

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Samo Pajk

Spletni urejevalnik slik

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: viš pred. dr. Alenka Kavčič

Ljubljana, 2014

Rezultati diplomskega dela so intelektualna lastnina avtorja. Za objavljanje ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje avtorja, Fakultete za računalništvo in informatiko ter mentorja.

Besedilo je oblikovano z urejevalnikom besedil \LaTeX .

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:
Spletni urejevalnik slik

Tematika naloge:

V okviru diplomskega dela izdelajte spletno aplikacijo, ki omogoča urejanje rastrskih slik. Aplikacija naj omogoča odpiranje slike in njen prikaz, urejanje slike ter njeno shranjevanje na uporabnikov računalnik. Aplikacija naj nudi tudi različna standardna orodja za obdelavo in izdelavo slik, kot so čopič, radirka, kapalka, premikanje plasti in podobna. Pri razvoju aplikacije se osredotočite na tehnologijo WebGL, ki preko strojne podpore omogoča izdelavo hitrejših in bolj odzivnih aplikacij.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Samo Pajk, z vpisno številko **63090333**, sem avtor diplomskega dela z naslovom:

Spletni urejevalnik slik

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom viš. pred. dr. Alenke Kavčič,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 7. oktober 2014

Podpis avtorja:

Kazalo

Povzetek

Abstract

1	Uvod	1
2	O aplikaciji	5
2.1	Cilji	6
3	Uporabljene tehnologije	7
3.1	Ruby on Rails	7
3.2	Middleman	7
3.3	HTML	7
3.3.1	Element HTML Canvas	8
3.4	CSS	9
3.5	SASS	9
3.6	JavaScript	9
3.7	WebGL	10
3.7.1	Modeli in teksture	11
3.7.2	Senčniki	12
3.7.3	Mešanje barv	13
3.7.4	Globinski pomnilnik	14
3.7.5	Izris v teksture	14
3.8	GIT	14
4	Teoretične osnove	15
4.1	Prikaz in hranjenje podatkov	15
4.2	Operacije	16

4.3	Sestava urejevalnikov slik	17
4.4	Potrebne funkcionalnosti WebGL	18
5	Urejevalnik slik	19
5.1	Knjižnica za delo z WebGL	19
5.2	Arhitektura aplikacije	21
5.2.1	Dokument	22
5.2.2	Izrisovalnik	24
5.2.3	Navigatorski in upravljalski navigatorja	25
5.2.4	Obdelovalec vhodnih naprav	25
5.2.5	Upravljalski bližnjic	26
5.2.6	Upravljalski barv	26
5.2.7	Upravljalski sredstev	26
5.2.8	Odpiranje in shranjevanje slik	27
5.3	Orodja	27
5.3.1	Orodje za premikanje plasti	27
5.3.2	Kapalka	28
5.3.3	Orodje za obrezovanje	28
5.3.4	Čopič	28
5.3.5	Radirka	32
5.3.6	Učinki	32
5.4	Uporabniški vmesnik	36
5.4.1	Panel barve	37
5.4.2	Panel plasti	37
5.4.3	Panel zgodovine	38
5.4.4	Panel informacij	39
5.4.5	Panel čopičev in radirke	39
5.4.6	Panel orodja za obrezovanje	40
5.4.7	Panel učinkov	40
6	Sklepne ugotovitve	41

Seznam uporabljenih kratic

RGB (angl. **Red Green Blue**) je barvni model, ki se uporablja za hranjene barv v računalnikih in za prikaz barv na zaslonu.

RGBA (angl. **Red Green Blue Alpha**) je barvni model za hranjenje prosojnih barv.

HSL (angl. **Hue Saturation Lightness**) je barvni format, ki vsebuje podatke o barvnem odtenku, nasičenosti in svetlosti.

HTML (angl. **Hyper Text Markup Language**) je označevalni jezik za izdelavo strukture spletne strani.

CSS (angl. **Cascading Style Sheets**) je označevalni jezik za opis stila elementov HTML.

SASS (angl. **Syntactically Awesome Style Sheets**) je nadgradnja jezika CSS.

WebGL (angl. **Web Graphics Library**) je grafični programski vmesnik, ki se uporablja na spletu.

API (angl. **Application Programming Interface**) je skrajšava za programski vmesnik.

MVC (angl. **Model View Controller**) je arhitekturni vzorec, ki se uporablja pri izdelavi uporabniških vmesnikov.

DRY (angl. **Do not Repeat Yourself**) je programersko načelo, pri katerem se skušamo izogniti ponavljanju kode.

ERB (angl. **Embedded Ruby**) je jezik za izdelavo predlog HTML s pomočjo jezika Ruby.

KAZALO

GPU (angl. Graphical Processing Unit) je grafični procesor računalnika (grafična kartica).

CPU (angl. Central Processing Unit) je splošno namenski procesor, prisoten v vsakem računalniku.

GLSL (angl. OpenGL Shading Language) je jezik, s katerim opisujemo programe (senčnike), ki se izvajajo na grafični kartici.

FIFO (angl. First in First Out) je metoda za organiziranje podatkov v vrsti - prvi v vrsti je prvi na vrsti.

Povzetek

S slikami beležimo spomine, trenutke, dogodke in delimo stvari, ki nas zanimajo. V današnjem času se velikokrat zgodi, da je potrebno kakšno sliko tudi obdelati, urediti ali pa dodati kakšen umetniški učinek, a pri roki nikoli nimamo urejevalnika, ki bi nam to omogočal. Zato je bil izdelan spletni urejevalnik slik, ki uporabnikom nudi hitro in učinkovito urejanje slik s katerega koli računalnika. Naloga opisuje izdelavo urejevalnika slik na spletu in njegovo implementacijo s tehnologijo WebGL za hitrejšo ter boljše delovanje aplikacije. V ta namen je izdelana knjižnica za lažje delo s programskim vmesnikom WebGL, ki je kasneje uporabljena pri izdelavi aplikacije. Opisana je arhitektura aplikacije in glavne podatkovne strukture, ki jih obdelujejo orodja, s pomočjo katerih lahko spreminjamo, dodajamo in urejamo slikovne elemente. Implementiranih je nekaj preprostih orodij, ki se pogosto uporabljajo v tovrstnih aplikacijah ter končnemu uporabniku omogočajo spreminjanje in urejanje slik.

Ključne besede: WebGL, digitalno procesiranje slik, risanje, urejanje fotografij, spreminjanje fotografij, Bézierova interpolacija, interaktivne spletne aplikacije

Abstract

Images are made every day to keep memories, moments and share things we like. Individuals need to edit, change or add an artistic touch to a photo many times, but they can't seem to find an image editing application in order to do it. A web based image editing application was created for that purpose so that we can use it on any computer in order to quickly and effectively edit photos. The thesis describes the implementation of a web based image editing application with technology WebGL, which was used for better and faster image processing. The main topic of the paper describes the creation of a simple library for easier interaction with WebGL, which serves as a base for implementing the main part of the application. After basics are covered, the application architecture and data structures are described and the tools that manipulate those data structures are presented. With the help of implemented tools the user can change and edit images inside the browser.

Keywords: WebGL, digital image processing, drawing, proto editing, photo manipulation, Bézier interpolation, interactive web applications

Poglavje 1

Uvod

Vse od začetka spleta pa do danes se je na internetu pojavilo mnogo novih rešitev, ki tako ali drugače koristijo končnim uporabnikom. Preko spletnih strani in aplikacij si lahko kupimo stvari, poiščemo pot, ogledamo kraj ali pa si pomagamo pri reševanju problemov in iskanju informacij, preprosto iz kateregakoli računalnika, ki je povezan v internet. Vse skupaj se je seveda začelo s spletnimi mesti, kot so IMDb [5], ki so uporabnikom omogočala enostaven pregled podatkov o filmih in iskanje med njimi. Kasneje so se pojavile spletne strani, preko katerih so lahko uporabniki kupovali stvari, kot sta Amazon [6] in Ebay [7], pojavili so se iskalniki, kot je Google [8], ter PayPal [9] za plačevanje preko spleta.

Zaradi vse večje dostopnosti in potrebe po podatkih v internetu zadnjih nekaj let narašča število storitev v oblakih, pri katerih ponudniki nudijo uporabnikom storitve, ki deluje preko spleta ter so dostopne s katerega koli računalnika, ki je povezan v internet. Aplikacija je tako uporabniku vedno na voljo, najpomembnejše pa je to, da lahko uporabnik vedno dostopa do svojih podatkov ne glede na to, s katerega računalnika uporablja aplikacijo. Na internetu pretežno uporabljamo spletne aplikacije za iskanje podatkov in izpolnjevanje obrazcev. Zadnjih nekaj let pa se pojavljajo spletne aplikacije, ki uporabnikom ponujajo možnost urejanja besedil, dokumentov, razpredelnic, grafov, izdelavo predstavitev, slik, vizualizacij, 3D-modelov, pa tudi interaktivnih vsebin in iger. Najbolj poznane aplikacije na tem področju so GoogleDrive [10], Dropbox [14], Pixlr [34], Asana [15], Prezi [16] ...

Splet je obsežen in tako je tudi število aplikacij in njihova raznolikost ter tehnologije, ki jih uporabljajo. V okviru diplomskega dela se bomo osredotočili na razvoj in implementacijo vidnega in interaktivnega dela spletnega urejevalnika slik s tehnologijam HTML5, JavaScript, WebGL ter CSS. Na to temo je bilo izdelanih že kar nekaj rešitev,

vendar je večina tovrstnih aplikacij realiziranih s programskim vmesnikom Canvas 2D ali pa s tehnologijo Adobe Flash, ki imajo ob zahtevnih operacijah manjšo zmogljivost. Zato smo se odločili, da aplikacijo realiziramo s tehnologijo WebGL, ki se že več let uporablja na mobilnih napravah, za strojno pospeševanje grafično intenzivnih operacij.

OpenGL ES 2, na katerem temelji WebGL, se že vrsto let uporablja na mobilnih napravah, kot so pametni telefoni in tablice, za strojno pospeševanje uporabniških vmesnikov, aplikacij za obdelavo slik, videov in drugih interaktivnih vsebin, kar pomeni, da je tehnologija testirana ter učinkovita. WebGL se je pojavil leta 2011 in ponuja skoraj identičen programski vmesnik, kot je OpenGL ES 2 za mobilne naprave. Z WebGL je nastalo mnogo manjših eksperimentov, ki so večinoma nastali za demonstracijo in testiranje samega programskega vmesnika ter njegovih zmožnosti, kasneje pa se je tehnologija pričela uporabljati tudi v produkcijskih aplikacijah, kot so Google Maps [11], Clara.io [12], SketchFab [13] itd. Ker je tehnologija še neizkoriščena in ker še ne obstaja urejevalnik slik, realiziran s tehnologijo WebGL, smo se odločili za realizacijo aplikacije z vmesnikom WebGL, predvsem tudi zato, da testiramo njegovo hitrost, uporabnost, stabilnost in da se v procesu naučimo nekaj novega.

Namizne aplikacije za urejanje slik so navadno implementirane v nizkonivojskih jezikih, kot so C ali C++, ki ponujajo boljši nadzor nad hitrostjo, vendar zahtevajo večjo pozornost programerja, traja več časa, da se prevedejo, in so težje za razvijanje zaradi kompleksnejših programskih vmesnikov. WebGL, pa nam za razliko od nizkonivojske implementacije ponuja hitrejše prototipiranje, samodejno upravljanje s pomnilnikom, enostavno distribucijo med platformami in hitro dostopnost.

V okviru razvoja diplomskega dela bomo torej skušali predstaviti probleme, s katerimi se srečamo pri implementaciji aplikacije za obdelavo slik in možnostmi za njihovo rešitev ter primerjavo s klasičnim načinom. Poudarek bo torej na razvoju same aplikacije z omenjenimi tehnologijami in ne na strežniškem delu, predvsem zato, ker aplikacija lahko deluje brez strežnika, saj lahko uporabniku omogočimo uvoz slik iz trenutnega računalnika in shranjevanje nanj ali pa na že obstoječe storitve, kot so GoogleDrive, Dropbox, Facebook, Instagram itd.

V naslednjem poglavju bomo okvirno predstavili aplikacijo in cilje izdelane aplikacije. V tretjem poglavju bodo opisane spletne tehnologije in ogrodja, ki smo jih uporabili pri izdelavi diplomskega dela. Četrto poglavje bo predstavilo teoretične osnove diplomskega dela, kot so hranjenje slik na računalniku, operacije, ki jih izvajamo nad slikami ter funkcionalnosti tipičnih obstoječih urejevalnikov slik. V petem poglavju se bomo posvetili implementaciji aplikacije. Opisali bomo proces izdelave knjižnice za

lažje delo z vmesnikom WebGL, arhitekturo in sestavo aplikacije ter njenih komponent, orodja za obdelavo slik in uporabniški vmesnik aplikacije. Zadnje poglavje povzame dosežene cilje in spoznanja, ki smo jih pridobili pri izdelavi aplikacije.

Poglavje 2

O aplikaciji

Spletni urejevalnik slik je aplikacija, ki uporabniku omogoča urejanje slik preko spleta. S pomočjo aplikacije lahko uporabnik obreže sliko, na njej poudari določene elemente, ki jih slika prikazuje ali pa jo spremeni zato, da bo lažje prikazal bistvo in pomen slike. Aplikacije s pomočjo orodij omogočajo tudi izdelavo novih vsebin in kombinacijo z obstoječimi vsebinami, zato da se doseže ali izdelava nekaj novega. Ker je spletni urejevalnik dostopen preko spleta, omogoča uporabnikom hiter dostop do aplikacije na vsakem računalniku, ki je povezan z internetom, kar pomeni, da uporabniku ni treba nameščati dodatne programske opreme, ki zavzame veliko količino prostora na računalniku.

Aplikacija bo izdelana s tehnologijo WebGL za strojno podporo pri izrisovanju in boljše odzivnost. Zaradi uporabljene tehnologije bo aplikacija kompatibilna z operacijskimi sistemi Windows, OS X in Linux, ki imajo nameščen brskalnik, ki podpira WebGL. Na mobilnih napravah aplikacija ne bo delovala, saj večina mobilnih operacijskih sistemov še ne dopušča izvajanja vsebin WebGL preko spleta.

Urejevalnik slik bo deloval kot vsaka običajna spletna stran, s tem da bo imel podobo in funkcionalnosti namizne aplikacije. Spletne strani so sestavljene iz dveh ključnih delov, vidni del spletne strani (angl. Frontend) in strežniški del (angl. Backend). Vidni del spletne aplikacije uporabnik neposredno spremlja in z njim upravlja, medtem ko se strežniški del aplikacije izvaja na oddaljenem računalniku, kjer procesiramo uporabniške zahteve, shranjujemo podatke v podatkovnih bazah ter uporabniku dostavljamo zahtevane podatke. Vsak izmed omenjenih delov je izveden z različnimi tehnologijami. Za razvoj spletnih strani se največkrat uporabljajo HTML, CSS in JavaScript ali Flash, medtem ko strežniški del razvijamo v jezikih PHP, Python, Ruby, JavaScript, ASP.NET ali Java. Vsaka izmed naštetih tehnologij ima določene prednosti in slabosti,

predvsem pa veliko knjižnic, s katerimi si lahko pomagamo in poenostavimo razvoj.

Diplomsko delo se ne bo posvečalo implementaciji strežniškega dela predvsem zato, ker je za implementacijo tega sprva potrebna dobra aplikacija, ki že sama po sebi zahteva veliko pozornosti, poleg tega bi bilo težko testirati tak sistem, ki bi lahko služil večjemu številu uporabnikom brez performančnih težav, saj je zaradi ogromne količine podatkov, ki bi jih bilo potrebno shranjevati na strežniku, naloga težje izvedljiva.

V končnem pogledu je aplikacija namenjena vsem uporabnikom, ki želijo hitro, predvsem pa skoraj kadarkoli in kjerkoli urediti in obdelati kakšno sliko. Aplikacija bo uporabniku nudila intuitiven uporabniški vmesnik, preko katerega bo lahko uporabnik odpiral in pregledoval slike ter jih shranil na svoj računalnik. Z njih bo lahko odčital dimenzije slike in slike obrezal. S čopičem bo mogoče risati najrazličnejše linije in poteze v različnih barvah ter z različnimi nastavitvami čopičev. Uporabniku bo aplikacija omogočala uvoz večih slik v en sam dokument ter razporejanje, urejanje in sortiranje slikovnih elementov po plasteh. Z učinki bo mogoče spremeniti lastnosti in stil slike, poskrbljeno pa bo tudi za pregled vseh akcij, ki so bile narejene in razveljavitev tistih, ki jih ne potrebujemo ali pa so bile narejene pomotoma.

2.1 Cilji

Cilj diplomskega dela je implementirati aplikacijo za urejanje slik, s katero bo mogoče narediti naslednje:

- odpiranje, pregledovanje in shranjevanje dokumentov,
- rezanje slik,
- razvrščanje slik po plasteh,
- premikanje različnih plasti,
- odčitavanje barve posameznih pikslov v sliki s pomočjo kapalke,
- risanje najrazličnejših vzorcev in linij s pomočjo čopiča,
- brisanje določenih delov slike,
- dodajanje različnih učinkov za izboljšanje fotografij,
- pregledovanje zgodovine operacij in možnost razveljavitve neželjenih operacij.

Poglavje 3

Uporabljene tehnologije

V okviru razvoja spletne aplikacije smo uporabili več različnih tehnologij in ogrodij, ki smo jih uporabili pri izdelavi spletne aplikacije. Te bodo predstavljene v tem poglavju.

3.1 Ruby on Rails

Ruby on Rails [22] je odprtokodno ogrodje in skupek knjižnic za izdelavo spletnih aplikacij, ki si prizadeva za implementacijo dobrih filozofij in pristopov, ki se uporabljajo pri programiranju, kot so model-pogled-nadzornik (angl. Model-View-Controller), konvencija pred konfiguracijo (angl. Convention over Configuration), ne ponavljaj se (angl. Don't Repeat Yourself ali DRY) ter drugimi.

3.2 Middleman

Middleman [23] je dodatek za Ruby on Rails, ki se uporablja za generiranje statičnih spletnih strani in razvijalcu ponuja možnosti, kot so izdelava predlog za spletne strani, ločevanje produkcijske in razvojne kode, optimizacija kode ter možnost uporabe vrste drugih jezikov, kot so Haml, ERB, SASS, CoffeeScript ter drugi. Ogrodje vsebuje paket drugih dodatkov, s pomočjo katerih si zelo poenostavimo razvijanje in optimizacijo spletnih aplikacij in delo z datotekami ter podatkovnimi bazami.

3.3 HTML

HTML (angl. HyperText Markup Language) je označevalni jezik za izdelavo spletnih strani. Obstaja več verzij jezika, zadnja je verzija 5, katere specifikacije so dostopne na <http://www.w3.org/html/>.

pne na viru [24]. Temeljni elementi jezika HTML so oznake, kot je na primer oznaka `<html></html>`, ki označuje začetek in konec dokumenta HTML. S pomočjo oznak določimo strukturo in pomen elementov v spletni strani, ki jih potem brskalnik interpretira ter prikaže. HTML-dokument sestoji iz dveh delov, glave in telesa. Glava vsebuje metapodatke spletne strani in vire, ki jih spletna stran potrebuje za prikaz in delovanje, medtem ko telo vsebuje hierarhijo elementov, ki jih vidimo kot spletno stran.

V jeziku HTML obstaja vrsta gradnikov oz. elementov vse od tabel, seznamov, slik, vnosnih polj in izvlečnih seznamov, s katerimi si lahko pomagamo pri gradnji spletnih strani. Poleg naštetih elementov obstajajo tudi elementi, ki jih uporabnik ne opazi, temveč služijo za gradnjo postavitve strani. Kontrole, ki privzeto niso na voljo, je mogoče narediti ali prilagoditi s pomočjo gnezdenja obstoječih elementov v manjše strukture. Večina gradnikov dokumenta HTML ima dokaj jasen namen, element za prikazovanje slike prikazuje slike, element za vnos podatkov služi za vnos podatkov, izbirni seznam služi za izbiro določene vrednosti itd. Obstaja pa element **Canvas**, s pomočjo katerega lahko izdelamo celotne aplikacije in igre, ki ga bomo spoznali v naslednjem podpoglavju.

3.3.1 Element HTML Canvas

Element **Canvas** se na prvi pogled ne razlikuje veliko od tipičnih elementov HTML. Tako kot pri vseh elementih lahko elementu določimo pozicijo, stil, velikost itd. Za razliko od ostalih elementov pa njegov vidni del deluje kot površina za risanje. S pomočjo jezika JavaScript lahko od elementa zahtevamo enega izmed dveh kontekstov, s katerim lahko rišemo v element **Canvas**. Eden izmed kontekstov za risanje se imenuje 2D oz. `HTMLCanvasRenderingContext2D` [31], s pomočjo katerega lahko na element **Canvas** rišemo kroge, elipse, poligone, slike, krivulje, gradiente, linije in besedilo. Izrisano sliko lahko nato iz elementa pridobimo in jo uporabimo za vrsto drugih namenov. Element **Canvas** nam torej omogoča, da v njem prikažemo kakršno koli vsebino, enostavno povedano deluje kot zaslon, ki prikazuje rezultat programa. S pomočjo njega lahko prikažemo nove komponente uporabniškega vmesnika, vizualizacije, grafe, igre itd.

Poleg 2D konteksta lahko od elementa **Canvas** pridobimo še en kontekst, ki se imenuje **WebGL** oz. `WebGLRenderingContext`. S pomočjo tega konteksta imamo dostop do programskega vmesnika **WebGL**, ki nam omogoča izris kompleksne 2D in 3D gra-

fike, končen izris pa je tako kot pri 2D kontekstu viden na površini elementa `Canvas`. Kontekst WebGL in njegove funkcionalnosti pa bomo spoznali v poglavju WebGL.

3.4 CSS

CSS (Cascading Style Sheets) [25] je označevalni jezik, ki opisuje stil spletnih strani in loči strukturalne del strani od stilskega. Z jezikom CSS lahko definiramo lastnosti gradnikov uporabniškega vmesnika, kot so barve, ozadje, velikost, pozicija, debelina, stil robov in velikosti teksta v njih. Jezik CSS sestoji iz selektorjev in opisnih blokov. Selektorji določajo, kateri element ali skupina elementov bo vsebovala lastnosti, ki jih definiramo v opisnem bloku. Tako kot HTML jezik CSS interpretira brskalnik in s pomočjo njega stilizira elemente, ki so definirani v dokumentu HTML.

3.5 SASS

SASS (Syntactically Awesome Stylesheets) [26] je skriptna nadgradnja jezika CSS, ki poleg stiliziranja ponuja dodatne programerske funkcionalnosti, kot so spremenljivke, operacije za računanje z različnimi enotami, enostavne funkcije za pogosto uporabljene lastnosti in gnezdenje. SASS deluje na strežniku in se uporablja samo za razvijanje, za produkcijsko rešitev pa se prevede v CSS, preden se servira odjemalcu, za kar navadno poskrbi razvojno okolje ali strežnik.

3.6 JavaScript

JavaScript [27] je dinamični skriptni jezik, s katerim lahko spletnim stranem dodamo interaktivnost in dinamičnost. S pomočjo njega je možno narediti preverjanje podatkov v obrazcih, animacije in interakcijo z gradniki spletnih strani, asinhrono komunikacijo s strežnikom itd. Z jezikom JavaScript lahko spreminjamo dokument HTML in stil gradnikov, brskalnik pa poskrbi za prikaz v primeru sprememb vrednosti elementov. Jezik lahko uporabljamo tudi za izdelavo logike iger ali kompleksnih spletnih strani in aplikacij.

Jezik JavaScript omogoča programerju pisanje logike v različnih vzorcih oz. paradigmah. Naloga je izvedena v objektno orientiranem slogu, ki nam omogoča hranjenje podatkov in skupnih funkcij zaključene enote v objektih, ki med sabo komunicirajo

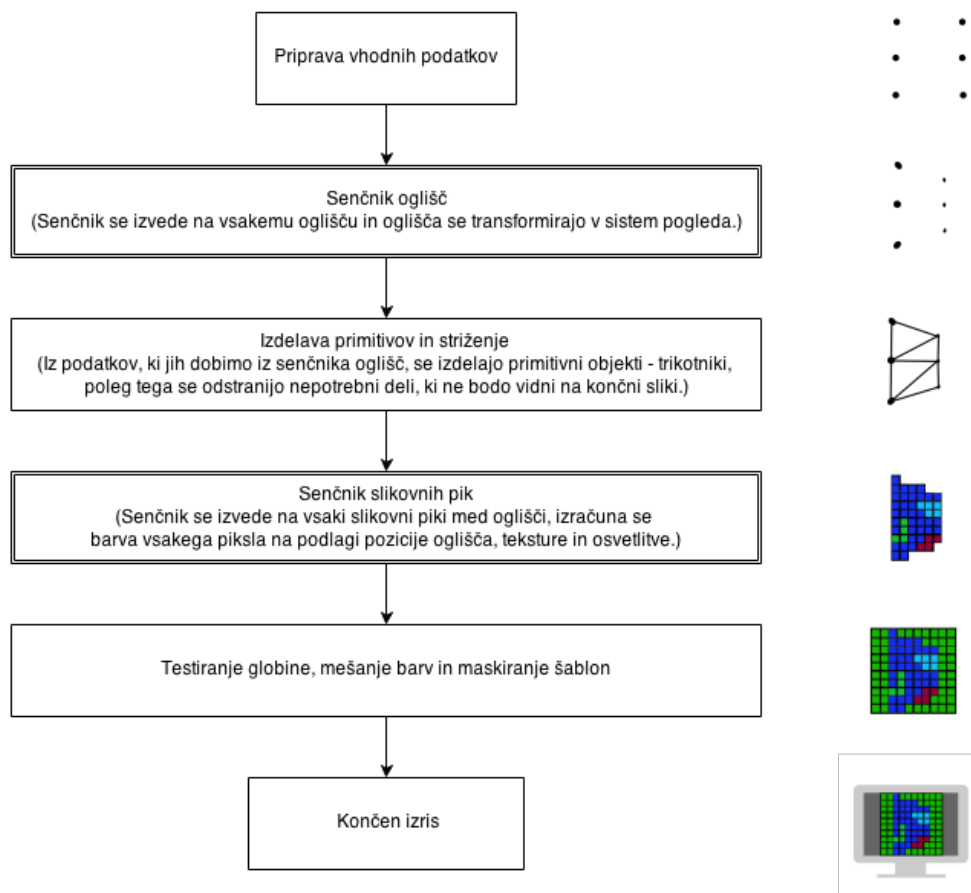
in hierarhično gradijo aplikacijo. Z jezikom JavaScript bo implementirana logika aplikacije, interakcija z uporabniškim vmesnikom in spreminjanje dokumenta HTML, obdelava vhodnih naprav in prikaz podatkov. S pomočjo jezika JavaScript dostopamo tudi do programskega vmesnika WebGL, ki ga bomo uporabljali za prikaz slikovnih podatkov.

3.7 WebGL

WebGL (Web Graphics Library) [29] je grafični programski vmesnik (API) za jezik JavaScript, s pomočjo katerega lahko rišemo kompleksno 2D in 3D grafiko s strojno podporo, brez dodatkov v brskalniku. Teksture in objekte izrisuje grafična kartica (GPU), ki je mnogo učinkovitejša za to nalogo kot pa splošno namenski procesorji (CPU), kar omogoča izdelavo hitrih interaktivnih aplikacij in iger. Jedro WebGL so senčniki (angl. Shaders), preko katerih nadziramo, kako bodo objekti in teksture prikazani na ekranu. Poznamo dva tipa senčnikov, ki se uporabljata v vseh 3D grafičnih vmesnikih, to sta senčnik objektov in senčnik pikslov. Prvi poskrbi za transformacijo modelov, drugi pa za izris posameznih slikovnih pik.

Programski vmesnik temelji na tehnologiji OpenGL ES 2 in ni objektno orientiran, temveč je skupek funkcij, ki jih uporabljamo po določenih postopkih. WebGL ima težko krivuljo učenja, kar pomeni, da je tehnologijo težje razumeti kot ostale spletne tehnologije in programske vmesnike, predvsem zato, ker je potrebnega veliko predznanja in osnovne kode, preden sploh lahko kaj dosežemo, poleg tega nimamo vedno takojšnjega odziva, kot smo tega vajeni pri izdelavi spletnih strani in pisanju stilskih predlog. Razhroščevanje zna biti naporno, saj senčnikov ne moremo razhroščevati, ker se nahajajo in izvajajo v pomnilniku grafične kartice, do katerega pa ne moremo dostopati.

Za lažje razumevanje, kaj nam omogoča programski vmesnik WebGL, bomo na kratko pregledali višje nivojske koncepte, da bomo bolje razumeli, kako deluje tehnologija in njemu podobni programski vmesniki. V končnem pogledu nam WebGL služi za to, da nekaj prikažemo na zaslonu. Da to dosežemo, se izvede serija postopkov, ki jo imenujemo grafični cevovod (slika 3.1). Grafični cevovod, kot ga opisuje vir [1], je skupek postopkov (avtomat), ki se izvedejo na grafični kartici zato, da lahko določene podatke prikažemo na zaslonu. Preden pa nekaj izrišemo, potrebujemo podatke, da grafični cevovod sploh lahko kaj izriše. Vhod grafičnega cevovoda so podatki o modelu, na katere nalepimo teksture, izrišemo pa jih s senčniki.

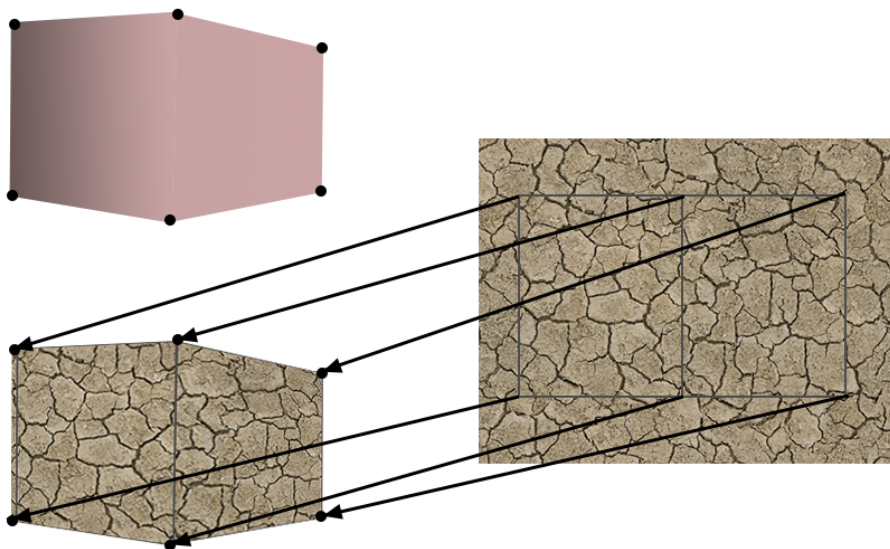


Slika 3.1: Prikaz delovanja grafičnega cevovoda.

3.7.1 Modeli in teksture

Model je struktura podatkov o točkah (angl. Vertices), iz katerih grafična kartica lahko izriše trikotnike, ki prikazujejo neko obliko, ki jo lahko vidimo na zaslonu. S trikotniki lahko prikažemo poljuben lik (kvadrat, pravokotnik, večkotnik), pokrajino, osebo ali pa celoten svet. Seveda so modeli sami po sebi enolični, saj lahko določimo le barve točk, ki se med točkami interpolirajo. Torej v primeru, da bi želeli prikazati sliko resolucije 20x20, bi porabili 722 trikotnikov (19 stranic * 19 stranic * 2 trikotnika), kar povzroči velike performančne probleme in težko nadziranje vseh točk, saj moramo za vsako točko podati točno lokacijo ter barvo. Za prikaz natančnih slikovnih podatkov torej ne uporabljamo modelov, temveč teksture. Teksture so slike, ki so nalepljene na model z namenom, da modelu dodamo več detajlov na površini modela. Zato, da v senčniku vemo, kam nalepiti teksturo na modelu, pa mora model vsebovati informacijo o teksturnih koordinatah (slika 3.2), ki pove, kateri del slike se nahaja v katerem

oglišču. Informacijo o teksturnih koordinatah pridobimo s senčnikom oglišč, medtem ko s senčnikom slikovnih pik pridobimo posamezen piksel iz teksture.



Slika 3.2: Prikaz mapiranja teksture z desne strani na model na levi strani. Vsako oglišče vsebuje koordinato iz slike, zato da vemo, kateri del slike se nahaja v katerem oglišču.

3.7.2 Senčniki

Spoznali smo torej modele in teksture, prvi so enostavni primitivni objekti ali pa kompleksne oblike, medtem ko teksture uporabljamo za to, da modelom dodamo detajle in informacijo o tipu površine. Za izris modelov in tekstur uporabljamo senčnike. Senčniki so programi pisani v jeziku GLSL (OpenGL Shading Language) [30], ki se izvajajo na grafični kartici. Kot smo že omenili, poznamo dva tipa senčnikov, senčnik točk in senčnik slikovnih pik. Prvi skrbi za izračun položaja posameznih točk in njihovih parametrov v prostoru, drugi pa izračuna končno barvo vsakega piksla posebej. Ob vsakem izrisu se senčnik oglišč izvede na vsakem oglišču, medtem ko se senčnik pik na vseh pikah med oglišči, kar prikazujeta drugi in četrti korak slike 3.1.

Za prikaz torej potrebujemo vsaj podatke o ogliščih, ki tvorijo model, teksture, s katerimi dodamo detajle površinam, ki niso obvezne in senčnike, s katerimi izračunamo položaj točk na zaslonu in barvo slikovnih pik. Senčniki so jedro tehnologije in so

edina enota v grafičnem cevovodu (slika 3.1), ki ima programabilen vmesnik, s katerim je možno rešiti zahtevne grafične probleme, ki pa jih lahko kombiniramo še z drugimi funkcionalnostmi tehnologije, kot so mešanje barv (ang. Blending), globinski pomnilnik (angl. Depth Buffer), izris v teksture (angl. Render to Texture) in šablone (angl. Stencil).

3.7.3 Mešanje barv

S pomočjo mešanja lahko določimo, kako grafični cevovod izračuna nove barve v primeru, da se dve barvi prekrivata. Funkcija za izračun seveda ni poljubna, ampak je izračunana po vnaprej definiranih formulah (enačba 3.1), v katerih lahko spreminjamo le faktorje in operacijo med pomnoženimi komponentami. Konkretno v napisanih enačbah (enačba 3.1) lahko spreminjamo vrednosti spremenljivk *SrcFact* in *DestFact* ter *SrcAlpFact* in *DestAlpFact*. Vrednosti teh spremenljivk so vnaprej definirane in so razvidne iz specifikacij [29] pod specifikacijami konstant (*BlendingFactorDest* in *BlendingFactorSrc*). *SrcRGB* je barva piksla, ki jo bomo trenutno izrisali, medtem ko je *DestRGB* barva, ki jo prekrivamo. Enako velja za prosojnost, *SrcA* določa prosojnost barve, ki jo izrisujemo, medtem ko *DestA* določa prosojnost, ki jo prekrivamo.

$$\begin{aligned} RGB &= SrcFact * SrcRGB + DestFact * DestRGB \\ A &= SrcAlpFact * SrcA + DestAlpFact * DestA \end{aligned} \quad (3.1)$$

S kombinacijo faktorjev in operacij med pomnoženima vrednostima lahko torej naredimo različne učinke in mešanja. Najbolj uporabne enačbe za mešanje barv so:

- zlivanje s prosojnostjo (angl. Alpha Blending)

$$\begin{aligned} RGB &= 1 * SrcRGB + (1 - SrcA) * DestRGB \\ A &= 1 * SrcA + (1 - SrcA) * DestA \end{aligned} \quad (3.2)$$

- dodajanje (angl. Add)

$$\begin{aligned} RGB &= 1 * SrcRGB + 1 * DestRGB \\ A &= 1 * SrcA + 1 * DestA \end{aligned} \quad (3.3)$$

- množenje (ang. Multiply)

$$\begin{aligned} RGB &= DestRGB * SrcRGB + 0 * DestRGB \\ A &= DestA * SrcA + 0 * DestA \end{aligned} \quad (3.4)$$

3.7.4 Globinski pomnilnik

Z mešanjem lahko določimo, kako trenutno izrisana barva prekrije prejšnjo. Privzeto vsak naslednji izris prekrije prej izrisanega, zato je potrebno oddaljene objekte izrisati prej kot tiste, ki so bližje očesu, lahko pa izkoristimo globinski pomnilnik, ki omogoča avtomatično detekcijo globine posamezne slikovne pike. Na žalost funkcionalnosti ne moremo uporabiti, ko izrisujemo prosojne objekte, saj morajo biti objekti za njimi izrisani pred objektom, ki jih pokriva, da pride do pravilnega mešanja barv. Globinski pomnilnik deluje tako, da si zapomni globino za vsak izrisan piksel in ne za vsak objekt, kar ne zagotavlja pravilnega prekrivanja, če izrišemo neprosojen objekt za prosojnim.

3.7.5 Izris v teksture

Omenili smo teksture, ki nosijo podatke o barvah posameznih pikslov slike. Te slike naložimo s trdega diska, kar pomeni, da nosijo vedno enake podatke. V primeru, da bi želeli teksturo spremeniti in tako hraniti dinamične teksture, lahko uporabimo možnost izrisa v teksturo. S pomočjo te funkcionalnosti lahko rišemo v teksturo namesto na zaslon. Teksturo lahko potem uporabimo za podatkovni vhod v senčniku ali pa za prikaz podatkov, ki niso del trenutnega izrisa in jih je potrebno izrisati ločeno.

3.8 GIT

Git [28] je porazdeljen sistem za upravljanje in hranjenje izvorne kode, ki temelji na nelinearnem poteku dela. Vsak git direktorij je enkopraven repozitorij, kar pomeni, da lahko izvorno kodo hranimo porazdeljeno in repozitorije poljubno združujemo med sabo. Git ponuja tudi nekaj naprednih funkcionalnosti, kot so reference na druge repozitorije, odlagališča, vejitve in integriteto podatkov. Sistem je bil uporabljen za hranjenje celotne izvorne kode projekta in za lažje primerjanje in beleženje zgodovine ter napredka.

Poglavje 4

Teoretične osnove

Da bomo lažje razumeli, kako implementirati funkcionalnosti urejevalnika slik, je ključno, da razumemo, kako hranimo slike na računalniku, kako jih prikazujemo in kakšne operacije lahko izvajamo nad njimi ter kaj pravzaprav urejevalniki slik sploh počnejo. Poglavje opisuje prikaz in hranjenje slikovnih podatkov na računalniku, osnovne operacije, s katerimi preoblikujemo slike ter predstavitev tipičnih funkcionalnosti urejevalnikov slik.

4.1 Prikaz in hranjenje podatkov

Računalniki so naprave, ki obdelujejo diskretne podatke, kar pomeni, da imajo omejeno natančnost. Slike, ki jih vidimo na ekranu, si lahko predstavljamo kot mrežo barvnih pikslov v primeru, da zaslon pogledamo pod drobnogledom. Pikel je podatkovna struktura, ki hrani informacijo o barvi. Ekran prikazuje barve s pomočjo 3 majhnih lučk (rdeča, zelena in modra - RGB), ki skupaj z različno intenziteto posamezne lučke prikažejo milijone različnih barv. Zaradi takega prikazovanja barv v računalniku navadno hranimo barve v formatu RGB, za katerega skupaj porabimo 24 bitov oz. 8 bitov za posamezno barvo. Poleg rdeče, modre in zelene barve hranimo še informacijo o prosojnosti barve (angl. Alpha), ki ji prav tako namenimo 8 bitov. Prosojna slika za zapis barve v formatu RGBA torej potrebuje 32 bitov oz. 4 bajte. Ekran prosojnih barv sicer ne prikazuje, informacijo o prosojnosti uporabimo samo takrat, ko je potrebno dve barvi zmešati. V takem primeru nam prosojnost pove, koliko posamezne barve moramo uporabiti, da bomo dobili končno barvo. Ta proces imenujemo zlivanje s prosojnostjo (enačba 3.2).

Poleg formata RGB obstaja še vrsta drugih zapisov, ki pa se večinoma uporabljajo

pri shranjevanju slik v datotečni sistem, pri pretvarjanju barv za tisk in kodiranju. V primeru, da sliko kakorkoli obdelujemo, se slika skoraj vedno pretvori v format RGB ali RGBA za hranjenje v pomnilniku računalnika ali grafične kartice.

Sliko na ekranu torej vidimo kot dvodimenzionalno tabelo barv, v pomnilniku pa jih hranimo kot enodimenzionalno polje bajtov RGBRGBRGB ... skupaj s širino in višino slike. S temi informacijami lahko izračunamo, kje v polju se nahaja barva na določeni koordinati s pomočjo enačbe 4.1, v kateri predpostavljamo, da dostopamo do polja barv in ne bajtov.

$$RGBbarvaNaKoordinatiXY = y * sirinaSlike + x \quad (4.1)$$

4.2 Operacije

Slike na računalniku hranimo kot polja barv, nad katerimi lahko izvajamo vrsto operacij. V nadaljevanju bomo predstavili primitivne operacije, ki so osnova za več zahtevnih postopkov, ki se uporabljajo pri obdelavi slik.

Zamenjava barv je preprosta operacija, pri kateri vzamemo določene barve s slike in jih zamenjamo z drugimi.

Premik je operacija, pri kateri vse barve slike ali pa le del zamaknemo v določeno smer za določeno število pikslov. Pri tej operaciji lahko izgubimo del slike, če premaknemo pike skozi velikost slike.

Povečanje ali pomanjšanje slike je operacija, pri kateri spremenimo dimenzije slike in s tem izgubimo določeno natančnost (resolucijo) slike. Nove barve slike izračunamo s pomočjo vzorčenja slike in interpolacije med barvami, rezultat ter kvaliteta pa sta odvisni od interpolacijskega algoritma. Za povečavo ali pomanjšavo boljše kvalitete se najpogosteje uporablja bikubična interpolacija (angl. Bicubic Interpolation) ali bilinearna interpolacija (angl. Bilinear Interpolation), v izjemnih primerih pa se uporablja tudi vzorčenje najbližjega soseda (angl. Nearest Neighbor).

Mešanje (angl. Blending) je postopek, pri katerem združimo dve sliki z različnimi metodami. Navadno za mešanje barv uporabljamo alfa kanal barve, ki določa, kakšen odstotek barve naj se uporabi za izračun končne barve. Združitev dveh slik ponavadi poteka z metodo zlivanja s prosojnostjo (enačba 3.2) ali pa z vrsto

drugih metod, kot so odštevanje, seštevanje (enačba 3.3), množenje (enačba 3.4) itd.

Rezanje slike (angl. Crop) je operacija, pri kateri iz določene slike vzamemo manjšo množico slikovnih pik in pri tem izgubimo del prvotne slike. Rezultat operacije je obrezana prvotna slika.

Učinki (angl. Effects) so operacije, ki po določenem algoritmu ali matematični formuli pretvorijo izbrane piksele. Poznamo več vrst učinkov, najpogosteje se uporabljajo učinki za spreminjanje lastnosti slike, kot so kontrast, svetlost, barvna uravnoteženost, nasičenosti, sence itd. Poleg učinkov za spreminjanje lastnosti pa poznamo tudi učinke, ki jih imenujemo filtri (angl. Filters), katerih rezultat je spremenjena slika. Sem spadajo učinki, ki stilizirajo sliko in konvolucijski filtri, kot so odkrivanje robov, zameglitev, izostritev itd.

Transformacija je sprememba slike ali dela slike, pri katerem se uporabijo matrične operacije, kot so premik, skalacija in rotacija.

Kompozitne operacije so operacije, pri katerih uporabimo zaporedje več različnih operacij in postopkov, da pridemo do željenega rezultata orodja, s katerim lahko obdelujemo ter preoblikujemo slike.

4.3 Sestava urejevalnikov slik

Aplikacije za urejanje slik uporabnikom nudijo orodja za preoblikovanje in urejanje slik. Glavni element, ki ga uporabnik ureja, je dokument, ki pa ga pogosto imenujemo kar slika. V dokumentu hranimo eno ali več plasti (angl. Layer), ki nosijo podatke o slikovnih elementih. Slikovni elementi so lahko slike, besedilo ali pa vektorska grafika oz. vsak element, ki ga lahko prikažemo oz. izrišemo. V enem dokumentu torej hranimo več slikovnih elementov, ki jih imenujemo kar plasti, te pa si lahko predstavljamo kot skladovnico polprosojnih elementov oz. slik, kjer vsaka naslednja slika prekriva prejšnjo, skupaj pa tvorijo končno sliko. V primeru, da zamenjamo vrstni red plasti, lahko spremenimo končno sliko, ki jo vidimo na zaslonu. Plasti uporabljamo za organiziranje elementov v dokumentu, za lažje urejanje kosov kompozicije in enostavnejše urejanje delov slike, ki drugače ne bi bila možna, če bi dokument obravnavali kot eno samo sliko.

Pri kreativnem procesu si pomagamo z orodji, s katerimi urejamo plasti, jih preoblikujemo in izdelujemo nove slikovne elemente, poleg tega pa nam orodja služijo kot pomoč pri delu ter posameznih postopkih. Enostavna orodja so na primer orodja za bližanje in oddaljevanje (angl. Zoom), s pomočjo katerega lahko pobližje pogledamo določene detajle dokumenta ali pa se oddaljimo, da vidimo celotno podobo. Orodje za navigacijo (angl. Pan) služi za premikanje v vse smeri po približanem dokumentu. Poleg orodij za navigacijo imamo na voljo orodja za premikanje plasti (angl. Move) in transformacijo plasti (angl. Transform). Sledijo jim orodja za risanje kot na primer orodje za risanje likov, črt in drugih oblik in čopič za risanje najrazličnejših vzorcev ter tekstur. Orodjem za risanje nasprotuje radirka, s katero lahko izbrišemo določen del slike, deluje pa podobno kot čopič, vendar ne dodaja barv, temveč jih odstranjuje. Besedilo in napise lahko urejamo in vstavimo v sliko z orodjem za tekst. Slike pa je mogoče povečati ali pomanjšati in obrezati. Za poudarjanje delov slike in transformacijo slikovnih pik skrbijo različni učinki in filtri, s katerimi lahko spreminjamo lastnosti slike, kot so nasičenost, svetlost, kontrast, ostrina, količina barve ter stil.

Poleg orodij vsak urejevalnik omogoča uvoz slik iz večih formatov in shranitev letih v različne formate. Uporabniški vmesnik navadno vsebuje veliko površino, kjer je viden trenuten dokument, obdaja pa ga več panelov, ki so namenjeni prikazu in urejanju podatkov dokumenta in orodij ter jih je možno razporediti po želji. Tipični paneli so panel za organiziranje plasti, panel za izbiro barve, panel za izbiro čopiča in nastavitve parametrov čopiča in panel za pregled zgodovine in razveljavitev akcij.

4.4 Potrebne funkcionalnosti WebGL

WebGL je napredna tehnologija, spoznali smo funkcionalnosti, ki nam jih ponuja, vendar za izdelavo aplikacije ne potrebujemo vseh. Za implementacijo urejevalnika slik bomo pretežno izkoriščali teksture, za modele bomo večinoma uporabljali samo ploskve, na katerih bodo teksture, ki bodo glavni nosilci podatkov. Izkoriščali bomo risanje v teksture, saj je to najhitrejši način za spreminjanje tekstur in hranjenje sprememb. V nasprotnem primeru bi lahko teksturo naredili tudi v navadnem pomnilniku in jo sprti prenašali na grafično kartico, kar pa bi bilo neučinkovito tako za procesor, ki bi za zapis slike potreboval mnogo več časa, kot tudi za stalno prenašanje podatkov na grafično kartico.

Poglavje 5

Urejevalnik slik

V poglavju bomo opisali, kako je potekala implementacija urejevalnika slik. Začeli bomo s predstavitvijo knjižnice za lažje delo z vmesnikom WebGL, sledila bo predstavitev arhitekture izdelane aplikacije in opis njenih komponent, podatkovnih struktur itd. Nadaljevali bomo s predstavitvijo implementacije orodij, zaključili pa bomo z uporabniškim vmesnikom.

5.1 Knjižnica za delo z WebGL

Spoznali smo, da WebGL vsebuje kompleksen programski vmesnik, zato za enostavno delo z WebGL potrebujemo knjižnico, ki je učinkovita pri izrisovanju večih tekstur tako na zaslon kot tudi v teksturo. Glede na to, da se večina knjižnic za delo z WebGL osredotoča na 3D grafične pogone, je bilo potrebno narediti knjižnico za enostavno delo z 2D primitivnimi objekti.

Knjižnica je bila narejena z namenom, da se programerju ni potrebo osredotočati na delo z vmesnikom WebGL, izdelavi podatkovnih struktur in optimizacijo podatkov, kot jih opisuje članek [4], v katerem so povzeti nasveti za učinkovito delo z OpenGL ES 2, na katerem temelji WebGL. Knjižnica prikriva drobovje vmesnika WebGL in za lažje delo programerju ponuja programski vmesnik `SpriteBatch` (izsek kode 5.1). Pri izdelavi knjižnice smo se zgledovali po knjižnicah za delo z 2D grafiko, pisanih v drugih jezikih, kot so XNA [18], libGDX [19] in DirectX Toolkit [20].

Za risanje slik s pomočjo knjižnice potrebujemo samo dva razreda, `Texture2D` in `SpriteBatch`. Prvi razred hrani slikovne podatke in nam omogoča nalaganje slik, preko povezav ali pa iz podatkov v formatu RGB ali RGBA. Razred `SpriteBatch` pa ponuja paleto funkcij, s katerimi lahko izrišemo določeno teksturo na zaslon. Pri

izrisu določene teksture lahko določimo njen položaj, velikost, rotacijo, skalacijo, center rotacije in skalacije, barvo za potemnitev ter teksturne koordinate. Seveda vsi atributi niso potrebni za izris ene slike, zadostuje že sama tekstura, ostali parametri pa so neobvezni.

```
interface SpriteBatch{
    void begin(Matrix4 transform);
    void draw(Texture2D t, float x, float y, float width, float height,
        float rotation, float xScale, float yScale, Color tintColor,
        float[4] textureCoordinates);
    void end();
    void setRenderTexture(RenderTexture texture);
    void setShader(Shader shader);
}
```

Izsek kode 5.1: Vmesnik za risanje slik - SpriteBatch.

`SpriteBatch`, kot že samo ime razreda pove, izvaja izrise v serijah zaradi narave vmesnika WebGL, kjer je podatke pred izrisom vedno potrebno prenesti v grafični pomnilnik. Ker je postopek prenosa dolgotrajen, je bolje, da ga ponovimo čim manjkokrat in ob prenosu podatkov prenesemo čim večje število podatkov v grafični pomnilnik, ali pa še bolje, da vse podatke že hranimo v grafičnem pomnilniku in jih samo ponovno izrišemo, ko je to potrebno. Razred `SpriteBatch` uporabljamo tako, da pred izrisom z metodo `draw()` kličemo metodo `begin()`, s katero zaznamujemo, da smo pričeli z risanjem. Nato sledi serija izrisov z metodo `draw()`. Ko so izrisani vsi objekti pokličemo metodo `end()`, s katero zaznamujemo, da smo prenehali z risanjem. Metoda `end()` tako opravi najkompleksnejšo nalogo, kjer je potrebno vse podatke o izrisih zapakirati v učinkovito polje stuktur ter jih prenesti na grafično kartico, ki obdela podatke s pomočjo senčnikov.

Vmesnik (izsek kode 5.1) poleg izrisa na zaslon omogoča tudi izris v teksturo. Postopek risanja v teksturo se ne razlikuje veliko od risanja na zaslon. Preden začnemo z risanjem v teksturo, je potrebno nastaviti teksturo, v katero nameravamo risati, s pomočjo metode `setRenderTexture()` in že smo pripravljeni za risanje v teksturo. Zatem sledi običajen postopek klicev `begin()`, `draw()` in `end()`, s katerimi izrišemo zaporedje slik. Z risanjem v teksturo prenehamo s klicem metode `setRenderTexture(null)`, s katerim zaznamujemo, da bi radi vse nadaljnje izrise prikazali na zaslonu.

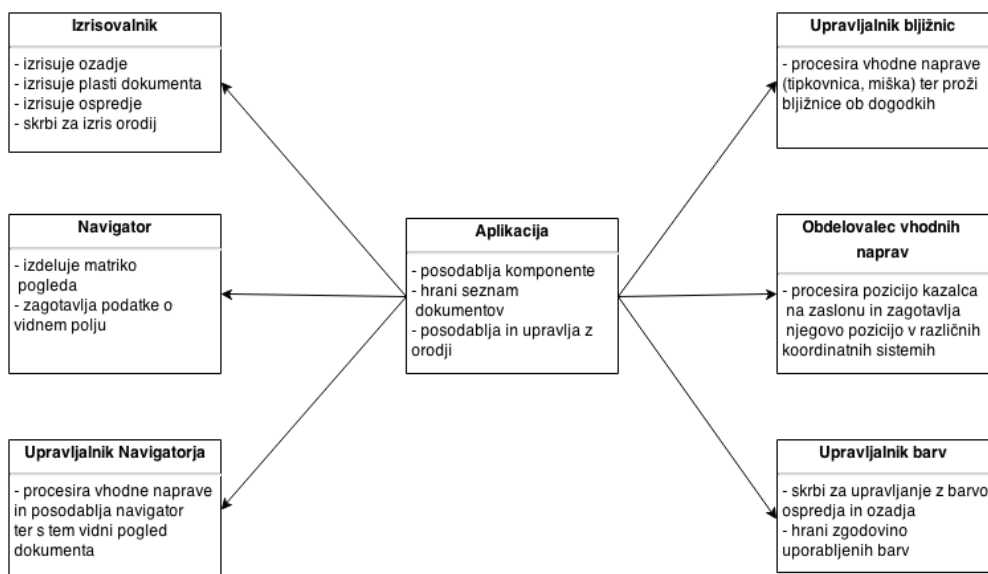
Razred `SpriteBatch` privzeto izrisuje teksture točno take, kot so njeni podatki. V primeru, da želimo kakorkoli spremeniti vhodno teksturo, je potrebno uporabiti drugi

senčnik. To lahko naredimo s pomočjo metode `setShader(Shader shader)`, s katero nastavimo senčnik oglišč in slikovnih pik, ki se bo nadaljnje uporabljal za izris.

Risanje v teksture in prilagodljivi senčniki niso nujno potrebni za uporabo vmesnika `SpriteBatch`, ampak so napredna funkcionalnost, ki se izkorišča v vseh kompleksnejših izrisih. Razred `Texture2D` hrani podatke o teksturi, pridobljene iz različnih virov, razred `Shader` pa hrani podatke o senčniku oglišč in senčniku pik ter vse vhodne spremenljivke, ki se uporabljajo v senčniku.

5.2 Arhitektura aplikacije

Za izdelave aplikacije, v kateri bo mogoče implementirati kompleksne funkcionalnosti, je potrebno postaviti stabilno arhitekturo in fleksibilne podatkovne strukture, v katerih bomo lahko hranili podatke zato, da bomo slike obdelovali hitro in učinkovito. V nadaljevanju bo opisana arhitekture aplikacije in osrednja podatkovna struktura - dokument, ki jo obdelujejo orodja.



Slika 5.1: Prikaz arhitekture aplikacije in njenih komponent.

Za organizacijo izvajanja nalog v aplikaciji in procesiranje podatkov se obsežnejše naloge v aplikaciji delijo med različne komponente. Vsaka komponenta ponuja svojo funkcionalnost, ki jo lahko druge komponente koristijo. Za inicializacijo komponent skrbi aplikacija, ki s komponentami tudi upravlja in jih posodablja. Komponente lahko delujejo samostojno ali pa so odvisne od drugih komponent in brez njih ne

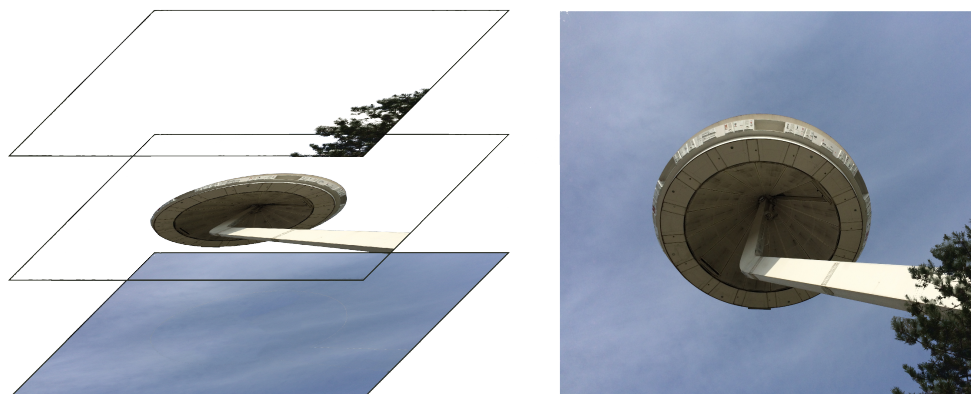
delujejo. Da je komunikacija med komponentami kar se da neodvisna, poteka po vzorcu dogodkovnega komuniciranja (angl. Event Driven Programming), ki se uporablja tudi v jeziku JavaScript za detekcijo interakcije na grafičnih elementih, kot je to opisano v viru [2]. V vzorcu dogodkovnega komuniciranja nastopajo dogodki in opazovalci, vsak dogodek zaznamuje določeno spremembo v aplikaciji, kot je na primer sprememba slike, pritisk tipke, shranitev nastavitvev itd., na katerega se lahko naročijo opazovalci, ki so obveščeni, ko se dogodek sproži. Opazovalec se ob spremembi odzove na dogodek s tem, da nekaj naredi. V primeru, da opazujemo pritisk gumba in je gumb bil pritisnjen, se bo opazovalec odzval z akcijo, kot je prikaz dialoga ali pa preusmeritev na novo spletno stran.

Aplikacija je torej skupek komponent (slika 5.1), ki porazdeljeno upravljajo večje naloge v aplikaciji, kot so izrisovanje dokumenta, upravljanje z vhodno-izhodnimi napravami, proženj bližnjic, procesiranje navigacije, upravljanje z barvami itd. V grobem se v aplikaciji pojavljata dva tipa komponent: komponente, ki ponujajo funkcionalnosti drugim komponentam in komponente, ki obdelujejo podatke. Slednje potrebujejo podatke oz. podatkovne strukture za obdelavo. V nadaljnjih podpoglavjih bomo predstavili zgradbo dokumenta in komponente aplikacije.

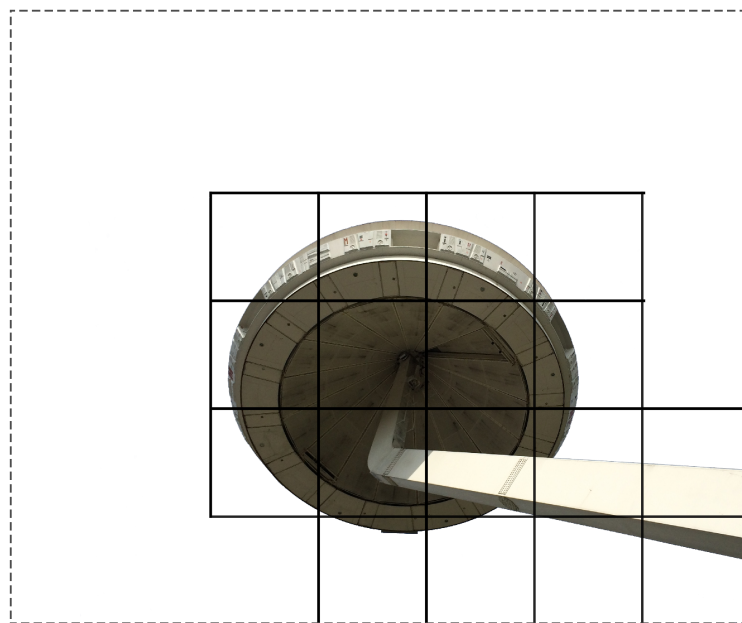
5.2.1 Dokument

Dokument je podatkovna struktura, ki vsebuje več plasti (slika 5.2), ki hranijo slikovne podatke in meta podatke, kot so ime, velikost ... Vsaka plast sestoji iz večih manjših tekstur, ki skupaj tvorijo površino (slika 5.3). Posamezno teksturo v površini imenujemo ploščica (angl. Tile), ki vsebuje podatke o velikosti teksturnih podatkov in položaj ploščice v površini. Površine delimo na več manjših, zato ker je velikost tekstur na grafični kartici omejena, da pa presežemo ta limit, delimo podatke med več manjših fragmentiranih podatkovnih struktur, kar omogoča porazdeljeno nalaganje slik, boljšo izkoriščenost pomnilnika ter dinamično rast površin. Površine torej lahko delimo na več vrst, in sicer navadne površine, ki vsebujejo fiksno število ploščic po višini in širini, in fleksibilne površine, ki spreminjajo velikost glede na potrebe velikosti risalne površine. Zato da omogočamo dinamično rast površin, mora vsaka komponenta ali orodje, ki izrisuje neki element v površino, zahtevati področje, na katerega želi risati. Površina je nato dolžna priskrbeti ploščice, na katere se lahko riše. Fiksne površine torej vrnejo le tiste ploščice v področju, ki jim je bilo podano ob inicializaciji, medtem ko fleksibilne površine dinamično dodajajo nove ploščice, v primeru da je zahtevana risalna površina

izven območja, ki ga površina že hrani.



Slika 5.2: Prikaz plasti v dokumentu s stranskega risa (levo) in končna kompozicija slikovnih elementov v dokumentu s pogledom od zgoraj (desno).



Slika 5.3: Prikaz zgradbe ene površine, ki sestoji iz večih manjših ploščic. Ploščice so poudarjene s črno obrobo, medtem ko je celotna velikost površine označena s črtkano črto.

Površina poleg vseh ploščic, ki jih hrani, vsebuje še podatke o zaklenjenosti, vidnosti in položaju površine, saj se le-ti lahko spreminjajo. Fleksibilne površine nadgrajujejo fiksne površine, ki jih uporabljamo kot osnovo za plasti, te pa hranimo v dokumentu in jih uporabljamo za prikaz vseh slikovnih elementov.

Dokument poleg plasti hrani tudi podatke o izbrani plasti, velikosti in imenu dokumenta ter zgodovino sprememb, ki so bile narejene od ustvaritve dokumenta. Naloga dokumenta je opazovanje sprememb, ki potekajo na plasteh, in beleženje vseh postopkov, ki so se izvedli na dokumentu. Vsak dokument vsebuje svojega upravljalca zgodovine, ki hrani vse akcije, ki so bile izvedene na njem. Upravljalec zgodovine ne hrani celotne podatkovne strukture dokumenta za vsako spremembo, temveč hrani le spremembe in meta podatke o spremembah, ki jih potrebujemo za rekonstrukcijo akcije. V primeru, da je potrebno razveljaviti določeno akcijo, se ponovno izvedejo vsi postopki vse od začetka izdelave dokumenta pa do izbrane akcije. Akcije v programu izdelujejo orodja, ki so odgovorna tudi za izdelavo metapodatkov in dodajanje akcij v upravitelja zgodovine. Orodja so načrtovana tako, da uporabniku omogočajo ustvarjanje določene spremembe, in ko se ta zavestno odloči za določeno spremembo, se sprememba zabeleži. Očiten primer uporabe akcij je na primer pri učinkih. Učinki omogočajo uporabniku, da spremeni sliko. Preko različnih kontrol lahko spreminjamo parametre in testiramo, kaj se bolje obnese. Ko je uporabnik zadovoljen, potrdi spremembo in naredi se nova akcija ter se izvede. Vsak tip akcije v aplikaciji zahteva svojo implementacijo in podatke, ki so potrebni za njeno rekonstrukcijo, metapodatke pa priskrbi orodje ali komponenta, odgovorna za izvršitev akcije. Dokumente hrani aplikacija, ki tudi upravlja z njimi in o spremembah obvešča opazovalce.

5.2.2 Izrisovalnik

Glavna funkcionalnost urejevalnika je prikazovanje izrisa dokumenta, za katerega skrbi izrisovalnik (angl. *Renderer*). To je komponenta, ki osvežuje okno dokumenta, ko se dokument ali pogled spremeni. Ob izrisu poskrbi, da se izrišejo vse plasti trenutnega dokumenta in vsi elementi, ki niso del dokumenta, kot na primer ozadje, robovi in deli izven risalne površine. Izrisovalnik vsebuje več podkomponent, ki se ukvarjajo z risanjem različnih elementov okna v katerem je prikazan dokument. Prvi element, ki je izrisan, je ozadje, katerega izriše izrisovalnik ozadja. Ozadje dokumenta je belo-siva tekstura v obliki šahovnice, ki je standardna tekstura za prikazovanje prosojnosti. Če dokument ne vsebuje nobene plasti s podatki ali pa vsebuje plasti s polprosojnimi elementi, bo v ozadju vidna šahovnica za občutek prosojnosti. Kvadratki v teksturi šahovnice so vedno enako veliki, ne glede na povečavo dokumenta, in se premikajo skupaj z dokumentom ob navigaciji. Na ozadje se izrišejo vse plasti dokumenta v istem vrstnem redu, kot je prikazana hierarhija plasti v uporabniškem vmesniku. Posamezna

plast je izrisana takrat, ko so izrisane vse ploščice, ki pripadajo plasti in si skupaj delijo lastnosti plasti, kot so položaj, prosojnost in vidnost, ki jih mora izrisovalnik upoštevati ter pravilno izrisati. Ko so plasti izrisane, izrisovalnik okvirja izriše obrobo dokumenta in zapolni prostor izven dokumenta z namenom, da se prekrijejo vse plasti, ki se morda nahajajo izven mej dokumenta. Okvir prav tako služi zato, da uporabnik vidi, kako bo izgledala kompozicija elementov v končni sliki. Poleg ozadja, plasti in okvirja izrisovalnik poskrbi tudi za izris sprememb orodja ter izris orodja samega v primeru, da ga ta potrebuje. Izrisovalnik ob vsaki spremembi ponovno izriše elemente dokumenta in pri tem upošteva trenutni pogled navigatorja.

5.2.3 Navigator in upravljalnik navigatorja

Navigator je komponenta, ki skrbi za upravljanje z vidnim poljem, ki ga uporabnik opazuje, in omogoča navigacijo po dokumentu tako, da lahko prikažemo celoten dokument ali pa le del tega, ne glede na to, kako velik je dokument, ki ga obdelujemo, ali zaslon, na katerem se vidi končna slika. Izrisovalnik vedno izriše vse plasti v koordinatnem izhodišču slike, ki se začne v zgornjem levem kotu risalne površine. Izris je nato transformiran z navigacijsko matriko, ki transformira celoten izris v pogled navigatorja. Navigator je torej zadolžen za izdelavo matrike, ki transformira trenutni izris dokumenta tako, kot ga uporabnik želi videti. Uporabnik nadzira vhodne naprave, kot so miška in tipkovnica, te naprave pa prožijo dogodke ob določenih akcijah uporabnika. Dogodke vhodnih naprav procesira upravljalnik navigatorja (angl. Navigation Controller), ki obdela podatke vhodnih naprav in jih pretvori v smiselne ukaze navigatorja, ki posodobi matriko pogleda. Matriko pogleda nato koristi izrisovalnik za transformacijo izrisa dokumenta in obdelovalec vhodnih naprav za transformacijo pozicij kazalca v koordinatni sistem slike.

5.2.4 Obdelovalec vhodnih naprav

Okno dokumenta prikazuje del dokumenta, tako kot ga uporabnik želi videti. Pogled transformira navigator z matriko pogleda, in če orodje uporablja pozicijo kazalca, je le-tega potrebno transformirati iz koordinatnega sistema pogleda v koordinatni sistem slike. Za preslikavo koordinat iz pogleda v koordinate slike skrbi obdelovalec vhodnih naprav (angl. Input Procesor), ki poleg preslikave vsem orodjem in komponentam ponuja še dostop do lokacije kazalca na zaslonu, lokacije kazalca na sliki ter spremembo položaja kazalca v primerjavi s lokacijo v prejšnjem posodobitvenem ciklu. Obdelovalec

vhodnih naprav torej skrbi samo za upravljanje kazalca, medtem ko za upravljanje s tipkami na tipkovnici in miški skrbi upravljalnik bližnjic.

5.2.5 Upravljalnik bližnjic

Upravljalnik bližnjic (angl. Shortcut Manager) skrbi za upravljanje vseh bližnjic v aplikaciji. Bližnjica je akcija, ki se sproži ob določeni kombinaciji tipk. Akcija bližnjice se lahko sproži glede na to, ali je bila določena kombinacija tipk pritisnjena ali spuščena ali pa je bila pritisnjena dalj časa. Če je bližnjica pritisnjena dalj časa, se akcija kliče vsak posodobitveni cikel, vse dokler je bližnjica pritisnjena.

5.2.6 Upravljalnik barv

Upravljalnik barv (angl. Color Manager) je komponenta, ki zagotavlja orodjem, ki uporabljajo barve, dostop do dveh barv, ki jih določi uporabnik. To sta barva ospredja in barva ozadja. Prva služi kot primarna barva, ki jo uporabljajo orodja, kot je čopič, medtem ko druga služi kot dodatna barva (na primer za obarvanje čopiča). Sekundarna barva se uporablja tudi za hitro izmenjavo dveh barv pri delu in primerjavo med dvema barvama. Poleg primarne in sekundarne barve upravljalnik barv skrbi še za hranjenje zgodovine barv, ki deluje po principu FIFO (angl. First In First Out), pri katerem imamo polje, ki hrani določeno število barv, novi elementi se vedno dodajajo na začetek polja in ko se polje napolni do svoje maksimalne velikosti, vedno odstranimo barvo, ki je v polju najdlje časa.

5.2.7 Upravljalnik sredstev

Poleg omenjenih komponent v aplikaciji uporabljamo še upravljalnik sredstev (angl. Resource Manager). Ta hrani seznam slik, ki so potrebne za delovanje programa in morajo biti prisotne v pomnilniku, ko jih potrebujemo. Upravljalnik sredstev rešuje problem asinhronne komunikacije, ki poteka za pridobivanje podatkov preko omrežja in sinhronega programskega vmesnika WebGL. Vsa komunikacija, ki poteka na spletnih straneh za pridobivanje slik in podatkov, je namreč asinhrona, kar pomeni, da pridobivanje slik s povezav ni instantno, temveč poteka v ozadju in traja nekaj časa, medtem ko WebGL sliko iz pomnilnika takoj prenese v pomnilnik grafične kartice. Zato da imamo slikovne podatke, kot so slike za čopiče in ozadja, vedno na voljo, uporabljamo upravljalnik sredstev, ki naloži seznam vseh slik, ki so potrebne za delovanje aplikacije,

preden zaženemo aplikacijo, te pa lahko nato kadarkoli pridobimo v aplikaciji preko upravljalnika sredstev.

5.2.8 Odpiranje in shranjevanje slik

Vsak urejevalnik slik mora uporabniku omogočati tudi uvoz in izvoz slik. Za uvoz v aplikaciji skrbi bralnik slik (angl. Image Reader), katerega naloga je, da prebere določeno datoteko, prepozna tip datoteke in v primeru, da je datoteka slika, prebere podatke iz slike ter uvozi podatke o slikovnih elementih v plasti dokumenta. Branje datotek je na spletnih straneh mogoče preko vmesnika File API [17], ki omogoča branje slik kot binarnih podatkov ali pa kot slike, kodirane v formatu base64, ki se uporablja za hranjenje slik v naslovu URL in za pridobivanje slik iz elementa `Canvas`. Branje binarnih slik je seveda mogoče, vendar je zaradi velikega števila različnih formatov težko podpreti vse, zato bralnik slik prebere slike v formatu base64, iz katerega lahko brez težav naredimo teksture in plasti dokumenta.

Za izvoz slike skrbi shranjevalnik slik (angl. Image Writer), ki mora vse plasti dokumenta pravilno združiti skupaj in priskrbeti podatke o slikovnih pikslih, ki jih lahko shranimo v datoteko. Slika je pred izvozom izrisana v element `canvas`, ki ima iste dimenzije kot slika sama. Element `Canvas` nam nato omogoča pridobitev slikovnih podatkov v formatu base64, ki ga lahko uporabnik shrani v obliki datoteke na svoj računalnik.

5.3 Orodja

V poglavju bomo opisali delovanje in namen orodij, ki so bila implementirana v aplikaciji in omogočajo uporabniku urejanje dokumentov ter slikovnih elementov.

5.3.1 Orodje za premikanje plasti

Orodje za premikanje plasti (angl. Move) nam omogoča premikanje plasti v dokumentu v vseh smereh. Orodje je samo po sebi enostavno in vse kar naredi, je to, da si zapomni koordinati ob začetku in koncu vlečenja plasti na zaslonu, te preslika v koordinatni sistem slike ter premakne trenutno plast na končno lokacijo.

5.3.2 Kapalka

Kapalka je orodje, ki nam omogoča pridobitev barv iz trenutnega slikovnega elementa. Orodje preslika lokacijo kazalca v koordinatni sistem slike in pridobi barvo točke na slikovni piki, ki se nahaja na trenutni plasti. Pridobljeno barvo nato nastavi kot izbrano primarno ali sekundarno barvo.

5.3.3 Orodje za obrezovanje

Orodje za obrezovanje (angl. Crop) služi za obrezovanje dokumenta. S pomočjo njega lahko iz dokumenta izrežemo manjši del slike. Orodje deluje tako, da s klikom in vlečenjem miške izriše polprosojno masko, ki ponazarja obrezano območje. Obrezano območje se sprti posodablja, medtem ko uporabnik vleče potezo, in ko je miškin gumb spušččen, je poteza zaključena ter uporabnik se lahko odloči za obrez slike ali pa ponovno naredi novo območje, ki ga želi izrezati.

Pravokotnik, ki nakazuje območje obreza, ki vsebuje črte, ki delijo dolžino in širino pravokotnika na tretjine za enostavnejšo postavitev kompozicije. Za natančne obreze pa lahko uporabnik vnese vrednosti obrezane slike v uporabniškem vmesniku. Ob potrditvi akcije za obrez orodje zmanjša velikost dokumenta in prestavi vse plasti tako, da se začnejo v zgornjem levem kotu obrezanega pravokotnika.

5.3.4 Čopič

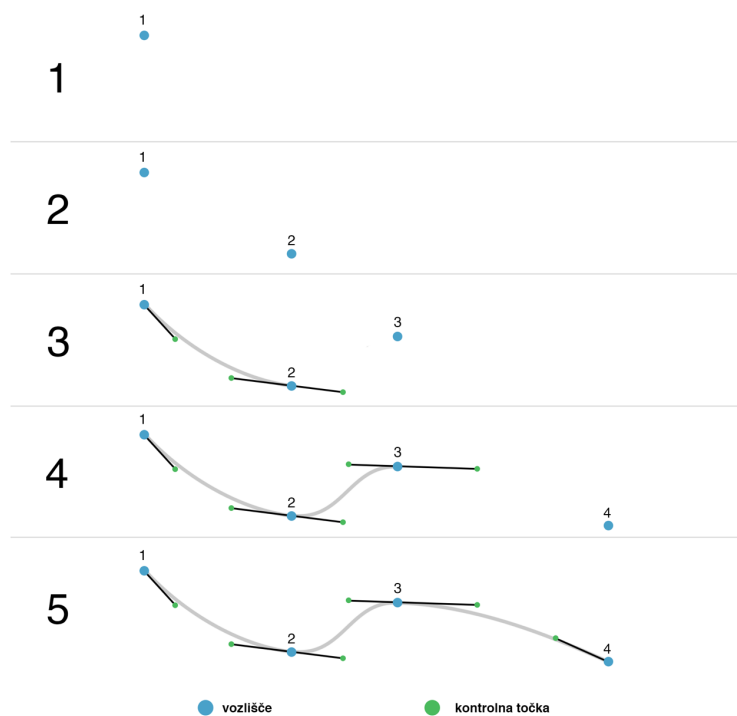
Čopič (angl. Brush) je eno izmed kompleksnejših orodij, saj omogoča risanje najrazličnejših vzorcev oz. tekstur na trenutno aktivno plast. Naloga čopiča je, da na krivulji, ki jo je uporabnik naredil s kazalcem, izriše določeno teksturo z nastavitvami čopiča. Tekstura določa obliko čopiča, nastavitve pa vplivajo na njegovo delovanje.

Orodje potezo čopiča nariše na začasno plast, kot to opisuje članek [3]. Uporabnik vidi izris začasne plasti, ne opazi pa, da se tam nahaja. Poteza se izrisuje na začasno plast, vse dokler uporabnik ne zaključi poteze. Ko je poteza zaključena, se združi s trenutno aktivno plastjo in postane del nje. Poteza čopiča se začne ob pritisku levega miškega gumba in se konča ob spustu. Krivulja, ki je narisana do spusta, se šteje kot ena poteza čopiča.

Na izris poteze čopiča vpliva več nastavitvev, ki jih lahko uporabnik nastavi:

- **Tekstura** je enokanalna slika, ki opisuje obliko in teksturo čopiča.

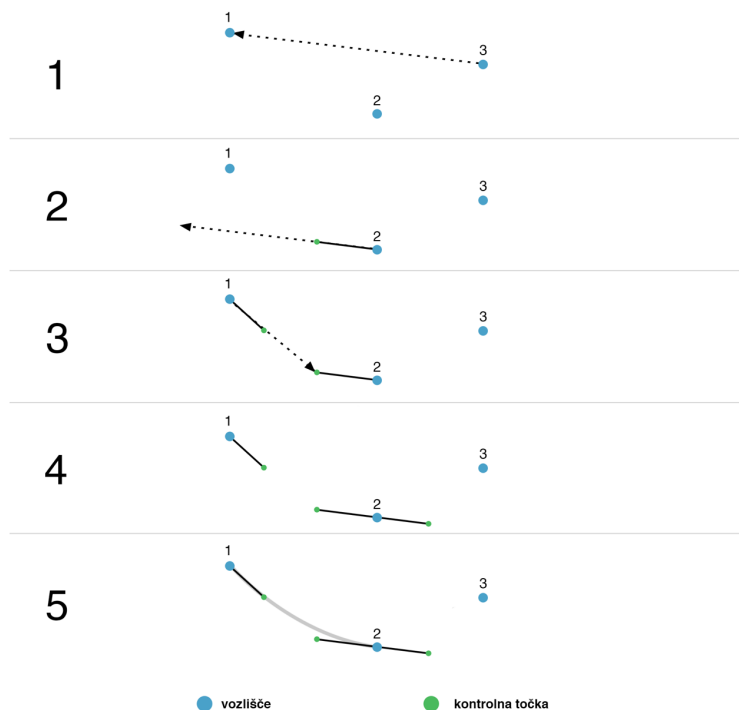
- **Razmik** (angl. **Spacing**) je prostor med dvema zaporedno izrisanimi teksturama.
- **Barva** (angl. **Color**) določa barvo čopiča.
- **Pretok** (angl. **Flow**) določa prosojnost posamezne teksture v potezi čopiča; v praksi s pretokom nadziramo, kolikšen odstotek barve odda posamezna tekstura.
- **Prosojnost** (angl. **Transparency**) je maksimalna prosojnost, ki jo čopič lahko doseže.
- **Velikost** (angl. **Size**) teksture čopiča v pikslih.
- **Rotacija** (angl. **Rotation**) posamezne teksture v potezi.
- **Skalacija** (angl. **Scale**) teksture čopiča po osi x.



Slika 5.4: Postopek rekonstrukcije poti z večimi Bézierovimi krivuljami, kjer kontrolne točke ležijo na tangentskih vozliščih, da se ohranja zveznost C^1 .

Izrisovanje čopiča poteka v realnem času po krivulji, ki nastaja na podlagi uporabnikove poteze z miško. V spletnih aplikacijah iz vhodnih naprav pridobivamo položaj

kazalca približno 60-krat v sekundi oz. manjkrat ob slabi odzivnosti aplikacije. Razdalja med dvema zaporednima pozicijama kazalca torej ni nikoli enaka, ker je odvisna od hitrosti premikanja kazalca po zaslonu in obremenitve spletne strani, zato je potrebno krivuljo gibanja rekonstruirati, da lahko izrišemo točke, ki so med sabo oddaljene za enak razmik.



Slika 5.5: Postopek računanja kontrolnih točk s pomočjo vektorjev. V prvem koraku se izračuna smerni vektor med vozliščem 3 in 1. V drugem koraku se vektor normalizira in se pomnoži s 30% vektorja razdalje med vozliščema 1 in 2 zato, da dobimo kontrolno točko. V tretjem koraku izračunamo kontrolno točko iz smernega vektorja od vozlišča 1 do izračunane kontrolne točke, zato da dobimo prvo kontrolno točko. V četrtem koraku izračunamo še kontrolno točko, ki bo služila za izdelavo Bézierjeve krivulje med točko 2 in 3. V petem koraku imamo vse podatke za izračun Bézierove krivulje med vozliščema 1 in 2.

Krivulja gibanja se rekonstruira z Bézierovimi krivuljami. Na podlagi vhodnih točk miškinega kazalca se naredi več Bézierovih krivulj drugega reda, ki skupaj tvorijo pot (angl. Bézier path), iz katere lahko določimo točke, ki so med sabo približno enako oddaljene. Te točke nato uporabimo za risanje posameznih tekstur čopiča. Pri izdelavi Bézierovih krivulj moramo seveda upoštevati to, da za majhne razdalje ne generiramo krivulj, temveč uporabimo kar linearno interpolacijo. Krivulje generiramo sproti z

interpolatorjem, ki kot vhod sprejema neenakomerno oddaljene točke, kot izhod pa zagotavlja točke z enakomerno oddaljenostjo in mehko interpolacijo. Bézierove krivulje generiramo tako, da ohranjamo zveznost krivulje C^1 , kar pomeni, da mora obstajati daljica, ki gre skozi vozlišče in kontrolno točko pred njo ter kontrolno točko za njo (slika 5.4). Algoritem za rekonstrukcijo sproti generira Bézierove krivulje tretjega reda, kjer vhodne točke nastopajo kot vozlišča krivulj, kontrolne točke pa se izračunajo iz smeri krivulje, kot je to opisano v viru [21] in prikazano na sliki 5.5.

V primeru, da imamo samo eno točko, interpolacije med točkami ne moremo izvajati. Če imamo dve točki, lahko izvedemo linearno interpolacijo in v primeru treh ali večih točk lahko izdelamo Bézierove krivulje, s katerih lahko določimo pozicije točk med dvema vozliščema. Bézierove krivulje torej uporabimo le takrat, ko imamo več kot dve točki, pri tem nam zadnja točka vedno služi za generiranje kontrolnih točk (slika 5.4).

Ko izračunamo kontrolne točke Bézierove krivulje, lahko izračunamo točke na njej, ki so enako oddaljene med seboj. S pomočjo enačbe 5.1, lahko izračunamo točko na Bézierovi krivulji z vozlišči A in D, kontrolnima točkama B in C ter vrednostjo t na intervalu $[0,1]$, ki zaznamuje pozicijo končne točke med vozliščema.

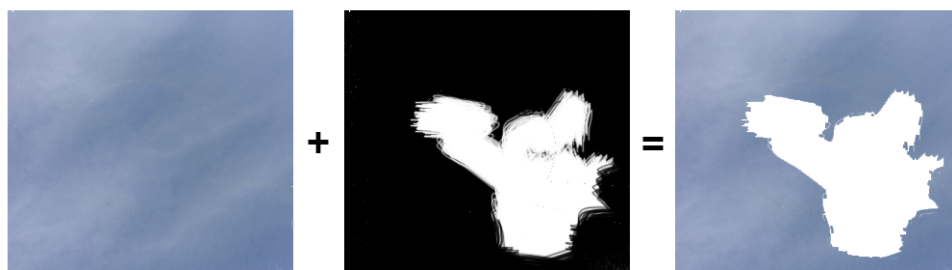
$$B(t) = A(1 - t)^3 + 3Bt(1 - t)^2 + 3Ct^2(1 - t) + Dt^3, t \in [0, 1] \quad (5.1)$$

Za izračun točk z enako oddaljenostjo na Bézierovi krivulji se sprehodimo po intervalu $[0,1]$ z določeno natančnostjo in merimo razdaljo med prejšnjo ter trenutno točko. Ob prekoračitvi razdalje naredimo linearno interpolacijo med prejšnjo in trenutno točko s faktorjem razdalje med točkama, iz česar nato izračunamo končno točko. Postopek ponavljamo vse dokler ne obidememo cele krivulje in izračunamo potrebne točke. Poleg Bézierove interpolacije med pozicijami točk interpolator čopiča poskrbi tudi za linearno interpolacijo med parametri čopiča, kot so prosojnost, pretok, barva, rotacija in skalacija.

Ob izračunu vseh parametrov in pozicij čopič izriše teksturo na izračunanih pozicijah s pripadajočimi parametri. Za izris tekstur skrbi poseben senčnik, ki upošteva nastavitve čopiča in izriše teksturo čopiča na izračunanih pozicijah na začasno plast. Začasna plast čopiča, ki prikazuje končen izris, je vedno vidna uporabniku, tako da lahko ta sproti vidi potezo čopiča na koncu poteze pa se začasna plast združi s trenutno aktivno plastjo in poteza čopiča je zaključena.

5.3.5 Radirka

Radirka deluje podobno kot čopič, vendar ne dodaja barve, temveč jo odstranjuje. Učinek radirke je dosežen z risanjem čopiča na ločeno površino, pri čemer rišemo samo črno-belo masko, ki jo uporabimo za to, da vemo, kateri del originalne plasti odstraniti (slika 5.6). Radirka tako kot čopič uporablja interpolator in nastavitve in teksturo čopiča za to, da izriše potrebno masko. Ob potezi radirke je vedno vidnen njen končni rezultat, saj orodje izbrano plast vedno pred prikazom izriše tako, da so videni samo nedotaknjeni deli. Končni rezultat radirke izrisuje poseben senčnik, ki združi izrisano masko s površino, ki jo radiramo. Ko je poteza narejena, orodje shrani spremembe na trenutno izbrano plast in akcija je zaključena.



Slika 5.6: Delovanje orodja radirka. Radirka izrisuje črno-belo masko s teksturo čopiča na ločeno površino, v kateri črna barva zaznamuje dele slike, ki bodo ostali nedotaknjeni.

5.3.6 Učinki

Učinki (angl. Effects) so orodje, ki omogoča spreminjanje trenutno izbrane plasti s pomočjo različnih učinkov. Orodje vsebuje svoj panel (poglavje 5.4.7), ki uporabniku ponuja na izbiro več učinkov. Uporabnik lahko učinek izbere in prikažejo se dodatne kontrole, s katerimi lahko nadzira lastnosti ter z njimi intenzivnost učinkov. Vsak učinek je edinstven in vsebuje svoj senčnik, ki preoblikuje izbrano plast. Ob izbiri učinka orodje trenutno plast shrani na eno samo teksturo zato, da je procesiranje enostavnejše za konvolucijske učinke. Tekstura se nato uporabi v senčniku, da se izriše učinek, ki je nato prikazan uporabniku. Orodje omogoča pregled in nadziranje učinkov posameznih plasti preko kontrol v realnem času. Na izbiro so naslednji učinki, njihov rezultat pa je prikazan tudi v tabeli 5.1:

Svetlost in kontrast (angl. **Brightness and Contrast**) omogoča spreminjanje sve-

tlosti in kontrasta slike. S svetlostjo nadziramo, kako svetla oz. temna je slika, s kontrastom pa lahko spremenimo razpon med svetlimi in temnimi barvami. Svetlost barve spremenimo tako, da vsaki komponenti barve dodamo določeno konstantno vrednost, kontrast pa spremenimo tako, da vrednosti barve zmanjšamo ali zvečamo glede na to, ali je barva svetla ali temna.

Barva in nasičenost (angl. Hue and Saturation) omogoča spreminjanje odtenka barve slike in nasičenosti (živost) barv. Učinek dosežemo tako, da barvo vsake slikovne pike pretvorimo v barvni format HSL. V formatu HSL spremenimo vrednosti odtenka in nasičenosti, nato pa ga pretvorimo nazaj v format RGB za prikaz.

Izravnava barv (angl. Color Correction) omogoča spreminjanje rdeče, modre in zelene barve v sliki. S pomočjo tega učinka lahko zamaknemo posamezno barvno komponento za določeno vrednost.

Sepija (angl. Sepia) sliko postara z (rdečkastim) odtenkom. Učinek pretvarja barve po enačbi 5.2, v kateri posamezne komponente vhodne barve lahko prepoznamo preko spremenljivk sR , sG in sB . Spremenljivke dR , dG in dB pa so komponente končne barve.

$$\begin{aligned}dR &= (sR * 0.393) + (sG * 0.769) + (sB * 0.189) \\dG &= (sR * 0.349) + (sG * 0.686) + (sB * 0.168) \\dB &= (sR * 0.272) + (sG * 0.534) + (sB * 0.131)\end{aligned}\tag{5.2}$$

Šum (angl. Noise) doda šum v sliko. Šum dodamo tako, da vsakemu pikslu dodamo določeno naključno vrednost. Količino šuma pa nadziramo z razponom naključnega števila.

Zameglitev (angl. Blur) zamegli sliko. Učinek dosežemo s pomočjo konvolucije, tako da za vsako slikovno piko izračunamo povprečje slikovnih pik, ki jo obdajajo. Koliko slikovnih pik povprečimo, pa je odvisno od tega, kako zamegljeno sliko želimo.

Izostritev (angl. Sharpen) izostril sliko. Učinek je dosežen s pomočjo konvolucije, pri kateri uporabimo jedro 3×3 . Sredinsko slikovno piko pomnožimo z 8, ostale pa z -1, nato pa dobljen rezultat dodamo prejšnji slikovni piki.

Mediana (angl. Median) odstrani šum iz slike. Postopek izračuna mediane deluje po principu drsečega okna, pri katerem za vsako slikovno piko izračunamo mediano iz sosednjih slikovnih pik.

Odkrivanje robov (angl. Edge Detection) detektira robove v sliki in jih prikaže v črno-beli tehniki. Robove se detektira s pomočjo Sobelovega filtra, pri katerem se po konvoluciji izračunajo vertikalni in horizontalni robovi, ki se jih nato združi skupaj.

Prag (angl. Treshold) spremeni vse barve pod določeno mejo v črno barvo in ostale v belo. Učinek deluje tako, da primerja povprečje barve piksla z mejo in v primeru, da je manjše, se piksel obarva s črno barvo, v nasprotnem primeru pa z belo.

Inverz (angl. Invert) obrne vrednost barv. Vrednosti barv obrnemo tako, da vsako barvno komponento odštejemo od maksimalne vrednosti.









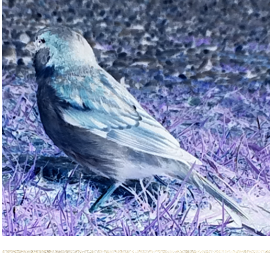
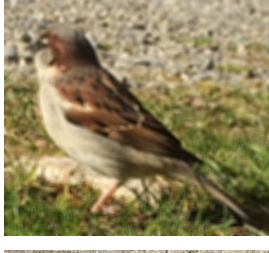


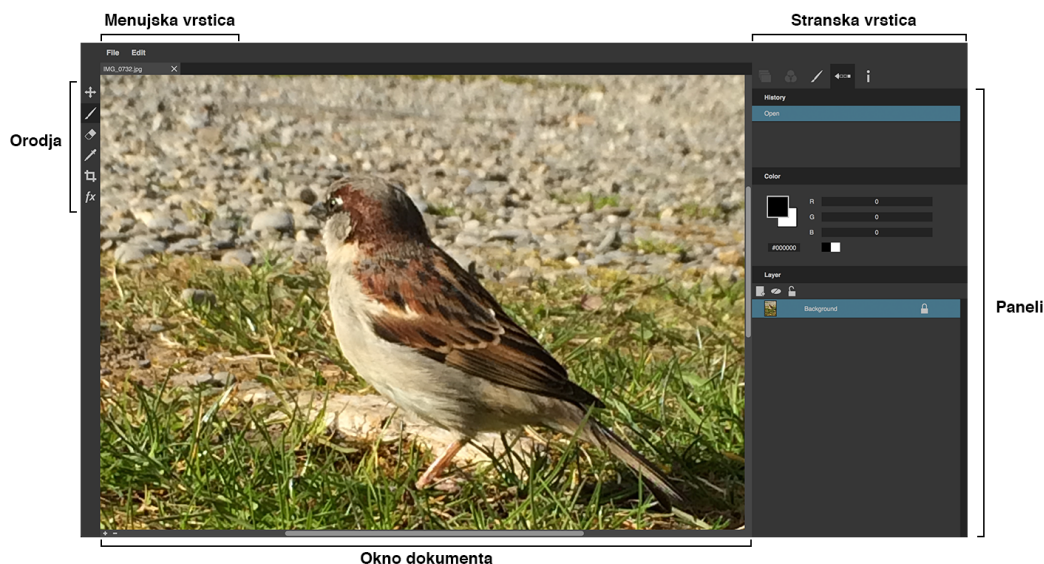
originalna slika		mediana	
svetlost in kontrast		sepija	
barva in nasičenost		črno belo	
izravnava barv		odkrivanje robov	
inverz		zamegljitev	
šum		izostritev	

Tabela 5.1: Slikovni prikaz učinkov.

5.4 Uporabniški vmesnik



Slika 5.7: Uporabniški vmesnik aplikacije.

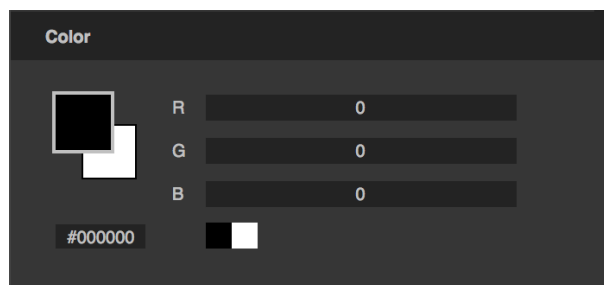
Uporabniški vmesnik aplikacije je sestavljen iz štirih večjih delov (slika 5.7). Na zgornjem delu je vrstica z menuji, v kateri se nahajajo izvlečni menuji za tipične funkcionalnosti vsakega programa, kot je odpiranje in shranjevanje dokumentov, razveljavljanje in ponavljanje sprememb itd. Na sredini je prikazan trenutni dokument in vsi ostali dokumenti v zavihkih. Na levi strani se nahajajo vsa orodja, ki so na razpolago, na desni strani pa se nahaja stranska vrstica s paneli, ki služijo prikazu podatkov, ter urejanju nastavitev orodij in dokumenta. Desna stranica vsebuje več panelov tiste, ki jih je možno prikazati, so prikazani, ostali pa so skriti in so dostopni z zavihkov na vrhu. Vsako orodje definira svojo paletu panelov, ki so smiselni pri uporabi orodja. Bolj pomembne panele skuša stranica ohraniti vidne, ostali pa so skriti in se prikažejo v primeru, da izberemo zavihkek panela na vrhu. Na vrhu stranske vrstice so torej zavihki vseh panelov, ki jih lahko uporabljamo pri izbranem orodju. Tisti, ki so zatemnjeni, so že prikazani in jih ne moremo izbrati. Izbrani zavihkek je vedno tisti, ki je najbolj na vrhu oz. tisti, ki ga je uporabnik izbral za vidnega. V primeru, da izberemo en zavihkek, bo ta vedno viden, poleg njega pa bo stranska vrstica zapolnila preostali prostor z ostalimi paneli, če bo to mogoče.

Vsak del uporabniškega vmesnika je tako zaključena celota in skrbi za predstavitev in manipulacijo podatkov v aplikaciji. Uporabniški vmesnik je narejen dinamično z jezikom JavaScript, kjer za vsak del uporabniškega vmesnika skrbi svoj nadzornik, ki

kontrolira spremembe gradnikov in te posodablja v primeru, da se podatki v aplikaciji spremenijo. Uporabniški vmesnik deluje po principu model-pogled-nadzornik. Za pogled se uporabljajo standardni elementi HTML, s katerimi upravlja nadzornik, podatke za prikaz pa pridobi od aplikacije. Za vizualno podobo in stil elementov dosledno skrbijo stilske predloge CSS.

Stranska vrstica vsebuje vrsto panelov za prikaz in urejanje podatkov, ki so na kratko predstavljeni v naslednjih podpoglavjih.

5.4.1 Panel barve



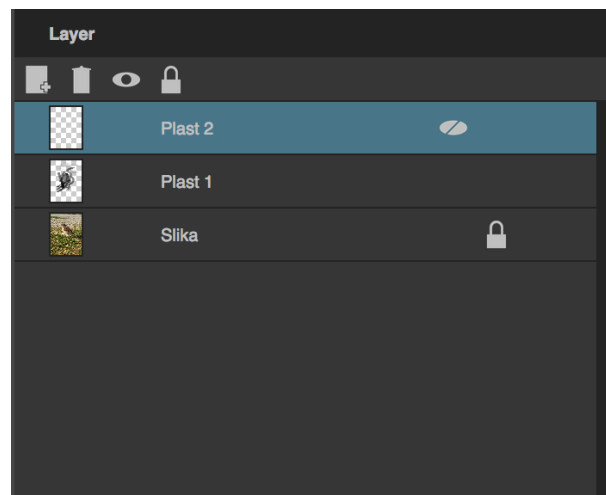
Slika 5.8: Uporabniški vmesnik panela za barvo.

Panel barve (slika 5.8) omogoča uporabniku nastavitve primarne in sekundarne barve, ki ju potem izkoriščajo različna orodja. Panel ponuja uporabniku tri drsnike, s katerimi lahko nadzira rdečo, zeleno in modro komponento izbrane barve. Če uporabnik potrebuje več kontrole nad barvo, lahko dvoklikne na barvo in prikazalo se bo sistemsko okno za izbiro barve. V spodnjem delu lahko odčitamo šestnajstiško vrednost barve ali pa jo nastavimo. Panel prikazuje tudi zgodovino uporabljenih barv v spodnjem desnem kotu, kjer lahko ponovno izberemo barvo, ki smo jo že uporabljali.

5.4.2 Panel plasti

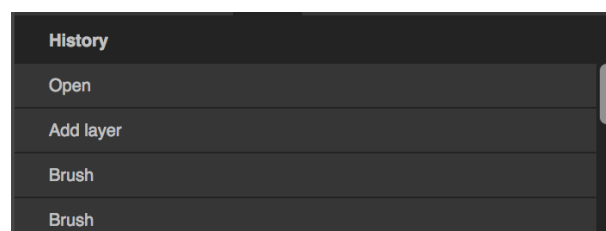
Panel plasti (slika 5.9) služi za predstavitev in upravljanje s plastmi v dokumentu. Plasti so nanizane od zgornje do spodnje plasti v istem vrstnem redu, kot so prikazane. Posamezno plast je mogoče prestaviti z vlečenjem, pri čemer spremenimo vrstni red izrisa plasti v dokumentu. Gornji del panela vsebuje vrstico z gumbi za dodajanje, odstranjevanje, skrivanje in zaklepanje plasti. Gumbi za spreminjanje lastnosti plasti upravljajo s trenutno izbrano plastjo, ki je označena z modro barvo. Vsak gradnik plasti vsebuje manjšo sliko, ki vizualno predstavlja vsebino plasti, njeno ime in status,

ki pove, ali je plast nevidna oz. zaklenjena. Nevidnih plasti uporabnik ne vidi v dokumentu, temveč samo v panelu, zaklenjenih plasti pa ni možno urejati. Ime plasti je mogoče spremeniti z dvoklikom na ime plasti.



Slika 5.9: Uporabniški vmesnik panela plasti.

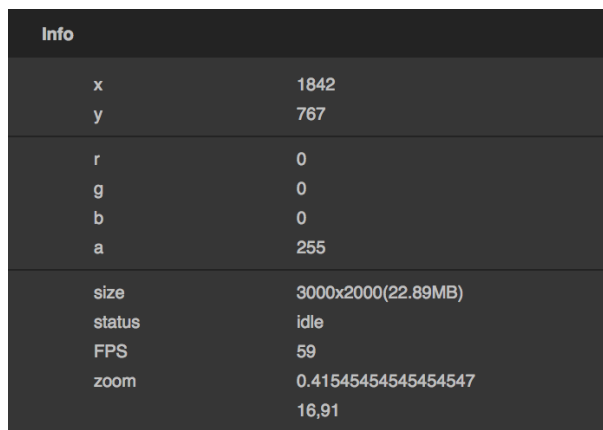
5.4.3 Panel zgodovine



Slika 5.10: Uporabniški vmesnik panela za zgodovino.

Panel zgodovine (slika 5.10) je namenjen pregledu zgodovine akcij, ki so bile narejene v določenem dokumentu. S klikanjem po preteklih akcijah se lahko premaknemo naprej in nazaj po zgodovini.

5.4.4 Panel informacij

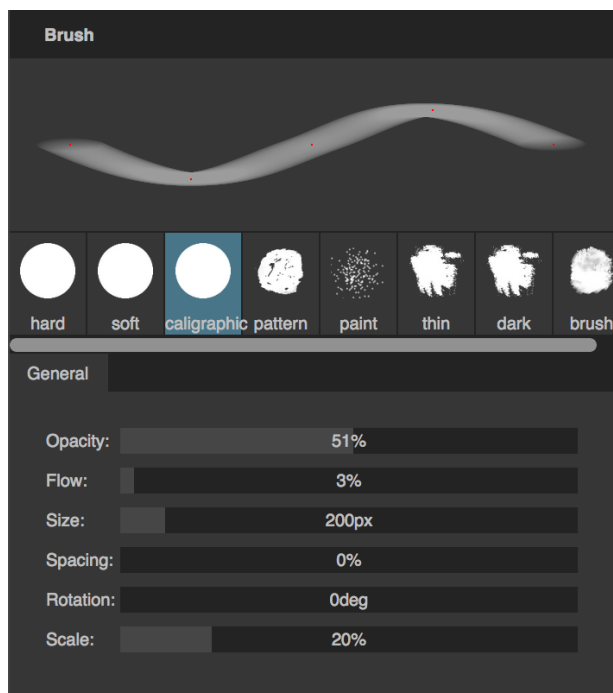


Info	
x	1842
y	767
r	0
g	0
b	0
a	255
size	3000x2000(22.89MB)
status	idle
FPS	59
zoom	0.41545454545454547 16,91

Slika 5.11: Uporabniški vmesnik panela za informacije.

Panel informacij (slika 5.11) prikazuje podatke, kot so trenutna pozicija kazalca, barva pod kazalcem, velikost dokumenta in trenutno povečavo.

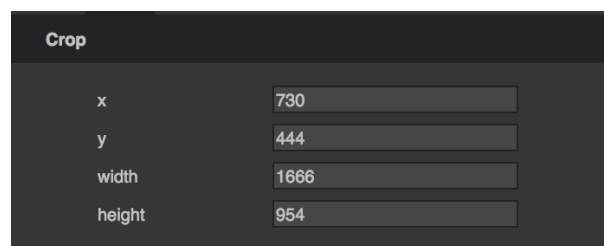
5.4.5 Panel čopičev in radirke



Slika 5.12: Uporabniški vmesnik panela čopičev.

Panel čopičev (slika 5.12) je namenjen spreminjanju nastavitev in lastnosti čopičev. Na vrhu panela je izrisana poteza čopiča, ki se spreminja ob spreminjanju nastavitev čopiča. Pod njo se nahajajo teksture čopičev, med katerimi lahko izbiramo, in drsniki za nadziranje lastnosti izbranega čopiča. Panel radirke je po videzu enak panelu čopičev, s tem da ima drugačno ime in nadzira lastnosti radirke, ne čopiča.

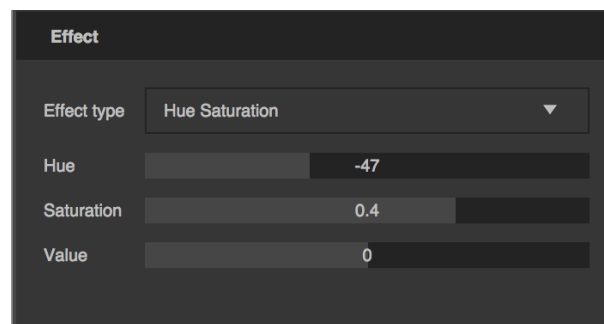
5.4.6 Panel orodja za obrezovanje



Slika 5.13: Uporabniški vmesnik panela orodja za obrezovanje.

Panel orodja za obrezovanje (slika 5.13) prikazuje informacije o velikosti in položaju obrezane površine, ki jih lahko spreminjamo, ko je aktivno orodje za obrezovanje.

5.4.7 Panel učinkov



Slika 5.14: Uporabniški vmesnik panela učinkov.

Panel učinkov (slika 5.14) ponuja preko izvlečnega seznama paletu učinkov, preko katerih lahko spreminjamo lastnosti učinkov. Vsak učinek ima svoje lastnosti, tako da se število in vrednosti drsnikov spreminjajo glede na izbran učinek.

Poglavje 6

Sklepne ugotovitve

V okviru diplomskega dela smo uspešno implementirali aplikacijo za urejanje slik na spletu s tehnologijo WebGL. Čeprav se tehnologija večinoma uporablja pri izdelavi iger, je bila implementacija aplikacije uspešna in performančno zadovoljiva. Aplikacija za hitro delovanje sicer potrebuje soliden procesor in grafično kartico, ki pa sta dandanes že zahteva vsake tovrstne namizne aplikacije. Poleg strojne opreme aplikacija za delovanje potrebuje brskalnik, ki podpira tehnologijo WebGL, katere po statistikah [32] uporablja približno 75 % uporabnikov popularnih spletnih mest.

Aplikacija zaradi tehnologije in arhitekture deluje drugače kot ostali urejevalniki slik na spletu, kot so DeviantArtMuro [35], Sketchpad [33], in Pixlr Editor [34]. Pri primerjavi hitrosti izrisovanja velikih čopičev in izrisovanja učinkov je implementacija v izdelani aplikaciji nedvomno hitrejša kot v ostalih.

Ugotovili smo torej, da nam WebGL ponuja hitrejšo izrisovanje in boljši nadzor nad končnim rezultatom. S pomočjo senčnikov lahko rišemo kompleksne izriske, ki bi bili drugače performančno preveč zahtevni za implementacijo z drugimi programskimi vmesniki, kot so Canvas 2D, ki ga uporablja večina ostalih aplikacij. Canvas 2D za razliko od aplikacij WebGL izvaja glajenje (angl. Antialiasing) na izrisani sliki, kar pomeni, da bi pobližani elementi slike izgledali megleno in slikovnih pik ne bi uspeli ločiti med sabo. Problem bi seveda lahko rešili s tvegano metodo, pri kateri bi lahko izrisali kvadrate za vse vidne slikovne pike, kar pa bi najverjetneje povzročilo performančne probleme in kasneje otežilo implementacijo drugih delov aplikacije.

Seveda ima tudi WebGL svoje slabosti, kot so daljši razvoj zaradi kompleksnega programskega vmesnika, težko razhroščevanje in omejitve grafičnih kartic. Pri razvoju so bile določene omejitve upoštevane, vendar na žalost ne vse. Tehnologija namreč ne omogoča zaznavanja porabe grafičnega pomnilnika, kar lahko pripelje do preneha-

nja delovanja aplikacije v primeru, da uporabljamo ves pomnilnik za elemente, ki jih potrebujemo.

Aplikacija ni bila optimizirana za vse možne situacije, za kar bi bilo potrebno širše testiranje na večih sistemih in brskalnikih za zagotavljanje produkcijske kakovosti. V knjižnici, ki je namenjena risanju, prav tako obstaja prostor za izboljšavo performančnosti, saj so določeni deli neoptimizirani, kar se pozna pri hitrih potezah čopičev.

Izdelana aplikacija torej prikazuje uspešno uporabo programskega vmesnika WebGL v spletnih aplikacijah. Na voljo sicer niso vse funkcionalnosti, ki so možne v ostalih aplikacijah, saj bi bila implementacija vseh časovno zahtevna, vsebuje pa dobro zasnovano arhitekturo in popolnoma delujočo aplikacijo, katere izboljšave so predmet prihodnjih verzij.

Literatura

- [1] Tomas Akenine-Moller Eric Haines Naty Hoffman, “Real-Time Rendering Third Edition”, 2008, str. 11–25.
- [2] David Flanagan, “JavaScript: The Definitive Guide, 6th Edition”, 2011, str. 445–464.
- [3] (2004) Erika Jansson, Brush painting algorithms.
Dostopno na:
<http://www.cse.chalmers.se/~uffe/xjobb/BrushPaintingAlgorithms.pdf>.
- [4] (2014) Tehnike za optimizacijo prenosa podatkov na grafično kartico.
Dostopno na:
https://developer.apple.com/library/ios/documentation/3ddrawing/conceptual/opengles_programmingguide/TechniquesforWorkingwithVertexData/TechniquesforWorkingwithVertexData.html
- [5] (2014) Spletna stran s podatki in novicami o filmih.
Dostopno na:
<http://www.imdb.com>
- [6] (2014) Spletna trgovina.
Dostopno na:
<http://www.amazon.com>
- [7] (2014) Spletna trgovina.
Dostopno na:
<http://www.ebay.com>

- [8] (2014) Spletni iskalnik.
Dostopno na:
<http://www.google.com>
- [9] (2014) Spletna stran za plačevanje preko spleta.
Dostopno na:
<http://www.paypal.com>
- [10] (2014) Spletna aplikacija za hranjenje in urejanje dokumentov in datotek.
Dostopno na:
<https://drive.google.com>
- [11] (2014) Spletna aplikacija za pregled zemljevidov in krajev.
Dostopno na:
<https://maps.google.com>
- [12] (2014) Spletna aplikacija za izdelavo 3D vsebin.
Dostopno na:
<https://clara.io>
- [13] (2014) Spletna stran za izmenjavo 3D modelov.
Dostopno na:
<https://sketchfab.com>
- [14] (2014) Spletna aplikacija za hranjenje in sinhronizacijo dokumentov.
Dostopno na:
<https://www.dropbox.com>
- [15] (2014) Spletna aplikacija za vodenje opravil in projektov.
Dostopno na:
<https://asana.com>
- [16] (2014) Spletna aplikacija za izdelavo interaktivnih prezentacij.
Dostopno na:
<http://prezi.com>
- [17] (2014) Specifikacije programskega vmesnika File API.
Dostopno na:
<http://dev.w3.org/2006/webapi/FileAPI/>

- [18] (2014) Razred SpriteBatch knjižnice XNA.
Dostopno na:
<http://msdn.microsoft.com/en-us/library/microsoft.xna.framework.graphics.spritebatch.aspx>
- [19] (2013) Razred SpriteBatch knjižnice LibGDX.
Dostopno na:
<http://libgdx.badlogicgames.com/nightlies/docs/api/com/badlogic/gdx/graphics/g2d/SpriteBatch.html>
- [20] (2014) Razred SpriteBatch knjižnice DirectX Tool Kit.
Dostopno na:
<https://directxtk.codeplex.com/SourceControl/latest#Src/SpriteBatch.cpp>
- [21] (2011) Algoritem za rekonstrukcijo Bézierovih poti iz vhodnih točk.
Dostopno na:
<http://devmag.org.za/2011/06/23/bzier-path-algorithms/>
- [22] (2014) Ogrodje Ruby on Rails za razvoj spletnih strani.
Dostopno na:
<http://rubyonrails.org/>
- [23] (2014) Ogrodje Middleman za razvoj in optimizacijo spletnih strani.
Dostopno na:
<http://middlemanapp.com/>
- [24] (2014) Specifikacije jezika HTML5.
Dostopno na:
<http://www.w3.org/TR/html5/>
- [25] (2011) Specifikacije označevalnega jezika CSS.
Dostopno na:
<http://www.w3.org/TR/CSS2/>
- [26] (2014) Jezika SASS.
Dostopno na:
<http://sass-lang.com/>

- [27] (2014) Specifikacije standarda ECMA iz katerega izhaja jezik JavaScript.
Dostopno na:
<http://www.ecma-international.org/ecma-262/5.1/>
- [28] (2014) Uradna stran sistema GIT za verzioniranje izvorne kode.
Dostopno na:
<http://git-scm.com/>
- [29] (2013) Specifikacije WebGL programskega vmesnika.
Dostopno na:
<http://www.khronos.org/registry/webgl/specs/1.0/>
- [30] (2009) Specifikacije jezika OpenGL Shading Language (GLSL).
Dostopno na:
http://www.khronos.org/files/opengles_shading_language.pdf
- [31] (2014) Specifikacije programskega vmesnika Canvas 2D.
Dostopno na:
<http://www.w3.org/TR/2014/CR-2dcontext-20140821/>
- [32] (2014) Statistika uporabe WebGL med uporabniki.
Dostopno na:
<http://webglstats.com/>
- [33] (2014) Spletni urejevalnik slik.
Dostopno na:
<https://sketch.io/sketchpad>
- [34] (2014) Spletni urejevalnik slik.
Dostopno na:
<http://apps.pixlr.com/editor/>
- [35] (2014) Spletni urejevalnik slik.
Dostopno na:
<http://muro.deviantart.com/>