Razvoj odprtokodnega JavaScript vtičnika za optimizirano nalaganje in prikaz grafik na spletnih platformah
Development of an open-source JavaScript plugin for optimizing the loading and display of graphics on web platforms

Luka varga[1], Helena Gabrijelčič Tomc[1]

[1] Katedra za informacijsko in grafično tehnologijo, Oddelek za tekstilstvo, grafiko in oblikovanje, Naravoslovnotehniška fakulteta, Univerza v Ljubljani
E-pošta: varga.luka93@gmail.com

Izvleček. *Cilj raziskave je bil načrtovati in razviti zmogljiv odprtokodni JavaScript vtičnik, ki se celovito spopada s problematiko nalaganja in prikaza grafik na spletu, hkrati pa je enostaven za uporabo in analizirati delovanje spletnega mesta pred in po uporabi JavaScript vtičnika. Zasnovan je bil JavaScript vtičnik, ki je bil razdeljen na dva dela - del za uporabo v sami fazi priprave kode in del za uporabo v fazi uporabe spletne platforme. Za evalviranje prihranka časa pri nalaganju spletne strani je bil vtičnik integriran v obstoječo spletno stran z realnimi uporabniki. Iz rezultatov je razvidno, da je bil izdelan JavaScript vtičnik, ki se celovito spopada s problematiko optimizacij nalaganja in prikaza grafik na spletni platformi, učinek vtičnika pa je odvisen od same spletne strani, kjer je vtičnik uporabljen.*

Ključne besede: *odprtokodni JavaScript vtičnik, optimizacija grafik, čas nalaganja, napredni grafični HTML elementi, kompresija*

Abstract. *This paper addresses the conception and implementation of an open-source JavaScript plugin for optimizing the loading and display of graphics on web platforms. The main goals of the paper were to conceptualize and implement a powerful JavaScript plugin that fully addresses the problematics of loading and display of graphics on web while making sure that the plugin is simple for use and to analyse the operation of a website before and after the inclusion of the implemented JavaScript plugin. To evaluate the load time saved by using the plugin, the plugin was integrated in an actual website with real users. Based on the results the implemented plugin does fully address the loading and display of graphics on web platforms, while the effect of the plugin depends greatly on the website itself where the plugin is used.*

Key-words: *an open-source JavaScript plugin, graphics optimization, load time, advanced graphical HTML elements, compression*

1 Uvod

Optimizacija grafik za prikaz na spletu zahteva zmanjševanje števila grafik, katere odjemalec zahteva od strežnika ter zmanjševanje količine podatkov potrebnih za prikaz neke grafike (pri tem pa naj se ohrani čim boljša kvaliteta grafik, ki se prikaže uporabniku). Eden izmed najlažjih načinov za zmanjšanje velikosti datotek je zmanjšanje samih dimenzij grafike glede na potrebe, pri čemer sprva pomislimo na kompresijo z izgubo. Tudi v primeru grafik pa lahko uporabljamo kompresijo brez izgube (tudi v kombinaciji s kompresijo z izgubo) in v primeru grafik je na spletu najbolj razširjen GZIP kompresijski algoritem [1].

Z uporabo kompresije z izgubo lahko sicer zmanjšamo velikost posameznih slikovnih datotek, s čimer pa izpolnimo le del pogojev za optimizirane datoteke - še vedno je namreč samo število slikovnih datotek, katere potrebujemo za prikaz določene spletne strani lahko zelo veliko. Da se izognemo velikemu številu datotek potrebnih za delovanje spletne strani, lahko uporabimo t.i. "CSS Image Sprites" način uporabe slik. V praksi to pomeni, da vse grafike, ki spadajo skupaj in tvorijo neko zaključeno celoto, združimo v eno samo sliko in nato z uporabo CSS-a maskiramo celotno sliko ter prikažemo le želen del slike, ki predstavlja npr. posamezno grafiko.

V CSS3 specifikaciji so specificirani različni grafični filtri, ki so podprti v vsakem modernem brskalniku in delujejo na podlagi modificiranja barvnih matrik elementov, s katerimi lahko nadziramo vrednosti RGB kanalov in alpha kanal nabora HTML elementov in njihovih potomcev. Ti grafični filtri omogočajo tako točkovne operacije, kot tudi lokalne in morfološke operacije [2].

Spletne platforme morajo dandanes biti dostopne na različnih napravah, zelo različnega razpona velikosti zaslonov. Tega se zavedajo tudi razvijalci brskalnikov ter posledično za spletne razvijalce razvijajo napredne grafične HTML elemente za optimizacij grafik. Ti elementi na podlagi zaznavanja specifik uporabnikovega zaslona in na podlagi “pravil” katere

razvijalec specificira za prikaz neke slike izberejo najprimernejšo sliko iz nabora slik, katerega je spletni razvijalec podal. Med takšne naprednejše grafične HTML elemente lahko umestimo element HTMLPictureElement [3], ki mora med potomci imeti en HTMLImageElement, ima pa lahko tudi več elementov tipa HTMLSourceElement [4, 5].
Namen raziskave je bila zasnova in izdelava enostavnega in zmogljivega odprtokodnega JS vtičnika za celovito optimizirano nalaganje in prikaz grafik na spletnih platformah z uporabo katerega lahko občutno skrajšamo čas nalaganja spletnih platform in olajšamo delo razvijalcem spletnih platform. V okviru raziskave je bilo tudi raziskano kako lahko izrabimo osnovne principe delovanja spletnih mest za boljšo optimizacijo nalaganja grafik, katere spletne tehnologije povezane z optimiziranim nalaganjem in prikazom grafik so v zadostni meri podprte za uporabo in kako zagotoviti kompatibilnost v primeru, da neka uporabljena tehnologija ni na voljo pri nekem uporabniku.

2 Eksperimentalni del

V eksperimentu so bili uporabljeni naslednji programi in orodja: program *JetBrains WebStorm 2018* za razvoj JS vtičnika [6], programsko orodje *Git* za nadzor verzij JS vtičnika [7], spletni repozitorij *Github* za gostovanje izvorne kode JS vtičnika [6,8], *NPM* repozitorij JS vtičnikov za enostavno integracijo JS vtičnika v projekte ostalih razvijalcev [9], *Travis CI* za neprekinjeno integracijo (ang. continuous integration) med Github repozitorijem in NPM repozitorijem, za izvajanje avtomatskih testov, kompilacije izvorne kode v produkcijsko kodo [10], orodje *Codecov* za nadzor pokritosti izvorne kode z avtomatskimi testi [11], odprtokodno orodje *semantic-release* za standardizirana sporočila ob nalaganju sprememb izvorne kode v Github repozitorij, za avtomatsko generiranje sporočil o spremembah ob izdajah novih verzij vtičnika, za avtomatsko semantično verzioniranje izvorne kode vtičnika na podlagi standardiziranih sporočil ob nalaganju sprememb izvorne kode [12], odprtokodni jezik TypeScript za izdelavo JS vtičnika, ki se v procesu kompilacije izvorne kode avtomatsko prepiše v JS [13], odprtokodno orodje *Webpack* za proces kompilacije izvorne kode [14], odprtokodno orodje *Jest* in njegov derivat *ts-jest* za izdelavo in izvajanje avtomatskih testov [14–16], odprtokodno orodje *Jimp* za generacijo nizko-kvalitetnih variacij grafik [17], odprtokodno orodje *promptly* za postavljanje vprašanj uporabniku in pridobivanje uporabnikovih odgovorov preko ukazne vrstice [18], odprtokodno orodje *rimraf* za omogočanje programskega brisanja datotek na različnih operacijskih sistemih z istim ukazom [19], odprtokodno orodje *tslint* za dodaten nadzor pisanja izvorne kode in zagotovitev konsistentnosti sintakse kode [20] ter spletni brskalnik *Google Chrome* za analiziranje vpliva vtičnika na spletno platformo.

2.1 Faza priprave kode

Za fazo priprave kode smo izdelali dva algoritma - algoritem, ki analizira grafike in algoritem, ki pripravi nizko-kvalitetne različice grafik, ki se lahko uporabijo za progresivno nalaganje grafik.
Algoritem, ki analizira grafike najprej pogleda stopnjo kompresije grafike. Pri temu upošteva dimenzije grafike in velikost grafike v bajtih. Če zazna, da ima grafike nizko ali srednjo stopnjo kompresije opozori uporabnika, da lahko grafiko dodatno kompresira. V primeru, da je kompresija grafike ustrezna pa algoritem nadaljuje in podrobno pregleda barvno vsebino grafike.
Pri pregledu barvne vsebine grafike algoritem najprej poviša kontrast grafike, da se na ta način znebimo dodatnih barvnih odtenkov, ki so nastali kot posledica glajenja robov znotraj grafike. Nato algoritem skenira barvne vrednosti vsake slikovne pike, njegove RGBA vrednosti uporabi za pretvorbo v Lab barvni prostor (preko pretvorbe iz RGBA v RGB, pri čemer za barvo ozadja uporabi referenčno belo točko z RGB vrednostmi 255, 255, 255, ter preko pretvorbe iz RGB v XYZ barvni prostor, pri čemer za referenčno osvetlitev uporabi D65/2° in na koncu še iz XYZ v Lab barvni prostor, ponovno ob referenčni osvetlitvi D65/2°) in primerja barvno razliko izračunano po funkciji dE76 z barvnimi vrednostmi že-skeniranih slikovnih pik. Če zazna majhno število unikatnih barvnih odtenkov (pri čemer za unikatne barvne odtenke smatra barvne odtenke, pri katerih je $\Delta E < 3$), uporabnika algoritem opozori, da je grafika primerna za konverzijo v SVG format.
Algoritem, ki pripravi nizko-kvalitetne različice grafik, za vse grafike znotraj direktorija, katerega poda uporabnik, ki še nimajo pripravljenih nizko-kvalitetnih različic, grafike pomanjša na 4% prvotne površine, s čimer se močno zniža tudi število bajtov potrebnih za nalaganje grafike. S pomočjo lokalne operacije glajenja pa poskrbi, da bo grafika v brskalniku še vedno dovolj kvalitetna.
Primer generirane nizko-kvalitetne različice lahko vidimo na sliki 1 (desno). Slika je bila generirana z algoritmom za pripravo nizko-kvalitetne različice primera slike, originala prikazane na sliki 1 (levo), in je, kot vidimo, kar 302x manjša od originala.

Slika 1: Velikost JPEG slike dimenzije 4925 x 3283 px znaša 10,3 MB (original, levo), velikost JPEG slike dimenzije 985 x 657 px znaša 34 kB (desno).

2.2 Faza uporabe spletne platforme

Za fazo uporabe spletne platforme smo izdelali tri algoritme - algoritem za preverjanje ustreznosti grafik v

sami fazi uporabe, ki je namenjen za uporabo predvsem v sklopu razvoja in ne na sami produkcijski verziji spletnega portala, algoritem za zakasnjeno nalaganje in algoritem za progresivno nalaganje grafik.
Algoritem za preverjanje ustreznosti grafik preveri, 1. če je grafika prikazana s pomočjo naprednih grafičnih HTML elementov, 2. če imajo napredni grafični elementi ustrezne atribute (atributa “srcset” in “sizes”) in 3. če imajo atributi pravilne vrednosti.
Algoritem za zakasnjeno nalaganje poskrbi za to, da se grafike na spletni platformi ne naložijo vse naenkrat, temveč, da se naložijo le tiste grafike, ki so potrebne. Ko algoritem naloži neko grafiko pa takoj začne izvajati tudi algoritem za progresivno nalaganje grafike. Algoritem za progresivno nalaganje grafik preveri vse grafike, ki uporabljajo nizko-kvalitetne različice grafik generirane z algoritmom za pripravo nizko-kvalitetnih različic grafike v fazi priprave kode, in tiste grafike, katerim je razvijalec preko specifičnih atributov dodelil morebitne druge nizko-kvalitetne različice. Algoritem nato vzame vse te grafike, katere še nimajo naložene visoko-kvalitetne različice, in v ozadju začne nalagati njihovo visoko-kvalitetno različico. Algoritem z uporabo CSS grafičnih filtrov in CSS animacij izvede eleganten, zvezen prehod ob zamenjavi nizko-kvalitetne različice grafike v visoko-kvalitetno različico grafike in na tak način zagotovi, da sam proces progresivnega nalaganja grafike ni moteč za končnega uporabnika.

3 Rezultati in razprava

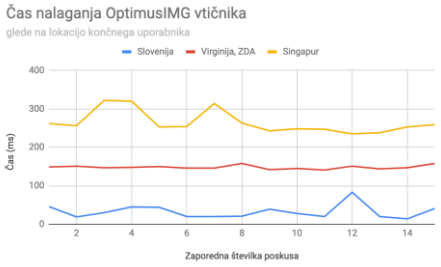
Rezultat raziskave je odprtokodni JS vtičnik OptimusIMG (sestavljenka iz “Optimus”, latinske besede, ki pomeni “najboljši” in iz “IMG”, ki je kapitelna različica HTML značke za HTMLImageElement - <img>), ki je v svoji produkcijski, pomanjšani različici z uporabo GZIP algoritma, velik le 3,6 KB. V času pisanja tega prispevka je bil prenesen že več kot 2000-krat. Vtičnik je ponujen na razpolago na NPM repozitoriju [21].

3.1 Trajanje nalaganja kode vtičnika

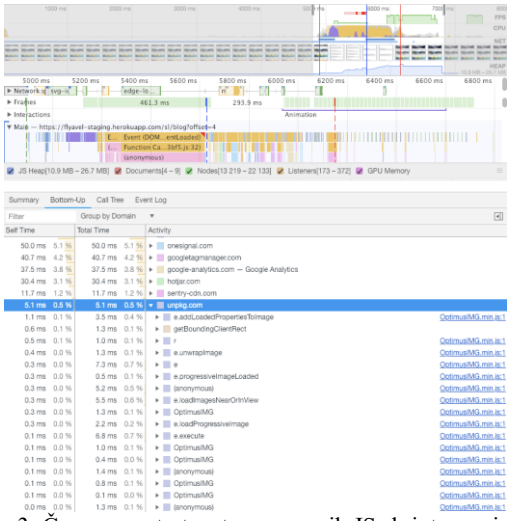
V primeru dodajanja OptimusIMG vtičnika na spletno platformo z uporabo NPM repozitorija se bo koda vtičnika tipično združila v eno samo datoteko skupaj z ostalo JS kodo, ki jo neka spletna platforma uporablja. Druga izmed glavnih predvidenih možnosti dodajanja OptimusIMG vtičnika na spletno platfromo je vključevanje reference z URLjem na katerem je produkcijska verzija OptimusIMG javno objavljena. V primeru dodajanja vtičnika na spletno platformo preko tega načina, pa je delta časa potrebnega za nalaganje vtičnika različna glede na lokacijo končnega uporabnika in zasedenost strežnika na katerem je OptimusIMG na voljo in bo, kot lahko vidimo na sliki 2, precej bolj variirala, kot v primeru združevanja kode iz OptimusIMG vtičnika skupaj z ostalo JS kodo, ki jo neka spletna platforma uporablja.

3.2 Trajanje izvajanja algoritmov

Kot lahko vidimo na sliki 3, ki prikazuje časovno potratnost raznoraznih JS skript, grupiranih po domeni, so bili algoritmi OptimusIMG vtičnika izvedeni (v temu poskusu) v roku 5,1 ms, kar je znašalo na primeru spletne strani (ki je dokaj povprečna kar se tiče zahtevnosti in trajanja izvajanja JS funkcij ob nalaganju spletne strani) 0,5% vsega časa porabljenega za izvajanje JS kode.

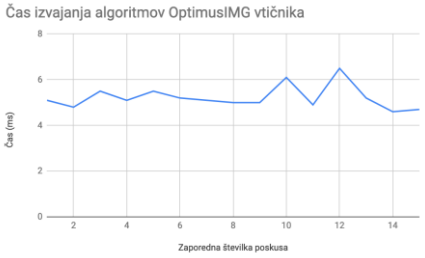


Slika 2: Čas nalaganja OptimusIMG vtičnika glede na lokacijo končnega uporabnika v primeru dodajanja vtičnika preko reference na URL



Slika 3: Časovno potratnost raznoraznih JS skript, grupiranih po domeni.

Naslednja pomembna točka pri evalviranju koristnosti vtičnika je trajanje izvajanja algoritmov v vtičniku. Povprečen čas izvajanja algoritmov OptimusIMG vtičnika pa je bil, po naših testiranjih 5,2 ms, kot je jasno tudi iz podatkov na sliki 4.

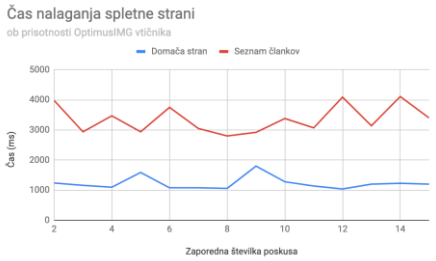


Slika 4: Čas izvajanja algoritmov OptimusIMG vtičnika

3.3 Čas nalaganja spletne strani

Čas nalaganja spletne strani smo merili na dveh primerih in sicer na domači strani spletne platforme in na strani s seznamom člankov spletne platforme. Izmerili smo čas nalaganja spletne strani v primeru, da je OptimusIMG vtičnik vključen in dodan na spletno stran preko URL reference (slika 5) ter izmerjene vrednosti časa nalaganja primerjali z referenčnimi

vrednostmi - t.j. ista spletna stran, na kateri ni dodan OptimusIMG vtičnik (slika 6). Pri testiranjih smo za namen konsistentnosti znova omejili hitrost internetne povezave na povprečno hitrost internetne povezave na mobilnih omrežjih v letu 2018 in simulirali uporabo mobilne naprave.
Kar se tiče grafik domača stran vključuje 5 grafik prikazanih na način vrtiljaka, stran s seznamom člankov pa 50 grafik - po ena grafika za vsak članek. Kot lahko razberemo iz podatkov na sliki 5 je povprečni čas nalaganja domače strani spletne platforme ob prisotnosti OptimusIMG vtičnika 1215 ms, povprečni čas nalaganja strani s seznamom člankov spletne platforme ob prisotnosti OptimusIMG vtičnika pa 3385 ms. Lahko rečemo, da je čas nalaganja strani s seznamom člankov višji od domače strani predvsem na račun izvajanja kode na strežniku, ki je očitno bolj potratna. Na sliki 6 vidimo, da se je povprečen čas nalaganja domače strani brez uporabe OptimusIMG vtičnika dvignil iz 1215 ms na 2087 ms, strani s seznamom člankov pa iz 3385 ms na 9759 ms.



Slika 5: Čas nalaganja spletne strani ob prisotnosti OptimusIMG vtičnika

Slika 6: Čas nalaganja spletne strani brez prisotnosti OptimusIMG vtičnika

4 Zaključki

Glede na dobljene rezultate in evalvacijo uporabnosti izdelanega odprtokodnega JS vtičnika podajamo naslednje zaključke. Uporaba izdelanega vtičnika lahko močno skrajša čas nalaganja spletnih platform. Izdelan vtičnik je možno uporabiti v vseh projektih, ne glede na tehnološki profil samega projekta. Uporaba algoritmov je zelo hitra in enostavna in zahteva minimalen napor s strani uporabnikov - v osnovi to pomeni en sam klic funkcije ki sproži zahtevan algoritem, in bodisi dodajanje potrebnih atributov na slikovne HTML elemente, bodisi podajanje opcijske konfiguracije ob klicu funkcije. Z gotovostjo lahko trdimo, da se tako absolutni kot tudi relativni prihranki ob uporabi našega vtičnika večajo z naraščajočim številom grafik uporabljenih na spletni strani.

Na podlagi rezultatov je uporaba JS vtičnika predvsem priporočljiva za projekte, ki vključujejo veliko število grafik. V vsakemu primeru pa tudi na projektih z majhnim številom grafik ne škodi, če je vtičnik dodan na spletno platformo - še posebej, če je združen v isto datoteko, kot preostala JS koda spletne platforme, kar je tudi priporočen način dodajanja kateregakoli JS vtičnika na spletno stran.

Literatura

[1] W.B. Pennebaker, J.L. Mitchell: JPEG: Still Image Data Compression Standard. Springer Science & Business Media; 1992. 638 str.

[2] C. Coyier: filter | CSS-Tricks [Internet]. CSS-Tricks, https://css-tricks.com/almanac/properties/f/filter/

[3] HTMLPictureElement, MDN Web Docs, https://developer.mozilla.org/en-US/docs/Web/API/HTMLPictureElement

[4] The Image Embed element. MDN Web Docs, https://developer.mozilla.org/en-US/docs/Web/HTML/Element/img

[5] HTML picture tag, https://www.w3schools.com/tags/tag_picture.asp

[6] WebStorm: The Smartest JavaScript IDE by JetBrains, JetBrains, https://www.jetbrains.com/webstorm/

[7] Git, https://git-scm.com/

[8] lukaVarga. lukaVarga/OptimusIMG, https://github.com/lukaVarga/OptimusIMG

[9] npm: optimusimg, https://www.npmjs.com/package/optimusimg

[10] Travis CI - Test and Deploy Your Code with Confidence, https://travis-ci.org/lukaVarga/OptimusIMG

[11] @codecov. Code coverage done right, Codecov, https://codecov.io

[12] semantic-release. semantic-release/semantic-release, GitHub, https://github.com/semantic-%20release/semantic-release

[13] TypeScript - JavaScript that scales, https://www.typescriptlang.org/

[14] webpack, https://webpack.js.org/

[15] Jest, Delightful JavaScript Testing, https://jestjs.io/index.html

[16] kulshekhar. kulshekhar/ts-jest, https://github.com/kulshekhar/ts-jest

[17] oliver-moran. oliver-moran/jimp, https://github.com/oliver-moran/jimp

[18] moxystudio. moxystudio/node-promptly, GitHub, https://github.com/moxystudio/node-promptly

[19] isaacs. isaacs/rimraf, GitHub, https://github.com/isaacs/rimraf

[20] TSLint, https://palantir.github.io/tslint/

[21] npm, https://www.npmjs.com/