TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Ilja Zolotnikov

Liikumisgraafika animeerimise tööprotsessi automatiseerimine programmis Adobe After Effects

Bakalaureusetöö (9 EAP)

Juhendaja: Lidia Feklistova

Tartu 2020

Liikumisgraafika animeerimise tööprotsessi automatiseerimine programmis Adobe After Effects

Lühikokkuvõte:

Käesoleva töö teoreetiline osa kirjeldab noort kunstivoolu nimega liikumisgraafika, selle ilmumise ajalugu, teose loomise protsessi ning meetodeid automatiseerimise jaoks. Töö praktilise osana valmis näidisprojekt, mille teostamisel on rakendatud töö teoreetilises osas uuritud automatiseerimismeetodeid. Neid kasutades on saavutatud suure ajalise võidu võrreldes uuritud meetodeid kasutamata jättes.

Võtmesõnad:

Liikumisgraafika, graafiline disain, automatiseerimine, arvutigraafika, animatsioon, skriptid

CERCS: P175 Informaatika

Automating the workflow of motion graphics animation in the program Adobe After Effects

Abstract:

The theoretical part of the following thesis describes the modern art flow known as motion graphics, its development history, the creation process of its pieces of art and methods of its automation. In the practical part of the thesis the sample project is made which uses the means of automation mentioned above, besides the time consumption of creating the art piece is greatly reduced compared to having the means disengaged.

Keywords:

Motion graphics, graphic design, automation, CGA, computer graphics, animation, scripting

CERCS: P175 Informatics

SISUKORD

SISSEJUHATUS

Kujundusgraafika (e graafiline disain, ingl *graphics design*) on kasvanud staatilisest kirjastuse paradigmast dünaamiliseks kommunikatsiooni tavaks nagu televisioon, animatsioon, multimeedium, veeb jne.

Krasneri järgi [1] paigalseisva kujundusgraafika ühitamine liikuvate elementidega tekitab hübriidset süsteemi suhtlemiseksvaatlejaga. Vaatlejaga sideme loomine läbi kujundusgraafika on filmitööstuse algusest olnud eraldiseisev nišš. Selle edasiarendamine (nt pealkirjade animeerimine filmide alguses või reklaamis) 1950-ndatel on loonud uue kujundusgraafika haru – liikumisgraafika disaini.

Nüüdisajal liikumisgraafika loomiseks on olemas palju erinevaid tööriistu, üks neist on Adobe'i poolt toodetud programmide pakett Adobe Creative Cloud. Paraku üheks suureks probleemiks liikumisgraafika kujundamisel on ajakulu, mis läheb teose loomiseks.

Kuna liikumisgraafika tihtipeale sisaldab nii animatsiooni, illustratsiooni, tüpograafiat kui ka videot, fotograafiat ja muusikat [2], siis nende osade kombineerimine üheks harmooniliseks teoseks (mis omakorda ka liigub ajas) võtab palju aega. Kõik tegevused nõuavad aega ja suure töö eraldi animeeritavate elementide kogus võib ületada mitutkümmet. Suur kogus elemente aga ei välista seda, et kõik elemendid on samalaadse liikumismustriga, seega nende elementide animeerimist võib automatiseerida ja lihtsamaks muuta.

Automatiseerimine kindlalt säästab kõige tähtsamat ressurssi tööprotsessi jooksul – aega. Säästetud aega omakorda võib kasutada teoselooja idee arendamiseks ja õigel ajal vigade märkamiseks, kui selline olukord peaks toimuma.

Lõputöö eesmärk on uurida ja võrrelda olemasolevaid võimalusi tööprotsessi automatiseerimiseks ning läbi praktilist lahendust kontrollida võimaluste ajalist kokkuhoidu liikumisgraafika animeerimisel programmis Adobe After Effects.

Lõputöö koosneb neljast osast. Esimeses osas uuritakse mis on liikumisgraafika, selle ilmumise ajalugu, vaadeldakse liikumisgraafika animeerimise põhimõtted. Teises osas vaadeldakse liikumisgraafika animeerimise põhimõtete teostus, tuuakse võrdlus kõige populaarsemate liikumisgraafikale suunatud programmidest, nende sarnasused ja erinevused programmist Adobe After Effects. Kolmandas osas analüüsitakse käsitletud programmi sisse ehitatud lahendusi töö voo automatiseerimiseks. Neljandas osas võetakse lahendused kasutusele näidisprojekti raames ning uuritakse ajalist kulu võrreldes automatiseerimise lahendusi kasutamata.

1. LIIKUMISGRAAFIKA DISAIN

Traditsioonilise filmimise ja ringhäälingu kõrval on nüüdisajal aina tihedamini kuulda terminit nagu liikumisgraafika disain (ingl *motion graphic design*). See on lai kujundusgraafika alamhulk, mis hõlmab tüpograafiat, illustratsiooni, animatsiooni, helikujunduse, arhitektuuri võtteid jne. Woolman [2] pöörab eraldi tähelepanu sõnaosale „graafika“: liikumisgraafika disaini peamised elemendid on kujud, jooned, kõverad, sümbolid, ikoonid, illustratsioonid. Graafilised kujundajad kasutavad liikumisgraafikat muljetavaldava disaini loomiseks nii filmitööstuses, televisioonis kui ka internetis. Liikumisgraafikat võib nimetada graafilise disaini ja animatsiooni sümbioosiks. Animatsioonist eristub liikumisgraafika sellepoolest, et see ei pea rangelt sisaldama stsenaariumi ega kindlat tegelast, vaid võib olla ka abstraktne. Liikumisgraafika suurim erinevust graafilisest disainist seisneb tarvilikus liikluse olemasolus.

1.1. Animatsiooni ajalugu ja liikumisgraafika alused

Liikumisgraafika ajalugu ei ole ühemõtteline, kuna selle mõiste piirid ei ole täpselt määratud [3]. Liikumisgraafika esimesed rakendused on tihedalt seotud filmitööstusega, selle ajalugu kohakuti samastub filmide ajalooga. Liikumisgraafika disaini mõiste defineerimiseks on kindlasti vaja välja tuua animatsiooni terminit, sest peamine aspekt liikumisgraafikas on ajas liikuvad kujundid.

Animatsioon (lad *anima* – elu, hing) on järjestatud staatiliste graafikakujunditega loodud liikumisillusioon [4]. Sujuva animatsiooni saavutamiseks üksteisele järgnevad kujundid peavad olema piisavalt väikese erinevusega.

Animatsiooni ei oleks võimeline tajuda, kui inimese silm ei omaks nägemise püsivust (ingl *persistence of vision*). See on optiline illusioon, mille tagajärjel silma saabuvate valgusekiirete tajumine ei kao koheselt ära, vaid jääb veel mõneks hetkeks meie ajus salvestatuna [5], [6]. Antud illusiooni pärast meie aju tajub vahetuvaid pilte ühe sujuva pildina.

Inimkonna esimestest sammudest on proovitud edastada liikumist läbi kunsti. Esimesed näited ulatuvad tagasi kiviaega (Lascaux ja Altamira koopamaalingud, kus kujutatud loomad on jadana joonistatud liikumise edastamiseks). Zorich’u [7] sõnadel kasutasid inimesed peamiselt lõkkevalgust animatsiooni illusoorseks tekitamiseks. Gatton [8] kirjutab, et optiline fenomen, mille jooksul väikse ava ees olev valgustatud ese on projekteeritud ava vastasseinale ümberpööratud kujul, nimega *camera obscura* (e pimekamber) [9], on mõjutanud juba neoliitikumi ehitisi. Samal põhimõttel põhinev ja samanimeline aparaat on tänapäeva kaamerate eelkäija. Renessanssi leiutis maagiline latern (lad *lanterna magica* [10], [11]

projektori eelkäija), mis on pimekambri edasiarendamine, oli laial kasutusel kuni kahekümnenda sajandini. Maagilise laterna või sellesse sisse käivate slaidide liigutamine võimaldas tekitada illusoorset liikumist, mis oli esimene samm tänapäevase liikuva pildi ja liikumisgraafika poole. Tähelepanuväärsete leiutiste hulka peetakse ka uusajal populaarseid *phénakistiscope'*i ja pilditrumlit (ingl *zoetrope*) [11], mille tööprintsiip põhineb vastavalt ketta või silindri kiiresti pöörlemisel animatsiooni tekkimiseks.

Kõik need leiutised on mänginud suure rolli tavapäeva filmi ja animatsiooni tekitamisel ning on olnud alus filmi ilmumisele.

1.2. Filmitööstus ja liikumisgraafika ilmumine

1895. aastal Lumière'i vendade poolt loodud kinematograafi turule tulemisega ilmus võimalus näidata liikuvat pilti ja lugusid suurele publikule. Sellest ajast algas animatsiooni- ja filmiarendamine sellel kujul, millel me teame seda tänapäeval. Krasneri sõnul [1], kuna filmi ilmumine ja järsk populariseerimine toimus just kahekümnenda sajandi alguses, industrialiseerimise, kultuuri ning sotsiaalsete aspektide nihke pärast, filmide loojad püüdsid keelduda klassikalistest „argipäevastest" võtetest, mis omakorda tõi filmikunstimaailma ohtralt abstraktsioone ja uusi ideid. Futurism, kubism, sürrealism ja teised modernsed kunstivoolud on mõjutanud mitte ainult maali ja skulptuuri üldkokkuvõttes, vaid on ka mõjutanud filmi. Katsetades pöörata tähelepanu liikumisele, vormile ja rütmile, filmitegijad on loonud uueavangardismi filmivoolu – *cinéma pur*'i (e puhas filmikunst) [12]. Krasner [1] kirjutab, et selle filmivoolu teosed sisaldavad nii erinevaid kujusid, kõveraid, joone kui ka koostööd rütmi ja heliga – kõike, mis on liikumisgraafika peamised elemendid. Kõik need elemendid leidsid aset ka filmi tiitrites: tummfilmides tiitrid aitasid vaatajatel jutustusest aru saada ja tihtipeale nende käekiri samastus filmi temaatikaga, edasi aga tiitrid hakkasid mängima suuremat rolli terve teose terviklikkuse formeerimisel. *Cinéma pur*'is kasutatud meetodid on kolinud ka ringhäälingusse, kuna televisioon oli uus keskkond animatsiooni edastamiseks.

Algse liikumisgraafika elemendid hakkasid ilmuma tänu graafilise disaineri Saul Bassile 1950. aastate jooksul. Krasner märkab [1], et Bassi innovaatilised ideed filmi tiitrite tegemisel on vaadeldud eraldi teostena. Just tema on uuesti määranud tiitrite tähendust ja ideed filmides ning soodustas liikumisgraafika arengu selleks, millena me teame liikumisgraafikat tänapäeval [13]. Arvutite arvutuskiiruse arendamise tõttu, erinevate efektide ja raskete kompositsioonide kokkupanek sai palju kiiremaks ja kergesti saavutatavaks, soodustades liikumisgraafika arengu. Ise termin "liikumisgraafika" on ilmunud 20. sajandi teisel pool ning tänapäeval on laiali kasutuses.

1.3. Liikumisgraafika tähendus 21. sajandil

„Liikumisgraafika disaini tööstus on imelik.“ [14]. Paljud liikumisgraafika disainerid hea töökvaliteedi jaoks peavad mõistma kohe mitut kunstiala. Kindlasti liikumisgraafikat võib nimetada noorte spetsialistide valdkonnaks, kuna The School of Motion'i [14] läbiviidud uuringus selgus, et peaaegu kolm neljandikku liikumisgraafika disaineritest on 35 aastat vana või nooremad. Peagi 80% disaineritest on tegelenud selles valdkonnas vähem kui 10 aastat. Uuringu autor arvab, et liikumisgraafika on paljude jaoks hobiks või kõrvaltööks, mida tehakse ainult projektide ja tellimuste olemasolul.

Tänapäeval liikumisgraafika rakendus on juba üpris lai, ning ilmub veel uusi kasutusvaldkondi. Nii nagu ennem, seda kasutatakse filmi ja seriaalide tiitrite tegemisel, kuigi nüüd seda võetakse kasutusse näiteks ka muusikavideotes, rahvusringhäälingu saadetes info edastamiseks, reklaamis ning võistlustes edetabeli ja koondstiili tekitamiseks ja kunstina. Liikumisgraafika disaini vormid on kasutatud ka veebilehekülgede dünaamilises sisus ja elektroonsete seadmete kasutajaliidestes.

Üleüldiselt võib öelda, et tänapäeva liikumisgraafika interpretatsioonid moodustavad suure osa digi- ja meediamaailmas ning on see element, mis teeb staatilisest pildist suure ahvatleva teose ning edastab sõnumit vaatajatele.

1.4. Liikumisgraafika animeerimise põhimõtted

Liikumisgraafika alused samastuvad klassikalise animatsiooni ja filmi põhimõtetega, kuna see kunstiliik on sündinud samast kunstivoolust.

Nägemise püsivuse illusiooni pärast kiiresti vahetuvad pildid tekitavad liikumist. Selle illusiooni tekitamiseks on vaja vähemalt 10-12 [15] erinevat pilti sekundis, kuigi sellise sagedusega pilt ei tundu sujuv, seega kasutatakse suurema sagedusega liikuvaid pilte. Maailma filmistandard on 24 kaadrit sekundis [16], kuigi liikumisgraafika jaoks kasutatakse vähemalt 30. Tihtipeale veelgi sujuvama liikumisillusiooni tekitamiseks ja hägususe vähendamiseks kasutatakse ka kaadrisagedusena 60, mis on sama kui uuendussagedus nutiseadmete ekraanidel [17].

Sõltumata teose liigist, lõpliku tulemuse saadakse kombineerimise (kombineeritud võtted, ingl *compositing*), järjestamise (monteerimine, ingl *sequencing*) ning kogu kompositsiooni visualiseerimise (ingl *rendering*) teel.

Kombineerimine koosneb mitme erineva visuaalse elemendi kokku panemisest ühtlasesse ruumi. Selle protsessi tagajärjeks on reaalses maailmas võimatute suhete loomine ühes

kompositsioonis [1]. Tavaliselt vaadeldakse kõik elemendid eraldi kihtidena. Kõik need elemendid võivad olla erinevat tüüpi: pildid, jooned, kõverad, sümbolid, tekstikujud [2] ja erinevat dimensiooni – liikumisgraafika ei piirdu ainult kahe dimensiooniga, vaid võib vabalt sisaldada kolmandas dimensioonis objekte. Kombineerimine ise sisaldab endas ka muid protsesse, mille seas on maskimine (ingl *masking*), video miksimine või värvi võti (ingl *chroma-keying*), pildi jälgimine (ingl *image tracking*), pildi stabiliseerimine (ingl *stabilisation*), ajalised muutused (ingl *temporal manipulation*), gamma parandus (ingl *gamma correction*), liikumise hägususe lisamine (ingl *motion blur*), erinevate efektide rakendamine jne, kuigi nende protsesside kasutamist selles töös ei uurita.

Kombineeritud kaader ise ei oma mõttelist koormust, selle tähtsuse lisamiseks on kasutatud järgmist sammu animeerimisel - järjestamist. Järjestamise jooksul kõik kompositsioonis olevad kihid reastatakse ajateljel nii, et anda lõpptulemuseks soovitud mõtet edasi, lisada rütmi ning tempot. Seda saavutatakse kas tavalise terava lõikega erinevate stseenide vahel või kasutades üleminekuid. Kõige tavalisemad üleminekud on lahustamine (ingl *dissolve*), tuhmumine (ingl *fade*) ja pühkimine (ingl *wipe*) [1]. Tänu nendele saab luua loogilist üleminekut ühest mõttest teisele ning muuta üleüldise kompositsiooni sõnumi arusaadavamaks.

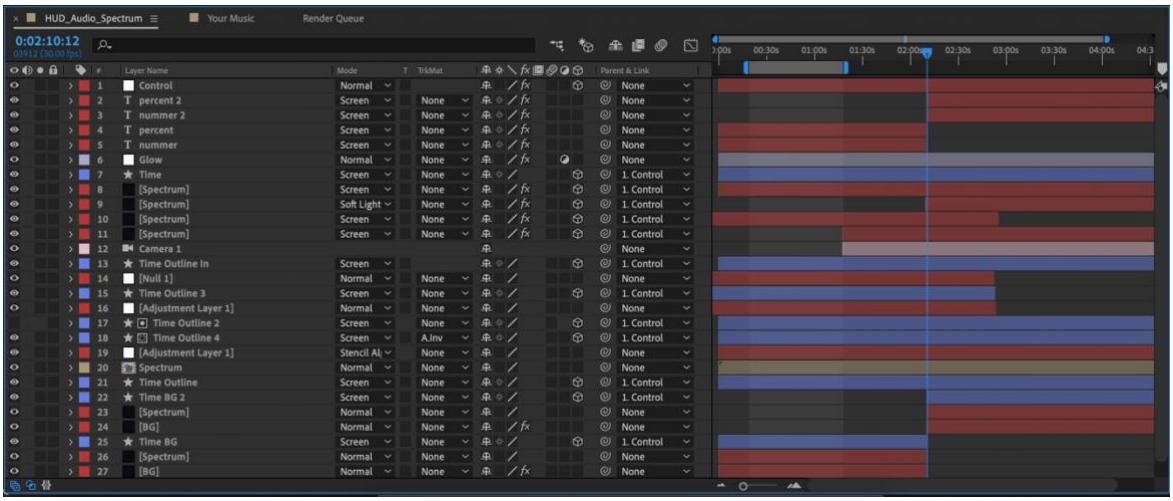
Kombineerimise ja järjestamise jooksul saavutatud tulemused ei ole arvutiga koheselt arvutatud eraldi failisse, vaid jäävad töötlemata käsklustena töötlemise programmis. Kuna need käsklused ei ole kiiresti arvutatavad, on kindlasti vaja, et arvutis ilma selle programmita oleks võimeline seda kompositsiooni kiiresti näha. Seda saavutatakse visualiseerimise jooksul, kus valitakse vajalik faililaiend, kodek, bitikiirus, kompositsiooni efektide kvaliteet ning lõpuks fail, kuhu seda kõike salvestatakse. Kui kompositsioon on salvestatud eraldi faili, seda saab lihtsalt ja kiiresti vaadata ning saata laiali kas kliendile või otse eetrisse, Internetti, telefonisse. Kuna on eeldatud, et enam kompositsioonis ei ole vaja midagi parandada ja kompositsioon on lõplik, siis salvestatud faili ei saa enam kuidagi muuta.

2. LIIKUMISGRAAFIKA ANIMEERIMINE

Selles osas uuritakse peamisi elemente animeerimise protsessis, mis tugineb eespool kirjeldatud põhimõtetel. Selle osa lõpus võrreldakse erinevate liikumisgraafika animeerimisele suunatud programmide võimekust, funktsionaalsust ja kasutust Adobe After Effects'i programmiga.

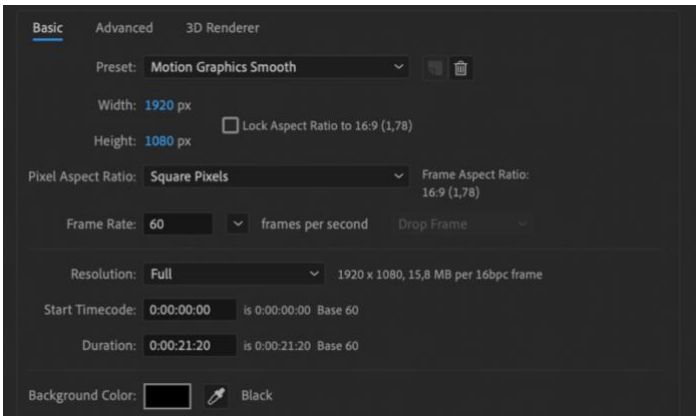
2.1. Kompositsioon ja selle elemendid

Kombineerimise jooksul põhiüksus on kompositsioon – konteiner, mis koosneb erinevatest kihtidest oma ajateljega (vt Joonis 1). Kompositsiooni võib vaadelda iseseisva projektina, kuna üks liikumisgraafika teos võib vabalt sisaldada ainult ühe kompositsiooni. Kompositsioon võib koosneda teistest kompositsioonidest ka, kuigi mitte rekursiivselt. Kihtide koguse maksimumi ei ole täpselt määratud, see sõltub kompositsiooni raskusest, programmist ja arvuti arvutusvõimekusest.



Joonis 1. Näide kompositsioonist programmis Adobe After Effects

Kompositsiooni peamised parameetrid on kaadrisagedus (ingl *frame rate*), lahutusvõime (ingl *resolution*) ja kestus (ingl *duration*) (vt Joonis 2).



Joonis 2. Kompositsiooni sätted programmis Adobe After Effects

Erinevate kihtide lisamine ei ole iseenesest raske tegevus, kuigi staatiline pilt ei anna vaatajale nii palju infot ega väljenda looja mõtet tervikuna. Selleks, et piltlik lugu arendada ning pöörata teose põhielementidele tähelepanu, neid elemente animeeritakse. Igal elemendil on olemas teistest mittesõltuv parameetrite kogum, mis omakorda võib eristuda sõltuvalt elemendi tüübist. Nendest kõige tähtsamad on esitatud tabelis 1.

Tabel 1. Kompositsiooni kihi peamised parameetrid ja nende tähendused

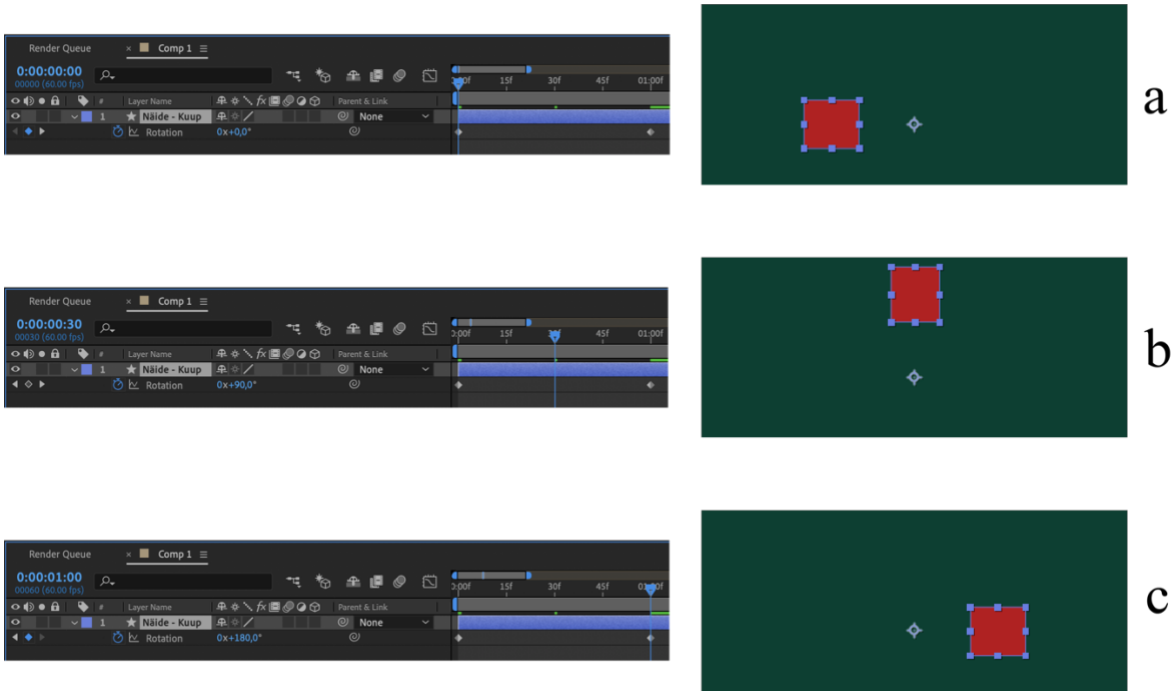
Parameetri nimi	**Tähendus**
Ankurpunkt (ingl *anchor point)*	Punkt, mille suhtes teostatakse kõik muudatused kihis.
Positsioon (ingl *position*)	Kahe parameetriga väärtus, milles üks näitab positsiooni x-teljel ja teine y-teljel (mõni programm võimaldab ka kolmandat dimensiooni lisada).
Suurus (ingl *scale*)	Näitab, kui suur on objekt protsentuaalselt oma algsest suurusest.
Pöörlemine (ingl *rotation*)	Pöörlemine ankurpunkti suhtes 0-360 kraadi.
Läbipaistvus *(*ingl *opacity)*	Näitab, kui hästi on selle objekti taga näha teisi objekte (0 – täiesti läbipaistev, 100 – ei ole üldse läbipaistev).

2.2. Võtmekaadrid

Kuna ühes sekundis on tavaliselt 24 kuni 60 kaadrit [16], siis iga eraldi kaadri parameetrite muutmine animeerimiseks võtaks tohutult palju aega, aga kombineerimisprogrammid oskavad ise arvutada nende parameetrite muutmist. Selleks kasutatakse võtmekaadreid (ingl *keyframe*) [18]. Kasutades võtmekaadreid, ühesekundilist klippi võib animeerida kahe võtmekaadrina ja ei pea kuuekümnele kaadrile unikaalset parameetri väärtust andma (vt Joonis 3). Kindlasti, detailirohke animatsiooni jaoks kahte võtmekaadrit ei ole piisav. Selle saavutamiseks lisatakse võtmekaadreid juurde.

Ajateljel valib kasutaja sobivat aega, kust algab animatsioon. Selles kaadris olles valitakse kihiparameetrite omadused ning salvestatakse võtmekaadrina. Peale seda minnakse ajateljel edasi (või tagasi) vajalik hulka kaadreid ning sätestatakse sama kihi parameetrid uuesti, nii nagu nad peavad selles lõpp- või vahekaadris olema. Saadakse kaks võtmekaadrit, mille vahel programm ise arvutab parameetrite vahetust. Protsessi, mille jooksul programm arvutab parameetreid, nimetatakse võtmekaadrite interpoleerimiseks (ingl *keyframe interpolation*) [19]. Interpoleerimine võib olla erinev, vastavalt vajadusele. Kõige levinum ja kergem on lineaarne interpoleerimine (ingl *linear interpolation*), mille jooksul parameetrite muutumine toimub

konstantse kiirusega, mis võib tekitada muljet liiga mehaanilisest liikumisest. Selle vältimiseks kasutatakse teisi interpolatsioone, mida võib väljendada funktsiooni graafikutena [20], näiteks Bezier (ingl *Bezier interpolation*), eksponentsiaalne (ingl *exponential interpolation*) ja logaritmiline (ingl *logarithmic interpolation*) interpolatsioon.



Joonis 3. Võtmekaadrite kasutus kuju pöörlemisel. (a – algusasend, b – interpoleeritud asend, c – lõpuasend)

2.3. Erinev programmide lähenemine

Põhiline osa liikumisgraafika animeerimise jooksul on erinevate elementide kombineerimine. Filmide jaoks prevaleeruv osa on aga loogiline järjestamine.

Võib tuua välja kaks liiki programme animeerimise jaoks: monteerimisprogramm (ingl *non-linear editing application*) ja kombineerimisprogramm (ingl *compositing application*), mille tähelepanu on pööratud järjestamisele või kombineerimisele vastavalt.

Liikumisgraafika jaoks on paremad kombineerimisprogrammid, kuna võimaldavad rohkem tegutseda eraldi kihtidega, samal ajal monteerimisprogrammid annavad rohkem ruumi loogilise teose järjestamise jaoks.

Peale liikumisgraafika ilmumist siiski ei ole otseselt olemas eraldi liikumisgraafika animeerimiseks loodud programmi. Kõik programmid, mis sobivad liikumisgraafika jaoks, sobivad ka filmide ja videote jaoks, kuna paljud loomise põhimõtted samastuvad.

2.4. Liikumisgraafikale suunatud programmid

Praegu turul on rohkesti liikumisgraafikale suunatud programme, mis eristuvad funktsionaalsuse, efektiivsuse, hinna, litsentsi, platvormi jm poolest.

Kuna liikumisgraafika võib sisaldada ka 3D elemente, ja mõned puhtalt 3D-le suunatud programmid, nagu Blender, sisaldavad ka kombineerimiseks ja monteerimiseks vajalikku funktsionaali (e kuuluvad liikumisgraafikale suunatud kombineerimisprogrammide hulka), siis selles töös sellist tarkvara ei vaadelda, kuna peamine dimensioon töös on kahemõõtmeline. Samuti autor ei käsitle programme, mis võimaldavad võtmekaadrite kasutust, aga on siiski monteerimisprogrammid pigem suunatud videotöötlusele.

Mõned monteerimisprogrammid ja kombineerimisprogrammid nö „käivad paaris“. Adobe ja Apple poolt arendatavatest lahendustest on näha, et üks programm on omas valdkonnas parem ning eeldatakse, et kunstnik kasutab mõlemat programmi. Adobe After Effects ja Apple Motion on programmid, mis on suunatud pigem kompositsioonile ja liikumisgraafikale, samas Adobe Premiere Pro ja Apple Final Cut Pro oskavad paremini järjestamisega toime tulla.

Autor on koostanud ülevaate (vt Tabeli 2) põhilise informatsiooniga programmide kohta, mis on valitud järgmiste kriteeriumite järgi:

- programmis on võimalik tühjast failist koostada kompositsioonlisamaterjale importimata;
- programmi peamine dimensionaalsus on kahemõõtmeline;
- programm on peamiselt kombineerimisprogramm;
- programmis on võimalik kasutada vahekaadreid.

Tabel 2. Ülevaade erinevatest liikumisgraafikale suunatud tarkvarast

Tarkvara	**Arendaja**	**Hind**	**Esmaväljalase**	**Platvorm**
After Effects	Adobe	23,99€ kuumakse	1993	Windows, macOS
Motion	Apple	54,99€ ühekordne	2004	macOS
Fusion	Blackmagic Design	269€ ühekordne	1996	Windows, macOS, Linux
DaVinci Resolve Studio	Blackmagic Design	269€ ühekordne	2004	Windows, macOS, Linux
Natron	Alexandre Gauthier, Frédéric Devernay	tasuta	2014	Windows, macOS, Linux

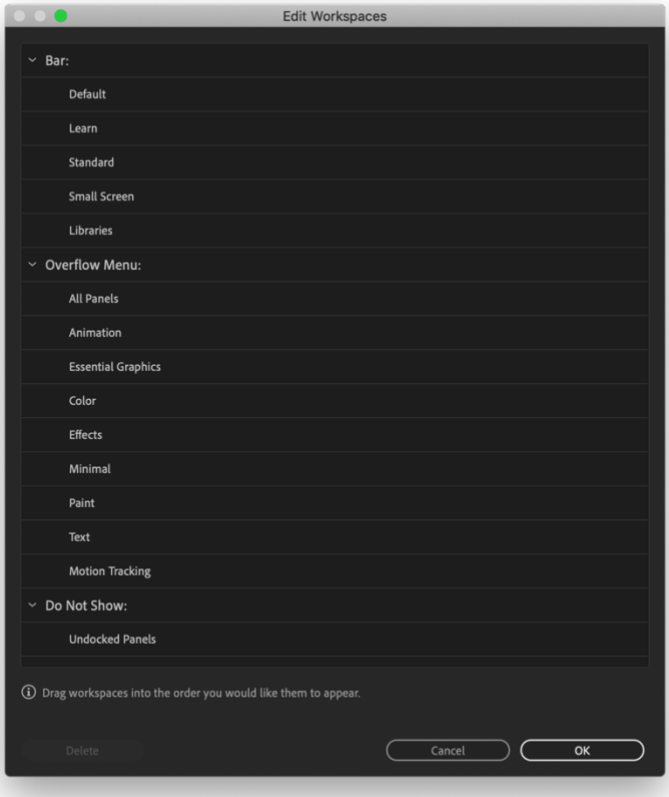
Läbiviidud uuringus [14] selgus, et 75% osalejatest kasutavad peamiseks programmiks Adobe After Effects'i. Programmi võimalused sobivad enamikutelekasutajatele ning selle integreeritavus teise Adobe poolt loodud tarkvaraga Adobe Creative Cloud paketi näol aina soodustab selle kasutamist, kuna paljud kasutajaliidese elemendid on samad. Edaspidi käesoleva töö autor käsitleb just nimetatud programmi võimalused.

3. LIIKUMISGRAAFIKA TÖÖVOO AUTOMATISEERIMINE PROGRAMMIS ADOBE AFTER EFFECTS

Liikumisgraafika animeerimise töövoog (ingl *workflow*) võib kindlasti eristuda teose liigist ning sellest, mis osad lähevad selle sisse. Töövoo kiirendamiseks on Adobe juba eelnevalt loonud erinevaid lihtsamaid võimalusi, mille ülevaade on antud käesoleva peatüki esimeses osas. Need kiirendamise võimalused kahtlemata aitavad animeerimise protsessi kiiremaks muuta, kuigi peamine vahendaja nende tegevuste vahel on ikkagi inimene ise. Automatiseerimine kaheldamatult ka kiirendab tööprotsessi, kuigi on suunatud asjadele, mida arvuti saab ise lahendada ilma selleta, et inimene peaks protsessi jooksul sekkuma. Töövoo automatiseerimisele on pühendatud käesoleva peatüki teine osa.

3.1. Töövoo kiirendamine

Kiirendamiseks kasutatakse erinevad tööruume (ingl *workspace*) ning kiirklahve (ingl *shortcuts*). Erinevad tööruumid aitavad seadistada programmi akna nii, et oleks mugavam toime tulla erineva tööprofiiliga (tekstitöötlus, efektide lisamine, liikumise jälgimine jne). Adobe on andnud kasutajale ka võimaluse enda jaoks vajalikku tööruumi seadistamiseks (vt Joonis 4).



Joonis 4. Erinevate tööruumide ülevaade programmis Adobe After Effects

Kiirklahvid aitavad kasutajal toiminguid kiiremini valida ja tänu sellele efektiivsemalt tööd teha. Programmis on ohtralt kiirklahve kihtide, maskeerimise, ajatelje, failidega manipuleerimiseks, efektide lisamiseks, kompositsiooni sätestamiseks jne. Kindlasti nende teadmine ei ole tarvilik programmi edukaks kasutamiseks, ehkki nende teadmine aitab tõsta töövoo kiirust. On olemas universaalsed, hästi tuntud kiirklahvid nagu *Ctrl+S* (Windows), *Cmd+S* (macOS) projekti salvestamiseks, *Ctrl+O* (Windows), *Cmd+O* (macOS) failide avamiseks, kui ka ainult sellele tarkvarale suunatud kiirklahvid nagu *P* kihi positsiooni sätestamiseks, *O* kihi läbipaistvuse sätestamiseks, *Ctrl+I* (Windows), *Cmd+I* (macOS) projektisse failide importimiseks jne. Pikemalt nendega saab tutvuda programmi manuaalist [21].

3.2. Töövoo automatiseerimine

Liikumisgraafika animeerimisel kompositsioonis võib olla palju kihte, ja nende animeerimine ükshaaval kasutades võtmekaadreid ei ole aja suhtes kiire ning mõne teose jaoks pole isegi mõistlik. Uuringust [14] selgus, et põhiline asi, millest on tunda puudust on aeg. Selleks, et ajakulu minimiseerida, on võimalus liikumisgraafika animeerimise protsessi automatiseerida.

Automatiseerimise jaoks on Adobe poolt pakutud neli erinevat võimalust:

- avaldised (ingl *expressions*): koodijupid [22], mis aitavad säästa võtmekaadrite loomisel aega. Iga avaldis rakendatakse eraldi kihile kompositsioonis, kuigi kood ise võib viidata ka teistele kihtidele;
- skriptid (ingl *scripts*): käsujadad [23], mis ütlevad programmile, mis operatsioone tuleb täita. Skriptide kasutamiseks neid tuleb installida selleks ettenähtud kausta ning aktiveerida programmis;
- pistikprogrammid (ingl *plugins*): suurema funktsionaalsusega tarkvara moodulid [24], nt efektid on ka pistikprogrammid; neid tuleb alguses installida;
- automaatne visualiseerimine (ingl *automated rendering*): on eraldi käsurea programm *aerender*, mis võtab argumendiks projektifaili, väljastamistee, pikkust jt ning automaatselt väljastab valmis videofaili.

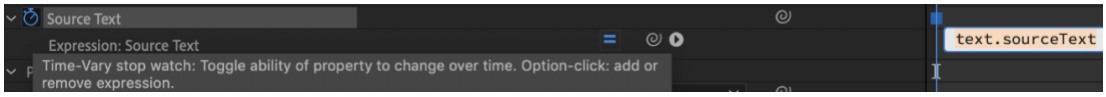
Alljärgnevalt autor uurib nende võimaluste käsitlemist programmis ning toob igast võimalusest kõige tähtsamaid näiteid.

3.2.1. Avaldised

Avaldiste kasutamine programmis on esimene samm töövoo automatiseerimiseks. Avaldisi kasutades võib muuta parameetri muutumist kihisiseselt, ning jälgida teisi parameetreid ka

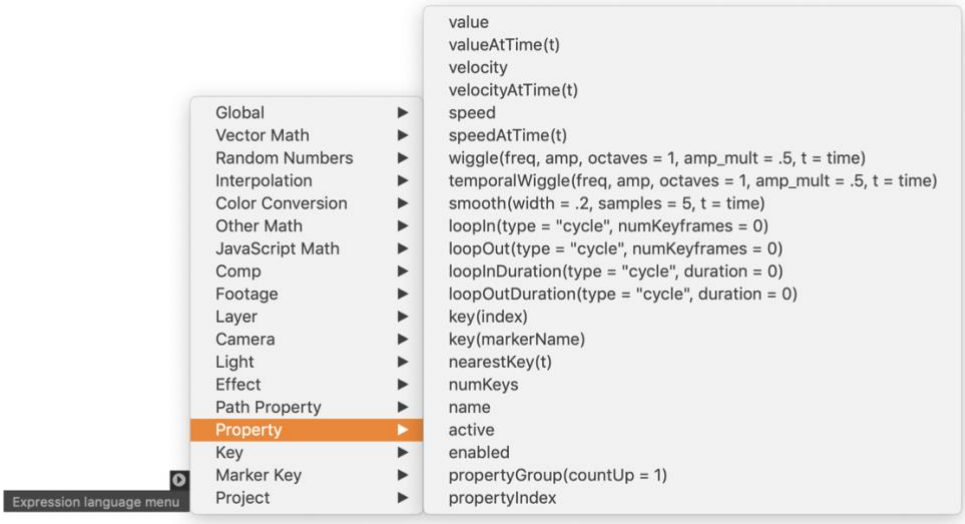
kihiväliselt. Adobe After Effects kasutab nende rakendamiseks avatud lähtekoodiga V8 JavaScript'i mootorit [25].

Selleks, et programmis aktiveerida võimalust kirjutada avaldise, tuleb kihi sätetes vajutada stopperkella peale hoides *Alt* (Windows) või *Option* (macOS) klahvi klaviatuuril, peale vajutamise ilmub aken, kuhu saab kirjutada avaldist (vt Joonis 5).



Joonis 5. Avaldise kirjutamise võimalus sisse lülitatud

Eelnevalt määratud atribuutide ja meetodite arv on üpris suur, vajutades avaldiste keele ikoonil ilmub nende nimekiri kategooriate kaupa (vt Joonis 6), kokku on 18 kategooriat ning 224 erinevat atribuuti ja meetodit (vt Tabel 3).



Joonis 6. Avaldiste kategooriate menüü on avatud

Tabel 3. Avaldiste kategooriad lahtiseletatud

Kategooria	**Tähendus**	**Funktsioonide arv**
Globaalne (ingl *Global*)	Globaalsed atribuudid, sh ka kompositsiooni (*comp(name), thisComp*) ja projekti (*thisProject*) info kättesaamine, ajatelje (*time, posterizeTime (framesPerSecond)*) ning kaadrisagedustega (*timeToFrames (t, fps, isDuration), timeToTimecode (t, timeCodeBase, isDuration), timeToNTSCTimecode (t, ntscDropFrame, isDuration)*) seotud operatsioonid.	13
Vektorite matemaatika (ingl *Vector Math*)	Funktsioonid vektoritega seotud tehete teostamiseks. *Selles töös ei vaadelda.*	11
Juhuslikud arvud (ingl *Random Numbers*)	Suvaliste arvude genereerimise funktsioonid, sh ka müra genereerimine (*noise(valOrArray)*), Gaussi ja tavaline suvalisus (*random(minValOrArray, maxValOrArray), gaussRandom(minValOrArray, maxValOrArray)*).	8
Interpolat-sioon (ingl *Interpolation*)	Funktsioonid erinevate interpolatsioonide rakendamiseks, sh lineaarne (*linear(t, value1, value2)*) ja Bezier (*easeIn(t, value1, value2), easeOut(t, value1, value2)*).	8
Värvi teisendus (ingl *Colour Conversion*)	Funktsioonid värvikoodide teisendamiseks. *Selles töös ei vaadelda.*	3
Muu matemaatika (ingl *Other Math*)	Funktsioonid nurkade teisendamiseks radiaanideks ja vastupidi. *Selles töös ei vaadelda.*	2

JavaScript'i matemaatika (ingl *JavaScript Math*)	JavaScript keeles olevad matemaatika funktsioonid, sh trigonomeetrilised (*Math.cos(value), Math.atan(value)* jne), logaritmiline (*Math.log(value), Math.LN2, Math.LN10* jne), ümardamine (*Math.round(value), Math.floor(value)*), astendamine (*Math.pow(value, exponent)*), pii (*Math.PI*), ruutjuur (*Math.sqrt(value)*)jne.	24
Komposit-sioon (ingl *Comp*)	Funktsioonid ja atribuudid, mis annavad infot käesoleva kompositsiooni kohta (*activeCamera, bgColor, name*), sh ka kompositsiooni suurus (*width, height*), kestus (*duration*), kihtide arv (*numLayers*) jms.	18
Kaadrid (ingl *Footage*)	Atribuudid antud kaadri kohta (*width, height, duration, pixelAspect* jne), kui kaadri näol on tegemist väljaspool programmi tehtud pildi või videoga jms.	13
Kiht (ingl *Layer*)	Atribuudid kihi kohta, jagatud viieks alamkategooriaks: • alamobjektid (*source, mask(name)* jne); • üldine info (*width, height, index, hasVideo, hasAudio, enabled* jne); • omadused (*position, scale, anchor point* jne); • 3D (*orientation, ambient, diffuse, rotationX* jne); • ruumiline teisendamine. *Selles töös ei vaadelda.*	52
Kaamera (ingl *Camera*)	Kaameraga seotud atribuudid, kasutatakse, kui kompositsioon on kolmedimensiooniline ning kompositsioonis on kasutusel kaamera. *Selles töös ei vaadelda.*	15
Valgus (ingl *Light*)	Valgusega seotud atribuudid, kasutatakse, kui kompositsioonis on kasutusel valgusallikad. *Selles töös ei vaadelda.*	7
Efekt (ingl *Effect*)	Efektidega seotud atribuudid (*active, name, param(name), param(index)*).	4
Tee omadus (ingl *Path Property*)	Teedega (ingl *path*) seotud funktsioonid ja atribuudid, sh tee nimetus, punktid jms. *Selles töös ei vaadelda.*	9

Omadus (ingl *Property*)	Funktsioonid ja atribuudid, mis annavad infot käesoleva kihiomaduste kohta, sh väärtus (*value*), kiirus (*speed, velocity*), nimetus (*name*), animatsiooni olemasolu (*numKeys, nearestKey(t), loopIn(type, numKeyFrames)*) jms.	22
Võti (ingl *Key*)	Võtmekaadritega seotud atribuudid, sh väärtus (*value*), indeks (*index*), aeg (*time*).	3
Märgistaja võti (ingl *Marker Key*)	Märgistajatega seotud atribuudid. *Selles töös ei vaadelda.*	9
Projekt (ingl *Project*)	Atribuudid projekti kohta (*fullPath, linearBlending, bitsPerChannel*).	3

Edasi töö autor vaatab oma kogemuse alusel kõige tähtsamaid programmi avaldisi. Mõned avaldised tabelist 3 võivad töötada iseseisvalt ilma teiste kihtide parameetrite teadmata ning eelduseid arvestamata. Mõnda avaldist kasutatakse kihtide ja/või projekti parameetrite kättesaamiseks vahetulemusena.

Näited iseseisvatest avaldistest peamiselt asuvad kategoorias „Omadus“ (ingl *property*), sealhulgas avaldised *loopOut(type, numKeyFrames)*, *loopIn(type, numKeyFrames)* mis võimaldavad programmil arvutada parameetrite muutust automaatselt võtmekaadrite alusel, selleks, et animatsiooni mitu korda korrata, või edasi-tagasi mängida ilma vastavaid võtmekaadreid lisamata. Tabelis 4 on välja toodud nende avaldiste kasutusjuhend.

Tabel 4. Avaldiste *loopIn* ja *loopOut* kasutusjuhend

Avaldis	**Kasutus**
loopIn(type, numKeyFrames)	Võtmekaadrite interpolatsiooni väärtuste kordumine ajatelje algusest kuni esimese määratud võtmekaadrini.
loopOut(type, numKeyFrames)	Võtmekaadrite interpolatsiooni väärtuste kordumine viimasest määratud võtmekaadrist kuni ajatelje lõpuni.
Argument	**Tähendus**
type	Kordamise tüüp (vaikimisi: "*cycle*"): - "*cycle*" – väärtuse muutumise kordamine algusest lõpuni; - "*offset*" – väärtuse muutumise kordamine, kuigi järgmise korduse alguspunkt on viimase võtmekaadri väärtus;

	- "*pingpong*" –väärtuse muutumise kordamine edasi-tagasi; - "*continue*" – väärtuste edasine arvutamine animatsiooni kiiruse alusel, sh *numKeyFrames* kasutamine selle argumendiga ei ole lubatud.
numKeyFrames	Võtmekaadrite arv, mille jooksul tuleb interpolatsiooni väärtusi arvesse võtta (vaikimisi: 0; võtab arvesse kõik võtmekaadrid): - *loopIn()* puhul korduvate võtmekaadrite väärtused võetakse esimesest määratud võtmekaadrist kuni argumendis *numKeyFrames* määratud võtmekaadrini; - *loopOut()* puhul alates viimasest korduvate võtmekaadrite väärtused arvutatakse argumendis *numKeyFrames* määratud võtmekaadrist kuni viimase määratud võtmekaadrini. Arvutatakse numKeyFrames + 1 kaader, kuna animatsiooni kordamise jaoks on vaja vähemalt kahte võtmekaadri.

Sarnase käitumisega on ka avaldised *loopOutDuration(type, duration)* ja *loopInDuration(type, duration)*, kuigi nende teine argument (*duration*) ei sõltu võtmekaadrist, vaid ajatelje ajast (vt Tabel 5). Selle kasutamine ei ole nii tihe, kuigi sellise avaldise kasutamine võib olla kohakuti täpsem.

Tabel 5. Avaldiste *loopInDuration* ja *loopOutDuration* kasutusjuhend

Avaldis	**Kasutus**
loopInDuration(type, duration)	Võtmekaadrite interpolatsiooni väärtuste kordumine ajatelje algusest kuni esimese määratud võtmekaadrini.
loopOutDuration(type, duration)	Võtmekaadrite interpolatsiooni väärtuste kordumine viimasest määratud võtmekaadrist kuni ajatelje lõpuni.
Argument	**Tähendus**
type	Kordamise tüüp (vaikimisi: "*cycle*"): - "*cycle*" – väärtuse muutumise kordamine algusest lõpuni; - "*offset*" – väärtuse muutumise kordamine, kuigi järgmise korduse alguspunkt on viimase võtmekaadri väärtus; - "*pingpong*" – väärtuse muutumise kordamine edasi-tagasi;

	- "*continue*" – väärtuste edasine arvutamine animatsiooni kiiruse alusel, sh *duration* kasutamine ei ole selle argumendiga lubatud.
duration	Sekundite arv, mille jooksul tuleb interpolatsiooni väärtusi arvesse võtta (vaikimisi: 0; võtab arvesse kõik võtmekaadrid): - *loopIn()* puhul korduvate võtmekaadrite väärtused arvutatakse esimesest määratud võtmekaadrist kuni argumendis *duration* määratud sekundini; - *loopOut()* puhul korduvate võtmekaadrite väärtused arvutatakse argumendis *duration* määratud sekundist kuni viimase määratud võtmekaadrini.

Efektset liikumist saab ka tekitada üldse ilma võtmekaadriteta, lihtsalt lisades liikumist suvaliselt. Tabelis 6 autor toob välja avaldist *wiggle*.

Tabel 6. Avaldise *wiggle* kasutusjuhend

Avaldis	**Kasutus**
wiggle(freq, amp, octaves=1, amp_mult=.5, t=time)	suvaline kõikumine
Argument	**Tähendus**
freq	sagedus: kõikumiste arv sekundis
amp	amplituud: väärtuse muutumise suurus
octaves	müra oktaavide kogus: väärtus reguleerib kui kvaliteetne on järelkõikumine
amp_mult	amplituudi suurus järelkõikumisel
t	aeg: avaldise väärtus ajal (kasutatakse, kui on soov avaldise väärtust teisel ajal väljastada vaikimisi: *time*, ehk sama aeg, mis ajateljel (tihti ei kasutata))

Avaldiste kasutamine ei oleks nii laialt levinud, kui nende kasutamine ainult piirduks ühe funktsiooniga, selleks programm võimaldab tekitada muutujaid, kasutada teiste kihtide ja kompositsioonide parameetreid, tingimuslauseid jne ning ei oma otsest koodipikkuse piiri.

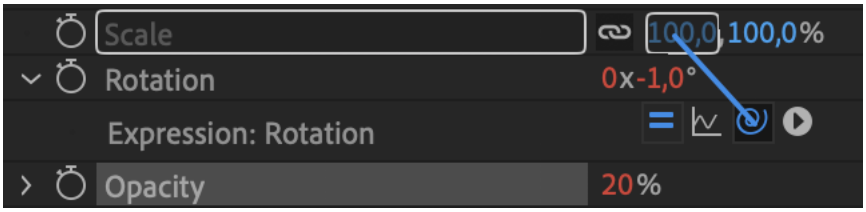
Autor toob välja näite ühest avaldisest, mis aitab tekitada põrke efekti animatsioonil, mis omakorda lisab animatsioonile reaalsustunnet [26], [27]:

```
amplituud = .05; // amplituudi parameeter
sagedus = 2;     // sageduse parameeter
kadu = 7;        // efekti kadu faktor
```

```
n = 0;          // võtmekaadri koguse alguspunkt
if (numKeys >0){
   n = nearestKey(time).index;
   if (key(n).time > time) n--;
}. // kui kihis on võtmekaadrid olemas, siis valitakse viimase võtmekaadri
if (n == 0) t = 0;
else t = time - key(n).time;
// kui kihis võtmekaadreid ei ole, siis ajaline faktor on võrdne nulliga, muidu on ajatelje aja ja viimase
võtmekaadri aja vahe
if (n >0) {// kui kihis on võtmekaadrid olemas
   v = velocityAtTime(key(n).time - thisComp.frameDuration);
   // põrke kiirusmäär on sama mis üks kaader enne viimast võtmekaadrit
   value + v * amplituud *
   Math.sin(sagedus * t * 2 * Math.PI)/Math.exp(kadu * t);
}
// väärtusele lisatakse vastava kiiruse ja amplituudiga põrke efekt, mis on saavutatud siinuse
funktsiooniga jagatud eksponentsiaalfunktsiooniga

else value;// kui kihis võtmekaadreid ei ole, siis väärtus jääb samaks
```

Projekti kihtide kogus võib olla suur ning kihtide nimede otsimine, et lisada neid teise kihi avaldisesse, võtaks äärmiselt palju aega. Selle vältimiseks programm pakub võimalust kohe avaldisesse vajalikku kihti parameetri viidet lisada kasutades valimispiitsa (ingl *pick whip*) (vt Joonis 7). Valimiseks tuleb lihtsalt lohistada piitsa vajaliku objekti peale.



Joonis 7. Valimispiitsa kasutamine *x*-telje suuruse parameetri üleandmiseks rotatsioonile

Valimispiitsa ja avaldisi võib kasutada teiste parameetrite ühesuguseks sätestamiseks terve kompositsiooni või projekti vältel, selleks tehakse eraldi nullkiht (ingl *Null layer*), millele rakendatakse avaldise kontrolli efekt (ingl *Expression Control effect*), mida omakorda seostatakse teiste kihtidega läbi avaldise. Kui kompositsioonis on palju kihte, siis on kindlasti kergem sätestada nende värvi, nurka jne ühest kohast ja korraga.

Autor toob välja avaldise, millega seostatakse parameetri värvuse väärtust nullkihi avaldise kontrolli efektiga samas kompositsioonis:

```
thisComp.layer("nullkihi nimetus").effect("Color Control")("Color")
```

Kui parameetri kontroll asuks teises kompositsioonis, siis avaldis näeks välja järgmiselt:

```
comp("kompositsiooni nimetus").layer("nullkihi nimetus").effect("Color Control")("Color")
```

Avaldise keele aspektide selgeks õppimine võib alguses olla raske ning mõni kasutaja võib kippuda teha kõik käsitsi, kuigi pikemas perspektiivis avaldiste osav kasutamine aitab parandada teose nii visuaalset aspekti, kui ka ajalist kulu projektil.

3.2.2. Skriptid

Skriptid on avaldiste edasiarendused. Need võivad alguses tunda väga sarnased avaldistega. Tõsi küll, skriptid võivad olla sama funktsionaalsusega kui avaldised, kuigi sisemiselt need on ikkagi erinevad.

Skriptide keel on sarnane avaldistes kasutava keelega. Erinevalt avaldistest, skriptid on kirjeldatud formaalsema keelega ning kasutavad ExtendScript'i keelt. See on ECMAScript'i loodud dialekt Adobe poolt [28]. ExtendScript'i keele on võimalik kasutada üle terve Adobe poolt pakutava paketti skriptide kirjutamiseks [29].

Põhiline vahe avaldistel ja skriptidel on see, et avaldised on rakendatud terve ajatelje vältel ehk nende väärtust arvutatakse iga kaader; skripte omakorda peab rakendama soovi korral. Skripte kasutades on võimalik luua kompositsioone, kihte, avaldisi, võtmekaadreid jne [30]. Skriptid võivad teha oma tööd nähtamatult, kuid võivad ka omada graafilist liidest.

Skriptid on tavaliselt .JSX (avatud kujul) või .JSXBIN (kompileeritudkujul) faililaiendusega ning nende kasutamiseks tuleb neid eelnevalt installida programmikausta *Program Files\Adobe\Adobe After Effects <version>\Support Files\Scripts*(Windows) või *Applications/Adobe After Effects <version>/Scripts*(macOS).

Skriptide kirjutamiseks kasutatakse Adobe poolt pakutavat arenduskomplekti (ingl *toolkit*) Adobe ExtendScript Toolkit. Tuleb mainida, et viimase versiooni macOS kasutajatele see ei sobi, kuna programm ei ole piisavalt uus, et töötada macOS uuemates süsteemides. Selle asemel pakutakse Microsoft poolt loodud Visual Studio Code programmi jaoks ExtendScript keele tugi.

Algavale kasutajale on Adobe poolt loodud 193 leheküljeline kasutusjuhend skriptide kirjutamiseks [31]. Edasi autor toob peamised aspektid skriptide kirjutamisel.

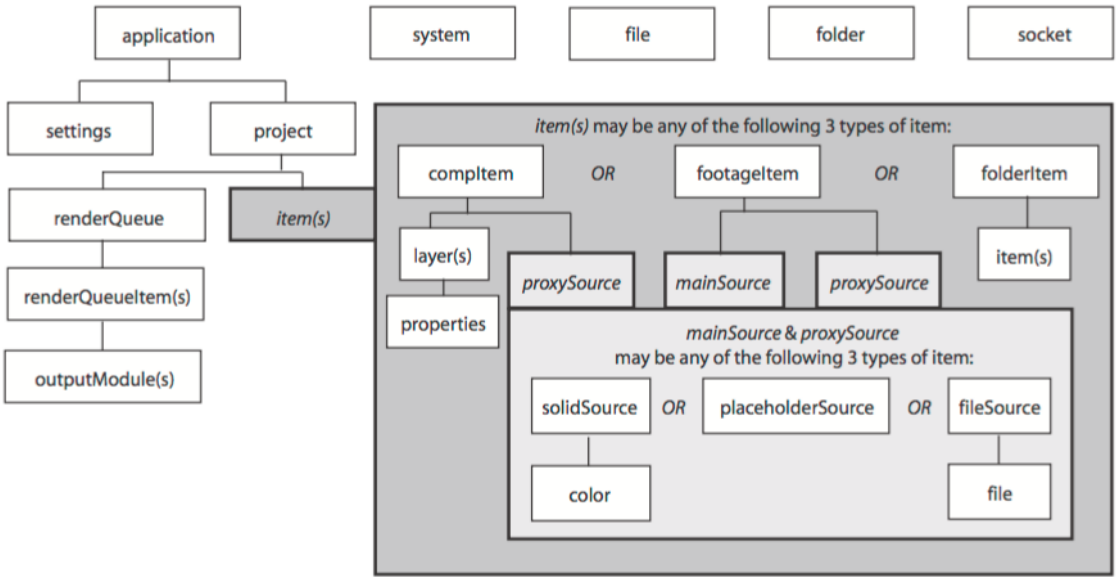
Adobe After Effects oskab käivitada skripte erineval ajal, sh ka käivitamisel või vahetult enne programmist väljumist (sel juhul tuleb skriptifail panna vastavalt kausta *Startup* või *Shutdown*). Skriptide keskkond on terve programmi käitusel sama, e käivitamisel defineeritud muutujad või funktsioonid võivad olla kasutatud teistes skriptides.

Mõni skript peab oma töö jaoks kirjutama informatsiooni failide sisse või küsima päringuid Interneti keskkonnast, kuigi turvalisuse mõttes selline käitumine on Adobe After Effects'i poolt vaikimisi välja lülitatud. Täieliku skripti funktsionaalsuse jaoks tuleb see esmapealt aktiveerida programmi sätetest.

Juhul, kui kasutatakse kasutajaliidesega skripte, mis ilmuvad programmis riputavate paneelidena, tuleb neid panna eraldi kausta *ScriptUI Panels*. Paneelina esitatakse objekt *this*, kusjuures kui paneel käivitatakse funktsiooni sees, on vaja *this* objekti funktsioonile parameetrina üle anda.

After Effects'i mudel koosneb projektist, kompositsioonist, kihtidest jms ning nende elementidest (vt Joonis 8). ExtendScript'i keele jaoks on need kõik objektid. Lisaks sellele mudelile ExtendScript lisab ka 5 kategooriat objektidest, sealhulgas:

- faili ja kaustade objektid (ingl *File and Folder Objects*) – platvormist sõltumatu failipuu navigeerimine, kuna erinevates operatsioonisüsteemides on erinev failipuu struktuuri lähenemine;
- ScriptUI kasutajaliidese moodul (ingl *ScriptUI User Interface Module*) – kasutaja sisendi ja väljundi saamise jaoks;
- vahendid ja utiliidid (ingl *Tools and Utilities*) – lokaliseerimiseks ning teadete edastamiseks dialoogikastis;
- välimine kommunikatsioon (ingl *External Communication*) – väliste süsteemidega sidemeks;
- rakendustevaheline side (ingl *Interapplication Communication*) – Adobe Creative Cloud'i rakendustega sideme jaoks.



Joonis 8. Adobe After Effects'i mudel (allikas: Adobe skriptide kasutusjuhend *[31]*)

Autor toob välja oma poolt kirjutatud skriptist, mis tekitab uue kompositsiooni koos ristkülikuga selle sees.

```
function kasutajaLiides(thisObj) {
   var paneel = thisObj; // Üle antud parameeter this ScriptUI akna jaoks
   btn = paneel.add("button", [20, 20, 100, 60], "Tekita ristkülik");
   btn.onClick = function(){joonista()}. // Interaktiivne nupp
return paneel;
}

function joonista() {
   var name = prompt("Nimetage ristkülik", "ristkülik"); // Interaktiivne aken, mis küsib ristküliku nime
   if(name != null){
      app.beginUndoGroup("ristkülik"); // Programm alustab käskudega,                    mida
grupeeritakse üheks tühistamata grupiks
      var curComp = app.project.items.addComp("Kompositsioon", 1920, 1080, 1, 1, 60); // Projekti
lisatakse kompositsioon nimega „Kompositsioon“, suurusega 1920x1080, pikslite kuvasuhtega 1,
sekundipikkune, kaadrisagedusega 60 kaadrit sekundis
      var shapeLayer = curComp.layers.addShape(); // Lisatakse kujukiht
      var shapeGroup =  shapeLayer.property("Contents").addProperty("ADBE Vector Group");// Kihile
lisatakse vektori grupp
      shapeLayer.name = name; // Kujukihile omistatakse nime
      var kuju = shapeGroup.property("Contents").addProperty("ADBE Vector Shape - Rect");// Vektori
```

```
grupile lisatakse ristkülik
        kuju.property("ADBE Vector Rect Size").setValue([curComp.width / 2, curComp.height / 2]); //
Ristkülikule omistatakse suurus, mis on kompositsiooni suurusest 2 korda väiksem
        shapeGroup.property("Contents").addProperty("ADBE Vector Graphic - Fill").color.setValue([255,
255, 255, 255]/255); // Ristkülikule omistatakse valge värv
        app.endUndoGroup(); // Käsujada tühistamata grupi jaoks on lõpetatud
    }
}

var myToolsPanel = kasutajaLiides(this); // Skripti käivitamiseks muutuja
```

Sellise triviaalse ülesande jaoks on kirjutatud 28 rida koodi, kuigi otse programmist tekitada ristküliku on kindlasti kiirem (tehtav 5 sekundi jooksul). Kuna skriptide kasutusvõimalus on palju laiem kui avaldiste oma, mõnikord vajava kiiruse ja funktsionaaliga skripti kirjutamine võib olla ajakulukam kui animeerimine käsitsi. Selle ajakulu minimiseerimiseks on siiski olemas kolmandate isikute poolt pakutavad skriptide kogumid, mis saavad hõlpsasti hakkama ka raskema töövooga. Tasuta kättesaadavad skriptid on avatud koodiga ning aitavad algajatel aru saada, kuidas skriptide kirjutamine käib. Üheks näiteks on *redefinery* poolt pakutav skriptide kogum [32], mis sisaldab 51 avatud koodiga skripti. Koodi keskmine pikkus on üle 250 rea, mis on märkavalt suurem kui avaldistes kasutatav kood. Sisukad skriptid nõuavad aega nende arendamiseks ja seetõttu muutuvad tasuliseks. Samuti nad on juba kompileeritud e kasutajale nähtamatud.

3.2.3. Pistikprogrammid

Pistikprogrammid on Adobe After Effects'i tarkvara moodulid, mis eristuvad nii avaldistest kui ka skriptidest. Kui skriptid automatiseerivad programmi sisest funktsionaalsust, siis pistikprogrammid lisavad funktsionaalsust programmile juurde ning ei põhine programmi mudelil [33].

Pistikprogrammid enamasti pakuvad After Effects'ile sellist funktsionaali, mis ei ole tavapäraselt ilma nendeta saavutatav. Nii pakuvad nad osakestega (ingl *particles*) ja täiendavat 3D-ga manipulatsioone, lisa maskeerimis- ja kombineerimisvõimalusi, abi avaldiste kirjutamisel ning paremat kiirvalikute ja kiirklahvide kasutust. Samas pistikprogrammide abil võib After Effects programmi siduda teiste graafikaprogrammidega. Nii näiteks on Adobe After Effects ühilduv 3D graafikaprogrammiga Cinema4D [34], mille kasutusel võib After Effects'i sisse importida 3D stseene ning kombineerida neid programmis loodud liikumisgraafikaga.

Erinevalt skriptidest on pistikprogrammid kirjutatud C/C++ keeles ning nende arendamise jaoks on Adobe poolt pakutud arendustarkvara (ingl *After Effects Software Development Kit*). Pistikprogramme peamiselt arendavad kolmandad osapooled, Adobe tavaliselt sellega ei tegele.

Pistikprogrammid on tavaliselt .AEX, .PBK, .PBG ja .8BI faililaiendusega [24] ning nende kasutamiseks neid tuleb eelnevalt installida programmikausta *Program Files\Adobe\Adobe After Effects <version>\Support Files* (Windows) või *Applications/Adobe After Effects <version>* (macOS). Kuna pistikprogrammid on tihtipeale raskema funktsionaalsusega programmid, mõned arendajad pakuvad ka oma installeerijat, mis paneb vajalikud failid õigesse kohta.

Nii nagu skriptide jaoks on ka pistikprogrammide kirjutamise jaoks on Adobe poolt loodud kasutusjuhend, mille pikkus on 417 lehekülge [35]. Selgub, et pistikprogrammide funktsionaalsus on palju laiem.

Arendustarkvara abil saab kirjutada järgmiseid pistikprogramme:

- efektide pistikprogrammid (ingl *effect plug-ins*) – pistikprogrammid, mille abil muudetakse kompositsiooni elemente ja nende parameetreid. Nende abil saab muuta lõpliku pildi välimuse ja sätestada selle vastavalt projektivajadusele. Efektide pistikprogramme on After Effects'i programmil juba integreeritud mitutkümmet;
- After Effects'i üldised pistikprogrammid (ingl *After Effects General Plug-ins (AEGP)*) – nende abil muudetakse projekte ja sätteid. Nende abil saab määrata ja konstrueerida eraldi paneele ning käivitada After Effects'i siseseid käske ja skripte;
- After Effects'i sisend/väljund (ingl *After Effects Input/Output (AEIO)*) – nende pistikprogrammidega saab lisada tuge erinevate faililaiendite importimiseks;
- *BlitHook* pistikprogrammid – nende abiga saab videoväljundi saata kohe teise riistvarale ringhäälingu kvaliteedi taas esitamiseks;
- *Artisans* pistikprogrammid – visualiseeritud 3D kihtide väljund, mille abil saab 3D visualiseerimist suunate teistele programmidele jättes 2D visualiseerimist Adobe After Effects'ile.

Iga pistikprogrammi tüübi jaoks on kasutusjuhendis [24] ka välja toodud näited, mille abil saab kasutaja aru kuidas pistikprogrammid on ehitatud.

3.2.4. *Aerender* programm

Lisaks avaldistele, skriptidele ja pistikprogrammidele on Adobe loonud ka automatiseerimise võimalust kõige viimasel liikumisgraafika teose tegemise etapil – visualiseerimisel. Selleks on

loodud eraldi käsurea programm *aerender* [36], mille abil saab automaatselt visualiseerima panna After Effects'i projekti nii lokaalselt, kui ka läbi võrgu.

Käsurea programmi käivitatav fail asub After Effects'i vaikekaustas *\Program Files\Adobe\Adobe After Effects CC\Support Files*(Windows) või */Applications/Adobe After Effects CC* (macOS).

Kasutuse mõttes on *aerender* üpris sarnane teiste käsurea programmidega, peale käsunime kirjutamist tuleb anda programmile ka vastav argument, nende argumentide nimekirja ja kasutusjuhendit on võimalik vaadata andes programmile argumenti *-help*, mis toob käsuritta kasutusjuhendi esile.

Autor toob välja näite tavapärasest käsust projekti visualiseerimiseks (macOS):

```
aerender -project
/Users/kasutajanimi/kasutajakaust/projektinimi.aep -comp
"Minu_Kompositsioon"-e 3600-output
/Users/kasutajanimi/kasutajakaust/visualiseeritud.mov
```

Antud käsk visualiseerib projekti *projektinimi.aep* kaustast *kasutajakaust* esimesed 3600 kaadrit ja salvestab neid faili *visualiseeritud.mov*.

4. NÄIDISPROJEKT

Selles osas eelnevalt kirjutatud tuginedes, autor rakendab teadmisi näidisprojekti loomisel. Samuti võrreldakse ajaline kulu automatiseerimise lahenduste kasutamise ja neid kasutamata puhul.

4.1. Idee

Selleks, et uurida kuidas automatiseerimine võiks aega säästa liikumisgraafika animeerimisel, pidi autor valima sellist liikumisgraafika tüüpi, mille tegemist oleks võimalik automatiseerida. Kuna mõni liikumisgraafika teos omab süžeed, siis selle jaoks automatiseerimise ajaline sääst võib olla märkamata. Näidisprojekti jaoks on kasutatud alumiste kolmandike animeerimise automatiseerimist.

Alumine kolmandik (ingl *lower third*) on liikumisgraafika elementide ja teksti kombinatsioon, mis ilmub ekraani alumises osas. Nimetusest järeldatult võtab see ainult kuni ühe kolmandiku ekraani osast [1]. Selle liikumisgraafika tüübi peamine ülesanne on edastada informatsiooni kas isiku, ürituse või muu ekraani peal toimuva kohta. Igapäevaselt neid kasutatakse rahvusringhäälingus – uudistes, saadetes või dokumentaalfilmides.

Autor valis alumisi kolmandike näidisprojektis kasutamiseks sellepärast, et uudistes ja saadetes on alati erinevad teemad käsil ning edastatav informatsioon alati eristub, kuigi alumiste kolmandike üldine ülesehitus ja disain püsib olla sama. Selleks, et mitte piirata ainult teksti vahetusega, on kasutatud Eurovisiooni lauluvõistluses kasutatavate alumiste kolmandike mudeli, kus teksti kõrval muutub ka kolmandikke värvus.

Valmiva projekti jooksul loob autor alumiste kolmandikke kogumi, mis näitavad 2020. aasta ära jäetud Eurovisiooni lauluvõistluses osalemiseks valitud artiste [37] koos nende riigi, laulunimetuste, heliloojate ja sõnade autoritega. Kokku saab valmis 42 videoklippi (41 riiki ja peamine aluskompositsioon). Autor teeb kaks sama sisuga projekti – ühe kasutades automatiseerimisvõimalusi ning teise vastavaid võimalusi kasutamata ning seejärel võrdleb ajalist kulu nende projektide tegemisel.

Valmiva projekti jaoks on osutud valituks avaldiste kasutamine kiirema värvikontrolli jaoks ja skripti kirjutamine kiireks kompositsioonide kopeerimiseks ja sõnede vahetuseks.

4.2. Esimene etapp – ülesehituse sätestamine

Adobe After Effects'i programmis oli loodud peamine kompositsioon eraldusvõimega 1920x1080, kaadrisagedusega 60 kaadrit sekundis ning pikkusega 10 sekundit. Peale selle on kompositsioonile lisatud 13 kihti:

- 1 korrigeerimiskiht värvide kontrolli jaoks;
- 4 kujukihti teksti kastide jaoks;
- 8 tekstikihti teksti jaoks.

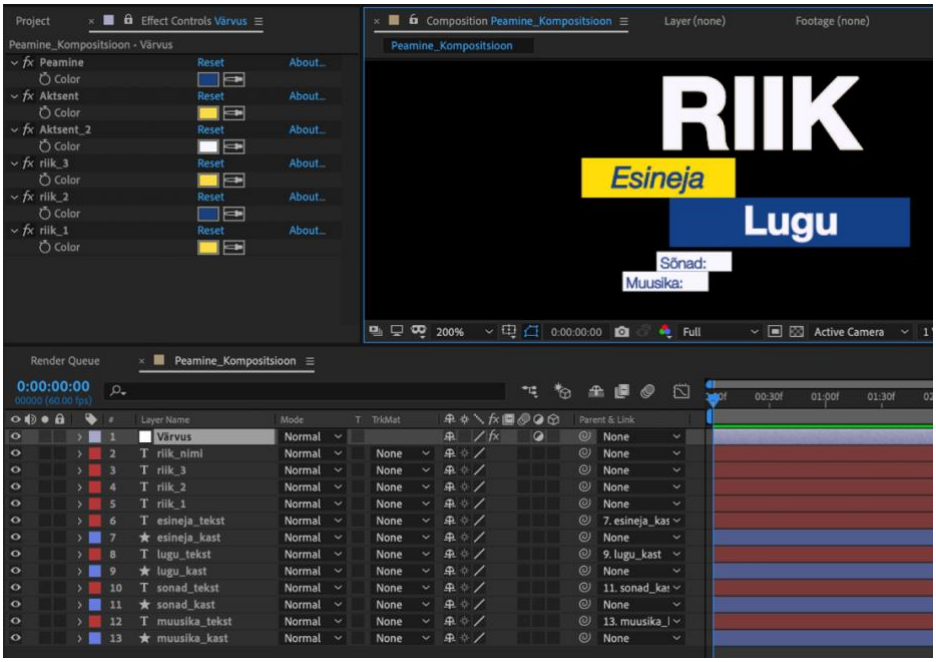
Need kihid on kasutatud üleüldise küljendamise jaoks ning ei sisalda praegu mingisugust animatsiooni.

Korrigeerimiskihis on 6 värvikontrolli efekti, millest pool on disaini jaoks, mis ei muutu, ning pool on iga riigi jaoks, mida pärast muudetakse skriptiga iga erineva riigiga.

Kujukihte kasutatakse tekstikihtide taustana selleks, et teksti sisu ei kaoks üleüldise video taustal. Need kihid kasutavad värvikontrolle, ning on osa disainist, st et nende värv ei pea muutma iga riigiga.

Tekstikihid on järjestatud järgmiselt (vt. Joonis 9):

- 1 tekstikiht riiginimetuse jaoks;
- 3 tekstikihti riiginimetuse jaoks;
- 1 tekstikiht esineja nime jaoks;
- 1 tekstikiht loonimetuse jaoks;
- 2 kihti sõnade ja helilooja jaoks.



Joonis 9. Esimese etapi projektistruktuur

Sellega on peamine projektistruktuur saavutatud, edasi autor määrab vastavaid avaldisi iga kihi jaoks ning lisab animatsioone.

4.3. Teine etapp – animatsiooni ja avaldiste lisamine

Teise etapi alguses on autor lisanud vastavaid seoseid värvikontrolli jaoks kasutades avaldisi. Värvide kasutus on seotud korrigeerimiskihiga järgmise avaldisega:

```
thisComp.layer("Värvus").effect("värviliik")("Color");
```

kus „Värvus“ on korrigeerimiskihi nimi, „värviliik“ on värvikontrolli efekti nimetus ning „Color“ on sisemine After Effects’i parameeter värvi valiku jaoks.

Kujukihis olevad kujud on seotud tekstiga läbi avaldise, mis määrab kuju suurust vastavalt tekstile. Edasi on välja toodud näide, kuidas esineja kasti suurus on määratud:

```
tekst = thisComp.layer("esineja_tekst"); // tekstikihi viide
vahe = 60; // teksti lõpu ja kasti äärevahe pikslites
info = tekst.sourceRectAtTime(time); // tekstikihi sisese kasti suurus ajahetkel
laius = info.width + vahe; // laiuse arvutamine

[laius, info.height] // lõplik suuruse väärtus
```

Iga kuju peab ka asuma teksti taga, selleks on positsioonile rakendatud järgmine avaldis:

```
tekst = thisComp.layer("muusika_tekst"); // tekstikihi viide

laius= tekst.sourceRectAtTime().width/2; //tekstikihi kasti laius jagatud kahega
korgus= tekst.sourceRectAtTime().height/2; //tekstikihi kasti kõrgus jagatud kahega
x= tekst.sourceRectAtTime().left; //tekstikihi kasti x koordinaat kompositsioonis
y = tekst.sourceRectAtTime().top; //tekstikihi kasti y koordinaat kompositsioonis

[laius + x, korgus + y]
```

Võttes arvesse, et iga loo taga võib seista suur hulk heliloojaid ja sõnade autoreid, vastavate kastide suurus võib olla pikem kui mahuks ekraanile. Arvestades seda tuleb vastav tekst jagada mitmeks reaks. Selleks, et nii sõnade autorite kui ka muusika kastid oleksid alati üksteise all, on muusikakasti kujukihi y-telje positsioon arvutatud võttes sõnade kasti positsiooni ja lisades juurde selle kõrgust:

```
pos = thisComp.layer("sonad_kast").transform.position[1]; // sõnade kasti positsiooni y koordinaat
kompositsioonis
korgus= thisComp.layer("sonad_kast").sourceRectAtTime().height; // sõnade kasti kõrgus
```

```
value = [value[0], pos + korgus.height]; //muusikakasti uus positsioon
```

Pikkuse probleem võib ilmuda ka riiginimes. Kuna riiginime ei saa mitmerealiseks muuta disaini ja viisakuse pärast, tuleb riiginime tekstikiht nihkuda vasakule, kuigi alles siis, kui selle pikkus on üle ekraaniääre. Selleks on kasutatud järgmine avaldis:

```
poolPikkus = thisLayer.sourceRectAtTime().width/2; // tekstikasti poolpikkus
compPikkus = thisComp.width; // kompositsiooni pikkus
positsioon = transform.position[0];// x-telje koordinaat
ots = compPikkus - transform.position.valueAtTime(2.5)[0] - poolPikkus;
// üle kompositsiooniääre ots
vahe = 20; // kompositsiooni ääre soovitud vahe

if(poolPikkus >compPikkus - positsioon){
   value = [transform.position[0] + ots - vahe,transform.position[1]]
} else value = transform.position;
```

Muusika tekstikihi ning sõnade autorite tekstikihi jaoks on vaja, et alguses oleks alati vastavad sõned *Sõnad:* ja *Muusika:*, selleks on kasutusele võetud järgmine avaldis (muusika tekstikihi näitel):

```
"Muusika: " + text.sourceText;
```

Sellega vastavad avaldised muudatuste jaoks on sisse viidud. on autor on lisanud ilmumise ja kadumise animatsioonid. Selleks kompositsioonile on lisatud 8 redigeerimiskihti maskidega. Need kihid on animeeritud kasutades võtmekaadreid lineaarse interpolatsiooniga. Kastide ilmumise jaoks on kasutatud võtmekaadreid Bezier interpolatsiooniga.

Animeerimisel on kasutatud ujuvat liikumist, mida on lisatud kihtidele kasutades avaldist. Avaldises on kasutatud siinuse funktsiooni perioodilisust:

```
if(time >2.5) { // kui on läinud rohkem kui 2.5 sekundit
   sagedus = 0.2; // sagedus
   amp = 3; // amplituud
   z = amp * Math.sin(sagedus* time * Math.PI * 2);

   value + [z,0]
} else {value};
```

Selleks, et tekstid liiguksid tekstikastidega koos, on tekstikastid määratud hierarhiliste emadena tekstide külge, sellise võtte kasutades ei ole vaja tekstide kihte animeerida, nad liiguvad automaatselt koos tekstikastidega.

Lõplik projektistruktuur koosneb 21st kihist.

4.4. Kolmas etapp – skripti kirjutamine

Kolmanda etapi jooksul saab valmis skript, mis tekstifaili alusel tekitab kompositsioone projektisse juurde võttes andmeid failist ja vahetades neid peamises kompositsioonis olevate kihtidega.

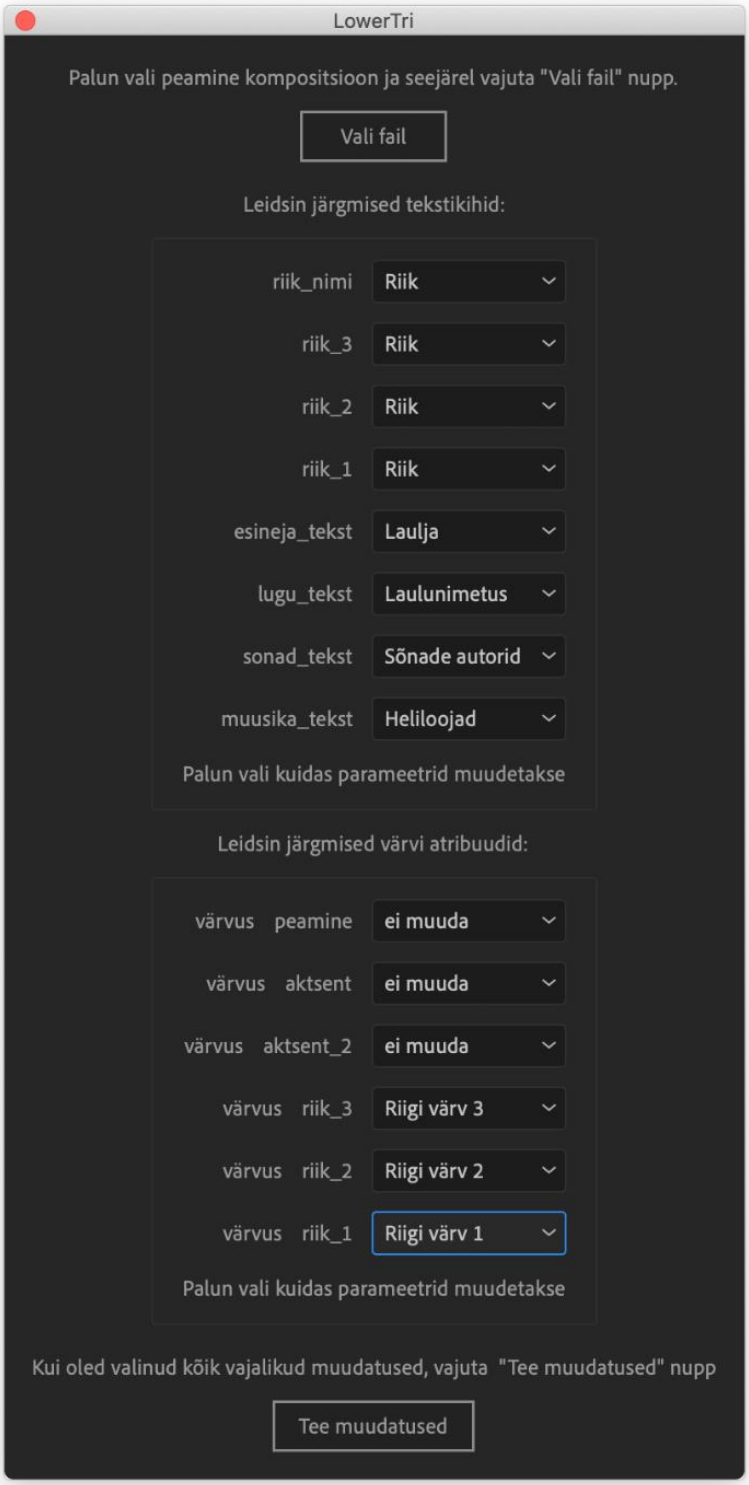
Esiteks oli loodud .TXT fail skripti jaoks, mille sees on toodud informatsioon kõigi Eurovisiooni 2020 osalejate kohta. Iga rea sisu on vastavalt:

- osaleja riik;
- esineja;
- loo nimi;
- loo muusika autorid;
- loo sõnade autorid;
- riigi värv 1;
- riigi värv 2;
- riigi värv 3;

Kokku faili sai kirjutatud 42 rida, millest esimene on päis, mis annab skriptile infot faili sees olevast informatsioonist. Iga infotüki eraldajaks on kasutatud semikoolon. Värvide jaoks on kasutatud nende heksadetsimaalkuju. Skript on kirjutatud programmis Visual Studio Code.

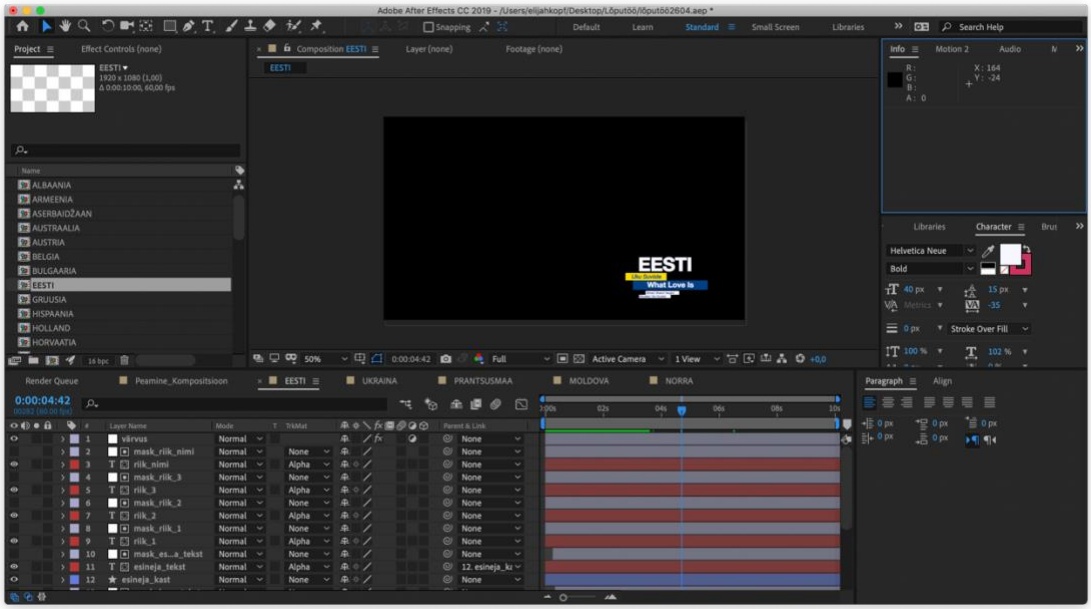
Skripti nimeks on *LowerTri*. Skriptiaken on moodustatud kolmest peaosast (vt Joonis 10). Esimeses osas öeldakse kasutajale, et ta valiks aluskompositsiooni ning peale selle valitakse vajalik fail informatsiooniga, pärast faili valimist ilmub teisse ossa nimekiri kompositsioonis olevate tekstikihtide ning ilma maskideta korrigeerimiskihtide ja nende värvikontrolli efektide sisu kohta, iga kihi kõrval on rippmenüü valitud failis olevate parameetritega. Vaikevalikuks on *ei muuda*. Kolmandas aknaosas on nupp, millel vajutades teostatakse rippmenüüs olevaid valikuid kasutades vastavad muudatused iga kihiga.

Iga failis oleva rea kohta on loodud uus kompositsioon, nimeks valitakse esimene sõna reast. Kui failis olev infotükk on liiga pikk (pikem kui 65 tähemärki), siis eeldatakse, et see on komadega eraldatud heliloojate või teksti autorite nimekiri. Seejärel otsitakse keskel olev komakoht (alates 30.st tähemärgist), ning peale seda lisatakse reavahesümbol. Kuna After Effects kasutab värvi jaoks kolmekohalist järjendit, skriptis on olemas funktsioon värvi teisendamiseks vajalikule kujule.



Joonis 10. *LowerTri* skripti aken

Faili viimase rea jõudmisel lõpetab skript tööd ning teavitab kasutajat, et töö on tehtud ning skriptiakna võib kinni panna. Näidisprojekti puhul ilmub projektisse 41 uut kompositsiooni (vt Joonis 11).



Joonis 11. Projekt peale skripti käivitamist

Valmis saanud skript on loodud peamiselt alumiste kolmandike jaoks ning on piiratud kasutusega. Samas skript eeldab, et kasutaja teab käsitlevat programmi, seega on kaetud ainult kriitilised vead (tühja faili valimine ja peamise kompositsiooni mittevalimine). Vaatamata piiratud funktsionaalile, lahendus ei ole kasutatav ainult antud projektis, vaid võib töötada ka teise ülesehitusega projektide ja failisisuga. Peamine on ainult tekstikihtide ja värvuse redigeerimiskihi olemasolu.

4.5. Saadud näidisprojekti ajaline võit

Võrdlemise jooksul on autor kasutanud näidisprojekti teise etapi seisuga, kui kõik animatsioonid olid sisse viidud. Kindlasti tuleb mainida seda, et teise etapi jooksul avaldiste kasutamine juba pakub hiiglast ajalist võitu, kuna paljud kihid arvutasid ise välja oma positsiooni ja suurust, mõned kihid ei vajanud võtmekaadreid animeerimiseks, vaid kasutasid ainult avaldisi parameetrite arvutamiseks.

Iga riigi jaoks iga kompositsiooni käsitsi kopeerimine ning informatsiooni sisestamine kokku annab tulemuseks keskmist ajalist kulu 1 minut 07 sekundit ühe kompositsiooni kohta. Juhul kui alumine kolmandik vajab lisamuutust sõnede pikkuse sätestamise näol, siis tuleb lisaks veel 3 sekundit. Isegi lisamuutusi arvesse võtmata, käsitsi terve nimekirja kompositsioonide loomiseks kokku võtaks umbes 47 minutit.

Skriptilahenduse kasutusel 42 kompositsiooni tekitamise jaoks läheb umbes 33 sekundit, e ühe kompositsiooni kohta läheb vähem kui sekund aega, mis on märkavalt vähem kui lahenduse kasutamata jätmine.

Käsitsi animeerimisel siiski jääb kontroll iga aspekti üle, kuigi automaatiseerimist kasutades läheb aega juurde ka iga kompositsiooni kontrolliks, selleks, et olla kindel, et kõik muudatused on edukalt läbi viidud. Vaieldamatu isegi vea olemasolul, selle parandamiseks läheb vähem aega. Enamgi veel, kui tuleb viia muudatusi juba saadud kompositsioonide sisse, siis skripti kasutades on mõistlik kaaluda saadud kompositsioonide kustutamist, peamise kompositsiooni muudatust ning uuesti skripti kasutamist, mis tavapärase käsitsi viisil oleks väga ebaratsionaalne.

5. KOKKUVÕTE

Liikumisgraafika on kujundusgraafikast kasvanud nüüdisaegne kunstiliik, mis ühendab erinevaid objekte ja kujundeid koos ühesse teosesse. Selle animeerimiseks ja küljendamiseks on kasutatud programmi Adobe After Effects. Erinevate objektide koosluse animeerimine võib võtta palju aega üheste animatsiooni- ja käitumismustrite pärast, mida on õnneks võimalik automatiseerida liikumisgraafika digitaalse loomu pärast.

Käesoleva töö eesmärk oli liikumisgraafika erinevate olemasolevate automatiseerimisvõimaluste kasutuse ja ajalise võidu välja selgitamine võrreldes neid olukorraga, kus võimalusi jäetakse kasutamata. Automatiseerimisvõimalused olid käsitletud programmi Adobe After Effects näitel. Püstitatud eesmärk on saavutatud ning uuritud võimaluste rakendamine tõepoolest drastiliselt vähendab ajakulu liikumisgraafika animeerimisel.

Bakalaureusetöö koosneb nii teoreetilisest kui ka praktilisest osast. Teoreetilise osa käigus oli selgeks saadud liikumisgraafika ilmumislugu, selle eeltingimused ja nende mõju tänapäevasele liikumisgraafikale. Edasi olid välja toodud peamised aspektid liikumisgraafika animeerimise jooksul ning põhimõtete kasutamine selle loomisel. Samuti oli võrreldud erinevate olemasolevate programmide populaarsust ja põhjust, miks Adobe After Effects oli valitud peamiseks uuritavaks programmiks. Olid analüüsitud programmisisesed lahendused animeerimise automatiseerimiseks ja lahti seletatud nende ülesehitus ja kasutamisala. Töö praktilises osas on valmis saadud näidisprojekt Eurovisiooni 2020 lauluvõistluse osalevate riikide kohta. Informatsioon on esitatud alumiste kolmandike näol. Näidisprojekti käigus on kasutatud eelnevalt töös välja toodud põhimõtteid ja lahendusi, mis aitasid suuremahulise ja monotoonse animeerimistöö automatiseerida. Näidisprojekti lõpptulemusega videote kujul on võimalik tutvuda YouTube lehel: https://www.youtube.com/playlist?list=PLCA4eatwtCb29ceTat1JtiKPhV0xX9sYz. Kasutatud meetmete seas on avaldised ja skriptid, lisaks on valmis saadud autori poolt kirjutatud skriptilahendus alumiste kolmandike automaatkopeerimise jaoks, mis kahtlemata toob tohutu ajalist võitu selle liikumisgraafika tüübi animeerimise jooksul, mida omakorda võib kasutada liikumisgraafika teose rikastamiseks. Autori poolt koostatud skripti lahendus on avalikustatud GitHub'i keskkonnas: https://github.com/elijahkopf/lowerTri.

Bakalaureusetöö kirjutamise jooksul on autor saanud täiendavaid teadmisi üldise liikumisgraafika ülesehituse ja mineviku kohta, laienes autori silmaring avaldistes, skriptides, pistikprogrammides ja teistes liikumisgraafika animeerimisaspektides.

Tulevikus autoril on võimalus saadud skriptilahendus täiendada ning laiendada selle funktsionaalsust, lisades tuge teiste kihtide jaoks. Samas on võimalik selle lahenduse kasutuskogemust parandada ning teha seda kasutajasõbralikumaks. Nende muudatuste vältel on võimalik luua liikumisgraafika disainerite seas vajaliku kommertslahenduse tüütu töö automatiseemiseks.

VIIDATUD KIRJANDUS

1. Krasner J. Motion Graphic Design: Applied History and Aesthetics. Focal Press. 2013.
2. Woolman M. Motion design: moving graphics for television, music video, cinema, and digital interfaces. RotoVision SA. 2004.
3. Cone J. “Motion Graphics” or “Motion Design”?. Motionographer, 2008. http://motionographer.com/2008/10/24/motion-graphics-or-motion-design/ (27.11.2019)
4. Instituut EK. Sõnaveeb. https://sonaveeb.ee (14.01.2020)
5. Kuhn A, Westwell G. A Dictionary of Film Studies. Oxford: Oxford University Press. 2012.
6. Nichol JP. A Cyclopædia of the Physical Sciences. Glasgow: Richard Griffin and Company. 1857.
7. Zorich Z. Early Humans Made Animated Art. Nautilus, 2014. http://nautil.us/issue/11/light/early-humans-made-animated-art (12.01.2020)
8. Gatton M. The Camera Obscura and the Megalithic Tomb: The role of projected solar images in the symbolic renewal of Life. Paleo-Camera, 2010. http://paleo-camera.com/neolithic/ (29.11.2019)
9. Niceron JF. La perspective curieuse ou magie artificiele des effets merveilleux. Paris. 1652.
10. Phantom M. The Magic lantern, how to buy and how to use it, also how to raise a ghost. London: Houlston and Sons. 1874.
11. Dulac N, Gaudreault A. La circularité et la répétitivité au cœur de l’attraction: les jouets optiques et l’émergence d’une nouvelle série culturelle. 1895. Mille huit cent quatre-vingt-quinze. Revue de l'association française de recherche sur l'histoire du cinéma. 2006, nr 50, lk 29-54.
12. Ian A. European film theory and cinema: a critical introduction. Bloomington: Indiana University Press. 2001.
13. Bigman A. Saul Bass: The man who changed graphic design. 99designs, 2012. https://99designs.com/blog/famous-design/saul-bass-graphic-designer-of-a-century/ (20.02.2020)
14. Motion So. The 2019 Motion Design Survey. School of Motion, 2019. https://www.schoolofmotion.com/blog/2019-motion-design-survey (28.03.2020)

15. Read P, Meyer MP. Restoration of Motion Picture Film. Elsevier. 2000.
16. Zamanian K. A Beginner's Guide to Frame Rates. Premium Beat, 2016. https://www.premiumbeat.com/blog/beginners-guide-to-frame-rates/ (28.03.2020)
17. Blaker D. Using VFX Assets in 60fps (After Effects + Premiere Pro). ProductionCrate, s.a. https://news.productioncrate.com/using-vfx-assets-in-60fps-after-effects/ (02.04.2020)
18. Adobe. Setting, selecting, and deleting keyframes. Adobe After Effects User Guide, 2016. https://helpx.adobe.com/after-effects/using/setting-selecting-deleting-keyframes.html (27.03.2020).
19. Adobe. Keyframe interpolation. Adobe After Effects User Guide, 2017. https://helpx.adobe.com/after-effects/using/keyframe-interpolation.html (27.03.2020)
20. Steketee S, Badler N. Parametric Keyframe Interpolation Incorporating Kinetic Adjustment and Phrasing Control. ACM Siggraph Computer Graphics. 1985, nr 19, lk 255-262.
21. Adobe. Keyboard Shortcuts Reference. Adobe After Effects User Guide, 2019. https://helpx.adobe.com/after-effects/using/keyboard-shortcuts-reference.html (07.04.2020)
22. Adobe. Expression Basics. Adobe After Effects User Guide, 2019. https://helpx.adobe.com/after-effects/using/expression-basics.html (03.04.2020)
23. Adobe. Scripts. Adobe After Effects User Guide, 2018. https://helpx.adobe.com/after-effects/using/scripts.html (07.04.2020)
24. Adobe. Plug-ins. Adobe After Effects User Guide, 2019. https://helpx.adobe.com/after-effects/using/plug-ins.html (15.04.2020)
25. Adobe. Expression Language Reference. Adobe After Effects User Guide, 2019. https://helpx.adobe.com/after-effects/using/expression-language-reference.html (03.04.2020)
26. Vandeput J. Super Easy Animations – 5 After Effects Expressions. Cinecom, 2017. https://www.cinecom.net/after-effects-tutorials/super-easy-animations-expressions/ (11.04.2020)
27. Plummer R. How to Use the Bounce Expression in After Effects. School of Motion, s.a. https://www.schoolofmotion.com/blog/after-effects-bounce-expression (11.04.2020)

28. Adobe. Syntax differences between the JavaScript and Legacy ExtendScript expression engines. Adobe After Effects User Guide, 2019. https://helpx.adobe.com/after-effects/using/legacy-and-extend-script-engine.html (11.04.2020)
29. Adobe. Scripting Developing Center. Adobe, 2020. https://www.adobe.com/devnet/scripting.html (12.04.2020)
30. Ebberts D. AE Scripting, Introduction. MotionScript, 2006. https://motionscript.com/ae-scripting/introduction.html (12.04.2020)
31. Adobe Systems Incorporated. ADOBE® AFTER EFFECTS® CS6 SCRIPTING GUIDE. Adobe, 2012. https://blogs.adobe.com/creativecloud/files/2012/06/After-Effects-CS6-Scripting-Guide.pdf (14.04.2020)
32. Almasol JR. rd: scripts. Redefinery, 2013. http://www.redefinery.com/ae/rd_scripts/ (14.04.2020)
33. Paul J. After Effects Plugins 101: What You Need to Know. The School of Motion, s.a. https://www.schoolofmotion.com/blog/after-effects-plugins (15.04.2020)
34. Maxon. CINEWARE Fast. Easy. Integrated. Maxon, 2020. https://www.maxon.net/en-us/products/cineware/after-effects/ (23.04.2020)
35. Adobe. After Effects SDK Guide. Adobe, 2019. https://readthedocs.org/projects/ae-plugin-sdk-guide/downloads/pdf/latest/ (23.04.2020)
36. Adobe. Automated rendering and network rendering. Adobe After Effects User Guide, 2017. https://helpx.adobe.com/after-effects/using/automated-rendering-network-rendering.html (16.04.2020)
37. EBU. Rotterdam 2020 Participants. Eurovision.tv, 2020. https://eurovision.tv/event/rotterdam-2020/participants (17.04.2020)
38. Ebberts D. Dan Ebberts' AE Expressions and Scripting Resource. MotionScript, 2012. http://motionscript.com (17.04.2020)
39. Motion SO. School Of Motion, 2020. https://www.schoolofmotion.com (18.04.2020)

LITSENTS

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Ilja Zolotnikov,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose

   Liikumisgraafika animeerimise tööprotsessi automatiseerimine programmis Adobe After Effects,

   mille juhendaja on Lidia Feklistova,

   reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Ilja Zolotnikov
08.05.2020