Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Rok Kogovšek

Konsistentna postavitev oznak znotraj razstavljenega diagrama 3D modelov

magistrsko delo

magistrski program druge stopnje
računalništvo in informatika

Mentor: Ass. Prof. Dr. Denis Kalkofen
Somentor: Ass. Prof. Dr. Matija Marolt

Ljubljana, 2018

University of Ljubljana
Faculty of Computer and Information Science

Rok Kogovšek

Coherent Label Placement for 3D Exploded View

master's thesis

the 2nd cycle master's study programme
computer and information science

Supervisor: Ass. Prof. Dr. Denis Kalkofen
Co-supervisor: Ass. Prof. Dr. Matija Marolt

Ljubljana, 2018

Povzetek

Naslov: Konsistentna postavitev oznak znotraj razstavljenega diagrama 3D modelov

Vizualna predstavitev informacij s pomočjo oznak znotraj slik nam je vsem že poznana. Redko kdo se pa zaveda, da samodejna postavitev oznak v 2D ali 3D prostoru spada med NP-polne in NP-težke probleme. Najsodobnejši in vodilni med algoritmi ježevo označevanje (*hedgehog labeling*) že ustvari neverjetne in časovno konsistentne postavitve oznak znotraj interaktivnih aplikacij v realnem času. Vendar tako kakor večina algoritmov za postavitev oznak v 3D prostoru predpostavlja statične in ne-spreminjajoče se objekte. Zato uspeh algoritmov močno pade kadar jih združimo s tehnikami predstavitve 3D modelov, ki modelom dodajo gibanje ali pa modele spremenijo tekom časa. Predstavnik obeh problematičnih tipov so razstavljeni diagrami 3D modelov, kjer predstavljamo strukturo modela s simulacijo eksplozije posameznih delov.

Za konsistentno postavitev oznak znotraj razstavljenega diagrama predlagamo razširitev ježevega označevanja s pomočjo gručenja »eksplodiranih« delov in njihovih oznak v razrede. Razredno ježevo označevanje (*clustered hedgehog labeling*) uporabi informacije o poteku premikov za razdelitev 3D prostora v podprostore dodeljene posameznim razredom oznak. Vsak izmed razredov izvaja lastno ježevo označevanje zgolj na oznakah znotraj razreda. Evalvacija predlagane metode je bila izvedena na njeni Textplosion implementaciji v obliki eksperimentov ocenitve uporabniške izkušnje s pomočjo sledenja vida prostovoljcev, kjer smo uspešno zaznali izboljšavo v

izkušnji v primerjavi z osnovnim algoritmom. Zaradi potrebe po testnih 3D modelih v eksperimentu smo tudi ustvarili zbirko 3D modelov, ki jo nameravamo deliti s skupnostjo z namenom postavitve osnove za do zdaj manjkajoče standardizacije testiranja postavitev 3D oznak.

Ključne besede

informacijski vmesniki, predstavitev, 3D postavitev oznake, berljivost oznak, razporeditev vizualizacije, eksplozijski diagram, grafični uporabniški vmesnik, interaktivni uporabniški vmesnik, gručenje, odprto-kodne knjižnice 3D modelov, evalvacija uporabniške izkušnje, evalvacija s sledenjem očem

Abstract

Title: Coherent Label Placement for 3D Exploded View

The use of labels in images represents the basics of visual object presentations that we are all familiar with. However, few know that automatic label placement in 2D or 3D space belongs to the set of NP-complete and NP-hard problems. While state-of-the-art algorithms such as hedgehog labeling already produce incredible coherent results in real-time interactive applications, they were only designed for static and non-deformable objects. Therefore, their performance decreases when combined with the dynamic and model-deforming the 3D model presentation techniques such as exploded diagrams a.k.a. exploded views, which present the structure of 3D model by "exploding" their parts.

We propose an extension of hedgehog labeling to work with exploded views by introducing clustering of model exploded parts and their labels. Clustered hedgehog labeling uses the explosion information to separate 3D space into sections belonging to individual label clusters, each running hedgehog labeling instances solely on the cluster members. The evaluation of the proposed solution and its Textplosion implementation was done by running a usability study enhanced with eye-tracking on a group of volunteers, where improvement of the original algorithm was detected. The need for 3D test models for experimentation resulted in the creation of a 3D Labeling dataset to be shared with the community in an attempt to fill the void of a missing standardized dataset for 3D labeling algorithms.

Keywords

information interfaces, presentation, 3D labeling, label readability, visualization layout, explosion diagram, graphical user interface, interactive user interface, clustering, open-source 3D model libraries,user experience evaluation, eye-tracking based evaluation

Acknowledgments

Firstly, I would like to express my sincere gratitude to both thesis supervisors Ass. Prof. Dr. Denis Kalkofenand Ass. Prof. Dr. Matija Marolt, who both supported me in creation of this master thesis, for their patience, motivation, and shared knowledge.A special acknowledgment is needed for Ass. Prof. Dr. Denis Kalkofen, who acted as the main supervisor even while we were in separate countries. He suggested the concrete topic of this thesis and also arranged access to people and solutions that were the building blocks of this thesis.

My sincere thanks goes also to Dr. Markus Tatzgern and Dr. Ciril Bohak, who provided great help with conceptual and implementation detailed ideas during stimulating discussions. Another important figure was also Dipl. Bernhard Kerbl, who was the leading person in building and providing access to the back-end system for generation of exploded views.

I would like to express my very great appreciation to those hidden behind the usernames of the open-source libraries BlenderSwap and GrabCAD: Harvey Fonseca, Kenny, Magnacad LLC, trinityscsp, Ayoola Adefemi Olaolu, Jabir Mahmud Siam, Bobby Dyer, DHANASEKAR VINAYAGAMOORTHY, Rick Lin, Saffet Firat, Tupay, SRBrandon, androgenius23, Adrian and Ahmed Diab. Thanks to them sharing their 3D models produced trough their hard work, we could build a dataset that can be used even outside this thesis for evaluation of 3D label rendering.

Special thanks also goes to all LGM laboratory members who helped set up the experiment procedure or gave their feedback during pilot runs. Naturally a big thanks goes to all pilot and main experiment participants, who took the time to come and were patient during the long experiment runs during their holiday season and so enabled the evaluation of our proposed solution. Regarding the evaluation I must again thank Ass. Prof. Dr. Matija Maroltand dr. Ciril Bohak for providing a a location for the experiment as well as the eye-tracking hardware.

Last but not least, I would like to thank my family: my parents, my sister and my brother for supporting me throughout my whole university life with their words, belief and deep patience during the writing of this thesis and my my life in general.

Rok Kogovšek, 2018

To my family,
living and recently departed,
I dedicate my final student work.

"Patience is bitter, but its fruit is sweet."

— Aristotle

Contents

List of used acronmys

acronym	meaning
AR	Augmented Reality
AOG	Area of Glance
AOI	Area ofInterest
AR	Augmented Reality
CGI	Computer Generated Imagery
CHL	Clustered hedgehog labeling
FL	Floating labels
GPS	Global Positioning System
GUI	Graphical User Interface
HCI	Human Computer Interaction
HL	Hedgehog labeling
NASA TLX	NASA Task Load Index
SEQ	Single Ease Question
UX	User Experience
VR	Virtual Reality
...	...

Razširjeni povzetek

»`Slika je vredna tisoč besed`«, vendar kako nekdo združi te "besede" oziroma podatke v informacijo se razlikuje med ljudmi. Kadar želimo s pomočjo slik, fotografij, posnetkov ali aplikacije sporočiti drugi osebi neko informacijo, moramo biti pazljivi kako jo predstavimo. Med osnove vizualne predstavitve informacij spadajo tudi oznake znotraj slik. Čeprav se zdi postavitev oznak znotraj slik trivialen problem, se izkaže, da gre v resnici za **NP-polni** in **NP-težki** problem, ki z razširitvijo v 3D prostor doda k težavnosti še šest prostostnih stopenj [1, 2, 3]. Zato moramo v 3D prostoru prenehati razmišljati o oznakah kot zaporedjih črk dodanih na vrh projekcijske ravnine, temveč moramo o njih razmišljati kot 3D objektih v prostoru, zasidranih na objekte, ki jih opisujejo. Za sidranje oznak moramo določiti sidro, ki je opazovalcu vidna točka na objektu, ki ga oznaka želi opisati. Sidro nam predstavlja začetek vodilne črte, ki jo potegnemo do središča oznake in s tem vodimo pogled opazovalca.

Medtem ko sidranje oznak omogoči njihovo postavitev v 3D prostoru, samo sidranje ne reši problema samodejne postavitve oznak, saj moramo najprej določiti meritev uspešnosti posamezne postavitve oznak. Götzelmann, Hartmann, in Strothotte [4] opredelijo tri glavne kriterije za meritev uspešnosti z izrazi **berljivost**, **jasnost** oziroma **nedvoumnost** in **konsitentnost med slikami**, kjer pri vsakemu kriteriju opišejo kopico dobrih praks. Njihove ugotovitve in dobre prakse Schmalstieg ter Hollerer [3] poenostavita v šest jasnih in jedrnatih ciljev za postavitev posamezne oznake v 3D prostoru, katere smo uporabili tekom magistrskega dela kot vodilo pri po-

stavtivi teorije, implementaciji rešitve in njuni evalvaciji. Šest ciljev, katerim smo sledili, smo prevedli sledeče:

Postavi oznako blizu objekta

Zmanjšaj količino kognitivne dejavnosti možganov za povezovanje oznake in z njo povezanega objekta.

Oznake se naj ne prekrivajo

Prekrivanje oznak zmanjša berljivost oznak.

Zunanje oznake se ne postavi pred objekte

Oznake ločimo na zunanje in notranje oznake. Notranje so vselej postavljene pred objektom in izgledajo kot nalepljene na objekt, medtem ko so zunanje oznake vselej izven obrisa objekta in tako ne skrijejo informacij, prisotnih na objektu. Zaradi želje po ohranitvi vseh informacij znotraj 3D modelov je pozornost tega magistrskega dela usmerjena na zunanje oznake.

Vodilna črta naj bo najkrajša možna

Zmanjšaj moteči element vodil. Naj obogatijo pogled, ne osiromašijo!

Sekanje vodilnih črt je nezaželeno

Sekanje vodilnih črt medseboj ali z oznakami zmoti pozornost.

Časovna konsistentnost naj bo prisotna

Postavitev oznake se ne sme spremeniti nenadno med slikami. Veliki skoki oznak ustvarijo zmedo namesto jasnosti.

Pri predstavitvi 3D modelov znotraj interaktivne aplikacije je uporaba oznak zgolj začetek opisovanja zgradbe modela, saj zgolj opisujejo trenutno stanje. Poleg opisov trenutno vidnih delov modela glede na njegovo trenutno pozicijo in rotacijo bi si znotraj interaktivne aplikacije želeli še pridobiti informacije o morebitnih trenutno nevidnih delih modela, skritih v notranjosti. Elementi skriti znotraj modela so še posebej značilni za CAD modele, ki

opisujejo natančne zgradbe predmetov za potrebe kot so proizvodnja. Informativni in intuitivni pristop predstavitve za dani problem so **razstavljeni diagrami 3D modelov**, ki razstavijo model skozi simulacijo eksplozije.

Samostojna postavitev oznak znotraj avtomatsko razstavljenih diagramov poljubnih 3D modelov v okviru interaktivne aplikacije, ki se izvaja v realnem času, je cilj in doprinos tega magistrskega dela. Naslednji doprinos je bila nadgradnja najsodobnejšega in vodilnega med algoritmi za postavitev oznak, **ježevo označevanje** (*hedgehog labeling*), za boljše delovanje znotraj razstavljenih diagramov poljubnih 3D modelov. Za potrebe evalvacije nadgradnje v primerjavi z obstoječim algoritmom smo razvili eksperiment ocenitve uporabniške izkušnje s pomočjo sledenja vida, kar je novost znotraj področja postavitev 3D oznak. Tekom raziskovanja in vse do pisanja tega dela nismo odkrili nobene standardizirane zbirke za preizkušanje algoritmov za postavitev 3D oznak, zato smo sestavili lastno zbirko 3D modelov, obogatenih z besedili oznak in meta-podatki, potrebnimi za pravilno izvedbo njihovega razstavljenega diagrama. Zbirko nameravamo deliti s skupnostjo z nameni zapolnitve vrzeli primerjanja algoritmov, kar prav tako predstavlja doprinos tega dela.

I Pregled sorodnih del

Razstavljeni diagrami 3D modelov so pravzaprav računalniška izvedba risarskega pristopa za opis zgradbe narisanih objektov z začetki v renesansi [5]. Kljub dolgi zgodovini koncepta in njegovi uporabni vrednosti pa ima avtomatizacija procesa začetke šele v letu 2003, ko so predstavili prvi generator navodil za sestavo razstavljenih objektov [6]. Med nedavnimi doprinosi glede pohitritve avtomatizacije procesa se najde tudi delo izpod rok Kerbla, Kalkofena, Steinbergera in Schmalstiega iz Tehniške univerze v Gradcu (TUG) [7], kjer najdemo tudi razne tehnike predstavitve in možnosti uporab razstavljenih diagramov.

Na področju postavitve oznak za 3D aplikacije so največji korak v zadnjih

letih naredili Tatzgern, Kalkofen, Grasset in Schmalstieg [8], ki so z razvojem **ježevega označevanja** prešli iz koncepta postavitve oznak na projekcijsko ravnino na koncept 3D oznak znotraj prostora. Kot osnovni gradnik za postavitev oznak so vzeli takrat 10 let star koncept **plavajočih oznak** (*floating labels*), ki je bil prispevek Hartmanna, Alija in Strothotta [9]. Hartmanna et al. so na področje vpeljali polja sil za določanje optimalne pozicije kot sistem vzmeti. J. B. Madsen, Tatzqern, C. B. Madsen, Schmalstieg in Kalkofen [10] so izvedli raziskavo časovne konsistentnosti in potrdila primernost predstavitve oznak z objekti znotraj 3D prostora.

Glede evalvacije algoritmov postavitve 3D oznak ni bilo narejenih veliko raziskav. Večina evalvacij je sledila splošnemu vzorcu za raziskovanje uporabniške izkušnje znotraj aplikacij za obogateno resničnostjo[11, 10], kjer s pomočjo vprašalnikov zbirajo zgolj subjektivne in empirične meritve poleg zajema potrebnega časa za zaključek naloge. Kot redka izjema sta Azuma in Furmanski [12], ki sta poleg meritev časa preučila še dodatni, vendar omejeni, nabor meritev o dogodkih na oznakah. Prav tako ne obstaja standardizirana zbirka za testiranje kot pri algoritmih za postavitve oznak na 2D kartografske zemljevide, kjer poznajo celo več standardiziranih zbirk [2]. Raziskave o postavitvi oznak znotraj zemljevidov so tudi prehitele naše področje pri uporabi sledenja oči znotraj evalvacij [13, 14].

Sledenje očem znotraj evalvacij rešitev je zaradi vse lažje dostopnosti komercialnih očesnih sledilcev rastoči se trend na področju računalniško ustvarjenih podob (*CGI*) [15]. V 140 letih zgodovine je sledenje očem določilo fiksacijo, sakade, trajanje fiksacij kakor tudi dogodki dogodke na očesu, npr. spreminjanje velikosti zenice ali mežik, kot osnovo za evalvacijske metrike [16, 17, 18, 19, 20]. Poleg numerične primerjave metrik lahko do ugotovitev o obnašanju uporabnika prispemo tudi prek vizualne analitike, kjer opazujemo razlike med grafi kot so toplotne slike (*heatmaps*) in poti iskanja (*scanpath*) [15].

II Predlagana metoda

Razredno ježevo označevanje (*clustered hedgehog labeling*) je bilo predlagano kot nadgradnja, ki obdrži jedrne elemente in zgolj enkapsulira obstoječe ježevo označevanje [8]. Ježevo označevanje projicira oznake na ravnino oznak, ki je vzporedna projekcijski ravnini in postavljena v 3D prostoru. Glede na projekcijo modela in oznak znotraj ravnine oznak izračuna polja sil za določitev najboljše postavitve posamezne oznake prioritetno v bližnji okolici trenutne lokacije, dokler se ne krši šest ciljev postavitve oznak, drugače pa na ravni celotne ravnine. Po postavitvi oznake se postavitev ohrani do določenih dogodkov kot npr. rotacija modela. Med zamrznjenimi stanji računanja se lahko določeni dogodki kot zakritje oznake s strani modela rešujejo s podaljšanjem vodilne črte kar tudi premakne oznako na preprost in nemoteč način.

Težave z razstavljenimi diagrami nastanejo zaradi hitrih sprememb v topologiji modela, kar močno spremeni polje sil med izračuni kakor tudi omeji preostali nezaseden prostor, kar hitreje privede do kršitev šestih citljev postavtive znotraj bližnje okolice oznake. Posledično se z napredkom eksplozij vedno več izračunov vrši z globalnimi iskanji, kar privede do skakajočega obnašanja oznak. Skakanja sicer ne moremo povsem odstraniti zaradi narave razstavljenih diagramov, lahko pa izboljšamo stabilnost postavitev vseh oznak. Stabilnost postavitev smo določili kot empirično mero glede na velikosti skokov oznak med izračuni. Stabilne postavitve so tiste, kjer se skoki oznak zgodijo v bližnji okolici pozicije prejšnjega izračuna oziroma so skoki majhni in tako gladki, da ne zmotijo pozornosti opazovalca. Nestabilne postavitve pa imajo ogromne skoke na nivoju celotne ravnine oziroma povzročijo s skokom preusmeritev pozornosti opazovalca, npr. zrcaljenje pozicije prek modela.

Razredno ježevo označevanje poskuša rešiti problem s prilagoditvijo, da imamo namesto začetnega modela ob vsaki eksploziji več manjših modelov v različnih odsekih prostora. Z gručenjem parov oznak in enotami začetnega modela glede na podane informacije o poteku eksplozije razdelimo problem

na več manjših. Nastali razredi oznak imajo dodeljen lastni podprostor z lastno ravnino oznak. Vsak izmed razredov izvaja lastno ježevo označevanje zgolj na oznakah znotraj razreda in tako ne vpliva neposredno na preostale oznake. Ker so razredi določeni z gručenjem po metrikah, povezanimi z eksplozijami delov, so si deli znotraj razredov razmeroma blizu po postavitvi in obnašanju, kar pomeni manj kršitev načel postavitve oznak, ki sprožijo skakajoče obnašanje. Prav tako omejitev ježevega označevanja na razred oznak omogoči ob kršitvi načel popravilo zgolj problematičnih razredov oznak namesto vseh oznak.

III Implementacija Textplosiona

Ker smo prejeli posredni dostop do sistema za pripravo razstavljenih diagramov 3D modelov predstavljenega s starni Kerbl et al. [7], smo ga uporabili kot naš zaledni sistem za pripravo modelov in tako povsem preusmerili pozornost na postavitev oznak ter na interakcijo z razstavljenimi diagrami znotraj implementacije interaktivnega uporabniškega vmesnika za predstavitev zgradbe 3D modelov. Vmesnik smo zaradi združitve znak in eksplozij razstavljenih diagramov poimenovali **Textplosion**. Textplosion smo zgradili znotraj popularnega in široko razširjenega razvojnega okolja za igre, Unity, kot aplikacijo, ki za vhod sprejme meta-datoteke, proizvedene s strani zalednega sistema in prek njih pravilno naloži priložene razstavljene 3D modele ter jih opremi z oznakami bodisi glede na imena OBJ datotek bodisi z branjem dodatne meta-datoteke s podatki oznak. Za potrebe neodvisnosti od zalednega sistema in njihovega, prek OpenInvertor proizvedenega, zapisa meta-datotek smo razvili lastni JSON zapis meta-datotek eksplozij in oznak, ki ga Textplosion zna tudi prevesti in shraniti kot zaledni zapis meta-datotek.

Sledeč predlaganim interakcijam za razstavljene diagrame 3D modelov s strani Li-ja, Agrawala-ja, Curlessa in Salesina [21], smo razvili tri skupine eksplozijskih interakcij ali na kratko eksplozij. Prva skupina vsebuje **animacije eksplozije** vseh delov v pozitivni in negativni časovni osi, kjer se

posamezni del ustavi, ko prispemo v začetno lego. Naslednja skupina eksplozij so **neposredna manipulacija**, kjer s potegi ali pritiski miške sprožimo eksplozijo posameznega dela modela. Zadnja vrsta eksplozij se imenuje **listanje** (*riffling*), ker sledijo konceptu listanja skozi knjigo, kjer opazovano stran izpostavimo, preostale strani pa odmaknemo za lažjo pozornost na izbrani strani. Podobno tudi v interakciji rahlo eksplodiramo izbrani del v fokus, medtem ko vse morebitne ovire na njegovi poti eksplodiramo mnogo bolj, da pridobimo prazni prostor okoli izbranega dela. Čeprav smo se omejili na tri skupine eksplozij, smo omogočili mnogo več interakcij s pomočjo možnosti združevanja treh osnovnih eksplozij, saj se učinki enostavno kopičijo na skladu.

Izračuni polja sil in iskanja v njih so bili implementirani za grafični procesor prek računskih senčilnikov (*compute shader*), ki jim dostavimo barvno kodiran zajem trenutnega pogleda v aplikaciji. Barvno kodiranje je bilo uporabljeno za razločevanje pripadnosti pikslov posameznim delom modela in se izvede samodejno ob zagonu, kjer glede na število delov modela enakomerno razdelimo HSV barvni model do več kot 7.000 možnih barv, ki so razločljive v številčnem zapisu, kakor tudi na oko za število delov v rangu stotic. Prek razdalj med piksli in uporabniško določenih konstant se izračunajo polja sil, ki jih med iskanjem s konvolucijo v dimenzijah oznake spremenimo v polja vsot vseh sil pikslov, ki bi jih oznaka prekrila. Iskanje optimalne pozicije se izvede z vzporednim redukcijskim algoritmom iskanja ekstremov, ki ga z določitvijo velikosti okolice lahko enostavno ločimo na lokalno in globalno iskanje. Ker polje vsot predstavlja pravzaprav razmazano obliko polja sil, smo uspeli pospešiti izračune z uporabo tekstur nižjih resolucij. Za pravilno delovanje algoritma smo ugotovili, da so dovolj že resolucije okoli dimenzije 100 pikslov. S stališča procesiranja na grafičnem procesorju, glede na vzporedno količino izračunov, se časovno porabi enako kot pri dimenzijah velikosti 128, zato smo se odločili uporabljati 128 x 128 pikslov velike teksture kot tudi podatkovna polja za vrednosti sil.

Z vključitvijo zamrzovanja izračunov smo pa prišli tudi do boljše časovne

konsistentnosti in hitrejšega procesiranja pogledov z enako količino oznak. Za pogoje odmrznitve izračunov smo določili:

1. začetno ali uporabniško sproženo postavljanje oznak,
2. spremembo kota kamere prek postavljene meje,
3. spremembo zooma prek postavljene meje,
4. prenehanje proženja vseh eksplozijskih interakcij.

Pri izboru načina gručenja oznak v razrede smo imeli širok nabor algoritmov za gručenje, vendar smo se zaradi še neraziskanega področja metrik gručenja, glede na razstavljene diagrame 3D modelov, odločili za uporabo metode k-voditeljev, ki se je po petih desetletjih močno uveljavila kot standardni algoritem za gručenje . Glede izbora metrik gručenja smo pa preizkusili veliko idej s poskusi na različnih modelih in določili sledeče štiri kot potencialne dobre kandidate:

Pri izboru grupiranja oznak v razrede smo imeli širok nabor algoritmov za grupiranje, vendar smo se zaradi še neraziskanega področja metrik grupiranja glede na razstavljene diagrame 3D modelov odločili za uporabo K-means algoritma, ki se je po petih desetletjih močno uveljavil kot standardni algoritem za grupiranje. Glede izbora metrik grupiranja smo pa preizkusili mnogo idej s poskusi na različnih modelih in določili sledeče štiri kot potencialne dobre kandidate:

Smer predstavlja normaliziran vektor eksplozijske smeri, kar postavi v skupni razred dele modela, ki se gibljejo v podobnih smereh.

Lokacija je trenutna lokacija dela modela, ki razdeli 3D prostor glede na bližino delov.

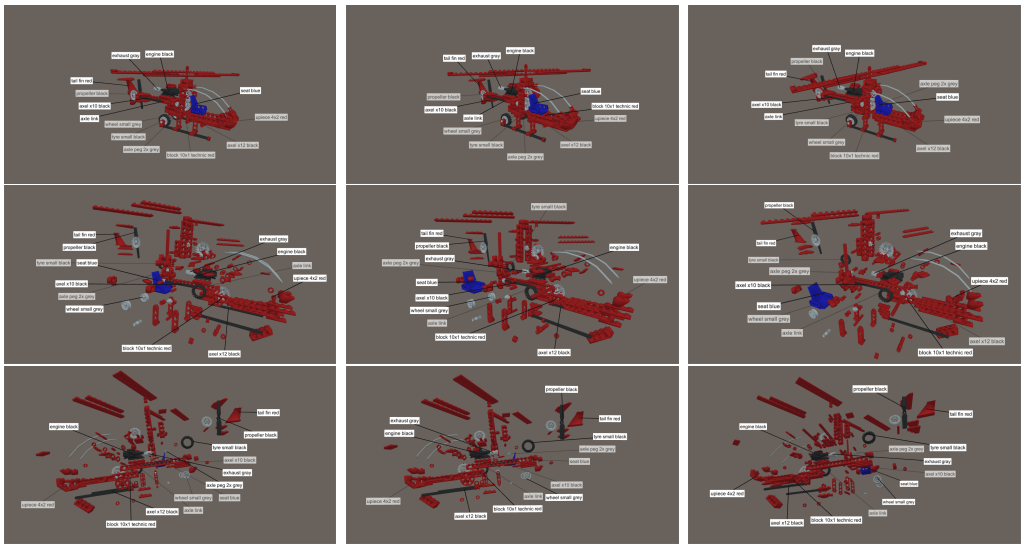
Sprememba opisuje razliko med začetno in trenutno lokacijo dela, kar sporoča napredek eksplozije. Napredek je pomemben, ker želimo združiti v skupnem razredu dele, ki se premikajo kot skupina v podobni smeri.

Razdalje do točke pozornosti so razdalje po komponentah med lokacijo dela in točkami v 3D prostoru, kamor je vselej usmerjena kamera. Rotacija in premikanje objekta v Textplosionu se namreč izvajata prek vrtenja kamere, postavljene na sferi okoli točke pozornosti, ki je določena kot dejansko središče celotnega modela v začetnem stanju. Razdalje do točke pozornosti so alternativni pristop k opazovanju napredka eksplozije, ki vključuje relativno pozicijo v 3D prostoru in tako težje dodeli skupni razred sosednjima deloma, ki se premikata v nasprotnih smereh.

IV Evalvacija

Za potrebe evalvacije smo izdelali lastno zbirko testnih 3D modelov, ker standardizirana zbirka ni obstajala. Iz odprto-kodnih knjižnic 3D modelov smo zbrali, upoštevajoč njihovo licenco, 15 modelov z različnih področij, ki smo jih lastno ročno razdelili na komponente in jim določili oznake. Izmed zbranih je zaledni sistem uspel desetim določiti razstavljeni diagrama. Poleg modelov smo potrebovali tudi primerjalne algoritme, ki smo jih ustvarili z izklopom posameznih komponent Textplosiona in tako prispeli do implementacij ježevega označevanja in plavajočih oznak. S pomočjo zbirke in nabora algoritmov smo lahko samostojno izvedli začetno evalvacijo z empirično primerjavo vizualnih rezultatov postavitev, kjer smo zgradili primerjalne matrike ob bok postavljenih zajetih slik implementacije. Na sliki 1 imamo prikazano primerjalno matriko, ki iz zajetih slik enega izmed 3D modelov iz zbirke prikazuje izboljšave naše rešitve v primerjavi z algoritmoma plavajočih oznak in ježevega označevanja.

Primerjalno matriko smo zgradili z vidika statičnih slik zajetih iz treh zaporednih stanj. Stanja smo določili tako, da prikazujejo tri osnovna stanja pomembna z vidika ocenjevanja postavitev oznak v razstavljenih diagramih. Začnemo z naložitvijo 3D modela v želeni legi pogleda, kar tudi sproži postavitev oznak v začetno stanje matrike primerjave. Naslednjo stanje se pridobi z eksplozijsko animacijo, ki razstavi model in postavi posamezne dele tekom



Slika 1: Matrika primerjave postavitev oznak glede na algoritem in interakcijo za primer 3D modela helikopterja, ki je del zbrane zbirke pod imenom `chopper`. Stolpci matrike predstavljajo rezultate posameznega algoritma v podanem vrstnem redu od leve proti desni: plavajoče oznake, ježevo označevanje in razredno ježevo označevanje. V vrsticah imamo tri zaporedna stanja znotraj istega poteka posameznega algoritma od zgoraj navzdol: začetna naložitev 3D modela in postavitev oznak, vmesno stanje po izvedbi eksplozijske animacije in končno stanje po spremembi lege pogleda. Oznake znotraj prikaza so: `1.` axel x10 black, `2.` axel x12 black, `3.` axle link, `4.` axle peg 2x grey, `5.` block 10x1 technic red, `6.` engine black, `7.` exhaust gray, `8.` propeller black, `9.` seat blue, `10.` tail fin red, `11.` tyre small black, `12.` upiece 4x2 red in `13.` wheel small grey. Metoda k-voditeljev je znotraj naše rešitvi v danem stanju eksplozije, določila sledeče tri razrede oznak: (8, 10, 11), (1, 4, 9, 13) in (2, 3, 5, 6, 7, 12).

celotnega zaslona, čemur se morajo oznake prilagoditi. Doseženo postavitev imenujemo enostavno po-eksplozijsko stanje. V tem stanju preverimo, koliko oznak je s prostim očesom videti, da so ostale znotraj bližnje okolice začetnega stanja in njihovo število uporabimo kot empirično mero eksplozijske robustnosti. Končno stanje v stolpcu matrike predstavlja postavitev po premiku pogleda iz eksplozijskega stanja z namenom opazovanja spreminjanja postavitev oznak znotraj razstavljenega diagram. V končnem stanju določimo empirično mero prilagodljivosti algoritma, ki opisuje obnašanje postavitve oznak znotraj zasičenega prostora z deli razstavljenega diagrama. Za njeno ovrednotenje določimo pojem prostorsko razmerje oznak, ki z vidika opazovalca opisuje relativno postavitev oznake glede na njen opisani del z besedami kot `spredaj`, `zadaj`, itd. Prilagodljivost algoritma enostavno določimo kot število oznak, ki po prehodu iz po-eksplozijskega stanja v končno stanje ohranijo opis svojega prostorskega razmerja z opisanim delom.

Na primeru slike 1 lahko tako opazimo, da imajo plavajoče oznake tri eksplozijsko robustne oznake (8, 10, 13), ježevo označevanje je po eksploziji ohranilo zgolj eno oznako v njeni bližnji okolici (13) in razredno ježevo označevanje je uspelo ohraniti šest oznak v njihovi bližnji okolici (2, 3, 4, 5, 6, 9). Tako iz vidika danega primera naš predlagani algoritem že pokaže napredek v časovni konsistentnosti tekom razstavljanja oziroma eksplozije modela. Povrhu prilagodljivost devetih oznak (1, 3, 4, 5, 6, 9, 10, 11, 13) v primerjavi z eno oznako plavajočih oznak (10) in šestimi oznakami (1, 2, 8, 10, 12, 13) pri ježevem označevanju doda še več podpore za oceno o boljših rezultatih naše rešitve znotraj razstavljenega diagrama za konkretni model. V prid oceni tudi govorijo zgrajeni razredi, od katerih vsebuje razred (1, 4, 9, 13) zgolj oznake z ohranjenimi prostorskimi razmerji, kar nakazuje, da je gručenje uspešno ustvarilo razred oznak bolj prilagojen razmeram v danem razstavljenem diagramu. Podobne ugotovitve smo prejeli z opazovanjem preostalih modelov zbirke. Nakazano je bilo, da naša rešitev mnogo bolj uspešno ohrani konsistentnosti postavitve oznak med deformiranje modela z eksplozijami delov kot primerjalni algoritmi, medtem ko se konsistentnost oznak zno-

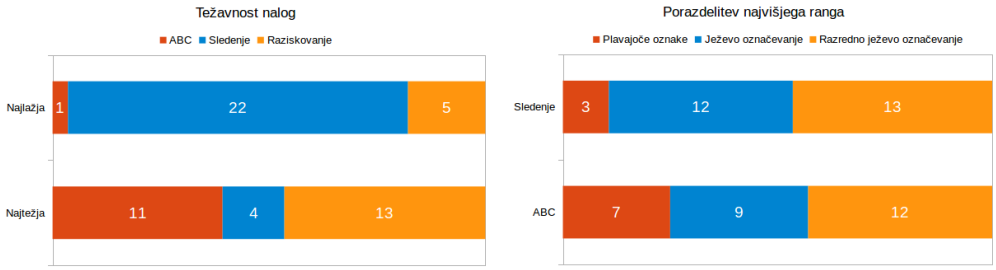
traj premikanja v že razstavljenem diagramu izboljša le pri kompleksnejših 3D modelih z veliko deli.

Čeprav so slike zgovorne, niso močan dokaz o izboljšavi, saj so uporabljene mere povsem empirične in subjektivne ocene. Zato smo dejansko evalvacijo izvedli s pomočjo eksperimenta na skupini prostovoljcev. Za eksperiment smo zmanjšali našo zbirko na izbor petih med seboj različnih modelov zaradi časovne omejitve eksperimentov. Prav tako nismo uporabili uradnih imen algoritmov tekom eksperimenta, temveč smo jim glede na vrstni red dodeli črke A, B in C z namenom preprečitve imen vplivanja na rezultate med nalogami in eksperimenti. Za preprečitev učinka učenja med algoritmi smo naloge oblikovali po principu latinskega kvadrata. Prav tako smo morali določiti nabor metrik gručenja, uporabljenih znotraj razrednega ježevega označevanja, kar smo določili najprej z lastnim testiranjem in logičnim premislekom do nastanka treh potencialnih podmnožic, ki smo jih uporabili v pilotnih testih s tremi razrednimi ježevimi označevanji. S pomočjo rezultatov pilotnih eksperimentov in logičnim sklepanjem smo na koncu izločili iz polnega nabora metrik razdalje do točke pozornosti.

Sam eksperiment je bil zgrajen iz dveh opazovalnih in ene interaktivne naloge. Pri opazovalnih nalogah smo s programom simulirali konstantne vnose uporabniških interakcij in je udeleženec moral le slediti oznakam z očesi, oprema za sledenje očesom Tobii Pro Eye-tracker 4C pa je zabeležila njegove poglede kakor tudi spremembe zenic in podatke iz Texplosiona, kar smo dosegli s popravki v Tobii Pro SDK Unity objektih. Znotraj interaktivnih nalog je udeleženec prejel nadzor nad Texplosionom in je moral v najkrajšem času končati raziskovalno nalogo brez pravilnih in nepravilnih odgovorov, saj je bil poudarek na uporabnosti implementacije. Poleg zapisa časa in podanih odgovorov smo tudi tukaj sledili očesom. Poleg sledenja očesom smo pa zbirali tudi subjektivne meritve prek standardiziranih vprašalnikov SEQ in NASA TLX, kakor tudi znotraj lastnega dodatnega vprašalnika o izkušnji s Texplosionom in subjektivnih rangih algoritmov v posamezni nalogi ter s končnim odprtim intervjujem.

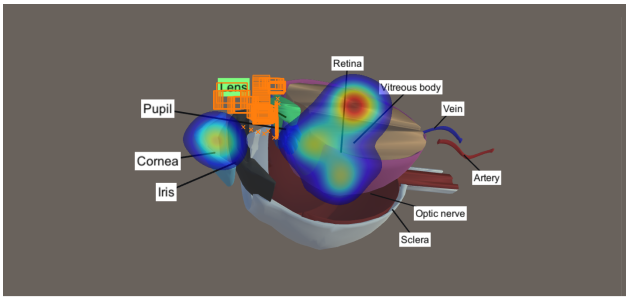
Eksperiment smo izvedli na populaciji 28 prostovoljcev izven matične fakultete in na starostnem razponu od 15 do 60 let. Sodelovalo je 15 predstavnikov moškega in 13 predstavnic ženskega spola. Iz zbranih subjektivnih in objektivnih podatkov smo statistično testirali hipotezo, da naš predlagani algoritem proizvede enakovredne, če ne boljše rezultate pri konstantni omejitvi statistične stopnje značilnosti $\alpha = 0.05$. Standardizirani vprašalniki so prinesli normalne porazdelitve in smo tam tako izvedli v parih t teste in wilcoxonove teste. Pri 35 raziskovanih objektivnih metrikah pa so bili večinoma vzorci v drugačnih porazdelitvah, zaradi česar smo tam uporabili neparametrični Kruskal-Wallis H-test za analizo variance. Preučiti smo morali 35 metrik, poznanih za sledenje z vidom, ker smo imeli opravka z dinamičnimi vizualnimi dražljaji, medtem ko ima veliko od standardnih metrik v sebi močno statično komponento, kar se močno pozna pri vizualni analitiki.

Po pregledu tako subjektivnih kot objektivnih rezultatov smo ugotovili, da razredno ježevo označevanje močno prekaša plavajoče vejice v vseh nalogah. V primerjavi z navadnim ježevim označevanjem pa prikaže podobne rezultate za 3D modele z majhnim številom komponent, medtem ko pokaže boljše rezultate v kompleksnejših modelih z velikim številom komponent. Iz subjektivnih rezultatov tako standardnih vprašalnikov kot tudi rezultatov



Slika 2: Subjektivni vtisi udeležencev eksperimenta razbrani iz rezultatov dodatnega vprašalnika. Levi graf prikazuje porazdelitev udeležencev, glede na nalogi, ki sta jim predstavljala najlažji in najtežji izziv. Desni graf poda število dodeljenih najvišjih rangov posameznemu algoritmu glede na nalogo.

dodatnih vprašalnikov, katerih del je viden v sliki 2, smo ugotovili, da je učinek očiten rahlo že na subjektivnem oziroma podzavestnem nivoju za nalogo, kjer smo opazovali sledenja oznaki v primerjavi s plavajočimi oznakami, medtem ko pri nalogi o razliki intuitivnosti postavitev oznak med algoritmi nismo zaznali nobene statistično podkrepljene razlike, zaznane s podzavestjo med vsemi tremi algoritmi. Smo pa na tem mestu zaznali statistično pomembno odstopanje v nekaterih metrikah zgrajenih na podatkih zajetih pri sledenju očem. Meritve so nakazale krajše čase iskanja oznak pri kompleksnejših modelih in podobne čase pri preprostejših modelih v primerjavi z ježevim označevanjem. To nakazuje na večjo intuitivnost postavitev naše rešitve znotraj razstavljenih diagramov. Prav tako smo z manjšimi povprečji razdalj med točko pogleda in pozicijo opazovane oznake znotraj naloge sledenja, nakazali manjšo količino skokov oznak znotraj eksplozijskega diagrama.



Slika 3: Primer toplotne slike za model očesa pri nalogi raziskovanja, kjer je udeleženec moral poiskati oznako z dano iskano besedo in opisati sosede od dela, ki ga je iskana oznaka opisovala. Znotraj toplotne slike imamo kot ozadje eno izmed zajetih slik sistema, ki prikazuje pogosto lego pogleda udeležencev in tako njim vidnega model z zeleno žarečo iskano oznako. Za bolj jasen pogled na dinamiko oznake, smo zrisali premikanja iskane oznake z oranžnimi okvirji in križci njeno sidro. Toplotna slika uporablja nabor barv od modre do rdeče za prikaz naraščanja fiksacij na posameznih delih slike, kjer področja brez fiksacije pusti v barvi ozadja. Na primeru vidimo področje raziskovanja zgoščeno na delih modela in oznakah v bližini iskane oznake.

S pomočjo vizualne analize zajetih objektivnih podatkov v obliki toplotnih slik, kot primer v sliki 3, smo tudi dokazali, da je naša implementacija primerna rešitev za raziskavo strukture kompleksnih 3D modelov, saj udeleženci z očmi ne tavajo po zaslonu, temveč koncentrirano iščejo v ožji bližini rešitev. Žal pa se je izkazalo, da smo pozabili na uporabniku prijazen nabor tipk, kar je postala ena izmed redkih kritik sistema znotraj intervjuja. S pomočjo objektivnih metrik smo z naknadno raziskavo to tudi dokazali kot dejstvo in ne samo mnenje dela udeležencev. Posledično je razumljivo, da se je za najzahtevnejšo nalogo izbralo ravno nalogo raziskovanja kakor je razvidno iz slike 2, kar je v zvočnih posnetkih eksperimenta pogosto spremljaj komentar o preveliki količini informacij v kratkem času, kamor spada tudi ne-intuitivni nabor tipk.

Glede težavnosti samega eksperimenta so subjektivni rezultati standardnih vprašalnikov pokazali povprečno težavnost, medtem ko so udeleženci komentirali eksperiment kot zabaven, zanimiv in hitro minljiv, čeprav je potekal v povprečju 60 min, od česar je bilo le 15–20 minut eksperimenta treba držati pozornost in opazovati dogajanje v Textplosionu, kar je še skladno z dobro prakso uporabniških eksperimentov.

V Sklep

Magistrsko delo predlaga razredno ježevo označevanje kot učinkovit pristop za konsistentno postavitev oznak v kompleksnih razstavljenih 3D modelih, kar smo podprli tudi s statističnimi rezultati eksperimentov primerjave uporabniške izkušnje, pridobljenimi z našim algoritmom in preteklimi algoritmi za postavitev 3D oznak.

Implementacija algoritma v aplikacijo Textplosion je ustvarila inovativen in v realnem času delujoči sistem za interaktivno in intuitivno predstavitev zgradbe kompleksnih 3D modelov, kot še ni bil viden v komercialnih aplikacijah v taki stopnji samodejne izdelave. Pomen sistema postane jasen, ko vidimo, da je v CAD industriji velika izdelava kompleksnih CAD modelov, ki

jih ustvarjalci pogosto predstavljajo svojim strankam na preprost in intuitiven način. Glede stopnje samodejne izdelave lahko rečemo, da je le omejena z izborom oznak in s pripravo 3D modelov v razstavljene diagrame, kar v veliki meri reši uporaba zalednega sistema. Omejitev glede interakcij tudi ni veliko, saj kopičenje eksplozijskih interakcij omogoči mnogo različnih rezultatov, kar je celo presenetilo udeležence eksperimenta, ki so dejali, da je bilo premalo časa za preizkus vseh možnosti, ki so jim bile ponujene v interaktivni nalogi. Podobno je bilo izraženo, da je eksperiment kljub povprečni zahtevnosti in precej dolgem trajanju minil prehitro, da so bile naloge zanimive in zabavne v tolikšni meri, da so udeleženci izgubili občutek za čas.

S pomočjo interaktivne naloge smo prek komentarjev in iz meritev sledenja očesom ugotovili potrebo po preureditvi sheme interakcij in njim povezanimi tipkami ter gibi, kliki mišk. Prav tako smo tekom analize eksperimenta ugotovili tudi druge težave, ki lahko nastopijo pri sledenju očem znotraj ovrednotenja interaktivnih aplikacij, katerih pred samo izvedbo eksperimenta še nismo zaznali v literaturi, nekaterih tudi ne ob zaključku pisanja tega dela. V zaključku dela, kjer si težave podrobneje ogledamo, tudi predstavimo potencialne rešitve za prihodnje eksperimente.

Dodatno vrednost magistrske naloge najdemo tudi v izdelani zbirki 3D modelov z metapodatki oznak in razstavljenih diagramov, kar lahko postane osnova za zapolnitev vrzeli v preizkušanju algoritmov za postavitev 3D oznak. Hkrati pa smo tudi postavili osnove za nadaljnja dela glede priprave in analize algoritmov za postavitev 3D oznak s pomočjo sledenja vida.

Chapter 1

Introduction

A picture may be worth a thousand words, however, how one selects and combines the information present in the picture differs between people. Prior knowledge, past experiences, personality traits and other differences make each person interpret an image in a unique way, which may be quite different from the message we intended to give when presenting the picture. The dilemma can be avoided by presenting the image with some guidance. How close we are to the desired interpretation across a variety of people depends on the quality of the presentation guidance.

The basics of visual object presentations are images with labels, where strings of text are simply placed on top of or next to an image. While positioning a small amount of labels in a 2D image can be an intuitive task, it is not the same with large amounts of labels. This is still a widely studied problem, with most research done on cartographic map labeling. Both static and dynamic label placements on 2D maps with overlap avoidance are classified in literature as **NP-complete** and **NP-hard** problems. To prove NP-completeness, we can translate the objective into a variant of the NP-complete **subset sum problem** called **general max total problem**. Among all possible screen positions, we are searching for the subset of positions to be used for all labels that maximize a defined energy function. On the other hand, NP-hardness can be proved by reducing the NP-complete

maximum independent set of rectangles problem, where we use labels with unit weights instead of triangles inside sets. The reduction results in a **W[1]-hard** problem [1, 2]. Since we have yet to tackle movement from 2D to 3D space, but we have already arrived at **NP-completeness** and **NP-hardness**, we can say in general that:

Theorem 1.1 *Label placement is an **NP-complete** and **NP-hard** problem.*

In 3D space we do not work with 2D positions, but with poses holding 6 degrees of freedom. Just placing labels in a readable and intuitive way becomes a hurdle, see more details in chapter 2, not to mention that we desire user interactions such as object manipulation or simple observations from arbitrary poses to cause no or at least minimum distractions and misinformation. Labels are more than just strings of text placed on the screen of the projected 3D space. Labels should be seen as text objects anchored to their referred object. Anchoring brings forth the definition of anchor points and anchor lines a.k.a. leader lines. An anchor point is a point of the object, be it at the center of the object or a visible point close to the center, from which a leader line extends to the center of the actual label object. Anchor lines can simply be conceptual, but since they provide good guidance, they are usually rendered as actual lines. A visual dissection of 3D label components can be observed together with an alternative approach proposed by Tatzgern, Kalkofen, Grasset and Schmalstieg [8] in Figure 1.1.

Götzelmann, Hartmann, and Strothotte [4] divide labels into internal and external labels. The internal ones are placed directly over the object as a sticker, which by Schmalstieg and Hollerer's [3] definition corresponds to the label being inside the object's silhouette projected on the view. Any label outside the object's silhouette is contrary an external label. This thesis focuses on external labels, since they do not occlude object parts and so only provide information without any information loss. Götzelmann et al. [4] define three criteria that labeling algorithms for internal and external labels

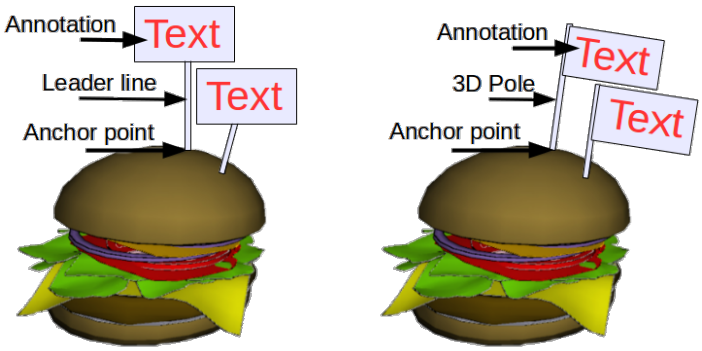


Figure 1.1: An alternative 3D label component setting proposed by Tatzgern et al. [8], where leader lines are used as poles for hanging their label or annotation object as a flag instead of pointing towards its center. Image was taken and modified from their hedgehog paper.

have to fulofill, while also nothing that the criteria can conflict each other. They define the first criteria as **readability**, where label position and font are judged. For external labels we should avoid overlaps between themselves, leader lines and the annotated object, while also avoiding leader line crossings. The second criteria of **unambiguity** considers the difficulty of the observer connecting labels and their referred objects together. Therefore external labels should be placed as close as possible to the referred object as well reduce the number of curves on the leader lines. Another important cause of ambiguity are anchor points clustered close together, which should be prevented. The final criteria of **frame-coherency** was introduced to reduce any layout flickering observed by the observer. Schmalstieg and Hollerer [3] take their criteria with examples of good practices for external labels and redefine them as six placement objectives:

Place label close to object

Minimize the brain work of connecting the label and its referred object.

Labels should not overlap

Label overlaps cause readability problems.

External labels not placed over objects

External labels are defined to be outside all object silhouettes.

Leader line length should be minimal

Minimize the amount of distraction caused by the guiding elements.

Leader line crossings are undesirable

Crossings between leader lines and other leader lines or labels cause distraction.

Temporal coherence maintained

Label position should not abruptly change between frames. Big label jumps cause confusion instead of guidance.

We use these six objectives inside the thesis as a guideline for readability and solution building as well as during the evaluation stage. Furthermore, if the six objectives are translated into mathematical constraints, we create a constraint optimization problem, which is NP-hard by its nature. This way we once again prove Theorem 1.1.

For a 3D application focusing on presenting 3D models, we do not only desire guidance with additional information, which are labels in our case. An important factor is also to include simple and intuitive human-computer interaction (HCI). HCI in 3D space is still a focus point of research in virtual and augmented reality (VR/AR) graphical user interfaces (GUI). In industries such as design, it also needs to include an option that presents a whole and clear overview of progress and mistakes without breaking the design work-flow, whether in video game design or in computer-aided design (CAD).

An informative way of presenting the structure of 3D objects is by disassembling them into so called explosion views or explosion diagrams, where disassembled parts move away from each other as in an explosion. Their behavior is presented in Figure 1.2, where we present an exploded view generated by the system Kerbl, Kalkofen, Steinberger and Schmalstieg [7] proposed. While such diagrams can already be automatically generated and be

used in interactive GUIs, accompanying them with additional information in labels was an unresolved problem until now. The problem with previous label placement solutions is that they only take static objects into consideration, and therefore do not take the 3D model deformation into account, which happens due to the dynamic nature of parts inside exploded views.

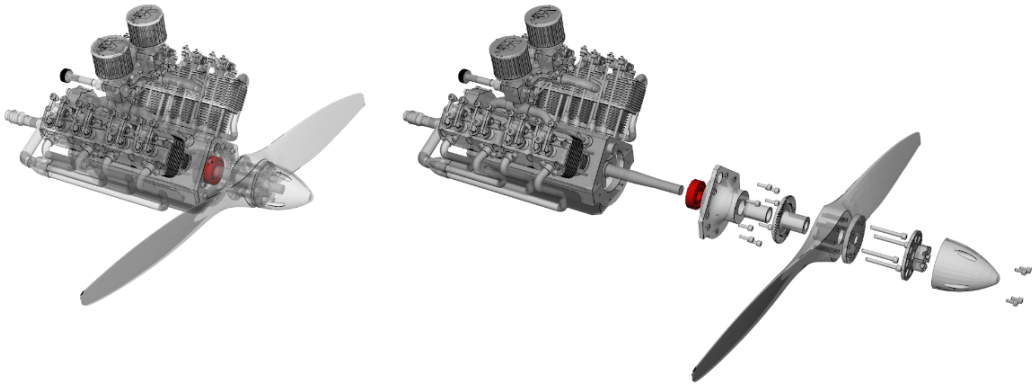


Figure 1.2: Kerbl et al. [7] designeg a system for disassembling CAD models into explode models based on their geometry. The images retrieved from the paper show an exploded view created from a plane motor CAD model by using their system. Selecting a part of the model to be moved causes all of the parts blocking its way to give space and produces the exploded view.

The goal of the thesis is to produce a solution for annotated 3D exploded views by extending previous works with clustering exploding parts into clusters. Each cluster runs its own instance of the algorithm to prevent subspace problems effecting the label layout of the whole model, which happens with currently used algorithms. With the right selection of clustering metrics we can cluster together exploded parts that behave in a similar fashion and are close to each other in space, which together reduces the chances of breaching the six objectives and therefore, increases the temporal coherence due to less global recalculations inside a cluster.

The proposed solution was implemented as a simple, interactive, intuitive and informative GUI for 3D explosion diagrams. Inside it we also used

proposed solutions from other works to enhance the user experience. We included well proven exploded view interactions [21] so that they can be stacked on each other to produce a limitless amount of exploded view variations for the same model and its set explosion directions. By using colors and their alpha channel as indicators of states we succeeded including more information into the scene without causing more distraction as proposed from GPS navigation enhancement studies [22]. While for reduction of ambiguity, we reintroduced the anchor point selection method of using 2D thinning algorithms proposed by older works [9], but later forgotten due to the transition of label placement algorithms form 2D to 3D space.

We evaluated our solution with a user-based usability study enhanced with eye-tracking. Its main contribution to the subjective empirical results of the usability study is the collection of objective numerical data of eye movement behavior, which can to a limited degree describe the brain's inner visual processing. However, visual stimuli with high enough sophistication can hinder assessment of complex interactions and knowledge discovery. Therefore, eye-tracking results should be taken with a degree of caution and some understanding of the differences between our and other eye-tracking studies [15, 16]. Inside our usability study, we compared the results of our solution with results of previous algorithms on actual CAD models provided by the open source community. By creating a test bed of 3D models, we also provided a dataset to be shared for future scientific work on the topic of 3D labeling, since none existed yet during our research.

Chapter 2

Related Work

Explosion diagrams have been used by illustrators as an informative way of presenting the structure of 3D objects since the Renaissance [5]. However, the automation of the process for 3D models started only after Agrawala et al. published their Step-By-Step Assembly Instructions [6]. Prior systems required a high amount of user input, including movement directions for each part that needed to be disassembled. Agrawala et al. had already prepared a system that generated assembly instructions that define the order and direction for parts while keeping blocking constraints in check. This was then used by Li, Agrawala, Curless and Salesin [21] to define explosion graphs to create an interactive automated explosion system. By using an existing part hierarchy or by removing the unblocked parts one by one, they generate explosion graphs that show the relative blocking relations between parts. This enabled a more interactive system for cases such as only exploding the parts that block the selected part from exposing itself.

Kerbl et al. [7] have explored effective disassembly algorithms for complex CAD models. They focused on minimizing the number of checks needed for interblocking parts that we wish to be disassembled in order to speed up the process significantly. They also incorporated interactive user selection of part grouping and even enabled search queries for similar groups to select all the parts of the same type that are found in the model. It was presented with

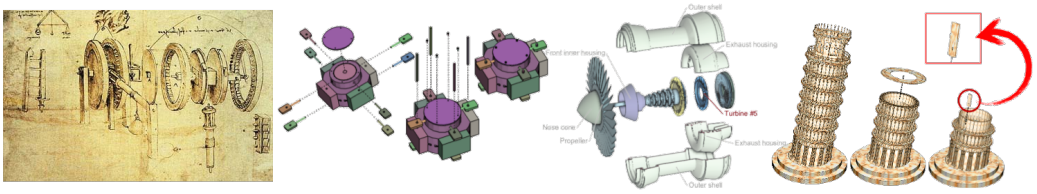


Figure 2.1: From left to right we see a time line of contributions that brought exploded views to its current state. Ferguson [5] found their first usage in Renaissance drawings of the great men of that period, including Leonardo Da Vinci's multiple works such as the presented exploded view of a weight-driven ratchet device. After centuries of drawing exploded views by hand Aragawala et al. [6] proposed the assembly instruction generator. Li et al. [21] followed with research into interactivity, while Kerbl et al. [7] went for disassembly optimization as well interactivity during disassembly. The images were taken from the corresponding papers [6, 21, 7], while Da Vinci's drawing is a color version of the one in the book found on Wikipedia [5, 23].

multiple visualization techniques, including disassembly explosion diagrams, simple object assembly animation and action diagrams.

A simple recent approach to the labeling of 3D objects can be observed in the proposed 3D content Web GUI from Jankowski and Decker [24]. They propose a billboard label placed on top of the rendered view of the world as shown in Figure 2.2. This provides a clear view of the label, but at the same time it occludes the object. Apart from 3D user interactions such as navigation and wayfinding, it shows worse results than a text document with images for other studied user experiences. For better results a label has to have a stricter definition than a string of text on top of the rendered view, as we have already defined in chapter 1.

Regarding the actual 3D annotation rendering research, most was done for static objects. Hartmann, Ali and Strothotte [9] started the current trend of generating potential fields a.k.a. force fields with their paper on floating labels. By calculating force fields based on view projections of each 3D

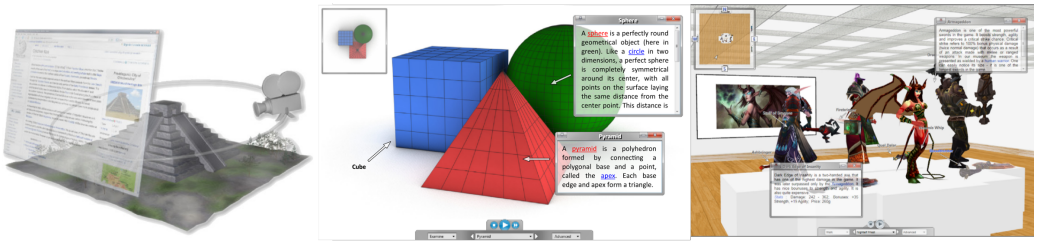


Figure 2.2: Jankowski and Decker use billboarding on top of the view plane as shown in left image as a way to present 2D labels accompanying 3D models. While comparing text files with rendered images of the 3D models with their billboard conversion as in middle image shows improvement in the experience, however for complex situations as the VR application presented in the right image results were worse for most metrics dueto the problem of occlusions. All images were taken from their paper [24].

model part, they determined good initial label positions. While their solution from 2004 was too slow for a real-time interactive application, it already showed promise for interactivity. The following year, Ali, Hartmann and Strothotte [25] proposed several real-time interactive placement algorithms, which were of simple design and worked with a preset of layouts, while also solving the leader line crossing problem. Some of their requirements already tackled most of six label placement objectives [3], seen in Chapter 1. Tatzgern, Kalkofen and Schmalstieg [26] reintroduced floating labels in 2013 when they proposed using label clustering and independently laying out the cluster representatives in the view plane to minimize the amount of labels to present. The initial layout was done by minimizing the energy function comprised of the distance between label and object, object size and object visibility. The objects were in fact semi-static compact explosion diagrams from previous works. The initial layout is checked for overlapping and corrected. Additionally, they added the distance between labels to the energy function for similar distances between labels. The clustered labels can all be rendered under the same label, or can simply be represented by the rendered

cluster representative.

A year later, Tatzgern et al. [8] proposed the current state-of-the-art leading solution, hedgehog labeling. It solves problems with positioning and overlapping of labels using a novel idea. Contrary to previous techniques, it places the labels in 3D space from the start instead of placing them into the scene after the projection to the 2D view plane is done. To achieve consistent results without producing overlapping of labels or disproportional labels, it uses 3D geometric constraints. The constraints define a 3D leader line anchored in the object's center, which extends towards the center of the label. Temporal coherence strategies for hedgehog labeling were researched later by J. B. Madsen, Tatzqern, C. B. Madsen, Schmalstieg and Kalkofen [10], who noted that the initial layout should be kept as long as possible for better user experience. They also proved the assumption that 3D space labels give better results than 2D view labels. Their results can be observed in Figure 2.3, where images taken from a real-time mobile AR application show sequential changes in the mobile phone's pose and the response of hedgehog labeling to the pose changes.

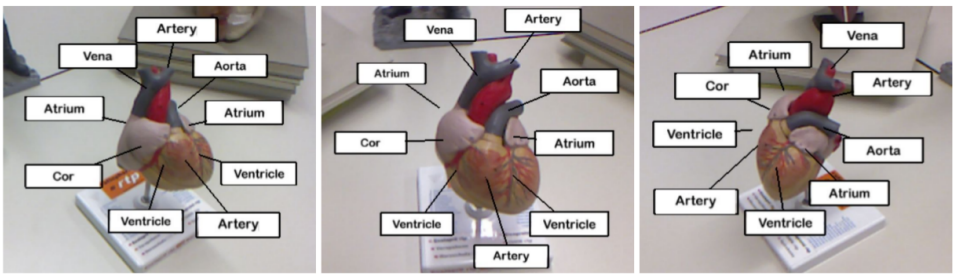


Figure 2.3: Tatzgern et al. [8] produced the state-of-the-art hedgehog labeling by positioning the label placement into 3 space onto poles extended as hedgehog's spikes. The images were retrived from the latter paper on time coherence by Madsen et al. [10], who also proved again the contribution of 3D labels in comparison to their 2D alternatives.

Figure 2.4: Vaaraniemi et al. [22] were observing various visual ways to improve GPS based car navigation. The most promising approaches were transparent buildings for presenting hidden streets as seen in middle image. The information gain is apparent just by comparing it with the initial state of the map presented in left image. In the right image is the second found solution, namely glowing roads. For glowing roads they also added transparency besides the glowing effect but the glow was the important factor. All images originate from their paper.

Hallqvist [27] researched the movement of labels for command terminals such as those found at airports. Therefore, he limited himself only to labels without objects in the world. Each label represents a moving point in the world and is positioned dynamically by predicting collisions with other labels.

Some research regarding label visual components that improve overall readability of the presented view was done for 2D map applications. One of the latest being the 2013 GPS navigation study by Vaaraniemi, Freidank and Westermann [22]. It showed that glowing and transparent elements can improve the user experience, which can be observed in Figure 2.4, where both are presented in comparison to a traditional GPS navigation setting. Another source for label readability research is AR, where labels are often used for providing information. Leykin and Tuceryan [28] have showed that, in regards of readability of labels, a strong enough contrast is needed between the label and background. They also produced an evaluation and correction system for automatic adjustment of label to background contrast based on their study results.

The evaluation of 2D or 3D label placement onto virtual 3D models or during AR enhancement with labeling is presented only in the minority of

its literature, while the majority just provides result examples through images or videos. Among the minority of articles evaluating their methods, most use the subjective empirical data gathered from usability studies on their case limited selection of models or AR enhancement targets [10, 12] similar to other AR studies as presented in the 2008 overview of experiments by Dünser, Grasset and Billinghurst [11], where objective numerical data is usually limited to only consist of task completion time (TCT). Azuma and Furmanski [12] went further than that and also included performed numerical measurements outside the usability study. They observed the computation time and cost, the number of produced label overlaps, the number of moved labels and the number of moved non-overlapping labels. In other computer science areas, we usually observe the objective numerical evaluations based on standardized benchmark dataset, however there are no such datasets shared among the researchers for 3D labeling, while cartographic 2D label placement problems actually have multiple in existence [2]. In cartographic research, they even reached out to eye-tracking for measuring the map reading performance [29] or the usability of produced dynamic 2D or 3D maps [13, 14].

Kurzhals, Burch, Pfeiffer and Weiskopf [15] found in recent years a growing trend for the use of eye-tracking in visualization and CGI research. The grow in trend was accredited to commercial eye-trackers reducing the required cost and work as well as to the fact that eye-tracking complements traditional assessments. While eye-tracking is a rising trend in recent years, it is far from a new concept and actually has a deeper history than one would imagine, going all back to 1878 with mechanical test devices [16, 17, 18]. In its 140 years three attributes of eye movement behavior were determined as those of greater importance. The first one is the gaze location or fixation, which is the most basic unit of visual attention and has a variety of metrics built upon it. The fixations are defined as aggregations of gaze points over a small pixel area during a time range between 100 and 600 ms [16], while more recent research overviews of the state-of-the-art eye-tracking by both Blascheck et al. [19] and Sharafi, Shaffer, Sharif, and Guéhéneuc [20]

suggest more exact numbers by defining the area to be limited between 20 and 50 pixels and in a time range between 200 and 300 ms. The fixation duration is the second important attribute of eye movement behavior, but it is not limited to only observing fixation times in general. Sharafi et al. [20] defined an area of glance (AOG), which can either involve the whole stimuli, in other words, the whole screen, or just a selection of the areas of interest (AOI). AOIs are defined as stimuli regions on which we wish to focus our analysis according to Blascheck et al. [19], but their size and positioning is arbitrary per study case. Djamasbi [30] divides AOIs into specific and broad types. Specific AOIs are of arbitrary size and position, and they are usually designed to fit an observed element, while broad AOIs split up the whole stimuli following a certain layout and forming a grid. Such specific AOI or grid divisions increase the possible metrics such as the translation matrix behavior or the fixation visitation count. The final attribute for eye-tracking is the movement, which is described with saccades from one fixation to another. Saccades have a rapid nature with time ranges between 40 and 50 ms associated to them. By combining the saccades and the fixations in order, we can form and observe the path of eye movement called the scanpath, which is another basis for producing metrics[20]. Sharifi et al. [20] group a variety of metrics into metric groups based on fixations, saccades, scanpaths and eye information. Examples of eye measurements are the pupil dilation and the blink rate, where the first metric indicates changes in the mood or attitude, while the latter with lower value describes the high amount of workload and fatigue for higher values [19, 20]. On the other hand, Fu, Noy and Storey [31] group the metrics based on their effect: measures of search, information processing, cognitive workload and speed.

Kurzhals et al. [15] mention visual analytics as an informative alternative approach to processing the eye-tracking results. However, they also warn us of a gap between the static and the dynamic stimuli in regards to the eye-tracking metrics since some have a more static nature, which can effect the visualization as well as the metric quality. They continue that studies

using eye-tracking become even harder to process if the stimuli is in a form of an interactive application. Kurzhals and Weiskopf [32] presented a possible solution for observing results on dynamic stimuli, where an innovative but complex visual analytics method in form of space-time cube visualization is designed to target dynamic stimuli as shown in Figure 2.5. In regards of done user based studies with dynamic stimuli research, Ho, Yeh, Lai, Lin and Cherng [33] evaluated human perception of the dynamic 2D flow. Herman, Popelka and Hejlova [14] also researched some dynamic stimuli during their evaluation of a dynamic 3D geovisualization. While Moacdieh, Prinet and Sarter [34] researched the use of an interactive flight simulator with eye-tracking.

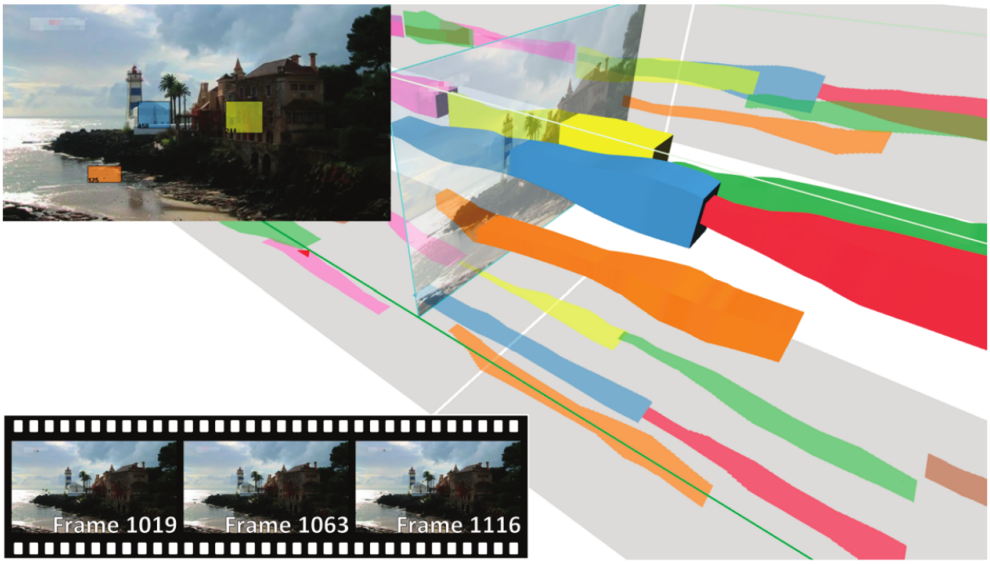


Figure 2.5: Kurzhals and Weiskopf [32] developed a space-time cube approach for visually analyzing eye tracking results for observing dynamic stimuli such a video recording. The image taken from their paper shows an example of examining results on a video stimuli with multiple AOIs, which are presented as snake like 3D lines with thickness relative to the fixation amount. By projecting the lines onto the cube's sides an attempt to simplify the comparison is made.

Chapter 3

Method

Based on previous work, see Chapter 2, generating interactive explosion diagrams with auto-positioned labels seems a trivial problem. However, previous solutions have assumptions not compatible with exploded views and need adjustments. The main problem is the assumption of static and non-deformable objects. In most CGI, AR or VR applications with previous label placement solutions, the only change was in the camera pose. Changing only the camera pose but not the 3D model itself points to quite stable label layouts. The wording "stable layouts" is used due to the nature of the problem, where the label can not move in its previous direction or even inside its local area during a label update because of complex 3D model topology hiding its next step or causing an overlap. The stability is only an empirical measurement of how huge the jump between the previous and the new label position is if it is still near the previous area. In cases where label updates include position changes so small and smooth that they do not distract the observer, such generated layouts are regarded as stable. Unstable layouts, on the other hand, are those that most updates force to huge jumps, e.g. mirroring the label over the 3D model, which causes distraction during label position transitioning.

Exploded views change the 3D model topology, which in turn changes optimal label positions and local area properties, e.g. the amount of 3D model parts in the area. Depending on the explosion direction, its magnitude and

blockers on the way, the optimal position or local area of a label can change drastically. Drastic changes naturally result in a lot of big label jumps. To prevent this from happening, existing algorithms must be adjusted to gather information about the deformations and motions that need to be added into consideration during label positioning. In case of exploded views, enough information is already provided by explosion directions and blockers in their way. While moving labels and their referred exploded part as a group can already improve the result, it does not solve the problem of deformation.

Our proposed solution of **clustered hedgehog labeling** is quite a straightforward extension of the state of the art hedgehog labeling [8]. Hedgehog labeling projects labels onto a label plane that is parallel to our view plane. The plane and labels on it are used inside the floating label algorithm, which generates a force field of attraction and repulsion forces. During updates, it tries to update label positions using extremes in the force field. First, it searches the local area, and in case of initialization or other triggers, e.g. leader line crossings, it switches to global force field search. Further information about the process will be discussed in Section 3.2. The deformations and motions during explosion simulations in an exploded view cause the topology to spread more across the screen than in the initial state. The change in topology results in quicker changes of the force fields, which again causes the reinitialization triggers to be triggered more often. The final end result equates to an unstable label layout. While we could try to create new force fields based on the deformation and motion information, we found a much simpler solution that simply encapsulates the hedgehog algorithm instead of changing its core too much. The proposed clustered hedgehog labeling does a clustering of labels based on movement and deformation information after any explosion simulation, while using a single cluster when in initial state. The generated clusters each have their own label plane and can be understood as multiple hedgehog labeling algorithms running at the same time, which for the case of one cluster results in traditional hedgehog labeling. Since the clustered elements in a cluster are close by proximity and explode

in a similar fashion, they trigger less reinitialization events. Furthermore, a triggered reinitialization can even reinitialize the elements in only one cluster instead of all clusters. Less triggers and locally limited reinitialization should improve the global stability of the label layout.

3.1 Floating Labels

Hedgehog labeling [8] uses the concept of potential fieldsor, as we call them, force fields. Before extending the state-of-the-art algorithm, we need to understand its core component first.

Hartmann et al. [9] introduced force fields into the field of label placement with their floating labels algorithm. They defined a potential field that uses label and model contour repulsion forces paired with attraction forces towards the center of a model part to push and pull individual labels in a spring-like system, which can be observed in Figure 3.1. While the paper defined five forces that are combined into a single potential force field for label placement, we added an additional force for penalizing label positions overlapping with leader lines:

A Attractive forces between model part and label centers,

B Repulsive forces at model boundary / contour / silhouette,

C Repulsive forces between the label and its non-referred model parts,

D Repulsive forces between label and view border,

E Repulsive forces between label and other labels,

F Repulsive forces around other label leader lines.

During a label position update run, the force fields formed with forces A-D remain constant regardless of label positions and therefore form the so called **static force field**. After adding forces E and F to the static force field, we arrive at a **dynamic force field**. In floating labels, they first positioned

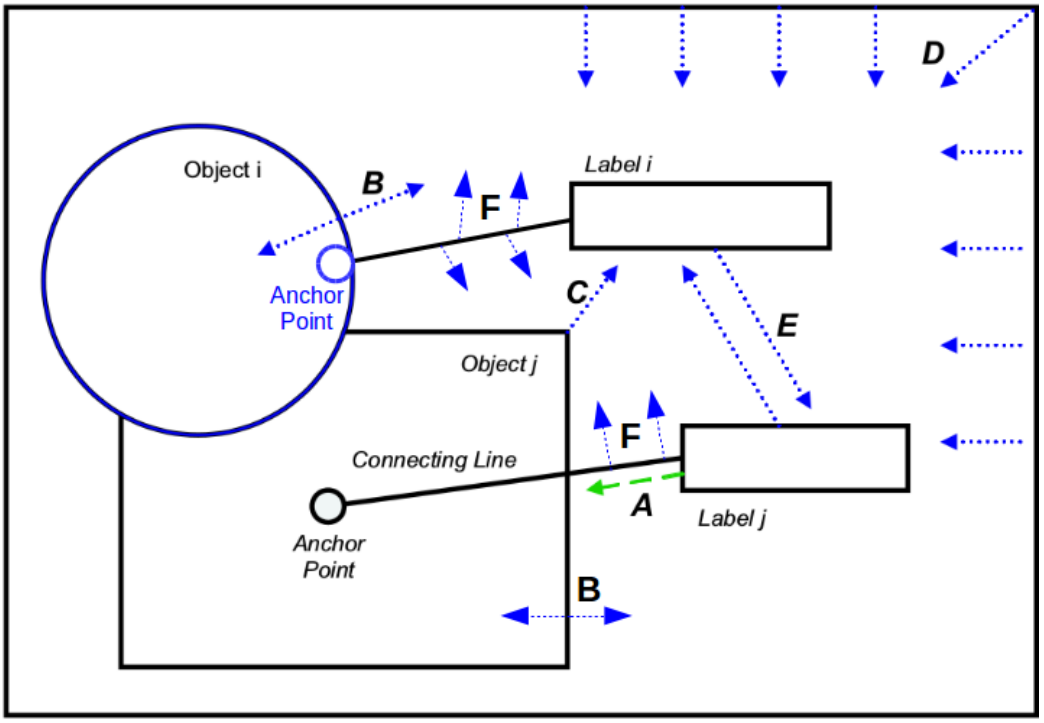


Figure 3.1: A diagram representing the individual floating label forces. We retrieved it from the work of Hartmann et al. [9] and modified it to include our leader line repulsion forces. Note that an anchor point may not be in the center of the silhouette.

the labels according to the extrema in the static force field. Based on the given initial positions, they calculated the dynamic force fields for each label and repositioned them to their final positions.

Since they defined floating labels as 2D label placement for static images, or in case of interactive applications as post processing the rendered image, we are required to know to which object inside the image each pixel belongs to. The term object is too abstract for 3D labeling on exploded views, so we rather use the terms model parts or meshes, where we mean the disassembled meshes. Hartmann and his colleagues proposed color coded images, where predefined or detected image sections are colored in unique colors to present individual objects to the domain experts for them to pair annota-

tions with individual colors. Such a principle can be easily translated to 3D labeling, where individual meshes only need to be colored uniquely, while label information just needs to be linked to the meshes.

Another crucial piece of information needed for force calculation is the **interia**, which is the 2D equivalent of an anchor point. It can be described as the visible pixel closest to the object center. It is important since we wish for leader lines to always point only towards their referred object's pixels and towards any other pixels. Hartmann's group proposes the use of a thinning algorithm to thin out the object pixels until they all disappear, and then use the last present pixel as the interia.

3.1.1 Force definitions

Following the force definitions from the paper by Hartmann et al. [9], we can express forces F_x, where x corresponds to the force letter seen on the list and in Figure 3.1, as equations of c_X weighted distances in regards to pixel `p`, label `L`, leader line `LINE`, object `O` and image `I`. The attraction force or $F_A(p)$ is defined for pixels not color coded in the observed object's color as a weighted ratio of the distance between pixel `p` in interia pixel and the distance between interia pixel and the farthest pixel from interia, while we use zero for the pixels with their color corresponding to the color code of the observed object :

$$F_A(p) = \begin{cases} c_A \frac{distance(interia,p)}{\max\limits_{p_{max} \in I}(distance(interia,p_{max}))} & , p \notin O \\ 0 & , p \in O \end{cases} \tag{3.1}$$

The contour repulsion forces $F_B(p)$ are defined to be equal to the weight for pixels close enough to the silhouette of the whole object and zero otherwise:

$$F_B(p) = \begin{cases} c_B & , distance(p, p_{silhouette} \leq inf_{silhouette} \\ 0 & , \text{otherwise} \end{cases} \tag{3.2}$$

In regards of repulsion forces from objects different from the observed object $F_C(p)$ we use the given weight for pixels with code corresponding to a color code not belonging to the observed object, while the pixels with colors to color codes of either the background or the observed object we use zero:

$$F_C(p) = \begin{cases} c_C & , p \in O_i \wedge O_i \neq O \\ 0 & , \text{otherwise} \end{cases} \tag{3.3}$$

Repulsion from the view border $F_D(p)$ is defined for pixels at a distance less or equal to the provided view border influence inf_{border} as a weighted ratio of the shortest distance between view borders and pixel and inf_{border} and zero otherwise:

$$F_D(p) = \begin{cases} c_D(1 - \frac{distance(p_{border}, p)}{inf_{border}} & , distance(p_{border}, p) \leq inf_{border} \\ 0 & , \text{otherwise} \end{cases} \tag{3.4}$$

The static force field is simply defined as a sum of attraction forces and the maximum between $F_B(p), F_C(p), F_D(p)$ forces per pixel:

$$F_{STATIC}(p) = F_A(p) + \max(F_B(p), F_C(p), F_D(p)) \tag{3.5}$$

Label repulsion forces $F_E(p)$ use just their prepared weight for pixels inside of the label area and zero otherwise:

$$F_E(p) = \begin{cases} c_E & , p \in L_i \wedge L_i \neq L \\ 0 & , \text{otherwise} \end{cases} \tag{3.6}$$

The final separate repulsion forces of leader lines $F_F(p)$ were defined as an extension from our side to reduce the amount of labels overlapping with leader lines. We defined it using the provided weight for pixels, whose distance to any leader line not belonging to the observed object is less or equal to the provided leader line influence inf_{line}, and as zero for other cases:

$$F_F(p) = \begin{cases} c_F & , distance(LINE_i, p) \leq inf_{line} \wedge LINE_i \neq LINE \\ 0 & , \text{otherwise} \end{cases} \tag{3.7}$$

While the the static force field uses only the maximum between repulsion forces inside a pixel, we defined the dynamic force field to penalize pixels with repulsion forces of both labels and leader lines by subtracting both from the static force field:

$$F_{DYNAMIC}(p) = F_{STATIC}(p) - F_E(p) - F_F(p) \tag{3.8}$$

3.2 Hedgehog Labeling

Hedgehog labeling's biggest contribution in comparison to previous work was the transition from 2D label placement in the post-rendering phase to 3D label object positioning before the rendering phase [8]. To achieve it using the 2D based floating labels algorithm [9], Tatzgern et al. introduced label planes [8]. They also introduced label freezing and redefined the separation of the label position initialization and the update, all while preparing a solution for leader line crossings and overlaps based on other previous work [8, 10].

3.2.1 Label Plane

While using the 2D floating labels algorithm in 3D space is easily achieved by projecting the current view to a texture, it was proven that the 3D orientation of leader lines, on which hedgehog extends the labels, cannot be arbitrary for prevention of leader line crossings, label overlaps and similar issues. Their proposed solution was to group labels onto planes instead of using the whole 3D space as shown in Figure 3.2. Good results were achieved by defining a plane parallel to the view plane and which intersects with the mesh and potentially its center. Since the planes were defined to be used as billboards for placing labels, we call them **label planes** [8].

Mathematical planes had many geometrical definitions proposed, among which we chose to use the description that uses a single point on the desired plane and its normal vector. Since the label plane was defined to be parallel to the view plane, we used the view vector, which is the vector pointing

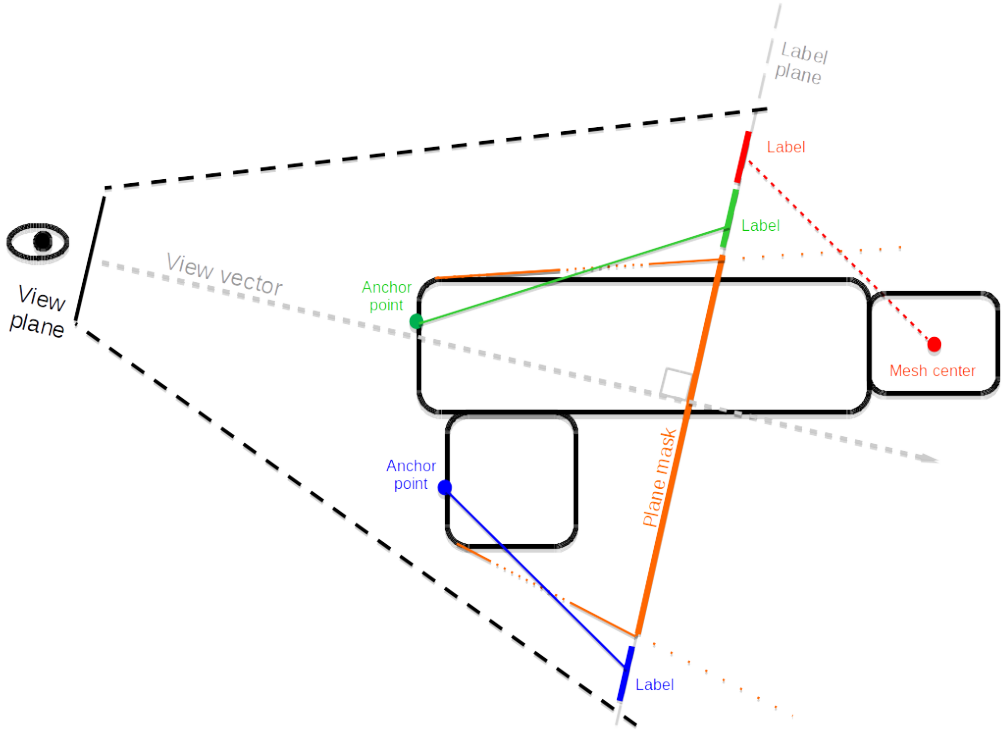


Figure 3.2: The label plane concept that enables us to place all of the labels at same distance from us for better readability. Note also the difference between a model part visible and invisible to the observer and their chosen anchor point. In case of invisible parts, we just use their mesh center until they become visible.

from the camera to the camera's focus point, as the base for the normal, since it is by definition perpendicular to the view plane and therefore also perpendicular to the parallel label plane. By using the view origin and the intersection of the view vector and a plane, we can define the view vector as a difference of vectors:

$$View_{vector} = Plane_{position} - View_{origin} \tag{3.9}$$

While projecting the labels on the defined plane will move all labels into our view, it has not yet prevented any occlusions or overlaps at this point of

definition. Therefore, we defined an occlusion mask, which is a projection of the current view onto the label plane, to mark potential occluded-label areas of the plane. While movement of the labels based on the free areas inside the plane was possible, we instead used the projection mask as the input image for the floating point algorithm.

3.2.2 Label Freezing

Temporal coherence research [10] shows that users perform tasks better with labels that are placed statically and do not update constantly, therefore we had to avoid too frequent label repositioning since we want to enhance temporal coherence and user experience for 3D model presenting. We started by defining **calculation frozen** and **unfrozen** states to freeze the labels in place until one of the threshold conditions for unfreezing are met. The frozen state was designed to be the default state and is always used after a label update finishes, resulting in no update in the poses of neither the label plane or the labels until a condition is met. With this, we redefined hedgehog labeling into five steps:

1. Produce mesh model, labels and label plane for label cluster,
2. Project view into a color coded texture,
3. Force field calculations,
4. Find new position for each label based on dynamic force field,
5. Freeze plane,
6. Leader line extension check,
7. If unfreeze condition met, align plane to view plane jump to step 2, otherwise jump to step 6.

While the frozen state was created to disable label placement with force field calculations, it was never meant to allow issues such as labels being

hidden and positioned behind or in front of the model silhouette due to the change in view not yet hitting the thresholds inside the unfreeze conditions. For handling such situations, we defined a repeated check that would trigger leader line extension in case of an issue occurring. Leader line extension was already proposed together with 3D labels by Tatzgern et al. [8].

3.2.3 Leader Lines Crossing Prevention

Another change done inside of hedgehog labeling in comparison with floating labels was in regards of how to handle the prevention of leader lines from crossing each other. In the case of floating labels [9], it was solved by adding additional constraints to label positioning, as seen in Figure 3.3. While such a solution may prevent certain problems, it also limits the possible label layouts. Tatzgern et al. [8] rather left it to the force fields to decide the layout without additional constraints, and then checked for leader line crossings and reordered labels producing a crossing to resolve it. The idea was borrowed from an earlier paper by Ali et al. [25]. Following their footsteps, we defined the leader line crossing check as a double loop over label pairs, where we

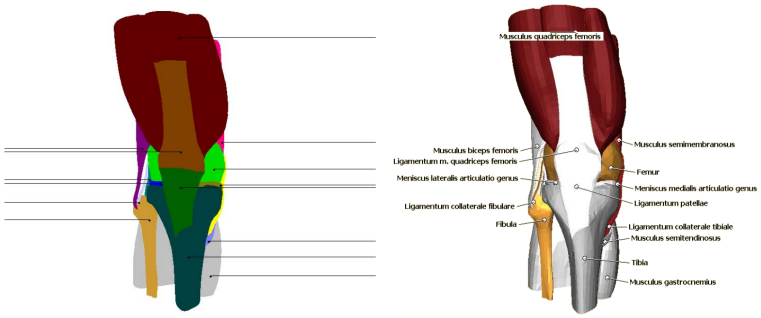


Figure 3.3: The presented floating labels followed the illustrator guidelines for defining label positioning constraints. One of such constraints was the user preferred axis as seen in the left image. While it produces cleaner results, as seen in the right image, it has a limit to the possible layouts it can produce. The images are taken from the paper written by Hartmann et al. [9].

checked for leader line crossings with an analytical method based vectros, which is described in Algorithm 1.

Algorithm 1 Line Intersection Check

Require: Vector3 LabelCenter1, AnchorPoint1, LabelCenter2, AnchorPoint2;

1: Vector3 $anchorLine1 = LabelCenter1 - AnchorPoint1$;
2: Vector3 $anchorLine2 = LabelCenter2 - AnchorPoint2$;
3: Vector3 $anchorPointDifference = AnchorPoint2 - AnchorPoint1$;
4: // cross products are zero when same or reverse direction of line
5: // as an exception they are 1 if direction equals 0
6: Vector3 $directionCross = AnchorPoint2 \times AnchorPoint1$;
7: **if** $\|directionCross\|^2 > 0$ **then**
8: // line directions are not parallel - find point of intersection
9: $multiplier = \frac{(anchorPointDifference \times anchorLine2) \cdot directionCross}{\|directionCross\|^2}$;
10: Vector3 $intersection1 = AnchorPoint1 + multiplier * anchorLine1$;
11: Vector3 $intersection2 = intersection1 - AnchorPoint2$;
12: // intersection is valid only if it is between the anchor point and label center
13: // if intersection is valid we have a leader line crossing - returns TRUE
14: **return** $multiplier > 0$ && $multiplier < 1$
&& $\frac{intersection2}{\|intersection2\|} \cdot \frac{anchorLine2}{\|anchorLine2\|} > 0$
&& $\|intersection2\| \leq \|anchorLine2\|$;
15: **end if**
16: // lines directions are parallel - do a check if lines collinear
17: // true if anchor points not the same, however lie on same line
18: **return** $\|anchorPointDifference\|^2 > 0$
&& $\|anchorLine1 \times anchorPointDifference\|^2 == 0$
&& $\frac{anchorPointDifference}{\|anchorPointDifference\|} \cdot \frac{anchorLine1}{\|anchorLine1\|} > 0$
&& $\|anchorPointDifference\| \leq \|anchorLine1\|$;

3.2.4 Local Extrema Search

Until now, we had always calculated new label positions from scratch based on global texture search. This was but a waste of effort from previous calculation since there should not be much change in label position between calculations. Therefore, we divided the calculation part into an update and a re-initialization run. The update run was corrected to use the previous

label layout in order to calculate the maximum in the area around the previous label position. On the other hand, the re-initialization run remained the same as the old update running a global search.

The goal of introducing local extrema was to mostly have update runs and to only use initialization during the first run and during leader line crossings. Textplosion was thus redesigned to match the following step order:

1. The unfreeze condition for calculation start was met.
2. Generate all static force fields for current camera pose and jump to step 10 if it is an initialization run.
3. Try the previous label layout to produce dynamic force fields for local extrema search.
4. Define search bounds based on label position, dimensions and user provided padding.
5. Clear force sum array to prevent old data leaks.
6. Generate a new force sum field and do a local extrema search inside search bounds.
7. Use the found extrema position as new label position.
8. Do leader line cross check on new label positions.
9. If any leader line crossing occurred, stop the check and skip to step 10, otherwise skip to step 13.
10. Ignore the previous layout and do a global force sum and global extrema search.
11. Do a leader line cross check on new label positions from global calculations.
12. If a leader line crossing appeared, switch label positions.
13. Change to frozen calculation state.

3.3 Clustering

While Section 3.2 described hedgehog labeling introducing label planes as in plural, we only mentioned the inner workings with the assumption of a single plane to simplify the theory. In the paper they also described a situation with multiple label planes positioned at equal distance between each other according to the view vector and limited with the model's bounding box dimensions. Such a setting would allow a quick implementation and some improvement in results for static models, however it would not work well with exploded views, since the equal distribution of the model's bounding box would only take into account the global information of all positioned parts and disregard any local relationships as well not know anything about the explosions.

Therefore, clustered hedgehog labeling introduces multiple label planes with a twist. Instead of first dividing the space and later project the labels onto the created planes by closeness, we instead propose to first cluster all the labels based on their referred parts and explosions and afterwards divide the space according to the label clusters by using each cluster's average of mesh centers to position the planes into space as seen in Figure 3.4. Since we decided on using mesh center averages as the points defining the planes together with the view vector, we needed to slightly adjust the original plane definition, since an arbitrary average will rarely fall onto the view vector. Correctly positioned planes allow us to use the force field calculations as defined in Section 3.1 and also produce as a side benefit a depth effect on labels due to their closeness in space to the their referred meshes. We regard it as a benefit, since the observer can easier connect elements at similar depths than depths far apart.

Using mesh averages directly would have resulted in the label planes disaligned with the model where the average of mesh centers is positioned relatively far from the view vector. From the same figure we could also see that both the plane position and the cluster center are part of the same plane, which means that we can find the actual plane position by finding the

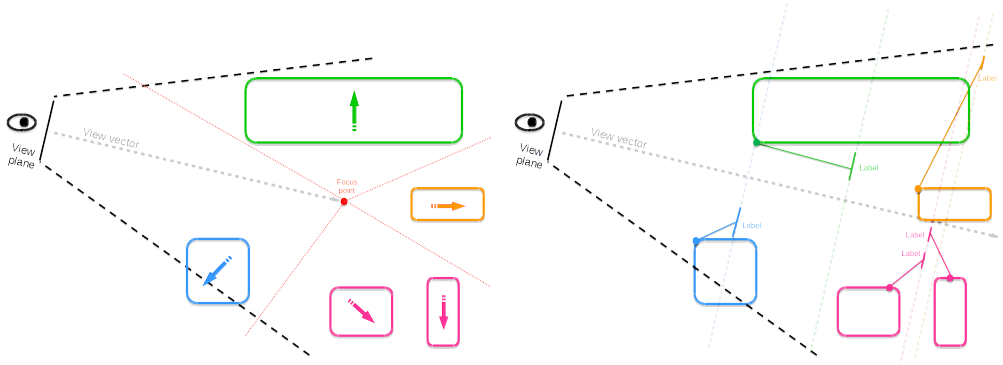


Figure 3.4: The left image shows meshes being clustered into four clusters based on their current position, movement and distance from the focus point. The right image shows how multiple label planes are positioned over cluster centers.

intersection of the view vector and the plane positioned in the cluster center, and simply translate that same plane into the intersection.

While we tested different clustering approaches, the one that showed the most promise and flexibility was the standard k-means clustering. The reason for focusing on k-means despite it being over five decades old and there being many alternative good algorithms was the yet uncharted area of clustering the exploded elements of exploded views in 3D space. No clustering algorithm has yet shown dominance over other algorithms across application domains, and most clustering algorithms, including k-means, are valid in most cases [35]. Having no knowledge of which data could be used as good clustering metrics as well as to which application area we should relate the most, we decided on the simple but valid k-means, which could take a vector of arbitrary length of metrics as input to cluster on. Therefore, we could focus more on the metrics research. The k-means algorithm steps we used were the standard ones with the addition of cluster centers besides its centroids:

1. Generate random cluster centroid values for all K clusters, where K

was provided by the user,

2. Place labels into cluster with the closest value based on squared euclidean distance to cluster centroids,

3. Average label values inside cluster to determine new centroids,

4. Check for labels switching clusters. If any switch happened, jump to step 2,

5. Use the average of model part positions as cluster centers.

Regarding good data to be used as clustering metrics, we found the following ones interesting after trial and error:

Direction The normalized explosion direction was chosen since same direction elements will be closer on the long run.

Position We added the mesh center to find observable groups in 3D space.

Changes The difference between the initial state model part's mesh center and the current position tells us the progress of an explosion. The explosion progress is important for single part explosions since the bigger the difference in progress is, the more apart the model part will most likely be in an exploded view.

Focus distances The differences between the individual coordinates of the view's focus point and mesh center is another way of determining the explosion progress, and it should be better for cases of mirrored directions. It tries to define the progress by predicting that closeness to the focus point means low progress.

Chapter 4

Implementing Textplosion

While **clustered hedgehog labeling** is a simple and straightforward concept, its implementation was a long and hard path. To differentiate it from the concept and the implementation, since there were implementation limitations and workarounds, we call the implementation **Textplosion**, which describes the unification of labels and exploded views. Textplosion implements the proposed algorithm as an Unity3D application. It becomes our front-end application that takes 3D model information from multiple OBJ and MTL files, and combines it with the exploded view and label meta-data files. Developed as an interactive real-time GUI, it allows the user to interact with annotated 3D model explosion diagrams.

Regarding the models and their meta-data, we took a helping hand provided by Graz University of Technology's Institute of Computer Graphics and Vision. We used the output files provided by the exploded view generator, which was implemented by Kerbl et al.[7]. Since we only provide 3D models to it and retrieve the produced exploded view meta-data files, we think of the system as our black box back-end system, of which we know the general inner workings but not the details, as explained in Section 4.2.

During development, we needed a way to test our solution, however no standardized 3D label placement dataset existed at the start of our development. Therefore, we created our own dataset for testing during development

and for use in the user study evaluation. Since modeling 3D models requires a lot of time and artistic talent, we did not create our own models, but borrowed existing ones. Naturally, copyrights also come into play with 3D models, and they can be as equally or even more expensive than the actual computer software, depending on the quality. Therefore, we found our dataset models inside open-source 3D model libraries, such as GrabCAD and Blenderswap. See more details regarding the dataset in Section 5.1.

4.1 Development in Unity

Unity is a popular and wide-spread game development platform, which also offers a limited but free version for personal development. While it is primarily a platform for developing games, it is not limited to that. Development of other CGI applications can also be simplified by using its rich and well documented library of built-in functions and profilers, as well as its good support community and lots of open-source resources. Another advantage is the simplicity of application building for different kinds of builds, which works in favor of the future plan of deploying the application as a web application and potentially a mobile one. Using it as our development platform simplified the overall development since we could skip basic phases, such as coding the rendering of 3D CAD models or the base of an interactive GUI.

The development environment may have simplified our starting steps, but it also limited some of our options. Such a case was GPU based computation, which was needed for us to speed up the performance speed of Textplosion. While Unity has support for GPU computation, it is limited to compute shaders. While approaches like CUDA programming could boost up our performance, it would at the same time limit us to NVIDIA based GPUs. During development, our guideline was a standalone application with minimum cross-platform limitations and minimizing cross-language calls. Therefore, as our main language we only needed to use C sharp with DirectCompute for GPU computing besides the CG and HLSL shaders.

4.1.1 3rd Party Unity Assets

Unity uses a concept of assets to define the set of scripts, models or other components that are available to the project, which are also compiled during a build. With its large community and the rich asset store, one can quickly find open-source or free-to-use licensed assets, which speeds up development. Some of these were also used during our development:

Runtime OBJ Importer , provided by AARO4130 at `https://assetstore.unity.com/packages/tools/modeling/runtime-obj-importer-49547`, solved our problem of loading OBJ files at runtime with file paths to the script.

SimpleJSON.cs , provided by Bunny83 at `http://wiki.unity3d.com/index.php/SimpleJSON`, is a simple JSON parser for our self-defined JSON meta-data files. This 3rd party asset is needed due to the built-in JSONUtility component not covering arbitrary JSON files.

TobiiPro SDK is the free SDK provided for programing TobiiPro code for eye-tracking experiments. We used it during user experience tests for storing eye data and other generated experiment data.

4.1.2 Unity's Cross-platform Notes

Unity was primarily developed for Windows and later on also became available for Mac OS and Linux, but for Linux it is still in experimental beta stage and not an official release. Our main development OS was Linux, while the experiment environment was Windows. While it was decided to go in this direction for faster development and due to cross-platform limitations, it also had to remove any cross-platform issues. One of the issues were computer graphics APIs, where Windows uses DirectX11/12 and Linux uses OpenGL. While mappings of functions from one API to the other are done, there naturally remain some shortcomings, for example the way it handles out-of-bounds exceptions or undefined variables on a compute shader.

4.1.3 Differences Between Coordinate Systems

When defining 3D coordinate systems in mathematics, the issue of the orientation of axis Z arises. It is basically a choice between a left-handed and a right-handed coordinate system. While in simple calculations this only involves a correction of equations, in CGI code it easily becomes a headache if we do not keep in mind which was chosen and why. The chosen system does not only define the direction of axis Z but also the direction of the positive and negative rotations. Naturally, both are correct, but none is the best choice between the two, since each has it own perks per problem [36].

While both DirectX and OpenGL allow use of either system, each gives priority to a different one. This can be observed in the documentations, which are used for tutorials and other teaching material. In the end, each set of documentation becomes the de facto rule for its API. This escalates further with programs built on top of them that directly imply which coordinate system should be used. If we look at the process of switching between coordinate systems, we can also observe how tricky the APIs themselves make it for the programs built on top of them. DirectX offers separated function calls between the coordinate systems while OpenGL allows loading of transformation matrices.

Inside Textplosion we deal with both types of coordinate systems since the back-end system works with OpenInvetor 3D development framework. This framework uses OpengGL and therefore outputs right-handed matrices and vectors inside the meta-data files. Unity is left-handed due to its base being built on DirectX.

4.2 Exploded View Base System

The main goal of the thesis was to produce an interactive GUI for presenting annotated 3D model exploded views. Because of this, we started by building a basic system for the loading and processing of meta-data and 3D geometry as well as handling user camera and explosion simulation interactions.

Explosion simulations or explosions, when talking in Textplosion context, are dynamic movement animations that move the model part and its corresponding label in the direction provided by the meta-data files and user input.

Since we were provided with a back-end system for generating explosion views, we started building the system under the assumption that in the future the back-end and front-end will be combined or at least connected via a web API. Based on this assumption, we decided to build a hierarchy of structs that corresponds to main file types of the back-end meta-data files and the references inside them that already implied some level of hierarchy. Since the thesis proposal was to add labels on top of the existing back-end, naturally there could be no label information generated without our input. Therefore, we extended the back-end outputs with a new label format meta-data file, which simply uses the existing ids to link to the desired strings of text. In order to simplify the generation of label files, we also added an option that generates a new file based on the parsed structure, where we use the model source filename as the annotation for each part. Details on the data hierarchy parsed from given input meta-data files are explained in Subsection 4.2.1, while the back-end files and their contents are described in Appendix A.

At the time of writing this thesis, the back-end system was still far from server deployment with an API for sending arbitrary 3D model formats. For the back-end to process an arbitrary 3D model, one had to manually cut up the desired 3D model and store the parts in a single OpenInvetor .IV format file. This file had to be clean of cut-up artifacts such as non-manifoldness due to edges or faces not closing up the model. Due to security concerns, one needed to forward the OpenInventor cut-up model to a middleman, who manually started the processing and after some time returned the generated meta-data files as well as the separated OpenInventor and OBJ files. Due to the low flexibility of the system, we created an alternative way of using it so that the back-end system access is not needed. We decided to create simple

JSON formated meta-data files, which can be observed in Appendix A. Since the inner data hierarchy is the same regardless of input type, we also added an option to store it in numerical back-end format.

The system was designed to parse the meta-data files provided by the run parameters. This causes Textplosion to generate an inner meta-data hierarchy and from there to import the actual model OBJ files. At runtime, the loading of OBJ models was achieved with the help of Runtime OBJ Importer, which loads all of the model parts after parsing is done. Since not all model parts are unique, we sped up the process of loading by creating prefab copies from the loaded unique model parts, and just changed TRS values according to the settings in the data hierarchy. On the other hand, label generation was achieved with a label prefab consisting of a TextMesh, a background Quad and leader line LineRenderer. We attached a script to the label prefab. When started, it corrects the label dimensions according to the length of the string set for the annotation. It also stores information such as anchor point reference, which enables leader line drawing during updates. Inside the label script, we also included many other helpful functions. One of them is the option to auto-rotate the label towards the camera by aligning its front axis to the vector that extends between the camera and the label or the leader line crossing checks. The checks project the anchor lines onto the view plane and compare them with other leader lines or the edges of a label background edge following algorithm 1.

4.2.1 Parsed Data Structure Hierarchy

For easier and more effective work with the meta-data files, we have constructed a hierarchy of structs. We started with the root struct Assembly, which represents the data from the assembly file header and its pointers to other meta-data files:

3D model format determines if the model can be read by the system.

Unique count tells the amount of unique model parts.

Unique groups is a list of the next level AssmlyGroup struct, which contains part model and its copy data.

Label information is a list of the next level MetaLabel struct, which contains all label information.

The next level of hierarchy added was AssemblyGroup, which stores individual assembly file references to geometry and copy specification files. Inside specification files, we retrieve the information needed for the explosions directions, blockers and TRS transformations. AssemblyGroups contains:

Part name is the back-end provided part name.

Geometry path gives the location of the corresponding OBJ file.

Specification count stores the amount of different specification files that form movement groups, where copies move in the same direction.

Specification list is a list of the next level MetaLabel struct, which contains all specification file information.

While AssemblyGroup was defined to hold information about unique parts to load, the Specification struct was designed to represent the movement groups of its copies. The elements of the movement group were named copies for simplicity. Movement groups have two possible directions of movement: **singular** and **cascading**. The singular directions are just 3D vectors, while the cascading ones are a set of 3D vectors that are used one after another until collision. The back-end system does not tell which movement groups have the least collisions with the current movement group, it only provides the set amount in the back-end system of optimal directions. Therefore, we added to Textplosion a keyboard and mouse interaction that provides information of the set direction by an arrow as shown in Figure 4.1. As a pair we also added an interaction to shift to next direction in the list to create a simple way of switching between specified directions during a Textplosion run. Including the directions a Specification contains:

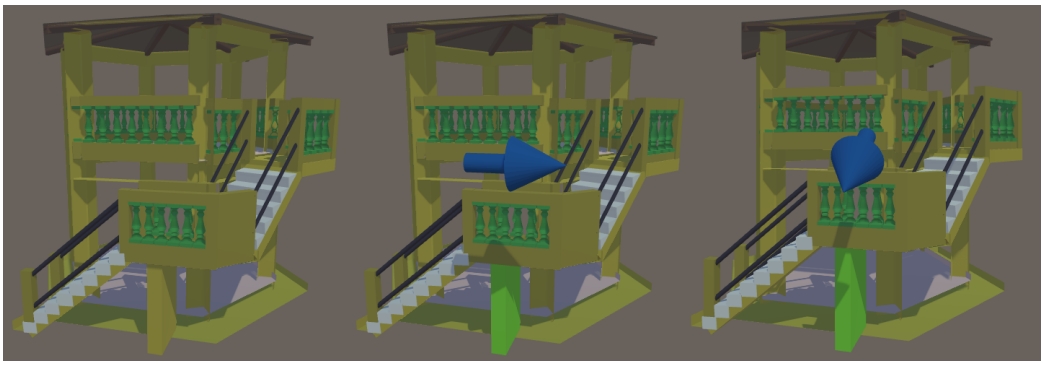


Figure 4.1: Examples of switching `kiosco` model's support part directions during run with direction arrow interaction.

Copy count provides the amount of prefab instantiation needed for the current movement group of geometry referenced by the parent AssemblyGroup.

Copies list is a list of the next level SpecPropagation struct, which contains all copy specific information.

Singular direction count provides the amount of singular directions.

Singular direction list is a list of all possible 3D singular directions.

Cascading direction count provides the amount of cascading directions.

Cascading direction list is a list of sets of three 3D directional vectors that follow in order of position.

The last exploded view related struct in hierarchy, SpecPropagation, was created to contain each copy's details. While parsing the id and matrix of each copy, it also recalculates the translation, rotation and scale from the TRS matrix provided in back-end source files. This was needed since Unity works with vectors instead of matrices in its inner representation. We also prepared a boolean parameter in its constructor to determine if the input source is stored in the right-handed system, which triggers correction to the

left-handed system of the values in the matrix as well as in individual vectors. SpecPropagation contains:

Copy id is the integer id provided by the back-end system that corresponds to the index in the blocker string inside direction specification files.

Transformation stores the TRS matrix provided and possibly corrected to the left-handed system if set so.

Translation gets extracted from the last column of the TRS matrix.

Rotation stores the quaternion extracted from the TRS matrix.

Scale gets extracted from the diagonal of the TRS matrix

Singular blocker list is a list of copy ids that potentially collide with this copy. The index in the list corresponds to the index in the Singular direction list from the Specification struct.

Cascading blocker list is the cascading direction version of the singular blocker list.

The root struct Assembly was designed to hold another struct besides the exploded view information, namely the label information. It is a simple struct called Metalabel and contains:

Copy id corresponds to the copy id inside the SpecPropagation struct. It tells us to which part of the model the label is connected.

Initial offset is used on the initial load before the positioning algorithm is run and simply adds the offset to the mesh center position of the corresponding model part.

Annotation contains the text annotated to the part.

4.2.2 Exploded View Interactions

The HCI inside Textplosion were designed to provide user control over the camera and the model part explosions via mouse and keyboard. The camera control interactions were limited to zooming and rotation on a sphere defined by a set focus point as shown in Figure 4.2. While other camera control interactions such as direct camera placement would be possible, we limited ourselves to those we deemed simple to use and essential for presenting exploded 3D models to avoid overwhelming the user with too many options.

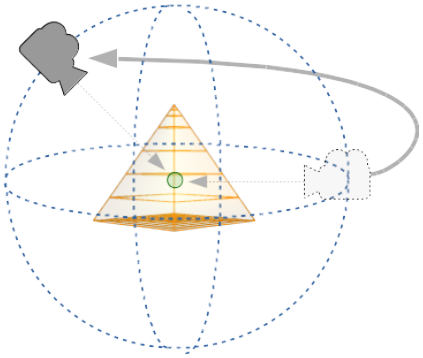


Figure 4.2: Texplosion builds camera related interactions on a model-centric concept to prevent the object going out of view except in case of extreme explosion interactions.

We set Textplosion to determine the camera focus point after loading the whole mesh with all prefab copies by selecting the center point of the bounding box encasing the whole mesh. We aimed to ensure that the whole mesh in its initial state is visible to the user at any moment. In order to achieve this, we limited the camera pose interaction to spherical rotation around the focus point with the view vector always orientated towards the focus point. While the topology changes due to explosions taken into account by the zooming interactions changing the radius of the camera's sphere of movement, the initial radius was set to be calculated with simple trigonometry using the mesh's bounding box measurements and some padding. Both camera con-

trols were implemented in single script TexplosionSetup.cs, which also hosts other core system code, making it the core script besides the load scripts.

Explosion interactions, on the other hand, are a result of multiple different scripts. During model load, these scripts are appended to individual mesh or label objects for each part of the cut up exploded model. The HCI essential script MouseEvents.cs was constructed to enable mouse interactions of hovering, clicking and dragging. At the same time, the script became a trigger for information events as well as a delivery system of user orders such as part explosions for other scripts. The information events include the direction indicator explained in Subsection 4.2.1 and glowing events. Previous studies [22] showed user experience improvements by adding glow to important elements in GPS navigation applications, therefore we also used it as an indicator of information in Textplosion. We decided that the main Texplosion use of glowing elements was the indication of the selected element, since it was our most basic and informative idea. The model part selection in Textplosion is set to trigger when the mouse hovers over a model part or label, which triggers both the model part as well as its paired label to glow. An alternative incorporated useful glow effect was the indication of potential blockers in the set direction of the currently selected model part of an exploded view.

Another HCI important script loaded to all model parts on load is the abstract ExplosionBase.cs script, which handles the logic for model part explosions based on the provided meta-data files. Since we wish for the labels to follow their referred exploded model parts, we store the loaded meshes and labels in a hierarchy of containers as shown in Figure 4.3 and move the containers of the mesh and label pairs during explosions.

While the back-end system provided meta-data files give information on potential blockers, the system does not provide the location or timing since this would change depending on the user interaction. We started solving the problem with basic Unity provided collision detection, however this brought problems of its own. Due to the explosive nature of the models, the model

```
Generated - Data - Group1 - Copy1 - model - Mesh1
          |      |        |       ! label - Text1
          |      |        |               ! Background
          |      |        |
          |      |        ! Copy2 - model - Mesh1
          |      |                ! label - Text2
          |      |                        ! Background
          |      |
          |      ! Group2 - Copy3 - model - Mesh2
          |                       ! label - Text3
          |                               ! Background
          ! Helpers - Planes - Plane0
                      |       ! Plane1
                      |
                      ! DirectionArrowIndicator
                     ...
```

Figure 4.3: Unity container hierarchy branches out per movement group to contain all copies of same mesh defined by the movement group. When exploding parts it is the copy level that is moved and therefore both mesh and label children move at the same time.

parts consist of mostly concave meshes, which Unity does not support since after its removal from the core PhysX library as a speed up. While convex collision detectors such as a bounding box or others could be added, they showed low performance because the parts that were initially close to each other kept colliding with the bounding boxes as seen in Figure 4.4.

We did not wish to use commercial Unity assets inside Textplosion or lose too much time developing efficient and fast collision detections for concave meshes. Because of this, we decided to use the back-end information as a trusted source, which provides us with accurate direction information that prevents the parts from phasing through each other. We rebuilt the explosion functions in order to cause them to propagate explosions towards blockers,

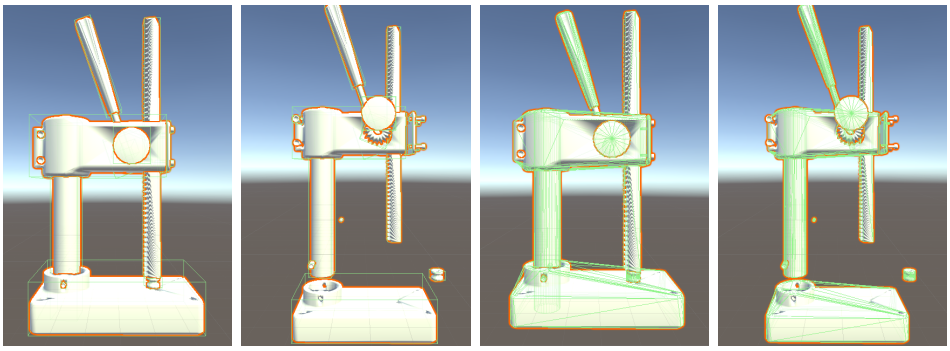


Figure 4.4: Collision detection with Unity's in-built box or convex mesh collision detectors was not possible due to the mostly concave nature of parts inside exploded views. By comparing states before and after in the images one can quickly observe the amount of concave mesh parts being high in the relatively small `prezz` model by just seeing how many parts needed to pass the bounding boxes for collision detections marked in green to arrive at their end position even though there was nothing in their path to block them.

which recursively propagate it further to their own blockers. By recursively propagating explosions, we created a system that is a workaround to collision detection, but when accurate meta-data is provided, it performs as if collision detection was added. For potential changes to collision detectors in the future, we defined the main explosion script as an abstract class, and extended it into the collision detection supported by SimulateExplosion.cs and the meta-data propagated PropagateExplosion.cs.

We defined that clustered hedgehog labeling triggers clustering after the explosion finishes, therefore we implemented Textplosion to have an indicator of explosion states for all model parts. We set it to store an initial state indicator as well as an animation-in-progress indicator, which are set during event notification calls inside the core script. Since both clustered and normal hedgehog labeling have triggers that force updates of label positions, we also needed to prevent distraction due to instantaneous jumps on updates. We handled it by transitioning the label position change over multiple frames.

This resulted in a label transition that is controlled by a user given duration input instead of a jumping behavior due to instantaneous updates of label positions. Due to smaller or larger changes of multiple label positions, this resulted in short term label overlaps with labels, leader lines and the model mesh. However, since it is a better solution than the alternative settings, it became a fault by design due to its merits.

Following the suggestions for exploded view interactions provided by Li et al. [21], we implemented three groups of explosion interactions: **animation**, **direct manipulation** and **riffling**. The interactions inside these groups produce different results. The fact used in our favor is that any generated explosion is actually a transition of model part position over rendered frames. We used this to implement all interactions as special calls of the basic movement logic run during updates. The implemented interactions therefore differed only in their trigger, the set direction multiplier and in the presence of propagation. We called the interactions for returning the models into their initial form reverse explosions because we designed them by using negative multipliers to reverse explosions. These, however, had their movement limited to prevent illogical behavior for exploded views. Since we defined all explosions and reverse explosions by using the same logic, we could create a system, in which explosion results can be stacked on each other, resulting in a wider variety of possible interactions by combining the basic ones.

Explosion Animation

The implemented interaction most true to the name explosion was coded to produce an animation of exploding the whole model. During the explosion updates, all model parts move the part container for a step, which is calculated based on time passed since the last frame update, an explosion speed constant, the direction of the explosion and the type of explosion animation. We designed the animation to be either forward or reverse for the duration of a key press. Which of the two animations is triggered depends on which key is being pressed by the user. Since all parts move at the same time in

their set direction, we did not need to propagate the explosion order to each blocker.

Direct Manipulation

Textplosion was designed to allow direct manipulation of individual model parts via mouse clicks. We implemented two types of direct manipulation interactions: **drag & drop** and **offset jump**. The drag & drop interaction tried to simulate a 3D version of the drag & drop icon produced on OS desktops, while offset jump was added as a simple animation of a single model part with a predefined step size. Both interactions focus on a single model part, which we select by hovering the corresponding children of the container till it glows.

The drag & drop explosion was done by comparing the current mouse position on screen with its previous position, resulting in a 2D vector of change on the view plane. This 2D vector was extended into its 3D world version and projected onto the normalized explosion direction to find the size of movement in set direction from meta-data. For the explosion propagation version only recursive calls were added to blockers.

Offset jump was implemented by using the direction vector in the meta-data and a user provided multiplier to determine the offset size of a click. The interaction was designed to differentiate moved and initial-state model parts. The initial-state model parts are set to move upon each update until they reach the offset position, and the moved model parts move in the reverse direction until they reach their initial state. In case of propagated explosions, we redesigned the function to differentiate behavior based on the distance from the trigger origin. The behavior stayed the same with the addition of the propagation call for the root part, which was clicked by the user. The new addition was meant for non-root parts, where the offset click event is blocked until the root element returns to its initial state and the offset is stacked for each level of propagation.

Riffling

The last explosion type we implemented is a special case limited to usage in exploded views. The name riffling comes from the analogy of riffling through the pages of a book, where we give focus to one page and the remainder is stacked on its left or right side. In the case of 3D exploded views, it was defined as moving the riffled model part a bit more into focus while at the same time moving all its blockers away from the riffled model part, resulting in an overview like dissection of the model based on the selected model part. We implemented it with explosion propagation from the start, where the root element and all its blockers move in the same positive direction set in the meta-data. However, we set a much bigger explosion constant for the blockers when compared to the root model part.

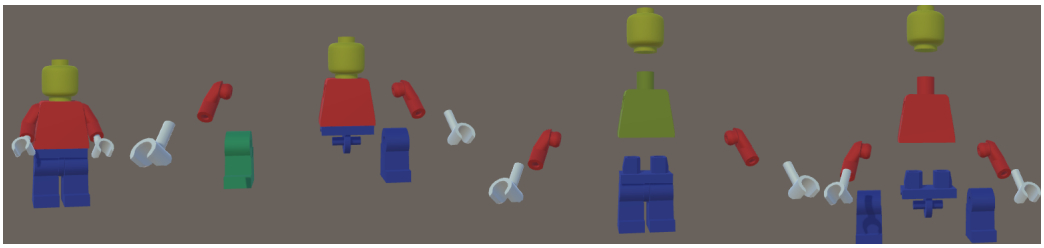


Figure 4.5: Examples of all propagated explosion interactions on `legoguy` model. From left to right we have initial model state, direct manipulation, riffling and explosion animation.

By observing Figure 4.5, we can see similarities between explosion interactions due to the same core function call. However, the differences can also be observed even in the static images of the figure. The direct manipulation model was dragged & dropped on its left half and deformed with offset click on the right side. Notice the difference in range due to offset click limit between the left and right side as well as the glowing left leg, which glows due to the part still being dragged at the time of image capture. In riffled model the glowing also indicates the riffling still being in progress and at the same time marks the trigger of explosion propagation. While the explosion animation

also has a lot of exploded parts similar to riffling, it has actually exploded all parts consistently for the duration of the key press. Riffling on the other hand exploded only the blockers of top part and propagated it towards the next triggers in the recursion, while it left the pants and leg parts intact due to not blocking the top. The propagation in riffling also gives on each level of recursion an extra offset step, which creates varies sizes of explosion steps, while explosion animation has only consistent step sizes.

4.3 Implementing Floating Labels

With the exploded view base from Section 4.2, we already created an interactive system for exploded views with label positions fixed with an offset from its referred model part mesh's center. However, such a solution requires manual labor for placing the labels, which we wish to be placed in optimal position automatically. Therefore, we started implementing hedgehog labeling's core [8], which is the floating labels algorithm, designed by Hartmann et al. [9] and discussed in detail in Section 3.1.

Since the concept of six forces from Figure 3.1 is needed for force field creation and the forces are defined by processing pixels of a rendered view, we decided to store its information in a texture and process it on the GPU via compute shader kernels located in ForceField.compute to speed up the solution. Among others, our compute shader includes kernels for edge detection, static and dynamic field calculation, extreme search, force field texture writing for result showing and so on. According to the official Direct computation shader documentation, each kernel has a limit of 1024 threads. This means that most of the written ones have a thread group of size 32x32 to accommodate the processing of a 2D texture, while reduction algorithms have a one-dimensional thread group. Another speed up was achieved by dividing the static force field calculation inside an update loop into part dependent and independent calculations. The independent calculations for forces B and D were removed from the label/part loop to be calculated only once and

then reused in each iteration of a label loop together with the part dependent forces A, C, E and F. Forces E and F were defined to only be calculated when non-negative UV position matching the provided rendered texture dimensions is stored on the GPU, which happens in each label loop iteration after finding the maximum value inside force fields. The whole process in stages can be observed the following subsections as well as in Figure 4.9 at the end of this section, where all produced views, textures, boolean indicators and force fields are presented in order of creation.

4.3.1 Color Coded Texture

While rendering a scene to a texture in Unity is no problem, we had to solve the problem of differentiating model parts from the rendered texture. Following the idea in the original paper [9], we simply changed the default Unity rendering to a color coded rendering, where each model part has a unique RGBA value on a background of RGBA value 0 = (0,0,0,0). We achieved it by replacing the standard shader with a custom shader, which renders each fragment with the same hard-coded color determined at model load. Results of color coded projections can be observed in Figure 4.6. To prevent the main rendering seen by the user from flickering between shaders we used besides the main camera an additional separate camera for rendering the textures for force field calculations.

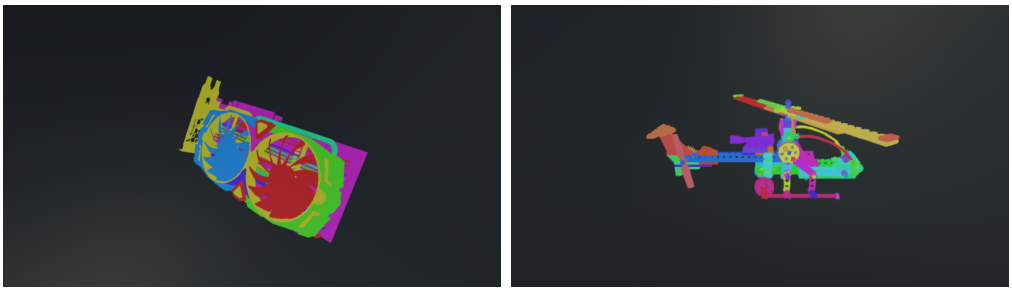


Figure 4.6: Examples of color-coded textures produced for the `gpu` and the `chopper` models with color differences observable by the eye.

While Unity uses the RGBA model in color definitions, it is not practical for defining contrast colors for color coding. Therefore, we used the HSV model as the entry point and transformed those to RGB. But HSV is also not problem-free. Hue transforms in a circle, resulting in values 0 and 360 defining the same red color, while colors with low Saturation and Value are hard to distinguish. For good results, visible to the eye for any number of parts, it quickly became a MAXMIN optimization problem, which is known to be NP-complete [37]. Since we only wish to mark separate parts during force field computation, we simplified the problem to numeric RGB color differences for our needs. Therefore, we just divided the ranges with the part count into equal parts. To remove problematic values, we limited the minimal step size and range for each value based on visual probing. Hue was set on a range [5,360] with minimal step size 5, while Saturation and Value on the other hand had the allowed range set to [0.5 1] with a minimal step size of 0.05. We also set the algorithm to be started by locking Saturation and Value to full capacity and just changing Hue, which results in 71 colors. If more than 71 parts are needed, we set Saturation to be unlocked first. When we reach the range limit with 710 colors, we unlock Value, and this results in more than 7000 colors.

We found the color code shader, which is used during the color coded texture generation, useful also as a material shader, when the color coding can be distinguished by the eye. As in the example of the colorless `brain` model in Figure 4.7 we can produce a nice visual separation of components just by using the color codes during the mesh's shading process if we simply change the material shader with the color code shader.

4.3.2 Interia

Besides distinguishing the texture-projected parts, we also needed to provide the GPU with the UV coordinates of each part's **interia**, which is the 2D projected anchor point location. The problem was to define what an anchor point should be. A simple solution was to use the mesh center as an anchor

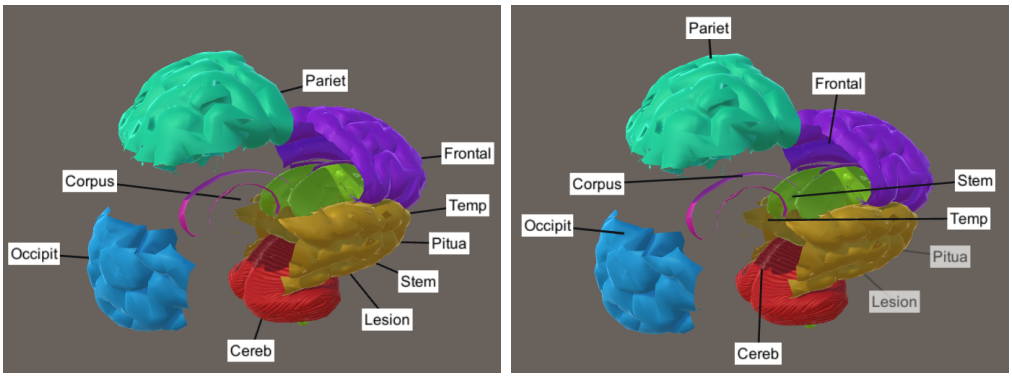


Figure 4.7: The left image demonstrates the problems that are caused by selecting the mesh center as an anchor point and its projection for the interia by using the `brain` model as an example. The Corpus pair with a hole in its concave mesh causes the leader line to point towards empty space. Misinformation occurs due to the anchor points for the leader lines of the Lesion, Pitua, Stem and Temp labels being invisible. Distinguishing the referred one of the four labels is impossible since all leader lines seem to point towards the yellow part. In the right image, we present our solution of detecting the anchor points with a thinning algorithm, using transparency as an invisibility indicator and switching the rendering order selectively based on visibility. The images also present how color coding as a material shader can enhance the structural information of the model.

point, however such a solution has horrible results for the not so rare concave meshes, where the anchor point may land in a hole of the part, resulting in the leader line pointing towards nothing. Another problem that arose was the clarity of which part corresponds to which label because this becomes tricky if the anchor point is not in a visible area. In such cases, the anchor line went through other objects and most users would not know which part it corresponds to without hovering the cursor over the label or part to find out. All of the described ambiguities can be observed in the `brain` model example in Figure 4.7.

Remaining loyal to the original paper, we followed their idea of using thinning algorithms to thin out all of the model part pixels until the last remains as our interia. Since thinning algorithms include many conditional clauses, and we also need the anchor point updated on the CPU for drawing the leader lines, we decided to keep the anchor point search on the CPU. While GPU thinning algorithms exist and are still widely researched [38, 39, 40, 41, 42], they are quite complex and case specific. At the same time, they would require an increase of the already big amount of CPU-GPU communications, which are a well-known bottleneck in GPU computing.

First, we simplified the approach by checking the mesh center and if its projection or interia is visible in the color coded texture. Even if it was determined as invisible, we set it to be a fail safe in case the thinning algorithm fails to find a better solution. From the big variety of thinning algorithms found in literature, we chose a quite old, but at the same time well-regarded, robust and simple one called Zhang and Suen's thinning algorithm, also known simply as the ZS thinning algorithm [43]. The ZS thinning algorithm checks the changes of color presence in the eight neighboring pixels. Color presence in their case was state 1 and lack of color was called state 0. They also numbered the pixels in a spiral way by using the current pixel as the starting point and continuing into the right direction. Using the indexes, they defined a transition function $A(P_1)$, which tells the count of transitions from 0 to 1 while proceeding towards the next neighbor in the order of: P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9 and finally P_2 again. See Algorithm 2 for pseudo code.

P_9	P_2	P_3
P_8	P_1	P_4
P_7	P_6	P_5

Table 4.1: Table cells represent a grid of 9 neighboring pixels with the pixel numbering used by the ZS thinning algorithm. The pixel in the center numbered P_1 represents the currently processed pixel.

Algorithm 2 ZS thinning algorithm

1: **repeat**
2: **for** pixel in texture **do**
3: **if** $pixel == 1$ && has 8 neighbors && $1 < \sum_2^9 P_i < 7$ && $A(P_1) == 1$
&&$(P_2 == 0 \parallel P_4 == 0 \parallel P_6 == 0)$
&& $(P_4 == 0 \parallel P_6 == 0 \parallel P_8 == 0)$ **then**
4: mark pixel
5: **end if**
6: **end for**
7: set all marked pixels to 0
8: **for** pixel in texture **do**
9: **if** $pixel == 1$ && has 8 neighbors && $1 < \sum_2^9 P_i < 7$ && $A(P_1) == 1$
&&$(P_2 == 0 \parallel P_4 == 0 \parallel P_8 == 0)$
&& $(P_2 == 0 \parallel P_6 == 0 \parallel P_8 == 0)$ **then**
10: mark pixel
11: **end if**
12: **end for**
13: set all marked pixels to 0
14: **until** no pixel was set to 0 in this iteration

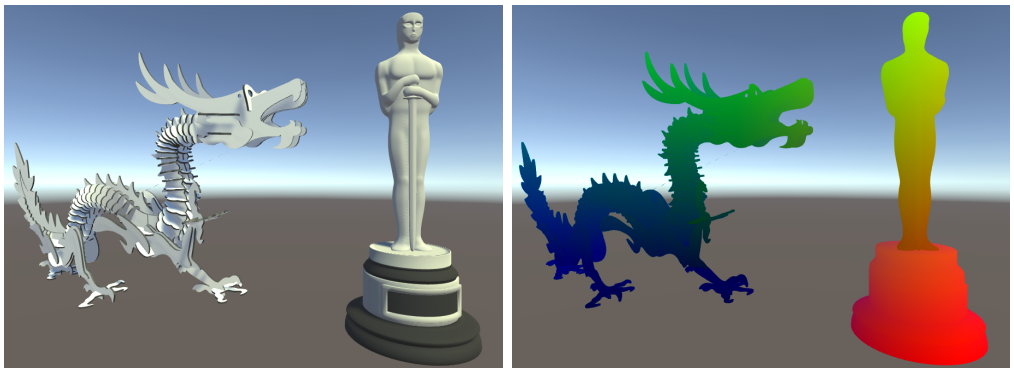


Figure 4.8: Results of interpolating the world coordinates and depth can be observed by using the interpolation shader as a material shader on actual models, since they are already normalized and stored as colors. In the images we present the interpolation results for the `dragon` and `oscar` model. We can observe how the X coordinate is represented in red, Y in green and Z in blue by the positions of the models. One can also see the dragon's belly being close to the origin of the coordinate system due to its black color.

Thinning algorithms will in most cases return multiple pixels still remaining set to state 1, therefore we had to select the one that works best for us. We decided to check its original 3D position, its distance from the mesh center and the distance from the camera. To determine the 3D position of an arbitrary pixel, we created a shader that interpolates world coordinates from the vertex to the fragment shader. In the fragment shader, we coded the interpolated values as colors to be stored in a texture. While Unity does provide high precision texture formats that do not normalize values to [0,1] ranges, their cross-platform compatibility is low and GPU dependent. Therefore, we used the overall supported ARGB32 format with normalization based on interpolated fragment and mesh center depth. Normalization to [0,1] range was achieved by subtracting the minimum values of the mesh bounding box before normalizing the values,which results in a shading as seen in Figure 4.8:

$$R = (X - X_{min})/Depth_{fragment} \tag{4.1}$$

$$G = (Y - Y_{min})/Depth_{fragment} \tag{4.2}$$

$$B = (Z - Z_{min})/Depth_{fragment} \tag{4.3}$$

$$A = Depth_{fragment}/Depth_{center} \tag{4.4}$$

We set Textplosion to loop over the remaining pixels and only check those that have their interpolated 3D position inside the mesh's bounding box in order to prevent artifacts from forming due to floating point precision errors. From the valid positions available, we decided to select the one closest to the mesh center and the camera as our anchor point and its UV position as its interia. The precision of the stored data naturally became lower due to many floating point operations. However, for our purposes it worked fine with rare slight mistakes such as the leader lines extending slightly beyond the mesh, but even in these cases it was clear to which mesh they pointed at.

Since we defined a way to check if the current anchor point is visible to the system, we decided to use this as information that helps us to further enhance the user experience. Besides setting the important elements to glow, the study on GPS navigation applications [22] also showed that user experience improves if the less important information is made transparent to show more of the important information. We decided to use transparency on labels that have their referred model part invisible from the current camera pose. At the same time, we set the rendering order for the model meshes, labels and leader lines. In case of the visible model parts, the leader lines and the labels are rendered in order on top of the mesh rendering, while in case of the invisible model parts, they are rendered in the same rendering phase as the mesh. This rendering order resulted in a solution that involves less ambiguity regarding which line points to which mesh.

4.3.3 Edge Detection

Before processing the color coding for forces, we produced a GPU based boolean array to indicate the edges in the texture. We used a simple discrete Laplacien of Gaussien negative convolution kernel or LoG as our edge detector on the GPU. It takes the neighbors of the pixel, calculates its derivatives and based on them decided the pixel's edge status. Because the color coding was designed to use a zero vector for background color, it resulted in a controlled environment, in which we could use the smallest and simplest negative kernel of 3x3 size to detect the edge:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \tag{4.5}$$

4.3.4 Static Force Field

For force field calculations, we needed to preload a lot of information for label position update runs. To prevent too much traffic between the CPU and the GPU, we designed a few GPU array Labels to be pushed to GPU in a single push while keeping rows in the memory word size. Such an example is the ModelParts look up array, which stores each model's cluster identifier, the interia U and V coordinates, the largest distance between the interia and a pixel and its color code. Similar to these eight floats, we also have other sets of row values, which form the 32B memory word size. By limiting ourselves to 32B per row in these arrays, we achieved faster performance due to the GPU memory retrieving all the needed information in one read and storing it locally. In case of ModelParts shown in Table 4.2, we also limited it to the model parts that are referred to by a label, resulting in less memory transfers and at the same time enabling us to use the same mapped index to indicate corresponding row pairs in different arrays.

The static force field kernel first needed to determine if the current pixel corresponding to the thread id is a color coded pixel of correct color code.

Cluster	U	V	Max Distance	R	G	B	A

Table 4.2: A row of eight float values in the ModelParts array. Each row index corresponds to the index that was mapped during the first initialization, and points to the same pair of label and model part among such arrays.

Color checks were simply done with boolean operations. This was done in order to determine equality to zeros inside the color vector for defining a model indicator and then compare individual color components with the color code in the ModelParts row to define a part indicator. Note that we only did boolean operations without actual branching besides the beginning pixel inside the dimension check. By generating booleans and converting them to 0/1 floats, we generate effective multipliers without slowing it down due to branching code, which is a GPU-hostile action considering performance.

The first force mentioned in the paper was the attraction force A, which attracts towards the interia UV coordinates provided in ModelParts. The attraction force distribution was redefined from Equation 3.1 as the following weighted normalized distance:

$$F_A = Const_A \times \neg PartIndicator(x, y) \times \frac{1 - \sqrt{(U - pixel.x)^2) + (V - pixel.y)^2))}}{MaxDistance} \tag{4.6}$$

For force B or contour repulsion from Equation 3.2, the user needs to set a so called contour influence integer, which was designed to tell how many pixels into any direction we are searching for contours, which results in a magnification of the detected edge pixels. We implemented it by simply using a for loop, which only combines surrounding pixels' corresponding detected edge booleans from Edges array with OR operations. While for loops contain branching statements, these are unrolled as repeated code instead of loops in compiler in case the if statement uses uniforms and is deterministic enough. In our case it was since we looped from the negative to the positive uniform value influence on rows and columns. The resulting boolean still needed to

be multiplied by the contour constant to enable parametrization.

$$F_B = Const_B \prod_{row=y-inf}^{y+inf} \prod_{col=x-inf}^{x+inf} Edges(row, col) \tag{4.7}$$

The repulsion from other parts or from force C from Equation 3.3 was simply implemented as a multiplication of the booleans that mark if the pixel is inside of the model and if the pixel does not belong to the current part. Naturally, the result is multiplied by the repulsion constant of the model.

$$F_C = Const_C \times \neg BackgroundIndicator(x, y) \times PartIndicator(x, y) \tag{4.8}$$

The last Equation 3.4 defined static force D as the view border repulsion. Here we compare the view border influence to the smallest distance between the current pixel and the border, which equals the smallest value from the set of values: U, V, remaining distance from right border, remaining distance from top border.

$$MinWallDist = Min(x, y, width - 1 - x, height - 1 - y) \tag{4.9}$$

$$F_D = Const_D \times (MinWallDist \leq WallInfluence)\frac{1 - MinWallDist}{WallInfluence} \tag{4.10}$$

The end static force field was then combined by subtracting the highest repulsion force from the attraction force as in Equation 3.5:

$$StaticForce = F_A - Max(F_B, F_C, F_D) \tag{4.11}$$

4.3.5 Extrema Search

The search for extrema in the force field was done on the GPU with a maximum search reduction algorithm. By using such an approach, we drastically changed the amount of memory calls by using shared memory and three kernels for better parallelism. The first kernel was prepared to produce a new field containing the force sums inside a label sized convolution kernel. The second kernel used shared memory to compute the local max values inside

multiple thread groups. And the final kernel used a single thread group to find the global extreme from the stored local extremes of previous kernel and stored it to the CPU retrievable buffer object because the CPU needed to know the new label location.

While observing form of the sum field, we noticed it being a smudged version of static force field. Since it looks like a low-quality like image, which however penalized small too positive areas, we came to the conclusion that there is no reason to use big textures, if we reduce their sharpness and therefore switched to low resolution textures instead. By moving from a 1280x700 texture resolution with over 150k pixels to a 128x128 resolution texture with a bit more than 1.6k pixels, we achieved a workload decrease of around 90% for force field calculations, while keeping results intact. While the efficiency increased greatly with equally good results on the extreme search, it did reduce the effectiveness of the thinning algorithm in Subsection 4.3.2 since we used the same texture for searching the best anchor point. A smaller texture regarding the anchor point search meant less pixel data to thin out and less world coordinate data to interpolate for determining the original 3D position from projection, which in turn resulted for really thin objects or only thin visible part pixel patches to be marked as invisible.

Another reoccurring problem was the incompatibility of the force sum and the label's auto-rotation towards the camera, which would need to be taken into account by predicting the label shape in any texture pixel by using the determined deformation in the sum kernel. Since this would overcomplicate and slow down the process while producing questionable results, we preferred to disable the auto rotation to prevent overlaps from occurring. We provided the user with the option of enabling it on condition that they understood that the overlaps will increase with models in with high label amounts.

4.3.6 Dynamic Force Field

The last calculation regarding floating labels that we have missed up to this point is the dynamic force field calculation. After the first label was

placed, we already produced data that could be used for the next label. For the information to be made available, we created the next eight value row based array called Labelsas shown in Table 4.3. Labels was designed with rows of eight integers since the data was used in regards to the UV texture coordinates. It was set to store the label's center, extents and the extrema search area bounds, which was the whole texture in case of just the floating labels algorithm.

To transform the static force field into the dynamic one, we only needed to include label repulsion or force E from Equation 3.6 and our newly defined Leader line repulsion or force F in Equation 3.7. Label repulsion forces were set to be generated by checking if each pixel is inside the label provided dimensions. Those label provided dimensions also had user provided padding included to achieve a user desired distance between labels.

$$F_E = Const_E \times (U - Right \leq x \leq U + Right) \times (V - Up \leq y \leq V + Up) \tag{4.12}$$

The new force E is similar to the contour repulsion or force B, see equation 4.7. We copied the principle of magnifying the effect of edges by an influence factor to our drawn leader lines. Since the leader lines were not directly provided to the GPU, we had to recalculate them on the GPU instead of again adding load to the CPU-GPU traffic.

$$V = SlopeX \times (U - U_0) + V_0 \tag{4.13}$$

$$U = SlopeY \times (V - V_0) + U_0 \tag{4.14}$$

Similarly as in the label repulsion forces calculations, we used line equa-

U	V	Right	Up	Search U	Search V	Search Width	Search Height

Table 4.3: A row of eight integer values in the Labels array. Each row index corresponds to the index that was mapped during the first initialization, and points to the same pair of a label and model part between similar arrays.

tions on the interia and the label center to determine the line slopes. Note that we added a small ϵ value to prevent divisions by zero.

$$SlopeX = \frac{V - V_0}{U - U_0 + \epsilon} \tag{4.15}$$

$$SlopeY = \frac{U - U_0}{V - V_0 + \epsilon} \tag{4.16}$$

With the line equations and calculated slopes, we designed the check to inspect each pixel's (x,y) coordinates between the anchor point and the label center and see if its X and Y calculated from the line equation correspond to x and y by using the given influence factor as an error threshold.

$$Y = SlopeX \times (x - U_0) + V_0 \tag{4.17}$$

$$X = SlopeY \times (y - V_0) + U_0 \tag{4.18}$$

$$InRangeY = Y - Influence \leq y \leq Y + Influence \tag{4.19}$$

$$InRangeX = X - Influence \leq x \leq X + Influence \tag{4.20}$$

$$F_F = Const_F \times (InRangeY \wedge InRangeY) \tag{4.21}$$

By combining everything together, we arrive at the dynamic field already defined in Equation 3.8, which we can unpack into:

$$DynamicForce = F_A - Max(F_B, F_C, F_D) - F_E - F_F \tag{4.22}$$

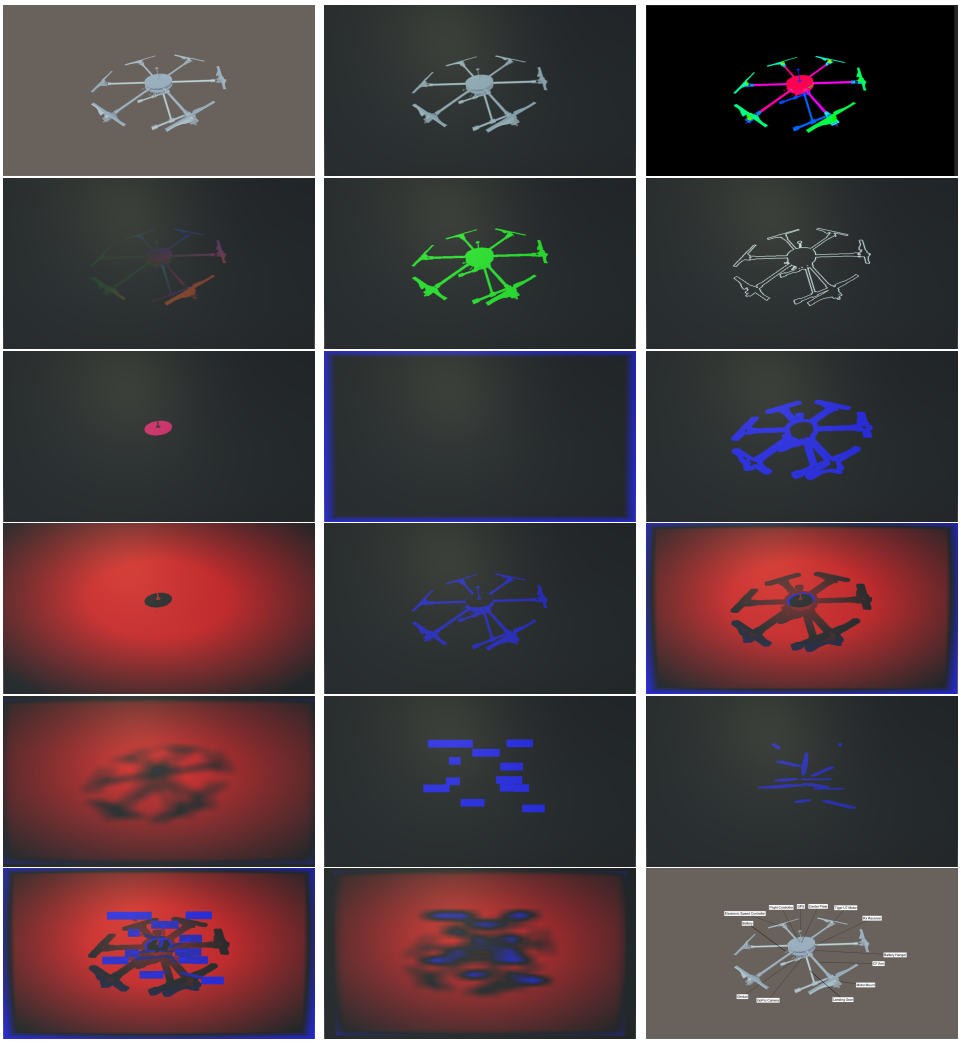


Figure 4.9: From left till right top till bottom we have in order of the force field pipeline the `engine` model's central plate part's calculation stages: the initial view of the scene and the view's normal projection, color coded projection, world coordinates interpolation, boolean indicator of model pixels, boolean indicator of edge pixels, boolean indicator of part pixels, view border forces, contour forces, attraction forces, forces from other parts, static force field, static force sum, label forces, leader lien forces, dynamic force field, dynamic force sum and resulting view with labels placed at the extremes.

4.4 Implementing Hedgehog Labeling

From the beginning of Chapter 4 up to Section 3.1, we had already arrived at an implementation of an interactive exploded view GUI that uses the floating labels algorithm [9] for automatic placement of 3D label objects. However, we have not yet gone into details of the label plane implementation, which is essential for using the 2D floating labels algorithm for positioning 3D labels in 3D space with good results. Neither have we taken a look at the freezing life cycle or the local extrema search yet.

4.4.1 Label Plane

The label plane was implemented in Textplosion as a pair of the mathematical model and its to the observer visible presentation in a form of Unity's Plane object and PrimitiveType.Plane mesh. The mathematical model was used for projection purposes during calculations, where we mostly used the ClosestPointOnPlane function to project points, whole labels and leader lines. On the other hand, the visible representation was used for storing pose information as well as for presenting force fields or indicators as 2D textures planted onto the plane, which was a useful tool for debugging the GPU code. In regards of defining the view vector from Equation 3.9 needed as the plane's normal, we simplified it as the distance between the camera and the previously defined focus point from Section 4.2.2, that used the combined mesh's bounding box center point as a point to rotate the camera around on a sphere.

$$View_{vector} = Plane_{position} - Camera_{position} \tag{4.23}$$

Since the complexity of the code rose with the addition of floating labels, we separated the label placement code in a separate script called LabelOcclusion.cs, which run the label update loop as a coroutine. By running the code as a coroutine, we achieved unbroken GUI interactivity as well as a seemingly parallel-like execution since waiting times are better hidden than

without coroutines. To prevent problems with coroutine starts and destructions, we coded Textplosion so that TextplosionSetup.cs loads, prepares and runs LabelOcclusion.cs as part of its execution. Naturally, due to the different behavior inside coroutines, we needed a way to store the label plane information that changes with any camera pose change. Since we had already prepared a visible mesh representation of the plane, we just used its Transform object to store pose related information.

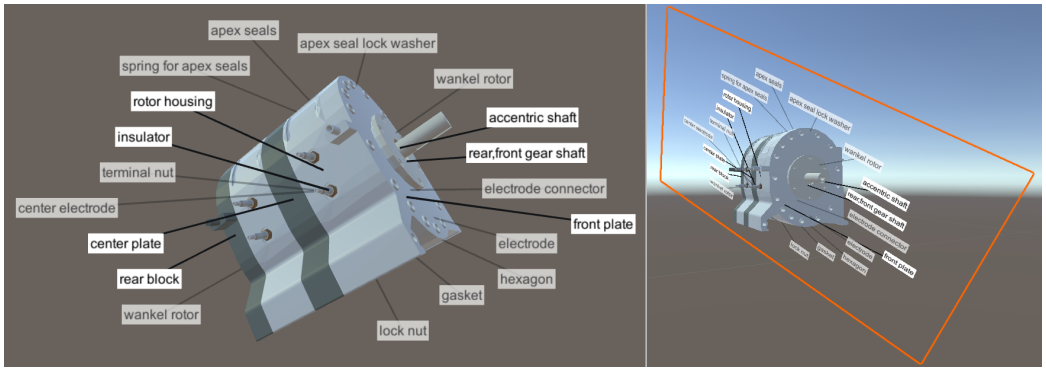


Figure 4.10: In the left image we have the usual view of Textplosion, where the labels are already projected to the label plane after setting an arbitrary camera pose. Since it is parallel to the view plane, unaware observers would think it is simply placed on top of the view plane instead of inside the 3D space. A rendering from an alternative camera, presented in the right image, shows the actual behavior of the whole scene. A plane going through the mesh' center is rotated to match the main camera's pose at any unfrozen moment and the labels are placed on top of the plane, while the model has not changed its initial from the moment it was loaded into the scene.

On camera pose change, there is naturally a change in the view plane orientation and therefore there are also more changes in label plane information, which resulted in us checking each camera update for changes in the camera pose. With the present changes the mathematical model was set to be redefined with new values. The pose of the mesh representation was set to update as well. Besides updating the orientation and the static focus point

positions, we also rescaled the mesh to store the plane dimension as a scale. The current plane dimensions are essential for a correct conversion between the 2D and 3D label positions, while at the same time it enables a presentation of GPU produced textures directly fitting to the model dimensions as seen in Figure 4.10. We used the distance between the plane and the camera positions together with the camera's provided field of view and aspect to first calculate the height and from it the width. The retrieved dimensions are then used to obtain the correct scaling factors from the initial plane mesh dimensions.

$$Height = 2 \times tan(\frac{FieldOfView}{2}) \times distance \tag{4.24}$$

$$Width = Height \times Aspect \tag{4.25}$$

The transformation from the 3D world coordinates to the rendered texture UV coordinates was defined as a switch between multiple coordinate systems. First, we expressed the positions in the plane local positions:

$$Vertex_{plane} = Vertex_{world} - PlanePosition_{world} \tag{4.26}$$

Afterwards, we rotated the plane local space position to the initial position of plane visualization that we call texture space by using the inverse rotation of the plane:

$$Vertex_{texture} = Inverse(Rotation_{Plane}) \times Vertex_{plane} \tag{4.27}$$

Within texture space, we were just one step away from the corresponding [0,1] UV coordinates. The equations were extracted by observing the texture behavior presented in Figure 4.11:

$$U = \frac{1}{2} - \frac{Vertex_{texture}.X}{Width} \tag{4.28}$$

$$V = \frac{1}{2} - \frac{Vertex_{texture}.Z}{Height} \tag{4.29}$$

$$U_{pixelized} = Round(U * \times Texture_{Width}) \tag{4.30}$$

$$V_{pixelized} = Round(V * \times Texture_{Height}) \tag{4.31}$$

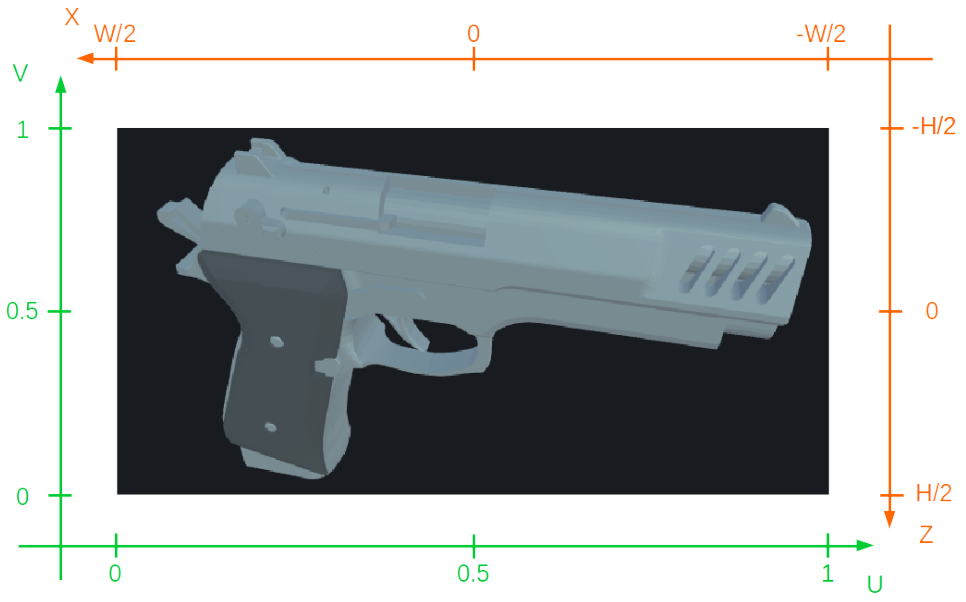


Figure 4.11: The comparison of our label plane texture coordinate system in orange color with Unity's UV coordinate system in green. In regards of the example with the gun model projection, we can observe our coordinate system being centered in plane center, while Unity's as many others are centered in bottom left of the texture or plane.

orange coordinate system is the plane coordinate system in Unity's units starting in plane center, which was recalculated before into the Unity's UV coordinate system in green

4.4.2 Label Freezing

The life cycle of hedgehog labeling was designed with **calculation frozen** and **unfrozen** states. The frozen state was set as the default state after each label positioning calculation finished, while the switch to unfrozen state was meant to happen after an unfreeze trigger condition was met. In the

Textplosion we defined four kinds of unfreeze trigger conditions:

1. Initial label positioning,
2. Camera angle threshold,
3. Camera zoom threshold,
4. All explosions finished their animation.

While condition 1 was only meant to be used upon start or on user forced reinitialization, and condition 4 was already solved by using explosion notification events from Subsection 4.2.2, the remaining two conditions were newly defined with the view vector from Equation 3.9. We designed the conditions to store the last view vector that triggered an unfreeze and to compare it to the current view vector. For the angle threshold, we simply calculated the smallest (absolute) angle between the stored and the fresh view vectors, and compared it to the user set threshold. Similarly, we added user input for the zoom threshold, which takes the percentage of zoom or radius change:

$$Change_{\%} = Abs(1 - \frac{||ViewVector_{stored}||^2}{||ViewVector_{fresh}||^2}) \tag{4.32}$$

To prevent issues between frozen label calculations leader line extending was and implementeda as a CPU check that only triggers during the frozen state. We defined the check to loop over the label objects and to obtain their UV positions and dimensions, which were used to read all the label hidden screen pixels. The check also inspected if any label pixels are positioned over non-background pixels inside the color coded texture. For labels with non-background pixels, we extended the leader line in its set direction towards the label center, which was proposed for the 3D label objects in the hedgehog labeling paper [8]. We designed the leader line extension to repeat itself until all label-covered pixels are background color and until the label is still in view, where for the later case we go one step back after breaking the condition. The leader line extension required us to set an extension step size, which was derived from the size of the label's right and up extent vectors.

The leader line extensions and the still running explosions can also cause overlaps with between labels or with between labels and leader lines, not to mention leader line crossings. While we succeeded at minimizing the overlaps between the labels to a satisfactory degree by extending the CPU frozen check, not much could be done regarding leader line overlaps and crossings since extending them would only increase the amount of leader line crossings, while subtracting the leader line would quickly result in overlaps with the model's silhouette. The CPU frozen check was extended by adding another label loop after the model overlap check code inside the existing label loop.

Inside the double label loop, we inspected all of the label pairs for intersections between the label background bounding boxes and, in case of an intersection due to an overlap, we extend the inner loop label's leader line to always move the outside label to limit individual label extensions in size. Note that this propagated extension due to label overlap required us to recheck all the labels inside the propagation. This could, in case of too many labels and too little space, cause infinite loops, which we prevented by limiting the outer label loop to iterate at most N^2 times for N labels.

In the end, the leader line crossings and overlaps together with the labels trying to cross the view border could not be solved in all cases during the frozen state due to the issue in the design of hedgehog labeling, where we only calculate the label positions with force fields and extend the leader lines during frozen states. While we could have tried other approaches, which would in turn distance ourselves from the simple concept with questionable results, it turned out that by using smart unfreeze condition thresholds, we can minimize the occurrences of the remaining issues or even prevent them from happening, depending on the 3D model.

4.4.3 Leader Lines Crossing Prevention

While the frozen CPU check could not solve the leader line crossings, we succeeded in solving the crossings in the unfrozen force field calculation loop. When all of the labels finished positioning themselves after a force field calcu-

lation, we ran a double loop to check all of the labels for leader line crossings with the analytical directional vectors check, found in Algorithm 1. We changed the positions of the labels in the label list of the found crossings. We also adjusted their 3D positions and restarted the double loop. While we ended up with using this CPU based solution, we did try moving to the GPU or finding an alternative GPU solution. However, in the end we failed to make any of the proposed ideas work in real-time due to a high amount of branching and memory reading required for solving the problem.

We also noticed that the CPU solution likewise had a potential problem with the infinite loops of two labels interchanging, which we solved by limiting the label switches to happen only when neither of the pointers holds the other's label list index.

4.4.4 Local Extrema Search

The switching of the local and global extrema searches was achieved by using the search columns of the Labels array, which has the row defined as shown in Table 4.3. The values were used to limit the force sum generation kernel as well as the reduction algorithm area of the search defined in Section 4.3.5.

4.5 Implementing Clustering

By clustering labels based on the meta-data provided explosion directions and the current exploded view state, we extended Textplosion with the concept of clustered labels focusing only on their own cluster. This reduced the amount of global reinitialization runs and in turn resulted in more stable label layouts as well as slight speed ups. We also enabled switching between hedgehog labeling whenever we returned the exploded view into its initial state and clustered hedgehog labeling after exploding it out of its initial state.

In Textplosion we handled different types of clustering methods by extending the abstract LabelClusteringBase.cs script, which was set to handle the plane and the clustered label list creation as well as looping over the data

hierarchy to collect all possible data for clustering guidance. A simple and useful extension example was the SingleCluster.cs, which places all the given labels inside a single cluster. It was used to switch to the original hedgehog labeling algorithm without clustering. The clustering execution was handled by keeping track of the explosion events, where clustering was triggered after all of the explosions had been stopped.

For K-means, we extended LabelClusteringBase.cs to another abstract class called KmeansBase.cs, which defined three abstract functions to be implemented in any K-means versions. The K-means versions were meant to be a hard-coded selection of metric vectors that simplify the usage and testing. The three abstract functions defined the vector length, how randomization was handled per vector index and which of the retrieved explosion and deformation data to use as metrics. The randomization specification function was added to prevent overly random starting vectors from producing single clusters instead of the desired K amount.

In Section 3.3 we mentioned four potentially good clustering metrics: `direction`, `position`, `changes` and `focus distances`. These metrics were the selection that we found most logical and without obvious correlations. The problem of correlation can be easily seen in an example of an abandoned metric like non-normalized direction, which correlates to the obvious normalized direction as well as to the change metric, which would have the directions magnitude hidden in the size of change. In case we used all four metrics together with the abandoned one, this would result in the weight of two metrics being unfairly amplified due to an error of the metric selection. This would become an even greater problem due to us not knowing if those two metrics would improve or regress the result.

While the proposed four metrics showed some promise during the trial and error phase, we were yet unsure about which out of the fifteen possible proper subsets of metrics produced the best results. By conducting further trial and error experimentation on different models and Textplosion settings combined with some logical reasoning, we determined some subsets to have

more potential than others. Since we planned to run some user experience tests regarding the effectiveness of the subsets, we had to limit ourselves to a reasonable number of subsets, which we determined to be a group of three subsets, see Subsection 5.3.1 for details on the reason for the amount. We determined the following three subsets to be worthy of running user experience tests on during evaluation:

Position Clustering will produce good results on label positioning when clusters will contain elements that are close to each other like chunks inside an actual explosion since the probability of leader lines crossing will be lower due to elements being close together.

Exploded Position Enhancing the position with information on explosion direction and explosion progress should prevent grouping of the elements that are close together but will in the future be far apart. Here we predict that the clustering subset of position, direction and change will create clusters, which will change less through time due to the explosion information being present. Fewer changes in clustering would also mean fewer re-initializations after each re-clustering.

Combined Adding Focus distances to the `Exploded Position` subset adds extra information in regards to the initial mesh center. It also potentially improves the explosion progress detection regardless of incompatible directions like mirrored directions. Since the extra metric forms the initial set again, we simply call it a combined subset since all the metrics are used.

The result of introducing clustering to Textplosion can be observed in Figure 4.12, where we exploded the `gun` model through the whole scene, which triggered K-means clustering with the `Exploded Position` metric set to produce four clusters and their label planes. Since K-means requires a input of desired maximum count of clusters, we decided to set the default value of this user input to be four after trial & error. Since scenes with more

than 20 labels quickly fill up the screen or require a much smaller label scale, making them hard to read, we take is an empiric estimate that to be a vague maximum of labels present in most cases. By assuming the division into clusters would result in optimal division with equal label count per cluster, we determined five labels to be an adequate label count and therefore set the K-value to four. Naturally the optimal division is just an assumption and rarely happens, however the selected k-means produces logical divisions of 3D space as observed in Figure 4.12.

4.5.1 Additional Extensions for Exploded Models

While the extension to clustered hedgehog labeling seemed trivial, we have not yet discussed the exploded view specific extensions for enhancing the experience, which we implemented to achieve better results.

Transferring Label Information Between Explosions

A starting option for the extension was that we simply start the reinitialization after the explosion had finished. While this did provide us with the correct results, it also resulted in disturbing the user focus since the topology can change too much to keep the previous offsets in regard to the mesh center still as global extrema. To prevent huge label jumps, we decided to enable the transfer explosion updated positions to the GPU, and, instead of reinitialization, started with an update run first.

Sharing Label Information Between Clusters

We decided to separate the labels into clusters to reduce the amount of needed calculations inside each cluster. This naturally resulted in the clusters not knowing the label information inside other clusters, which lead to label overlaps between the clusters. The label overlaps frequency grew with the amount of parts, labels and explosion progress because they all limited the available free space. To prevent such overlaps from occurring, we added

an option that allows sharing of label information between clusters. Using this option slightly limited the contribution of our approach, but the main contribution of clustering, that not all labels have to be reinitialized when irregularities such as anchor line crossings occur, remained.

Negative Extrema Handling

The limited free space produced by exploded views also caused problems in local extrema search, since it easily generated local areas with non-positive force field sums. In implementing hedgehog labeling we rarely arrived at a situation where we had harshly limited space that resulted in negative local areas before a placement objective was broken, resultin in reinitialization before negative areas appeared. Exploded views with label information transfers enabled can however produce low-count clusters with bad initial label positions that are not detected due to a low amount of labels not not causing placement objective breachments as line crossings. Non-positive extrema of such cases were regarded as good positions by the system. In reality, this was not true because they produce overlaps with some model parts or even labels. To prevent the issue without reinitializing the whole cluster because of a single poorly positioned label, we added a non-positive extrema check after positioning each label and run an additional global force sum and extrema search on only the detected problematic label.

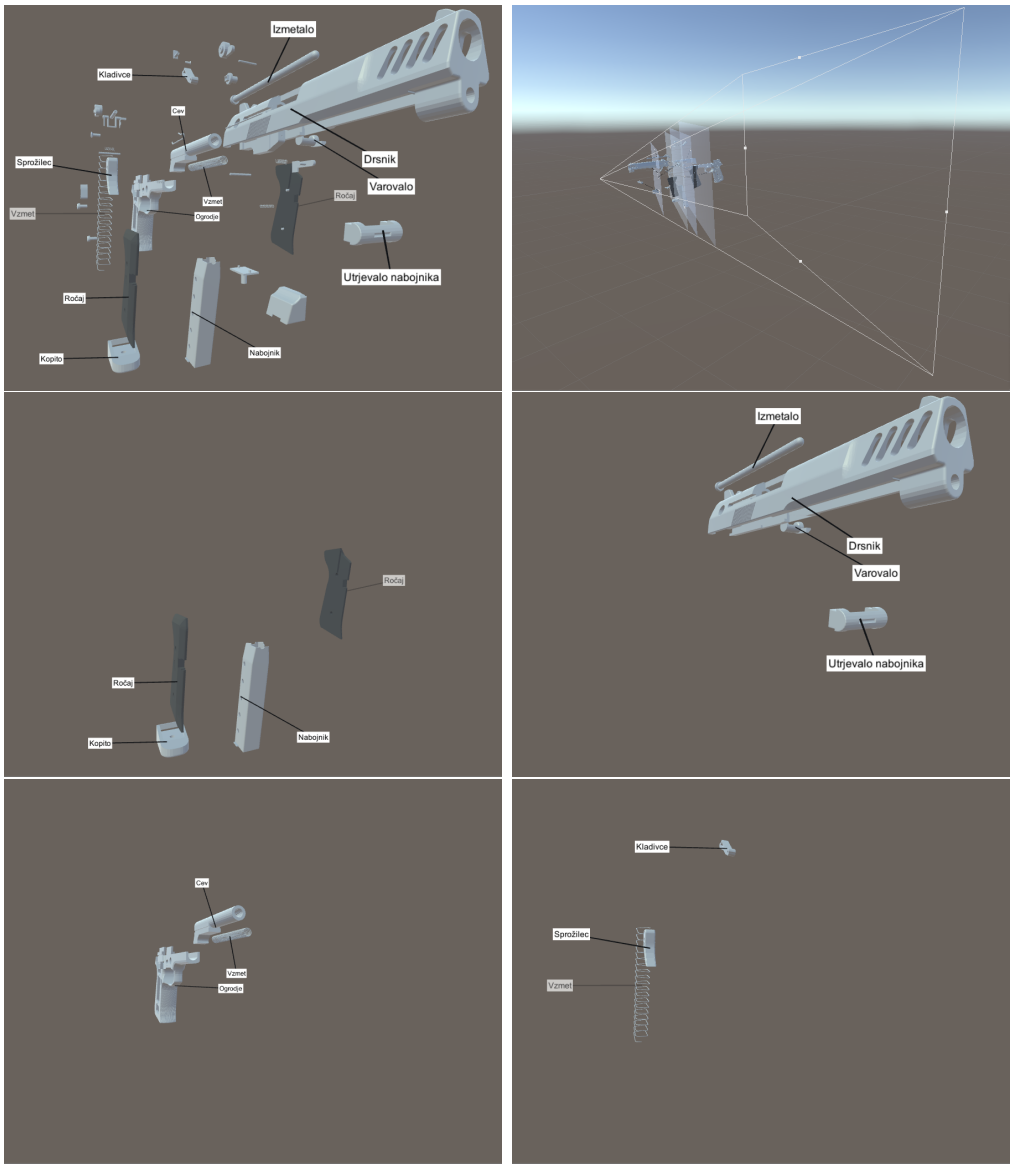


Figure 4.12: Clustering in Textplosion with the default settings produces for the given exploded view setup of the `gun` model shown in top left image four label clusters and their label planes presented the top right image. The overview of planes is a side view produced with an alternative camera to observe how the planes were fitted to the camera field of view based on the cluster's center position and the view vectors plane intersection. Textplosion offers also an option of only presenting one cluster at a time, which results in the four images bottom images. Since not all parts have labels, which means we do not cluster them either, we naturally never present parts without labels in cluster only mode. The distance between label planes can be also observed in the initial general view of the exploded model, since it causes a depth effect on labels between clusters, making their 3D localization easier.

Chapter 5

Evaluation

To prove that our suggested clustered hedgehog labeling improves the user experience on exploded views of 3D models, we decided to run a study using the implemented Textplosion system. By enabling and disabling its features, we were able to produce the implementations of floating labels and normal hedgehog labeling as the study's alternative algorithms. For simplicity and clarity of algorithm comparisons in regards to the study results we shall introduce acronyms for the three algorithms. Floating labels is shorten to as FL, hedgehog labeling as HL and clustered hedgehog labeling as CHL.

We gathered data from observing 3D label positioning with clustered hedgehog labeling and the prepared alternative algorithms based on consistent simulations of a user's interactions with the system. The gathering of unbiased comparison data was achieved by inviting a group of volunteers, to whom the whole thesis concept was fairly new, from outside of our institutions, which also created a population sample that is closer to the general public than the experts in the field are. Furthermore, we referred to each algorithm as a method marked with a random letter from the set {A,B,C} to prevent the participants being influenced by the algorithm names due to previous task experience or even the implied extension between algorithms. By using a selection of exploded 3D model from our dataset, see Section 5.1, we divided each task into subtasks of model and method pairs to fur-

ther limit the effect of models on the general results. Following Leykin and Tuceryan's conclusion on the need of a contrast between the label text and the background for good readability [28], we used the most often seen color set in literature with the labels being written in a black font on a white background, while the skybox was changed to a gray brown color to give contrast to the label background as well as to the annotation font for cases of transparent labels due to anchor point invisibility.

While we could have just gathered the subjective information with topic specific and standardized questionnaires, we decided to also gather objective information through the use of eye-tracking hardware and software. From the measurements gathered through eye-tracking, we constructed several standard metrics without focusing on any of them due to most standard metrics having a static nature to them. The static metrics have a hard time contributing useful information for dynamic stimuli inside interactive applications such as Textplosion [15]. By producing many metrics, we tried to infer their information contribution directly from the resulting measurements during the analysis. All of the observed metrics are described in detail in Appendix E, while only the ones with statistically significant information contribution are discussed in this chapter.

We also used the experiment as an opportunity to gather information on needed Textplosion improvements from interactive tasks, where participants controlled Textplosion by themselves. The interactive tasks were a smaller version of usual user experience tests to keep the experiment tasks consistent in form. Compared to a typical user experience test, we had a smaller but an eye-tracking enhanced version, which we used to determine the difficulty of using the system as well as obtaining any other potentially informative comments from the participants. While interactive tasks have an even harder time producing comparable metrics, using visualization of the eye-tracking data can produce more information when combined with standardized questionnaires [44].

5.1 Dataset

During our research of the related work, we were unable to find any standardized dataset for evaluating label rendering in 3D space, and even less for rendering on exploded views. Therefore, we built our own dataset by using the resources located in the open-source 3D model libraries BlendSwap and GrabCAD. The mentioned open-source 3D model libraries offer a rich collection of shared 3D models from a variety of topics. While the libraries are a rich source of 3D models that come in variety of formats ranging from CAD and mesh models to volumetric and parametric models, while Textplosion was built to only include load support for OBJ mesh models together with the material stored in MTL files. Therefore, we also needed to convert them into mesh models during their processing into exploded view models, which was done manually due to limitations of the back-end system. Most of the collected models were CAD models since they already contained a hierarchy of parts that were easily extracted and stored separately with additive manufacturing CAD software such as Autodesk's Netfabb. The remaining models were either first converted into CAD model formats and processed with Netfabb or loaded into Blender and cut up manually. This way we collected a variety of fifteen exploded view models, see Table 5.1. Processing them on the back-end side resulted in ten exploded views, see Table 5.2 and Figure 5.1, with generated directions and blockers information, while five models had geometry issues incompatible with the back-end system even after we had processed all 15 models according to given instructions. For all of the models we provided label information that was formed by generating a label for each part based on the geometry file name and later on by manually selecting a subset of labels and correcting some of them. This procedure produced label sets in multiple languages, due to the different origins of the models. We decided to take it as a side benefit, since it enabled us to make a dataset not favoring language groups. Since labels are used to convey information, participants of experiments could have used any hidden information inside labels to their benefit, which in turn implicates research results. The reduc-

tion of automatically produced label sets into their subsets was due to the distraction amount growing with higher label counts, which we determined to start around the count of 20 in Section 4.5.

5.1.1 Licensing

While open-source 3D model libraries provide shared 3D models, they are not necessarily allowed to be used in any kind of work or product. It is important to understand the licensing to prevent future problems regarding copyrights. Since we decided to provide our dataset as an attachment to the master thesis to be potentially used in future work of 3rd parties, we limited ourselves to models that gave enough freedom of use and sharing of the models. In BlendSwap we selected models licensed under a variation of the Creative Commons CC BY license, which allows us free use of the models by also attributing the original authors for their creation. GrabCad, on the other hand, rarely provides any licensing information per model, however their agreement policy during registration states that the agreement limits our use of the models. After getting in touch with the library administrators in detail we were allowed to use the models, since our intended uses went in line with the provided limitation details:

> "CAD models from the GrabCAD Community free CAD Library are generally for private use. Some examples of private use are educational self-directed learning, creating a concept design for internal use, etc. So of course, you're welcome to download these models and make local changes to them, but showcasing these models publicly does require some additional steps.
> For non-commercial but public use, like sharing on social media, including in a student presentation, or uploading to a free CAD library like GrabCAD, please make sure to attribute the author and include a link to the original model on GrabCAD."
>
> — Matt Firmani, administrator of GrabCAD

Code name	Original name	Author	Source
Brain	Cavernous Haemangioma-With Brain & Skull in Autocad 3D	Harvey Fonseca	GrabCAD
Chopper	Lego Helicopter	Kenny	GrabCAD
Digestive	GI Tract	Magnacad LLC	GrabCAD
Dragon	Laser-Cut Dragon Wooden Toy	trinityscsp	GrabCAD
Drone	Surveillance Hexacopter	Ayoola Adefemi Olaolu	GrabCAD
Engine	Wankel Rotary Engine	Jabir Mahmud Siam	GrabCAD
Eye	Human Eye Model	Bobby Dyer	GrabCAD
F1	Formula 1 Car-assembled	DHANASEKAR VINAYAG-AMOORTHY	GrabCAD
Gpu	MSI GTX 1060 Armor Graphics Card	Rick Lin	GrabCAD
Gun	Semi-auto weapon	Saffet Firat	GrabCAD
Kiosko	Kiosco	Tupay	BlendSwap
LegoGuy	LEGO GUY	SRBrandon	BlendSwap
Oscar	Oscar Award Statue	androgenius23	BlendSwap
Pocketwatch	Pocketwatch (all inside parts included)	Adrian	GrabCAD
Prezz	prezz	Kerbl et al.[7]	TUG
Tractor	Tractor	Ahmed Diab	GrabCAD

Table 5.1: 3D models used during development and evaluation, while also used as examples inside the thesis. All GrabCAD models are used under an agreement presented in Subsection 5.1.1, while Kerbl et al.[7] provided the `prezz` model as a learning example for the back-end system and allowed its use for the thesis and dataset. Tupay and androgenius23 on the other hand shared under the CC-BY license, while SRBrandon shared it under the CC-BY-SA license. Sources also include hyperlinks to the original files.

Figure 5.1: Images of the whole dataset shown in Table 5.1. From left till right top till bottom we have the models: `brain`, `chopper`, `digestive`, `dragon`, `drone`, `engine`, `f1`, `gpu`, `gun`, `kiosko`, `legoguy`, `oscar`, `prezz`, `pocketwatch` and `tractor`.

Model	Label Count	Language	Part Count	Polygons
Chopper	13	English	142	648,892
Digestive	6	English	6	98,658
Dragon	13	Slovene	147	82,921
Eye	10	English	10	12,330
F1	15	Polish	20	43,474
Gpu	7	English	7	36,370
Gun	14	Slovene	45	319,038
PocketWatch	17	Spanish	136	142,188
Prezz	23	English	23	19,866
Tractor	18	English	47	218,672

Table 5.2: Dataset of ten 3D models successfully processed by the back-end system that we wish to share with the community.

5.2 Visual Comparison

The dataset was first used to observe the behavior of our proposed solution in comparison to floating label and hedgehog labeling algorithms. Since Textplosion is an interactive application, naturally the best comparison can be seen during a live presentation or through recordings, but only outside of this thesis. An alternative way, which can also be included inside the thesis, is a visual comparison of multiple images taken during the applications execution stages. While a lot of information about the behavior during the time difference between images is lost, we can still notice obvious improvements. Since we built clustered hedgehog with complex exploded views in mind, we present the comparisons between algorithms on the two complex `chopper` and `dragon` models, each having around 140 model parts. Naturally we also show results for simpler models but limit it to the `brain` model.

To make a feasible comparison between algorithms, we defined three stages for all of the models: initial, exploded and end stage. The initial

stage is present to only observe the initial differences between the settings as well as the changes that happen when the next stage arrives. From the initial stage, we simply used the explosion animation to prepare similar exploded views. After exploded stage was reached, we counted the amount of labels, whose change in screen coordinates was low enough to be considered in same place or in its near local area of the space, when observed just by human eye. We called this empiric measurement explosion robustness, since we measured how robust the algorithms are to explosions. Another empiric measurement that we wished to achieve, was the algorithm's flexibility of handling limited free space, since the exploded views take up most of the available space. Its retrieval was done after the end stage was reached by comparing label spatial relationships with their referred part between the exploded and end stage. We defined the spatial relationship of a label as its relative position seen from the point of the referred part, where we only considered the direction with a threshold and not the actual size of the vector. Spatial relationship was also defined as an empiric description observable with the human eye such as in-front, behind and other spatial descriptions. By comparing the descriptions of spatial relationship at exploded and end stage, we arrived at the definition of flexibility as the count of preserved spatial relationships between stages.

For comparison we prepared Figures 5.2, 5.3 and 5.4, where the individual stages are presented in rows of a grid, while the columns represent the algorithms. Since the comparison is best executed by putting the images side by side, we resized the images to a significant degree. For simpler comparison, let us list all the present labels inside each of the used models before observing the results and use their list numbers as indicators during result reporting. Starting with the `chopper` model, we observe the following English instruction-like labels: **1.** axel x10 black, **2.** axel x12 black, **3.** axle link, **4.** axle peg 2x grey, **5.** block 10x1 technic red, **6.** engine black, **7.** exhaust gray, **8.** propeller black, **9.** seat blue, **10.** tail fin red, **11.** tyre small black, **12.** upiece 4x2 red and **13.** wheel small grey.

In regards to the `chopper` model, we observe that the floating labels

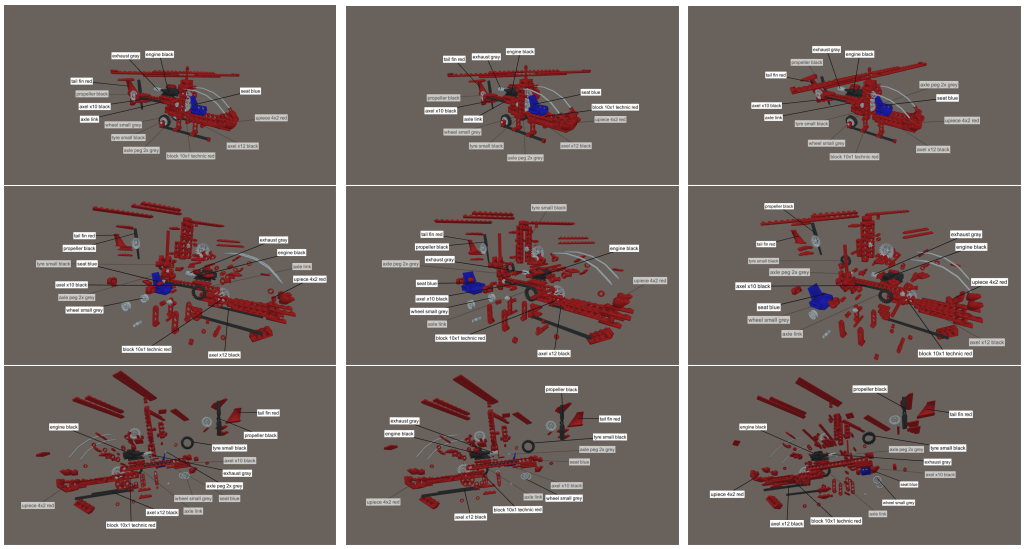


Figure 5.2: The grid comparison of label behavior between different algorithms in different stages for the `chopper` model as a representative of complex models. Each column belongs to one algorithm, where from left to right we have floating labels, hedgehog labeling and clustered hedgehog labeling results. The rows in turn represent the model's initial, exploded and end stage.

approach preserved three labels (8, 10, 13) in their local area, while having a flexibility of only preserving a single label's spatial relationship, namely for label 10. On the other hand, hedgehog labeling only preserved label 13 in its local area, while preserving the spatial relationships of six labels (2, 3, 4, 5, 6, 9). Clustered hedgehog showed much better results for this model by preserving the local areas of 6 labels (1, 2, 8, 10, 12, 13) after an explosion as well as showing a flexibility of 9 (1, 3, 4, 5, 6, 9, 10, 11, 13) or 70% of all labels. We can quickly check if the improvement between hedgehog labeling and our solution is due to the introduced clustering by observing how the labels counted by the flexibility are divided between clusters. In case of the `chopper` model we received the following three clusters: (8, 10, 11), (1, 4, 9, 13) and (2, 3, 5, 6, 7, 12). Out of the three clusters the second one

was wholly preserved its label's spatial relationships, which can point to a confirmation of clustering preventing sub-layout disruptions. The remaining clusters also had some spatial relationships preserved, which indicates that no reinitialization but only an update run of the hedgehog labeling limited to the cluster was executed.

The `dragon` model was described with the following Slovene annotations: **1.** Desna brada, **2.** Desna čeljust, **3.** Greben, **4.** Jezik, **5.** Kremplji, **6.** Leva brada, **7.** Leva čeljust, **8.** Noga, **9.** Oči, **10.** Rep, **11.** Smrček, **12.** Šapa and **13.** Vrat.

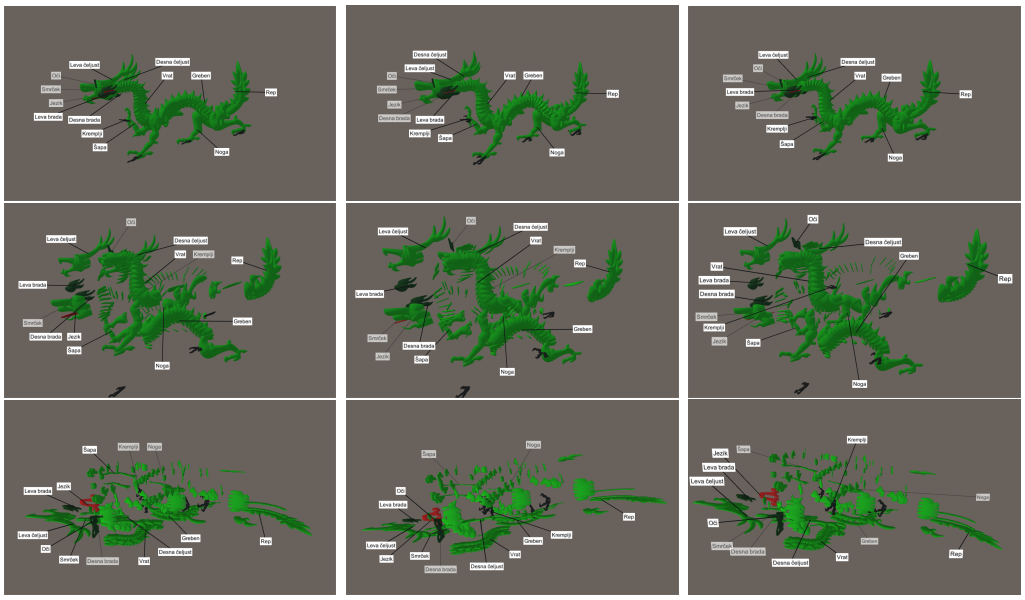


Figure 5.3: The grid comparison of label behavior between different algorithms in different stages for the complex `dragon` model. Each column belongs to one algorithm, where from left to right we have floating labels, hedgehog labeling and clustered hedgehog labeling results. The rows in turn represent the model's initial, exploded and end stage.

`Dragon` as an alternative complex model with different explosion directions presented similar results. Floating labels preserved the local area of four (2, 7, 12, 13) labels after the explosion and had a flexibility of two (2,

3) labels. Hedgehog labeling preserved the local area of two (7, 13) labels, while producing a flexibility of three (5, 6, 10) labels. Clustered hedgehog had eight labels (2, 3, 5, 6, 7, 8, 10, 12) preserving their position in local area, while producing lower results in regards to flexibility with only three labels (1, 3, 7). The same flexibility count as with hedgehog labeling can be accredited to unequally sized four clusters produced by our solution: (1, 2, 4, 5, 6, 7, 9, 11, 13), (3, 8), (10) and (12). It may be that the small clusters were in bad positions for a direct transition between stages, while the big cluster was simply too big to work well in limited space.

The final grid uses the `brain` model as the representative of simpler models. The labels are English medical terms: **1.** Cereb, **2.** Corpus, **3.** Frontal, **4.** Lesion, **5.** Occipit, **6.** Pariet, **7.** Pitua, **8.** Stem and **9.** Temp.

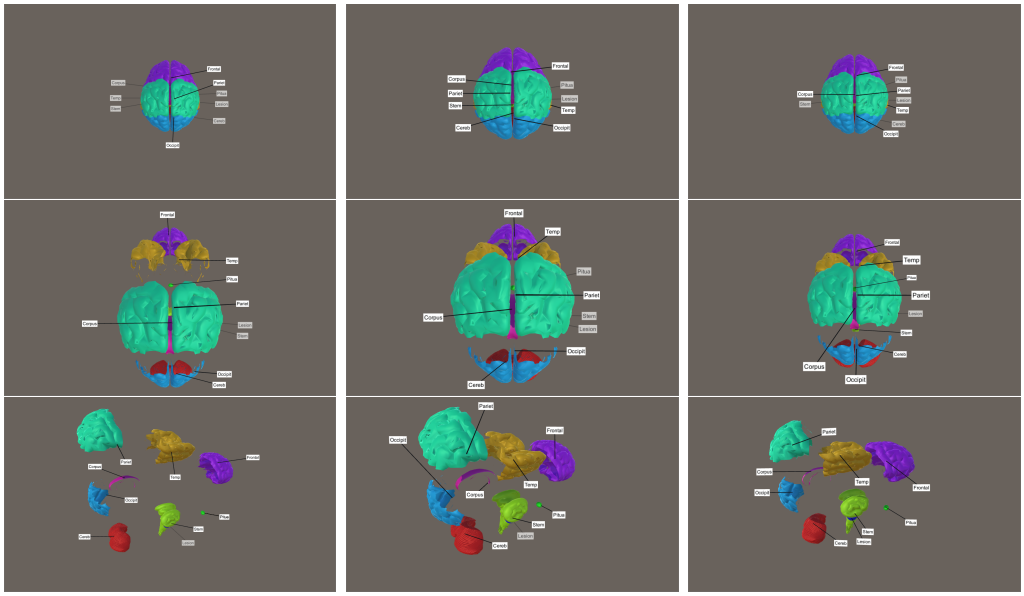


Figure 5.4: The grid comparison of label behavior between different algorithms in different stages for `brain` model, which is of relatively low complexity. Each column belongs to one algorithm, where from left to right we have floating labels, hedgehog labeling and clustered hedgehog labeling results. The rows in turn represent the model's initial, exploded and end stage.

Inside the `brain`'s grid we observed that floating labels only preserved a single label (7) in its local area, while five remained in their spatial relationships (1, 3, 4, 6, 9). Explosion robustness again caused trouble to hedgehog labeling, which preserved only two labels (4, 7) in their local area, while it again succeeded in its flexibility with six labels (3, 4, 5, 6, 7, 8). Our solution survived the explosion situations in the brain with five labels (1, 3, 4, 6, 7), while producing same numeric results with its flexibility as hedgehog labeling (2, 3, 4, 5, 7, 9). While the flexibility of our solution can be accredited due to two out four cluster preserving label spatial relationships: (1, 4, 7, 8), (2, 5), (6, 9) and (3). However due to the low amount of labels as well as low model complexity, the performance does not differ to hedgehog labeling's flexibility performance.

In general, these three visual comparisons show us that our solution seems to be solving the explosion based layout problems present in hedgehog labeling with a high degree of robustness, while in regards of flexibility it depends on the model's own complexity, on the complexity of its exploded view and the effectiveness of clustering. Regarding the effectiveness of clustering we mentioned in Sections 3.3 and 4.5 that the resulting metrics and their sets were a product trial & error and some logical reasoning. Therefore, there exists a possibility of better metrics and metric sets existing, which would improve the clustering and with it the flexibility for both simpler models and complex models.

While the visual comparison already gives some information on the improvement, we have to first take a look at the results of the user based study, which provides more information on the differences between algorithms based on the experience of multiple participants. By combining the results of the visual comparison, subjective feedback from questionnaires and objective measurements from eye-tracking, we shall arrive at the end to a final verdict for our solution.

5.3 Experiment Design

We defined the experiment to involve two kinds of tasks based on the user interaction: **observer** and **interactive** tasks. In observer tasks **ABC** and **Track**, the test subject was only able to observe a Textplosion interaction simulation, while in the interactive task **Exploratory search** the subject interacted directly with Textplosion via keyboard and mouse controls to finish the given task. Since the topic was new to most test subjects, we decided to get users more familiar with the topic by starting with observation tasks before proceeding to interactive tasks.

During task execution, we measured objective measurements such as time duration, pupil behavior and eye movement with our eye-tracking code, while after each task completion, we also gathered the test subject's subjective experience with the standard Single Ease Question (SEQ) and NASA Task Load Index (NASA TLX) questionnaires. Due to strict standards on NASA TLX, we decided to use its paper version to simplify the whole process, and therefore also applied the paper version concept to the remaining prepared questionnaires as well as other paperwork observable in appendix B.

SEQ is a standard single question used in application usability tests. It is defined as a 7-point rating scale to assess how difficult overall the test subjects found a task immediately after completing it. Despite its simplicity, in previous works it performed as well as or even better than more complicated measures of task-difficulty. From observing a wide range of studies it showed that the average SEQ score is around 5,which to us denoted average difficulty [45, 44].

While the SEQ measures difficulty on a scale from 1 to 7, NASA TLX on the other hand consists of six divided difficulty perceptions scales that range from 0 to 100 points. With a rich history of being used in a variety of fields and studies for more than three decades, NASA TLX became a de facto standard test for difficulty and usability. The original NASA TLX would require us to weight the individual scales based on participant feedback. Since it would add additional difficulty to our experiment, we decided to

use a widely recognized variation of it, called RAW TLX. RAW TLX skips the weighting of scales phase and produces a value with a normal average of scales, which was shown to be more sensitive than the original NASA TLX in some studies [46, 47].

Besides the data retrieved from standard questionnaires we also retrieved some personal data from participants as well as their subjective impressions of the algorithms with custom questionnaires and open talk interviews. Our unstandardized questionnaire and the interview were meant for assessing the subjective preferences that we could have failed to notice if we only conducted standard tests. While such an addition may only produce observations in the form of "I do not like this" or "this was fun", to the developers it can indicate an overlooked problem that might have occurred due to them focusing too much on other development tasks. The difficulty of the initial experience is also regarded as important information that we gather during the interview. Pagulayan, Steury, Fulton and Romero [48] noted that the initial experience of applications is an important factor for motivating users to use the application.

Due to unknowns in cluster metric sets mentioned in Section 3.3, we first ran pilot experiments of all tasks on different clustering subsets to determine the one used for the main experiments. The pilot experiment was conducted on a much smaller scale on the members of the LGM laboratory, Faculty of Computer and Information Science at University of Ljubljana, as well as on a few early outside volunteers. While the main experiment was done solely in Slovene, the pilot experiment was also conducted in English due to international participants. Another difference was also the presence of changes among the participants, since we were fine tuning and testing the instructions as well as the task progress and code debugging. Combining both changes in execution and a scale that is useless for statistical hypothesis testing made the results of the pilot run not worthy of detailed discussing in this thesis, however the ranking of clustering metric sets did help us to decide on the metric set used during the main experiment. Naturally, the

main experiment did not have such abnormalities and followed the same pattern which was set at the end of the pilot experiment.

5.3.1 Task Order and Randomization

For the usability test and other kinds of psychological tests, we always need to beware of the learning effect due to the patterns present in the experiments. While total randomization inside and outside of the tasks can solve this issue, smart randomization had to be implemented to still retrieve useful data. Since we wished to retrieve subjective impressions about the presented methods, we needed a way for the test subject to know which method we are asking them about. Therefore, we did not randomize the order of everything inside a task. To enable such subjective questions as which method approach was better, we grouped the subtasks into method groups, where each group consisted of randomized models and the same method. To prevent the learning effect between method groups, we also randomized the method group order. Since we had grouped the subtasks, we also needed a smart way to randomize the method group order. We decided to use the Latin square design [49] for method group ordering. The Latin square was also the origin of the idea to rename the methods by using single letter names from the set {A,B,C} by the order of the method group. Naturally, the Latin square method order differed between the test subjects groups as well as between the tasks in order to prevent any leaks of information from occurring between the tasks or test subjects. In practice we simply shuffled to the next Latin square pattern when going to the next experiment group and by shuffling to the next Latin square pattern for the next task.

5.3.2 Experiment 3D Model Selection

From our dataset of back-end exploded 3D models, see Table 5.2, we limited ourselves to five models due to experiment time constraints. To prevent influencing the results by randomly selecting the models during the experi-

ment execution, we limited all experiments to a preset selection, where we focused on extending the variety of complexity in the models, the languages in labels as well as the areas of origin (automotive, medical, games, education, ...). The resulting subset of models for the experiment was finalized with `digestive` as the demonstration model and `eye`, `f1`, `tractor`, `pocketwatch`, `gpu` as the main experiment models.

5.3.3 Labeling Algorithm Selection

The goal of the experiment was to prove that clustered hedgehog labeling performs better than normal hedgehog labeling and floating labels. Since Textplosion was buildtby extending the floating labels into hedgehog labeling and that further into clustered hedgehog labeling, we just needed to set the flags correctly to switch between the algorithms, see Table 5.3.3 in Appendix D for details on flags. However, in Section 3.3 we noted the problem of the unknown quality of the metrics used inside clustering. Just by limiting ourselves to the three proposed metric subsets from Section 4.5, we already arrived at five different possible methods to test during the experiment:

1. floating labels,
2. hedgehog labeling,
3. clustered hedgehog labeling with position metric subset,
4. clustered hedgehog labeling with exploded position metric subset,
5. clustered hedgehog labeling with combined metric subset,

Testing five conditions would have worked against us during the Latin square randomization and result processing. For the first we would have to build a 5x5 Latin square matrix, where rows would be the conditions and columns individual runs or test subject groups to make the learning effect reduced correctly. Since for a study to be considered reliable, one needs at least from 15 till 24 participant by a rule of the thumb as seen in studies

presented in Chapter 2, which in terms of a standard Latin square design with size of 3x3 [49] would mean 7 participants per experiment group. Going by that estimation of minimal participants per group, we would suddenly need at least 35 participants to make it reliable enough. At the same time it would have required almost twice the time to be spent on each task in comparison to a version with the standard Latin square. And finally the most important reason would be the short capacity of the human brain to keep information about the impression between different method groups that behave similarly over a longer period of time. While asking participants to rank three algorithms may already cause hesitation on their side, requesting to rank five of them would have resulted in unreliable results. All such conditions would also have to be considered during the statistical processing, while having little promise of any results.

Therefore, we limited ourselves to three methods as per standard design [49], where two spots were immediately filled the alternative algorithms, since we wished to produce a comparison to them. To resolve the issue of selecting an appropriate metric subset for the third condition of clustered hedgehog labeling, we took use of a small pilot experiment we ran for finding faults in the experiment environment as well as in the tasks beforehand. To determine the best subset, we ran the pilot test using only clustered hedgehog labeling methods:

A clustered hedgehog labeling with position metric subset,

B clustered hedgehog labeling with exploded position metric subset,

C clustered hedgehog labeling with combined metric subset.

From the comments, questionnaire results and further logical reasoning, we determined method B as the best subset. While method C was also close in subjective results, both were much better than method A. Since the pilot experiment's sample of experiment runs was too small for statistically meaningful results, the sample and statistical tests run on it could not be used as

concrete evidence in the thesis. Therefore, their processing was not included in Section 5.8, even though we used the same procedures. However, using the subjective results only as pointers to possibilities and adding participant comments and logical reasoning to it enabled us the selection of a seemingly safe metric subset for the main experiment. Following the result we reasoned that the positive effect of the focus distance seemed questionable since explosion directions are defined independently of the whole mesh center position. While we reasoned that the distance from the focus point could indicate 3D space sectioning as well as indicate the progress of an explosion, we did no research to use as actual evidence for proving or rejecting the concept. Combining this uncertainty with little or none improvement in the subjective measures became the tipping point for us. Therefore, we chose method B for the main experiment and thus defined the three method groups as:

A floating labels,

B hedgehog labeling,

C clustered hedgehog labeling with exploded position metric subset.

5.3.4 User Input Simulation

Since the observation tasks focused on eye-tracking, we needed to provide a way of simulating the user interactions that would achieve consistent results between the task runs. We created a system that reads a predefined scenario format file to determine the parameters of a simulation of a user using our application, see Appendix C for scenario file format. The system was further extended to ensure the consistency of experiment behavior during the eye-tracking tests with randomization. The randomization of subtasks inside the method group was achieved using Fisher-Yates shuffle [50] as seen in Algorithm 3.

Algorithm 3 The modified Fisher-Yates shuffle skipping demonstration model

Require: model count N and model array M

```
1: n = N - 1;
2: // prevent demonstration model (index 0) to be shuffled
3: while n > 0 do
4:     // select random element in range is [1,n)
5:     k = RandomIntFromRange(1, n);
6:     // shuffle the elements
7:     tmp = M[k];
8:     M[k] = M[n];
9:     M[k] = tmp;
10:    n = n − 1:
11: end while
```

5.3.5 Eye-tracking

According to several pieces of literature [16, 17, 18] eye-tracking is not of recent design. It dates back to 1878, when mechanical torture-like devices were used to observe the eye position changes. The first practical example of a study using cameras for eye-tracking beyond photographs goes back to 1947, where observation of film recordings was done to follow a pilot's eye movement during a flight. The use of computers for eye-tracking was already producing results in the 1960s, but the actual modern concept of eye-tracking as we use it today in UX testing was established in the late 1990s [16, 17, 18]. In recent years due to commercial eye-trackers on the market a boom in eye-tracking happened, which can be observed also in the growing trend of its use inside visualization and CGI research, as observed by Kurzhals et al. [15].

Using the current trend in our favor, we borrowed the eye-tracking hardware Tobii Pro Eye-tracker 4C, which we attached to the bottom of the computer screen to record gaze positions on the screen. The recording of coordinates was achieved by using the Tobii Pro SDK asset for Unity. We modified its prefabs for tracking screen eye location to also store the exploded view labels application coordinates, the clustering information, the timestamps, the experiment run details and other Textplosion information

besides its initial tracked eye gaze position and pupil behavior, into multiple XML files. The XML files were generated for each separate subtask and stored inside folders with task unique identifiers, which was simply produced by concatenating the date and time values like a custom timestamp. Since we had yet to end our eye-tracking metrics research, we simply stored any information that seemed useful at the time. This also allowed us to later on recalculate the values missed during storing due to coding mistakes, saving us from losing any of the metrics from Appendix E. However, such a safety measure in the end produced over 67 GB of XML files, which was reduced to 1 GB during processing. The data processing was achieved using our own python code together with the open-source projects SciPy and PyGazeAnalyser. The first one was used for statistical analysis, while the second one was used for simple eye-tracking data processing in regards of determining fixations and saccades as well as for ploting them in the form of heatmaps, fixation plots, scanpaths and raw data plots.

5.3.6 Experiment Procedure

The general experiment procedure was defined by the following fifteen steps that the experiment supervisor and test subject needed to follow:

1. Introduce the subject to the topic with the help of the introduction images attached in Appendix B, and describe the general procedure of the experiment. Also explain the questionnaires.

2. Let the test subject fill out the agreement paper for collecting data and audio/video recordings during the experiment. Start recording the session with the audio recording software for purposes of processing the results.

3. We ask the test subject to fill out the questionnaire regarding basic personal information such as age, gender, experience with 3D modeling/ CAD software, see Appendix B.

4. The test subject sits down comfortably in front of the computer screen and starts Tobii's calibration program before running theTextplosion experiments to setup the computer screen angle and height according to their eye position.

5. The supervisor starts the loading of the Textploson experiment for test subject's experiment group. While the experiment is loading, the the test subject needs to read instructions for the current observation task while also receiving a vocal explanation from the supervisor. The order of observation task is always the same: the ABC task comes first, followed by the Track task.

6. Another calibration is run after the experiment finishes loading. This is repeated before each method group starts. Constant calibration was added to allow the test subject to move their head while filling out the questionnaire.

7. After the calibration is done, a start screen appears to notify the test subject that they can start with the experiment task. The same black screen with instructions appears between all subtasks to allow the test subject to relax and recheck the instructions if needed. The task begins by the subject again pressing the on-screen noted key SPACE.

8. When the test subject finishes a randomly ordered subtask method group, they will be notified by a gray screen that the method group was completed and that they should fill out the SEQ and NASA TLX questionnaires. The gray screen is a fail safe that prevents jumping to the next method group too fast. The questionnaires can be observed in Appendix B. Take note to write down the unique identification number presented on the gray screen.

9. After the filling out is done, the subject proceeds by pressing space to run the calibration again and to access the next method group. Repeat

step 8 and 9 until the last method group is finished. The end will be marked by another gray screen.

10. After the observation task is completed, the test subject needs to write down the ranks of individual method groups, where they give the highest rank to the method group that they found the easiest of the whole task. In the ranking questionnaire, attached in Appendix B, there is also a section for writing down the task identification number presented on the gray screen.

11. If the completed task was task ABC, then load the Track task experiment and return to step 5. Otherwise, load the last Exploratory search task.

12. We explain to the test subject that now they will be interacting with the application by using the keyboard and the mouse. As well as providing the task instructions with the interaction layout included, which can be observed in Appendix B.

13. We give the test subject enough time to get familiar with the controls and the task by using the demonstration model. We also explain the transparency concept of Textplosion, while for the remaining tasks, the test subject is limited to one minute to finish a subtask. During a subtask, we write down the vocally given exploratory results.

14. The test subject finishes the last task by reaching to to the gray screen since in the last task only our method is tested.

15. The test subject fills out the final SEQ and NASA TLX questionnaires as well as our case specific questionnaire. After filling out everything, the supervisor leads a short open talk interview based on the written answers, and also asks for any opinions about the experiment as well as the solution.

5.3.7 ABC Task Scenario

In the ABC task, we tested on the intuitiveness of label placement. The idea was to indicate three labels for to the test subject to remember in three seconds, and to run a three second long animation of rotations, zooming and explosions. After the animation finished and there was no change in the Texplosion view, the test subjects needed to find the labels again during their allocated ten seconds. When all three labels were found, the test subject notified the system by pressing the ESC key and moving their eyes from label to label in order to form a triangle with their gaze. The gaze triangle was added to have a fail safe besides the ESC key. Altogether, a single subtask amounted to 18 seconds, which in turn for 18 subtasks in the task results in 5.4 minutes of concentration time for the whole task.

5.3.8 Tracking Task Scenario

On the other hand, the tracking task ignored the behavior of the scene since it was about focusing on a single label and following it for 14 seconds. The label intended for following was indicated for 2 seconds in the initial state of the exploded view, which resulted in 16 seconds per subtask, or in turn for 18 subtask resulting in another 4.8 minutes of test subject concentration time. During this concentration period, the subject had to find the label again if they lost it during the simulation proceeding.

5.3.9 Exploratory Search Task Scenario

The only interactive task required the user to get familiar with the controls in a short period of time, which sadly in turn caused quite some stress for the test subjects during the actual experiment execution. The task by itself was a classical usability experiment, where the test subject tried to explore the models to finish the challenge. The challenge was for the test subject to find the model part corresponding to a given label text. When the test subject found the referred part, the supervisor further asked them to name

any other part that seems to be the neighbor of the found the part. We defined a neighbor as a part of the model that is by eye discernible to be in contact with the previously found part or is almost touching it. Since it was an exploratory task, there where no right or wrong answers, and the test subject was asked to name all the parts that they were sure about being neighbors while the experiment supervisor was tasked with writing down the answers.

While the demonstration had unlimited time to get the test subject familiar with the system, each later subtask was limited with minute designated time slots, which could be shortened by pressing the ESC key after declaring all neighbors to be found. With 5 minutes dedicated to the subtask and an arbitrary time for the demonstration, which was usually less than 5 minutes, we arrived at a time range of 5 to 10 minutes of concentration time needed from the test subject.

5.3.10 Experiment Scenario Time Constraints

It was said to be a good practice to limit such user based experiments to 15-20 minutes due to human task focusing limit. When asking about the clear focus time required from the test subject for all tasks, we just barely fit in this time range. However, the whole experiment including the task concentration, the instruction reading, the time for filling out the questionnaire and so on resulted in the experiment length ranging from 45 to 60 minutes per test subject due to the

5.4 Experiment Hypothesis

The experiment was built around the null and alternative hypothesis:

$$\boldsymbol{H_0 : CHL\ performs\ similar\ to\ alternative\ algorithm} \tag{5.1}$$

$$\boldsymbol{H_A : CHL\ performance\ differs\ from\ alternative\ algorithm} \tag{5.2}$$

We wished to prove that our solution is better than previous solutions. When means, distributions or predicted probability differs in small amount between methods and the resulting p-value from corresponding statistical test shows a p-value higher than the significance level of $\alpha = 0.05$, we cannot reject the H_0. Not rejecting H_0 goes against our desired outcome, since it indicates that our solution produces label layouts with similar performance inside exploded views as previous solutions. In case of the difference being high enough to produce a p-value below the significance level of $\alpha = 0.05$ and therefore rejecting the H_0, we are aligned with our wish if the difference shows improvement, since we proved that the difference is statistically significant enough to matter. When comparing subjective measurements retrieved from questionnaires, namely SEQ and RAW TLX, lower values mean a better result. Therefore, rejecting the H_0 due lower mean value of RAW TLX is a prove in improvement. In case of the objective results from eye-tracking, the behavior differs between the 35 metrics described in Appendix E.

5.5 Experiment Environment

The experiment was performed inside the LGM laboratory during the August general holidays, which resulted in a peaceful and well controlled environment. The stimuli presentation as well as the eye-tracking software were run on a Windows Pro desktop machine, since the Unity Tobii SDK asset was only available for Windows. The testing machine was an all-in-one PC with integrated Intel(R) HD Graphics 4400 family GPU with 1 GB of memory. The main CPU was a 64-bit 3GHz Intel i3 with 8 GB of RAM memory. The provided display resolution was 1920x1080 on a 23 inch display with a 60Hz frame rate. Textplosion was run in Unity Editor due to the runtime loading limitations preventing a successful build at the time of the experiment. The cause of the limitations was due to the development version referencing the resources located outside of Unity project's asset folder for practicality with

3D model handling. Via USB we connected the eye tracking hardware Tobii Eye Tracker 4C and attached it to the bottom of the screen. It has an operating distance of about 50 to 95 centimeters with support for screen sizes between 20 and 37 inches. It takes measurements with a 90Hz rate and with the support of infrared illumination, where it achieves tracking of 97% of the population.

5.6 Participants

In the main experiment we gathered 28 participants, out of which 15 were male and 13 female. Their characteristics can be seen in detail in Figure 5.5. The age of participants range between 15 and 60 years old, while 90% or 25 participants were in the age range between 17 and 29 years old. In regards to their experience with 3D applications that offer HCI with a 3D world seen via a 2D screen, only five participant or 18% had none before the experiment, while another five participants only knew such interactions from 3D games. 3D games were counted as valid experience since we were asking about their familiarity with using a mouse and keyboard for interacting inside a 3D coordination system. Interestingly, we succeeded in gathering participants with a relatively high percentage of 43% having experience with CAD programs, while with more art designated programs like Blender, Maya or Unity only four participants had experienced them, the experience with Unity being limited to just one.

5.7 Subjective Results

After combining all of the subjective ranking inputs from questionnaires, we arrived at the ranking distribution for individual given ranks as seen in Figure 5.6. Inside the distribution, we see that our clustered solution receives the biggest amount of 1st ranks in both observation tasks. When compared to the floating label rank, it shows the biggest perceived difference

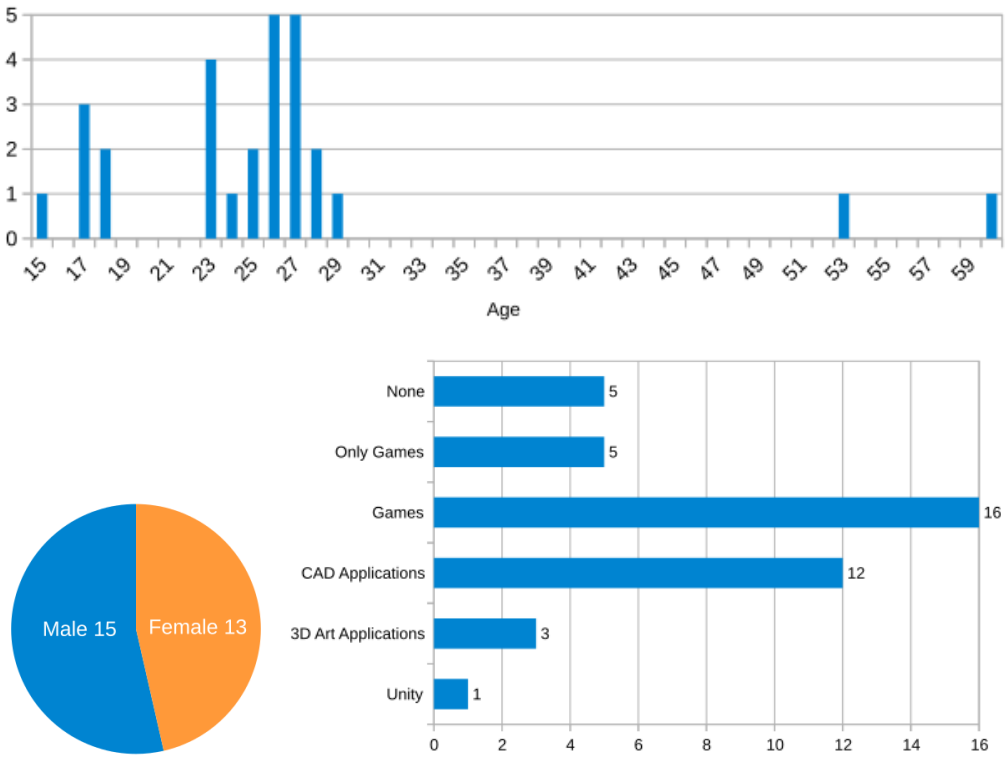


Figure 5.5: The 28 experiment participants' gender ratio is almost equal with 90% of them in the age range between 17 and 29. Regarding their experience with 3D applications, only five have none and another five are limited to 3D games, while 12 or 43% had already experienced 3D applications for model modifications with a high count of CAD based experiences.

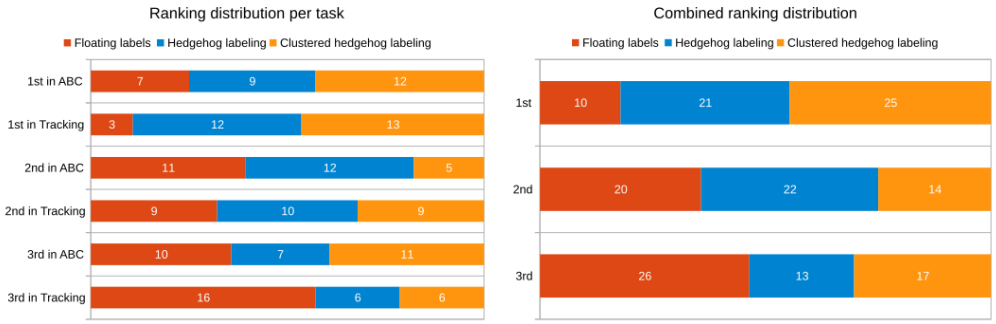


Figure 5.6: The count of subjective user ranks per method creates a distribution of the 1st, 2nd and 3rd ranked algorithms according to task and when combined.

in performance with a lead of five in the ABC task and a lead of ten in the Tracking task, giving a task combined rank difference of fifteen. It also received more 1st ranks, but with a much smaller difference of three for the ABC task and a single one for the Tracking task, generating a task combined ranking difference of only four. We already anticipated participants of marking hedgehog labeling and clustered hedgehog labeling closer together than floating labels, since clustered hedgehog labeling is by design multiple hedgehog labeling running in divisions of 3D space, while floating labels have only the reinitialization run of our solution.

To see the statistical significance of the results, we first slightly modified the null hypothesis 5.1 and 5.2 to better match our data as:

$$\boldsymbol{H_0 : The\ probability\ of\ CHL\ recieving\ 1st\ rank\ is\ \frac{1}{2}} \tag{5.3}$$

$$\boldsymbol{H_A : The\ probability\ of\ CHL\ recieving\ 1st\ rank\ is\ not\ \frac{1}{2}} \tag{5.4}$$

We selected a significance level of $\alpha = 0.05$ and calculated the ranking p-values with python's statistical module SciPy's binomial tests[49] pair-wise with clustered hedgehog results. A pair consisted of the success count being the count of 1st ranks for clustered hedgehog labeling and the failure count

as the count of 1st rank for the paired algorithm. The resulting p-values of pairing up with floating labels were **0.3592** for the ABC task and **0.0213** for the Tracking task, which meant that we could only prove the rejection of H_0 for the Tracking case. This meant that when compared to floating labels, our solution performs better in tracking tasks already on subjective impression, while intuitiveness of label placement does not improve but remains of similar subjective impression. Comparing the results with normal hedgehog labeling, on the other hand, produced p-values of **0.6636** for ABC task and **1.0000** for Tracking task. Therefore we cannot statistically prove that the slight difference in rank counts as statistically significant.

Sample	**Floating labels**	**Hedgehog labeling**	**Clustered hedgehog labeling**
ABC SEQ	0.8261	0.3291	0.1805
ABC RAW TLX	0.9872	0.1168	0.6459
Tracking SEQ	0.2306	0.5555	0.3370
Tracking RAW TLX	0.2652	0.1077	0.4054

Table 5.3: Normal test p-values for the SEQ and RAW TLX samples across the different tasks and methods.

Before determining the differences in the difficulty and workload described by SEQ and RAW TLX values, we needed to determine the distribution inside the samples to determine the way to examine them. By performing a normal test [51] on all by task and method group separate samples, we tried to prove the null hypothesis H_0 for each sample being of normal distribution with a significance level of $\alpha = 0.05$. The results observable in Table 5.3 show all p-values not rejecting the H_0 and thus confirming the normal distribution. Therefore, we could compare the means by t-test and standard deviations with the wilcoxon test[49].

In Tables 5.4 and 5.5, we present the SEQ and RAW TLX mean values

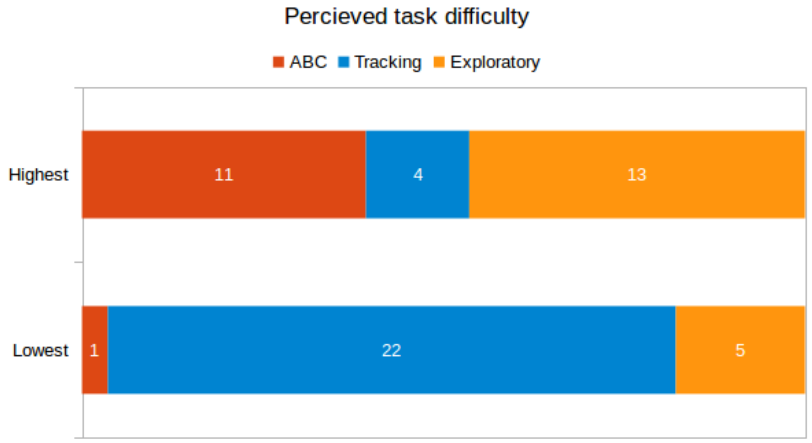


Figure 5.7: The participant-perceived most difficult and easiest task collected during the final questionnaire fill-out.

accompanied by standard deviation. The measurements describe the impression of task difficulty and the required workload that participants subjectively perceived after finishing the individual tasks. Both the mean values as well as the perceived impression collected in the final questionnaire, presented in Figure 5.7, show that the least difficult task was indeed the tracking task, as one could expect from the task of focusing on a label. On the other hand, all measurements point towards the Exploratory task as the most difficult one, closely followed by the ABC task. We found out from the experiment audio recordings that the main reason for determining the difficulty of the two was finding all of the labels in case of the ABC task, which aligns with the main point of the task, while in case of the Exploratory task, the biggest difficulty was remembering and getting familiar with all the possible interactions and controls that Textplosion allowed them to use in such a short time, which was not main point of the task.

Regarding the differences between the method groups in observation tasks, we noticed the differences being incredibly small. In the case of comparing our solution with floating labels, we only see a 0.3 SEQ mean difference and a 0.7 RAW TLX mean difference for the ABC task, while the Tracking task

SEQ values	**Floating labels**	**Hedgehog labeling**	**Clustered hedgehog labeling**
ABC μ	3.5714	3.4643	3.9286
ABC σ	1.3997	1.2672	1.5336
Tracking μ	3.4643	2.9286	3.0000
Tracking σ	1.5232	1.1931	1.1339
Exploratory μ	x	x	4.3929
Exploratory σ	x	x	1.3185

Table 5.4: SEQ values per algorithm.

RAW TLX values	**Floating labels**	**Hedgehog labeling**	**Clustered hedgehog labeling**
ABC μ	39.3452	37.6190	40.03
ABC σ	15.2401	17.3358	18.967
Tracking μ	33.4226	24.8214	26.6250
Tracking σ	15.8542	14.0195	13.8681
Exploratory μ	x	x	47.1131
Exploratory σ	x	x	17.5530

Table 5.5: RAW TLX values per algorithm.

had a SEQ difference of 0.5 and the highest TLX difference, which measured a whole 7 points. Comparing our solution with hedgehog labeling produced a SEQ difference of 0.5 and a RAW TLX difference of 2.5 for task ABC, while the Tracking task had the smallest SEQ difference of 0.1 and a RAW TLX difference of 2. In overview, we see that the SEQ mean difference had a range of [0.1, 0.5], while the RAW TLX mean difference had the range [0.7,7]. In terms of scales, we see that both the 100 point RAW TLX scale and the 7 point SEQ scale produce the highest differences to be at 7%, which is not high. So even though in the case of normal hedgehog and our solution we found most of the means to favor normal hedgehog, there was still a possibility of this only being a result of the sample.

To see if there is any significance in the mean differences, we transform the original 5.1 null and 5.2 alternative hypothesis into:

$$\boldsymbol{H_0 : CHL's\ \mu\ equals\ to\ alternative\ method's\ \mu} \tag{5.5}$$

$$\boldsymbol{H_A : CHL's\ \mu\ differs\ from\ alternative\ method's\ \mu \frac{1}{2}} \tag{5.6}$$

The result analysis has been carried out by using a significance level of $\alpha = 0.05$ using the wilcoxon test to determine the SEQ p-values and the t-test for the RAW TLX p-values. All the calculated p-values when pairing up all methods together with the previous ranking p-values can be observed in Table 5.6. Inside the table we can observe that most SEQ and RAW TLX values are far from $\alpha = 0.05$ and that the only exception, having a p-value of **0.0063**, was the result of comparing the TLX RAW mean between floating labels and our solution for tracking task, where our solution produces on average 7 points less of workload. Close but still not significant is the same task's comparison of the SEQ values, which produces a p-value of 0.0618 just slightly above our selected $\alpha = 0.05$. While the second measure is statistically not significant enough, it still gives support to the previous observation of our solution producing better results in tracking tasks already on subjective

p-values	Floating labels	Hedgehog labeling
Ranking 1st in ABC	0.3592	0.6636
Ranking 1st in Tracking	0.0213	1.0000
Combined 1st ranking	0.0167	0.6587
SEQ ABC	0.1905	0.0739
SEQ Tracking	0.0618	0.7033
RAW TLX ABC	0.7962	0.2666
RAW TLX Tracking	0.0063	0.3382

Table 5.6: Statistical significance of the results was tested by comparing, as defined by null hypothesis 5.1, clustered hedgehog labeling results with results of floating labels and hedgehog labeling separately. To see, if we can reject the null hypothesis, we selected a significance level of $\alpha = 0.05$ and calculated ranking p-values with binomial tests, SEQ p-values with wilcoxon test and TLX p-values with t-test.

impression level, especially when accompanied with the significant RAW TLX p-value. On the other hand, the comparison with normal hedgehog labeling also produced an insignificant but close to significant p-value of 0.0739 for the ABC task's SEQ value in favor of original hedgehog labeling by 0.5 point. While this p-value is not significant enough to denounce our solution to rise the difficulty by 7% for tasks of finding labels after being repositioned, it rises a doubt, which we tried to clear with our objective results.

The notion of a small difference between the algorithms is also observed in the results regarding label stability questions presented in Figure 5.8, where 61% of participants mark the label layout stability to be either the same or more influenced by the model than the method. While the leading opinion with 39% was that methods determined layout stability, it was a win by a single vote only. Combining it with the result of questioning the participants about the difference between the label layouts based on method

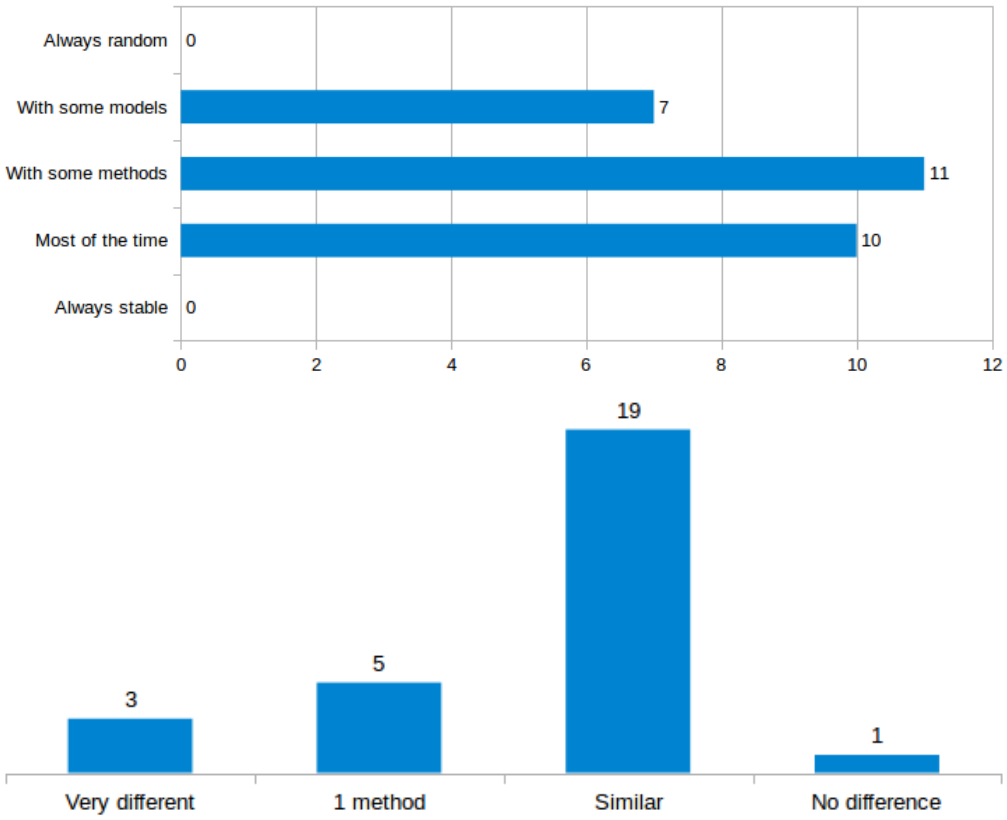


Figure 5.8: The top image presents the results regarding the question of observed label stability and which factor was more important. None of the participants selected the border options of there being no label stability or of all labels having a constantly stable layout, while most votes went for label stability to differ between methods or for all methods and models being stable most of the time. Not far behind is also the influence of models option. When asking directly about the difference in the layouts between algorithms, we observed in the bottom image that the 68% majority of participants mark the methods to have a similar effect on the layouts.

groups, where 68% are of opinion that the label positioning layouts behave the same. There was a close run in opinion that there is no difference and that a difference exists between methods was observed. All together it goes in line with previous perceived difficulty and workload results as well as the fact that algorithms using the same core concept.

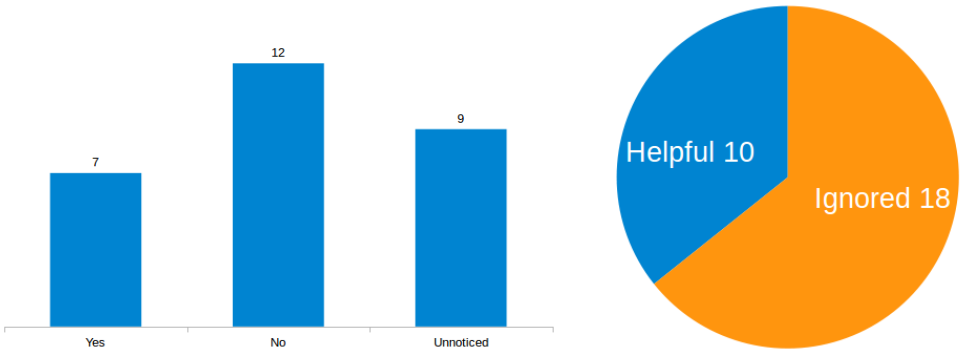


Figure 5.9: Results on label transparency questions indicate that 19 or 68% of participants noticed the transparency, meaning the contrast was high enough, however only 7 participants intuitively knew the meaning it represented just from observations. In the exploration task only 10 found it useful while 18 participants decided to ignore it.

Regarding the results in Figure 5.9 about the use of transparent labels, we found out that the majority or 68% of the test subjects did notice them during the observation tasks, however, according to audio recordings, most mistook them as our way of handling label overlaps during label transitions. Only seven participants, who had some CAD experience from before, guessed the invisibility due to similar usage in CAD programs. Even after explaining the usefulness of it, only ten participants used it with the Exploratory search task, while the others ignored it and none found it distracting. According to the audio recordings, the most likely reason seems to be the thick congestion of information presented before starting the last task. Remembering four explosion interactions, their controls as well as the camera controls and the task objective was simply too much for the test subjects to remember. This

goes in line with the previous observations of the last task contending for the most difficult task.

In general the subjective results show the three methods being at least equally good on the level of subjective impression. On the other hand there was not much statistical support for improvement from the normal hedgehog labeling implementation to our end clustered hedgehog labeling implementation. Subjective results only show statistically significant improvements in results, when concerning label jumping during Track task between floating labels an our approach.

Since overall SEQ estimation given at experiment end resulted in a mean value of 4.29 with 1.13 standard deviation, which is close in regards the official SEQ average difficulty positioned around 5, see beginning of Section 5.3, we confirmed the experiment design to be of average difficulty. With individual task SEQ values ranging between 2.92 and 4.39 with standard deviations from 1.13 and 1.53 confirming it again. The average difficulty can be also observed from the audio recording, where the majority of participants found the experiment to be fun, interesting and mostly of adequate pace. The pace comment was surprising, since we shown the experiment lasted for an hour for the majority of participants in Subsection 5.3.10. By pointing out the used up time to participants we received a surprised expression, which just confirmed their immersion into the experiment and its ability to cause the loss of feeling for time, which goes in line with the lack of negative adjectives appended to the experiment descriptions. The only reappearing negative feedbacks were in regards the interaction task being too short to get a good tryout of the system and the system layout of keyboard and mouse actions. The action layout difficulty was also described by participants as the reason for the high difficulty estimation of the interactive task.

5.8 Objective Results

Since the eye-tracking resulted in over 67GB of XML data, we first decided to extract only the relevant data into CSV form combined per user and task for faster processing before starting the metric calculations and their statistical significance. During processing, we had to keep the official Tobii SDK documentation in mind. An important statement from inside was that the coordinates of the gaze in the display position start in the top left corner of the display while Unity's texture coordinates start in the bottom left corner, as seen in figure 5.10.

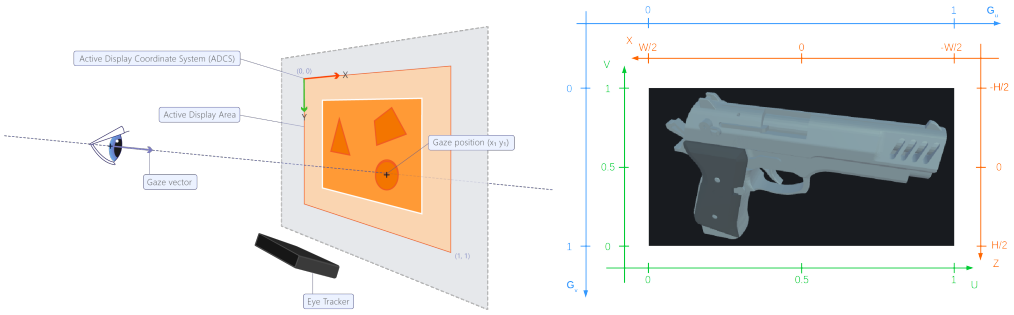


Figure 5.10: On the left we have an image from Tobii SDK's official documentation presenting its active display coordinate system (ADCS) in regards to the world and to the display area [52]. The right image is an extension of Figure 4.11 that shows the change in ADSC in regard to our system, where the orange coordinate system is the plane coordinate system in Unity's units starting in the plane center, which was recalculated before into the Unity's UV coordinate system in green. Additionally, we appended ADCS in blue, and we see that only a flip of the V or Y axis is needed in order to switch between the coordinate systems.

We adjusted the coordinate Equations 4.26 - 4.29 as:

$$U_{gaze} = U_{texture} = \frac{1}{2} - \frac{Vertex_{texture}.X}{Width} \tag{5.7}$$

$$V_{gaze} = 1 - V_{texture} = \frac{1}{2} + \frac{Vertex_{texture}.Z}{Height} \tag{5.8}$$

Since we wished to confirm the correct projections into the gaze coordinate system, we had to invert the label plane rotation following the Equations from 4.26 to 4.27 and afterwards used the new Equations 5.7 and 5.8. According to its official documentation, the Euler angle processing is done in Unity by first performing the rotations around axis Z, then X and lastly around axis Y. We defined the classical rotation matrices and combined them into a single matrix:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & -sin\phi \\ 0 & sin\phi & cos\phi \end{bmatrix} \tag{5.9}$$

$$R_y(\theta) = \begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix} \tag{5.10}$$

$$R_z(\psi) = \begin{bmatrix} cos\psi & -sin\psi & 0 \\ sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.11}$$

$$R(\phi, \theta, \psi) = R_y(\theta)R_x(\phi)R_z(\psi) \tag{5.12}$$

$$R(\phi, \theta, \psi) = \begin{bmatrix} cos\theta cos\psi + sin\theta sin\phi sin\psi & -cos\theta sin\psi + sin\theta sin\phi cos\psi & sin\theta cos\phi \\ cos\phi sin\psi & cos\phi cos\psi & -sin\phi \\ -sin\theta cos\psi + cos\theta sin\phi sin\psi & sin\theta sin\psi + cos\theta sin\phi cos\psi & cos\theta cos\phi \end{bmatrix} \tag{5.13}$$

Since rotation matrices are orthogonal matrices, which are square matrices Q defined by their property $QQ^T = I$, we can easily define the inverse rotation matrix as:

$$R^{-1}(\phi, \theta, \psi) = R^T(\phi, \theta, \psi) \tag{5.14}$$

$$R^{-1}(\phi, \theta, \psi) = \begin{bmatrix} cos\theta cos\psi + sin\theta sin\phi sin\psi & cos\phi sin\psi & -sin\theta cos\psi + cos\theta sin\phi sin\psi \\ -cos\theta sin\psi + sin\theta sin\phi cos\psi & cos\phi cos\psi & sin\theta sin\psi + cos\theta sin\phi cos\psi \\ sin\theta cos\phi & -sin\phi & cos\theta cos\phi \end{bmatrix} \tag{5.15}$$

After the data preprocessing was finished, we created CSV files separated by participant and task with rows of 38 or 64 columns depending on the amount of labels important for each task, which resulted in the reduction of 67 GB of data to only 1.2 GB. In case of the ABC task we stored the information of all three selected labels, while for the Tracking task and the Exploratory search task we only stored the information about the label the test subject had tracked or the label corresponding to the search term. For details on the format of the produced CSV files, see Appendix E.

5.8.1 Statistical Metric Comparison

By using the measures mentioned in the papers of related eye-tracking work mentioned in Chapter 2 we defined 35 metrics that are described in detail in Appendix E, while in this section we will focus only on those of statistically significant importance. For simplicity we shall introduce acronyms for the three algorithms. Floating labels is replaced by the acronym FL, hedgehog labeling by HL and clustered hedgehog labeling by CHL. We used a normal testt[51] on individual samples of data combined with the same model and algorithm as a starting point. All of the p-values results are observable in Tables E.1, E.2, E.3, E.4, E.5 and E.6. While some combinations have a p-values that do not reject the normal distribution in the null hypothesis, the majority of values do reject it. There are even cases where the same model confirms a normal distribution of metric values with one algorithm while rejecting it when paired with another algorithm.

The initial idea was to use a one-sided analysis of the variance, better known as ANOVA, to compare the metrics between algorithms, however ANOVA requires a normal distribution across samples due to its own assumptions. In such cases it is advised to use non-parametric equivalents, which in case of ANOVA is the Kruskal-Wallis H-test. Similar to ANOVA, we assume under the null hypothesis that all gathered samples have a similar distribution and fall into the same population based on the ratio of the inner and the between variance difference or, in this case, the H-statistic. While ANOVA

uses the mean for calculalting the statistic, the Kruskal-Wallis H-test uses the median [49].

Since we had a large variety of metrics across five different models, we first conducted a Kruskal-Wallis H-test to determine if the samples can be combined across models or algorithms without any changes being made to the distribution. This way we had tested the dependence of the results on the algorithm and the model. To achieve such a test, we combined all of the data samples into a single sample set while we prepared another two sample sets with numerical indicators of models and algorithms, as seen in Figure 5.11. By observing the p-values in Tables E.8 and E.7 we can see that all of the metrics are bellow the significance level of $\alpha = 0.05$ regardless if we just pair up metrics with just model indicators, algorithm indicators or both. Such a result indicates that all metrics depend on the model and the algorithm and therefore cannot be combined across either. In case of the models this simply meant some extra work, while we confirmed that the algorithms effect all of the metrics, which could indicate a difference between algorithm performances.

To observe the differences between the algorithms, we calculated the median and the standard deviation for each metric sample across the possible combinations of task, algorithm, model and metric, which produced results present in Tables E.13, E.14, E.15, E.16, E.17, E.18, E.19, E.20, E.21 and E.22. For determining the statistical significance of the observable differences between the algorithms, we did Kruskal-Wallis H-tests for each metric, where the samples used were simply the metric's value for the same model but separated by algorithms. The p-values of metrics that would reject the null hypothesis of the same population median indicate a difference among the distributions and therefore potentially indicate a significant difference produced by change in the algorithms on a given model. The p-values can be observed in Tables E.9, E.10, E.11 and E.12.

The observed significantly important metric differences between the samples for each model change a lot between the models and tasks. Comparing

Model	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
Algorithm	1	1	1	1	1	2	2	2	2	2	3	3	3	3	3
Metric	X_1	X_2	X_3	X_4	X_5	Y_1	Y_2	Y_3	Y_4	Y_5	Z_1	Z_2	Z_3	Z_4	Z_5

Figure 5.11: The concept of testing if the models and algorithms effect distribution. The indicators for the model and the algorithm were generated based on the metric in equal position in the selected metric set. E.g. X_1 has a correspondingly positioned model 1 and algorithm 1 in other sets. We could assume that X_1 is the fixation count metric for the `eye` model with floating labels. If the row samples for the model indicator, the algorithm indicator and the observed metric belong to the same population, meaning that their distribution should be similar enough to combine them across different models or methods and afterwards we could conduct Kruskal-Wallis H-test on samples combining all models instead of per model.

them all in detail would be long and time consuming. Therefore, we focus on the metrics that show significant differences in values and with p-values rejecting the same population hypothesis. The first such metric was the task completion time (TCT), which was used only in the ABC tasks, since it measures the time of finding all the three labels again. It was measured from the pressing of the ESC key and the detection of the triangle forming from fixation behavior. We used the lower of the two values as the end measurement. It had p-values low enough for rejecting the null hypothesis of similar populations in the following three models:

f1 had a TCT for CHL that was 1s faster than for FL, but 2s slower than for HL.

tractor difference in TCT for CHL was increased to 2s faster than for FL while reducing the lag behind HL to 1s. At the same time, it also produced a 500 ms smaller standard deviation of 2.8s when compared to both alternative algorithms.

pocketwatch produced a situation, in which CHL produced better results than both of the alternative algorithms. It beat FL by 5s and HLby 200ms.

The TCT results indicate the time needed to get familiar with the label placement after the simulated user input stops. A smaller amount of time indicates a layout of labels that is easier to understand. By following this logic, CHL outperforms FL in all cases while it performs worse in models with a low number of parts and similar or slightly better in models with large amounts of meshes when compared to HL. A similar time related metric is the first fixation time (FFT), which indicates when the first fixation formed. Lower values indicate a possibly faster detection due to less time used for searching. FFT also produced low enough p-values only in ABC task, where we can observe the differences between four models as boxplots in Figure 5.12, which we can read as:

eye produced 200ms faster FFT for CHL than for FL, while 70ms slower than HL.

f1 created a FFT with CHL smaller than both alternative algorithms with 100ms standard deviation, which is 170-400ms less then in other algorithms. The difference in medians was 2.4s for FL and 100ms for HL.

tractor produced a result lower for 4s when comparing the medians of CHL and FL, while it was 50ms higher when compared with HL.

pocketwatch has a similar result of CHL being 4s faster than FL, while being the same as HL.

A metric having significant differences across both tasks was the average label gaze distance (ALGD), which was defined as an average of the Euclidean distances between the label centers and the gaze point in pixels.

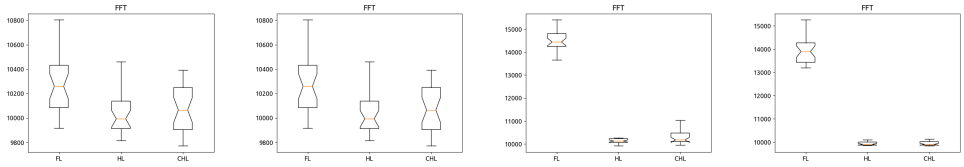


Figure 5.12: From left to right we have boxplots of the FFT metric for models `eye`, `f1`,`tractor` and `pocketwatch`. We see that CHL is always better than FL, while it has similar time values with HL with the difference only being up to 140ms but mostly around 50 ms. We consider 100ms a relatively small difference since it is bellow fixation's minimal duration of 200ms, which was determined by Sharafi et al. [20].

eye in the ABC task it produced 6-7 px longer distance than the alternative methods for CHL, while in the tracking taks it produced a distance of 10 more than FL and only an insignicant 1 px difference from HL.

f1 was declared by p-value important only for the ABC task, where it produced 20-40px more distance than the alternative methods.

gpu produced a distance that is 20 px shorter than FL but 80 px longer then HL in the ABC task for CHL. But in the Tracking task it produced 25-30px less distance than the alternative algorithms.

tractor had a significance by p-values indicated only in the ABC task, where it produced a 10-30 px increase for CHL when compared to the alternative methods.

pocketwach produced a 10 px larger distance for CHL than FL in the ABC task, while it had 10 px less distance than HL. In regard to the Tracking task it produced 20 pixels less distance for CHL than both alternative methods.

The varying behavior of the ALGD in different tasks is logical due to the difference in the nature of the tasks. In the Tracking task, the test subject

just had to observe the same label from the task start until its completion, which would produce smaller distances in case of an algorithm preventing huge label jumps from occurring too often. Task ABC, on the other hand, was a static stimuli after stopping the animation, where the test subject had to find all 3 labels as fast as possible. In this case, the ALGD was created by averaging the individual label ALGD, which may be the reason for the strange behavior since the metric was designed for the Tracking task.

By combining the results of the three metrics we see that our proposed algorithm outperformed the floating label algorithm time-wise in task ABC across models. When compared with hedgehog labeling, it produced worse or similar results for less complex models, while improving the performance on highly complex ones. This indicates an improvement regarding label layout intuitiveness for complex models, which was one of the goals of this thesis. The other goal was the reduction of label jumping, which can be detected in the ALGD metric during tracking. Both metric's behavior can be observed the boxplots inside Figure 5.13.

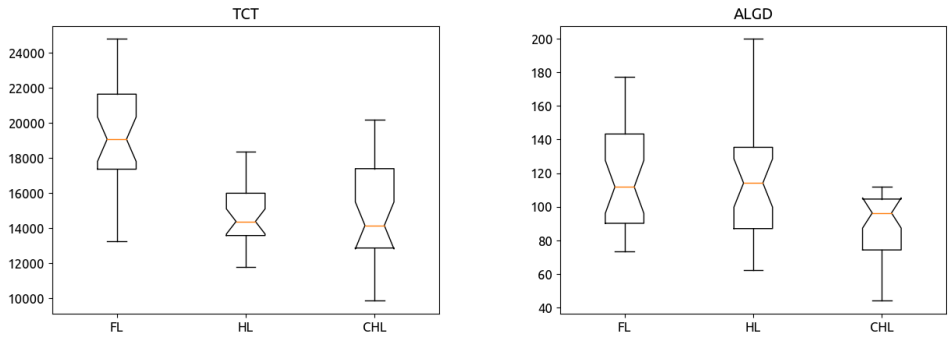


Figure 5.13: The boxplots of seemingly significant differences when comparing the task completion time (TCT) and the average label gaze distance (ALGD) for the `pocketwatch` model.

Besides the discussed three metrics and those removed from the test due to being too low in sample size, which was only required to be five samples per metric-algorithm pair in a task, we also got a pointer for statistically

significant differences in 27 other metrics: AFD, AFDR, AFDT, AS, ASA, ASL, ASR, FAS, FAS, FC, FDR, FDR, FR, FSC, FSC, FSD, FSD, FTD, FTR, FVR, GSD, SC, SDS, SL, SS, STDFD and STDSA. However, most of these metrics were determined to only have a statistically significant differences in one or two metric-algorithm pairs, which would limit us to a more model focused discussion if we went into details in this section. However, we also had exceptions such as AFDT and FVR, which were present across multiple models and tasks. The problem of those metrics was that their differences were relatively low ones of 100ms less dwell time and 3-8% less fixation rate per label, which in the end mostly corresponds to the difference of fixation counts (FC), which was higher in CHL than other algorithms by an amount of 2-4 fixations more. However, the FC metric itself was regarded as statistically significant only in two cases, even making the AFDT and FVR results questionable. For those interested in observing the differences in the disregarded metrics, please see the tables inside Appendix E.

Since we only used CHL inside the interactive task of Exploratory search, we could not really study the general metrics as in other tasks. Therefore, we focused on finding indications of how the participant's progress went inside the task. While we could not use the blink metric in previous tasks due to a low amount of samples, or in other words, almost no blinking present, we still kept the measurements. By comparing the ranges of average blink metrics across the models for each task on CHL's method groups, we could detect the participant looking away from the screen since the blink count and durations are measured from invalid gaze measurements. In Table 5.7, we see that the ABC and the Tracking task had model averages with no blinking present, which naturally means that no blinks happened for the model in either task during a CHL method group run in case of some participants, since we cannot have negative counts. On the other hand, Exploratory search almost certainly produced a blink per model, which we actually translate into moments when the participant moved their gaze to the keyboard. The maximum ranges may have increased, but not that much compared to Track-

Metrics	BC_{min}	BC_{max}	ABD_{min}	ABD_{max}	$STDBD_{min}$	$STDBD_{max}$
ABC	0	1.42	0ms	187ms	0ms	41ms
Tracking	0	6	0ms	406ms	0ms	137ms
Search	1.42	2.27	141ms	234ms	6ms	48ms

Table 5.7: The blink metric comparison of minimum and maximum values for CHL averages across models. The changes in the Exploratory search task indicate a rise of invalid gaze measurements, which also happens when the subject is looking away from the screen.

ing. But the almost certain look away from the screen to the keyboard and mouse for confirming the interaction layout confirmed the participant's observations that the used keyboard layout was not good, which in term took their valuable time during the neighborhood search. This proves that initial experiences with applications are important as mentioned by Pagulayan et al. [48], even more so during time-limited experiments.

While the keyboard and mouse layout of interactions was far from perfect, it did not hinder the participants from gathering any results. Since the task was an open question type where no wrong answers were possible, we can only check their finish times between the metrics and not their success rate. While we wished to compare task completion times (TCT), we noticed that almost all of the participants finished the individual subtask by using up all of their time either due to the high complexity of the model or due to taking their time to confirm their decisions from different angles and with the help of explosions, not to mention the problems of the interaction keyboard layout. Thus it resulted in the TCT comparison to be useless.

5.8.2 Visual Analytics Observations

A useful way of handling the eye-tracking data is also to use visual analytics such as heatmaps, where some even plot averages across participants and

produce informative plots. However, as Kurzhals et al. [15] mentioned, the dynamic stimuli such as the ones in the Tracking task, where during tracking there is no stop, result in no information. In our case only the starting point of the label was recorded as a fixation since it took the participant a while to notice the first jump of the label. Luckily for us, both the ABC and the Exploratory tasks had enough static components. Actually, the whole ABC task became static during the period of 10s meant for searching, which is also the only section of the data that we compare for the ABC task across metrics. The Exploratory search task was executed by the participants first finding the search term and determining a desirable camera pose with whole model in view including with the searched term model part. This was then followed with almost no camera interactions and only explosion interactions on their own intervals, creating an almost static setting to observe.

For the Exploratory search task, we prepared view presentations per model of where most participants were positioned at least once during their neighborhood search phase, which can be observed in Figure 5.14, where the produced heatmaps for a random selection of participants were used. Since the interactive task gave too much freedom to the test subjects, we decided to prepare randomly selected plots to indicate the similarity of the results across models and participants, meaning that the plots were not chosen from a group of the best ones although it may seem so. Since the representations do not take into account the individual differences between camera interactions, the labels do not necessarily fall into the same place as is suggested by the representative image. Therefore, we first plotted the searched term label's bounding boxes across time as colored squares according to the dimensions of the label at the moment of given position as well as the position of the anchor point as an X of the same color in the same moment. We then plotted the actual heatmap on top of the marked representative image. In all of the heatmap plots, including those not present in Figure 5.14, we can observe the participants focusing on the search term labels since the searched term label represented their reference point in space as well as on the model

parts around it, which is what we expected from an exploratory search. Interestingly, there are cases such as the `f1` model, where focused on model parts lie far from the goal, however this co-aligns with the observations that were written down according to the participant's vocal answers. In the case of `f1` model particularly, many participants at first misinterpreted which part represents the search term of the model, which was the console hidden inside of the main body, but from the side it looked as if the label was marking the main body. This can be easily attributed to the fact that most participants ignored the invisibility information coded as label and leader line transparency due to too much initial information during the learning phase. Regarding the fixations in the empty space of the representative image, notice that they are not far from the labels and the model, which indicates that there was potentially a label or even a part positioned there after an explosion was triggered to observe the inner structure of more complex models. Therefore, we conclude that our system suited the exploratory task well while the interaction control layout needs improvements.

We also wished to prepare representative images for the ABC task plots, however we were too late at noticing that a mistake had occurred in the code regarding the user input simulation. While all of the same model subtasks did execute the same sequence of instructions in the allocated time, we did not take into account the differences caused by the CPU allocating process times and only the passed time as an indicator of progress instead of the number of passed instructions. This meant that the instructions were used during their predefined time slots or key-frames, however the count differed between participants and even among method groups. In practice this meant that we had experiments not on identical stimuli as stimuli such as videos, but only on similar stimuli. This is also most likely the reason for most of distributions not being Gaussian across even same pair subtasks, during the metric analysis, which was already taken into account during the analysis by the use of non-parametrized tests and medians instead of averages in Subsection 5.8.1. Since we only had limited information available for reconstructing

the whole scenes per stored record inside the already big original XML, we did not decide to go into scene reconstruction, which would also likely overload the thesis time constraints. Therefore, we did not use a representative image for the heatmaps but only empty space with the marked label and anchor point positions instead. With Figures 5.15 and 5.16 we compare the heatmaps across method groups and Latin square groups on the `eye` and `pocketwatch` models to confirm the previously discovered differences in the performance of CHL between the complex and the relatively simple models. Inside the plots, we compare the amount of fixation away from the label AOIs as well how strong the fixation was, which goes from blue to red according to the growth of the fixation strength. The plots show situations aligned with previous observations, since we can observe that CHL behaves similar to HL in the case of simple models. We can also observe better performance in regards to FL in general, while with complex objects it also performs better than HL.

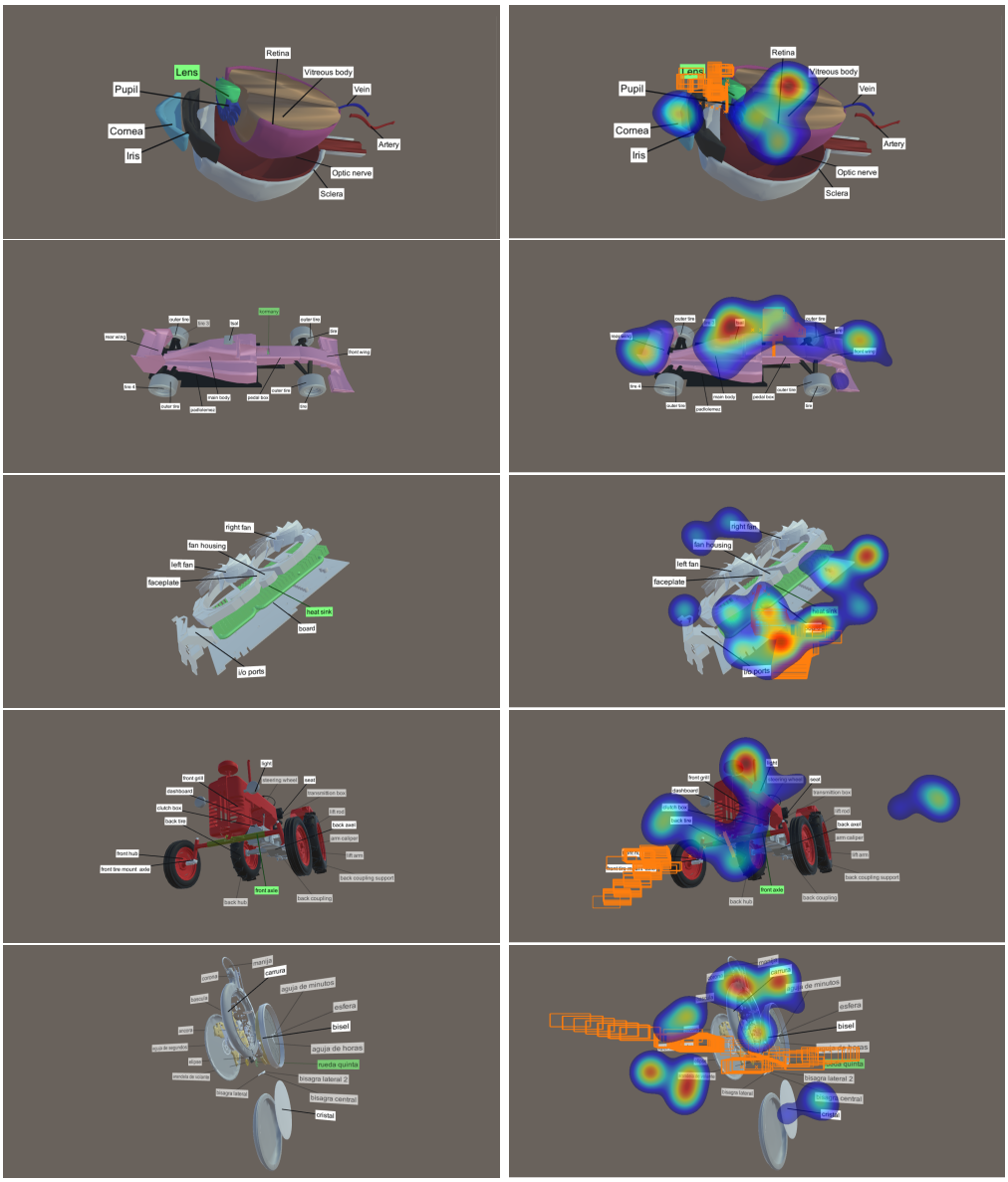


Figure 5.14: On the left side we have representations of what a participant may have seen during the Exploratory Search task. The representation was created based on the observed general participant behavior, where they at some point arrived at the camera pose presented in the representation, however depending on the participant's previous interactions the label layouts were most likely different. On the right side we can observe the heatmaps with label bounding box and anchor point indicators for the label containing the search term of randomly selected participants. While the background image of the heatmap is just a representation with mistrustful label positions, we can still observe the participant focusing on the search term, its referred part as well as on the near neighborhood or their labels.

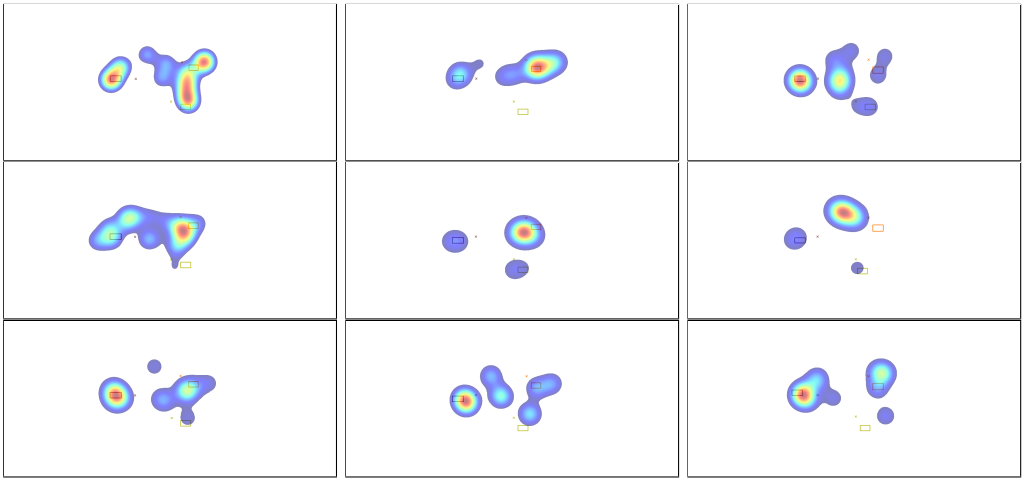


Figure 5.15: The ABC task's `eye` model heatmaps from three selected participants of different Latin square groups. Each row is one participant's result in the following order: floating labels (FL), hedgehog labeling (HL) and clustered hedgehog labeling (CHL). The rows are arranged in LAtin square groups according to the A, B, C order. Group A started with FL and finished with CHL, group B started with HL and finished with FL, while group C started with CHL and finished with HL. Notice the similarity between CHL and HL results when compared to FL.

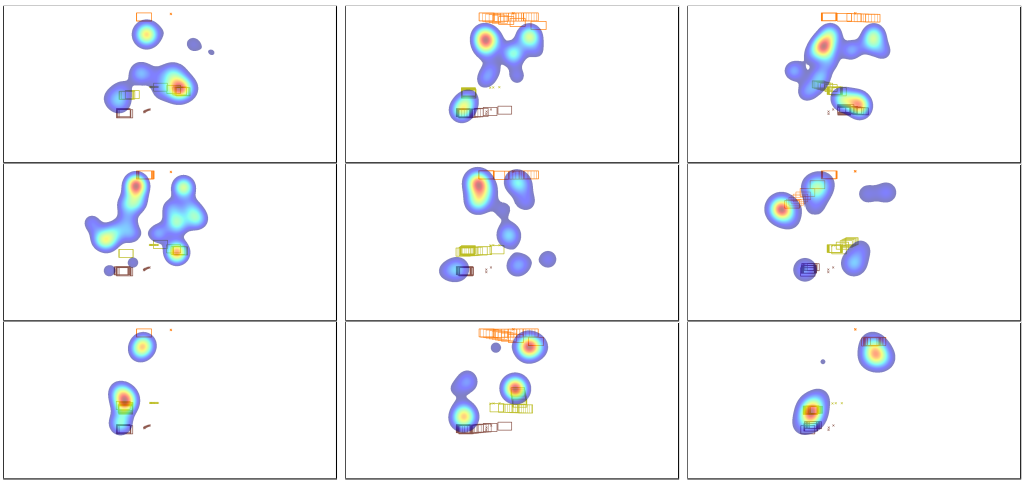


Figure 5.16: The ABC task's `pocketwatch` model heatmaps from three selected participants of different Latin square groups. Each row is one participant's result in the following order: floating labels (FL), hedgehog labeling (HL) and clustered hedgehog labeling (CHL). The rows are arranged in LAtin square groups according to the A, B, C order. Group A started with FL and finished with CHL, group B started with HL and finished with FL, while group C started with CHL and finished with HL. Notice the amount of concentrations around the label bounding boxes in CHL, which also produced heatmaps with less general area fixations.

Chapter 6

Conclusion

The thesis proposed, implemented and evaluated the implementation of the clustered version of the existing state of the art 3D label placement algorithm Hedgehog labeling, proposed by Tatzgern et al. [8]. We extended his concept from targeting only static and non-deforming objects to also include dynamic and deformable objects, where we chose 3D exploded views as representative of both deformable and dynamic 3D models.

We used the calculation system provided by Kerbl et al. [7] as a back-end system for the automated production of 3D exploded view models from arbitrary CAD models, where labels can also easily be produced from the part names from inside the CAD model. The proposed clustered hedgehog labeling algorithm was implemented as Textplosion together with the support added for the back-end system support provided. The development of Textplosion resulted in an innovative real-time interactive 3D model presentation application. The application allows interactive annotated presentations of the outer and the inner structures of complex arbitrary CAD models, which is yet to be seen in commercial application in regards of automatization level. The level automatization is only limited with manual labour of selecting a reasonable subset of Textplosion generated labels or modifying them and also by the preparation of arbitrary 3D models to be accepted as input of the back-end system, which from our perspective is a black box

due to separate development and security issues. While the number of HCI was limited to two interactions for controlling the camera pose and four for the explosion simulation, we have also shown results in unlimited varieties of exploded views by stacking the interactions on top of one another.

The difference in using our proposed solution or the alternative floating labels or hedgehog labeling was already noticed in the visual comparison of the results. It was visible enough to be even noticeable in a visual comparison of the static images while it is even more apparent in live action or in the recordings. To scientifically prove that our solution works better we also went further and prepared and executed a new kind of study for the area of 3D labeling algorithms, where user based studies enhanced with eye-tracking are a novelty. This made us pioneer the area based on practices from other areas.

In general, the subjective results of the experiments showed that the results of clustered hedgehog labeling are at least as equally good as the results of original hedgehog labeling and better than floating labels. On the other hand, there was not much statistical support for pointing out the improvements from the normal hedgehog labeling. While no statistical support was found, we cannot just deem there being no improvement observable to the user on a subjective level since the results could have also turned out like this due to a poor clustering metric set selection, the wrong K-means size limitation or even by the user interaction simulation being too fast, which was noted in the audio recordings. Many of the participants felt that the system performed better during the interactive task since they could control the speed of each interaction, while during observation tasks the speed was automated according to the average focus of between 15 and 20 minutes. During the interactive task it was noted that even the participants of our experiments were fazed by the amount of possible interactions produced by stacking only four basic explosion interactions. They even reported a lack of time to try everything out to a satisfying degree.

The objective eye-tracking results further confirm that clustered hedgehog

produces results of similar quality as hedgehog in less complex models in regards to search tasks depending on the intuitiveness of the produced label layout. However, it shows its real potential in models of higher complexity due to the amount of exploded parts, where it outperforms the hedgehog labeling. The objective results also confirm the significant improvement in the Tracking task when compared to the floating labels algorithm. It goes further by showing a similar amount of improvement in comparison to the hedgehog labeling algorithm, which could not be proven in subjective results.

With a visual analysis of the plotted heatmaps, we confirmed the results of the statistical analysis of objective metrics while we also looked at the interactive task, which served as a usability test in its true meaning for our implementation. While we confirmed the application to be well suited for exploratory tasks on 3D exploded views, we also confirmed within its metric ranges that the initial experience during the learning phase was strained due to too much information being introduced in too little time.

Combining all of the study results together shows that our proposed algorithm produced label layouts outperforming all of the floating label's produced ones time-wise in the ABC task regardless of model selection, while when compared with hedgehog labeling the layouts produced similar times for less complex models and shorter times for highly complex ones. This indicates an improvement regarding label layout intuitiveness for complex models, which was one of the goals of the thesis. The other goal was the reduction of label jumping, which was also detected by the ALGD metric during inside the Tracking task results. Sadly, we did not have the time to define or use a previously proposed dynamic stimuli visual analytics method, such as the space-time cube proposed by Kurzhals and Weiskopf [32], to further confirm the reduction of label jumping by using visual analytics. Visual analytics can be a resourceful way of analyzing the eye-tracking data gathered from interactive applications, as we have shown in the case of the heatmaps of the semi static Exploratory search task. It was also effective in determining the possible origin of the non-normal distribution of the observation

task results across metrics. However, further research and carefully prepared studies are needed to find ways of analyzing the fully dynamic stimuli in an efficient and simple way.

While our study showed some positive indications of improvement of our proposed algorithm, it was also full of learning experiences, which could be useful for future studies on Textplosion as well as for other researchers. An obviously good practice for the future would be to separate the observation and interaction usability tasks either among participants or by a longer pause between the task changes or even as separate studies, since it requires some effort to switch from observing to interacting. Our study actually barely fit the allowed time range for focus time limitations of 15 to 20 minutes, not to mention that the whole procedure for a single participant took a whole hour.

By adding the poorly designed interaction and input device mapping to the tight time constraints, we observed that the participants were at a loss in the start. Inside the invalid gaze measurements, we confirmed a rise of participants looking away from the display in order to reconfirm the keyboard keys outside the demonstration task, as well as forgetting the meaning of the transparent labels, which delayed their progress. Since we were developing the implementation on the set keyboard layout for a long time, we got too used to it to notice the hidden problems that effected participants as new users. Therefore, another good lesson for future studies is to construct pilot tests with the initial application experience in mind even if it requires additional steps and pilot participants. By including it inside the experiment design requirements, it can potentially remove unexpected difficulties, which appear due to poor thought processing, an example of this being our key and mouse click mapping of the interactions. This bad mapping of the keys and mouse was also one of the few negative remarks made during the open interviews.

We also learned that another important part of the preparations for the interactive application stimuli is being sure of the consistency of stimuli between the participants. This is a problem we can only solve by ourselves

and not inside the pilot tests. Since only we know which stimuli need to be identical, naturally if we do not notice the mistake, the pilot participants will also fail to notice it. In our case, it was a problem caused due to relying on the time passage instead of the CPU instruction count, which fairly said is a natural mistake for novices in building studies on a level of interacted applications measured by eye-tracking. Our proposed solution for future studies is to either measure both the instruction count and the elapsed time or to simply prepare recordings of the stimuli and present those. While the first option requires more input at the beginning, it also offers a way to store all the gathered data already in sync as shown by us reconstructing some of the faulty information from the huge amount of data we stored as our fail-safe. Recordings, on the other hand, are simple to prepare and can easily be restarted in case of failure, however the post-processing requires additional work with synchronizing the data as well not providing a fail safe option for faulty stored information.

Our final contribution inthe thesis is the creation of an exploded 3D model dataset, which we plan to share with the community to potentially become the starting block for an actual standardized dataset since none yet exists. While it was designed with exploded views and similar dynamic and model-deforming 3D model presentation techniques, it can also serve as a test bed for the comparison of 3D labeling algorithms for static models or exploded view interaction research.

Appendix A

Textplosion Input Formats

Section 4.2 mentions that indirect access was given to produce back-end OpenInvertor meta-data files for describing generated exploded views via the system provided by Kerbl et al. [7]. Therefore, Textplosion's internal data hierarchy described in Subsection 4.2.1 was modeled after these meta-data files. In order to achieve greater flexibility, we also developed our own JSON format for keeping exploded view information. Both of these meta-data file formats are described in this appendix.

A.1 Backend Provided Input Files

A.1.1 Assembly File Type (.ass)

Specifies how model parts can be loaded and which specification files belong to them. It represents the starting point of meta-data loading. The first 6 lines represent header information in the following order:

1. Shebang indicating the file: `Assembly file`
2. File type tells format of geometry data: `iv, obj, ...`
3. Unique part count, which tells the number of unique geometry loads

4. Unknown count - the information was not disclosed, but was also not critical for us.

5. Count of movable parts without blockers

6. Movable & blocking part count tells the number of parts blocked in their movement.

Following the header we have the content in a 3 line pattern, where third line pattern is repeated based on the count of first:

1. Specification file count + part name: `1:  press1__default_`

2. Path to geometry file: `iv, obj, ...`

3. Copies count + path to main specification file:
   `.1 ../testcases/prezz/filespecs/press1__default_0.spec`

The filename for the main specification file does not only point towards the main specification file type (.spec), but also to its derivatives (.dspec, .fspec, .xspec, .xspec_a), which all share the filename excluding the extension. ×

A.1.2 Specification File Type (.spec)

The main file that includes details regarding the loading geometry is the specification file type. It defines an integer ID of the part in the system and its transformation matrix. It has 17 values per part copy, in which the first value of the first line is the Part ID, and the remaining 16 values form a 4x4 Transformation Matrix in rows:

```
17
-1 0 6.98297e-015 34.9389
0 1 0 103.53
-6.98297e-015 0 -1 -8.15629
0 0 0 1
```

While a single geometry may have multiple copies, not all are part of the same movement/specification group. Only the copies inside the same specification file belong to the same group. Therefore, a geometry can actually be used by multiple specification groups as observed in the assembly file type, where we actually retrieved the amount of specification files per geometry.

A.1.3 Direction Specification File Type (.dspec)

This derivative file type specifies the actual movement directions for the movement/specification group of geometry copies. It also specifies the blocking and non-blocking copies in the given direction. The directions can be either single-directional or cascading ones. File types of .xspec and .xspec_a are its binary version. The actual structure of the type is:

1. Unknown is a number or ID: `0`
2. Single-directional direction count: `6`
3. D Line specifies the 3D vector of singular direction:
   `D: 0.000000 0.530150 -0.847904`
4. The blocking boolean series is a series of 0s and 1s, and specifies which copy potentially blocks the current part in the previous line direction. The 1s in the string mean a potential collision with the part, whose id corresponds to the string index: `00100101100000000000000`
5. Cascading direction count: `23`
6. Line X specifies the intial pair 3D vector of singular direction:
   `X: -1.000000 0.000000 -0.000000 - -0.000000 -0.530150 0.847904`
7. The last cascading direction vector: `-1.000000 0.000000 -0.000000`
8. Blocking boolean series: `00100101100000000000000`
9. Empty line between copy specifications.

Line pattern 3-4 repeats itself for the count defined in line pattern 2, while line pattern 6-8 repeats itself for the count defined in line pattern 5.

A.1.4 Connection File Type (.csv)

It tells us which of the parts are touching each other in a form of a 0/1 matrix, where row and columns correspond to the received copy ids.

A.1.5 Label Specification File Type (.lspec)

This added format links label information to the exploded view model information. It was self-defined following existing format patterns:

1. Shebang indicating the file: `# Beta label format file`
2. Label count: `13`
3. Copy id: `2`
4. Label offset: `0.000000 0.000000 0.000000`
5. Label text:
   `This is a label text example`

A.2 JSON Input Files

Since we will work with multiple JSON files, we have defined a JSON container structure for read function re-usability. Each of the different JSON formats is meant to be placed inside the data array for each copy. It counts towards the same amount as in the count attribute, which works as a check for corrupted files.

```
{    // file description
    "id" : "id/name of label file",
    // for double checking the user input
    "count" : 0,
    // array of label or other JSON objects
    "data" : [ ... ]
}
```

A.2.1 Model JSON Format

This format is a JSON Object that describes geometry, which correspond to the back-end's lines in assembly and specifications files. The part_id of the model JSON is used as an ID when grouping model parts, labels and movement.

```
{    // object name in Unity and movement group id
    "part_id" : "model part id",
    // path starts in same folder as JSON file
    "source" : "path to OBJ file",
    // model position
    "position" : { "x": 0, "y": 0, "z": 0 }
    // the Euler angle rotation
    "rotation" : { "x": 0, "y": 0, "z": 0 }
}
```

A.2.2 Explosion JSON Format

A JSON Object describing a singular direction but without blockers defined. Since labels and parts of same group should move in same direction, the part_id is also used as the id/description of the movement JSON. The only

additional information needed is the actual movement described by its direction.

```
{    // the id of model/movement group
    "part_id" : "model part id",
    // direction components
    "direction" : { "x": 0, "y": 1, "z": 0 }
}
```

A.2.3 Label JSON Format

This format is a JSON Object that describes labels with part_id, initial offset and the actual label text.

```
{
    // the id of model/movement group
    "part_id" : "model part id",
    // initial relative offset from model center
    "position" : { "x": 0, "y": 0, "z": 0 }
    // the displayed text in the label
    "text" : "Label string of text"
}
```

Appendix B

Experiment Paperwork

For evaluating our solution with the user based study, we had to prepare several documents. Since the experiments where run in both the Slovene and English speaking environments, both versions were prepared. We present the produced documents with short descriptions and afterwards with actual documents following the same order as the descriptions.

Topic Introduction

The topic introduction used images for an easier explanation of the area of our topic and experiment goals. Due to unclarity of true sources of the images, we only present this document with a description.

As an icebreaker, we started the experiment by presenting a 2D example of labeling the human body. This is something that most people should have already seen in the course of their compulsory education. Using this example, we also explained Schmalstieg and Hollerer's six label placement objectives [3] that we wish to fulfill automatically and how many are broken in the given example. As an example of a real-time application with good results, hedgehog labeling is presented in an AR application. We also explained that it was developed at the Graz University of Technology, which also developed an exploded view generator as shown in the following image. Using the exploded view images, we describe the benefits of using explosions

as a way to present 3D models. Presenting the benefits of both produced systems, we continued to explain the natural thought of combining them, as seen in the last pair of images. By showing only a simple integration of both concepts in the images, we presented its issues and challenges for improvement that became a topic of a whole master thesis.

Data Collection Agreement

Since user based studies require us to collect personal information, collection of which is considered a breach of privacy, we also prepared agreement forms for collecting personal information and opinions as well as for recording video and audio sessions. All participants were required to fill out and sign the form or otherwise be removed from the study. By signing the forms, the participants also agreed to transferring their rights of ownership on all collected data.

Study Specific Questionnaires

The following document combines multiple questionnaires on the same page to reduce the amount of overall paperwork, which would also effect the test subject's performance. The first part is a short personal data collection questionnaire that also asks about prior experience with 2D screen and 3D world HCI, in which we even considered 3D games as experience. The middle questionnaire was used after an individual task was finished to set ranks as how the test subject observed different methods. The final questionnaire was used after all tasks had been completed. It provides extra feedback information regarding the system we built.

SEQ & NASA TLX Questionnaire

After each method group of subtasks inside of a task finishes, the test subject is asked to fill out SEQ & NASA TLX questionnaires, which measure the perceived difficulty of the current task run. To reduce the paper amount

and stress due to it, both questionnaires are united on the same page. Both questionnaires are well established in regards of retrieving user experience from the perceived difficulty. While SEQ asks only about general difficulty on a scale from 1 to 7 and always needs to be answered first, NASA TLX on the other hand consists of divided difficulty perceptions into 6 different scales from 0 to 100 points [45, 47]. Due to the six well defined scales in NASA TLX defined for English, we used a standardized Slovene version, which was selected from a variety of translations to match the results to the English equivalent.

Task Instructions

We also present the instructions for each task prepared for the test subject to read besides hearing our vocal explanation of the task. It was a way for them to reconfirm the explained instructions or application controls.

B.1 Data Collection Agreement

Evaluation of Algorithms for Label Placement inside 3D Exploded Views

Personal data and experiment video/audio data collection agreement

By signing this agreement I agree to have my eyes recorded by an eye-tracking device for the whole duration of the evaluation of algorithms for label placement inside 3D exploded views. With my signature I also agree to have my voice recorded during the evaluation for purposes of better post analysis of the gathered data. At the same time the signature is also used as an agreement to collect my personal data, opinions and computer measurements taken by the experiment programs.

Finally the signature also transfers all the rights of ownership of any collected data during the evaluation to both the Institute of Computer Graphics and Vision from the Graz University of Technology and the Laboratory for Computer Graphics and Multimedia from the University of Ljubljana Faculty of Computer and Information Science.

Full name: _____ Place & date: _____

Signature: _____

Evalvacija algoritmov za postavitev oznak v razstavljenih 3D modelih

Soglasje o zajemu osebnih podatkov in video/audio vsebine testiranja

Spodaj podpisani soglašam s tem, da se celotna evalvacija algoritmov za postavitev oznak v razstavljenih 3D modelih snema z napravo za sledenje vida. Soglašam tudi z zajemom zvoka tekom raziskave za namene izboljšave kasnejše analize podatkov, kakor tudi s pisno predanimi osebnimi podatki in mnenji ter z meritvami zajetimi s strani računalniškega eksperimentalnega programa.

S podpisom soglašam tudi s tem, da so posnetki, meritve in zapisi nastali tekom raziskave last Inštituta za računalniško grafiko in vid, Tehnične univerze Gradec in Laboratorija za računalniško grafiko in multimedije, Fakultete za računalništvo in informatiko, Univerze v Ljubljani.

Ime in priimek: _____ Kraj, datum: _____

Podpis: _____

B.2 Study Specific Questionnaires

Initial Personal Data Collection

Age: _____ Gender: _____

Experience with 3D graphics applications (Blender, CAD, Maya, MeshLab, …) :

Midway Ranking Questionnaire

Task	ABC			Tracking			Exploratory Search
Identification Number							
Method Success Rank	1.	2.	3.	1.	2.	3.	

Final Questionnaire

(Circle your chosen answers, a multiple answer choice needs also an importance rank)

Overall, how difficult or easy did you find this experiment (all tasks together)?

1	2	3	4	5	6	7
Very Easy						Very Difficult

The most difficult task was **ABC** **Tracking** **Exploratory Search**

The easiest task was **ABC** **Tracking** **Exploratory Search**

The label layouts were stable (when it changed, I knew intuitively, where a label was):

Always for each method **With some models** **With some methods**

Most of the time **Never, it was always randomly positioned**

How different were the label layouts between methods inside a task (for same model)?

Very different **Maybe 1 method differed** **Similar** **No difference**

Did you understand the meaning of label transparency? **Yes** **No** **Didn't notice**

Did label transparency help or distract you in your tasks?

Was helpful **Was distracting** **Caused no difference / I ignored it**

Začetni zajem osebnih podatkov

Starost: _____ Spol: _____

Dosedanje izkušnje z aplikacijami za 3D grafiko (Blender, CAD, Maya, MeshLab, …) :

Vmesni vprašalnik za po zaključku posamezne naloge

Naloga	ABC			Sledenje			Raziskovanje
Identifikator							
Rank uspešnosti metod	1.	2.	3.	1.	2.	3.	

Zaključni vprašalnik

(Zgolj obkroži ponujene vrednosti, po vprašalniku sledi pogovor)

Zahtevnost celotnega eksperimenta (vseh nalog)

1	2	3	4	5	6	7
Zelo lahko						Zelo težko

Najbolj zahtevna naloga je bila **ABC** **Sledenje** **Raziskovanje**

Najbolj enostavna metoda je bila **ABC** **Sledenje** **Raziskovanje**

So bile razporeditve oznak stabilne (so se spreminjale, vendar sem vedel, kaj je kje):

Vselej vsako metodo **Pri posameznih modelih** **Pri določenih metodah**

Večino časa **Ne, vselej naključna postavitev**

So se razporeditve oznak razlikovale med metodami znotraj posamezne naloge?

Zelo različne **Morda ima 1 metoda drugačno** **Bile podobne** **Ni bilo razlike**

Ste razumeli prosojnost oznak? **Razumel** **Nisem razumel** **Nisem opazil**

Je prosojnost pomagala ali škodila? **Pomagala** **Škodila** **Ni razlike / sem ignoriral**

B.3 SEQ & NASA TLX Questionnaire

Fill out the identification number and the questions!

ABC Task Method A

Identification number: _____

Overall, how difficult or easy did you find this task?

1	2	3	4	5	6	7
Very Easy						Very Difficult

Mental Demand — How mentally demanding was the task?

Very Low — Very High

Physical Demand — How physically demanding was the pace of the task?

Very Low — Very High

Temporal Demand — How hurried or rushed was the pace of the task?

Very Low — Very High

Performance — How successful were you in accomplishing what you were asked to do?

Perfect — Failure

Effort — How hard did you have to work to accomplish your level of performance?

Very Low — Very High

Frustration — How insecure, discouraged, irritated, stressed and annoyed were you

Very Low — Very High

Izpolni identifikator in lestvice po danem vrstnem redu!

Naloga ABC Metoda A

Identifikator: _____

Splošna zahtevnost

1	2	3	4	5	6	7
Zelo lahko						Zelo težko

Mentalna zahtevnost

Kako mentalno zahtevna je bila naloga?

Zelo nizko — Zelo visoko

Fizična zahtevnost

Kako fizično zahtevna je bila naloga?

Zelo nizko — Zelo visoko

Časovna zahtevnost

Kako zahteven (hiter oz. nagel) je bil tempo naloge ?

Zelo nizko — Zelo visoko

Izvedba

Kako uspešni ste bili pri doseganju tega, kar je naloga od vas zahtevala?

Odličen — Neuspešen

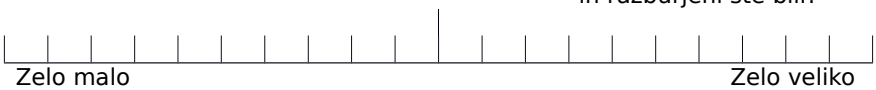
Trud

Kako zelo ste se mogli potruditi, da ste dosegli vaš nivo uspešnosti reševanja naloge?

Zelo malo — Zelo veliko

Frustracija

Kako negotovi, nemotivirani, razdraženi, pod stresom in razburjeni ste bili?

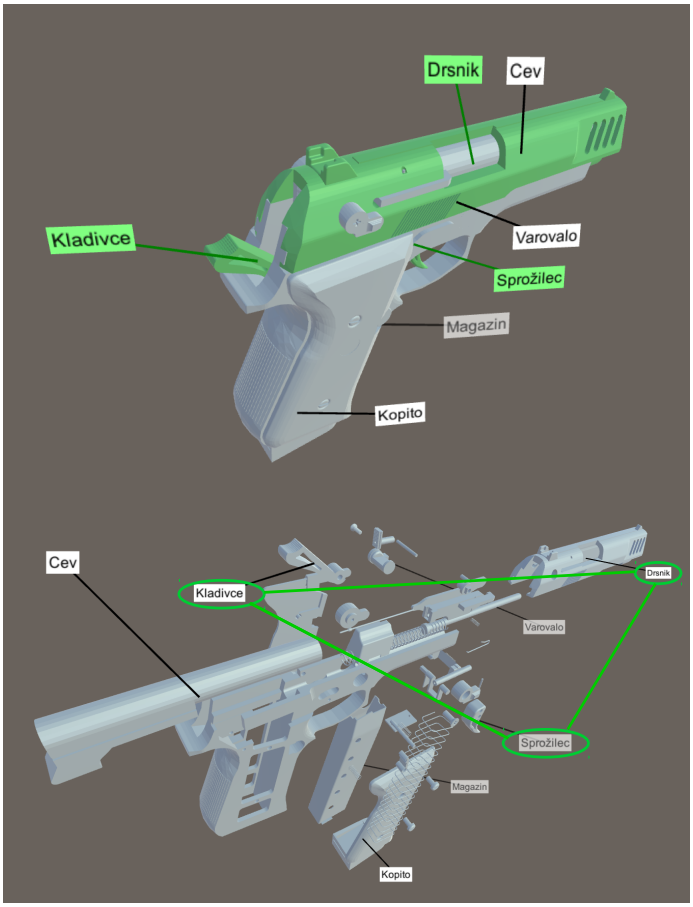


Zelo malo — Zelo veliko

B.4 Task Instructions

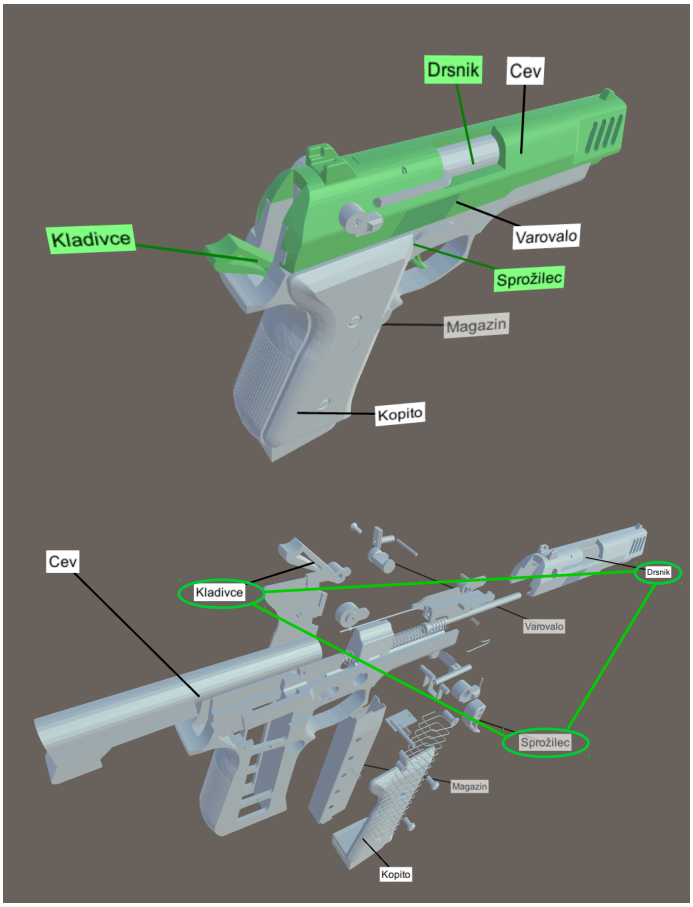
ABC Task Steps

1. In the ABC Task, we test on the intuitiveness of label placement.
2. Prepare your left hand finger on the ESC key, which will need pressing.
3. When you press SPACE to start the the subtask, three green colored labels will glow for 3 seconds.
4. During those 3 seconds, **remember the text of all three labels**.
5. After the glowing stops, an animation with camera rotation, zooming and explosion interactions is run for 5 seconds.
6. After the animation, the program will wait 10 seconds for you to find the three labels again.
7. When you find all three labels, **press ESC and start circling with your eyes between them to form a triangle**. Continue doing this until the black screen appears, but you can release the ESC key.



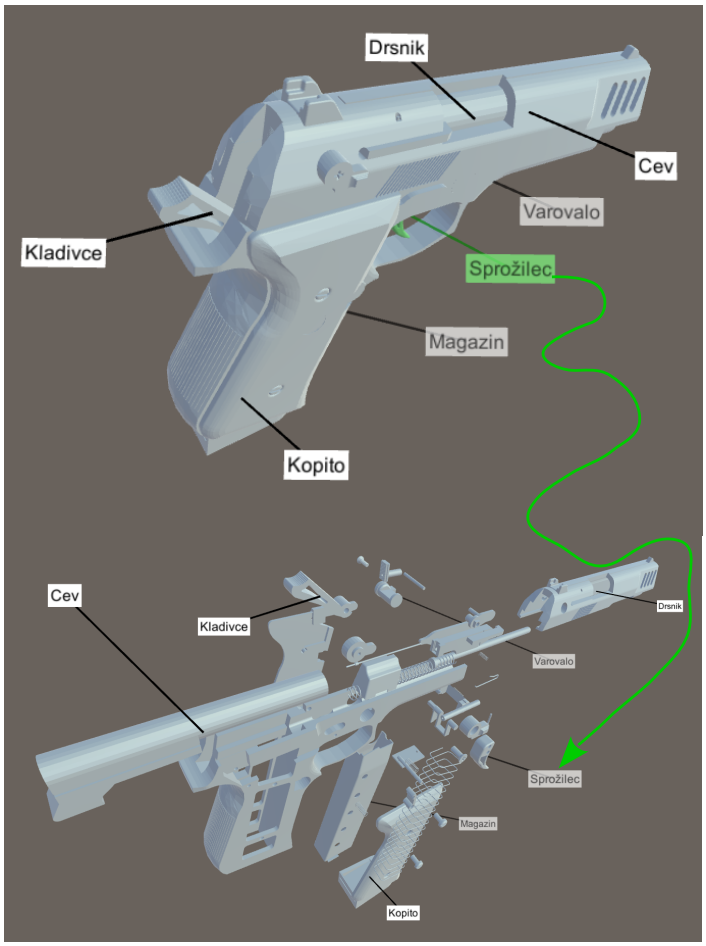
Naloga ABC

1. V nalogi ABC preizkušamo intuitivnost porazdelitve oznak.
2. Pripravi prst leve roke na tipko ESC, ki jo boš moral pritisniti v nadaljevanju.
3. Ko ob začetku pritisneš tipko PRESLEDEK / SPACE za začetek podnaloge, bodo na prikazanem 3D modelu 3 sekunde svetile tri oznake in njim pripadni kosi.
4. **Zapomni si besedilo oznak** v dodeljenih 3 sekundah.
5. Ko kosi in oznake prenehajo svetiti, se prične 5 sekund animacije, kjer se bo pogled vrtel, približal, oddaljil in sam 3D model bo šel narazen oz. eksplodiral.
6. Po 5 sekundah animacije se bo model ustavil na mestu in na voljo imaš 10 sekund, da najdeš zapomnjene oznake.
7. Ko najdeš vse 3 oznake, **pritisni tipko ESC in prični z očmi med njimi skakati, da tvoriš trikotnik**, kot na prikazani sliki. Počni to, dokler ne poteče 10 sekund.



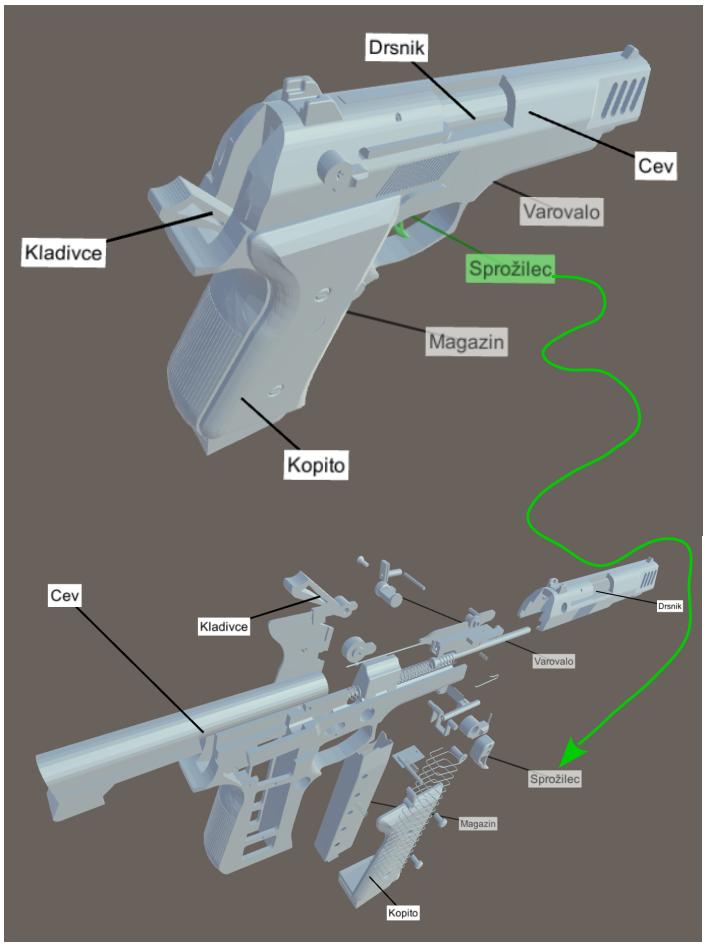
Tracking Task Steps

1. This task focuses on label jumping inside the scene.
2. You only need to press SPACE to start a subtask, but otherwise only use your eyes.
3. A glowing green label will blink for 2 seconds.
4. Find it and focus on it and its corresponding model part pair, which also glows.
5. After the glowing stops, another 14 seconds animation will be run. **During this animation, focus on the label and do not lose it. If you do lose it, find it.**
6. **Focus on the label unitl the next instruction screen appears**



Naloga Sledenje

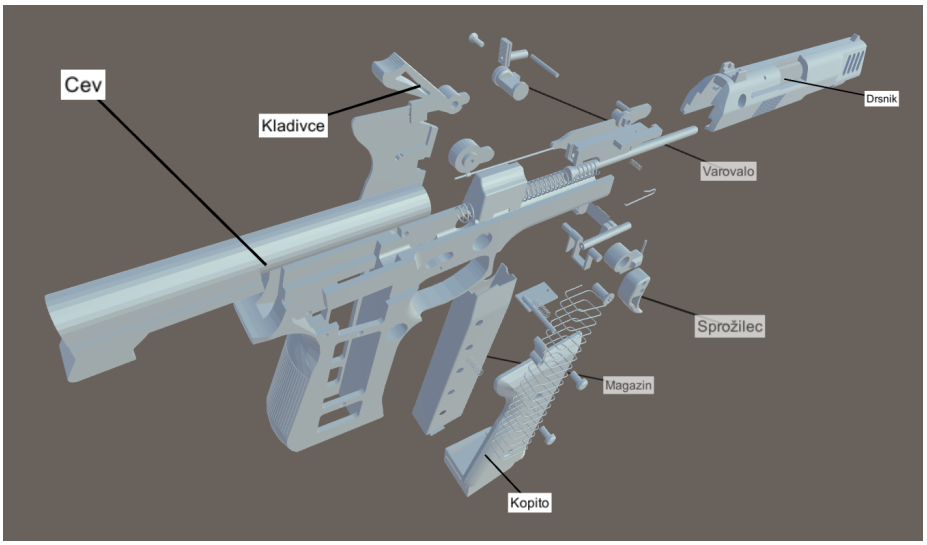
1. Ta naloga preučuje skakanje oznak po prostoru.
2. V tej nalogi potrebuješ uporabiti zgolj svoje oči po začetnem pritisku PRESLEDKA.
3. Na začetku naloge bo ena izmed oznak zeleno utripala 2 sekundi.
4. Najdi utripajočo oznako in kateri kos modela prikazuje.
5. **Do konca naloge z očmi sledi oznaki** in jo poskusi ne izgubiti.
6. Po prenehanju utripanja se bo pričela nova animacija vrtenja in eksplozij, ki bo trajala 14 sekund.
7. **Če med animacijo izgubiš oznako, jo le poišči znova in nadaljuj sledenje.**
8. **Naloga in sledenje se konča ko se pojavi črn oz. siv zaslon navodil.**



Exploratory Search Task Steps

1. In this task, you will interact with the system and not only observe it.
2. We wish to get feedback about the system.
3. For that to happen, we need to introduce you the interactions you can access via mouse and keyboard. **First, remember all controls and explosion interactions.**
    - Camera/View movement controls:
        - **Zoom in:** **LEFT SHIFT** key
        - **Zoom out:** **RIGHT SHIFT** key
        - **Moving/rotating the camera in 4 directions:** **AWSD** or **ARROW** keys
            - the keys correspond to the direction the camera moves
            - the scene appears to move in opposite direction than the pressed key.
        - **Part selection – to make it glow green:** **MOUSE** hover
            - hovering over either label or its paired model part will make both glow.
            - any explosion interaction on either will be applied to both.
    - Explosion interactions:
        - Explosion Animation: Keys **1** and **2** – **1** for **forward, 2** for **reverse**
            - It explodes all parts while holding the key down.
            - When released, it stays in set positions.
            - Reverse explosion moves parts until they reach their initial positions.
        - Drag & Drop Explosion: **Mouse Click, Hold, Drag** and **Release**
            - It takes the drag size of the mouse but uses an internal direction to apply the size and move it there.
            - Until mouse release, it keeps dragging.
        - Offset Explosion: **Mouse Click + SPACE** bar
            - Each part has a predefined step size for its set explosion direction.
            - It moves the part and label for one step only or returns it to its initial state.
        - Riffling Explosion: **Mouse Hover + CTRL** key
            - The hovered part gets slightly moved while its predefined blockers get moved for a much bigger step, which makes us see the internal structure.
    - Experiment jump to next task is **ESC** key.

4. You will see a label text present in the scene on the instructions page.
5. **You have one minute to find its paired model part and explore its neighborhood. A part is a neighbor if it seems touching or almost touching our part.**
6. The supervisor will write down the found parts for you.
    - Not all parts have labels due to screen constraints.
    - Describe them as best as you can.
    - You can even say: *It is the part between the part with label A and the part with label B.*
7. When you are sure about your answer or the time is up, press ESC to continue to the next model.
8. There is only one method group here, so each model will be visited only once.
9. Use the Demo model to get familiar with the controls and the task. There is not time limit set for the demo.



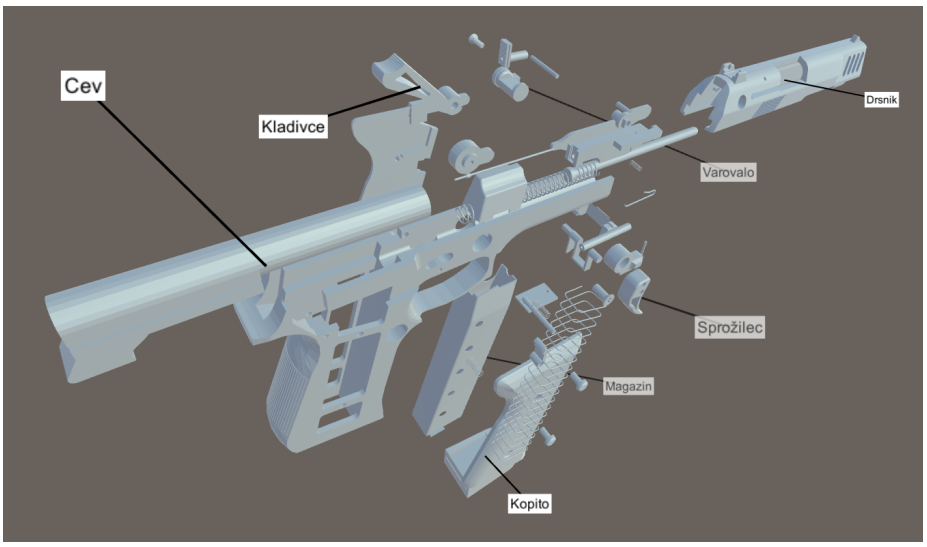
Kopito's neighborhood:

- Magazin,
- Spring,
- The wall touching Kopito, Magazin and spring.

Naloga Raziskuj

1. V tej nalogi boš sistem lahko tudi upravljal, ne samo opazoval.
2. Želimo pridobiti kakšen komentar o sistemu po končani nalogi.
3. Da lahko z nalogo pričnemo, si moraš najprej zapomniti načine upravljanja 3D Modela. **Pred nadaljevanjem si zapomni vse tipke in načine eksplozij!**
   - Upravljanje kamere/pogleda:
     - **Zoom in / Približaj:** **LEVA SHIFT** tipka
     - **Zoom out / Oddalji:** **DESNA SHIFT** tipka
     - **Premikanje/vrtenje kamere v 4 smeri:** **AWSD** ali **SMERNE** tipke
       - pritisk tipke premakne kamero v podano smer, gor premakne gor kamero.
       - ker se pogled premika s kamero, se zdi, da se model premika v obratno smer, torej da bo pritisk gor navidezno zavrtel model dol.
     - **Izbira kosa modela – zasveti zeleno:** postavi **MIŠKO** na element.
       - ko postaviš miško na kos ali oznako, se bosta obarvala/izbrala oba.
       - premik/eksplozija kosa bo tudi premaknil oznako.
   - Razstavljanje modela / Eksplozije:
     - Eksplozijska animacija: Tipki **1** in **2** – **1 eksplodira, 2** pa **povrne nazaj**
       - Eksplozija se nadaljuje dokler pritiskaš tipko.
       - Povratna/vzratna/rikverc eksplozija se premika do osnovne začetne lege.
     - Povleci In Spusti eksplozija: **z MIŠKO povleči izbrani kos**
       - Kolikor povlečeš, za toliko se bo premaknil kos, vendar v vnaprej definirano smer, ne nujno v smeri vlečenja.
       - Dokler ne spustiš pritiska na miški, lahko vlečeš prej izbrani element.
     - Eksplozijski korak: **hkrati pritisni MIŠKO in PRESLEDEK/SPACE tipko**
       - Vsak kos ima definiran osnovni korak v eksploziji.
       - S to metodo premaknemo izbrani kos za točno en korak oz. če je kos že v odmaknjeni legi, ga povrnemo v začetno lego.
     - Listanje or. Riffling: **drži CTRL tipko in izberi element**
       - Vsak kos ima definirane ovire v svoji smeri eksplozije, v katere lahko trči.
       - Listanje uporabi te definicije, da premakne izbrani kos malce iz začetne lege, hkrati pa še odmakne vse njegove ovire za veliko večji korak.
       - Tako vidimo kakšni so odnosi med deli ter notranjo zgradbo modela.

4. V tej nalogi poleg osnovnih navodil prejmeš tudi besedilo izbrane oznake.
5. Tvoja naloga je najti oznako z danim besedilom in njej pripadajoči kos modela.
6. Uporabi prej opisana upravljanja/interakcije, da najdeš sosede iskanega kosa.
    - **Sosed je kos, za katerega se zdi, da se (skoraj) dotika iskanega kosa.**
7. **Da ne bo predolgo trajalo, imaš minuto časa in ni nepravilnih odgovorov!**
8. Jaz bom zapisoval vse kose, ki jih označiš za sosede
    - Pazi, nimajo vsi kosi oznake, ker ni prostora na zaslonu.
    - Za kose brez oznak uporabi domišljijo in poskusi jih opisati.
    - Lahko jih tudi opišeš kot: *KOS, KI JE SOSED KOSU A IN KOSU B*
9. Ko si zadovoljen z odgovori oziroma je minunta potekla, pritisni **ESC** tipko za nadaljevanje.
10. V tej nalogi je zgolj en sklop, tako boš vsak model raziskoval zgolj enkrat.
11. Demonstracijski model nima časovne omejitve. Uporabi ga za seznanitev s tipkami.



Primer soseščine kosa Kopito:

- Magazin,
- Vzmet,
- Stranica na kopitu in ob magazinu in vzmeti.

Appendix C

Scenario File Format

Scenario files are files that define a simulation of user inputs needed for consistent simulation during experiments. For clarity, we name the files with a .scenario extension. The first line should always be a simulation duration **time-frame**, which defines the duration in seconds such as also other notations:

```
time-frame 60
```

All instruction lines are additive, which means that any time overlap will be combined. E.g. overlapping up and right will cause a rotation toward the top right corner for the second of overlap. On the other hand, contradictory values will negate each other's effect. E.g. overlapping up and down will produce nothing.

We also included glowing and blinking instructions for triggering part glowing to indicate a part to the user:

```
glow 18 0 5
blink 18 7 10
```

The camera instructions can be either zoom or rotations, of which the

latter are just arrow-key left-right or up-down options for simulating consistent rotation around the model as a user would. Both types start with the instruction type, followed by the start and end times that form an inclusive interval[start, end]:

```
up 0 5
down 7 10
right 0 5
left 10 15
zoom-in 6 12
zoom-out 50 60
```

For explosion interactions, we define keywords with prefixes **explode** and **reverse** and suffixes **all**, **offset**, **drag** and **riffling**. Note that riffling only has an effect limited by time, therefore no reverse version exists. Reverse time is defined to start closing riffling for the same duration it was being opened.

```
// start explosion animation for duration [3s, 12s]
explode-all 3 12
reverse-all 15 20
// offset click for part 18 on 22/25s
explode-offset 18 22
reverse-offset 18 25
// drag & drop explosion takes as parameters:
// part id, magnitude, start and end time.
explode-drag 18 50 40 50
reverse-drag 18 50 50 55
// parameters are: part id, start and reverse time
explode-riffling 18 30 35
```

Appendix D

Experiment Parameters

Textplosion allows a variety of changeable input parameters that can drastically effect the performance of the algorithm. For reproduction purposes, we recorded the used default values and boolean flags in the experiments. Values used during the experiment:

Texture resolution 128x128

Label transition duration 300f

Rotation constant 20.0f

Zoom speed 20.0f

Anchor point invisibility 0.66f

Unfreeze angle 25f

Unfreeze zoom 0.25f

Label margin 0.5f

Local search padding 7

Anchor attraction 1.5f

Surrounding repulsion 0.8f

Contour repulsion 0.3f

View border repulsion 1.0f

Label repulsion 2.0f

Anchor line repulsion 1.0f

Contour influence 1

View border influence 5

Anchor line influence 1.1f

K-means cluster count 4

Flags	**1**	**2**	**3**	**4**	**5**
Initializeonly	X	-	-	-	-
Reoderclusterlabels	-	X	X	X	X
Freezecheck	-	X	X	X	X
Anchorlinerepulsion	-	-	X	X	X
Sharedlabelrepulsion	-	-	X	X	X
Transferlabelrepulsion	-	-	X	X	X
Clustering	-	-	X	X	X
Position	-	-	X	X	X
Explosiondirection	-	-	-	X	X
Positionchange	-	-	-	X	X
Focuspointdistance	-	-	-	-	X

Table D.1: The difference of option support between prepared user experience label placement methods. An X stands for the flag to be set to use the option inside the method, while - is for ignoring the option. The method numbering corresponds to the method numbering in Subsection 5.3.3

Appendix E

Metric evaluation

Since we collected several tens of GB of data due to storing everything we could think of as a fail safe, we first needed to filter the data, which we did by creating CSV files per participant and task with rows containing only information important for our metric calculations. By setting the limit to 38 or 64 columns per row depending on amount of labels important for each task, we reduced the amount of data to just over a single GB of data. The difference in column count is due to ABC task storing information on three selected labels, while the remaining tasks only store information for the tracked or search term label. Note also that all coordinate values were recalculated into Tobii's active display coordinate system(ADCS) following the equations 5.7 and 5.8. The row structure of the CSV was designed as followed:

1. **Participant ID** stores the row number of participants inside the subjective results CSV,
2. **Task type** can be either: ABC, TRACKING, EXPLORATORYSEARCH,
3. **Task ID** is the experiment identification number,
4. **Subtask ID** equals the suffix of the corresponding XML file name,
5. **Method** is the subtask's method,

6. **Method letter** corresponds to the method name presented to the test subject,
7. **Model** is the codename of 3D model used in subtask,
8. **Model order** tells the order in task's shuffle of 5 models in range [1,5],
9. **Unique part count** holds a count of unique OBJ files,
10. **Part count** represents the count of all model parts, unique or copied prefabs,
11. **Label count** tells the amount of labels,
12. **Timestamp** identifies measurements inside the XML file and is generated as a long value of microseconds since 1990,
13. **Subtask time** represent the time since subtask start and is simply a difference of current and first retrieved timestamp,
14. **Gaze validity** determines if row represent valid eye gaze information, which is needed for the row to be useful,
15. **Gaze U average** is the average between left and right eye's gaze U coordinate on display screen in ADCS,
16. **Gaze V average** is the V version of Gaze U Average,
17. **Left pupil validity** determines if left pupil diameter is valid,
18. **Left pupil diameter** recorded left eye pupil diameter,
19. **Right pupil validity** determines if right pupil diameter is valid,
20. **Right pupil diameter** recorded right eye pupil diameter,
21. **Scenario key-frame** determines the progress stage of an experiment scenario,

22. **Scenario Δ_t** represents subtask time in seconds,
23. **Manual time notification** determines if the test subject already reported the time with ESC key,
24. **Cluster number** can be either 1 for single cluster mode or K-mean's K value, which was in our case 4,
25. **Empty cluster count** tells if any of the clusters holds no part and label pair.

Label subrow follows after the 25th column and repeats itself for each label the test subjected had to keep track of. It was defined to contain 13 values in the following order:

1. **Part ID** corresponds to the id connecting mesh and label,
2. $\boldsymbol{L_U}$ = label center's U coordinate,
3. $\boldsymbol{L_V}$ = label center's V coordinate,
4. **Gaze distance** is the euclidean distance between gaze and label center,
5. $\boldsymbol{Start_U}$ = label's bounding box start U coordinate,
6. $\boldsymbol{Start_V}$ = label's bounding box start V coordinate,
7. $\boldsymbol{End_U}$ = label's bounding box end U coordinate,
8. $\boldsymbol{End_V}$ = label's bounding box end V coordinate,
9. $\boldsymbol{AP_U}$ = label anchorpoint's U coordinate,
10. $\boldsymbol{AP_V}$ = label anchorpoint's V coordinate,
11. **Label center visibility** marks if whole label is visible in view area,
12. **Anchor point visibility** informs if any pixel is visible to the system,

13. **Annotation visibility** tells if label, anchor point and leader line are visible to the system.

After data reduction was finished, we could finally focus on producing actual metrics from the data. Since we were unsure, which metrics would work with our interactive and dynamic system, we started the analysis by aggregating all possible metrics mentioned in eye-tracking literature from Chapter 2 and calculated the values for the following 35 unique metrics:

1. [**ABD**] **- Average Blink Duration**,
2. [**AFD**] **- Average Fixation Duration**,
3. [**AFDR**] **- AOI Fixation Duration Ratio** is the ratio between time duration on AOI label and whole time duration between AOIs,
4. [**AFDT**] **- AOI Fixation Dwell Time** is the time of fixations spent on individual AOI labels.
5. [**ALGD**] **- Average Label Gaze Distance** is the Euclidean distance between gaze coordiantes and projected label centers,
6. [**AS**] **- AOI Attention Switch** is the fixation switch count to other AOI's count rate over all AOIs duration time,
7. [**ASA**] **- Average Saccade Angle** is the angle by which a saccade leaves a fixation to enter another,
8. [**AS**] **- Average Saccade Duration**,
9. [**ASL**] **- Average Saccade Length**,
10. [**ASR**] **- AOI Sample Rate** is the number of gaze samples gathered onto the AOI label divided by the total duration time,
11. [**BC**] **- Blink Count** is the count of longer durations of invalid gaze data,

12. [**BR**] - **Blink Rate** is the blink count divided by the finish time,
13. [**F2SR**] - **Fixation to Saccade Ratio**,
14. [**FAS**] - **Fixation Attention Switch** is the AS alternative over whole task duration,
15. [**FC**] - **Fixation Count**,
16. [**FDR**] - **Fixation Duration Ratio** is the ratio of AOI label's fixation duration sum over AOG's fixation duration time,
17. [**FDS**] - **Fixation Duration Sum**,
18. [**FFT**] - **First Fixation Time** is the overall first fixation without regard to location,
19. [**FR**] - **AOI Fixation Rate** is the fixation count inside the AOI label divided by FC,
20. [**FSC**] - **Fixation AOI Switch Count** is the count of all jumps between AOIs inside a translation matrix,
21. [**FSD**] - **Fixation Spatial Density** is the density of fixation visits to a 10x10 equally spaced grid dividing the view,
22. [**FTD**] - **Fixation Transition Density** is the density of the AOI translation matrix,
23. [**FTR**] - **Fixation Time Rate** is the ratio between FDS and TCT,
24. [**FVR**] - **AOI Fixation Visit Rate** is the percentage of fixations belonging to the AOI,
25. [**GSD**] - **Grid Sample Density** is the raw gaze sample version of FSD,
26. [**SC**] - **Saccade Count**,

27. [**SDS**] - **Saccade Duration Sum**,
28. [**SL**] - **Scanpath Length**,
29. [**SS**] - **Scan Speed** is the SL divided by TCT,
30. [**STDBD**] - **Standard Deviation of Blink Duration**,
31. [**STDFD**] - **Standard Deviation of Fixation Duration**,
32. [**STDPD**] - **Standard Deviation of Pupil Diameter**,
33. [**STDSA**] - **Standard Deviation of Saccade Angle**,
34. [**STDSD**] - **Standard Deviation of Saccade Duration**.
35. [**TCT**] - **Task Completion Time** is either the end time allocated to each task or in case of ABC task the minimum between a manually reported time by pressing the ESC key and the time detected from detecting triangle forming with AOI translation matrix,

We calculated all of the 35 metrics due to uncertainties of our interactive application generating dynamic stimuli. In truth ABC task could have been mostly considered a task with observed static stimuli, since we researched the data collected after the animation stops to observe his search behavior. We say mostly, since we are uncertain if the metrics were not effected due the dynamic stimuli from the animation part occurring just before metric calculations start. For tracking task we were certain of its dynamic component, therefore we even researched gaze sample related metrics besides the standard fixation ones. We thought the interactive task would also follow the dynamic stimuli route, however after observing participants getting familiar with the system at a slow pace, we noticed it had actually a strong static component due to the relatively long intervals between interactions, which we used in our favor to draw fixation heatmaps instead of the initially planned grid-sample related metrics. We preferred fixations to gaze samples due to their higher probability of being initiated by the observer, while gaze

samples as individual point may even be parts of saccades just passing by or even just a false positive by the eye-tracker in worst case. We say higher probability instead of certainty, since literature still questions, how much are fixations initiated by the observer instead of subconscious actions [16].

The analysis began with normal test's on individual samples, which produced p-values noted in Tables E.1, E.2, E.3, E.4, E.5 and E.6. After normal test failure we decided to use the Kruskal-Wallis H-test to determine if samples can be combined across models as shown in Figure 5.11. The resulting p-values in Tables E.8 and E.7 confirmed samples being of different populations and we had to analyze metrics by individual models between metrics, where Kruskal-Wallis H-test was used to determine any significant difference between method on individual metrics. The metrics and model pairs with potential significant difference between methods have bold p-values in Tables E.13, E.14, E.15, E.16, E.17, E.18, E.19, E.20, E.21 and E.22 and bold metrics inside the individual model's overview of medians and standard deviations in Tables E.9, E.10, E.11 and E.12. Note that not all the metrics are written in the tables due to some metrics being of no use for the task, such as metrics focusing on jumps between AOI's would have not potential in Tracking task due to only 1 AOI label present. Another reason for metrics missing is the absence of enough data in the samples for the statistical tests to give reasonable results. Such an example were the blinking metrics due to absence of blinking in case of most models as seen in Table 5.7.

Metric	FL_{eye}	HL_{eye}	CHL_{eye}	FL_{f1}	HL_{f1}	CHL_{f1}
AFD	**0.104**	0.001	7.4E-09	**0.581**	2.1E-07	**0.723**
AFDR	4.6E-41	1.6E-15	1.6E-15	2.5E-10	2.5E-10	3.4E-05
AFDT	**0.183**	**0.293**	**0.070**	0.042	0.006	**0.153**
ALGD	1.6E-13	3.0E-12	1.2E-05	1.5E-07	3.1E-05	**0.818**
AS	1.5E-04	2.6E-07	9.7E-09	**0.237**	7.7E-05	1.9E-04
ASA	7.4E-05	4.1E-13	3.5E-08	**0.086**	**0.330**	**0.104**
ASD	1.4E-06	**0.058**	2.9E-12	2.2E-05	7.0E-10	5.1E-04
ASL	0.044	0.006	5.4E-08	6.6E-08	1.3E-11	4.6E-05
ASR	0.001	**0.109**	**0.162**	**0.146**	8.6E-11	3.6E-12
F2SR	**0.151**	0.047	2.2E-06	3.7E-04	5.0E-07	5.8E-05
FAS	6.6E-06	0.002	0.007	**0.668**	**0.117**	0.005
FC	**0.102**	**0.204**	**0.110**	**0.092**	**0.208**	5.7E-04
FDR	**0.175**	**0.810**	**0.530**	**0.501**	**0.097**	2.1E-09
FDS	0.031	0.020	0.023	**0.066**	**0.113**	6.8E-07
FFT	5.1E-09	0.006	8.1E-15	4.4E-07	4.3E-04	**0.216**
FR	4.6E-41	1.6E-15	1.6E-15	2.5E-10	2.5E-10	3.4E-05
FSC	3.4E-04	0.020	**0.543**	**0.392**	**0.070**	0.016
FSD	**0.352**	0.029	**0.560**	**0.612**	**0.282**	**0.339**
FTD	**0.050**	**0.633**	**0.843**	**0.910**	**0.055**	**0.164**
FTR	**0.171**	**0.230**	**0.149**	**0.940**	**0.409**	1.9E-04
FVR	**0.177**	**0.536**	**0.375**	**0.814**	0.015	1.1E-09
GSD	**0.139**	0.001	**0.288**	0.003	0.024	**0.560**
SC	2.3E-06	0.014	6.4E-04	**0.373**	**0.067**	**0.385**
SDS	0.002	0.004	0.006	**0.140**	0.032	2.1E-14
SL	1.2E-12	2.7E-05	5.2E-06	0.003	6.2E-04	**0.421**
SS	6.6E-14	5.9E-07	4.6E-07	6.6E-04	1.3E-04	**0.200**
STDFD	0.023	4.3E-08	9.2E-07	**0.300**	**0.877**	**0.126**
STDPD	0.005	0.028	0.002	**0.108**	0.038	5.1E-04
STDSA	0.005	0.007	**0.357**	**0.218**	0.005	**0.256**
STDSD	0.008	0.001	**0.061**	2.3E-06	3.4E-10	5.8E-06
TCT	0.001	0.003	**0.119**	**0.928**	**0.111**	**0.205**

Table E.1: Normal distribution test results for ABC task per algorithm for `eye` and `f1` models. Presented are the p-values, which are in bold when normal distribution is not rejected by the significance level of $\alpha = 0.05$

Metric	FL_{gpu}	HL_{gpu}	CHL_{gpu}	$FL_{tractor}$	$HL_{tractor}$	$CHL_{tractor}$
AFD	0.002	3.2E-07	1.0E-06	**0.850**	**0.228**	**0.342**
AFDR	4.6E-41	4.6E-41	2.5E-10	2.5E-10	1.6E-15	9.8E-04
AFDT	**0.106**	**0.647**	0.002	**0.481**	0.008	**0.214**
ALGD	4.0E-07	0.003	0.019	9.9E-08	**0.208**	**0.598**
AS	**0.167**	0.004	**0.327**	**0.527**	5.9E-04	2.0E-04
ASA	**0.108**	**0.340**	**0.136**	**0.354**	6.5E-09	0.038
ASD	4.9E-05	3.8E-06	1.2E-07	1.8E-07	8.8E-08	3.5E-05
ASL	8.9E-07	3.0E-05	9.3E-05	0.011	4.6E-11	0.013
ASR	**0.106**	0.002	4.7E-12	**0.478**	4.2E-11	2.3E-09
F2SR	1.6E-05	2.2E-07	0.045	2.4E-09	6.9E-05	4.4E-04
FAS	**0.520**	0.017	**0.402**	**0.742**	0.037	0.009
FC	0.030	**0.123**	0.003	0.017	2.4E-04	2.5E-09
FDR	**0.085**	**0.086**	4.8E-04	**0.291**	2.3E-07	5.7E-05
FDS	2.1E-05	0.022	0.002	0.020	7.9E-04	4.0E-09
FFT	4.0E-13	4.0E-07	6.8E-05	1.0E-04	**0.190**	7.3E-12
FR	4.6E-41	4.6E-41	2.5E-10	2.5E-10	1.6E-15	9.8E-04
FSC	**0.623**	0.012	**0.465**	**0.579**	**0.500**	0.044
FSD	**0.878**	**0.153**	**0.459**	**0.700**	**0.059**	**0.391**
FTD	**0.078**	**0.224**	**0.344**	**0.573**	**0.147**	0.008
FTR	**0.097**	**0.282**	0.039	0.013	0.032	0.009
FVR	**0.187**	**0.258**	5.4E-06	0.017	6.3E-08	2.0E-04
GSD	**0.751**	**0.472**	**0.272**	**0.103**	**0.169**	0.009
SC	**0.061**	**0.501**	**0.810**	**0.551**	**0.075**	**0.934**
SDS	4.0E-04	3.9E-04	1.2E-07	0.024	3.2E-06	**0.661**
SL	**0.069**	0.006	**0.681**	5.4E-06	**0.725**	**0.252**
SS	2.6E-04	4.5E-04	**0.475**	6.7E-08	**0.444**	0.009
STDFD	1.2E-05	5.1E-12	7.5E-11	0.019	**0.341**	**0.399**
STDPD	0.005	**0.818**	0.018	6.5E-04	4.0E-04	**0.245**
STDSA	**0.324**	**0.108**	**0.993**	**0.322**	**0.997**	0.042
STDSD	0.002	0.028	2.4E-04	1.0E-05	5.2E-13	4.8E-06
TCT	**0.178**	0.030	**0.105**	**0.690**	**0.092**	**0.157**

Table E.2: Normal distribution test results for ABC task per algorithm for `gpu` and `tractor` models. Presented are the p-values, which are in bold when normal distribution is not rejected by the significance level of $\alpha = 0.05$

Metric	$FL_{pocketwatch}$	$HL_{pocketwatch}$	$CHL_{pocketwatch}$
AFD	2.9E-07	**0.878**	1.1E-04
AFDR	4.6E-41	4.6E-41	4.6E-41
AFDT	**0.484**	**0.368**	8.2E-04
ALGD	0.020	3.5E-04	**0.086**
AS	2.6E-05	**0.145**	0.010
ASA	0.012	3.1E-09	**0.208**
ASD	2.9E-14	0.002	1.2E-08
ASL	0.001	**0.499**	0.008
ASR	0.002	**0.059**	4.7E-10
F2SR	1.1E-07	5.4E-06	8.6E-05
FAS	**0.172**	**0.174**	1.2E-04
FC	**0.150**	**0.973**	**0.085**
FDR	0.010	0.046	0.016
FDS	**0.113**	**0.058**	**0.065**
FFT	**0.240**	1.5E-07	6.5E-06
FR	4.6E-41	4.6E-41	4.6E-41
FSC	**0.214**	**0.252**	0.002
FSD	**0.788**	**0.217**	**0.603**
FTD	**0.143**	**0.057**	**0.976**
FTR	**0.685**	0.010	**0.311**
FVR	0.002	**0.278**	0.001
GSD	**0.245**	1.2E-07	**0.116**
SC	**0.495**	**0.358**	**0.248**
SDS	**0.130**	0.035	0.014
SL	**0.314**	0.004	**0.748**
SS	**0.261**	**0.478**	**0.505**
STDFD	2.0E-10	**0.757**	0.003
STDPD	0.015	**0.126**	**0.111**
STDSA	**0.290**	2.9E-08	0.018
STDSD	1.0E-11	7.9E-04	1.3E-06
TCT	**0.654**	**0.574**	**0.468**

Table E.3: Normal distribution test results for ABC task per algorithm for `pocketwatch` model. Presented are the p-values, which are in bold when normal distribution is not rejected by the significance level of $\alpha = 0.05$

Metric	FL_{eye}	HL_{eye}	CHL_{eye}	FL_{f1}	HL_{f1}	CHL_{f1}
AFD	**0.503**	**0.540**	**0.151**	0.008	9.8E-04	1.4E-08
AFDT	**0.289**	**0.250**	**0.273**	2.7E-04	**0.085**	**0.135**
ALGD	**0.381**	**0.057**	1.4E-14	8.7E-05	0.026	0.037
AS	**0.670**	**0.841**	**0.829**	**0.874**	**0.374**	**0.877**
ASA	1.0E-10	3.6E-07	**0.612**	**0.376**	1.1E-07	**0.403**
ASD	9.3E-13	9.1E-09	1.7E-13	0.026	1.4E-05	2.6E-04
ASL	1.6E-12	1.8E-11	8.6E-07	2.5E-07	8.9E-08	2.3E-05
ASR	**0.051**	0.008	**0.106**	7.1E-08	5.0E-06	0.013
F2SR	0.015	6.0E-08	6.7E-05	0.008	1.2E-05	0.001
FC	0.003	**0.195**	0.030	0.008	0.019	**0.243**
FDR	**0.058**	0.042	**0.194**	**0.657**	**0.230**	**0.637**
FDS	3.1E-04	1.4E-09	1.6E-04	0.004	9.3E-05	8.7E-04
FFT	4.6E-04	2.2E-13	2.1E-09	2.8E-05	5.8E-07	2.6E-13
FSD	**0.141**	**0.064**	**0.096**	0.003	0.036	**0.575**
FTD	**0.582**	**0.582**	**0.582**	**0.582**	**0.582**	**0.582**
FTR	3.5E-04	**0.084**	0.029	0.007	0.010	1.3E-05
FVR	6.7E-05	3.6E-06	**0.118**	0.036	**0.086**	**0.055**
GSD	**0.212**	**0.258**	**0.105**	0.003	**0.202**	**0.363**
SC	**0.079**	**0.102**	0.042	**0.213**	**0.167**	**0.100**
SDS	0.002	5.3E-15	1.5E-07	2.3E-06	5.5E-09	1.6E-04
SL	7.1E-07	0.008	8.2E-14	2.3E-05	0.050	0.002
SS	5.0E-04	**0.074**	2.4E-12	**0.058**	**0.051**	0.028
STDFD	**0.121**	**0.119**	0.017	**0.236**	**0.252**	**0.211**
STDPD	**0.651**	**0.268**	0.025	**0.320**	**0.542**	**0.518**
STDSA	0.019	0.013	**0.168**	**0.117**	0.036	0.036
STDSD	8.1E-11	7.3E-11	**0.524**	**0.091**	1.7E-06	3.3E-07

Table E.4: Normal distribution test results for Tracking task per algorithm for `eye` and `f1` models. Presented are the p-values, which are in bold when normal distribution is not rejected by the significance level of $\alpha = 0.05$

Metric	FL_{gpu}	HL_{gpu}	CHL_{gpu}	$FL_{tractor}$	$HL_{tractor}$	$CHL_{tractor}$
AFD	**0.991**	**0.440**	5.0E-04	6.4E-05	1.1E-07	1.3E-04
AFDT	**0.944**	**0.105**	0.004	**0.356**	**0.210**	**0.560**
ALGD	0.005	7.3E-14	5.4E-06	**0.320**	4.1E-14	**0.299**
AS	**0.503**	0.050	**0.731**	**0.316**	**0.402**	**0.721**
ASA	**0.058**	5.5E-10	0.001	6.7E-05	**0.260**	**0.299**
ASD	0.005	0.013	0.002	1.1E-09	0.043	2.2E-09
ASL	0.011	2.3E-10	0.003	0.009	0.027	0.009
ASR	2.9E-05	0.001	**0.057**	**0.068**	9.4E-05	4.9E-08
F2SR	0.005	5.0E-08	1.2E-04	**0.294**	8.9E-05	2.3E-04
FC	0.030	7.5E-04	0.014	**0.110**	1.1E-05	0.004
FDR	**0.451**	**0.059**	**0.142**	**0.478**	**0.808**	**0.263**
FDS	**0.126**	**0.054**	0.001	9.5E-04	4.8E-08	2.1E-06
FFT	1.3E-04	1.2E-10	1.7E-13	3.5E-06	2.8E-04	4.4E-15
FSD	**0.287**	0.002	3.5E-04	**0.056**	0.004	**0.368**
FTD	**0.582**	**0.582**	**0.582**	**0.582**	**0.582**	**0.582**
FTR	4.1E-07	**0.175**	**0.619**	0.003	**0.070**	**0.120**
FVR	**0.674**	**0.201**	**0.270**	**0.151**	**0.109**	0.027
GSD	**0.653**	1.8E-04	0.008	0.014	**0.208**	**0.360**
SC	3.7E-04	5.5E-04	**0.084**	0.028	**0.241**	**0.156**
SDS	**0.098**	0.033	0.004	4.4E-09	1.7E-11	0.004
SL	0.002	1.4E-07	1.0E-10	**0.182**	0.020	3.1E-04
SS	9.9E-06	0.024	3.6E-12	**0.229**	**0.235**	0.001
STDFD	**0.264**	**0.081**	0.029	**0.713**	**0.633**	**0.617**
STDPD	**0.111**	**0.205**	**0.439**	3.7E-05	**0.478**	0.032
STDSA	0.003	**0.057**	**0.095**	**0.687**	8.5E-05	**0.053**
STDSD	**0.132**	**0.461**	1.0E-08	3.8E-12	0.008	0.015

Table E.5: Normal distribution test results for Tracking task per algorithm for `gpu` and `tractor` models. Presented are the p-values, which are in bold when normal distribution is not rejected by the significance level of $\alpha = 0.05$

Metric	$FL_{pocketwatch}$	$HL_{pocketwatch}$	$CHL_{pocketwatch}$
AFD	**0.058**	**0.187**	0.047
AFDT	**0.247**	**0.352**	**0.153**
ALGD	**0.101**	4.6E-14	0.011
AS	0.039	**0.491**	2.0E-04
ASA	**0.251**	0.007	**0.055**
ASD	4.7E-04	**0.104**	4.9E-11
ASL	3.7E-10	7.0E-11	1.3E-06
ASR	4.3E-05	**0.054**	**0.492**
F2SR	0.004	1.7E-07	0.007
FC	9.4E-04	0.048	0.005
FDR	0.045	**0.106**	0.028
FDS	0.022	0.041	0.003
FFT	**0.051**	0.008	6.4E-08
FSD	1.1E-05	0.015	2.8E-04
FTD	**0.582**	**0.582**	**0.582**
FTR	**0.074**	**0.182**	**0.286**
FVR	0.045	**0.109**	**0.098**
GSD	2.1E-05	0.002	1.5E-07
SC	2.4E-05	0.013	1.3E-04
SDS	0.022	0.031	0.002
SL	3.3E-05	0.018	5.0E-06
SS	0.007	0.021	**0.059**
STDFD	0.004	**0.088**	**0.081**
STDPD	**0.324**	**0.072**	**0.181**
STDSA	0.005	0.001	7.8E-04
STDSD	0.040	**0.163**	0.008

Table E.6: Normal distribution test results for Tracking task per algorithm for `pocketwatch` model. Presented are the p-values, which are in bold when normal distribution is not rejected by the significance level of $\alpha = 0.05$

Metric	H_{model}	p_{model}	$H_{algorithm}$	$p_{algorithm}$	H_{both}	p_{both}
AFD	632.413	1.5E-139	638.113	8.6E-141	899.798	4.1E-196
AFDR	706.860	9.6E-156	713.990	2.7E-157	929.161	1.7E-202
AFDT	496.133	6.6E-110	500.605	7.0E-111	715.706	3.9E-156
ALGD	632.413	1.5E-139	638.113	8.6E-141	899.798	4.1E-196
AS	632.427	1.5E-139	638.128	8.5E-141	899.804	4.1E-196
ASA	112.935	2.2E-26	201.380	1.0E-45	270.146	2.2E-59
ASD	627.890	1.4E-138	633.551	8.4E-140	894.044	7.3E-195
ASL	621.882	2.9E-137	633.551	8.4E-140	890.529	4.2E-194
ASR	632.414	1.5E-139	638.115	8.6E-141	899.799	4.1E-196
F2SR	576.732	1.9E-127	548.724	2.4E-121	806.004	9.5E-176
FAS	633.119	1.0E-139	638.833	6.0E-141	900.100	3.5E-196
FC	410.630	2.7E-91	488.198	3.5E-108	656.673	2.5E-143
FDR	632.442	1.5E-139	638.143	8.5E-141	899.811	4.1E-196
FDS	632.413	1.5E-139	638.113	8.6E-141	899.798	4.1E-196
FFT	632.413	1.5E-139	638.113	8.6E-141	899.798	4.1E-196
FR	706.860	9.6E-156	713.990	2.7E-157	929.161	1.7E-202
FSC	132.857	9.7E-31	17.851	2.4E-05	170.587	9.1E-38
FSD	633.085	1.1E-139	638.798	6.1E-141	900.085	3.5E-196
FTD	638.711	6.4E-141	644.526	3.5E-142	902.468	1.1E-196
FTR	632.413	1.5E-139	638.113	8.6E-141	899.798	4.1E-196
FVR	632.497	1.4E-139	638.199	8.2E-141	899.834	4.0E-196
GSD	632.525	1.4E-139	638.228	8.1E-141	899.846	4.0E-196
SC	519.303	6.0E-115	571.118	3.2E-126	783.306	8.1E-171
SDS	627.890	1.4E-138	633.551	8.4E-140	894.044	7.3E-195
SL	627.890	1.4E-138	633.551	8.4E-140	894.044	7.3E-195
SS	623.104	1.6E-137	625.474	4.8E-138	885.698	4.7E-193
STDFD	490.814	9.5E-109	495.238	1.0E-109	708.520	1.4E-154
STDPD	632.413	1.5E-139	638.113	8.6E-141	899.798	4.1E-196
STDSA	198.467	4.5E-45	258.608	3.5E-58	360.965	4.1E-79
STDSD	592.273	8.0E-131	597.612	5.5E-132	845.934	2.0E-184
TCT	632.413	1.5E-139	638.113	8.6E-141	899.798	4.1E-196

Table E.7: Kruskal-Wallis test results from comparing Tracking task metrics paired of with corresponding model or algorithm indicators. With all of the p-values under the significance level of $\alpha = 0.05$ we rejected the hypothesis of same population needed for combining metrics across models.

Metric	H_{model}	p_{model}	$H_{algorithm}$	$p_{algorithm}$	H_{both}	p_{both}
AFD	624.875	6.5E-138	630.515	3.9E-139	888.961	9.2E-194
AFDT	526.685	1.5E-116	531.439	1.4E-117	756.314	5.9E-165
ALGD	629.398	6.8E-139	635.072	3.9E-140	895.415	3.7E-195
AS	624.881	6.5E-138	630.521	3.9E-139	888.963	9.2E-194
ASA	75.396	3.9E-18	130.150	3.8E-30	193.119	1.2E-42
ASD	608.293	2.6E-134	613.787	1.7E-135	866.347	7.5E-189
ASL	552.600	3.4E-122	599.834	1.8E-132	823.148	1.8E-179
ASR	628.239	1.2E-138	633.173	1.0E-139	893.451	9.8E-195
F2SR	577.110	1.6E-127	561.056	5.0E-124	812.161	4.4E-177
FC	78.889	6.6E-19	159.456	1.5E-36	218.104	4.4E-48
FDR	565.592	5.1E-125	532.714	7.3E-118	785.392	2.8E-171
FDS	624.875	6.5E-138	630.515	3.9E-139	888.961	9.2E-194
FFT	624.876	6.5E-138	630.516	3.9E-139	888.961	9.2E-194
FSD	627.215	2.0E-138	632.897	1.2E-139	889.957	5.6E-194
FTD	510.154	5.9E-113	394.577	8.3E-88	590.486	6.0E-129
FTR	622.469	2.2E-137	626.422	3.0E-138	884.743	7.6E-193
FVR	565.737	4.7E-125	532.853	6.8E-118	785.453	2.8E-171
GSD	629.805	5.5E-139	635.487	3.2E-140	895.589	3.4E-195
SC	384.584	1.3E-85	478.306	5.0E-106	633.240	3.1E-138
SDS	608.293	2.6E-134	613.787	1.7E-135	866.347	7.5E-189
SL	593.459	4.4E-131	607.892	3.2E-134	853.204	5.4E-186
SS	604.691	1.6E-133	607.680	3.6E-134	860.050	1.7E-187
STDFD	129.934	4.2E-30	131.110	2.3E-30	220.083	1.6E-48
STDPD	629.398	6.8E-139	635.072	3.9E-140	895.415	3.7E-195
STDSA	107.863	2.9E-25	147.735	5.4E-34	223.458	3.0E-49
STDSD	521.310	2.2E-115	526.018	2.1E-116	748.852	2.4E-163

Table E.8: Kruskal-Wallis test results from comparing Tracking task metrics paired of with corresponding model or algorithm indicators. With all of the p-values bellow the significance level of $\alpha = 0.05$ we rejected the hypothesis of same population needed for combining metrics across models.

Metric	H_{eye}	p_{eye}	H_{f1}	p_{f1}	H_{gpu}	p_{gpu}
AFD	0.490	0.783	2.270	0.321	4.034	0.133
AFDR	1.012	0.603	1.092	0.579	4.049	0.132
AFDT	1.443	0.486	12.385	**0.002**	11.918	**0.003**
ALGD	6.046	**0.049**	17.817	**1.4E-04**	31.889	**1.2E-07**
AS	4.649	0.098	7.008	**0.030**	8.977	**0.011**
ASA	1.367	0.505	0.527	0.768	0.186	0.911
ASD	1.556	0.459	0.488	0.783	0.619	0.734
ASL	0.229	0.892	1.134	0.567	2.483	0.289
ASR	0.992	0.609	26.811	**1.5E-06**	21.452	**2.2E-05**
F2SR	1.635	0.442	0.227	0.893	0.732	0.694
FAS	2.675	0.263	5.529	0.063	0.563	0.755
FC	1.337	0.512	6.272	**0.043**	7.831	**0.020**
FDR	1.234	0.540	21.461	**2.2E-05**	10.807	**0.005**
FDS	1.362	0.506	2.866	0.239	1.513	0.469
FFT	15.979	**3.4E-04**	59.562	**1.2E-13**	0.457	0.796
FR	1.012	0.603	1.092	0.579	4.049	0.132
FSC	1.842	0.398	8.058	**0.018**	0.511	0.775
FSD	0.277	0.871	6.445	**0.040**	0.007	0.996
FTD	1.856	0.395	6.315	**0.043**	0.832	0.660
FTR	1.585	0.453	11.761	**0.003**	5.455	0.065
FVR	0.503	0.778	17.785	**1.4E-04**	14.494	**7.1E-04**
GSD	0.174	0.916	7.898	**0.019**	0.829	0.661
SC	0.599	0.741	4.529	0.104	4.134	0.127
SDS	0.747	0.688	3.020	0.221	12.371	**0.002**
SL	0.975	0.614	7.182	**0.028**	4.498	0.105
SS	1.304	0.521	9.212	**0.010**	4.091	0.129
STDFD	0.689	0.709	2.602	0.272	4.318	0.115
STDPD	0.591	0.744	2.090	0.352	2.752	0.253
STDSA	2.127	0.345	0.447	0.800	0.145	0.930
STDSD	0.839	0.658	0.512	0.774	1.945	0.378
TCT	0.845	0.655	8.856	**0.012**	0.878	0.645

Table E.9: Kruskal-Wallis test results from comparing ABC task metrics separated per algorithms on `eye`, `f1` and `gpu` models. The difference in medians between algorithms is statistically significant for those metrics that have bold p-values bellow the significance level of $\alpha = 0.05$.

Metric	$H_{tractor}$	$p_{tractor}$	$H_{pocketwatch}$	$p_{pocketwatch}$
AFD	0.574	0.750	1.464	0.481
AFDR	10.951	**0.004**	Too similar samples	
AFDT	18.772	**8.4E-05**	0.236	0.889
ALGD	10.321	**0.006**	9.254	**0.010**
AS	20.275	**4.0E-05**	8.376	**0.015**
ASA	1.581	0.454	8.279	**0.016**
ASD	1.523	0.467	0.366	0.833
ASL	3.796	0.150	6.530	**0.038**
ASR	39.491	**2.7E-09**	0.899	0.638
F2SR	4.087	0.130	0.021	0.989
FAS	13.587	**0.001**	10.867	**0.004**
FC	3.798	0.150	0.743	0.690
FDR	38.198	**5.1E-09**	0.647	0.723
FDS	3.411	0.182	0.595	0.743
FFT	55.896	**7.3E-13**	55.344	**9.6E-13**
FR	10.951	**0.004**	Too similar samples	
FSC	16.309	**2.9E-04**	5.840	0.054
FSD	6.521	**0.038**	0.596	0.742
FTD	32.555	**8.5E-08**	0.752	0.687
FTR	21.685	**2.0E-05**	11.255	**0.004**
FVR	35.766	**1.7E-08**	0.623	0.732
GSD	9.126	**0.010**	0.991	0.609
SC	6.798	**0.033**	0.308	0.857
SDS	1.132	0.568	2.409	0.300
SL	11.976	**0.003**	3.587	0.166
SS	20.454	**3.6E-05**	13.116	**0.001**
STDFD	1.171	0.557	0.023	0.989
STDPD	0.658	0.720	0.160	0.923
STDSA	0.940	0.625	7.623	**0.022**
STDSD	0.946	0.623	0.674	0.714
TCT	12.097	**0.002**	24.597	**4.6E-06**

Table E.10: Kruskal-Wallis test results from comparing ABC task metrics separated per algorithms on `tractor` and `pocketwatch` models. The difference in medians between algorithms is statistically significant for those metrics that have bold p-values bellow the significance level of $\alpha = 0.05$.

Metric	H_{eye}	p_{eye}	H_{f1}	p_{f1}	H_{gpu}	p_{gpu}
AFD	0.081	0.960	1.439	0.487	11.181	**0.004**
AFDT	0.024	0.988	3.016	0.221	9.517	**0.009**
ALGD	6.546	**0.038**	0.641	0.726	11.306	**0.004**
AS	0.702	0.704	1.156	0.561	2.909	0.234
ASA	1.657	0.437	1.395	0.498	6.236	**0.044**
ASD	1.068	0.586	0.786	0.675	0.946	0.623
ASL	4.501	0.105	4.440	0.109	5.038	0.081
ASR	3.429	0.180	6.858	**0.032**	1.320	0.517
F2SR	0.899	0.638	1.691	0.429	4.043	0.132
FC	1.857	0.395	0.563	0.755	3.375	0.185
FDR	1.366	0.505	2.875	0.237	5.769	0.056
FDS	2.824	0.244	4.501	0.105	1.012	0.603
FFT	0.530	0.767	0.448	0.799	0.078	0.962
FSD	2.656	0.265	0.738	0.691	4.700	0.095
FTR	2.731	0.255	2.319	0.314	3.404	0.182
FVR	2.257	0.324	0.405	0.817	3.327	0.189
GSD	1.562	0.458	2.000	0.368	6.630	**0.036**
SC	1.569	0.456	5.045	0.080	1.691	0.429
SDS	1.069	0.586	4.453	0.108	1.546	0.462
SL	5.265	0.072	0.671	0.715	2.628	0.269
SS	6.271	**0.043**	2.541	0.281	2.526	0.283
STDFD	0.273	0.873	4.106	0.128	1.883	0.390
STDPD	0.328	0.849	0.520	0.771	1.581	0.454
STDSA	3.180	0.204	1.255	0.534	6.110	**0.047**
STDSD	3.042	0.218	0.610	0.737	0.528	0.768

Table E.11: Kruskal-Wallis test results from comparing Tracking task metrics separated per algorithms on `eye`, `f1` and `gpu` models. The difference in medians between algorithms is statistically significant for those metrics that have bold p-values bellow the significance level of $\alpha = 0.05$.

Metric	$H_{tractor}$	$p_{tractor}$	$H_{pocketwatch}$	$p_{pocketwatch}$
AFD	2.645	0.266	2.367	0.306
AFDT	5.341	0.069	3.270	0.195
ALGD	4.072	0.131	6.912	**0.032**
AS	3.998	0.135	0.787	0.675
ASA	0.763	0.683	0.645	0.724
ASD	0.951	0.621	1.003	0.606
ASL	0.649	0.723	0.514	0.773
ASR	1.408	0.495	1.928	0.381
F2SR	2.130	0.345	0.675	0.714
FC	0.959	0.619	1.378	0.502
FDR	2.831	0.243	2.759	0.252
FDS	3.698	0.157	2.872	0.238
FFT	2.535	0.282	0.927	0.629
FSD	0.079	0.961	1.296	0.523
FTR	3.032	0.220	5.907	0.052
FVR	1.425	0.490	2.039	0.361
GSD	0.256	0.880	0.411	0.814
SC	2.929	0.231	1.439	0.487
SDS	3.220	0.200	2.112	0.348
SL	0.939	0.625	0.026	0.987
SS	0.139	0.933	0.264	0.876
STDFD	21.918	**1.7E-05**	3.478	0.176
STDPD	4.009	0.135	0.134	0.935
STDSA	1.125	0.570	0.648	0.723
STDSD	0.663	0.718	0.770	0.680

Table E.12: Kruskal-Wallis test results from comparing Tracking task metrics separated per algorithms on `tractor` and `pocketwatch` models. The difference in medians between algorithms is statistically significant for those metrics that have bold p-values bellow the significance level of $\alpha = 0.05$.

Metric	FL_{med}	FL_{std}	HL_{med}	HL_{std}	CHL_{med}	CHL_{std}
AFD	399.588	119.031	384.448	147.408	388.611	183.159
AFDR	0.333	1.1E-16	0.333	0.062	0.333	0.062
AFDT	284.056	134.052	282.206	153.300	224.839	189.430
ALGD	191.551	26.491	192.784	28.824	198.675	18.890
AS	1.9E-04	1.5E-04	2.6E-04	1.9E-04	1.8E-04	2.0E-04
ASA	3.688	11.121	4.537	16.663	8.098	9.173
ASD	222.063	195.093	292.384	197.366	242.533	289.284
ASL	84.577	76.924	83.319	101.299	84.391	96.880
ASR	0.054	0.085	0.078	0.057	0.065	0.047
F2SR	0.536	0.271	0.536	0.511	0.500	0.389
FAS	1.0E-04	1.2E-04	1.7E-04	1.3E-04	1.7E-04	9.0E-05
FC	5.000	6.402	7.500	6.432	7.000	6.175
FDR	0.189	0.090	0.178	0.075	0.159	0.095
FDS	2176.218	2774.667	2970.098	2759.168	2376.077	2899.215
FFT	10259.329	456.086	9992.871	197.225	10065.029	1264.309
FR	0.333	1.1E-16	0.333	0.062	0.333	0.062
FSC	1.000	1.731	2.000	1.780	2.000	1.145
FSD	0.040	0.028	0.045	0.024	0.045	0.027
FTD	0.444	0.185	0.556	0.204	0.444	0.189
FTR	0.168	0.181	0.247	0.189	0.190	0.231
FVR	0.156	0.088	0.162	0.075	0.162	0.091
GSD	0.090	0.055	0.095	0.045	0.090	0.039
SC	11.000	16.698	12.500	17.183	12.500	16.610
SDS	2681.414	3553.268	3924.970	3239.243	3614.081	3302.657
SL	895.015	3181.249	1714.786	2993.643	1381.813	2021.945
SS	0.068	0.271	0.120	0.199	0.112	0.173
STDFD	133.014	106.313	143.067	145.476	138.370	198.784
STDPD	0.128	0.052	0.117	0.056	0.110	0.063
STDSA	11.149	15.431	10.777	12.932	23.499	12.401
STDSD	149.972	160.538	183.129	192.237	150.794	127.043
TCT	11552.842	2851.913	12236.635	2517.149	11963.655	1915.761

Table E.13: List of metric medians and standard deviations on the `eye` model during ABC tasks. The bold metrics correspond to the statistically significant median differences between metrics of different algorithms, according to Table E.9.

Metric	FL_{med}	FL_{std}	HL_{med}	HL_{std}	CHL_{med}	CHL_{std}
AFD	443.030	105.538	477.251	170.188	421.611	100.950
AFDR	0.333	0.086	0.333	0.086	0.333	0.117
AFDT	387.501	241.771	319.340	221.988	216.512	132.879
ALGD	184.655	31.613	200.196	26.751	219.691	31.894
AS	2.2E-04	1.3E-04	1.9E-04	1.6E-04	1.1E-04	1.3E-04
ASA	4.493	3.733	5.993	4.779	4.629	4.146
ASD	225.117	168.389	212.401	241.998	214.998	171.730
ASL	91.818	78.524	94.488	157.709	102.724	58.690
ASR	0.057	0.042	0.043	0.061	0.011	0.032
F2SR	0.403	0.370	0.481	0.347	0.404	0.319
FAS	1.0E-04	7.0E-05	7.6E-05	8.2E-05	5.2E-05	6.8E-05
FC	13.000	6.082	15.000	6.200	17.000	4.796
FDR	0.145	0.069	0.110	0.093	0.046	0.065
FDS	6273.295	2731.469	6922.828	2626.416	7655.645	2025.253
FFT	12230.147	502.445	9970.650	269.697	9876.270	99.529
FR	0.333	0.086	0.333	0.086	0.333	0.117
FSC	2.000	1.280	1.000	1.255	1.000	1.017
FSD	0.070	0.025	0.070	0.027	0.085	0.026
FTD	0.389	0.155	0.333	0.178	0.333	0.149
FTR	0.325	0.130	0.405	0.158	0.438	0.121
FVR	0.118	0.064	0.105	0.090	0.040	0.065
GSD	0.135	0.055	0.130	0.054	0.155	0.049
SC	28.500	17.766	30.000	20.279	44.000	17.633
SDS	9137.896	3114.686	9015.768	3172.167	9226.727	1485.405
SL	2800.495	1947.869	2987.640	2542.705	3690.064	1806.278
SS	0.133	0.117	0.175	0.173	0.234	0.125
STDFD	242.253	135.070	225.573	147.299	182.331	114.997
STDPD	0.096	0.045	0.108	0.051	0.097	0.062
STDSA	17.694	9.568	20.069	11.966	18.221	10.051
STDSD	167.927	150.353	217.588	219.735	172.239	146.550
TCT	17998.250	3087.376	14857.572	3268.101	17048.914	3118.660

Table E.14: List of metric medians and standard deviations on the `f1` model during ABC tasks. The bold metrics correspond to the statistically significant median differences between metrics of different algorithms, according to Table E.9.

Metric	FL_{med}	FL_{std}	HL_{med}	HL_{std}	CHL_{med}	CHL_{std}
AFD	406.191	94.715	388.302	98.202	385.526	88.274
AFDR	0.333	1.1E-16	0.333	1.1E-16	0.333	0.086
AFDT	306.355	109.010	248.588	101.133	172.099	129.366
ALGD	346.977	35.677	242.201	50.113	329.554	53.179
AS	2.8E-04	1.9E-04	3.4E-04	1.9E-04	2.0E-04	1.3E-04
ASA	6.266	4.453	7.069	4.258	5.938	4.614
ASD	239.724	165.878	244.000	204.283	267.330	193.745
ASL	156.861	100.868	143.642	89.723	164.501	101.045
ASR	0.070	0.034	0.053	0.040	0.019	0.059
F2SR	0.460	0.247	0.483	0.355	0.515	0.236
FAS	1.5E-04	9.5E-05	1.8E-04	1.3E-04	1.5E-04	1.2E-04
FC	11.500	5.249	16.500	5.381	17.500	5.157
FDR	0.157	0.053	0.099	0.064	0.077	0.076
FDS	4785.463	2292.793	6756.274	2074.574	6994.994	2145.627
FFT	9754.131	503.662	9704.181	330.180	9693.074	403.018
FR	0.333	1.1E-16	0.333	1.1E-16	0.333	0.086
FSC	2.000	1.326	2.000	1.615	2.000	1.617
FSD	0.070	0.021	0.065	0.030	0.065	0.030
FTD	0.333	0.123	0.333	0.114	0.333	0.167
FTR	0.339	0.153	0.472	0.151	0.502	0.169
FVR	0.129	0.039	0.096	0.051	0.067	0.070
GSD	0.210	0.056	0.210	0.075	0.210	0.072
SC	22.000	14.782	34.000	15.972	34.000	16.432
SDS	7811.071	2733.442	9093.491	2380.049	9193.415	1761.721
SL	4181.564	2964.907	4662.694	2910.241	6358.970	2783.847
SS	0.297	0.233	0.349	0.233	0.412	0.240
STDFD	192.776	113.557	140.605	184.821	139.551	146.578
STDPD	0.105	0.041	0.127	0.043	0.127	0.045
STDSA	19.441	10.259	22.196	11.270	20.019	11.488
STDSD	182.959	147.577	202.317	187.809	242.371	165.151
TCT	13967.782	2012.417	12990.718	2365.847	12885.224	2788.422

Table E.15: List of metric medians and standard deviations on the `gpu` model during ABC tasks. The bold metrics correspond to the statistically significant median differences between metrics of different algorithms, according to Table E.11.

Metric	FL_{med}	FL_{std}	HL_{med}	HL_{std}	CHL_{med}	CHL_{std}
AFD	389.998	71.568	387.224	59.517	374.003	64.008
AFDR	0.333	0.086	0.333	0.062	0.333	0.156
AFDT	359.928	149.518	160.379	97.716	103.629	121.482
ALGD	363.378	28.805	384.372	25.596	399.118	41.591
AS	1.5E-04	9.1E-05	1.4E-04	1.1E-04	5.3E-05	8.7E-05
ASA	6.098	4.416	4.899	5.737	4.304	4.103
ASD	215.164	145.539	265.405	293.298	241.565	131.339
ASL	137.483	78.604	179.346	177.122	165.155	73.058
ASR	0.054	0.038	0.027	0.039	0.006	0.011
F2SR	0.452	0.253	0.479	0.444	0.400	0.241
FAS	1.0E-04	6.2E-05	9.8E-05	9.7E-05	0.0E+00	6.4E-05
FC	12.500	6.889	16.500	4.919	17.000	3.413
FDR	0.124	0.074	0.064	0.076	0.018	0.029
FDS	5212.941	2851.574	6339.914	2063.256	6767.376	1378.862
FFT	14439.660	718.970	10126.100	202.546	10176.057	577.723
FR	0.333	0.086	0.333	0.062	0.333	0.156
FSC	2.000	1.282	1.500	1.299	0.0E+00	0.958
FSD	0.090	0.041	0.090	0.030	0.110	0.032
FTD	0.444	0.208	0.333	0.087	0.111	0.110
FTR	0.263	0.112	0.371	0.131	0.364	0.105
FVR	0.111	0.071	0.057	0.077	0.020	0.026
GSD	0.210	0.104	0.265	0.094	0.290	0.065
SC	28.500	14.661	34.500	16.689	39.000	13.035
SDS	6983.890	3377.490	9265.597	2705.032	9187.873	304.584
SL	3459.500	3177.423	4581.602	2715.956	5576.107	1846.152
SS	0.158	0.162	0.254	0.166	0.350	0.138
STDFD	151.002	94.447	167.155	80.502	134.060	59.386
STDPD	0.110	0.046	0.107	0.053	0.105	0.049
STDSA	19.279	11.371	18.401	10.436	17.033	11.522
STDSD	210.714	138.525	208.102	427.406	201.322	132.797
TCT	19252.883	3479.262	15983.008	3317.863	16998.970	2800.162

Table E.16: List of metric medians and standard deviations on the `tractor` model during ABC tasks. The bold metrics correspond to the statistically significant median differences between metrics of different algorithms, according to Table E.10.

Metric	FL_{med}	FL_{std}	HL_{med}	HL_{std}	CHL_{med}	CHL_{std}
AFD	373.638	102.357	383.075	70.180	355.789	101.902
AFDR	0.333	1.1E-16	0.333	1.1E-16	0.333	1.1E-16
AFDT	221.138	92.437	235.356	113.992	212.502	136.893
ALGD	224.851	30.683	245.020	24.877	235.461	22.904
AS	1.4E-04	1.3E-04	2.5E-04	2.3E-04	2.1E-04	2.2E-04
ASA	9.315	6.574	6.602	5.223	5.012	3.623
ASD	243.765	480.316	236.512	152.921	221.153	238.777
ASL	99.732	55.954	140.587	51.404	111.838	61.812
ASR	0.025	0.049	0.050	0.040	0.040	0.061
F2SR	0.455	0.359	0.459	0.319	0.485	0.310
FAS	5.0E-05	3.9E-05	1.1E-04	1.0E-04	9.5E-05	9.4E-05
FC	16.000	7.120	16.000	4.944	19.000	6.995
FDR	0.086	0.093	0.094	0.055	0.094	0.084
FDS	6484.248	2759.040	6911.728	2041.533	6711.857	2436.115
FFT	13895.630	500.032	9904.052	250.196	9904.027	298.102
FR	0.333	1.1E-16	0.333	1.1E-16	0.333	1.1E-16
FSC	1.000	0.753	2.000	1.537	1.000	1.321
FSD	0.075	0.039	0.070	0.022	0.070	0.028
FTD	0.333	0.116	0.333	0.118	0.333	0.128
FTR	0.297	0.126	0.423	0.136	0.406	0.156
FVR	0.083	0.090	0.086	0.056	0.087	0.079
GSD	0.160	0.078	0.175	0.051	0.180	0.063
SC	30.000	17.748	29.500	14.929	32.500	18.028
SDS	9904.030	3417.802	9254.503	2262.834	9310.005	3072.460
SL	2753.145	2093.497	3800.885	1943.116	4231.157	2386.748
SS	0.140	0.100	0.264	0.125	0.236	0.160
STDFD	138.938	166.652	144.294	65.721	144.425	112.413
STDPD	0.102	0.053	0.115	0.053	0.120	0.065
STDSA	22.936	11.914	20.871	12.573	16.993	9.819
STDSD	203.383	277.826	193.227	173.760	180.430	216.442
TCT	19075.224	3319.534	14384.150	2170.699	14156.550	2915.079

Table E.17: List of metric medians and standard deviations on the `pocketwatch` model during ABC tasks. The bold metrics correspond to the statistically significant median differences between metrics of different algorithms, according to Table E.10.

Metric	FL_{med}	FL_{std}	HL_{med}	HL_{std}	CHL_{med}	CHL_{std}
AFD	931.278	329.893	848.798	317.154	959.498	539.697
AFDT	932.666	482.863	914.161	534.940	889.641	708.214
ALGD	50.521	17.046	58.463	22.060	59.979	68.583
AS	2.7E-04	7.8E-05	2.7E-04	1.3E-04	2.7E-04	1.5E-04
ASA	6.276	18.090	7.004	9.488	9.530	6.798
ASD	169.892	198.938	192.194	280.756	170.906	288.910
ASL	20.543	196.847	37.322	189.738	40.723	270.995
ASR	0.271	0.340	0.188	0.282	0.200	0.301
F2SR	0.220	0.321	0.200	0.417	0.203	0.255
FC	1.000	4.494	6.500	6.537	5.500	4.487
FDR	1.000	0.318	0.679	0.359	0.718	0.323
FDS	1104.767	4719.047	3991.605	5319.447	3911.685	4993.898
FFT	3652.945	280.203	3630.741	560.755	3672.006	419.546
FSD	0.010	0.019	0.030	0.027	0.025	0.022
FTR	0.292	0.229	0.572	0.215	0.537	0.240
FVR	1.000	0.332	0.487	0.374	0.633	0.335
GSD	0.050	0.040	0.050	0.037	0.060	0.036
SC	9.000	35.056	21.000	35.877	21.500	35.465
SDS	954.874	5961.486	4485.685	5909.473	5385.032	5535.718
SL	360.013	2581.026	934.484	1723.589	1975.090	8588.603
SS	0.042	0.158	0.065	0.108	0.153	0.492
STDFD	0.0E+00	409.622	364.753	321.491	312.323	372.677
STDPD	0.114	0.042	0.103	0.040	0.105	0.055
STDSA	17.448	14.164	19.422	14.274	25.103	12.745
STDSD	117.385	197.371	150.914	260.176	144.028	84.527

Table E.18: List of metric medians and standard deviations on the `eye` model during Tracking tasks. The bold metrics correspond to the statistically significant median differences between metrics of different algorithms, according to Table E.11.

Metric	FL_{med}	FL_{std}	HL_{med}	HL_{std}	CHL_{med}	CHL_{std}
AFD	842.234	363.693	761.184	744.213	773.891	714.099
AFDT	1210.247	891.005	877.149	812.367	993.735	937.432
ALGD	68.864	44.107	93.301	65.945	71.812	62.686
AS	2.6E-04	1.2E-04	1.7E-04	1.3E-04	2.6E-04	1.3E-04
ASA	7.331	4.396	5.694	9.713	7.391	4.333
ASD	210.960	129.796	263.803	249.988	219.775	141.715
ASL	31.770	49.925	58.662	200.509	37.987	112.832
ASR	0.153	0.191	0.054	0.148	0.052	0.075
F2SR	0.250	0.159	0.353	0.371	0.290	0.243
FC	13.000	7.132	9.000	8.028	12.000	6.190
FDR	0.554	0.274	0.373	0.318	0.498	0.287
FDS	8482.823	7368.827	4878.720	4405.166	9748.598	4399.358
FFT	3714.008	271.491	3708.457	329.299	3719.559	540.539
FSD	0.045	0.028	0.040	0.042	0.050	0.029
FTR	0.640	0.175	0.628	0.172	0.683	0.159
FVR	0.379	0.279	0.333	0.306	0.333	0.275
GSD	0.070	0.050	0.090	0.071	0.100	0.050
SC	44.000	44.304	23.000	20.701	38.500	31.728
SDS	14311.990	8193.222	7350.320	5452.712	11108.718	4897.336
SL	1806.445	2869.952	2433.185	2472.167	2921.275	2425.002
SS	0.093	0.114	0.170	0.235	0.165	0.203
STDFD	661.720	467.664	411.326	344.141	614.191	263.162
STDPD	0.128	0.043	0.112	0.045	0.119	0.046
STDSA	23.867	10.464	19.287	13.793	22.595	9.002
STDSD	171.538	121.733	234.659	297.714	171.715	161.112

Table E.19: List of metric medians and standard deviations on the `f1` model during Tracking tasks. The bold metrics correspond to the statistically significant median differences between metrics of different algorithms, according to Table E.11.

Metric	FL_{med}	FL_{std}	HL_{med}	HL_{std}	CHL_{med}	CHL_{std}
AFD	585.496	183.380	920.175	464.652	804.473	573.940
AFDT	535.727	251.228	960.424	595.586	916.013	803.803
ALGD	93.842	36.307	86.641	176.844	65.798	50.203
AS	3.2E-04	1.3E-04	2.8E-04	1.1E-04	2.9E-04	1.1E-04
ASA	1.486	6.907	3.879	18.858	10.057	10.995
ASD	203.466	150.355	171.893	149.050	169.699	136.786
ASL	60.075	144.164	28.013	188.378	33.938	103.380
ASR	0.264	0.208	0.363	0.339	0.212	0.292
F2SR	0.429	0.321	0.333	0.512	0.273	0.297
FC	6.000	6.192	3.500	5.794	4.000	5.974
FDR	0.573	0.295	0.931	0.348	0.785	0.304
FDS	3775.081	3725.006	2603.692	3695.341	3436.433	4339.476
FFT	3608.530	212.095	3586.325	369.077	3597.428	397.192
FSD	0.035	0.030	0.020	0.031	0.025	0.026
FTR	0.522	0.209	0.513	0.211	0.594	0.282
FVR	0.500	0.286	0.750	0.341	0.667	0.291
GSD	0.085	0.048	0.040	0.051	0.050	0.044
SC	13.500	18.830	13.000	24.640	12.000	30.737
SDS	3858.356	4242.811	2853.516	4112.511	3297.642	4606.738
SL	1449.002	1524.611	451.690	1639.523	810.448	4946.744
SS	0.145	0.178	0.060	0.151	0.082	0.376
STDFD	203.778	224.309	270.924	392.744	379.267	509.222
STDPD	0.092	0.041	0.090	0.033	0.104	0.033
STDSA	2.148	14.189	2.381	14.701	26.674	14.624
STDSD	128.284	142.781	123.097	101.263	136.200	167.179

Table E.20: List of metric medians and standard deviations on the gpu model during Tracking tasks. The bold metrics correspond to the statistically significant median differences between metrics of different algorithms, according to Table E.11.

Metric	FL_{med}	FL_{std}	HL_{med}	HL_{std}	CHL_{med}	CHL_{std}
AFD	669.228	363.492	600.958	379.748	637.687	237.434
AFDT	975.690	454.182	743.913	506.294	827.188	420.949
ALGD	137.745	33.382	130.529	170.798	114.701	46.445
AS	3.1E-04	1.3E-04	3.1E-04	1.4E-04	2.4E-04	1.2E-04
ASA	6.936	5.882	5.218	4.937	6.217	4.934
ASD	265.575	183.335	257.796	198.740	210.961	214.624
ASL	63.916	102.789	78.360	104.044	73.089	112.636
ASR	0.112	0.125	0.092	0.145	0.060	0.145
F2SR	0.313	0.209	0.364	0.341	0.322	0.315
FC	10.000	14.287	16.000	8.643	15.000	8.338
FDR	0.492	0.273	0.395	0.276	0.323	0.290
FDS	6067.881	9383.476	9426.583	4757.987	10203.812	5025.721
FFT	3786.177	254.317	3763.975	258.450	3730.668	1307.345
FSD	0.065	0.064	0.090	0.051	0.080	0.049
FTR	0.665	0.149	0.603	0.143	0.623	0.181
FVR	0.373	0.222	0.333	0.258	0.236	0.265
GSD	0.130	0.127	0.165	0.121	0.170	0.103
SC	34.500	54.101	34.000	29.743	42.000	30.691
SDS	6539.757	11092.762	11847.091	5508.535	13468.161	5291.714
SL	3510.086	3763.033	2531.057	3630.999	2813.576	3610.272
SS	0.205	0.160	0.226	0.214	0.229	0.225
STDFD	647.409	257.783	305.414	136.666	374.907	166.040
STDPD	0.130	0.065	0.110	0.044	0.102	0.040
STDSA	21.358	10.611	13.637	11.373	20.988	10.928
STDSD	224.992	217.949	218.909	222.276	177.651	143.362

Table E.21: List of metric medians and standard deviations on the `tractor` model during Tracking tasks. The bold metrics correspond to the statistically significant median differences between metrics of different algorithms, according to Table E.12.

Metric	FL_{med}	FL_{std}	HL_{med}	HL_{std}	CHL_{med}	CHL_{std}
AFD	1027.042	833.063	860.495	619.429	697.758	773.984
AFDT	1221.350	932.320	832.738	676.175	1025.192	755.519
ALGD	111.853	37.170	114.155	144.946	96.331	30.808
AS	2.7E-04	1.2E-04	2.7E-04	1.3E-04	2.7E-04	1.3E-04
ASA	8.157	6.372	5.000	7.535	5.658	6.206
ASD	205.871	144.025	232.401	167.003	190.462	214.906
ASL	21.663	88.520	35.511	179.878	30.773	276.665
ASR	0.489	0.358	0.413	0.331	0.543	0.285
F2SR	0.250	0.232	0.207	0.664	0.250	0.264
FC	3.000	12.115	3.000	7.172	1.000	6.285
FDR	0.887	0.309	0.703	0.344	1.000	0.274
FDS	3436.426	7931.409	3441.978	4919.819	2370.527	3983.885
FFT	3747.329	251.806	3730.664	219.500	3758.417	384.412
FSD	0.020	0.049	0.020	0.039	0.010	0.029
FTR	0.694	0.239	0.564	0.221	0.522	0.220
FVR	0.633	0.339	0.667	0.341	1.000	0.314
GSD	0.040	0.080	0.045	0.083	0.040	0.064
SC	20.500	47.875	23.000	28.515	13.000	19.662
SDS	3131.087	8704.672	3541.917	5443.854	2209.533	4563.342
SL	584.408	3242.376	454.547	2335.679	558.323	1487.325
SS	0.071	0.159	0.061	0.164	0.068	0.137
STDFD	332.298	500.107	275.399	326.937	0.0E+00	258.554
STDPD	0.112	0.047	0.108	0.051	0.118	0.044
STDSA	23.277	13.133	18.191	14.418	19.909	13.319
STDSD	161.056	128.955	142.823	152.491	122.704	122.199

Table E.22: List of metric medians and standard deviations on the `pocketwatch` model during Tracking tasks. The bold metrics correspond to the statistically significant median differences between metrics of different algorithms, according to Table E.12.

Bibliography

[1] A. Gemsa, B. Niedermann, M. Nöllenburg, Trajectory-based dynamic map labeling, in: International Symposium on Algorithms and Computation, Springer, 2013, pp. 413–423.

[2] R. L. Rabello, G. R. Mauri, G. M. Ribeiro, L. A. N. Lorena, A clustering search metaheuristic for the point-feature cartographic label placement problem, European Journal of Operational Research 234 (3) (2014) 802–808.

[3] D. Schmalstieg, T. Hollerer, Augmented Reality: Principles and Practice, Usability, Pearson Education, 2016.
URL `https://books.google.si/books?id=qPU2DAAAQBAJ`

[4] T. Götzelmann, K. Ali, K. Hartmann, T. Strothotte, Form follows function: Aesthetic interactive labels., Computational aesthetics 5.

[5] E. Ferguson, Engineering and the Mind's Eye, Engineering/history of science and technology, MIT Press, 1992.
URL `https://books.google.si/books?id=WcqaKE_Eg1IC`

[6] M. Agrawala, D. Phan, J. Heiser, J. Haymaker, J. Klingner, P. Hanrahan, B. Tversky, Designing effective step-by-step assembly instructions, in: ACM Transactions on Graphics (TOG), Vol. 22, ACM, 2003, pp. 828–837.

[7] B. Kerbl, D. Kalkofen, M. Steinberger, D. Schmalstieg, Interactive disassembly planning for complex objects, Computer Graphics Forum 34 (2)

(2015) 287–297. doi:10.1111/cgf.12560.
URL http://dx.doi.org/10.1111/cgf.12560

[8] M. Tatzgern, D. Kalkofen, R. Grasset, D. Schmalstieg, Hedgehog labeling: View management techniques for external labels in 3d space, in: Virtual Reality (VR), 2014 iEEE, IEEE, 2014, pp. 27–32.

[9] K. Hartmann, K. Ali, T. Strothotte, Floating labels: Applying dynamic potential fields for label layout, in: International Symposium on Smart Graphics, Springer, 2004, pp. 101–113.

[10] J. B. Madsen, M. Tatzqern, C. B. Madsen, D. Schmalstieg, D. Kalkofen, Temporal coherence strategies for augmented reality labeling, IEEE transactions on visualization and computer graphics 22 (4) (2016) 1415–1423.

[11] A. Dünser, R. Grasset, M. Billinghurst, A survey of evaluation techniques used in augmented reality studies, Human Interface Technology Laboratory New Zealand, 2008.

[12] R. Azuma, C. Furmanski, Evaluating label placement for augmented reality view management, in: Proceedings of the 2nd IEEE/ACM international Symposium on Mixed and Augmented Reality, IEEE Computer Society, 2003, p. 66.

[13] W. Dong, H. Liao, F. Xu, Z. Liu, S. Zhang, Using eye tracking to evaluate the usability of animated maps, Science China Earth Sciences 57 (3) (2014) 512–522.

[14] L. Herman, S. Popelka, V. Hejlova, Eye-tracking analysis of interactive 3d geovisualization, J. Eye Mov. Res 10 (2).

[15] K. Kurzhals, M. Burch, T. Pfeiffer, D. Weiskopf, Eye tracking in computer-based visualization, Computing in Science & Engineering 17 (5) (2015) 64–71.

[16] J. R. Bergstrom, A. Schall, Eye tracking in user experience design, Elsevier, 2014.

[17] A. T. Duchowski, Eye tracking methodology, Theory and practice 328.

[18] R. J. Jacob, K. S. Karn, Eye tracking in human-computer interaction and usability research: Ready to deliver the promises, in: The mind's eye, Elsevier, 2003, pp. 573–605.

[19] T. Blascheck, K. Kurzhals, M. Raschke, M. Burch, D. Weiskopf, T. Ertl, State-of-the-art of visualization for eye tracking data, in: Proceedings of EuroVis, Vol. 2014, 2014.

[20] Z. Sharafi, T. Shaffer, B. Sharif, Y.-G. Guéhéneuc, Eye-tracking metrics in software engineering, in: Software Engineering Conference (APSEC), 2015 Asia-Pacific, IEEE, 2015, pp. 96–103.

[21] W. Li, M. Agrawala, B. Curless, D. Salesin, Automated generation of interactive 3d exploded view diagrams, in: ACM SIGGRAPH 2008 Papers, SIGGRAPH '08, ACM, New York, NY, USA, 2008, pp. 101:1–101:7. doi:10.1145/1399504.1360700.
URL http://doi.acm.org/10.1145/1399504.1360700

[22] M. Vaaraniemi, M. Freidank, R. Westermann, Enhancing the visibility of labels in 3d navigation maps, in: Progress and new trends in 3D geoinformation sciences, Springer, 2013, pp. 23–40.

[23] Wikipedia, Exploded-view drawing, https://en.wikipedia.org/wiki/Exploded-view_drawing, [Online; accessed 18-September-2018] (September 2018).

[24] J. Jankowski, S. Decker, On the design of a dual-mode user interface for accessing 3d content on the world wide web, International Journal of Human-Computer Studies 71 (7) (2013) 838–857.

[25] K. Ali, K. Hartmann, T. Strothotte, Label layout for interactive 3d illustrations, The Journal of WSCG 13.

[26] M. Tatzgern, D. Kalkofen, D. Schmalstieg, Dynamic compact visualizations for augmented reality, in: Virtual Reality (VR), 2013 IEEE, IEEE, 2013, pp. 3–6.

[27] K. Hallqvist, Dynamic label placement for moving objects (2017).

[28] A. Leykin, M. Tuceryan, Automatic determination of text readability over textured backgrounds for augmented reality systems, in: Proceedings of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality, IEEE Computer Society, 2004, pp. 224–230.

[29] A. Brychtova, A. Coltekin, An empirical user study for measuring the influence of colour distance and font size in map reading using eye tracking, The cartographic journal 53 (3) (2016) 202–212.

[30] S. Djamasbi, Eye tracking and web experience, AIS Transactions on Human-Computer Interaction 6 (2) (2014) 37–54.

[31] B. Fu, N. F. Noy, M.-A. Storey, Eye tracking the user experience–an evaluation of ontology visualization techniques, Semantic Web 8 (1) (2017) 23–41.

[32] K. Kurzhals, D. Weiskopf, Space-time visual analytics of eye-tracking data for dynamic stimuli, IEEE Transactions on Visualization and Computer Graphics 19 (12) (2013) 2129–2138.

[33] H.-Y. Ho, I.-C. Yeh, Y.-C. Lai, W.-C. Lin, F.-Y. Cherng, Evaluating 2d flow visualization using eye tracking, in: Computer Graphics Forum, Vol. 34, Wiley Online Library, 2015, pp. 501–510.

[34] N. M. Moacdieh, J. C. Prinet, N. B. Sarter, Effects of modern primary flight display clutter: Evidence from performance and eye tracking data, in: Proceedings of the Human Factors and Ergonomics Society annual

meeting, Vol. 57, SAGE Publications Sage CA: Los Angeles, CA, 2013, pp. 11–15.

[35] A. K. Jain, Data clustering: 50 years beyond k-means, Pattern recognition letters 31 (8) (2010) 651–666.

[36] D. H. Eberly, 3D game engine design: a practical approach to real-time computer graphics, CRC Press, 2006.

[37] J. Steinruecken, L. Pluemer, A web service to personalise map colouring, in: International Cartography Conference, 2009.

[38] L. B. Boudaoud, B. Solaiman, A. Tari, A modified zs thinning algorithm by a hybrid approach, The Visual Computer (2017) 1–18.

[39] L. B. Boudaoud, B. Solaiman, A. Tari, Implementation and comparison of binary thinning algorithms on gpu, Computing (2018) 1–27.

[40] J. M. Weiss, C. Karlsson, Efficient grayscale thinning on parallel hardware, Journal of Computational Methods in Sciences and Engineering 17 (S1) (2017) S61–S70.

[41] W. Yang, Q. Jia, H. Liu, Y. Wu, H. Guo, An improved gpgpu-accelerated parallelization for rotation invariant thinning algorithm, in: Image Processing (ICIP), 2016 IEEE International Conference on, IEEE, 2016, pp. 1784–1788.

[42] W. Chang, Q. Xin, Y. XiaoPing, Y. Yi, 3d parallel thinning algorithm based on topological invariance and its applications, Beijing Biomedical Engineering 4 (2015) 004.

[43] T. Zhang, C. Y. Suen, A fast parallel algorithm for thinning digital patterns, Communications of the ACM 27 (3) (1984) 236–239.

[44] G. Burger, J. Guna, M. Pogačnik, Suitability of inexpensive eye-tracking device for user experience evaluations, Sensors 18 (6) (2018) 1822.

[45] J. Sauro, 10 things to know about the single ease question (seq), https://measuringu.com/seq10/, [Online; accessed 6-September-2018] (October 2012).

[46] S. G. Hart, L. E. Staveland, Development of nasa-tlx (task load index): Results of empirical and theoretical research, in: Advances in psychology, Vol. 52, Elsevier, 1988, pp. 139–183.

[47] S. G. Hart, Nasa-task load index (nasa-tlx); 20 years later, in: Proceedings of the human factors and ergonomics society annual meeting, Vol. 50, Sage Publications Sage CA: Los Angeles, CA, 2006, pp. 904–908.

[48] R. J. Pagulayan, K. R. Steury, B. Fulton, R. L. Romero, Designing for fun: User-testing case studies, in: Funology 2, Springer, 2018, pp. 419–433.

[49] R. M. Bethea, Statistical methods for engineers and scientists, Routledge, 2018.

[50] P. E. Black, Fisher-yates shuffle, Dictionary of algorithms and data structures 19.

[51] R. B. d'Agostino, An omnibus test of normality for moderate and large size samples, Biometrika 58 (2) (1971) 341–348.

[52] Tobii Pro, Coordinate systems, http://developer.tobiipro.com/commonconcepts/coordinatesystems.html, [Online; accessed 16-September-2018] (2018).