali marně. Je to proto, že služba GitHub neví, které výstupní soubory chce uživatel skutečně vidět nebo stáhnout. Tyto kýžené soubory se nazývají „artefakty“. Nyní přidáme do předchozího postupu další krok s akcí `actions/upload-artifact@v4`, která soubory PDF označí jako artefakty:

```
13      - name: Ulož dokumentaci
14        uses: actions/upload-artifact@v4
15        with:
16          name: Documentation
17          path: "./*.pdf"
```

Po dalším úspěšném běhu postupu bude možné přistoupit k souborům PDF, které budou zabaleny do archivu ZIP a dostupné na stránce se stavem běhu. Kliknutím na běh na stránce s přehledem akcí se dostaneme k artefaktu:



Služba GitHub bohužel momentálně nenabízí možnost stáhnout jednotlivé nezabalené soubory a také neumožňuje vytvořit statický odkaz na nejnovější artefakty. (Jiné služby jako GitLab toto umožňují.) K vyřešení tohoto problému a pro snazší přístup k souborům PDF na GitHubu je třeba použít další akce. Nejčastější způsob je publikování PDF souborů do samostatné gitové větve. Tento mechanismus se obvykle používá k vytváření webových stránek a nazývá se „github-pages“.

Pro zápis do repozitáře je nejprve nutné upravit oprávnění v souboru YAML:

```
8     permissions:
9       contents: write
```

Většina akcí, které se používají k nahrávání artefaktů do gitových větví, navíc požaduje adresář namísto jednoho souboru. Proto je nutné všechny soubory přesunout do samostatného adresáře před tím, než tyto akce použijeme:

```
15      - name: Přesuň soubory PDF do adresáře build
16        run: mkdir -p build && mv *.pdf build/
17      - name: Publikuj soubory PDF do gitové větve pdf-output
18        uses: crazy-max/ghaction-github-pages@v4
19        with:
20          build_dir: build
21          target_branch: pdf-output
22        env:
23          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

Token na posledním řádku je nutný k tomu, aby postup mohl měnit repozitář.

4.2 Rozdíly oproti Forgejo Actions

Forgejo [16] je další webová služba pro správu kódu, jejímž cílem je být otevřenější než GitHub. Lze ji provozovat na vlastním serveru a poskytuje mechanismy podobné GitHub Actions. Tyto mechanismy nikdy nebudou fungovat zcela stejně jako na GitHubu, ale jsou až na několik výjimek kompatibilní.

Forgejo nejprve hledá konfigurační soubory v adresáři `.forgejo/workflows/`. Pokud žádné nenajde, zkouší je hledat v adresáři `.github/`. Jediný problém, na který pravděpodobně narazíte při použití stejných postupů jako na GitHubu, je ten, že na většině instancí Forgejo buď nejsou dostupné výpočetní uzly, nebo jsou konfigurované odlišně od těch na GitHubu.

Pokud však dokážete nastavit svůj výpočetní uzel na vlastním serveru se stejnými štítky jako na GitHubu, můžete použít stejné konfigurační soubory pro postupy na obou službách [17]. V ukázkových repozitářích na serveru `codeberg.org`, což je instance platformy Forgejo, najdete také popis, jak jsou zde konfigurovány výpočetní uzly.

4.3 Kompilace dokumentu pomocí GitLab CI

Postupy GitLab CI se příliš neliší od GitHub Actions. Dokonce je zde konfigurace o něco jednodušší, protože některé kroky jako stažení repozitáře se provádí automaticky. Konfigurační soubor musí být umístěn v kořenovém adresáři repozitáře a pojmenován `.gitlab-ci.yml`. Zde je náš ukázkový postup:

```
1 ukázkový-postup-gitlab-ci:
2   image: registry.gitlab.com/islandoftex/images/texlive:latest
3   script:
4     - 'latexmk --lualatex'
```

```
  artifacts:
    paths:
    - "./*.pdf"
```

Veškeré kroky v postupu GitLab CI jsou elementární příkazy, které se vykonají jeden po druhém. Náš ukázkový postup obsahuje jen jeden příkaz pro spuštění příkazu `latexmk`. Na rozdíl od GitHubu poskytuje GitLab rozhraní, které umožňuje vytvořit statické odkazy na artefakty. Proto soubor `README.md` v ukázkových repozitářích na GitLabu obsahuje odkaz v následujícím tvaru:

⟨*adresa repozitáře*⟩`/-/jobs/artifacts/`⟨*gitová větev*⟩`/browse?job=`⟨*úloha*⟩

Takovýto odkaz zobrazí všechny artefakty připojené k dané úloze. Např. následující odkaz zobrazí všechny artefakty z ukázkového repozitáře na GitLabu [18]:

`https://gitlab.com/islandoftex/texmf/tug2024-workflow-document-gitlab/-/jobs/artifacts/main/browse?job=check:+[latest,+lualatex]`

5 Testování s více TEXovými distribucemi a s různými stroji

Jak jsme slíbili v Sekci 2.2, je možné postupy rozšířit pro testování kompilace s různými verzemi TEXových distribucí a s různými TEXovými stroji.

Další TEXové stroje můžeme přidat v samostatných krocích s odlišnými elementárními příkazy. Stejně tak můžeme další stroje použít v samostatných úlohách, nebo dokonce v samostatných konfiguračních souborech. Volba závisí pouze na tom, zda chceme spustit vše zároveň, nebo šetřit prostředky pomocí podmíněného nebo sekvenčního spuštění.

Pro použití různých verzí TEX Live poskytuje IoT obrazy `texlive/texlive` všech historických verzí s tagy `TL`⟨*verze*⟩`-historic` a té aktuální s tagem `latest`:

```
jobs:
  testování-s-IoT-obrazy-TeX-Live:
    runs-on: ubuntu-22.04
    strategy:
      matrix:
        tag: ["TL2022-historic", "TL2023-historic", "latest"]
    name: "Test s TeX Live ${{ matrix.tag }}"
    container:
      tag: texlive/texlive:${{ matrix.tag }}
    steps:
```

matrix: Seznam uživatelských parametrů a jejich hodnot, který úlohu přemění na šablonu. Při spuštění postupu vygeneruje šablona úlohy pro všechny kombinace hodnot z matice kartézského součinu parametrů. V našem ukáz-

kovém postupu máme jen jeden parametr `tag` s různými tagy obrazů `texlive/texlive`. K hodnotě parametru můžeme uvnitř šablony přistupovat pomocí syntaxe `${{ matrix.tag }}`. V našem ukázkovém postupu dojde k vygenerování tří úloh, které budou používat obrazy s TEX Live 2022, 2023 a s poslední dostupnou verzí TEX Live (v době psaní článku TEX Live 2024, pozn. red.). Úplný seznam dostupných obrazů najdeme na adrese `https://gitlab.com/islandoftex/images/texlive`.

Postupy GitLab CI také podporují šablony úloh. Následující ukázkový postup ilustruje spuštění několika TEXových strojů na několika různých verzích TEX Live:

```
1 check:
2   image: registry.gitlab.com/islandoftex/images/texlive:$TAG
[ … přeskočili jsme sekce script a artifacts … ]
6   parallel:
7     matrix:
8       - TAG: ['TL2022-historic', 'TL2023-historic', 'latest']
9         STROJ: ['pdflatex', 'xelatex', 'lualatex']
```

V tomto případě je syntax pro přístup k hodnotám parametrů podobná syntaxi proměnných příkazové řádky unixových operačních systémů, ale funguje stejně jako syntax v ukázkovém postupu pro GitHub Actions.

6 Použití malého dockerového obrazu

V předchozích příkladech jsme pro jednoduchost používali plnou instalaci TEX Live.[3] Obrazy od IoT dokonce obsahují všechny závislosti doplňujících nástrojů jako arara nebo minted, takže všechny tyto nástroje fungují bez další konfigurace.

Ne vždy ale potřebujeme plnou instalaci TEX Live a menší obrazy se stahují rychleji a šetří místo na disku výpočetních uzlů.[4] V takovém případě ale musíme vědět, které balíčky z TEX Live potřebujeme nainstalovat pro kompilaci dokumentů…

A zde přichází na scénu nástroj DEPP pro výpis závislostí v TEX Live [20] (anglicky „DEPendency Printer for TEX Live“, pozn. překl.), hrdě vytvořený na Ostrově TEXu.[5] Protože se tento článek zaměřuje na postupy CI/CD, nebudeme nástroj DEPP podrobněji popisovat. Místo toho ukážeme seznam všech balíčků z TEX Live potřebný pro kompilaci, který DEPP vygeneroval pro náš ukázkový projekt [21]:

```
# Proudly generated by the Island of TeX's DEPendency Printer…
blindtext …
```

[3] Nejde o zcela plnou instalaci: použité obrazy neobsahují dokumentaci ani zdrojové kódy.

[4] Pokud vás zajímá, jak snížit nikoliv velikost dockerového obrazu, ale dobu kompilace velkého množství dokumentů, pak doporučujeme vaší pozornosti loňský článek od IoT [13].

[5] Pokud mluvíte německy, možná vás překvapí, že tento nástroj je chytřejší, než by jeho název napovídal. („Depp“ je jihoněmecký výraz pro hlupáka, pozn. překl.)

6.1 GitHub Actions

Na GitHubu je k dispozici několik akcí pro instalaci TEX Live jako součást postupu [22; 23]. Díky tomu můžeme použít menší obraz, což zkrátí dobu kompilace:

```
9       - name: Instaluj TeX Live
10          uses: zauguin/install-texlive@v3
11          with:
12            package_file: .github/tl_packages
```

Tento úryvek umístíme do sekce `steps` před kompilaci dokumentů. Direktiva `package_file` odkazuje na výstup nástroje DEPP nebo na ručně vytvořený seznam balíčků. Direktiva `container` je nyní nadbytečná a můžeme ji odstranit.

6.2 GitLab CI

Na GitLabu je instalace TEX Live trochu složitější, protože nemáme k dispozici akce, ale jen elementární kroky. Repozitář nástroje DEPP proto poskytuje skript `minimal-portable-tl-setup.sh` [24] pro instalaci podle souboru s balíčky:

```
3   before_script:
4     - minimal-portable-tl-setup.sh tl_packages
```

Tento úryvek umístíme pod direktivu `image`, kterou můžeme následně změnit na menší dockerový obraz (jako např. `debian:latest`, pozn. red.).

Další možností je připravit dockerový obraz, který již potřebné balíčky obsahuje. Pokud používáme vlastní výpočetní uzel, obvykle je to nejlepší řešení, protože můžeme obraz udržovat v mezipaměti a aktualizovat ho podle potřeby.

7 Postupy CI/CD pro vývojáře balíčků

Jak jsme již zmínili, výhody používání postupů CI/CD jsou ještě větší, když je použijeme při vývoji balíčků nebo šablon. V takovém případě očekáváme, že repozitář bude obsahovat balíček a nějakou konfiguraci pro nástroj `l3build`.[6]

7.1 GitHub Actions

Příkaz `latexmk` z předchozích ukázek nahradíme příkazem `l3build`:

```
9       - name: Spusť l3build
10        run: l3build check --show-log-on-error -q -H
```

Tento krok spustí všechny testy balíčku podle konfigurace nástroje `l3build`.

[6] Podobné postupy fungují i pro jiné nástroje, ale to již přesahuje rámec tohoto článku.

Protože se jedná o spuštění testovací sady balíčku, artefakty jsou zcela odlišné. Zatímco při kompilaci dokumentu nás zajímá výstup ve formátu PDF, v tomto případě je důležitý výstup testů. Jeden z autorů tohoto článku proto vytvořil akci, která se o tento výstup postará [25]:

```
      - name: Archivuj výstupy neúspěšných testů
        if: ${{ always() }}
        uses: zauguin/l3build-failure-artifacts@v1
        with:
          name: testfiles
          retention-days: 3
```

Kromě nahrání artefaktů tento úryvek ilustruje další důležitý bod: Na rozdíl od PDF dokumentu, kde nás zajímá pouze úspěšný výstup, u testovacích sad nás zajímá především výstup z neúspěšných testů. Přidali jsme proto podmínku `if: ${{ always() }}`. Ta zajistí, že krok bude spuštěn, i když předchozí krok selže.

7.2 GitLab CI

Jak už jsme zmínili u předchozích ukázek, GitLab nativně podporuje nahrávání artefaktů bez potřeby načítání externích rozšíření:

```
  script:
    - l3build check --show-log-on-error -q -H
  artifacts:
    when: on_failure
    paths:
      - ./build/test/*.diff
```

Tím se sice zjednodušuje základní nastavení, ale zároveň se snižuje flexibilita. Pokud bychom například chtěli použít různé artefakty pro úspěšné a neúspěšné kroky, museli bychom pro to použít dvě samostatné úlohy.

8 Lokální spuštění

Během naší prezentace na konferenci TUG 2024 jsme si v rychlosti ukázali, jak můžeme postupy CI/CD spouštět lokálně. To může být užitečné při ladění, protože máme možnost ručně spouštět jednotlivé kroky, nebo když potřebujeme zajistit, aby lokální prostředí odpovídalo výpočetnímu uzlu.

Pro služby GitHub a Forgejo existuje nástroj `act` [26], který umožňuje spouštět akce z postupů GitHub Actions pomocí dockeru. Služba GitLab pak umožňuje jednoduše nainstalovat výpočetní uzel lokálně. Díky tomu můžeme spouštět postupy GitLab CI tak, že v adresáři s repozitářem spustíme příkaz `gitlab-runner exec`.

9 Toto je průběžný vývoj!

Všechny zde uvedené konfigurace jsou samozřejmě pouze příklady a můžeme je rozšířit podle požadavků projektu. Můžeme spouštět libovolné příkazy, což může být nezbytné zejména u složitějších projektů.

Například při vytváření časopisů je možné průběžně generovat tiskovou a online verzi, výňatky jednotlivých článků i výstupy ve formátu HTML/EPUB, přičemž redaktoři musí počkat pouze na kompilaci článku, na kterém právě pracují.

Totéž platí pro studijní materiály, kde můžeme mít vykreslené verze se zahrnutými řešeními i bez nich, nebo dokumenty, které obsahují vzájemně propojené odkazy, a vyžadují proto několik kompilací.

Integrace postupů CI/CD s pokročilejší prací s gitem, jako je smysluplný koncept větvení, může také zlepšit spolupráci nebo zjednodušit proces přispívání do open source projektů. To sníží pracovní zátěž správců projektů, protože některé problémy nebude třeba kontrolovat ručně.

10 Závěr, výzva k zapojení a žádost o zpětnou vazbu

V článku jsme si ukázali, jak můžeme používat služby GitHub, Forgejo a GitLab pro kompilaci dokumentů a vývoj balíčků v (LA)TEXu. Předvedli jsme si, jak můžeme využít různé verze distribuce TEX Live i různé TEXové stroje pro usnadnění testování napříč platformami. Také jsme se postarali o to, aby bylo možné přistoupit k výsledným souborům PDF nebo k výsledkům testování.

Doufáme, že tato řešení povedou ke zvýšení stability všech částí vývoje v (LA)TEXu tím, že usnadní testování balíčků a ušetří čas při kompilaci složitějších projektů.

Pokud spravujete nějaké balíčky, bylo by skvělé, kdybyste mohli zkusit nastavit testy, které by ověřovaly jejich funkčnost proti nejnovějšímu vydání TEX Live (v době psaní článku TEX Live 2024, pozn. red.), aktuální vývojové verzi TEX Live, nebo dokonce proti testovacím verzím TEX Live před dalším vydáním [27], pro které IoT od roku 2024 připravuje dockerový obraz `texlive/texlive` s tagem `pretest`. Pokud narazíte na problémy, rádi vám pomůžeme.

Rádi bychom tento tutoriál dále udržovali a zjednodušili použití automatizace pro uživatele TEXu a jeho přátel. Pokud objevíte nějaké nejasnosti, budeme rádi, když se nám ozvete, abychom mohli tento tutoriál vylepšit. Tabulka 1 na straně 61 shrnuje všechny související ukázkové repozitáře.

Vaše příspěvky do tohoto projektu jsou vítány!

Odkazy

1. PEISCHL, Marei; KRÜGER, Marcel; KOPP, Oliver. Creation of LATEX documents using a cloud-based pipeline. *TUGboat*. 2024, **45**(2), 227–233.
2. NOVOTNÝ, Vít. Overleaf: Kolaborativní webový editor LATEXu. *Zpravodaj CSTUGu*. 2021, **31**(1–4), 3–8. Dostupné z DOI: 10.5300/2021-1-4/3.

3. STANO, Michal; STARÝ NOVOTNÝ, Vít. Overleaf Community Edition: Postup inštalácie a rozdiely oproti Overleaf Professional [online]. [B.r.] [cit. 2024-09-03]. Dostupné z: http://www.overleaf.com/read/hmxhshbgnbzm.
4. *The LaTeX Project* [online]. [cit. 2024-08-29]. Dostupné z: https://github.com/latex3/.
5. THE LaTeX PROJECT. *l3build: A testing and building system for (La)TeX* [online]. [cit. 2024-08-29]. Dostupné z: https://ctan.org/pkg/l3build.
6. WRIGHT, Joseph. l3build: The beginner's guide. *TUGboat*. 2022, **43**(1), 40–43. Dostupné z DOI: 10.47397/tb/43-1/tb133wright-l3build.
7. PEISCHL, Marei; KRÜGER, Marcel; KOPP, Oliver. *TeXhackse/tugboat--tug2024-cicd* [online]. [cit. 2024-08-30]. Dostupné z: https://tug.org/l/peischl-cicd2024.
8. XU, Cheng. *xu-cheng/latex-action: GitHub Action to compile LaTeX documents* [online]. [cit. 2024-09-03]. Dostupné z: https://github.com/xu-cheng/latex-action/.
9. *Workflow syntax for GitHub Actions* [online]. [cit. 2024-09-03]. Dostupné z: https://docs.github.com/en/actions/using-workflows/workflow-syntax-for-github-actions.
10. *Choosing GitHub-hosted runners* [online]. [cit. 2024-09-03]. Dostupné z: https://docs.github.com/en/actions/using-workflows/workflow-syntax-for-github-actions#choosing-github-hosted-runners.
11. *GitHub Workflow template for LaTeX packages* [online]. [cit. 2024-09-03]. Dostupné z: https://github.com/islandoftex/tug2024-workflow-github.
12. ISLAND OF TEX. Providing Docker images for TeX Live and ConTeXt. *TUGboat*. 2019, **40**(3), 231. Dostupné také z: https://tug.org/TUGboat/tb40-3/tb126island-docker.pdf.
13. ISLAND OF TEX. Living in containers: on TeX Live (and ConTeXt) in a Docker setting. *TUGboat*. 2023, **44**(2), 249–252. Dostupné z DOI: 10.47397/tb/44-2/tb137island-docker.
14. *Checkout action* [online]. [cit. 2024-09-03]. Dostupné z: https://github.com/actions/checkout/.
15. COLLINS, John. *latexmk: Fully automated LaTeX document generation* [online]. [cit. 2024-09-03]. Dostupné z: https://ctan.org/pkg/latexmk.
16. FORGEJO. *Forgejo Repository* [online]. [cit. 2024-07-27]. Dostupné z: https://codeberg.org/forgejo/forgejo.
17. *Using Forgejo Actions (Self-hosted)* [online]. [cit. 2024-09-10]. Dostupné z: https://docs.codeberg.org/ci/actions/.
18. *GitLab Workflow template for LaTeX documents* [online]. [cit. 2024-09-10]. Dostupné z: https://gitlab.com/islandoftex/texmf/tug2024-workflow-document-gitlab.

19. *TEX Live Docker image* [online]. [cit. 2024-07-20]. Dostupné z: https://gitlab.com/islandoftex/images/texlive.
20. *DEPP: Dependency Printer for TEX Live* [online]. [cit. 2024-09-15]. Dostupné z: https://gitlab.com/islandoftex/texmf/depp.
21. KOPP, Oliver. *File tl_packages* [online]. [cit. 2024-09-15]. Dostupné z: https://github.com/islandoftex/tug2024-workflow-document-github/blob/main/.github/tl_packages.
22. KRÜGER, Marcel. *teatimeguest/setup-texlive-action: A GitHub Action to set up TEX Live* [online]. [cit. 2024-09-15]. Dostupné z: https://github.com/zauguin/install-texlive.
23. *teatimeguest/setup-texlive-action repository* [online]. [cit. 2024-09-15]. Dostupné z: https://github.com/teatimeguest/setup-texlive-action.
24. PEISCHL, Marei. *File minimal-portable-tl-setup.sh* [online]. [cit. 2024-09-16]. Dostupné z: https://gitlab.com/islandoftex/texmf/depp/-/blob/main/setup-examples/texlive-minimal-portable/minimal-portable-tl-setup.sh.
25. KRÜGER, Marcel. *zauguin/l3build-failure-artifacts* [online]. [cit. 2024-09-16]. Dostupné z: https://github.com/zauguin/l3build-failure-artifacts.
26. *nektos/act: Run your GitHub Actions locally* [online]. [cit. 2024-09-16]. Dostupné z: https://github.com/nektos/act.
27. *Pretesting TEX Live 2024* [online]. [cit. 2024-09-16]. Dostupné z: https://www.tug.org/texlive/pretest.html.

Creation of LaTeX documents using a cloud-based pipeline

Using web-based platforms for collaborative editing of LaTeX documents is common these days. These tools focus on writing documents and not on creation of templates or packages. Using build servers with automated pipelines is common within software development but can easily be adapted for TeX & friends.

This article will show how to get started using automated workflows on platforms like GitHub, Forgejo, or GitLab for document authors as well as TeX developers.

Keywords: continuous integration & delivery, GitHub, Forgejo, GitLab, docker

Marei Peischl
Gneisenaustr. 18
202 53 Hamburk
Německo
marei@peitex.de
https://peitex.de

Marcel Krüger
Hamburk
Německo

Oliver Kopp
Sindelfingen
Německo

Zpravodaj Československého sdružení uživatelů TEXu
ISSN 1211-6661 (tištěná verze), ISSN 1213-8185 (online verze)

Vydalo:	Československé sdružení uživatelů TEXu vlastním nákladem jako interní publikaci
Obálka:	Antonín Strejc
Ilustrace na obálce:	Donald Knuth a Duane Bibby
Počet výtisků:	240
Uzávěrka:	22. 11. 2024
Odpovědný redaktor:	Jan Šustek
Redakční rada:	Pavel Haluza, Lukáš Novotný, Vít Starý Novotný, Michal Růžička a Jan Šustek (šéfredaktor)
Vědecká rada:	Ján Buša (předseda), Jiří Demel, Jaromír Kuben (zástupce předsedy), Jiří Rybička a Petr Sojka
Technická redakce:	Vít Starý Novotný
Jazyková korektura:	Zbyněk Michálek
Evidenční číslo MK:	E 7629
Adresa:	CSTUG, Nejedlého 373/1, 638 00 Brno
Email:	cstug@cstug.cz

Zřízené poštovní aliasy sdružení CSTUG:

`bulletin@cstug.cz`
korespondence ohledně Zpravodaje sdružení

`board@cstug.cz`
korespondence členům výboru

`cstug@cstug.cz`, `president@cstug.cz`
korespondence předsedovi sdružení

`gacstug@cstug.cz`
grantová agentura CSTUGu

`secretary@cstug.cz`, `orders@cstug.cz`
korespondence administrativní síle sdružení, objednávky CD a DVD

`cstug-members@cstug.cz`
korespondence členům sdružení

`cstug-faq@cstug.cz`
řešené otázky s odpověďmi navrhované k zařazení do dokumentu CSFAQ

`bookorders@cstug.cz`
objednávky tištěné TEXové literatury na dobírku

ftp server sdružení:
`ftp://ftp.cstug.cz`

www server sdružení:
`https://www.cstug.cz`

CONTENTS