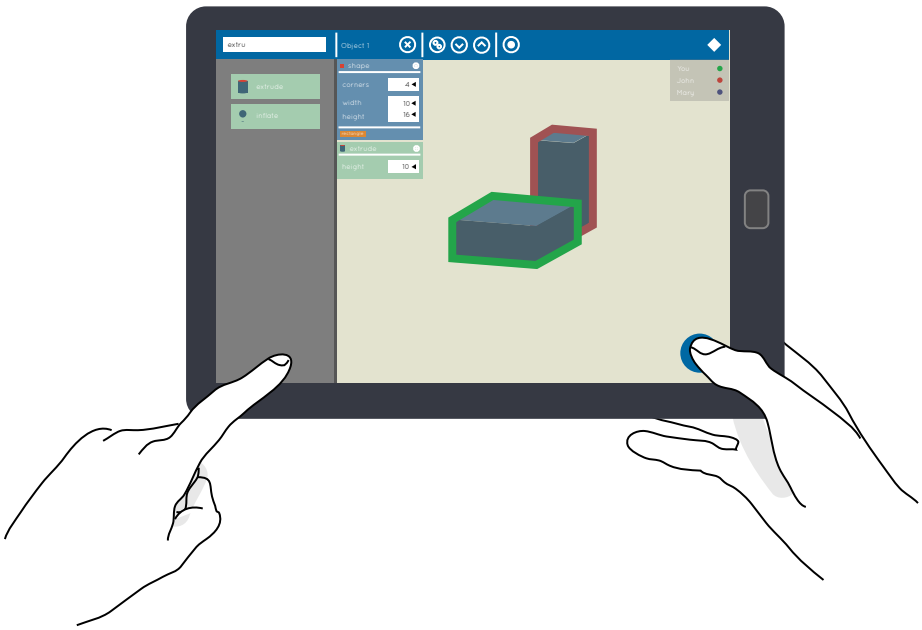
Het ontwerp van een collaboratieve en multimodale virtuele ontwerpomgeving



Marcel Goethals
Rawshaping Technology
Universiteit Twente Industrieel Ontwerpen

2

Het ontwerp van een collaboratieve en multimodale virtuele ontwerpomgeving

Student	M.G.J Goethals
Studentnummer	s0193437
Opleiding	Industrieel Ontwerpen
	Rawhaping Technology
Eerste examinator	ING R.E. Wendrich
Tweede examinator	Dr.ir. H.J.M. Geijselaers
Voorzitter examencommissie	Prof. Dr. Ir. A.O. Eger
UT-Begeleider	ING R.E. Wendrich

Voorwoord

Voor u ligt het resultaat van mijn Bacheloropdracht. Een opdracht die ik voor Rawshaping uitvoerde om Robert Wendrich verder te helpen bij het ontwikkelen van vernieuwende ontwerptools. In de opdracht werd onderzoek gedaan naar de mogelijkheid voor collaboratieve en multimodale ontwerpsoftware. Een erg intressante opdracht, omdat het mijn intresse in de invloed van systemen en data structuren op de resultaten van het ontwerpproces combineert met het zoeken naar een nuttige gebruikers interactie.

Voor de totstandkoming van deze opdracht wil ik graag Robert Wendrich bedanken. Daarnaast wil ik graag iedereen bedanken die me geholpen heeft door met me te praten en discussies te voeren over mijn resultaten, waardoor ik op momenten dat ik vastliep weer nieuwe inzichten kon krijgen.

Samenvatting

Dit verslag beschrijft een onderzoek naar de mogelijkheden van een nieuwe ontwerpomgeving. Het doel van de ontwerpomgeving is om een platform te bieden om collaboratief en multimodaal te werk te kunnen gaan. Dat wil zeggen dat de ontwerpomgeving met meerdere mensen tegelijkertijd en op meerdere manieren te gebruiken is. De ontwerpomgeving zal een software pakket zijn. De opdracht maakt deel uit van Rawshaping Technology aan de Universiteit Twente.

Eerst zal er een literatuur analyse omscherven worden waarbij gezocht wordt naar een theoretische basis waarop de omgeving ontworpen kan worden. Vervolgens worden een aantal ideeën en tests omschreven die gedaan werden op basis van de analyse. Zo werd er een natuurlijke taal interface ontwikkeld en gezocht naar een gegeneraliseerde oplossing voor ontwerptools.

Daarna wordt een concept ontwerp voor de omgeving omscherven. Er werd een interface ontwikkeld en een systeem architectuur bedacht. Vervolgens Wordt het Implementeren van een prototype omschreven. Aan de hand van het prototype wordt duidelijk dat het ontwerp haalbaar is.

Naar aanleiding van het onderzoek en het prototype worden aanbevelingen gedaan over het verder ontwikkelen van de ontwerpomgeving

Abstract

This paper describes the research on the possibility for a new designenvironment. The purpose of this environment is to provide a platform for collaborative and multimodal interaction. This means the designenvironment can be used by many people at the same time, and that there are many different ways to use it. The environment wil be a software package. The assignment is part of Rawshaping Technology at the University of Twente.

Firstly there will be an analysis of current literature where a theoretical framework is developed upon which the environment can be designed. Next a few ideas and tests are described that were done on the basis of the analysis. A Natural language interface was developed and a generalised approach to design tools was found.

A concept for the design environment will be described. A user interface and a system architecture were developed. Using this design, a prototype was implemented to verify that the design was feasible.

As a result of the research and the prototype recommendations are made about further devlopment of the environment.

Inhoud

Inleiding

In deze Bacheloropracht werd onderzoek gedaan naar een collaboratieve en multimodale ontwerpomgeving voor Rawshaping Technology(RST).

Door Rawshaping wordt onderzocht en gezocht binnen het hybride spanningsveld tussen digitaal/abstract en analoog/haptische ontwerppraktijken. Dit gebeurt onder andere door het ontwikkelen van een ‘Loosely Fitted Design
Synthesizer’(LFDS), een ontwerpomgeving die intuïtie, creativiteit en verbeelding stimuleert. Deze ontwerpomgeving wil RST verder ontwikkelen en in het domein van de driedimensionale ruimte brengen.

Afgelopen jaar is onderzoek gedaan naar volumetrische modeleringstechnieken. Uit dit onderzoek is een eenvoudig concept
voor een ‘3d Intuitive Voxel Shaper’(3d IVS) gekomen. Deze softwaretool stelt de ontwerper in staat om ruimtelijke volumetrische schetsen te maken in een virtuele omgeving. RST wil nu graag de opgedane kennis van de 3d IVS toepassen in een toegankelijke, LFDS gelijkende ontwerpomgeving.

Het doel van de opdracht is om te onderzoeken hoe de 3d software gecombineerd kan worden met tactiele ontwerptools in een intuïtieve, congruente, collaboratieve ontwerpomgeving. De ontwerpomgeving moet op zo’n manier ontworpen worden dat het multi-modale gebruikersinteractie ondersteunt. Dat betekent dat er naast een aantal integrale ontwerptools ook gemakkelijk nieuwe (tactiele)ontwerptools en shaping methodes kunnen worden ‘ingeplugd’. Hierbij is het belangerijk dat het systeem aan de ene kant sterk gefocust is op usability en user-interaction en aan de andere kant zoveel mogelijk open laat over hoe de gebruiker het systeem zal/kan
gebruiken.

De opdracht werd uitgevoerd in vier stappen: Analyse, Generatie, Synthese en Implementatie. In de Analyse fase werd een literatuur onderzoek gedaan naar de mogelijkheden van een dergelijk systeem. In de Generatie fase werden ideeën gegenereerd door testen en experimenten uit te voeren. In de Synthese fase werden de resultaten gecombineerd tot een ontwerp van de interface en een systeem architectuur. In de Implementatie fase werd een werkend prototype van de omgeving gemaakt.

Analyse

In de analyse fase werd gekeken naar literatuur en naar bestaande softwarepakketten. Dit om een inzicht te krijgen in het ontwerpproces en om mogelijke zoekvelden te ontdekken.

CAD omgevingen

CAD(Computer Aided Design) software wordt in de ontwerppraktijk steeds vaker en overheersender toegepast. Het gebruik van deze software is echter vaak complex: ze hebben een steile 'leercurve' en zijn meestal gericht op exacte (mathematische) representaties. Wanneer deze rigide tools moeten worden ingezet tijdens het fluïde creatieve proces ontstaat er wrijving. De voordelen van het gebruik van CAD software boven traditionele methoden en technieken verdwijnen wanneer ze op dergelijke wijze worden gebruikt. Ze werken soms zelfs contraproductief. Zoals er wordt geschreven door Wendrich: "[The use of CAD tools] ... often leads to frustration, time consumption, problem reduction and mediocrity in finding the 'right' solution in problem solving. This is not to speculate that CAD has no added or intrinsic value or meaning, on the contrary CAD created a virtual world experience next to the real physical world in which we can create, simulate and visualize endlessly to some extent infinitly" [1]. Ook Sener et al uit kritiek wanneer zij schrijft over CAD tools: "The general concern in literature is that the creatively intense early phase of industrial design, where the form of a product is in a conceptual and fluid state, is very poorly supported" [2].

Veel CAD software maakt gebruik van rigide numerieke input parameters om ruimtelijke vormen te definiëren. Een ultra-rationele werkwijze die niet goed aansluit bij de intuïtieve, (subcognitieve) praktijk van creatief product ontwerp. We kunnen concluderen dat er behoefte is aan een CAD tool die het mogelijk maakt om in een vroeg, conceptueel stadium ruimtelijke ideeën te ontdekken en te onderzoeken door op schetsmatige, fluïde wijze te werk te kunnen gaan. Wendrich: "We need to be able to express our thoughts, initial ideas, fuzzy notions and dreams to become manifest and convey to others in a spontaneous and intuitive way"

Ook ontwerpers zelf geven aan dat er een dergelijke behoefte bestaat. Uit interviews die door RST zijn uitgevoerd onder (student)ontwerpers blijkt dat zij het gebruik van CAD en Virtual Reality in ontwerp alleen als nuttig ervaren wanneer:

① De tool zorgt voor meer inzicht en begrip.

② De tool een lage leercurve heeft.

③ De tool de snelheid verhoogt waarmee de ontwerpruimte verkend kan worden.

④ De tool zowel visuele als tactiele representaties impliceert.

⑤ De tool eenvoudige conceptualisatie en ideatie teweegbrengt.

⑥ De tool de mogelijkheid tot simulatie en prototyping ondersteunt.

⑦ De tool intuïtieve interactie ondersteunt.

⑧ De tool nadruk legt op specifieke vaardigheden.

De non lineariteit van het ontwerp proces

Verschillende soorten taken kennen een verschillende structuur. Volgens Norman [3] kunnen die structuren breed of diep zijn. Het kiezen van een gerecht uit een menu is een voorbeeld van een brede structuur: Er zijn veel verschillende opties, maar als de keuze eenmaal gemaakt is, valt er weinig meer te doen. Het volgen van een recept heeft een diepe structuur: er zijn veel stappen die uitgevoerd moeten worden, maar iedere stap leid maar tot één of twee opvolgende stappen . Zolang een taak alléén een diepe of een brede structuur heeft, is hij relatief eenvoudig om uit te voeren.

Ontwerpen is een taak die zowel een brede als een diepe structuur heeft. Er zijn veel keuzes die gemaakt moeten worden, en iedere stap die gemaakt wordt leidt tot een veelvoud aan nieuwe keuzes die gemaakt moeten worden.

Hierdoor is ontwerpen geen lineair proces, het vloeit niet voort in elegante, nette stappen. Het is chaotisch, het springt voor zichzelf uit, en zet stappen terug. Ontwerpen gebeurt niet puur en alleen door het toepassen van rationaliteit of logica. Mentaal springt de ontwerper van idee naar idee, dingen die niets met elkaar te maken hebben worden samengebracht om tot nieuwe inzichten en concepten te komen. Er worden fouten gemaakt, maar fouten kunnen ook tot nieuwe oplossingen leiden.

Dit is de enige manier waarop een complex proces zoals ontwerpen doorlopen kan worden. Het is voor een ontwerper onmogelijk om mentaal alle verschillende combinaties en permutaties 'door te rekenen', onmogelijk om puur en alleen op redeneringskracht te voorzien wat de beste keuze is. De enige manier om dit te doen is om een keuze te maken, die uit te proberen, en te observeren wat de invloed is van die keuze op het ontwerp. Het is belangrijk dat een ontwerp omgeving deze modus van opereren ondersteunt, en de ontwerper dus niet in een strikt lineair keurslijf dwingt.

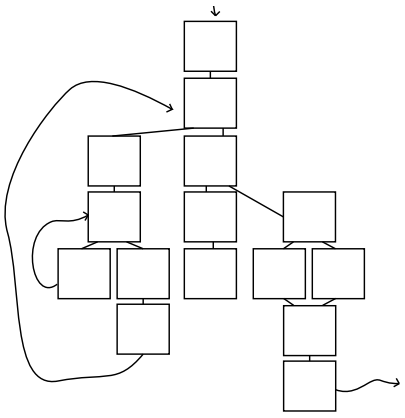


fig.1 rhizome structuur

De comlexe aard van het ontwerpproces kan mooi gevangen worden met het concept van het rhizoom. Het Rhizoom wordt door Deleuze en Guattari gebruikt om theorie en onderzoek te omschrijven die meervoudige en non-hierarchische in- en uitgangspunten in data representatie en interpretatatie heeft [4]. "Rhizomes oppose the idea that knowledge must grow in a tree structure from previously accepted ideas.". Het ontwerpproces is in het kader van het rhizoom te beschouwen als volgt:

Een ontwerp groeit niet voort als een boom structuur uit eerdere iteraties, maar is eerder een meervoudig en non-hierarchisch proces, waarbij er meerdere in- en uitgangspunten voor data en interpretatie bestaan.

Door dit concept toe te passen in de context van een ontwerpomgeving wordt een aantal eisen duidelijk die we aan de ontwerpomgeving kunnen stellen:

① Er moet de mogelijkheid zijn om op verschillende punten data in te voeren, te wijzigen, te combineren en te recombineren.

② Het moet mogelijk zijn data op verschillende manieren te bekijken.

③ Het moet mogelijk zijn om data uit zijn context te halen en in juxtapositie met elkaar te brengen, om zo tot nieuwe ideeën te komen.

Flow en iteratie

Het doorlopen van het ontwerpproces is een menselijke activiteit, en is dus onderhevig aan menselijk gedrag en psychische factoren. Soms is duidelijk wat van de ontwerper verwacht wordt, maar vaker zijn de inkaderingen en eisen grillig en onduidelijk. De ontwerper moet in een groot en onduidelijk zoekveld tot een eenduidige oplossing komen. Daarom is het belangrijk iteratief te werken. Iedere iteratie die de ontwerper maakt is onder te verdelen in drie stappen:

① De ontwerper heeft een idee of notie van hoe een ontwerp eruit zou moeten zien.

② Door dit idee op een bepaalde manier in de wereld te manifesteren kan de ontwerper zijn idee testen.

③ De ontwerper observeert hoe zijn initiële notie van de daadwerkelijke manifestatie verschilt, en zal hierdoor zijn idee over het ontwerp moeten aanpassen, waardoor een ontwerpstap wordt gemaakt. Dit is dus een cyclisch proces.

De stap waar ontwerpsoftware relevant is, is tijdens de manifestatie fase. Het manifesteren van ideeën in een virtuele omgeving geeft de ontwerper een potentieel krachtige manier om snel en goedkoop (zonder hoge materiele kosten) te kunnen itereren.

Zoals iedere menselijke bezigheid kan het ontwerpen soms tegenzitten. Hoe goed het ontwerpen 'lukt' heeft te maken met iets wat je 'flow' [5] zou kunnen noemen. Als het goed gaat, dan zit je in een flow. Je bent je niet meer bewust van de tijd of je omgeving. Je bent volledig geconcentreerd op het ontwerp, en je niet eens per se bewust van je directe handelingen, of de tools waarmee je werkt. Als het plotseling tegen zit, dan wordt de flow verbroken. Dat kan bijvoorbeeld doordat de tool niet meewerkt of omdat er een stap gemaakt moet worden die te moeilijk of ingewikkeld is. Dat resulteert in frustratie. Een goede ontwerpomgeving kan bijdragen aan het ontstaan van flow door een zorgvuldige afstemming op de gebruiker en interactie. Om dit te bereiken zou je de bediening intuïtief en 'non-blocking' kunnen maken. Daarbij zou het induceren van een gevoel van Aanwezigheid ('Presence') of Immersie hieraan kunnen bijdragen.

Aanwezigheid, immersie en continuiteit

De mate waarin een gebruiker een gevoel van aanwezigheid in een virtuele omgeving heeft, geeft aan in hoeverre de gebruiker zich 'in' de virtuele wereld voelt. Dat gevoel kan je opwekken door 3D brillen, indrukwekkende grafische weergaven, taktiele handschoenen of allerlei andere randapparatuur. Maar naast zintuigelijke prikkels, is een gevoel van aanwezigheid ook in hoge mate afhankelijk van de hoeveelheid emotionele investering door de gebruiker. Een gebruiker voelt zich meer en natuurlijker betrokken bij een omgeving waar hij op een intuïtief niveau dichter bij staat [6]. Als de gebruiker zich niet betrokken voelt bij het proces gaat het vervelen. Ook kan immersie doorbroken worden wanneer er een breuk in de continuïteit van de wereld plaatsvind. De virtuele wereld voldoet aan bepaalde regels, en wanneer er plotseling iets gebeurt wat 'niet klopt' ervaart de gebruiker dit als negatief.

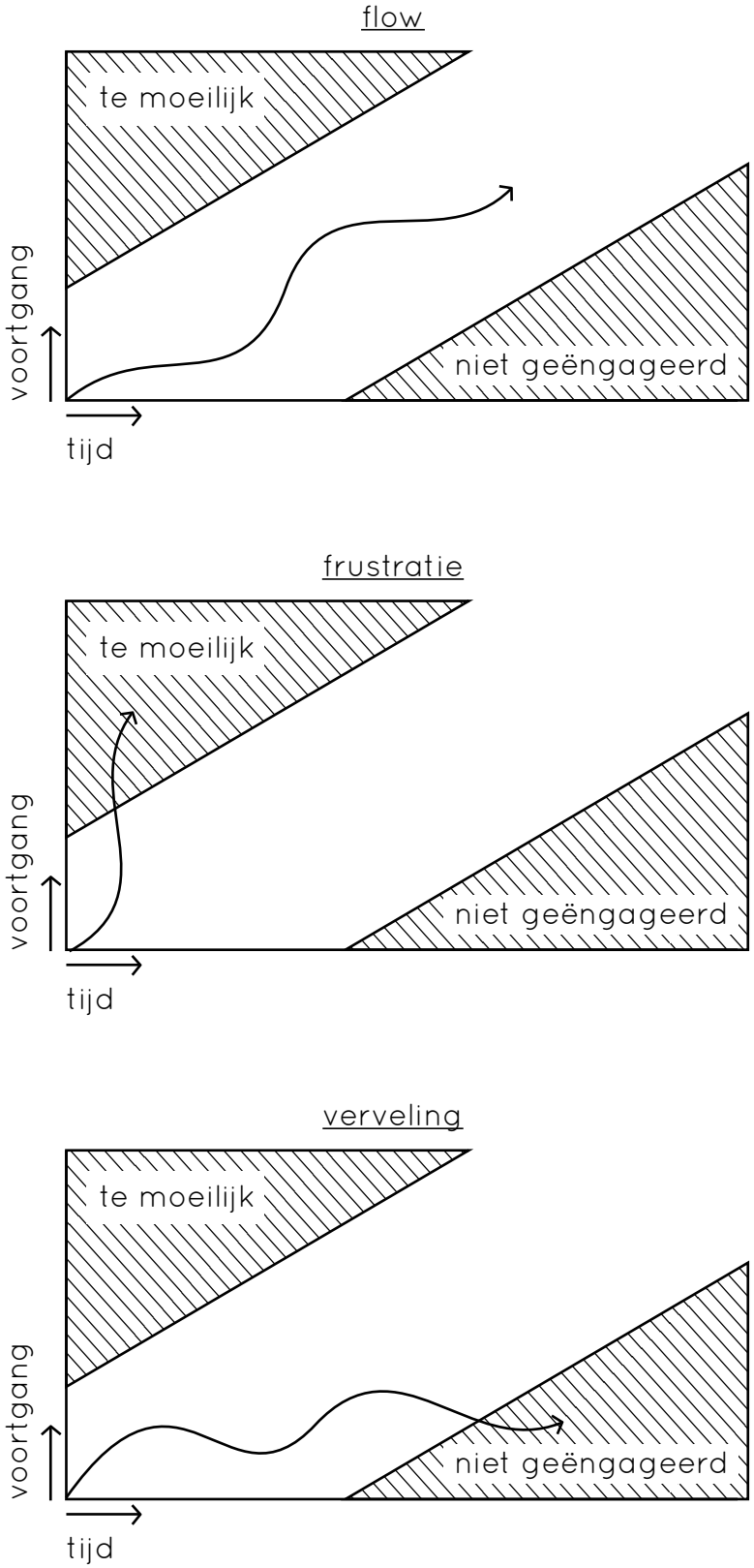


fig.2 flow in het ontwerp proces

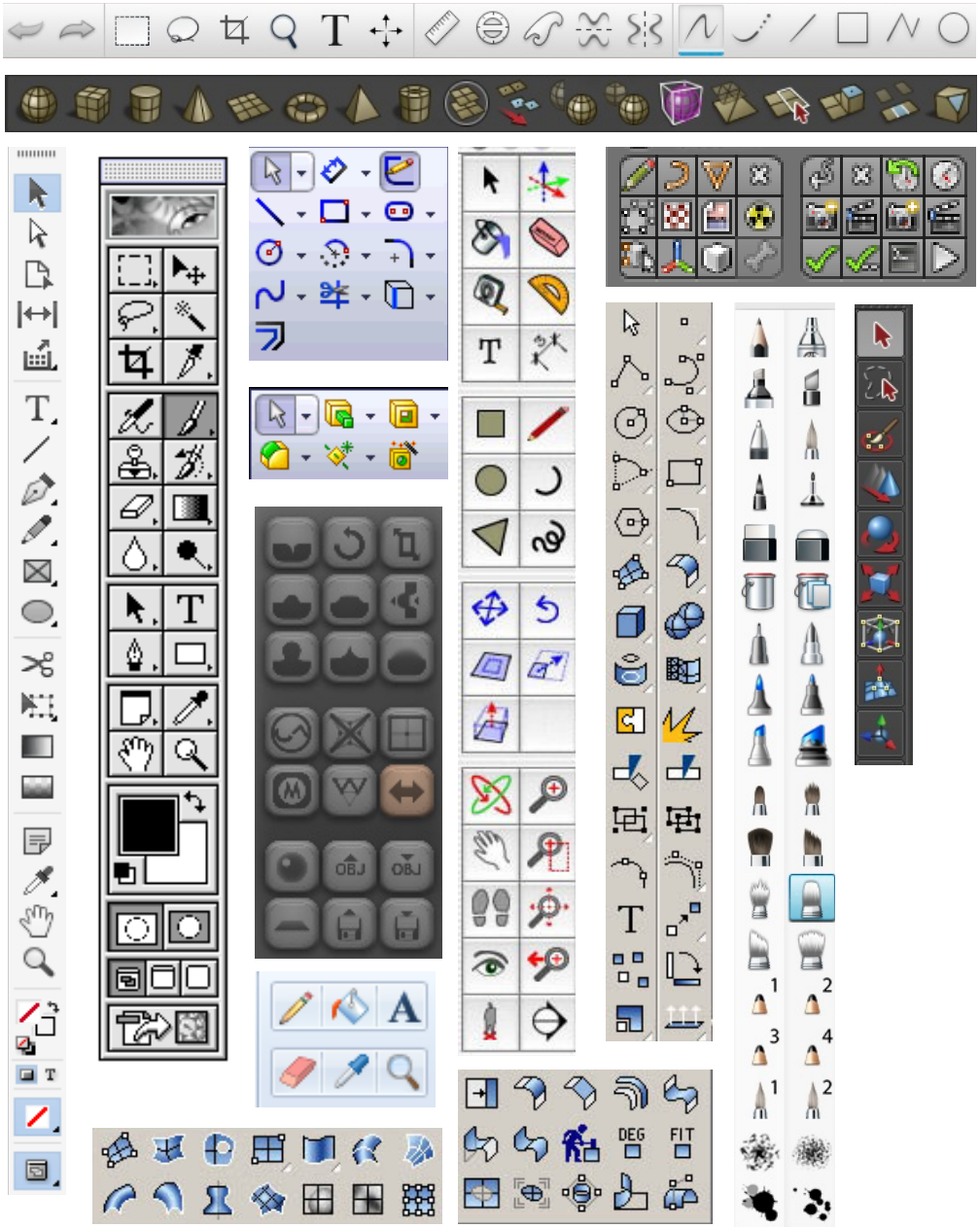


fig.3 toolsets uit verschillende ontwerp omgevingen, waaronder solidwoks, maya, photoshop, illustrator & blender 3d

Tools en Toolsets
Ontwerpers kunnen bij het ontwerpen vele verschillende gereedschappen en materialen gebruiken. Pen, stift, schaar, papier, karton, schuim, mes, schuurpapier, klei, hout of zaag. Ieder met hun eigen eigenschappen en functies.

Een ontwerpomgeving bevat potentieel een enorme hoeveelheid functies en tools. Om een functie te kunnen vinden moet die voor de gebruiker zichtbaar zijn. Iedere functie moet ook een duidelijke eenduidige mapping naar een bedieningselement hebben. Als er heel veel functies zijn, ontstaan er dus ook veel bedieningselementen. Maar niet alle functies kunnen op ieder moment op ieder object uitgevoerd worden. Hoe geeft de software aan welke handelingen wel of niet mogelijk zijn? Wanneer er een grote hoeveelheid bedieningselementen zichtbaar zijn wordt het voor de gebruiker complex om overzicht te houden over alle functies. Een oplossing hiervoor is modularisatie. Door de elementen op te delen in modules wordt de complexiteit verminderd. Modularisatie reduceert echter ook de zichtbaarheid. Er moet een balans gevonden worden tussen modulariteit en zichtbaarheid. Een manier om tools te modulariseren is door ze in toolsets op te delen.

Ontwerp-softwarepaketten gebruiken deze tools vaak als metafoor, door ze te presenteren in een 'toolset': een virtuele gereedschapskist. De virtuele gereedschapskist heeft een statische indeling en de gereedschappen die er in liggen staan vast. De analogie werkt dus maar tot op zekere hoogte: een 'echte' gereedschapskist is eigenlijk een dynamisch geheel. Er worden gereedschappen aan toegevoegd en uitgehaald. Soms is er maar één soort van een bepaald gereedschap, of worden er meerdere variaties van het zelfde gereedschap gebruikt. Het is eigenlijk vreemd dat virtuele gereedschapskisten zo statisch zijn. Het feit dat ze digitaal zijn maakt ze juist geschikter voor een dynamische indeling.

Een nadeel van een dynamische toolset is dat ze snel verworden tot chaos. Als je een specifiek gereedschap nodig hebt moet je graven en zoeken in de kist, en dat kan lang duren. Statische toolsets lossen dat probleem op door gebruik te maken van conventie: iedere tool heeft een vaste plaats en is daardoor gemakkelijk terug te vinden. Dit lost het probleem echter slechts ten dele op. Ieder softwarepakket heeft zijn eigen conventies, waardoor in de onderlinge pakketen de vergelijkbare tools verschillend geordend zijn. Zoiets zou in een dynamische toolset alleen opgelost kunnen worden door de gebruiker zelf. De gebruiker moet dan zelf conventies ontwikkelen en de tools op een voor hem natuurlijke manier rangschikken. Dit vergt echter een zekere mate van zelfdiscipline: De gebruiker moet af en toe zijn toolset opruimen en rangschikken.

Tools en interactie
De manier waarop tools verzameld en gerangschikt worden is slechts een deel van het verhaal. Hoe de gebruiker vervolgens een tool gebruikt in de virtuele omgeving is een veel complexer probleem. Er moet een handeling in de echte wereld vertaald worden naar een handeling in de virtuele ruimte. Er kunnen metaforen gebruikt worden: tekenen, knippen, plakken, maar niet iedere virtuele handeling heeft een voor de hand liggende 'echte' evenknie. De verschillende soorten handelingen die de gebruiker moet uitvoeren zijn niet altijd voor de hand liggend. Handelingen moeten geleerd worden of het gebruik moet worden uitgelegd. Er kan een beroep worden gedaan op conventies, of gebruik worden gemaakt van visuele representaties en artefacten zoals selectievakjes, of 'handles', die de gebruikes hints geven over wat er moet gebeuren.

Presence door tools

Tools zijn de voornaamste manier waarop een gebruiker aanwezig is in een virtuele wereld. Het zijn de 'handen' waarmee de gebruiker een invloed kan uitoefenen op deze wereld. De muis en de cursor worden 'onzichtbaar' voor de gebruiker, de aanwezigheid van de gebruiker wordt als het ware door deze twee elementen gemedieerd. Hoe deze tools worden vormgegeven hebben in grote mate invloed op hoe de gebruiker aanwezig is in de virtuele wereld. Oftewel: de manier waarop de cursor of 'avatar' in de virtuele wereld 'aanwezig' is heeft een directe invloed op de manier waarop de gebruiker, de bestuurder van deze 'avatar', in de virtuele wereld aanwezig is.

Een computer kan op twee manieren bediend worden. [1] De computer kan gecommandeerd worden: door hem een instructie of opdracht te geven, of hij kan bediend worden door directe manipulatie: bijvoorbeeld bij een autorace spelletje. In een ontwerpomgeving is het verschil tussen deze twee modes van bediening van groot belang. Commando's kunnen de computer complexe taken laten uitvoeren, maar geven de gebruiker weinig invloed over de nuances en details van de actie die uitgevoerd moeten worden. Een directe manipulatie geeft de gebruiker de volledige controle over alle details, maar die volledige controle kost vaak veel moeite om te leren.

Een potlood geeft de gebruiker volledige vrijheid in zijn gebruik. Daar staat tegenover dat niet iedereen over adequate vaardigheden (skills) beschikt om met het potlood grote gedetailleerde tekeningen te maken. Het gebruik van een potlood vergt oefening. Wat niet wil zeggen dat het gebruiken van commando's geen oefening vergt: vaak moeten ook de specifieke commando's van een systeem geleerd worden, voordat iemand er efficiënt gebruik van kan maken.

Op de vraag welke onderdelen van een ontwerpomgeving door directe manipulatie en welke door commando moeten worden bediend is geen eenvoudig antwoord te vinden. Directe manipulatie is verwant aan wat vaak 'intuïtie' genoemd wordt. Een actie heeft een directe feedback, waardoor de gebruiker direct ziet wat de effecten zijn van zijn acties. Binnen een ontwerpomgeving is er ook in zekere zin sprake van 'manipulatie van materialen' waardoor directe manipulatie vaak een logische keuze lijkt. Commando's zijn gerelateerd aan geautomatiseerde taken. Het is alsof je iemand anders (de computer) de opdracht geeft om een bepaalde taak uit te voeren. En sommige taken zijn het weldegelijk waard om uit te besteden. Het verschil zit in de keuze tussen het zelf bouwen van een kast of iemand vragen voor je een kast te maken. De eerste optie kost meer tijd maar geeft je meer controle, de tweede bespaart je tijd maar je geeft de controle uit handen.

Een ontwerpomgeving zou beide wijzes van operatie moeten ondersteunen, maar ze niet onderling uitsluiten. Wanneer iets geautomatiseerd kan worden moet het ook gecommandeerd kunnen worden, maar dat mag niet uitsluiten dat de gebruiker ervoor kan kiezen iets d.m.v. directe manipulatie te doen. Mogelijke commando's moeten dan

Ontwerpen als spelen, (playfulness)
Dynamische toolsets zoals een gereedschapskist of een bak met losse onderdelen zijn misschien moeilijk te onderhouden, maar bieden ook voordelen. Gravend door een bak met legosteentjes vindt je dynamisch oplossingen voor de problemen die zich voor doen. Je hebt niet vantevoren een precies beeld gevormd van welke onderdelen je nodig hebt om iets te maken. Maar je vindt terwijl je door de onderdelen zoekt een oplossing die past. Een dergelijke situatie stimuleert een meer ad hoc manier van ontwerpen. Dit in contrast tot een meer mentale en pragmatisch/lineaire werkwijze.

De speelsheid die een dergelijke aanpak heeft zorgt ervoor dat er iteratief aan een ontwerp gewerkt kan worden. Eerst wordt er een makeshift model gemaakt of gebouwd. Aan de hand van dit model kan de ontwerper al verschillende problemen identificeren, die vervolgens door tinkering kunnen worden opgelost. Zoals Alexander schrijft: "The failure and inadequacy of the form leads directly to the action ... Failure and correction go side by side. There is no deliberation in between the recognition and the reaction to it" [7]. Of door Dennet: "This general technique of making a more-or-less educated guess, working out its implications, and using the result to make a correction for the next phase has found many applications. A key element of this tactic is making a mistake that is clear and precise enough to have definite implications" [8]. Een bepaalde oplossing wordt gebruikt zolang deze volstaat, wanneer blijkt dat die niet volstaat kan er gezocht worden naar een andere oplossing. Tools die voor handen zijn worden gebruikt zolang ze volstaan. Als blijkt dat ze niet toereikend zijn kan er gezocht worden naar een betere of uitgebreidere variant.

Veel CAD tools zijn complexe omgevingen, de gebruiker moet leren om er mee om te gaan door een cursus te nemen of tutorials te volgen. Ze hebben een steile leercurve. Dat wil zeggen dat er veel geleerd moet worden, eer de gebruiker er op een redelijke wijze mee om kan gaan. Om een omgeving intuïtiever te maken is het dan ook niet zo zeer van belang om de omgeving eenvoudiger te maken, of om complexiteit weg te halen, als meer het 'vlakker' maken van deze leercurve. Anders gezegd: de gebruiker kan meteen aan de slag, en leert gaande(spelenderwijs) hoe hij de omgeving gebruikt. Niet door instructie of rote learning* maar door exploratie en experimentatie maakt de gebruiker zich de omgeving eigen.

Dit 'spelen' kan worden opgedeeld in twee manieren: verkenning door intentie, of verkenning door toeval/nieuwsgierigheid. Verkenning door intentie betekent dat de gebruiker graag iets wil doen, bijvoorbeeld een object verplaatsen. Vervolgens zal de gebruiker op bewust (met intentie) op zoek gaan naar een manier om het object te verplaatsen. Verkenning door toeval betekent dat de gebruiker geen directe intentie heeft, anders dan zich bekend maken met een functie. Eenvoudiger gezegd: de gebruiker wil 'kijken wat de knopjes doen'. Beide wijzen van verkenning zijn legitiem, en zullen dus ondersteund moeten worden. Alhoewel er in de beginfase, vooral uit nieuwsgierigheid zal worden gehandeld. Als de gebruiker zich vervolgens meer bekend heeft gemaakt met de omgeving, zal er voornamelijk met intentie verkend worden.

* rote learning is het uit het hoofd leren door middel van herhaling, zoals bijvoorbeeld het onthouden van een pin-code of een telefoonnummer

Generatie

In de generatie fase werden er aan de hand van de analyse ideeën en concepten onderzocht en ontwikkeld. Er werd een aantal experimenten uitgevoerd waarin werd gezocht naar geschikte onderliggende structuren en principes waarop de ontwerpomgeving gebouwd kan worden. De insteek was hier een “top-down“ benadering. Dat wil zeggen dat er niet zo zeer vanuit de onderliggende bestaande technologieën beredeneerd werd (a.i. welke toepassingen kunnen we verzinnen voor bestaande technologieen en algorithmen?). Eerder werd gezocht naar interessante abstracte theoretische modellen en concepten die een richting kunnen geven aan een implementatie van een programma en interface(a.i. welke technologieën zijn er die toegepast kunnen worden binnen een bepaald denkkader).

Navigeren door de ontwerpruimte

Een ontwerp heeft een groot aantal eigenschappen en parameters. De verzameling van alle mogelijke combinaties van eigenschappen en parameters noem je dan de 'ontwerpruimte'. Ontwerpen is het doorzoeken van deze ontwerpruimte. Er is gezocht naar concepten voor structuren en principes die het doorzoeken van de ontwerpruimte voor de gebruiker kunnen vergemakkelijken.

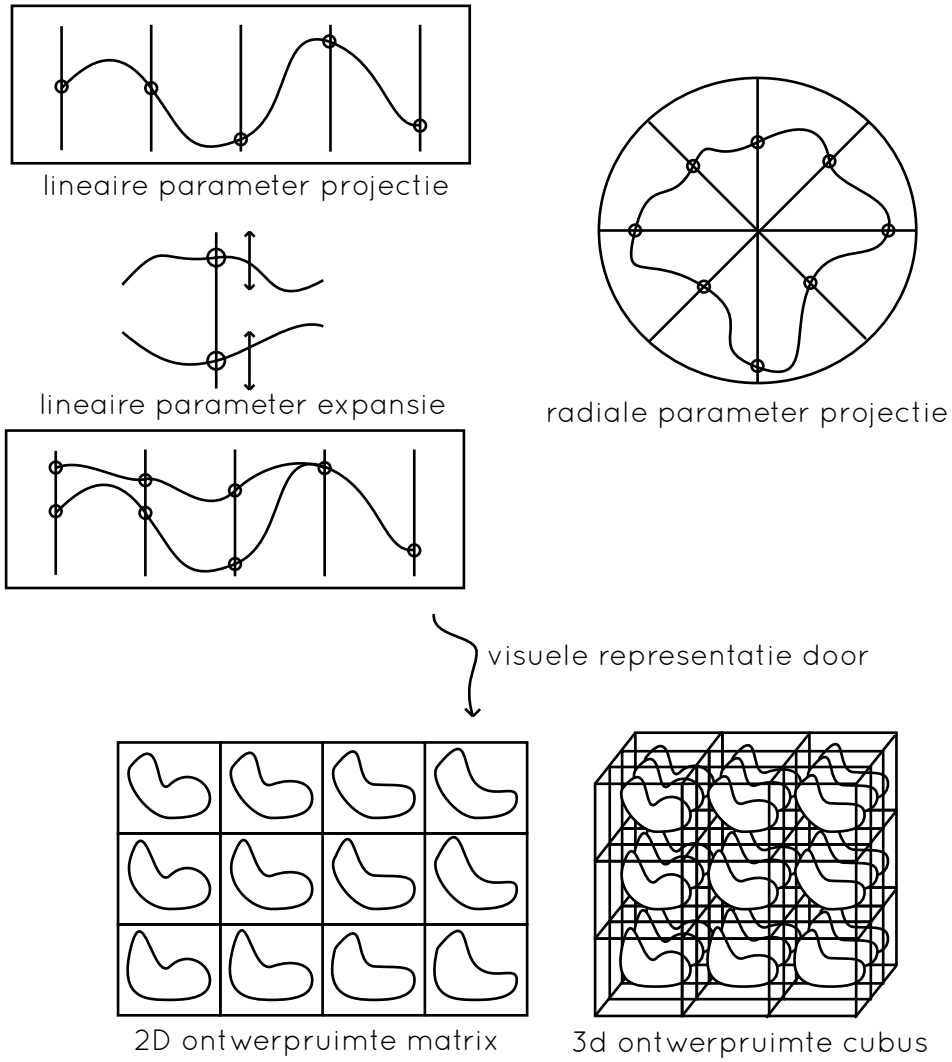


fig.4 Projecteren van parameters op 'ontwerp ruimtes'

Eigenschappen

Objecten bevatten naast geometrische eigenschappen ook eigenschappen van het materiaal in een object. Eigenschappen zoals kleur, gewicht, stijfheid, buigsterkte of weerstand. Deze gegevens kunnen gebruikt worden om berekeningen uit te voeren, om schattingen te kunnen maken over of het ontwerp. Bovendien kan een object nog bepaalde meta-data zoals een naam, auteur, opmerkingen, labels en categoriseringen bevatten.

Ruimtelijke configuratie, Relaties

Verschillende onderdelen van een ontwerp moeten ge(re)configureerd en ge(re) groepeerd kunnen worden. Zo worden hun onderlinge (ruimtelijke) relaties vastgelegd. Er moet gebruik gemaakt kunnen worden van operaties om relaties vast te leggen en te ontkoppelen. Naast voor de handliggende ruimtelijke relaties zoals “op“ of “naast“, zijn er ook andere soorten relaties zoals “bevattend”, “verbonden met“, “passend in“, “omkeerbaar”, “aansluitbaar”, “verwisselbaar” of “demonteerbaar”. De ontwerpomgeving moet ook dit soort relaties kunnen omschrijven. Dit is nuttig voor het vastleggen van ideeën en noties die niet direct puur in geometrie vastgelegd kunnen/hoeven worden.

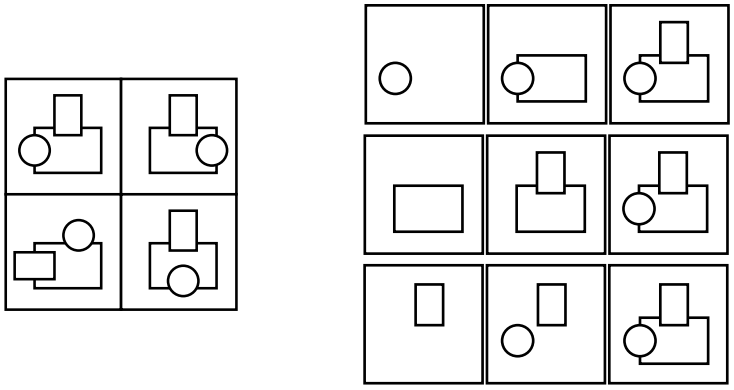


fig.5 Verschillende configuraties, en verschillende volgordes van configureren

Split Views: Virtuele representaties moeten vanuit verschillende punten tegelijkertijd bekeken en vergeleken kunnen worden. Verschillende variaties combinaties en concepten van hetzelfde object moeten naast elkaar kunnen bestaan en bekeken kunnen worden. Dat zou bijvoorbeeld kunnen door parameters te visualiseren op een grafiek, en ze verschuifbaar te maken. Door voor bepaalde parameters een bereik te specificeren zou de software een matrix van alle verschillende resultaten kunnen weergeven. (fig 4)

Tijdlijn views: Het creatieve proces moet worden vastgelegd en omkeerbaar zijn. De onstaansgeschiedenis van virtuele representaties moet kunnen worden bekeken. De gebruiker kan ten alletijde beslissen om terug in de tijd te gaan en opnieuw te beginnen vanaf een ander ingangspunt. (fig 6)

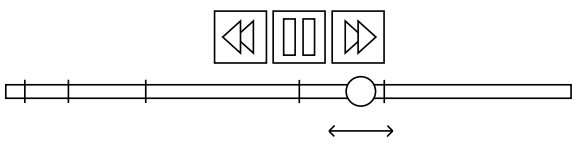


fig.6 Een tijdlijn van het ontwerp proces

Componenten en Archetypen

Vaak bestaat ontwerpen hoofdzakelijk uit het combineren van reeds bestaande componenten, en zelf ontworpen delen. Bijvoorbeeld bij het ontwerpen van een mixer, waarbij componenten zoals een elektromotor en schakelaars verwerkt worden. De omgeving zou een database met zulke COTS(Commericlal off the shelf) Componenten kunnen implementeren, waaruit de ontwerper makkelijk componenten kan halen. Zo kan de ontwerper snel componenten bij elkaar zetten en inzicht krijgen in hoe het ontwerp functioneert. Sommige componenten, zoals tandwielen, zijn bovendien makkelijk te genereren. Door bijvoorbeeld een overbrengingsverhouding te specificeren kan de omgeving een voorstel doen voor een set tandwielen. De omgeving zou dus ook ondersteuning moeten geven aan algoritmes die componenten en vormen genereren.
Niet alleen componenten, maar ook andere artefacten zijn nuttig voor de ontwerper. Zo zou de ontwerper contexten en omgevingen inladen om de ontwerpruimte in te kaderen. Modellen van archetypen zoals mensen, huizen, landschappen, lampen, etc. leveren visuele en ruimtelijke informatie waaraan het ontwerp gemeten kan worden. Het ontwerpen binnen een context geeft de ontwerper een duidelijk startpunt om mee te beginnen.
Bovendien zouden artefacten en componenten gelinkt kunnen worden aan api's van online componenten winkels en 3d printing services. Dat zou ervoor zorgen dat de ontwerper snel en eenvoudig zijn model kan bestellen om er een fysiek prototype mee te bouwen.

In context problem solving

Een dergelijke manier van ontwerpen zou je 'in context problem solving' kunnen noemen. Door componenten te verzamelen, te maken en te combineren ontdekt de ontwerper al doende waar er problemen zijn, en wat de verschillende mogelijke oplossingen zijn. Dat geldt in ieder geval voor problemen op het gebied van ruimtelijke configuratie, maar zou verder ondersteund kunnen worden door verschillende soorten simulaties. Door bijvoorbeeld simulaties van elektronische circuits te ondersteunen zou de ontwerper zelfs correct werkende elektronica kunnen ontwikkelen. Het voordeel hiervan is dat de ontwerper "spelenderwijs" te werk kan gaan, en al experimenterend verschillende circuits kan bouwen en testen. Andere mogelijke simulaties zijn: Eenvoudige statica en dynamica, Omgevings simulaties zoals regen, zonneschijn en wind, FEA(Finite Element Analysis), Produceerbaarheids tests(Is iets spuitgietbaar? Is iets te frezen?). De insteek van zulke simulaties is om "in context" op een intuïtief niveau feedback te kunnen geven aan de ontwerper, niet per se om ontwerpen te optimaliseren, dat gebeurt immers pas in een veel later stadium van het ontwerp proces.

fig.7 Schetsen over een model hoofd

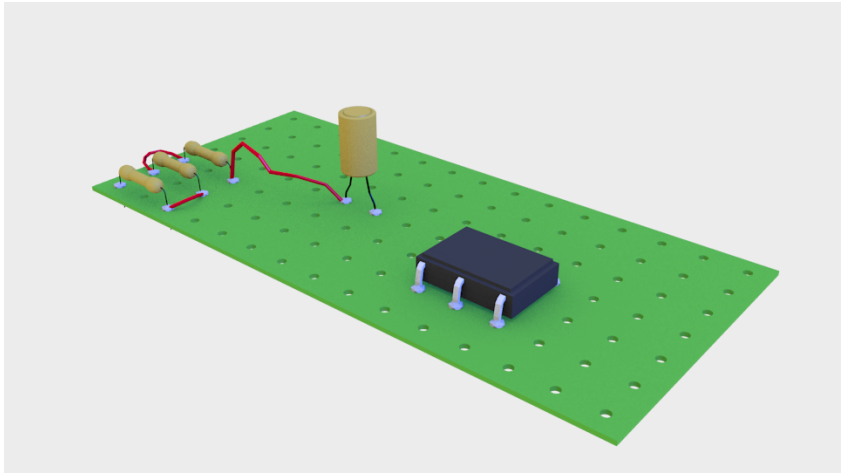


fig.8 Maak een circuit

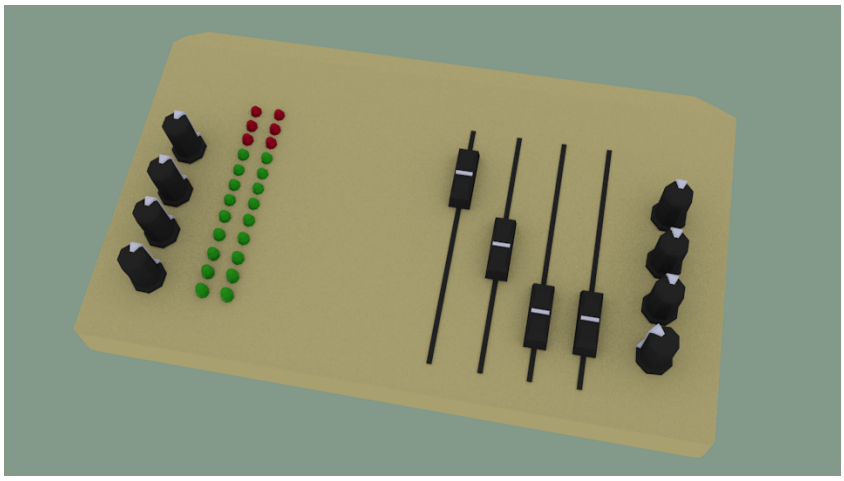


fig.9 Configureren van COTS componenten

Tools gegeneraliseerd
Tools zijn het centrale element van de ontwerpomgeving. Tools zijn de voornaamste manier voor de ontwerper om zijn ontwerp aan te passen. Tools komen in verschillende vormen voor. Ze kunnen op verschillende manieren omschreven en bediend worden. Maar wat is een tool nu eigenlijk? Om dat te kunnen beantwoorden werd gezocht naar de algemene eigenschappen van een tool. Een lijst van tools die waarschijnlijk in de omgeving gebruikt zullen worden werd opgesteld om dit algemene model aan te meten.

In zijn eenvoudigste vorm is een tool een proces dat een input neemt, er iets mee doet en een output genereert. Wat er precies gebeurt is niet per se van belang, de gebruiker ziet voornamenlijk wat hij erin stopt en wat eruit komt. Hieraan defineert hij of zij wat er gebeurt. Een soort 'black box' dus. (fig 10)
Er zijn verschillende soorten input die de gebruiker kan geven. Ten eerste de statische input: bijvoorbeeld een bestaand object waarop een actie uitgevoerd kan worden. Ten tweede de dynamische input: dit heeft bijvoorbeeld te maken met de hoeveelheid en richting van een extrusie.

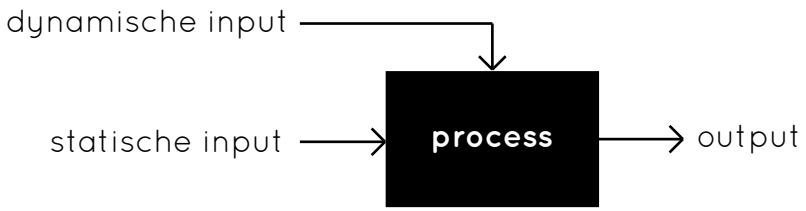


fig.10 Black box proces

Alle tools doorlopen in gebruik overeenkomstig drie stadia:

① Start: De gebruiker klikt op de knop, de tool wordt geladen, en de gebruiker defineert input.

② Actie: De gebruiker ziet een 'preview' van het object en kan de input tweaken en veranderen om de output te verbeteren.

③ Bevestig: De gebruiker bevestigt deze output, of besluit om het resultaat weg te gooien en terug te keren naar de vorige staat. Het einde van het gebruik van de tool.

Tools kunnen omschreven worden met zelfstandig naamwoorden zoals: Mes, Podlood, Gum, Schaar.
Of door werkwoorden: Snijden, Tekenen, Gummen, Knippen. Waar zelfstandig naamwoorden als een metafoor verwijzen naar een echt object, zijn werkwoorden meer directe omschrijvingen van acties. Vaak wordt een mengeling gebruikt maar in dit geval werd gekozen om uitsluitend werkwoorden te gebruiken. Dit om de tijdsgebonden aard van het gebruik van tools te onderstrepen. Ook om het verschil te benadrukken tussen enerzijds acties en anderzijds objecten waarop ze uitgevoerd worden.

Ontwerpomgevingen zijn onderhevig aan iets wat 'creeping featurism' heet. Dat wil zeggen dat het software pakket begint met een eenvoudige set aan functies en tools. Sommige gebruikers zullen graag functionaliteit willen die het pakket nog niet ondersteunt, dus wordt deze functionaliteit toegevoegd aan het pakket. Maar verschillende gebruikers hebben verschillende wensen, en willen verschillende functies en tools kunnen gebruiken. Dit heeft als gevolg dat het pakket steeds meer functies krijgt, en als een gevolg daarvan steeds complexer is om te gebruiken. Het eens zo eenvoudig te gebruiken pakket is nu een complex geheel geworden. In essentie zijn er tot nu toe twee oplossingen voor dit probleem. Ten eerste kun je weigeren meer functionaliteit toe te voegen, onder het motto: 'we doen één ding, en dat doen we goed'. Ten tweede kan je zoveel mogelijk functionaliteit proberen toe te voegen, en de complexiteit van het pakket zo klein mogelijk proberen te houden door functionaliteit te groeperen in submenus. Bovendien kan er in sommige paketten door derden functionaliteit worden toegevoegd in de vorm van zogenaamde plugins. Zo kunnen individuele gebruikers zelf functionaliteit schrijven en die toevoegen.

Dit zijn echter allebei niet echt oplossingen. Het is meer een afweging die gemaakt moet worden tussen eenvoud en complexiteit. Daarom werd gezocht naar een andere oplossing voor dit probleem.

Wat nu als alle functionaliteit per definitie als plugin wordt geïmplementeerd? Dat zou betekenen dat de gebruiker de omgeving helemaal naar zijn eigen wensen kan inrichten, maar ook alle functionaliteit zelf moet toevoegen. Dat zou onhandig zijn aangezien de gebruiker dan telkens wanneer hij een nieuwe functie zocht daarvoor een nieuwe plugin zou moeten zoeken en installeren. Als het ontwerppakket inherent een zoek en installeer modus zou hebben, in plaats van dat de gebruiker op het internet zou moeten zoeken, zou ook dat probleem al enigszins zijn opgelost. Als het pakket een database heeft met mogelijke plugins waarbinnen de gebruiker vanuit het pakket in kan zoeken, kan de gebruiker snel en eenvoudig functionaliteit vinden. Hoe dit zoeken zou kunnen werken werd verder onderzocht in het experiment over natuurlijke taal(zie ook Natural Language Interface)
Door functionaliteit alleen als plugin te implementeren, kun je eigenlijk niet meer spreken van een 'softwarepakket'. Het zou meer gaan om een soort "ecosysteem" van tools. Zeker wanneer functies en tools van elkaar afhankelijk kunnen zijn. Oftewel wanneer 'hogere' tools gebruik maken van tools die 'lagere' functionaliteit hebben.

Mapping: Dynamische input kan op verschillende manieren door verschillende apparaten geleverd worden. De gebruiker kan een muis, camera of wacom tablet gebruiken, zie ook bijlage L. Niet iedere gebruiker heeft de beschikking over dezelfde rand apparatuur. Om iedere tool met ieder willekeurig randapparaat te kunnen gebruiken moet er een laag tussen deze twee systemen zijn die ze verbind. Deze mapping vertaalt de parameters van de randapparatuur in parameters van de tool. Voor iedere tool en voor ieder apparaat heeft de omgeving een mapping. De gebruiker moet makkelijk zelf mappings aankunnen maken, en mappings delen door ze openbaar te maken.

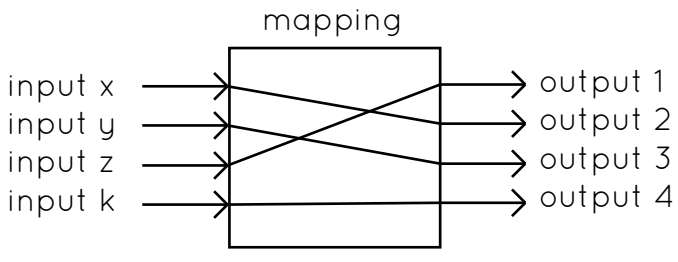


fig.11 Mapping

Toolsets
Om verschillende tools te verzamelen, te organiseren en bij te houden moet de omgeving een bepaald mechanisme hebben. Hiertoe werd een aantal ideeën ontwikkeld over hoe dit in zijn werk zou kunnen gaan. (fig 12)

Een van de ideeën was, dat toolsets gecreëerd konden worden door ze als "knikkers" te verzamelen. De gebruiker kan groepjes knikkers vormen door de knikkers heen en weer te slepen. Knikkers die bij elkaar in de buurt zitten plakken aan elkaar om zo tot zelf-organiserende groeperingen te komen. Uiteindelijk werd een simpele applicatie geschreven om dit principe te demonsteren. (fig 13)

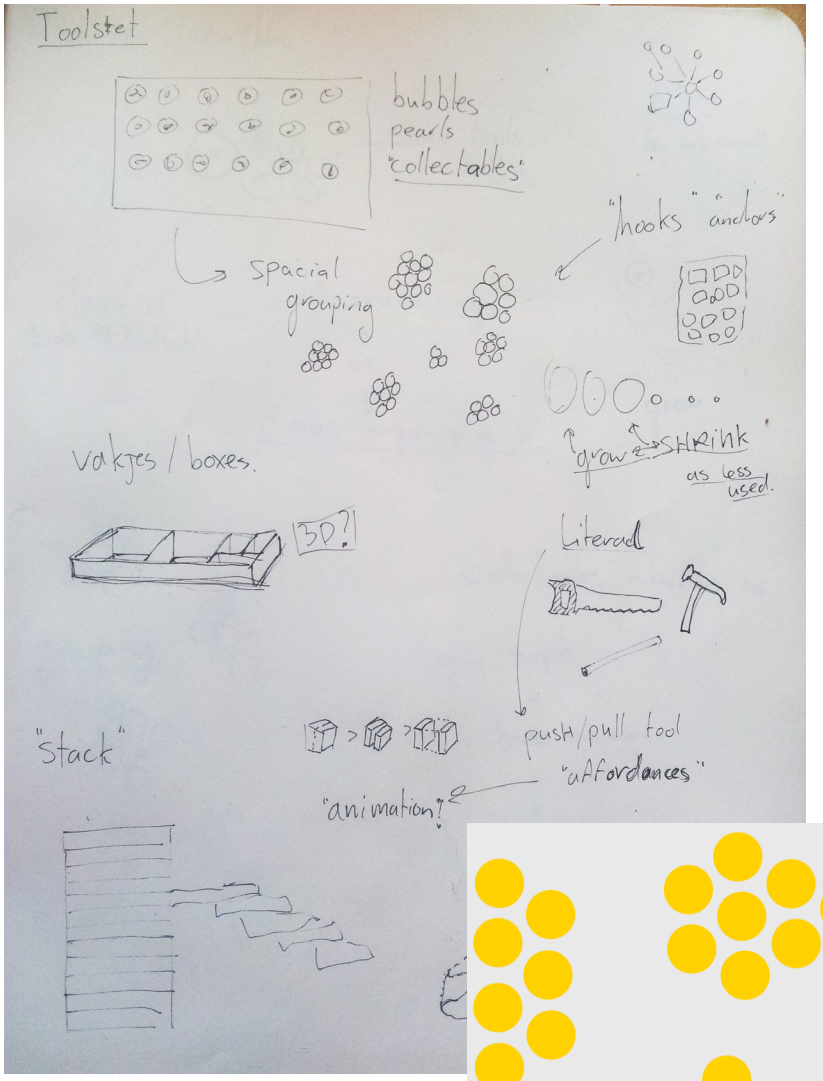


fig.12 Concepten voor toolsets

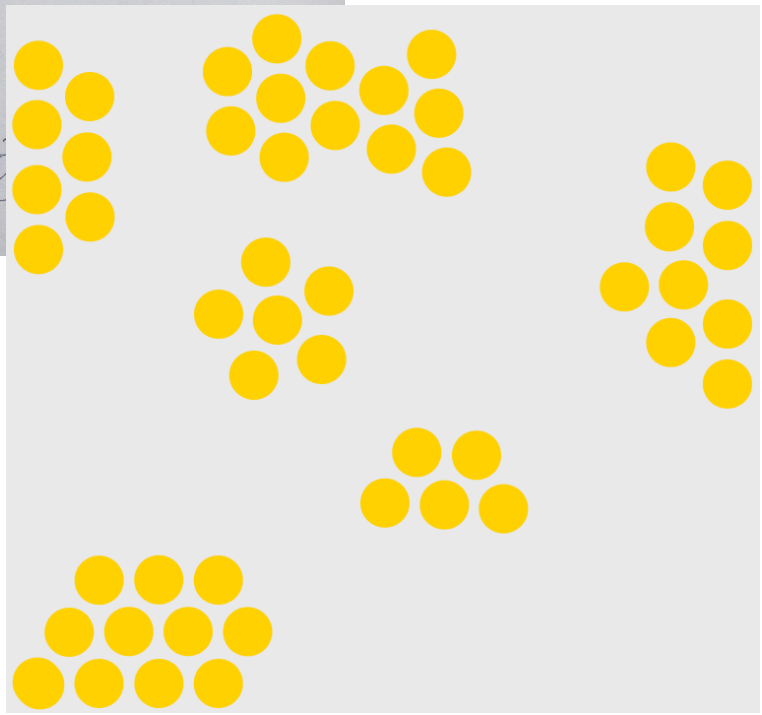


fig.13 Emergente Organisatie

Collaboratieve/multimode omgeving

Ontwerpprojecten zijn vaak projecten die met meerdere mensen samen gedaan worden. Meerdere mensen werken tegelijkertijd aan verschillende of dezelfde onderdelen van een ontwerp. Dat doen ze vanuit verschillende standpunten en misschien wel op verschillende plaatsen, fysiek van elkaar verwijderd. Dit feit wordt niet of nauwelijks ondersteund door bestaande ontwerpomgevingen en het zou intressant zijn te onderzoeken in hoeverre de ontwerpomgeving dit kan ondersteunen. Een van de mogelijkheden zou zijn om een cloud architectuur toe te passen. Waarbij de ontwerpen op een server opgeslagen worden, en vervolgens door clients bekeken en bewerkt worden. Het voordeel hiervan is dat meerdere clients tegelijkertijd aan de zelfde data kunnen werken, zonder dat er verschillende versies hoeven te zijn op verschillende computers. Het voorkomt dat gebruikers bestanden heen er weer moeten sturen. Clients kunnen verschillende soorten apparaten zijn, zolang ze een internet verbinding hebben, en de software kunnen draaien, kunnen ze allemaal tegelijkertijd dezelfde dataset bekijken. Deze multimodaliteit biedt de gebruiker een extra mogelijkheid: Hij kan vloeiend van het ene naar het andere apparaat overschakelen. Zo is een scenario denkbaar waarbij de gebruiker eerst schetsen maakt met behulp van een LFDS. Hier vervolgens op zijn laptop een 3d model van maakt. Een collega kan hier vervolgens met zijn tablet opmerkingen en aantekeningen bij maken. Zo vormt zich een vloeiend proces, waarbij de apparaten functioneren als 'ramen' die uitkijken op de dezelfde virtuele wereld.

Het zou ook mogelijk zijn om een peer-to-peer architectuur te gebruiken waarbij er geen centrale server is, maar alle clients onderling een netwerk vormen. Hoewel een dergelijke architectuur om verschillende redenen prefereerbaar is (lagere kosten, directe verbinding) heeft een centrale server toch de voorkeur. Dit omdat alleen een server kan garanderen dat de data consistent blijft tussen clients.

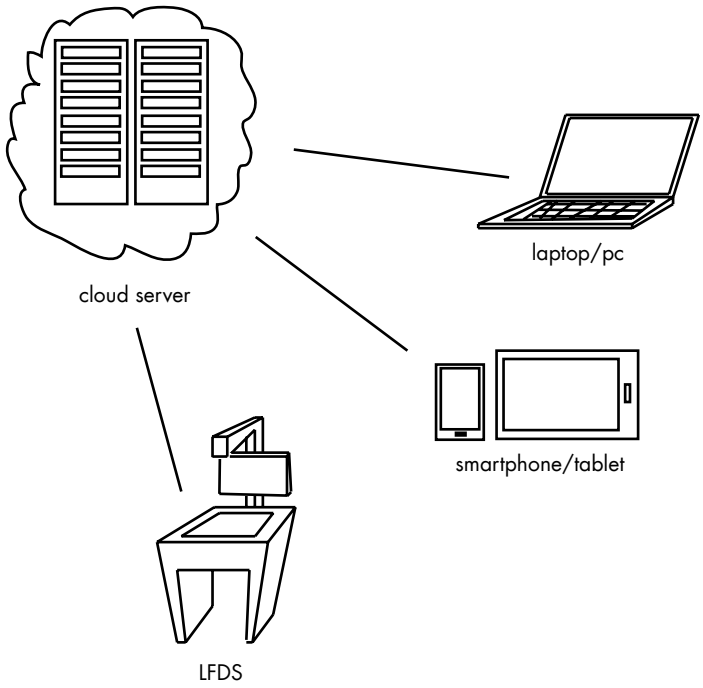


fig.14 Cloud architectuur

Natural Language Interface

In dit experiment werd gekeken hoe het met behulp van natuurlijke taal geïnteracteerd kan worden met een ontwerpomgeving. Zoals al eerder geschreven kan er op twee manieren met een computer geïnteracteerd worden: door directe manipulatie of door commanderen. Commanderen gebeurt veelal in een formele taal. De zogenoemde commandline interface in verschillende besturingssystemen is hiervan een voorbeeld. Commandline interfaces zijn vaak heel handig als het om complexe taken gaat, ze zijn echter steeds meer in onbruik geraakt sinds de opkomst van de grafische user interface (GUI). Een Commandline vereist namenlijk dat de gebruiker tientallen zo niet honderden formele commando's uit zijn hoofd leert, waar een GUI veel 'verkenbaarder' is, het maakt zichtbaar welke opties mogelijk zijn. Een interface waarbij de gebruiker met een natuurlijke(niet formele) taal met de computer kan interacteren lijkt dan een mooie middenweg. Het invoeren van de zin: "maak dit groen", is immers veel minder werk dan het zoeken van de kleur optie en vervolgens het zoeken van de kleur groen in een menu.

In eerste instantie werd een puur natuurlijke interface onderzocht. Een eenvoudige hypothetische ontwerpomgeving zou met de volgende zinnen kunnen worden aangestuurd:

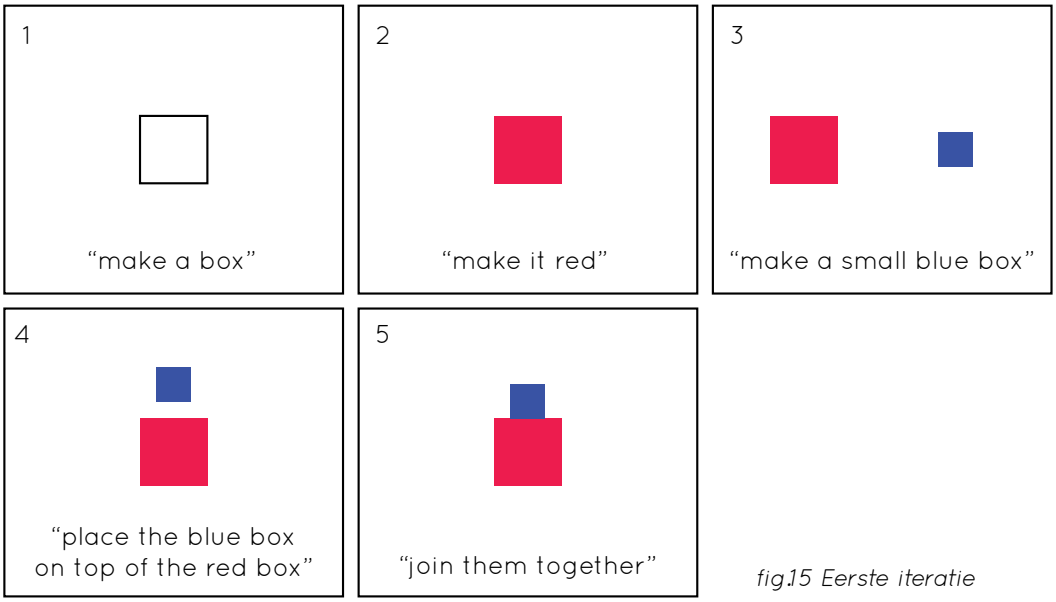


fig15 Eerste iteratie

Meteen zien we een groot aantal struikelblokken:
Het eerste commando is vrij eenvoudig te begrijpen: voer de actie "make" uit op het object "a box".

Het tweede commando wordt al ingewikkelder: voer de actie "make" uit op het object "it" met de eigenschap "red".

We zien hier ten eerste dan het woord "make" ambigu is. Het kan verwijzen naar zowel "het creëren van een ding" als "het veranderen van een eigenschap van een ding". Bovendien heeft het woord "it" geen directe betekenis. Het verwijst naar het "a box" van het vorige commando. Dat betekent dat de interface de individuele commando's alleen kan begrijpen als het de onderliggende semantiek van het geheel begrijpt. Oftewel: dat "it" verwijst naar hetgeen hij net heeft gedaan.

Het derde commando laat zien dat volgorde van de woorden ook van belang is. Waar het in de eerste twee commando's voldoende was om alleen de losse woorden

te herkennen, moet nu ook de onderliggende grammatica van de zin begrepen worden. “Make a small blue box” betekent immers iets anders dan “Make a small box blue” of “Make a small blue, box”. Verwijst “a” nu naar een al bestaande doos? Of naar een nieuw te maken doos? Is het te maken object een “kleine blauw die doos is” of een “kleine doos die blauw is”? De computer moet kennis hebben van welke woorden eigenschappen omschrijven en welke objecten.

Het vierde commando is het meest complex: De computer moet niet alleen kennis hebben over de wereld waarnaar de woorden verwijzen. Maar ook de onderlinge ruimtelijke semantiek moet begrepen worden. Woorden als “on top” of “left” zijn subjectief en afhankelijk van het perspectief van de gebruiker. Wat is boven en wat is onder? Bovendien laat dit commando zien dat het niet altijd efficiënter is om een puur natuurlijke interactie te hanteren. Het is weldegelijk makkelijker om zelf de blauwe doos boven op de rode te zetten, dan om de computer dit uit te proberen te leggen. In communicatie van mens tot mens gebeurt dit ook niet: Je gebruikt geen formele zinsbouw maar spreekt hakkelend en gebruikt ook gebaren als onderdeel van je communicatie: “deze moet hier bovenop” (al wijzend)

Natuurlijke taal betekent dus niet direct natuurlijke interactie. Bij nader onderzoek blijkt dat een dergelijke puur natuurlijk interface weldegelijk mogelijk is, en zelfs al eens in experimentele vorm ontwikkeld is onder de naam SHRDLU [9]

Een simpele hybride tussen natuurlijke taal en directe manipulatie werd hierna als experiment geïmplementeerd. We vermijden hiermee het implementeren van een complex semantisch model, en de interactie wordt er natuurlijker door. De omgeving werkt als volgt:

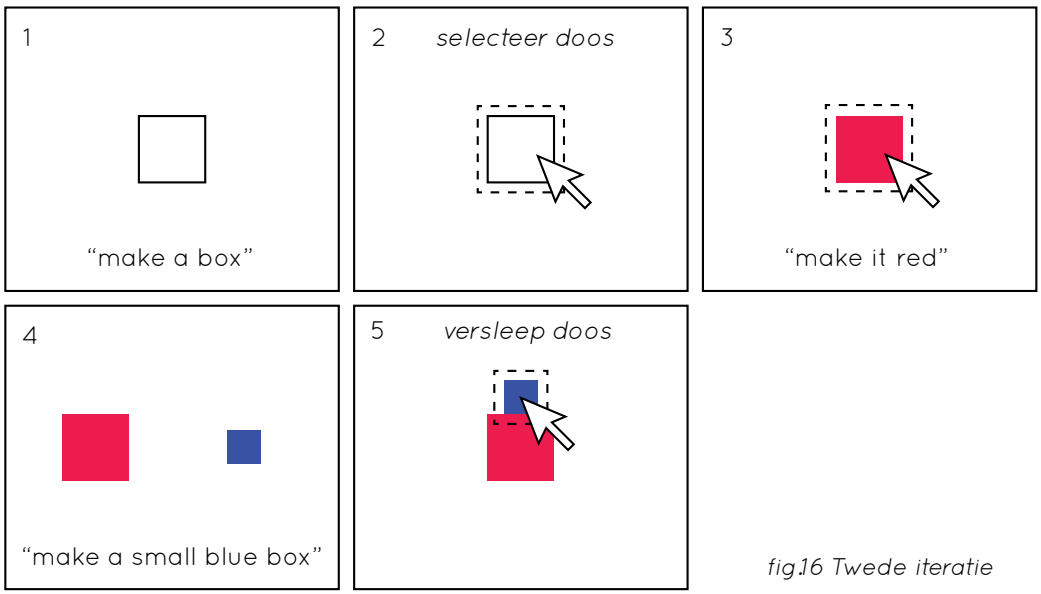


fig.16 Twede iteratie

Dit is een natuurlijke taal met formele elementen. Het woord “it” verwijst bijvoorbeeld altijd naar het geselecteerde object. De taal maakt gebruik een ‘woordenboek’ die vormen en eigenschappen omschrijft. Woorden als “box, circle, triangle” omschrijven een geometrie. Woorden als “red, green, small, large” omschrijven eigenschappen als kleur en maat. Eigenlijk is een natuurlijke zinsbouw haast overbodig, deze taal wordt in essentie alleen gebruikt om eigenschappen van dingen te omschrijven.

Misschien is het gebruik van een 'natuurlijke' grammatica wel helemaal overbodig, en is het makkelijker een meer directere, vrije vorm te hanteren:

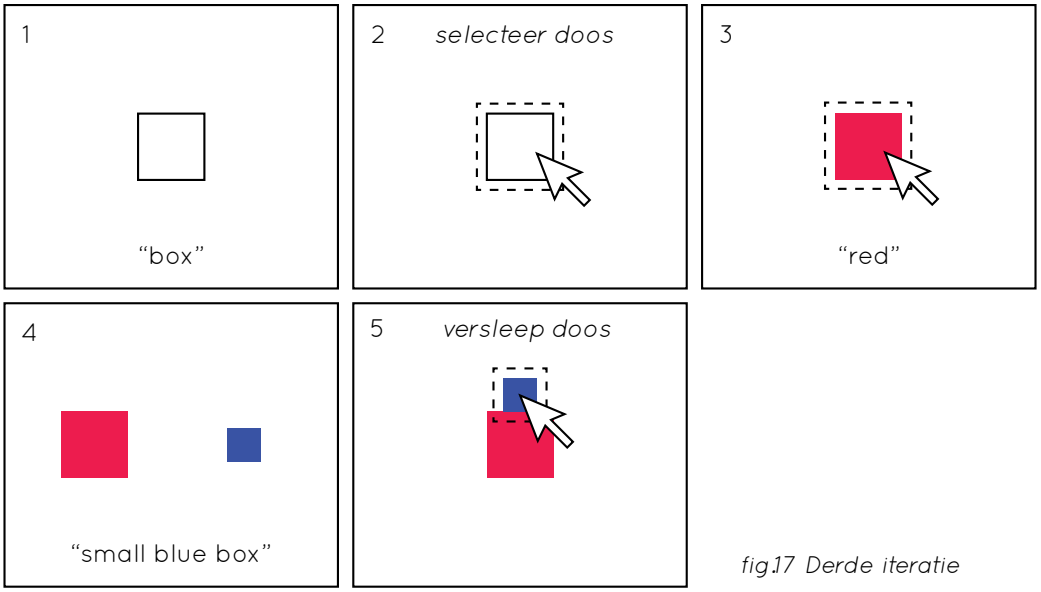


fig.17 Derde iteratie

De taal lijkt in vorm op de formele invoer van een commandline interface. Maar is veel vloeibaarder en kent geen formele commando's of syntax die geleerd moet worden. Met behulp van descriptieve woorden kunnen de eigenschappen van een object worden aangepast. Dat kan zowel relatief als absoluut, precies of vaag: "big", "bigger", "4cm", "red", "greener", "heavy", "20 gram", "15 ohm". De computer interpreteert deze descriptieve woorden en vertaalt deze naar formele eigenschappen: "heavy", "20 gram", "light", "45,5 kilo" zijn bijvoorbeeld allemaal termen die de formele eigenschap "weight" omschrijven.

De taal wordt dan eigenlijk een vervanging van het eigenschappenformulier in veel gui's. Het gebruik van zo'n taal is efficiënter dan een formulier. De gebruiker kan snel en met descriptieve termen omschrijven welke eigenschappen het object moet hebben, en hoeft niet in een formulier naar de juiste eigenschappen te zoeken. Een formulier biedt echter wel een 'zichtbaarheid' die zo'n taal niet heeft. Een formulier laat zien dat er een eigenschap "kleur" is, een text invoer laat alleen een knipperende cursor zien, de gebruiker kan alleen maar raden welke woorden hij moet intypen.

Daarom werd er gekeken naar een hybride vorm tussen formulier en textinvoer. Een dynamisch invoerveld die de gebruiker tijdens het typen direct feedback geeft over de invoer. Als voorbeeld nemen we een abstracte klasse van tekenvoorwerpen zoals pennen, potloden, stiften en kwasten. Het omschrijft een groep dingen die een lijn op papier maakt, maar met verschillende eigenschappen zoals dikte, hardheid, vorm en kleur. In een conventioneel tekenprogramma zoals photoshop zijn deze eigenschappen weergegeven in een formuliervenster.
Dergelijke objecten zouden omschreven kunnen worden met de commando's "pencil HB blue", "thin red pen" of "big black marker". Het programma heeft een woordenboek die een descriptieve term vertaalt naar een of meerdere formele eigenschappen. Zo heeft het woord "pencil" invloed op eigenschappen zoals vorm en grootte. "HB" en "blue" hebben invloed op de eigenschappen "hardheid" en "kleur". Sommige termen vertalen naar eigenschappen die overlappen met andere termen. In dat geval wordt een regel toegepast: specifiekere termen (die minder formele eigenschappen hebben) overschrijven bredere termen. In het geval van

"big black marker": eerst wordt de bredere term "marker" toegepast en vervolgens worden de eigenschappen overschreven door eigenschappen van de termen "big" en "black".

Om de zichtbaarheid van een natuurlijke taal interface te vergroten werd gekeken naar verschillende vormen van visuele feedback.

Dynamisch formulier: een formulier dat met behulp van natuurlijke taal aan te passen is. Het geeft de eigenschappen die door de taal beïnvloed zijn in een lijst weer. (fig 18)

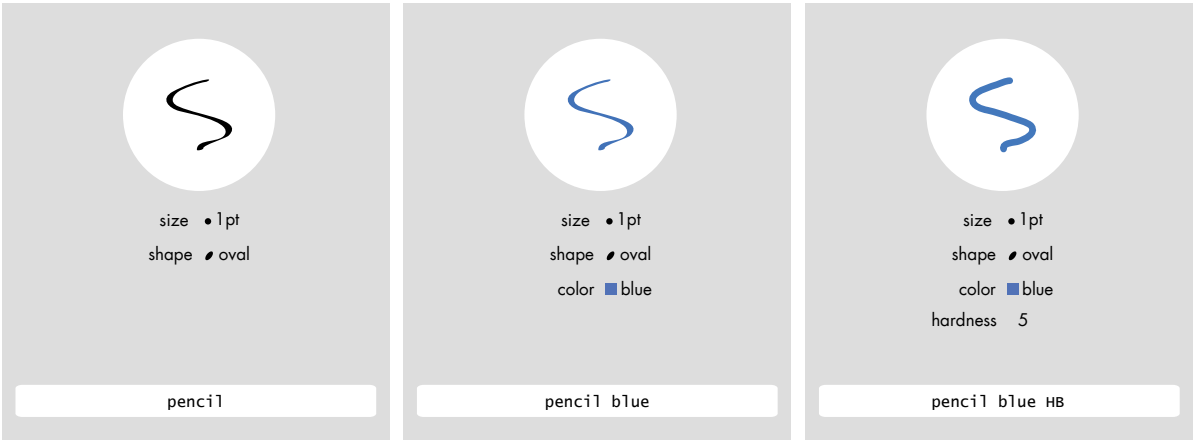


fig.18 Dynamisch formulier

Woorden ketting: waarbij ieder woord een blokje in de ketting vormt. De gebruiker rijgt gelijdelijk een ketting door steeds meer eigenschappen toe te voegen. Ieder blokje geeft weer welke eigenschap beïnvloed wordt (fig 19).

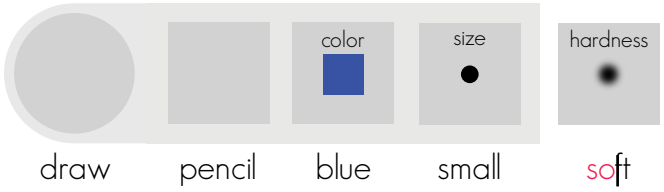


fig.19 Woorden ketting

Bovendien zou ieder blokje achteraf met de hand aangepast kunnen worden om een meer directe manipulatie te kunnen hebben over de eigenschap. (fig 20)

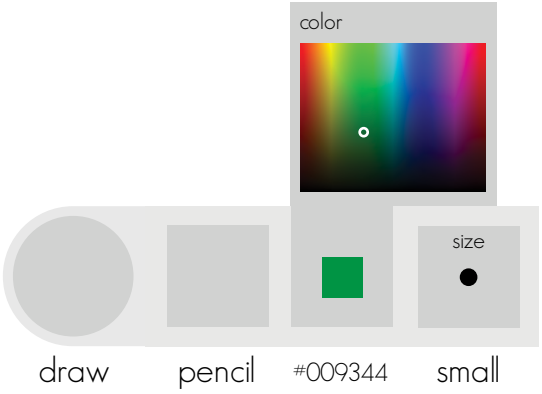


fig.20 Aanpassen van de kleur

Animatie & tijd
Ontwerpen speelt zich inherent in de tijd af, het is niet iets statisch of stapsgewijs. Zelfs wanneer de gebruiker op dat moment niets doet, geen acties uitvoert, vloeit de tijd vooruit. In dit experiment werd gekeken in hoeverre het element tijd een rol speelt in het bevorderen van flow. Een van de aannames was dat bewegingen het verlopen van de tijd benadrukken en dat dit de gebruiker betrokkener zou doen voelen.

Animatie Knoppen: Omdat tools doormiddel van werkwoorden omschreven worden (zie ook: *Tools gegeneraliseerd*) zouden de iconen ook een actie moeten uitdrukken. Er werd een aantal iconen ontwikkeld die met pijlen de beweging weergeven. Vervolgens werden in dezelfde stijl geanimeerde iconen ontwikkeld (fig 21). De geanimeerde iconen communiceren veel duidelijker de achterliggende functie. Om de iconen constant te laten animeren zou misschien wat overdreven zijn, maar ze zouden bijvoorbeeld hun animatie kunnen weergeven wanneer de gebruiker er met zijn muis overheen gaat.

Inertie: Er werd gekeken in hoeverre gesimuleerde traagheid iets toevoegt aan de ervaring van flow. Het idee was dat objecten in de echte wereld zich zelden met een constante snelheid bewegen, en dat het immergence ten goede zou komen als objecten in de virtuele wereld zich versnellen en vertragen bij het bewegen. Bij het draaien van een camera lijkt dit een positief effect te hebben, omdat het de beweging vloeiender maakt, en minder desoriënterend. Bij het bewegen van objecten in de ruimte is het nog onduidelijk of er een toegevoegde waarde is. Er zal daarom waarschijnlijk een kwantitatieve gebruikstest moeten komen om definitief vast te kunnen stellen of een dergelijk effect een positieve invloed heeft.

Overige experimenten
Er werden nog een aantal experimenten uitgevoerd rond 3d modeleren. Er werd onderandere een algoritme voor schets-achtige invoer, en een algoritme voor het virtueel opblazen van een 2d vorm tot een 3d vorm bedacht (bijlage J).
Daarnaast werd geëxperimenteerd met invoer van de leap-motion(bijlage K)

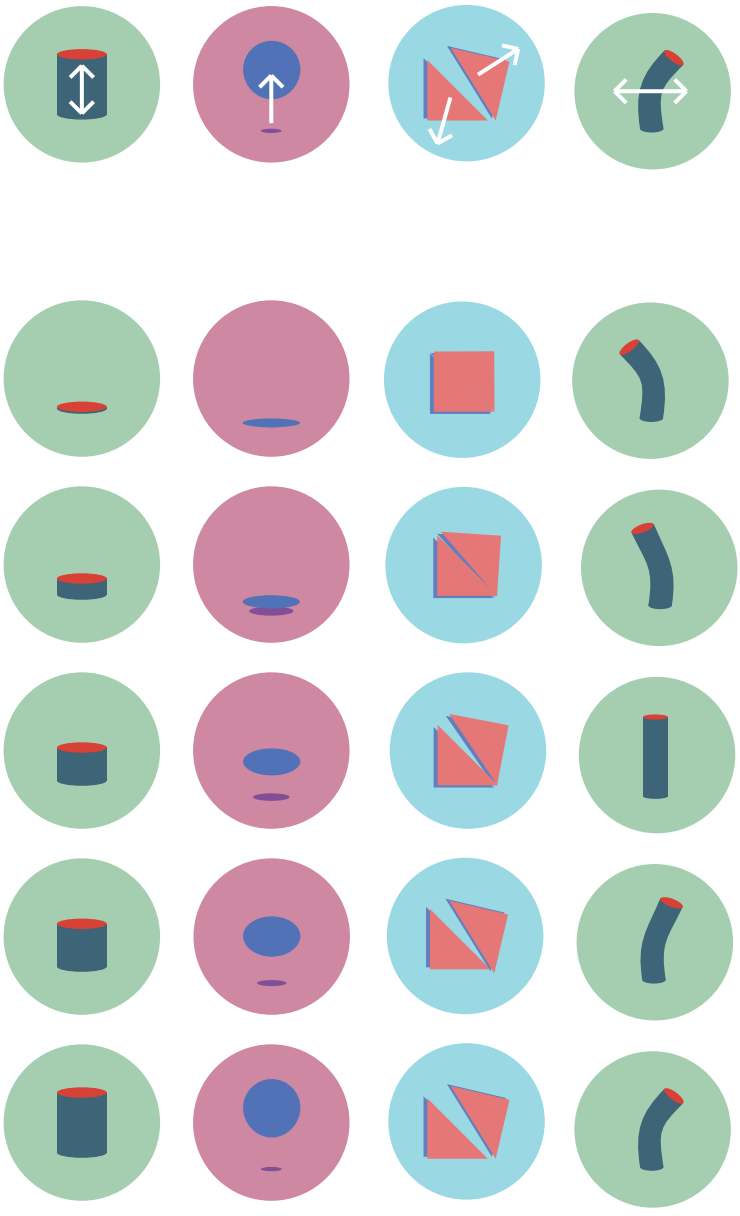


fig.21 Geanimeerde knoppen

Synthese

Tijdens de synthese fase werden de verschillende ideeën die in de generatie fase bedacht waren samengebracht tot een geheel ontwerp voor de ontwerpomgeving. Er werden ontwerpen gemaakt voor een interface, en er werd een systeem architectuur ontwikkeld, aan de hand waarvan de software geïmplementeerd kan worden.

In dit hoofdstuk zullen verschillende aspecten van de interface en de architectuur uitgelicht worden. Voor een overzicht van de volledige systeem architectuur zie bijlage A. Voor een overzicht van de Volledige interface zie bijlgen B & C.

De omgeving

De uiteindelijke ontwerpomgeving is een collaboratieve omgeving dat op meerdere platforms draait en door meerdere mensen tegelijkertijd in teamverband gebruikt kan worden. De omgeving staat de gebruiker meerdere manieren toe om (3d) ontwerp data in te voeren. De omgeving staat gebruikers toe om deze 3d modellen te ordenen tot een ontwerp. Er is daarbij geprobeerd om zoveel mogelijk functionaliteit te bieden terwijl de complexiteit van de interface zo laag mogelijk gehouden wordt. De omgeving is niet bedoeld om volledige en gedetailleerde CAD modellen in te maken, maar eerder om in collaboratie schet-sachtige modellen in 3d te kunnen maken om zo snelle ideatie en communicatie te bevorderen.

De ontwerpomgeving kan omschreven worden aan de hand van deze vier pijlers:

Eenvoud

De ontwerpomgeving biedt een eenvoudige interface, die voor iedereen snel te begrijpen is, en die voorkomt dat de gebruiker zich 'verliest' in het aantal mogelijke functies.

Samenwerking

De ontwerpomgeving is specifiek ontwikkeld om samenwerking tussen mensen te bevorderen. Het delen van een ontwerp, en het samenwerken aan een ontwerp is net zo eenvoudig als alleen aan een ontwerp werken.

Buigbaar

De ontwerpomgeving is buigbaar en vormt zich naar het specifieke gebruik van de individuele ontwerper. In plaats van een vaste workflow op te leggen laat de omgeving een breed spectrum aan gebruiksmogelijkheden toe.

Openheid

De ontwerpomgeving is modulair opgebouwd, en gebruikers zijn vrij die modules op iedere manier te combineren. Bovendien kunnen ze hun eigen modules toevoegen en die openstellen aan alle gebruikers.

De basis elementen
Om te beginnen werd een overzicht van alle elementen waaruit de ontwerpomgeving bestaat gemaakt.

De ontwerpomgeving bestaat in essentie uit twee basis elementen:
① Ruimte: waarbinnen gewerkt kan worden, een verzameling objecten
② Objecten: die zich binnen een ruimte bevinden

Objecten hebben de volgende eigenschappen:
① Positie, Rotatie, Schaal: Data die de relatie van een object tot een ruimte omschrijft
② Vorm: Data die omschrijft hoe het object wordt weergegeven, en de ruimte die het inneemt. Vorm kan de volgende typen data omschrijven:
- ㉑ Punt
- ㉒ Lijn
- ㉓ 2d Vlak
- ㉔ 3d Vlak (een gekromd vlak)
- ㉕ 3d Volume

③ Materiaal: Data die verdere eigenschappen van het object omschrijft (Kleur, Stijfheid, Elektrische Weerstand, Gewicht, etc.)

Daarnaast zijn er operaties: acties die de gebruiker op- of met objecten en ruimte kan uitvoeren Operaties zijn in te delen in vier categorieën:

① Instantieren: Het uit het "niets" creëren van een object. (tekenen, fotograferen, importeren)
② Manipuleren: Het wijzigen van de geometrie of eigenschappen van een object. (extruderen, buigen, knippen)
③ Combineren/Compositie: Het samenvoegen van twee of meer objecten om tot een nieuw geheel te komen. Anders: een (ruimtelijke) relatie tussen twee of meer objecten definiëren. (het verplaatsen, roteren en opschalen van objecten tenopzichte van elkaar. Groeperen of aan elkaar bevestigen)
④ Analyseren: Het bekijken en berekenen van eigenschappen van objecten. (opmeten, passen, testen, FEM analyse)

Al deze elementen zijn op de volgende manier met elkaar verwant:

De gebruiker kan in een ruimte nieuwe objecten en ruimtes instantieren. Dit gebeurt met behulp van het **Ruimte-Object model**.

Vervolgens kan de gebruiker de eigenschappen vorm, materiaal van objecten manipuleren. Dit gebeurt met behulp van **toolmodules** en de **toolchain**.

Hierna kan de gebruiker objecten combineren door ze samen in een ruimte te plaatsen.

De gebruiker kan onderdelen in een ruimte tenopzichte van elkaar composeren door het aanpassen van de eigenschappen Positie, Rotatie, Schaal van objecten en ruimten.

De gebruiker kan op objecten en ruimten analyses uitvoeren doormiddel van specifieke **toolmodules.**

Het Ruimte-Object Model

De manier waarop de software een ontwerp omschrijft en opslaat heeft een grote invloed op hoe de gebruiker omgaat met de ontwerpomgeving. Onze ontwerpomgeving slaat de ontwerp op met behulp van de concepten 'ruimte' en 'object'. Dit Ruimte-Object Model is een graaf met een boomstructuur (fig 22).

Een ruimte omschrijft een lokaal assenstelsel en de posities van de objecten die de ruimte bevat. Een ruimte kan ook andere ruimtes bevatten, die op hun beurt weer objecten bevatten. Door deze verzameling van (relatieve) coördinatenstels en posities terug te vertalen naar een globaal coördinatenstel kan het gehele ontwerp bekeken worden.

In deze representatie stelt een knoop in de graaf dus een object of een ruimte voor, en een referentie een relatieve translatie in de ruimte.

In de ontwerpomgeving wordt niet gewerkt met "projecten", alleen met verzamelingen ruimtes en objecten. Alle data wordt ontkoppeld en naar referentie opgeslagen. Het ROM omschrijft dus alleen de onderliggende relaties tussen objecten en ruimtes, niet de objecten zelf (fig 26). Het voordeel hiervan is dat daardoor alle data volledig herbruikbaar en herstructureerbaar is. Wanneer bijvoorbeeld een bepaald object herbruikt wordt in een ander project, is het naar referentie eenvoudig over te nemen, onafhankelijk van veranderingen in de structuur van het originele project. Gezien het feit dat ontwerpprojecten sterk naar verandering en reorganisatie neigen (zie ook: *Analyse: De non lineariteit van het ontwerp proces*), is het onverstandig om data naar relatieve referentie (fig 25) op te slaan. Wanneer er veranderingen in het originele project optreden verliest het referenende project zijn verbinding (een broken link). Het is ook mogelijk de data telkens te kopieren (fig 24) maar dit heeft niet de voorkeur, aangezien er dan veel dubbele informatie wordt opgeslagen, en bovendien de onderliggende relationele verbinding verloren gaat.

De referentiële manier van opslag benadert het beste de rhizomale structuur van een chaotisch ontwerpproces, door meerdere in en uitgangspunten te bieden. Iedere "ruimte-knoop" biedt een ingangspunt voor de gebruiker om op verschillende niveaus naar delen van een ontwerp te kijken. Bovendien heeft de data meerdere uitgangspunten doordat het in verschillende contexten herbruikt kan worden.

Een gevaar van de refrentiele manier van opslag is dat er recursie kan optreden. Dat is wanneer een ruimte, direct of indirect, naar zichzelf verwijst. Zo'n cyclische verwijzing is niet oplosbaar omdat het oneindig naar zich zelf blijft verwijzen. Daarom moet het systeem een algoritme implementeren die zo'n verwijzing vermijdt of op z'n minst beperkt.

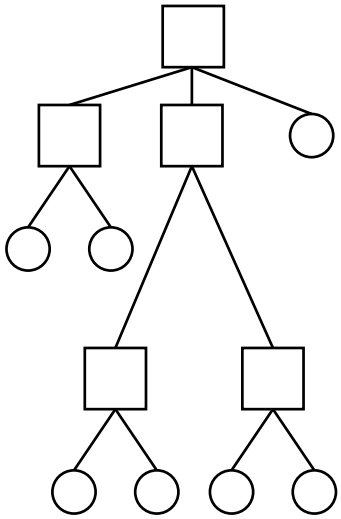


fig.22 Boom structuur

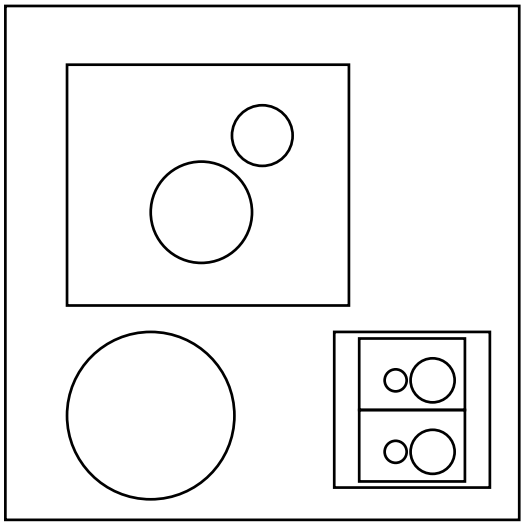
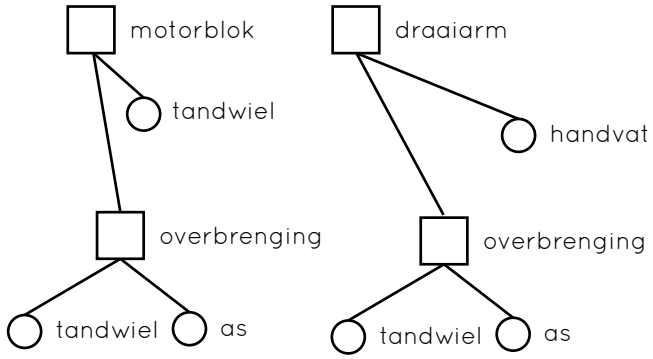


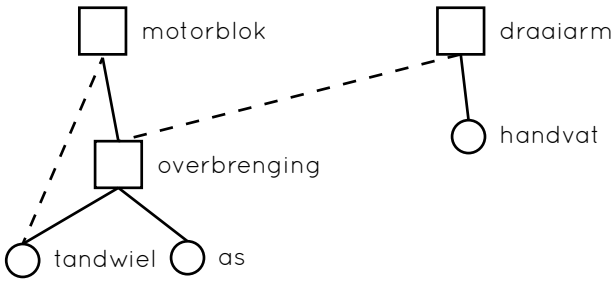
fig.23 Objecten in ruimtes

fig.24 Herbruik door kopie



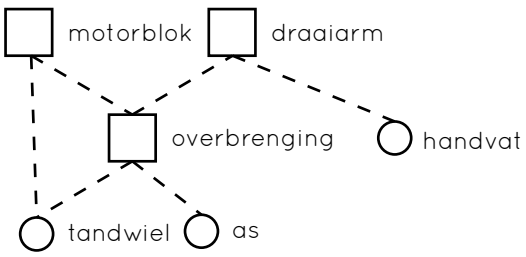
- motorblok
  - tandwiel
  - overbrenging
    - tandwiel
    - as
- draaiarm
  - handvat
  - overbrenging
    - tandwiel
    - as

fig.25 Herbruik door relatieve referentie (link)



- motorblok
  -> overbrenging -> tandwiel
  - overbrenging
    - tandwiel
    - as
- draaiarm
  - handvat
  -> motorblok -> overbrenging

fig.26 Herbruik naar referentie



r1 - overbrenging
  -> o1
  -> o2
r2 - motorblok
  -> r1
  -> o1
r3 - draaiarm
  -> o3

o1 - tandwiel
o2 - as
o3 - handvat

Toolmodules
Een toolmodule is abstract gezien een loskoppelbare eenheid van functionaliteit. Een toolmodule bepaalt of vervormt de eigenschappen van een object in het ruimte-object model.

Als eerste heeft een toolmodule een naam. De naam omschrijft wat de toolmodule precies doet, en is uniek tenopzichte van alle andere toolmodules. Naast de naam heeft hij eventueel ook een (geanimeerd) icoon, die op een visuele manier weergeeft wat de module doet. De toolmodule heeft verder ook een verzameling "tags", andere woorden die de functie van de module omschrijven. Deze hoeven niet uniek te zijn, maar zijn nuttig voor de gebruiker om een module te kunnen vinden.

Als tweede heeft een toolmodule een lijst met parameters. Die parameters hebben in ieder geval een naam, een bepaald type, en een standaard waarde. Eventueel heeft een parameter ook nog een bepaald bereik, waarbinnen de waarde moet liggen. Met behulp van deze waarden kan het programma een GUI genereren. Voor ieder type is er een standaard GUI element.
Voor een parameter kan ook een ruimtelijke "handle" gedefiniëerd worden, die de gebruiker naast het GUI element kan gebruiken om de parameter aan te passen.

De toolmodule heeft ook een collectie presets. Een preset koppelt bepaalde waardes van parameters aan tags, die de eigenschappen in woorden omschrijven.

De toolmodule definiëert ook voor zijn input en output wat voor type ze zijn. Bijvoorbeeld(van een afbeelding naar een vlak, van een vlak naar een volume).
Input en output kunnen van hetzelfde type zijn, of van verschillende types. Bovendien hoeft er niet persé een input of een output te zijn.

Als laatste definiëert de toolmodule de eigenlijke functionaliteit. Dit is de code die aan de hand van parameters en input een output genereert. Deze code wordt aangeroepen telkens als er iets aan de parameters of input verandert. Zo blijft de output altijd up-to-date.

Visueel wordt een toolmodule op verschillende manieren weergegeven. Als icoon in een toolset, als zoekresultaat, en als volle module. De volle module (fig 27) wordt weergegeven als blok met bovenin de naam en het icoon, daaronder de parameters en hun bijbehoorende GUI elementen. Onderaan het blok worden de toegepaste presets weergegeven.

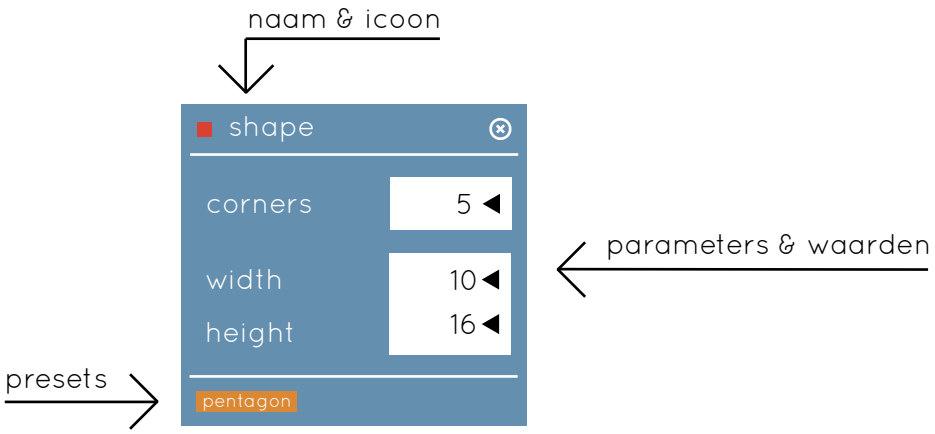


fig.27 Toolmodule

Toolbox search

Toolmodules moeten op een centrale plek verzameld worden. Dit gebeurt in de toolbox. Hier kan de gebruiker door middel van een natural commandline interface(zie ook: *Generatie: Natural Language Interface*), de juiste module vinden. Doordat zowel modules als presets met tags omschreven worden kan de gebruiker makkelijk de functionaliteit vinden die hij/zij nodig heeft. De gebruiker kan ook meerdere woorden achter elkaar intypen, om specifiekere zoekresultaten te krijgen. Zo kan de gebruiker "pencil" intypen. Een resultaat zal dan de "draw" module zijn, met een preset "pencil" (fig 29). De gebruiker kan ook "big blue pencil" intypen, deze zoekopdracht zal door de parser geïnterpreteerd worden om zoveel mogelijk juiste operaties op te leveren, de meest waarschijnlijke eerst (fig 30).

De gebruiker kan zijn of haar favoriete modules in een balk aan de zijkant plaatsen, om deze modules sneller te kunnen gebruiken, zonder telkens een zoekopdracht te gebruiken. Deze snelkoppelingen zijn zo nodig te verdelen over verschillende tab-bladen (fig 28). Zie ook bijlage B & C voor een overzicht van de hele interface.

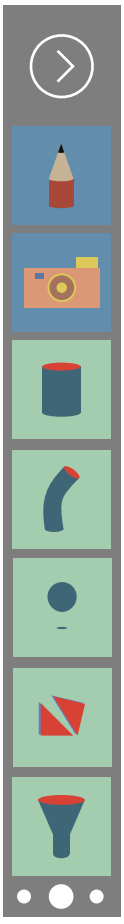


fig.28 Toolbox

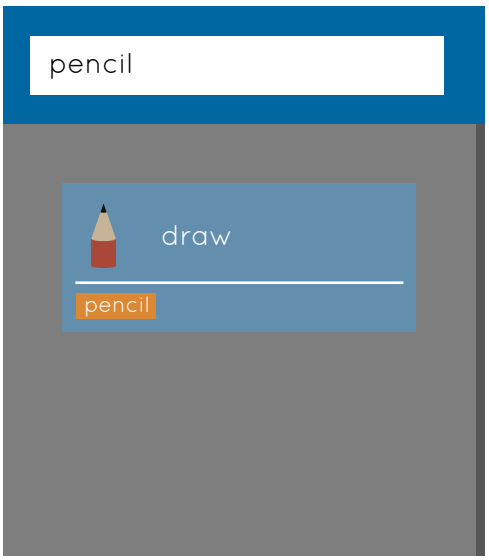


fig.29 Toolbox search

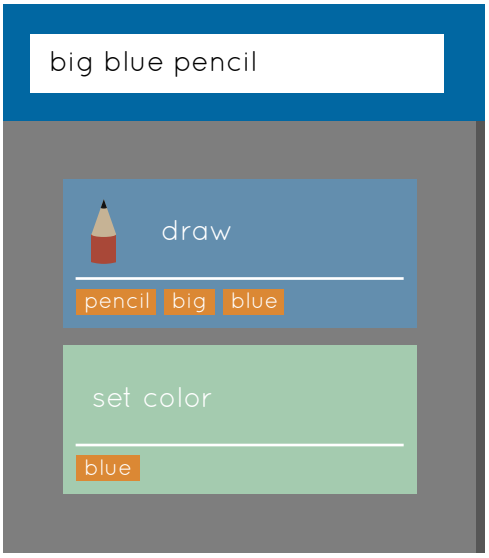


fig.30 Toolbox search met tags

Toolchain
De eigenschappen van de objecten in de omgeving worden bepaald door de "Toolchain". De toolchain is een aaneenschakeling van tool-modules. (fig 31)

Er bestaan 3 verschillende soorten toolmodules, die achtereenvolgens in de toolchain geplaatst kunnen worden.

Een generator is een bron. Een tool die data plaatst of genereert in de ontwerpomgeving. Hier onder kunnen we onderandere verstaan: Tekeningen, geometrische vormen, afbeeldingen, componenten, 3d modellen. Generatoren worden omschreven met zelfstandignaamwoorden.

Een modificator is iets wat data neemt en de data bewerkt om er iets anders van te maken. Bijvoorbeeld om van een 2d vlak een 3d artefact te maken, of om geometrieën te vervormen. Hieronder vallen operaties als: extruderen, uitrekken, filteren, buigen, knippen, verbinden, vermenigvuldigen. Een modificator wordt benoemd met een werkwoord.

Een simulator is iets wat data analyseert, en er andere gegevens uit opmaakt over eigenschappen van het object. Hieronder vallen simpele berekeningen van prijs, of gewicht. Maar ook stijfheid & sterkte, statica en dynamica, FEA(Finite Element Analysis).

Een chain heeft minimaal en maximaal één generator. Daaronder kunnen er een ongelimiteerd aantal modificatoren en simulatoren geplaatst worden.

Het toolchain principe benadrukt het verloop van de tijd in het ontwerpproces door duidelijk te laten zien hoe een artefact opgebouwd is: van boven naar beneden. (fig 32) Bovendien staat de toolchain toe eigenschappen van eerder geplaatste modules aan te passen. Hierdoor kan de gebruiker non-lineair te werk gaan en eerder gemaakte beslissingen bijschaven. Zo kan de gebruiker in een toolchain op een laag niveau parameters aanpassen, en meteen op een hoog niveau feedback krijgen over zijn beslissingen. Als de gebruiker in een eerder geplaatste module een parameter aanpast, wordt de hele toolchain opnieuw doorlopen waardoor het gehele ontwerp dynamisch wordt aangepast. Dit kan gebruikt worden om in real time vormen te genereren (zie ook bijlage E).

Het zorgt er ook voor dat de benodigde GUI elementen alleen zichtbaar zijn op het moment dat ze nodig zijn. Er zijn geen extra panelen, menu's of submenu's meer nodig in de GUI. Alle elementen die van belang zijn zitten in de toolchain modules. Terwijl het ontwerpproces vordert komen er nieuwe modules in beeld en verdwijnen anderen. Dit lost effectief het probleem van feature-creep op (zie ook *Generatie: Tools gegeneraliseerd*), omdat de complexiteit van de interface niet toe neemt, bij een toenemende hoeveelheid functionaliteit. Het programma zou duizenden modules kunnen bevatten, alleen de modules die op dat moment gebruikt worden zijn voor de gebruiker zichtbaar. Zie ook bijlage B & C voor een overzicht van de hele interface.

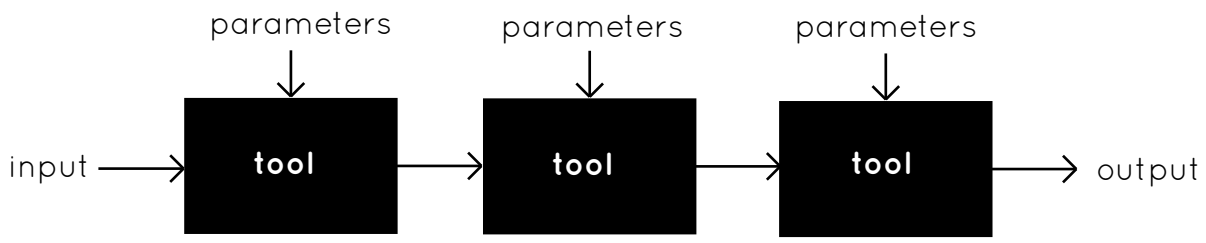


fig.31 Toolchain flow

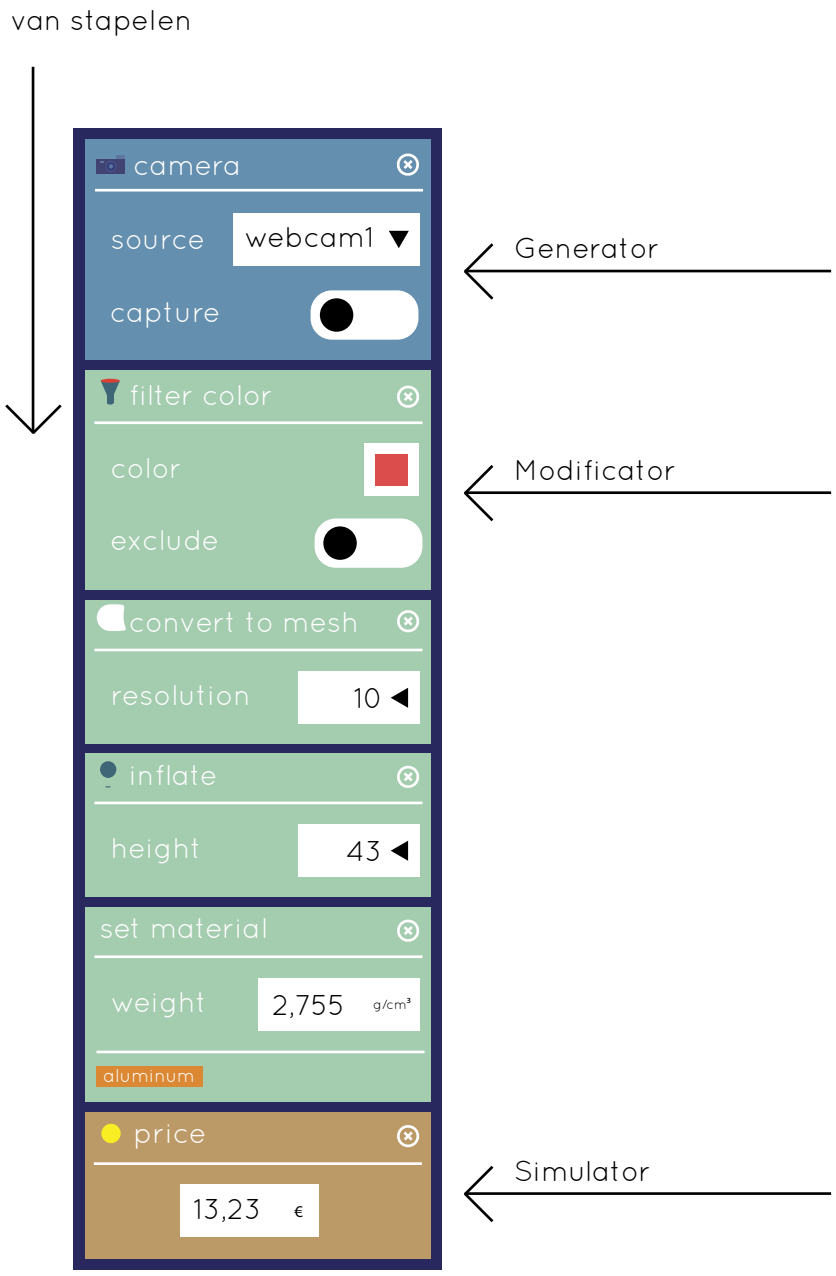


fig.32 Toolchain GUI

Mapping

Om een meer dynamische controle over tools te krijgen, is het mogelijk tools te mappen naar verschillende invoer-apparaten. Voor een verzameling en analyse van de mogelijke invoer apparaten zie bijlage L.

Door bij een bepaalde parameter een mapping menu te open kan de gebruiker kiezen aan welk apparaat hij de parameter wil koppelen (fig 33). Vervolgens kan hij een eigenschap van dat apparaat kiezen (fig 34). Vervolgens is het nog mogelijk om een bereik voor beide waardes in te stellen. Een waarde tussen 0 en 100 in de sensor kan bijvoorbeeld een waarde tussen 50, 230 in de parameter opleveren (fig 35).

Door de mapping helemaal open te laten, wordt een speelse werkwijze bevorderd. Doordat de gebruiker vrij kan experimenteren met verschillende apparaten , parameters en bedieningen kan er makkelijk gezocht worden naar nieuwe intresante combinaties en ideeën. Door externe bedieningen te gebruiken wordt het zoeken van vorm een stuk speelser.

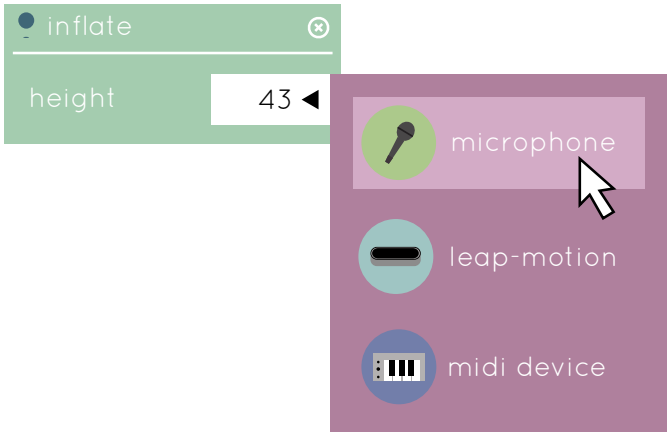


fig.33 Kies apparaat

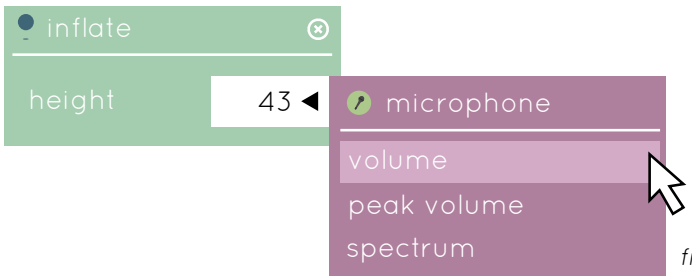


fig.34 Kies parameter

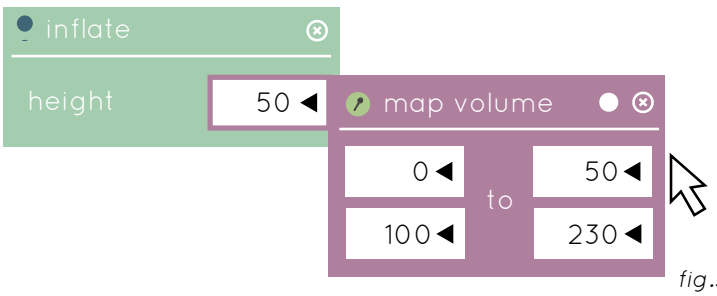


fig.35 Kies range

3d in een multimodale omgeving

De ontweromgeving moet functioneren op verschillende apparaten. Hiervoor is het belangrijk om de interface niet afhankelijk te laten zijn van een bediening die niet overal te krijgen is. Er echter niet altijd een eenduidige bediening mogelijk of zelfs wenselijk, omdat verschillende apparaten nu eenmaal verschillende soorten bedieningen hebben. Er is gekozen om in ieder geval een desktop PC, een laptop, een tablet en een smartphone te ondersteunen. Het belangrijkste verschil tussen deze apparaten is dat een laptop of PC met een muis bediend wordt, en een tablet of smartphone door middel van touch. Bovendien hebben alle apparaten een ander formaat scherm. Een laptop heeft soms een losse muis, of soms een touchpad. Soms heeft hij 3 en soms 2 knoppen. Daarom is besloten om de Grafische Interface zo te ontwerpen dat die met 1 muisknop, of een "touch-vinger" te bedienen is.

In het geval van de 3d ruimte is dit een heel ander verhaal, omdat er sprake is van veel meer vrijheidsgraden. Er zijn er 6 voor de camera(de positie vanwaar de gebruiker kijkt) en 6 voor een object wat op dat moment verplaatst wordt.

De camera kan in ieder geval vanuit de grafische interface bediend worden met behulp van de camera controle knop, in de rechter onderhoek van het scherm (fig 36). Zie voor een overzicht van hoe de interface op verschillende platforms werkt bijlage G.

Er is verder onderzoek nodig naar de verschillende mogelijkheden voor de verschillende apparaten, maar door ook in deze bediening een mapping interface te implementeren, kan dit onderzoek worden uitgesteld, of zelfs helemaal aan de gebruiker worden overgelaten. Individuele gebruikers kunnen de interface aanpassen aan hun eigen omstandigheden. Omdat sommige gebruikers verschillende soorten apparaten zullen hebben. Zie voor uitgevoerde experimenten met verschillende soorten mappings met een leap-motion bijlage K.

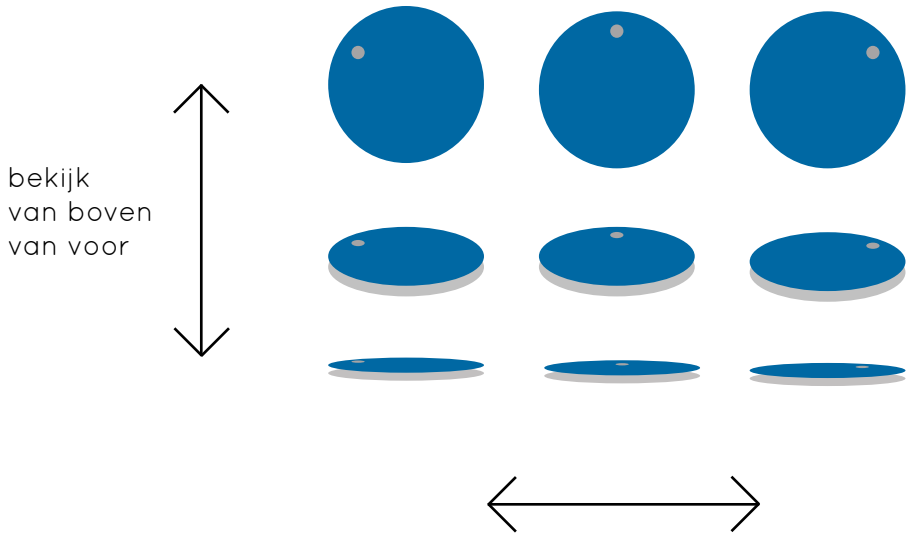
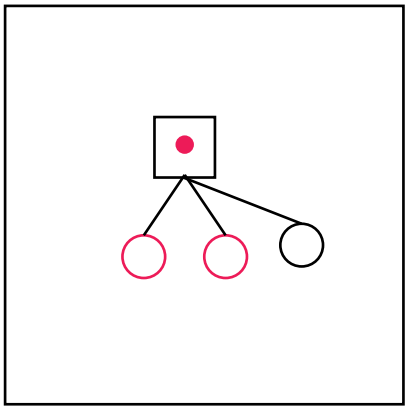


fig.36 3d camera knop

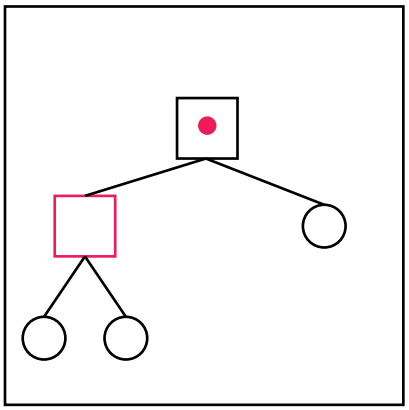
Combineren en navigatie

Twee of meerdere objecten kunnen samengevoegd worden in een nieuwe ruimte. Dit doet de gebruiker door de objecten te selecteren en op de “voeg samen“ knop te drukken (fig 37). Nu zijn de samengevoegde objecten als een geheel door de ruimte te verplaatsen. Om de objecten onderling te kunnen wijzigen kan de gebruiker de pijl naar beneden klikken, om de focus te verleggen naar deze subruimte (fig 38). Om de focus vervolgens weer terug te leggen op de hogere ruimte klikt wordt de pijl omhoog gebruikt.

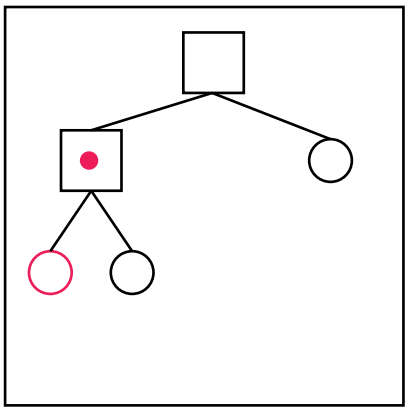
fig.37 Ordenen en navigatie knoppen



Selecteer twee of meerdere objecten, en voeg ze samen in een nieuwe ruimte



Selecteer de nieuwe ruimte en verplaats de focus naar deze ruimte.



Selecteer een object in deze ruimte om hem te verplaatsen of aan te passen

fig.38 Het combinatie proces

Geavanceerd gebruik tools

Open tool module editor

Omdat tool modules loskoppelbare stukjes code zijn. Zou het mogelijk zijn een API(Application Programming Interface) aan te leveren waarmee gebruikers zelf hun eigen modules zouden kunnen ontwerpen en programmeren. Er zou hier een specifieke editor interface voor ontwikkeld kunnen worden, zodat gebruikers makkelijk parameters en GUI elementen kunnen uitkiezen voor hun tool. Daarnaast kunnen ze de tool zo makkelijk in het systeem krijgen met een naam en tags.

Combineren

Naast het zelf schrijven van toolmodules zouden modules door de gebruiker gecombineerd kunnen worden in een nieuwe module die alle functionaliteit van de onderliggende modules omvat. De resulterende module biedt dan functionalitiet op een hoger abstractie niveau, die vervolgens weer met andere gebruikers gedeeld zou kunnen worden.

Recursie en Conditionals

De toolchain is op dit moment een linear systeem. Er is een input en een output. Er zouden echter speciale toolmodules ontwikkeld kunnen worden die complexer gedrag zouden kunnen implementeren. Zo zou er een “op voorwaarde dat“ module kunnen zijn, die alleen bepaald gedrag uitvoert onder een bepaalde voorwaarde. Of er zouden “map” of “range” modules kunnen zijn die bepaald gedrag meerdere malen uitvoeren met verschillende parameters. In essentie is het mogelijk om de toolchain volledig uit te breiden met concepten uit het functioneel programmeren[25]. Hiervoor is echter eerst wat extra onderzoek nodig.

Naast mappings tussen apparaten is zou het ook mogelijk zijn om mappings te maken naar interne parameters om zo parametrische ontwerpen te maken. Bijvoorbeeld door de hoogte van een object te mappen naar de breedte van een ander.

Een Collaboratieve omgeving
De ontwerpomgeving heeft een Client-Server architectuur en ondersteunt collaboratie, doordat er met meerdere mensen tegelijk vanaf verschillende clients aan een ontwerp gewerkt kan worden.

Alle data wordt opgeslagen op de server, het bekijken en bewerken van die data gebeurt bij de clients. De meest up-to-date data is altijd beschikbaar vanaf de server, onafhankelijk van welke clients zijn aangesloten. De Client-Server Architectuur maakt mogelijk dat gebruikers zowel tegelijkertijd als los van elkaar aan dezelfde, of verschillende onderdelen en niveaus van een project kunnen werken. De Server bevat alle data van het hele project, de clients 'zien' alleen dat deel van de data waaraan ze op dat moment aan het werken zijn. Dit kan omdat een client op een bepaalde ruimte in het Ruimte-Object model focust. Een gebruiker kan dus een hele samenstelling van componenten bekijken, tewijl een andere gebruiker aan één specifiek component in het geheel aan het werk is.

Wanneer een gebruiker inlogt in de ontwerpomgeving kiest de gebruiker op welk punt hij in het Ruimte-Model instapt. Die data wordt dan door de client vanaf de server ingeladen. Vanaf dat moment bestaan er dus twee versies van die data, een op de server en een op de client. Het is van belang dat die twee versies synchroon gehouden worden. Dat betekent dat telkens wanneer de gebruiker iets aanpast bij de client, die aanpassing naar de server moet worden doorgestuurd. De aanpassing moet dan in de data aan de serverkant ook plaatsvinden. Bovendien moet de server naar de client toe bevestigen dat die aanpassing ook daadwerkelijk heeft plaatsgevonden. Zo weten beide kanten dat ze nogsteeds synchroon lopen. Ingewikkelder wordt het wanneer er meerdere clients verbonden zijn die dezelfde data bekijken en bewerken. Er bestaan nu meerdere versies van dezelfde data op meerdere clients. De server moet zorgen dat de data over alle clients synchroon is. Telkens wanneer de server een wijziging van een client krijgt, stuurt hij die door naar alle andere clients die die data bekijken. Dat betekent dat de server precies moet bijhouden welke clients welke data kunnen zien. Bovendien moeten ook alle clients aan de server bevestingen dat ze die wijziging ontvangen hebben. Bovendien kan er als extra controle bij elke wijziging een unieke opeenvolgende versie ID meegestuurd worden. Als een client dan een wijziging krijgt die niet logisch opeenvolgt op de vorige wijziging, dan weet die client dat hij niet synchroon meer loopt met de server.

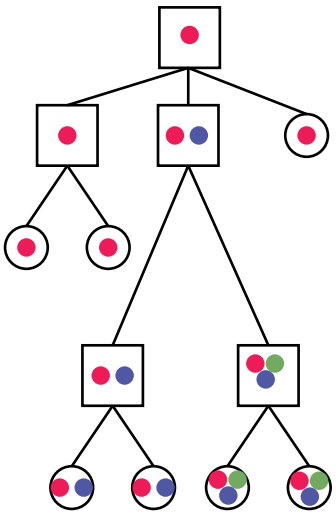
Omdat verschillende gebruikers verschillende delen van het Ruimte-Object model bekijken, betekent dat, dat die verschillende gebruikers ook over verschillende delen updates moeten ontvangen. Het bijhouden van welke Client, over welk deel updates moet ontvangen kan worden opgelost met een Subject-Observer Pattern [22]. Hierbij houdt iedere knoop en relatie in de graaf bij dat hij bekeken wordt door een bepaalde client. Wanneer de data in de knoop of relatie wijzigt stuurt die een update naar de client (fig 39).

Om te voorkomen dat er conflicterende verschillen in data zijn doordat twee mensen tegelijkertijd hetzelfde stukje data aan het bewerken zijn, moet voor iedereen kenbaar zijn dat de data bewerkt wordt. Dit kan met behulp van een Writer lock [24]. Op het moment dat een gebruiker een bepaald object selecteert, wordt dat object voor andere gebruikers 'op slot' gezet. Alleen de 'eigenaar' kan nu de data in dat object bewerken. Als de gebruiker het object deselecteert wordt het weer open voor alle gebruikers.

In de GUI wordt dit weergeven door iedere gebruiker een kleur toe te wijzen, een object dat door een gebruiker geselecteerd wordt, wordt omlijnd door die kleur. Zo is makkelijk te zien, wie er op dat moment het ontwerp aan het bekijken of wijzigen is.

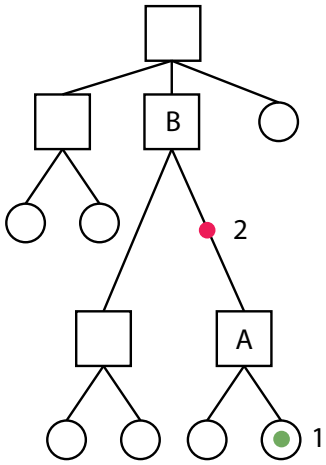
Overigens betekent dit niet dat het grote delen van het ontwerp niet te wijzigen zijn op het moment dat iemand iets selecteert. Zo is het mogelijk dat een gebruiker aan een bepaald component werkt, terwijl een andere gebruiker dit component tegelijkertijd alvast in een grotere assemblage plaatst. Dit kan omdat dit in het ruimte-object model verschillende onderdelen zijn die los van elkaar op slot kunnen worden gezet (fig 40).

In dit concept moet een cliënt ten alle tijde verbonden zijn met de server. Op het moment dat de verbinding verbroken is, worden wijzigingen die de gebruiker uitvoert niet opgeslagen. Daarom is in de interface een indicator in de linker bovenhoek die bevestigt dat er een verbinding is, en die de gebruiker waarschuwt als de verbinding weg valt.



subject-observer pattern
Iedere knoop(subject) houdt bij door wie hij bekeken wordt(observer). Rood bekijkt de gehele graaf, terwijl blauw en groen slechts een deel bekijken.

fig.39



Writer lock
Gebruiker 1 werkt aan een deel van een component A, terwijl gebruiker 2 dat component binnen een grotere samenstelling B verplaatst. Merk op dat gebruiker 2 een lock heeft op een referentie, oftewel de translatie van dat component.

fig.40

Versie-controle

Het kan zo zijn dat een bepaald object in verschillende vormen, met verschillende aanpassingen in verschillende ruimtes bestaat. Daarom heeft de ontwerp omgeving ook een versie controle model. Dat wil zeggen dat er van een bepaald object of ruimte verschillende versies worden opgeslagen. Telkens wanneer een gebruiker een object of ruimte aanpast, slaat het systeem een nieuwe versie op. De gebruiker kan door de verschillende versies heen navigeren. De gebruiker kan terug gaan in het verleden, en opnieuw beginnen vanaf een bepaald punt, zonder dat de andere versie verloren gaat. Zo kan een gebruiker makkelijk aan verschillende iteraties werken van een model. Verschillende gebruikers kunnen ook met hetzelfde object werken zonder dat ze elkaars project beinvloeden. Door een 'afsplitsing' te maken kan een object herbruikt en veranderd worden, zonder dat de staat van het object in het originele project veranderd, die verwijst immers een andere 'tak' van het object.

De versie data wordt opgeslagen in een versie boom (fig 41). Iedere knoop in de boom stelt een verandering aan het object voor. De veranderingen worden relatief opgeslagen, dat wil zeggen dat iedere knoop alleen de data bevat die verschilt tenopzichte van de vorige knoop. Om de volledige dataset te verkrijgen moet je de boom vanaf de oorsprong aflopen, en telkens de verandering toepassen. Een nadeel van deze methode kan zijn dat het relatief lang duurt om de volledige data in te laden. Bovendien is deze methode vrij fout gevoelig, als er perongeluk een knoop wordt overgeslagen breekt de hele dataset. Een niet triviaal detail als de data over een netwerk vezonden moet worden, en niet gegarandeerd kan worden dat ieder bericht ontvangen wordt. Daarom kan er ook gekozen worden om telkens de hele dataset in zijn geheel opnieuw op te slaan(fig 42), dit kost echter significant meer ruimte, zeker als er veel veranderingen plaats vinden. Een derde optie is dan om alleen versies die in gebruik zijn in hun geheel op te slaan, zodat die sneller geladen kunnen worden (fig 43). In eerste instantie wordt gekozen voor de eerste optie. Als bij de implementatie blijkt dat dit te langzaam of onbetrouwbaar is, kan t.z.t. gekeken worden of dit plausibel is.

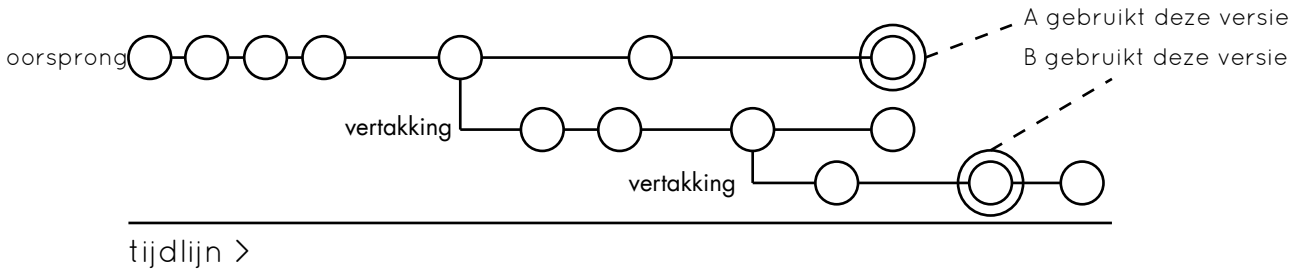


fig.41 Tijdlijn

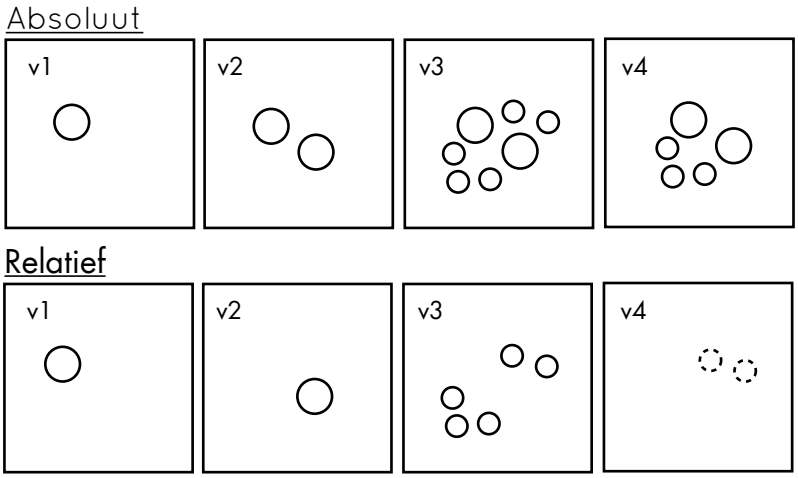


fig.42 Absoluut vs relatief

Drie verschillende manieren om versies op te slaan.
Het getal staat voor het aantal elementen dat wordt opgeslagen

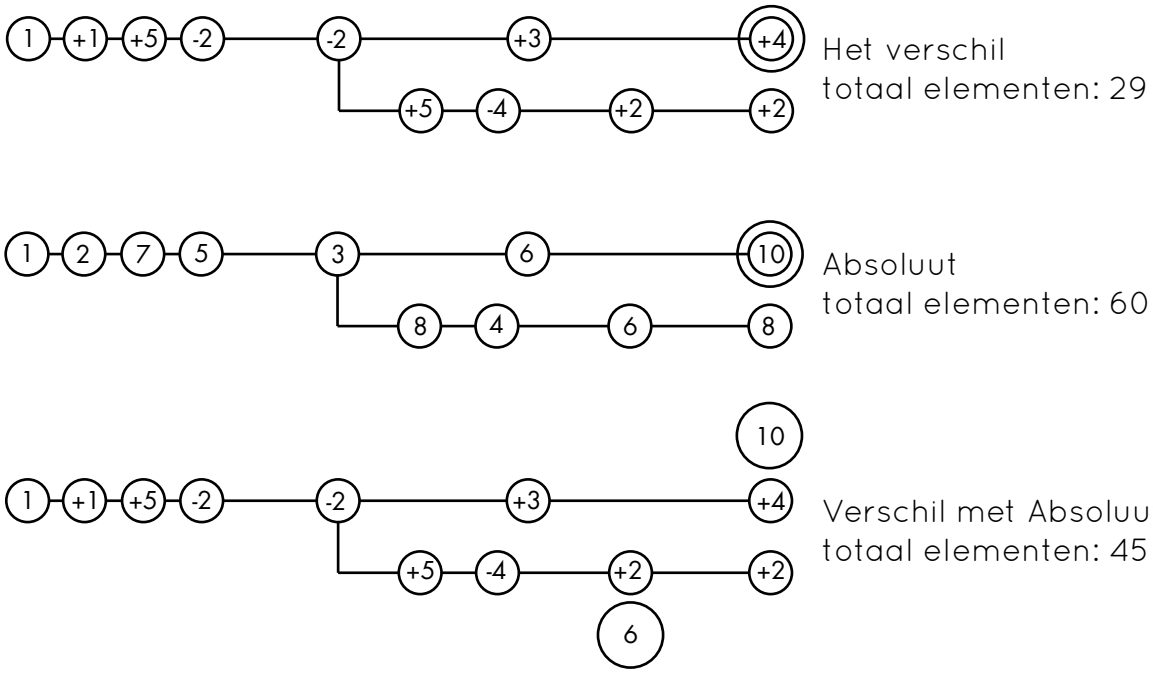


fig.43

Implementatie

Tijdens de Implementatie fase werd een werkend prototype van het ontwerp gemaakt. Het voornaamste nut van deze implementatie is als proof-of-concept van de technische haalbaarheid van het systeem. Verder kan dit prototype op een later moment gebruikt worden om er gebruikstests mee uit te voeren.

Webbased
Er zijn verschillende mogelijkheden voor het implementeren van de ontwerpomgeving. De meest voor de hand liggende keuze zou zijn om de omgeving in een gecompileerde taal zoals C++ of Java te schrijven. Dit omdat een ontwerpomgeving uiteindelijk een behoorlijk zware, en gewichtige applicatie is, en dit nou eenmaal talen zijn die hiervoor geoptimaliseerd zijn. Daarnaast zijn er grafische capaciteiten nodig in de vorm van een grafische bibliotheek zoals OPENGL. Deze optie heeft echter een belangrijk nadeel. De applicatie moet uiteindelijk op verschillende platforms werken, en dat zou betekenen dat voor ieder platforms een 'native' versie gecompileerd zou moeten. In het ergste geval zou het hele programma omgeschreven moeten naar een platform specifieke taal. Daarom is er gezocht naar een alternatief om de applicatie zo toegankelijk mogelijk te maken.

Uiteindelijk is gekozen voor een implementatie in Javascript. Dit is een taal die in een internetbrowser draait. Het voordeel hiervan is dat het met één implementatie op bijna ieder platform draait. Bovendien hoeft de gebruiker niet van tevoren een heel softwarepaket te downloaden en te installeren, maar kan meteen aan de slag. Dit is zeker voordelig wanneer het gaat om collaboratieve projecten, waarbij de drempel hoger ligt om er gebruik van te maken omdat het idee pas werkt als iedereen de software kan gebruiken.

Javascript is echter geen gecompileerde taal, maar een geinterpreteerde taal, dat zorgt ervoor dat javascript iets minder efficiënt is dan bijvoorbeeld C++ of Java. Dit nadeel achten we echter verwaarloosbaar omdat javascript ons daarvoor in de plaats een hoop expressief vermogen en dynamiek terug geeft [10]. Ondanks de noemer "script" is javascript een volwaardige programeer taal.

Bovendien gaat een webbased implementatie standaard uit van een client-server architectuur. Er wordt namenlijk per definitie verbinding gemaakt met de server om de applicatie in te laden.

Open source
Om te voorkomen dat grote delen van het systeem zelf geïmplementeerd zouden moeten worden is er gezocht naar al verkrijgbare sub-oplossingen. Om de kosten van het implementeren van het project zo laag mogelijk te houden is er naar gestreefd zoveel mogelijk Open source projecten te gebruiken.

WebGL & Three.js
Om 3d visualisaties te doen in de browser kan gebruik gemaakt worden van WebGL. Dit is een variant van OpenGL die aangestuurd kan worden door javascript. WebGL is een nog vrij jonge standaard die dus alleen in modernere browsers ondersteunt wordt [11]. Om WebGL aan te roepen maken we gebruik van de Three.js [12] bibliotheek, die een handige abstractielaag vormt over het vrij complexe WebGL. Met behulp van Three.js kan eenvoudig een 3d omgeving geïmplementeerd worden waarin objecten verplaatst en aangepast kunnen worden.

SnapSVG
Om de 2d GUI weer te geven wordt gebruik gemaakt van SnapSVG [13]. Dit is een bibliotheek die interactieve, resolutie onafhankelijke vector afbeeldingen kan weergeven in de browser. Het biedt een interface tussen javascript en het SVG(Standaard Vector Graphics) bestands formaat.

Node.js
De server wordt geimplementeerd met behulp van Node.js [14] Dit is een framework waarmee de server in javascript geimplementeerd kan worden. Zowel client als server zijn dus in de zelfde taal geimplementeerd, waardoor functionaliteit heel eenvoudig tussen de twee uitgewisseld kan worden. Hierdoor kan vrij laat besloten worden of bepaalde functionaliteit uiteindelijk op de server of op de client uitgevoerd moet worden. De Node.js server kan in productie gedraaid worden op een linux distributie zoals Ubuntu [15].

Sockets.io
In onze applicatie is het nodig dat de client informatie kan opvragen bij de server, en dat de server informatie naar de client kan sturen.

Een veel gebruikte manier om te communiceren tussen client en server in web applicaties is XMLHTTPrequest[16]. Bij deze methode vraagt de client aan de server om nieuwe informatie. Dit is echter een Stateless protocol. Hiermee wordt bedoeld dat alle 'sessie informatie' opgeslagen wordt bij de client, en de server verder niet bijhoudt wat een client doet. De client neemt als het ware alleen maar contact op met de server. De server geeft alleen antwoord, maar neemt nooit zelf contact op met een client.

Hiervoor kan gebruik gemaakt worden van WebSockets [17]. Dit is een full-duplex protocol. Hiermee kan dus volledig in twee richtingen gecommuniceerd worden. Als er eenmaal een verbinding is gemaakt kunnen zowel client als server op eigen innitiatief data naar elkaar toe sturen.
Dit is nodig omdat de client niet alleen data opvraagt van de server. De server moet een client ook live op de hoogte brengen van wijzigingen.

We maken in dit geval gebruik van Socket.io [18] Een implementatie van websockets voor node.js en javascript.

Neo4J
Om de ontwerpdata op te slaan op de server is er een database nodig. Dit zou kunnen in een klassieke relationele database zoals een MySQL database[19]. Maar omdat de ontwerpdata specifiek als graaf is gedefiniëerd is er gekozen om de data de graaf-database Neo4J [20] op te slaan. Deze database is ontworpen voor grafen, en is daardoor efficiënter dan een relationele database in het opslaan en weergeven van zulke data. Waar bij een relationele database een meervoud aan requests gebruikt zou moeten worden, kan bij een graaf-databse een groot deel van de graaf in een keer ingeladen worden.

Model - View - Presenter

De implementatie van het prototype maakt gebruikt van modularisatie. Dat wil zeggen dat de code opgedeeld wordt in verschillende, loskoppelbare modules met ieder een eigen specifieke functionaliteit. De onderliggende functionaliteit in zo'n module werkt los van de andere modules.

De modules zijn opgedeeld op basis van het "Model - View - Presenter" patroon [21]. Waarbij de functionaliteit als volgt opgedeeld wordt: (fig 44)

Model is het deel van de code dat alle data bijhoudt. Hieronder vallen het Ruimte-Object model, alle eigenschappen van objecten, de toolchains, alle tools en alle gebruikers.

View is het deel van de code die de data op het scherm laat zien. Het zet de informatie in Model om in een 3d vizualizatie, of in 2d grafische elementen.

Input is het deel van de code die gebruikers input neemt uit bijvoorbeeld muis en toetsenbord en die doorgeeft aan de view. Met behulp van deze module kan de gebruiker interacteren met het systeem en de data aanpassen.

Presenter functioneert als een brug tussen view en model. Het zet de data uit het model om in data begrijpbaar voor de view. Daarnaast neemt het aangepaste data, de input uit view, en zet die om naar begrijpbare data voor het model.

Network is het deel van de code die ervoor zorgt dat het model up to date blijft. Dit is het deel dat met de server communiceert en berichten van en naar de server omzet in een coherent model.

Door deze indeling wordt inmenging van functionaliteit voorkomen. View is een 'domme' module die in principe niets over van de daadwerkelijke uitvoering of aard van het programma weet, dit gebeurt allemaal in het model. Aan de andere kant weet het model niets over de visuele representatie van de data. Dit zorgt voor een sterke ontkoppeling waardoor de code over het algemeen beter te onderhouden, te begrijpen, en uit te breiden is.

In deze architectuur moet veel data tussen modules heen en weer gestuurd worden. Om te voorkomen dat iedere module synchroon gehouden moet worden, doordat iedere module als het ware moet wachten op een antwoord van andere modules voor zij verder kunnen, word veelvuldig gebruik gemaakt van Asynchrone callbacks [23]. Dit is een in javascript veelgebruikte manier om ervoor te zorgen dat een programma niet vastloopt omdat onderlinge modules op elkaar moeten wachten. Het zorgt ervoor dat de gebruiker vloeiend kan blijven interacteren met de omgeving, ondanks het feit dat een bericht van de server er een tijdje over doet om aan te komen.

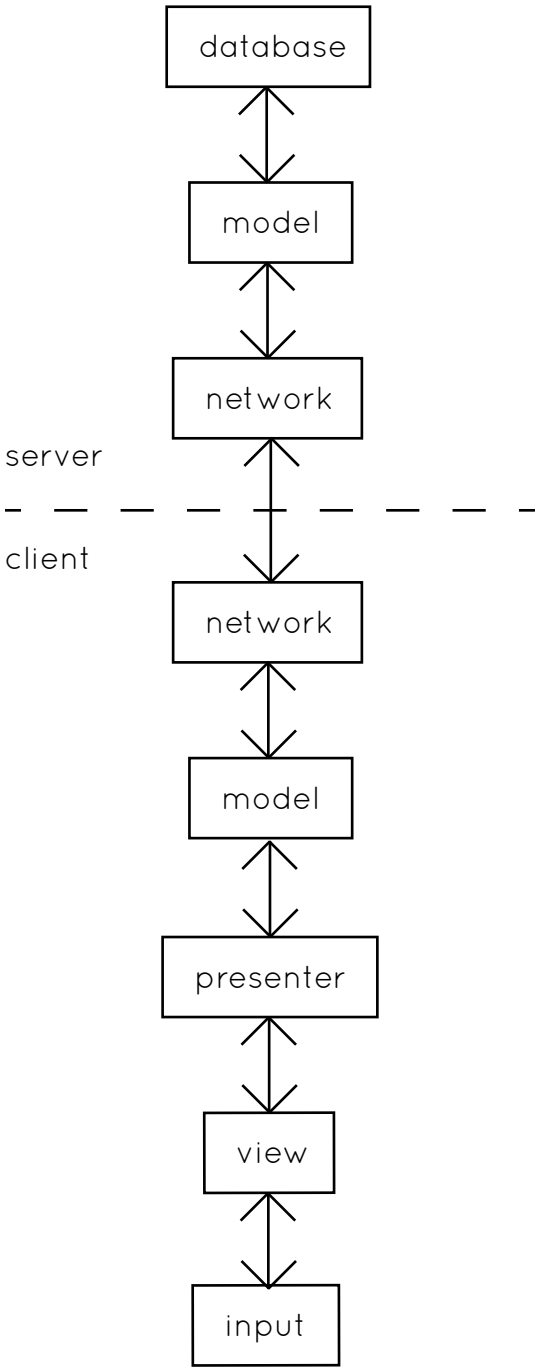

database
model
network
server
client
network
model
presenter
view
input

fig.44 MVP

Model

Het model implementeert een aantal klassen (fig 45). De belangrijkste daarvan is die voor het Ruimte-Object Model. Het heeft een structuur om de graaf in op te slaan met behulp van twee sub-classen “element” en “referentie”. Bovendien heeft het functionaliteit om de data in die classen bij te werken. Dit wordt gedaan met behulp van het observer pattern [22] waarbij zowel de netwerk module, als de presenter module op de hoogte gehouden worden van wijzigingen aan de data.

De element klasse houdt op zijn beurt weer data bij over ruimtes en objecten. De referentie klasse houdt data bij over tussen welke twee elementen de referentie ligt, en over de relatieve translatie die de referentie aangeeft.

Objecten hebben een aparte subclasse voor de toolchain, die data bijhoudt over welke tools in welke volgorde in de toolchain geplaatst zijn. Deze klasse is uiteindelijk verantwoordelijk voor het bijhouden van de eigenschappen van een object. Hij zorgt ervoor dat wanneer er een parameter verandert, de hele toolchain opnieuw doorlopen wordt om de uiteindelijke (geometrische) eigenschappen van een object vast te leggen.

De toolchain bevat dus een lijst met tools. De tool klasse bevat de daadwerkelijke code die uitgevoerd wordt door een tool module, de parameters en hun waardes en de in- en uitvoer data. Voor iedere eigenschap die een object kan hebben is er een standaard formaat, de belangrijkste daarvan zijn type, vertecies, faces en color. Dit is nodig om vloeiend data van de ene naar de nadere tool door te kunnen geven in de toolchain. Een tool kan immers alleen valide output garanderen als de input gegarandeerd valide is. Niet alle tools passen na elkaar, een extrude kan bijvoorbeeld alleen op een 2d vlak uitgevoerd worden, niet op een 3d vorm. Daarom geeft een tool ook altijd een type als output. Er zou ook telkens analytisch bepaald kunnen worden of de tool valide input krijgt, maar dit kost overduidelijk meer rekenkracht en is in de meeste gevallen overbodig.

Tool code wordt asynchroon ingeladen via de toolloader klasse. Het programma heeft in eerste instantie van geen enkele tool de broncode. Die wordt pas ingeladen als de gebruiker die tool ook daadwerkelijk wil gebruiken. Dit is mogelijk omdat javascript een geinterpreteerde taal is, waardoor niet alle broncode persé direct ingeladen hoeft te worden. Het voordeel hiervan is dat het het programma behoorlijk lichtgewicht maakt. Er vanuit gaande dat een substanteel deel van de totale broncode uiteindelijk uit code voor tools zal bestaan wordt een hoop inlaadtijd bespaard door dat zoveel mogelijk in te perken. In collaboratie projecten zal een groot deel van de objecten die door een persoon gemaakt zijn nooit door anderen worden gewijzigd. Zolang een gebruiker aan een bepaald object zelf niets verandert, hoeft die ook de tool-code niet te hebben. Dat betekent dat we doormiddel van de toolloader de omvang van het programma tot een minimum kunnen beperken zonder ooit aan functionaliteit te hoeven inleveren.

Tools inladen gebeurt over het netwerk. De toolloader houdt bij welke toolcode al eerder is ingeladen om te voorkomen dat code dubbel ingeladen moet worden. Als de code voor een bepaalde tool nog niet ingeladen is vraagt de tool-loader via de network module deze code bij de server op.

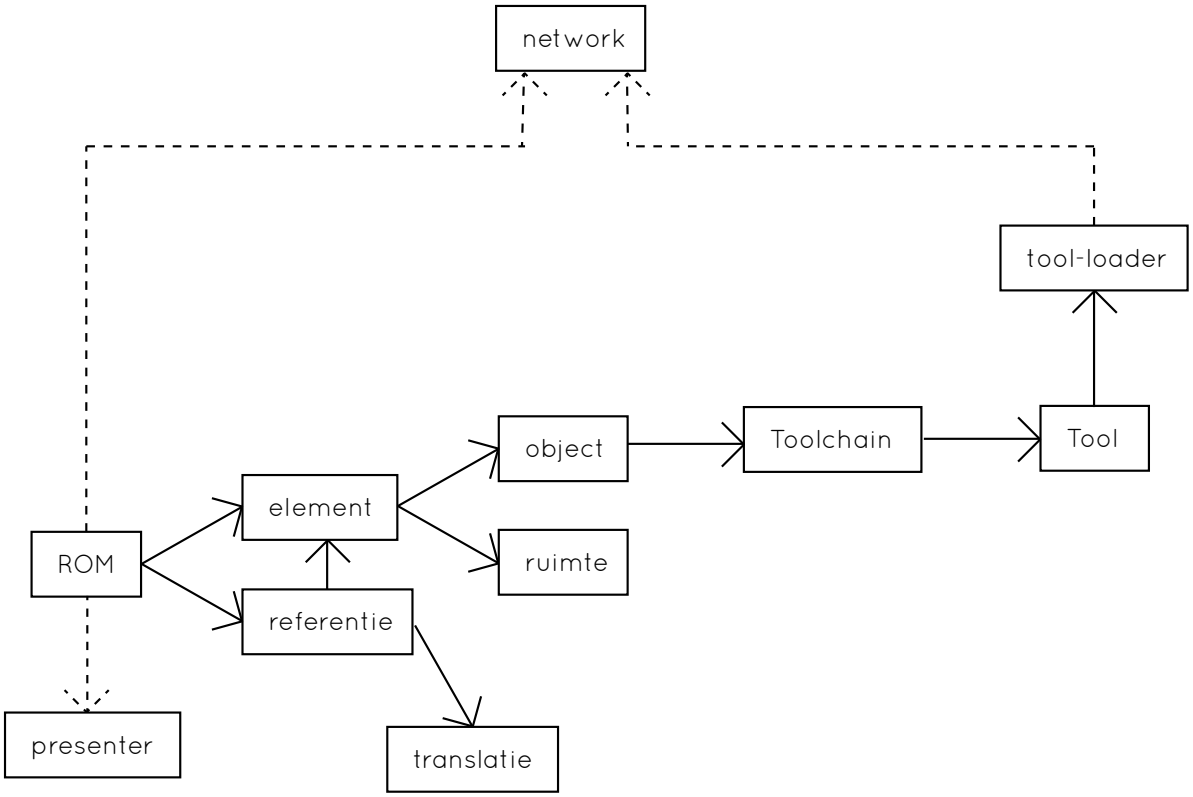

network
tool-loader
object
Toolchain
Tool
element
ROM
ruimte
referentie
presenter
translatie

fig.45 Model

View

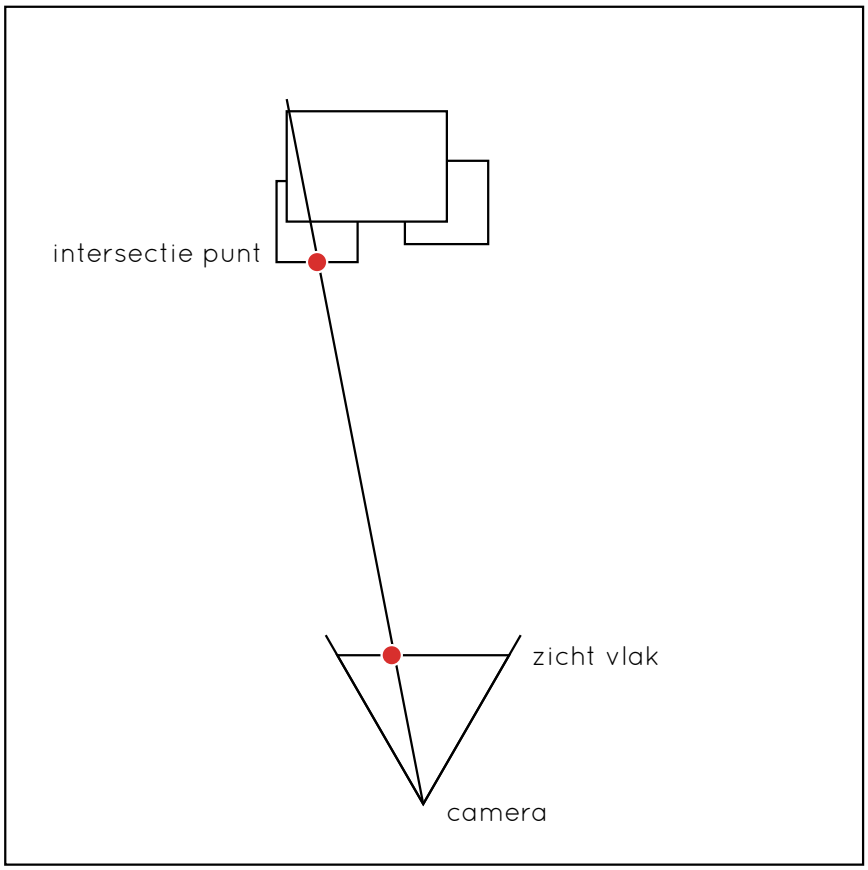
In de view module wordt een onderscheid gemaakt tussen de 2d view en de 3d view. De klasse voor de 3d view maakt gebruik van three.js voor het weergeven van de ontwerpdata. Deze klasse heeft functies voor het instellen van de camera positie, de belichting en schaduwen. De belangrijkste functie is echter die voor het renderen van de objecten. Hiervoor wordt data uit het Ruimte-Object model omgezet in en 3d model in three.js, door de “vertecies”, “faces” en “color” die als output uit de toolchain van een object komen te gebruiken. Vervolgens wordt de absolute positie, rotatie en schaal van een object berekend aan de hand van de relatieve translaties.

De 3d view klasse heeft bovendien nog een functie voor het projecteren van een 2d positie in het scherm naar een vector in de 3d ruimte. Dit is nodig zodat de gebruiker objecten kan selecteren. De gebruiker kan namenlijk alleen op het 2d vlak van zijn scherm een positie aangeven. Om er nu achter te komen op welk object in de 3d ruimte de gebruiker klikt moet aan de hand van de camera positie een lijn geprojecteerd worden de ruimte in. Deze lijn kan gebruikt worden om intersectie met 3d objecten de controleren. (fig 46)

Een laatste functie die de 3d view heeft is om geselecteerde objecten met een kleur te “highlighten“.

De 2d view tekent alle andere grafische elementen op het scherm. Hieronder vallen alle menu’s en knoppen. De 2d view geeft ook de gegevens van het geselecteerde object weer, waaronder de toolchain. De 2d view genereert de tool modules met behulp van de tool parameters uit de toolchain. Wanneer de gebruiker een parameterwaarde aanpast stuurt de 2d view deze aangepaste waarde naar de presenter die ervoor zorgt dat de waarde in het model aangepast wordt. In het model wordt een nieuwe geometrie gegenereerd in de toolchain. Uiteindelijk wordt de nieuwe geometrie in de 3d view bijgewerkt.

zicht van boven af



zicht vanuit camera

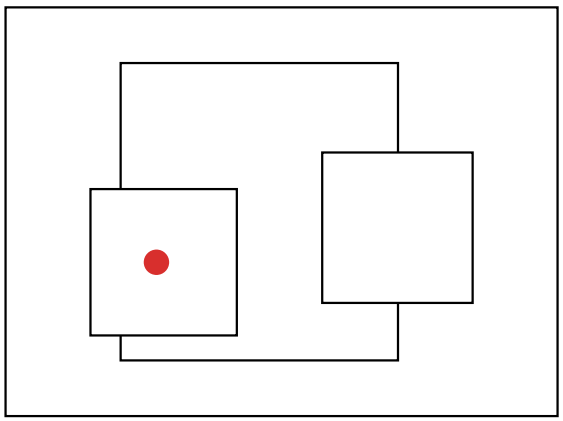


fig.46 2d naar 3d projectie

Input

De input module bevat alle klassen die input innemen van externe apparaten zoals de muis en het toetsenbord: de Input handlers. Deze bieden gestandaardiseerde interface functies aan, die losgekoppeld zijn van enige actie in het programma. Deze functies kunnen worden gekoppeld aan een specifieke actie door middel van een mapping. Een mapping is een klasse die een verbinding maakt tussen een gebeurtenis in de input en een actie in de omgeving. Zo kan bijvoorbeeld het bewegen van de muis gekoppeld worden aan het draaien van de camera. (fig 47)

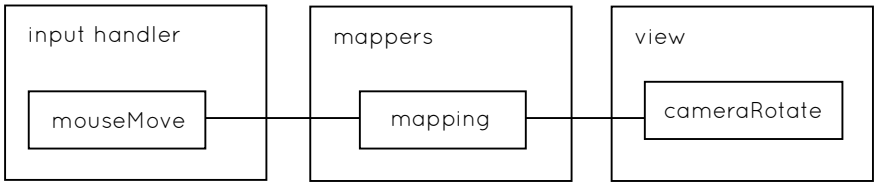


fig.47 Een mapping

Een mapping kan meerdere inputs aan meerdere acties koppelen om zo als geheel een bepaald blok aan functionaliteit te bieden. Zo kan er een mapping zijn voor het bedienen van de camera met de muis, of een andere mapping voor het verslepen van objecten met een leap motion [26]. De mapping koppelt niet alleen de twee elementen aan elkaar, maar omschrijft ook hoe de twee zich verhouden, hoe de input waarden precies omgezet moeten worden naar output waarden. Een mapping heeft daarom ook zelf weer parameters waarmee bijvoorbeeld de gevoeligheid van een beweging ingesteld kan worden.

Network
De network module bevat slechts een klasse die de verbinding met de server faciliteert met behulp van socket.io. De klasse bevat functies om data naar de server te versturen, en event handlers om berichten van uit de server te kunnen verwerken.

De network module geeft data door aan de model module. De code voor de server bevat eveneens een network module, die de data doorgeeft aan een model module in de server. Omdat deze twee netwerk modules elkaar constant updates geven over wijzigingen houden ze de model modules synchroon. Hiervoor gebruikt de server in het model de eerder omschreven observer pattern om bij te houden welke van de clients een update krijgt.

De server heeft een functie om unieke ID's te genereren. Dit is om ervoor te zorgen dat ieder object in het model uniek te identificeren is, en objecten niet perongeluk door elkaar gehaald worden. Als de gebruiker een nieuw object wil aanmaken, moet de model module in de client dus eerst een uniek ID opvragen bij de server. Hierbij wordt gebruik gemaakt van een asynchrone callback zodat de client pas een nieuw object aanmaakt in het model als die een nieuw ID van de server heeft ontvangen.

Database
Als laatste communiceert het model in de server met de database. Data die bewerkt of bekeken wordt door een gebruiker wordt in het tijdelijk geheugen bewaard zodat die snel aangepast kan worden. Pas wanneer alle gebruikers zijn uitgelogd wordt de tijdelijke data permanent opgeslagen in de database en uit het tijdelijk geheugen gehaald. Wanneer een gebruiker een element opvraagt die nog niet in het tijdelijk geheugen is wordt die vanuit de database ingeladen. Dit principe heet caching. Het zorgt ervoor dat de server altijd alleen het minimum aantal elementen in zijn tijdelijk geheugen heeft, en beperkt het aantal (langzamere) berichten van en naar de database tot een minimum. Door deze methode te gebruiken is voor de rest van het software systeem niet zichtbaar of de data uit de database of uit het tijdelijk geheugen komt, er is een eenduidige interface voor het wegschrijven en uitlezen van de data.

Demo

De uiteindelijke implementatie laat zien dat het mogelijk is om in de browser een collaboratieve ontwerp omgeving te maken. Er is een functionerende 3d wereld, die met meerdere gebruikers tegelijkertijd bewerkt kan worden. Er is ook een eenvoudige variant van de toolchain geïmplementeerd. Bovendien zijn er een aantal eenvoudige toolmodules geïmplementeerd om het principe van de toolchain te kunnen demonstreren. Met deze demo zijn een aantal vrije test sessies uitgevoerd met twee en met drie personen samen(fig 48 en 49).

Deze demo implementeert slechts een deel van de uiteindelijke architectuur. Door hierop voort te bouwen kan uiteindelijk het hele programma geïmplementeerd worden. Voor een overzicht van welke delen al geïmplementeerd zijn zie bijlage M.

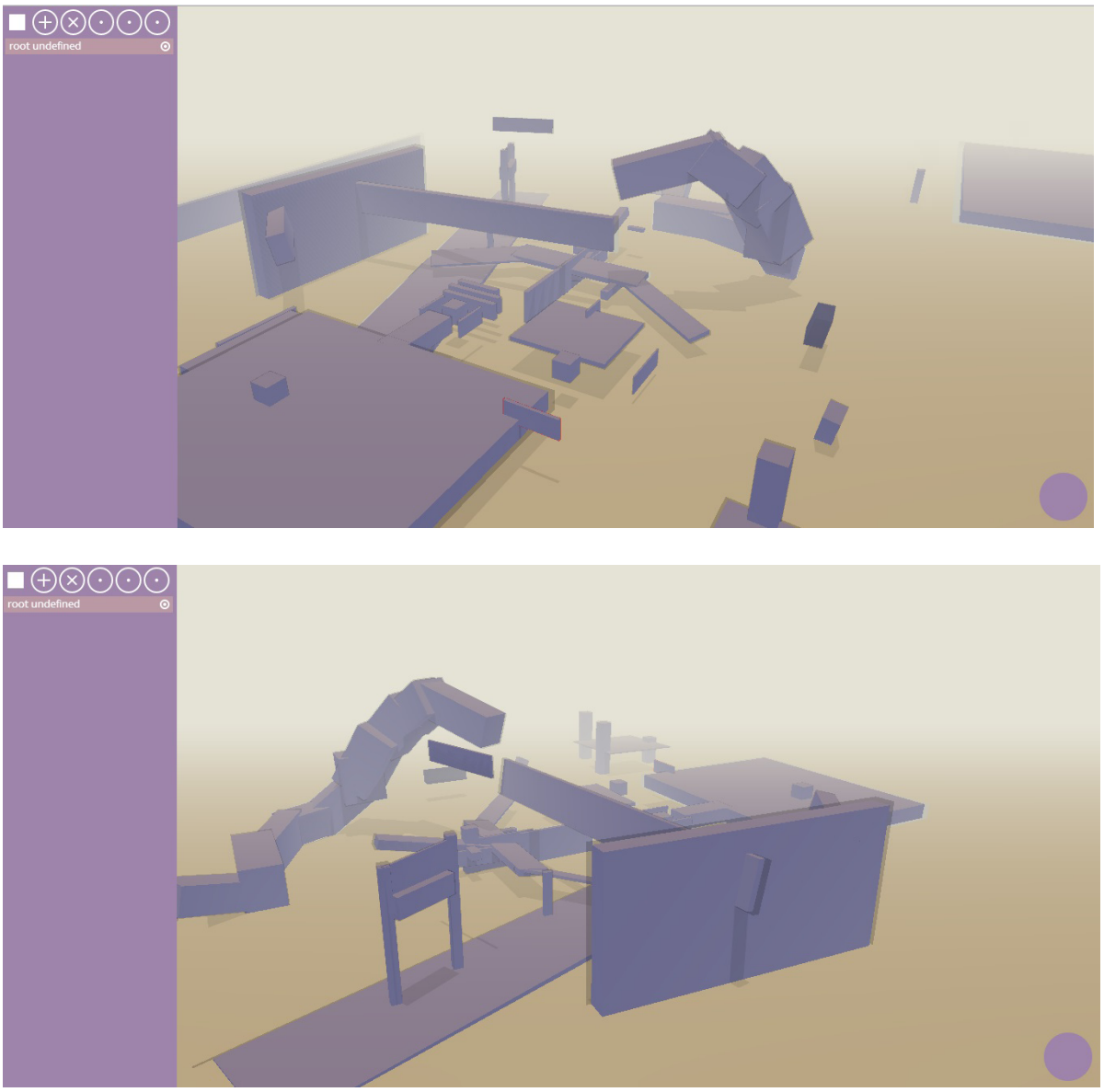


fig.48 resultaten van een uitgebreide sessie in een vroege versie

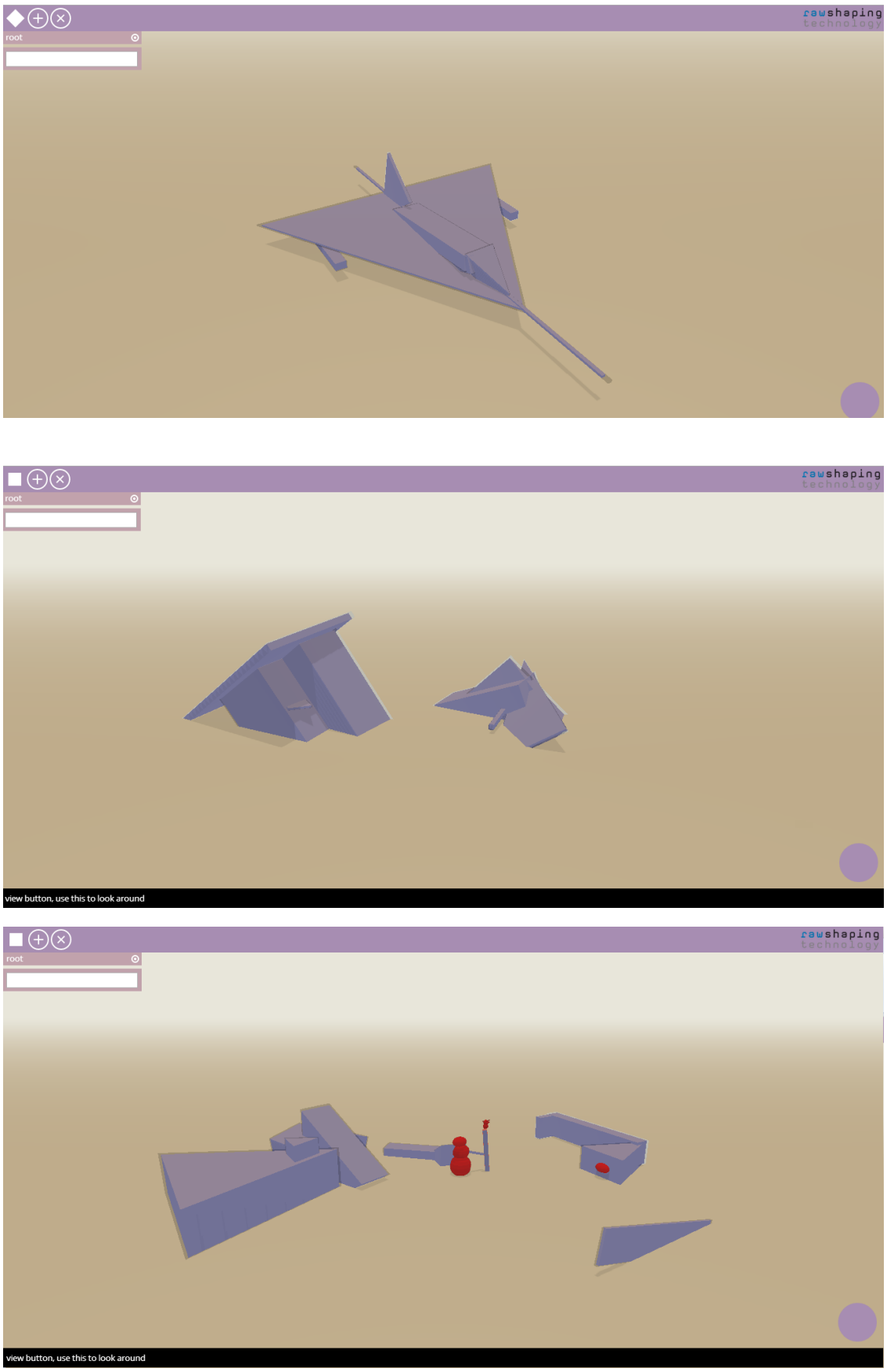


fig.49 resultaten van korte sessies in de nieuwste versie

Conclusie

In dit project is onderzoek gedaan naar het ontwikkelen van een software programma waarmee in 3d op schets-achtige wijze gemodeleerd kan worden in een collaboratieve omgeving. Er is eenliteratuuronderzoek gedaan naar de verschillende aspecten die zo'n omgeving zou moeten hebben. Er experimenten en tests gedaan met verschillende soorten interfaces en bedieningsmogelijkheden.

Aan de hand hiervan is een gebruiksinterface en een systeem architectuur ontwikkeld. Uiteindelijk is van dit ontwerp een prototype geïmplementeerd om als proof of concept te dienen. Het prototype laat zien dat het mogelijk is in de browser een ontwerpomgeving te implementeren die collaboratie via het internet faciliteert. Het prototype laat in simpele vorm zien hoe het principe van een toolchain zou werken.

In de toekomst kan dit protype uitgebreid worden om er een volwaardige ontwerpomgeving van te maken. Hiervoor zullen in zowel in de implementatie als in het ontwerp nog een aantal stappen gemaakt worden. Hiervoor zijn een aantal aanbevelingen te maken:

Gebruikstests
Er zullen met het prototype een aantal kwantitatieve gebruikstests uitgevoerd moeten worden, om te bevestigen dat de principes ook daadwerkelijk functioneren. Zo kunnen er tests uitgevoerd worden om te kijken of de tool search box begrijpelijk is, en hoe de prestaties zijn tenopzichte van een meer traditionele interface.

Onderlinge relaties
Op dit moment zijn objecten alleen te groeperen in ruimtes. Complexere onderlinge relaties tussen objecten zijn niet te definiëren. Het Ruimte-Object model kan worden uitgebreid om dit soort onderlinge relaties wel te ondersteunen.

Inlog interface
De omgeving heeft op dit moment nog geen interface voor het beheren van onderlinge projecten, of het uitnodigen van mensen bij een nieuw project. Er moet worden onderzocht of het nodig is dat mensen registreren en inloggen in het systeem, of dat een groot deel van de omgeving ook zonder te registreren kan functioneren.

3d interface
Er moet verder onderzoek gedaan worden naar een natuurlijke modus van interacteren met 3d objecten.

Mapping
Het prototype ondersteunt op dit moment nog geen mappings. Er moet nog onderzocht worden in hoeverre het concept voor het mappen van apparaten naar parameters in tools nuttig is. Het zou kunnen zijn dat mapping de interface ingewikkelder maakt dan nodig. Hiervoor kunnen uiteindelijk ook kwantitatieve gebruikstests voor worden uitgevoerd.

Recursie en Conditionals
Er is een diepgaand onderzoek nodig naar functioneel programmeren met behulp van de toolchain. Dit zou een zeer krachtige omgeving kunnen opleveren, maar er moet eerst onderzocht worden in hoeverre dit concept haalbaar is.

Versie Controle
Op dit moment is er nog geen interface of implementatie voor versie controle.

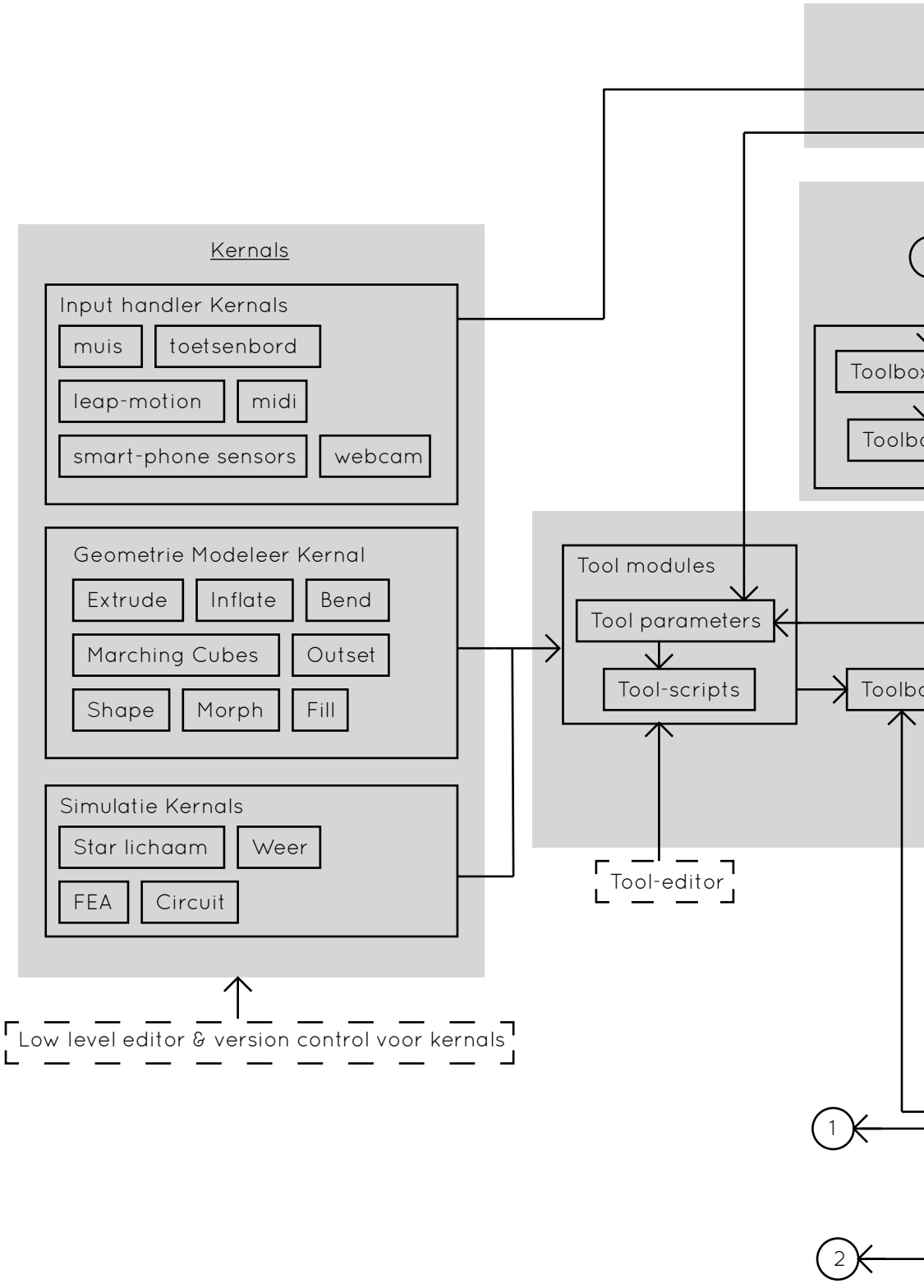
Referenties

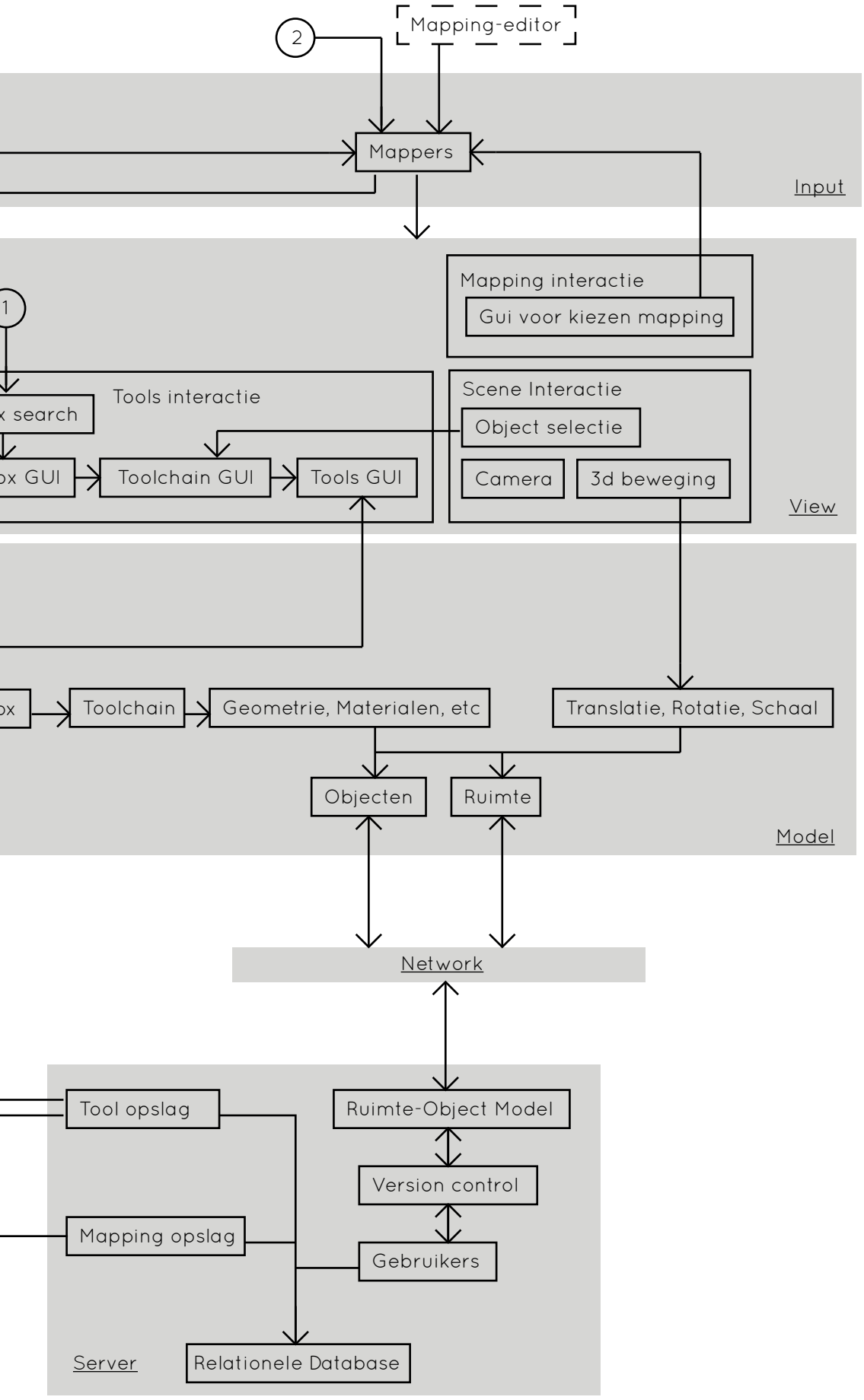
[1] Wendrich, R.E. (2012), Multimodal Interaction, Collaboration, And Synthesis in design and engeneering processing
[2] Sener, B., Pedgley, O. (2008). Novel Multimodal Interaction for Industrial Design, Human Computer Interaction, Ioannis Pavlidis (Ed.), InTech
[3] Norman, D. A. (2002). The design of everyday things. Basic books.
[4] Deleuze, G., Guattari F. (1987). A thousand plateaus: Capitalism and schizophrenia, Introduction: rhizome. , 3-25.
[5] Csikszentmihalyi, M. (2009). Creativity: Flow and the Psychology of Discovery and Invention. HarperCollins.
[6] Baños, R. M., Botella, C., Alcañiz, M., Liaño, V., Guerrero, B., & Rey, B. (2004). Immersion and emotion: their impact on the sense of presence. CyberPsychology & Behavior, 7(6), 734-741.
[7] Alexander, C. Notes on the synthesis of form (1964), The unselfcontious process., 46-54., Harvard University Press
[8] Dennet, D.C. (2012) Intuition pumps and other tools for thinking., Making mistakes., Norton & Company
[9] Winograd, T., (1972) Understanding natural language, Academic Press
[10] Crockford, D., (2001). JavaScript: The World's Most Misunderstood Programming Language, verkrijgbaar op: (http://www.crockford.com/javascript/javascript.html)
[11] WebGL public wiki, (http://www.khronos.org/webgl/wiki/Main_Page)
[12] Three.js (http://threejs.org/)
[13] SnapSVG (http://threejs.org/)
[14] Node.js (http://nodejs.org/)
[15] Ubuntu (http://www.ubuntu.com/).
[16] Holdener, A. T., (2008). Ajax: The Definitive Guide., O'Reilly Media
[17] Websocket (http://www.websocket.org/)
[18] Socket.io (http://socket.io/)
[19] MySQL (http://www.mysql.com/)
[20] Neo4j (http://www.neo4j.org/)
[21] Osmani, A. (2012) Learning Javascript Design Patterns., O'Reilly Media
[22] Martin R. C. (2003) Agile Software Development, Principles, Patterns and Practices, Prentice Hall
[23] Crockford, D. (2008). Javascript: The Good Parts., O'Reilly Media
[24] Grand, M. (2002), Patterns in Java: A Catalog of Reusable Design Patterns., Wiley
[25] Maclennan, B. J. (1990). Functional Programming: Practice and Theory., Addison-Wesley
[26] Leap motion(http://www.leapmotion.com/)

Bijlagen

Systeem architectuur

A

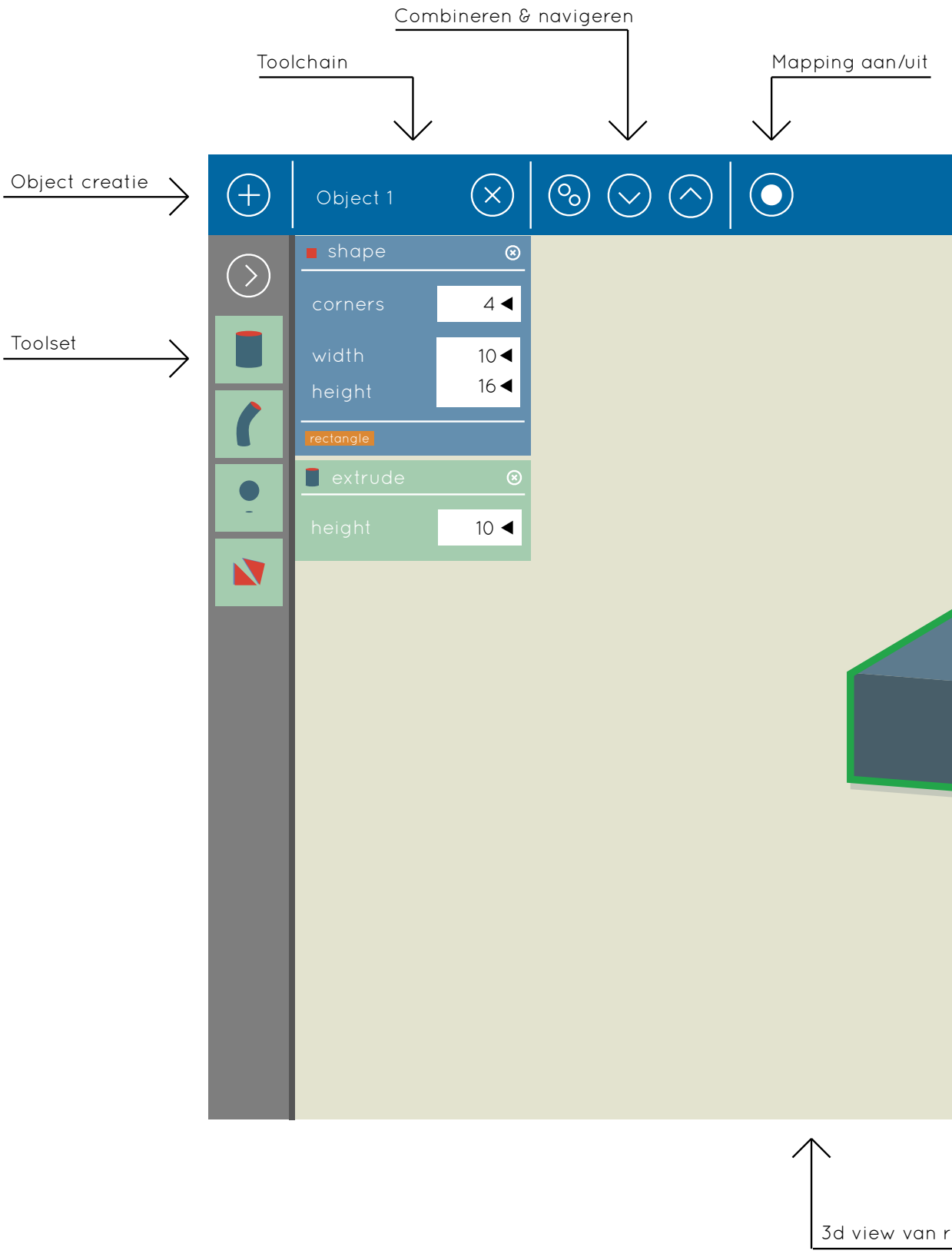



2
Mapping-editor
Mappers
Input
Mapping interactie
Gui voor kiezen mapping
1
Tools interactie
x search
ox GUI
Toolchain GUI
Tools GUI
Scene Interactie
Object selectie
Camera
3d beweging
View
ox
Toolchain
Geometrie, Materialen, etc
Translatie, Rotatie, Schaal
Objecten
Ruimte
Model
Network
Tool opslag
Mapping opslag
Ruimte-Object Model
Version control
Gebruikers
Server
Relationele Database

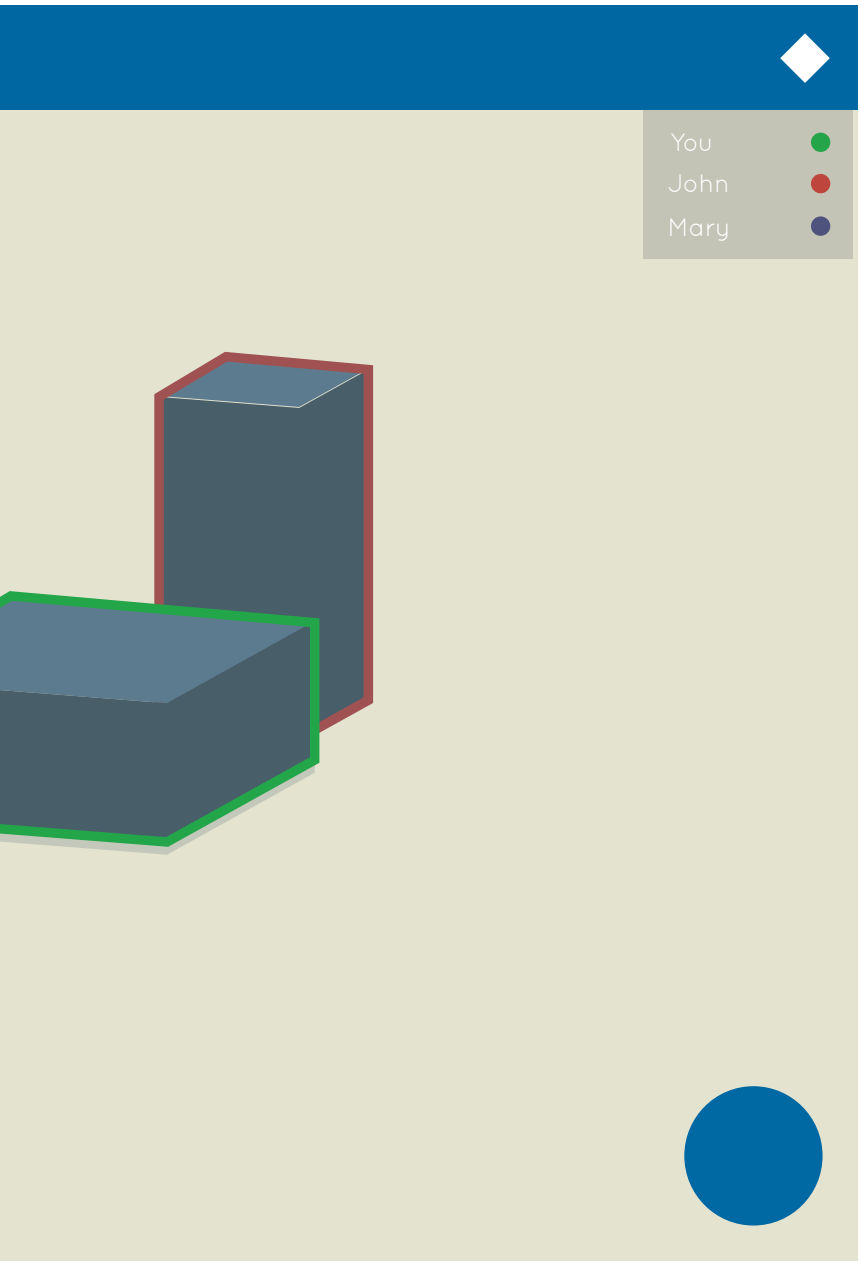
A

Het hoofdscherm

B



B



Verbindings indicator

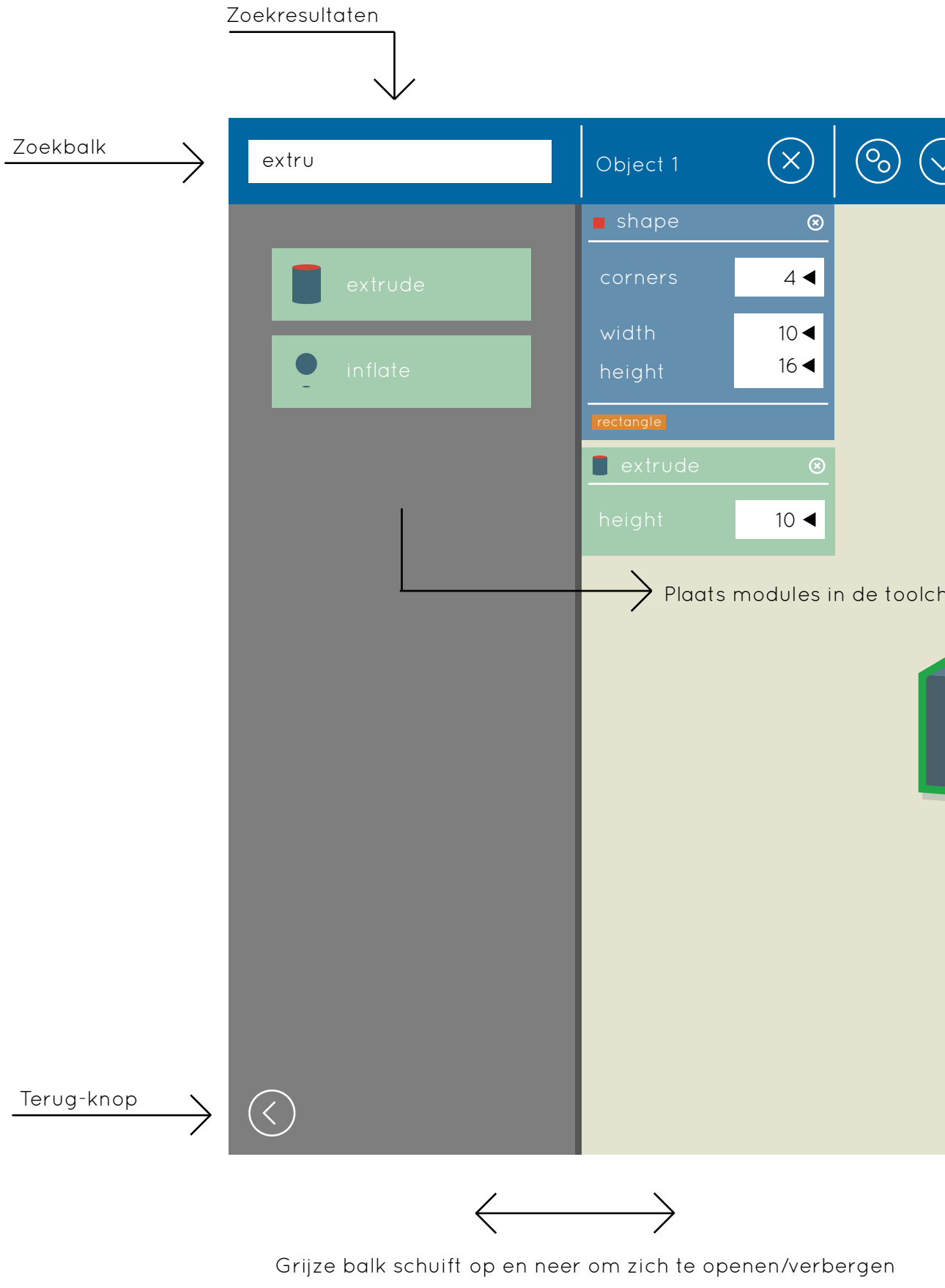
Verbonden collaborators

3d Orientatie

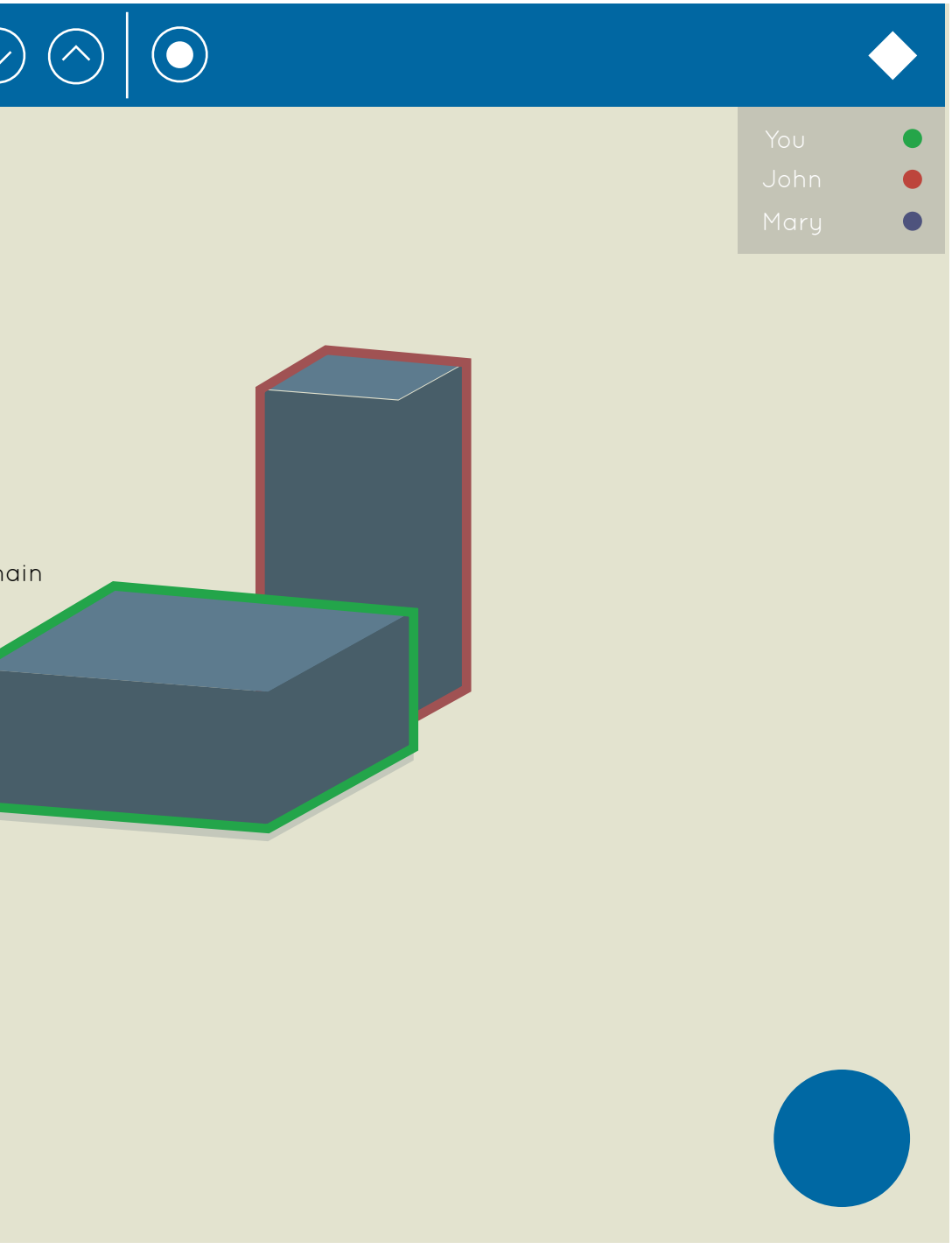
uimte met object

Toolbox search scherm

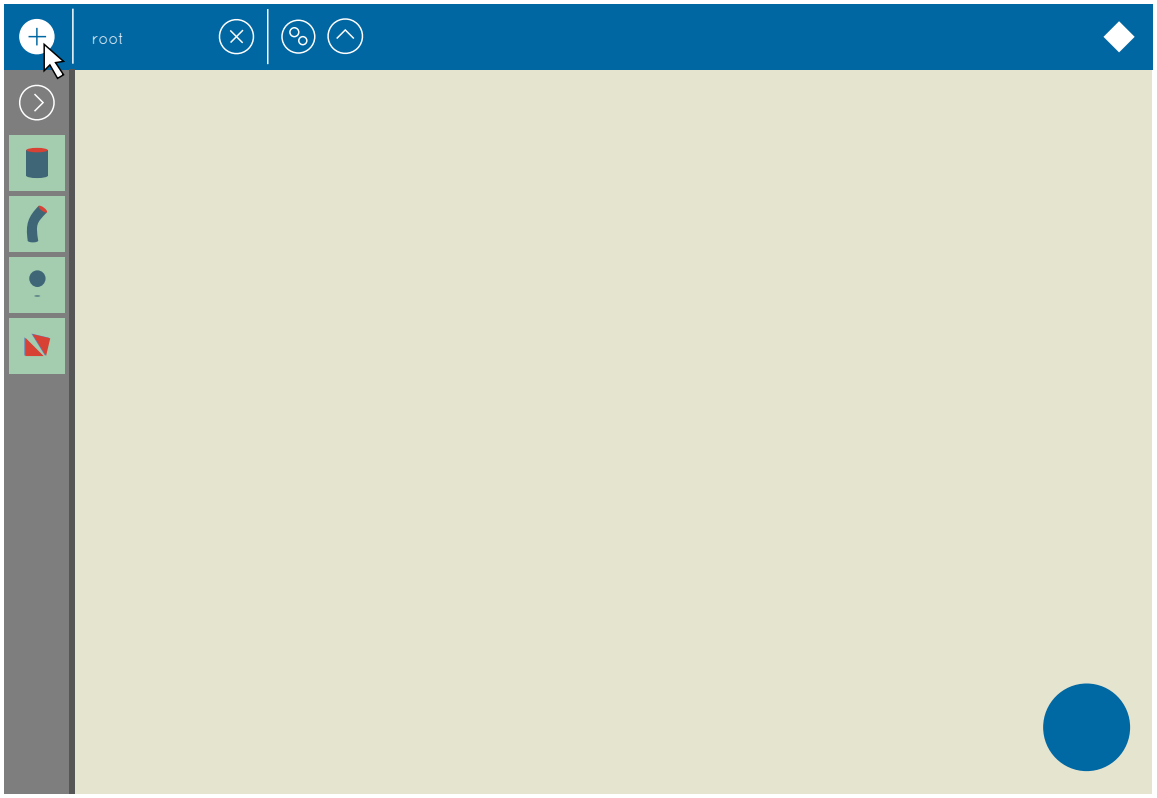
C



C

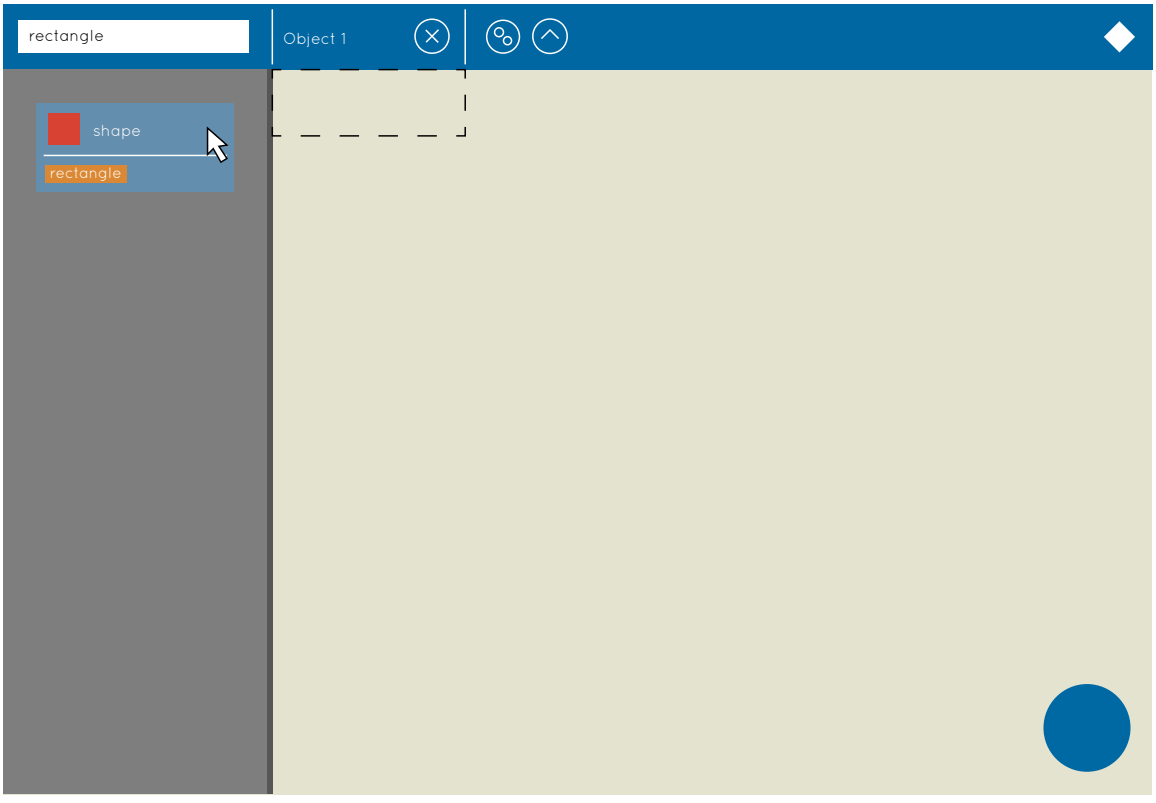


Gebruiksaanwijzing
Klik op de plus knop links bovenin om een nieuw object aan te maken.

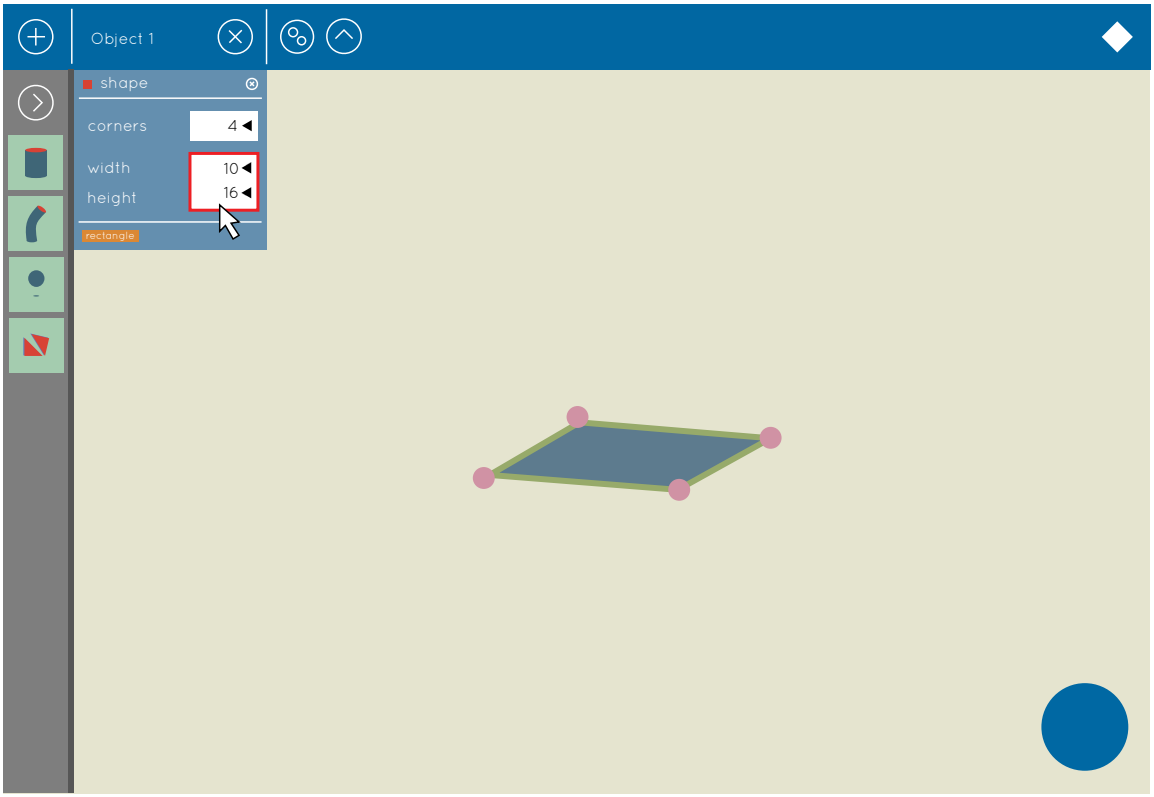


D

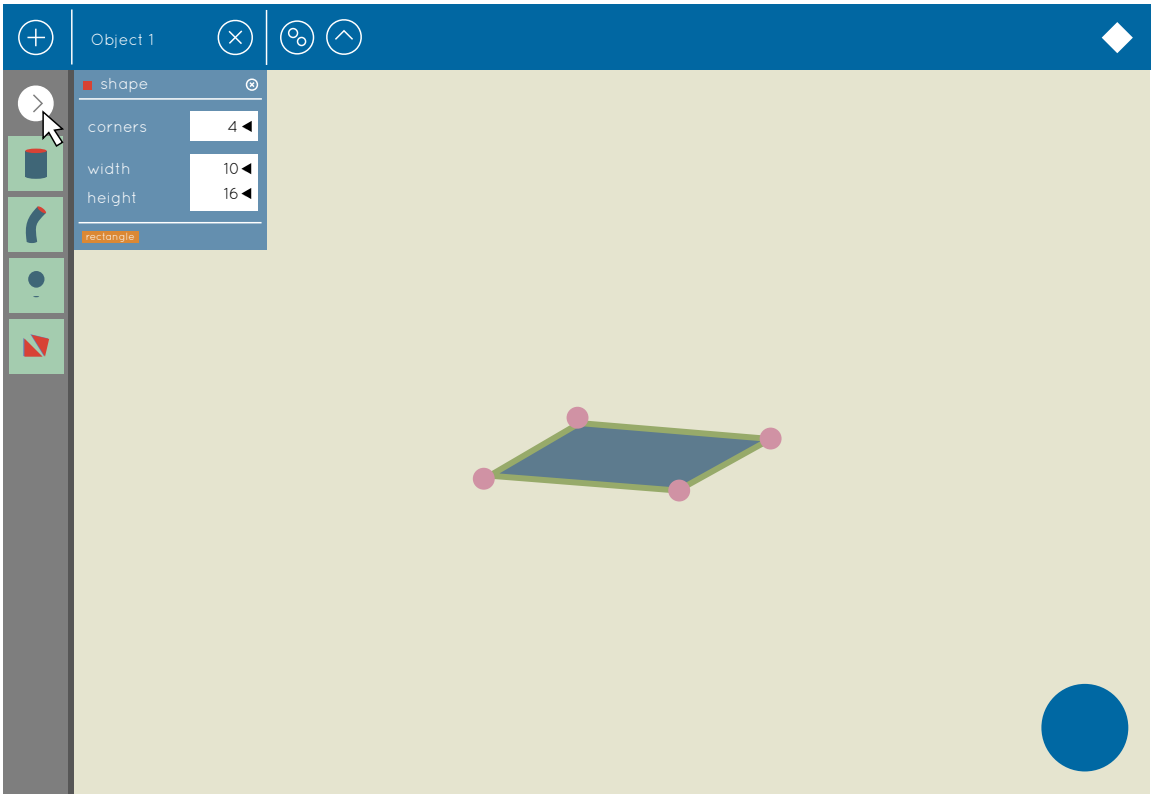
Er opent zich nu de zoekbalk. Gebruik het zoekveld om een object te zoeken, bijvoorbeeld “rectangle“. De resultaten worden weergegeven in het grijze veld eronder. Het resultaat van de zoekterm “rectangle“ is een “shape”-module met een preset “rectangle“. Druk op enter, of sleep de module naar rechts. Om hem te gebruiken.



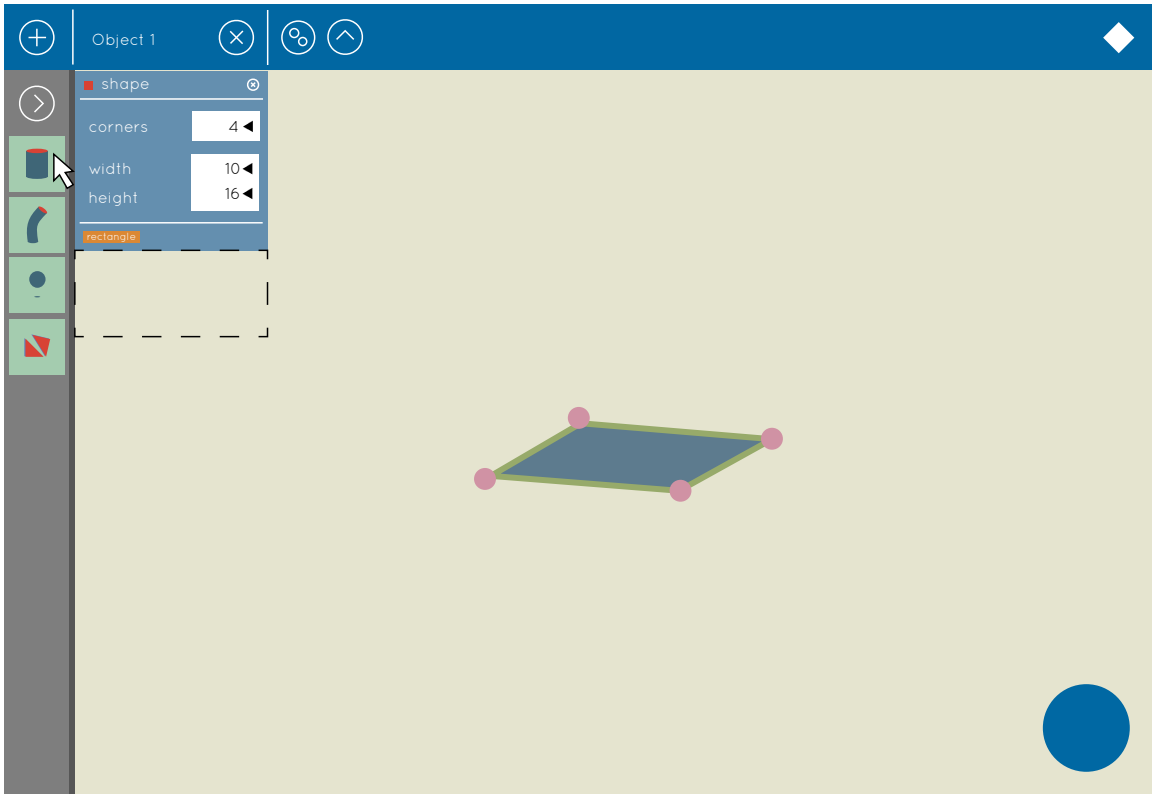
De module verschijnt in de linkerbovenhoek, en de rechthoek verschijnt in de 3d view. Gebruik nu de eigenschappen in de module, of de handles in de 3d view om de rechthoek het juiste formaat te geven.



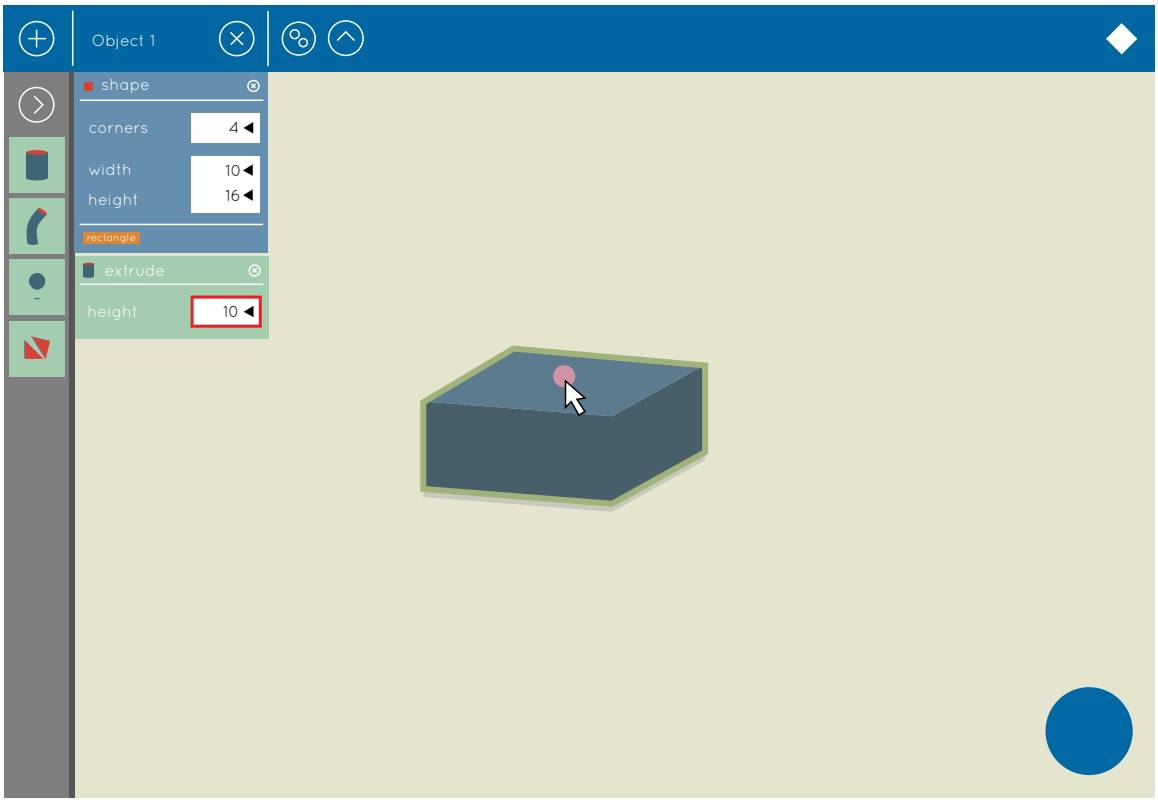
Om verdere bewerkingen uit te voeren klik op de expansie knop om te zoeken, of sleep een van de tools uit de toolset naar rechts.



D



De module verschijnt onder de vorige module. Gebruik zoals eerder, de eigenschappen in de module, of de handles in de 3d view om de bewerking aan te passen.



D

Use case: Shaping met rijst

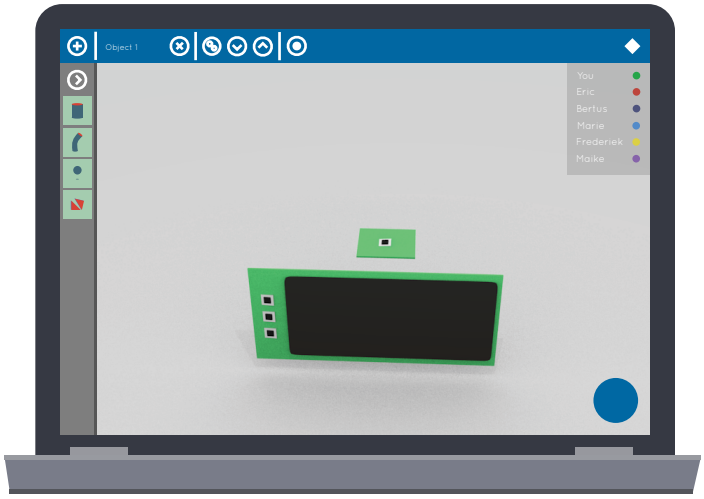
Een ontwerpstudent krijgt de opdracht een nieuwe verpakking te ontwerpen voor een bepaald product. Hij besluit opzoek te gaan naar een intressante vormgeving en dus opent hij het softwarepakket. Hij wil graag iets met zijn handen doen, dus besluit hij een camera module te gebruiken om zijn bureau te filmen. Op zijn bureau gooit hij nu een pak rijst leeg, en gaat in de rijst vormen tekenen. Door een filter module te gebruiken kan hij bepaalde kleuren uit het beeld nemen en zo een specifieke vorm uit het beeld destilleren. Hij gebruikt ook een inflatie module om de 2d vorm in de rijst om te zetten in een mooie ronde vorm op het scherm. Hij koppelt een parameter van de inflatie module aan zijn leap-motion om directer te kunnen interacteren met de vorm. Zo kan hij vormen die hij in de rijst maakt, direct in 3d zien op het scherm. Door een bepaald materiaal aan de vorm toe te kennen, kan hij bovendien direct feedback krijgen over de prijs van de verpakking.


Object 1
camera
source
webcam1
capture
filter color
color
exclude
convert to mesh
resolution
10
inflate
height
43
set material
weight
2,755
aluminum
price
13,23

Use case: Collaboratieve wekker

Een groepje van 6 ontwerpstudenten krijgt de opdracht samen een nieuwe wekker te ontwerpen. Ze gebruiken standaard componenten om samen snel een PCB en een scherm en wat knoppen te modeleren. Dit zijn de componenten die de wekker in ieder geval moet hebben. Nu besluiten ze ieder op basis van deze componenten een concept te maken voor een ontwerp. Dus splitsen ze de ruimte op in 6 sub-ruimtes. Wanneer iedereen klaar is bekijken ze elkaars werk, en overleggen ze over welke concepten de beste ideeën zijn. Ze besluiten een combinatie te maken van twee concepten. Ze gebruiken de elementen uit deze twee objecten om samen tot één nieuw eindontwerp te komen.

F


Object 1
You
Eric
Bertus
Marie
Frederiek
Maike

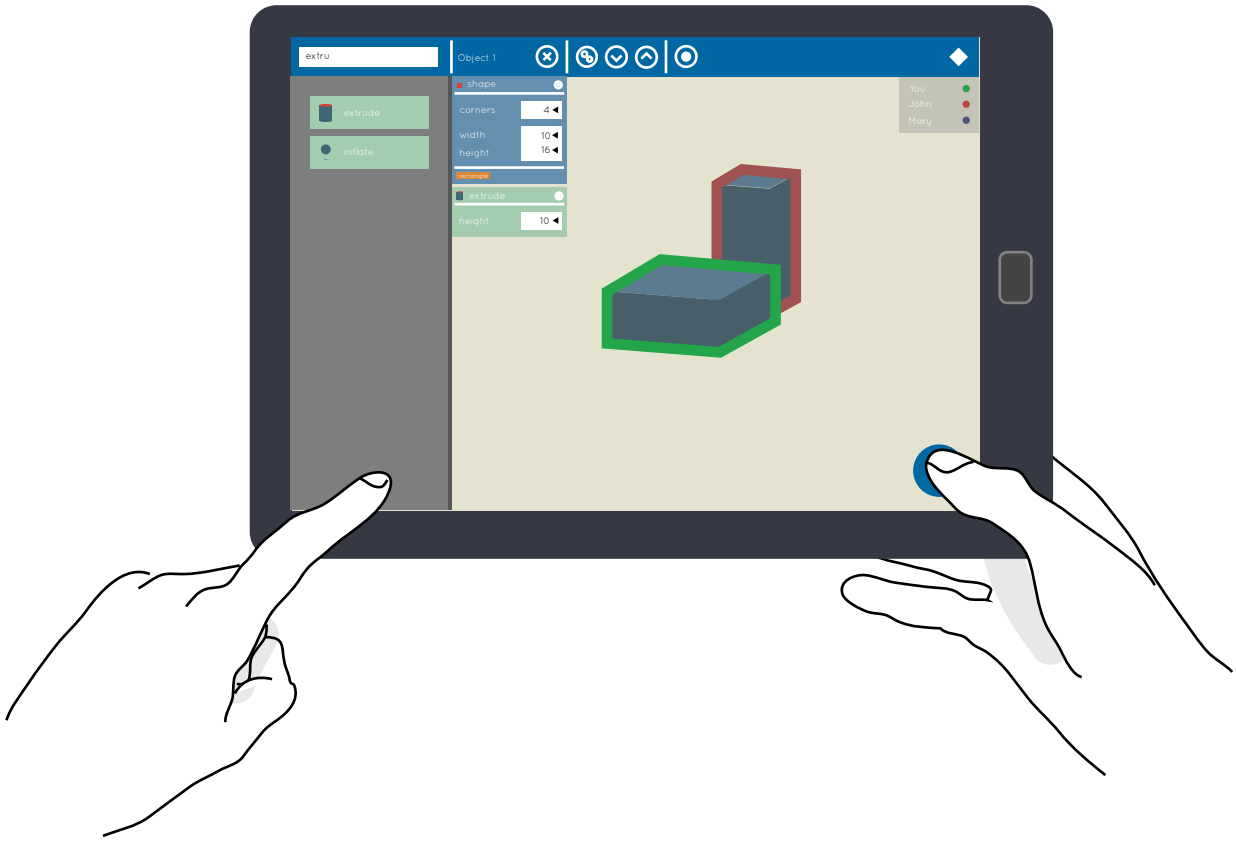

Object 1
You
Eric
Bertus
Marie
Frederiek
Maike

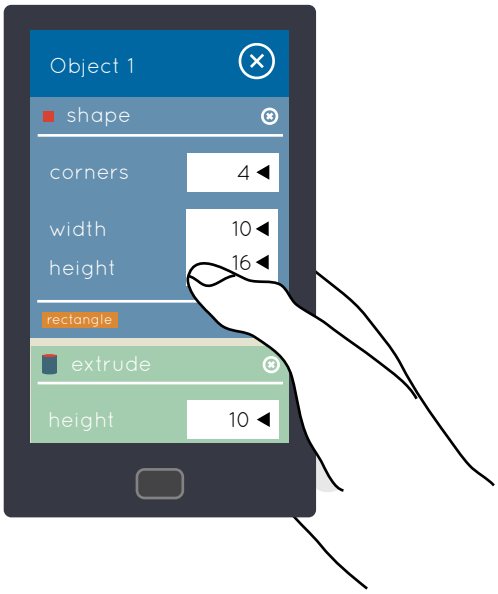

Object 1
You
Eric
Bertus
Marie
Frederiek
Maike

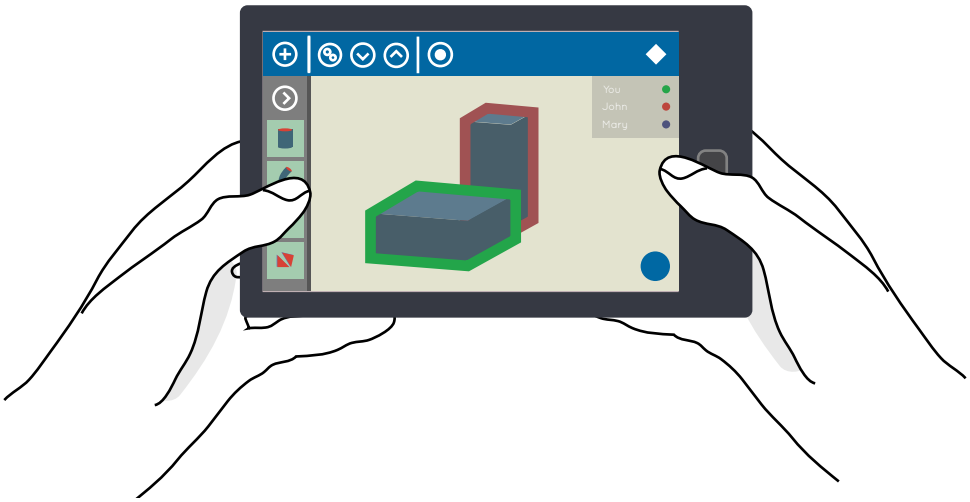
F

Interface op verschillende draagbare apparaten

G

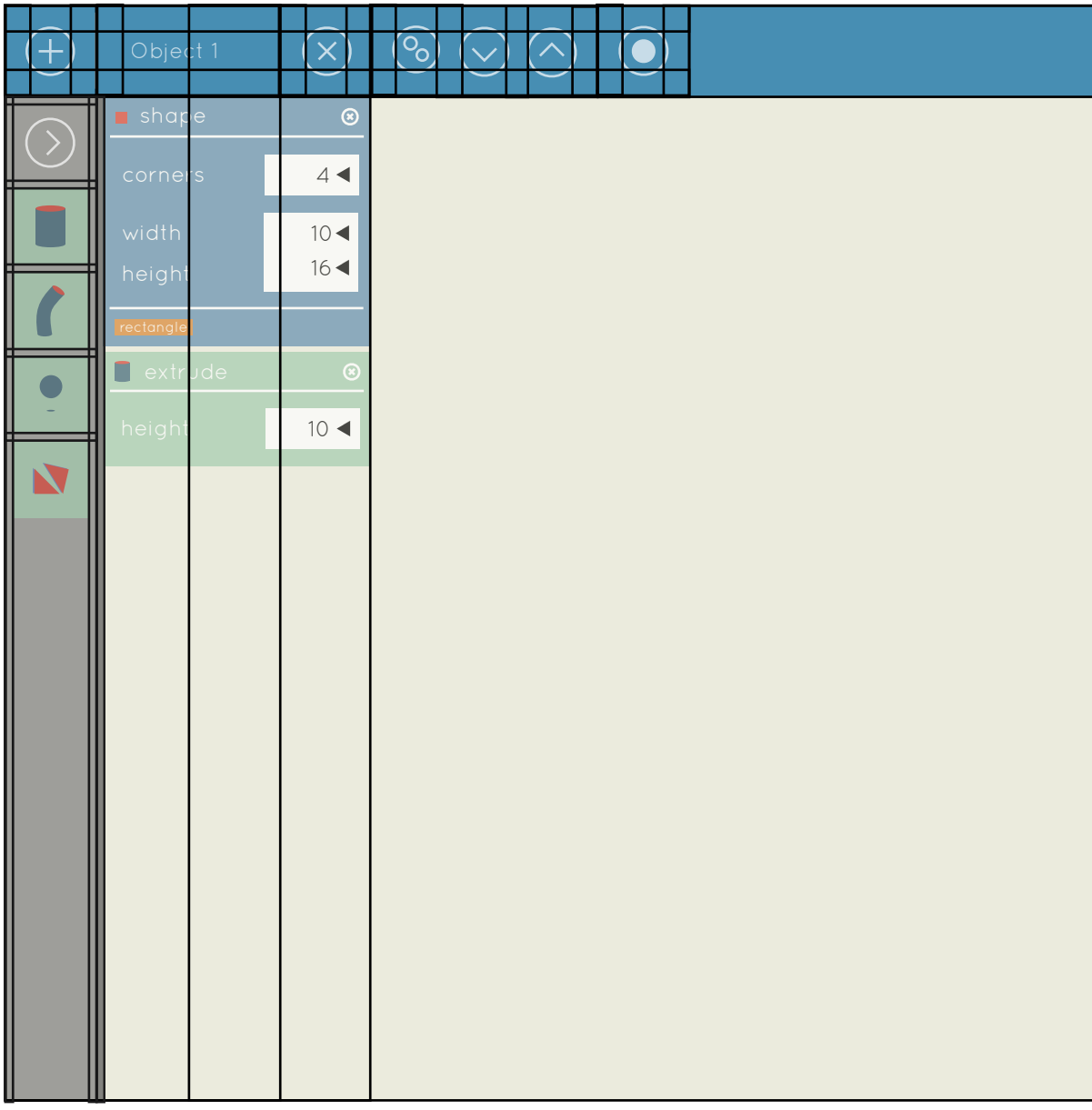



Object 1
shape
corners
4
width
10
height
16
rectangle
extrude
height
10

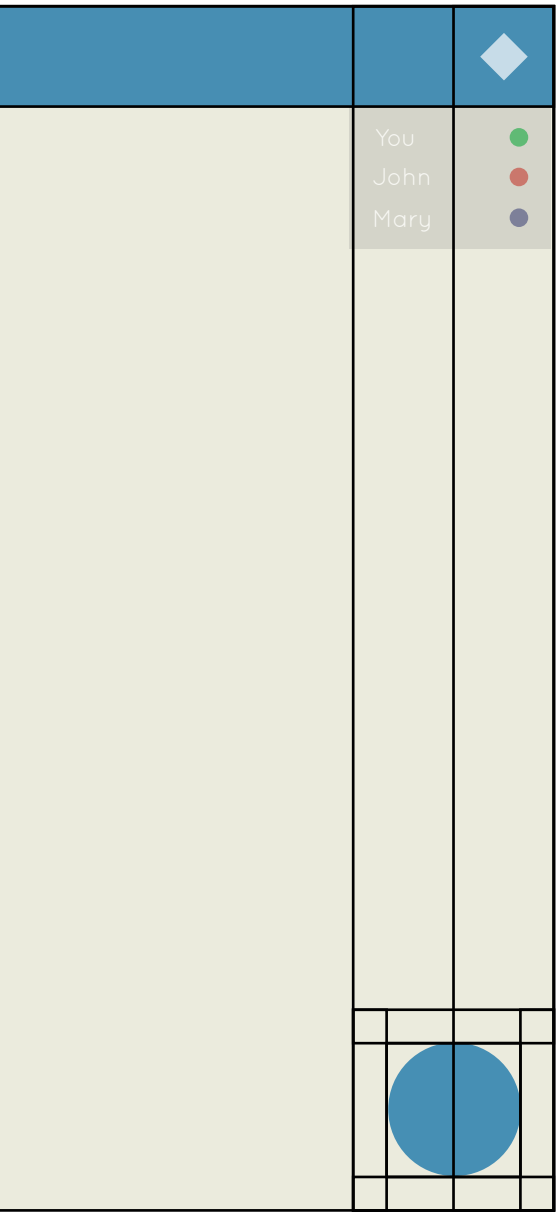

You
John
Mary

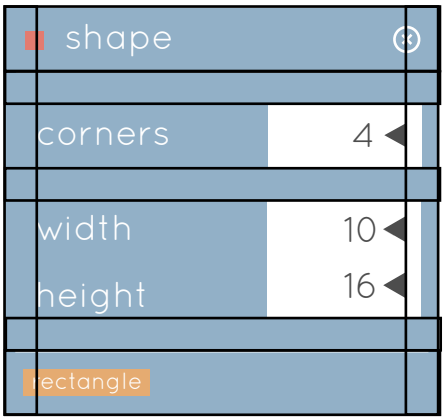
Visueel grid
Dit grid kan gebruikt worden om softwarematig de plaatsing van GUI elementen te bepalen.

H



H





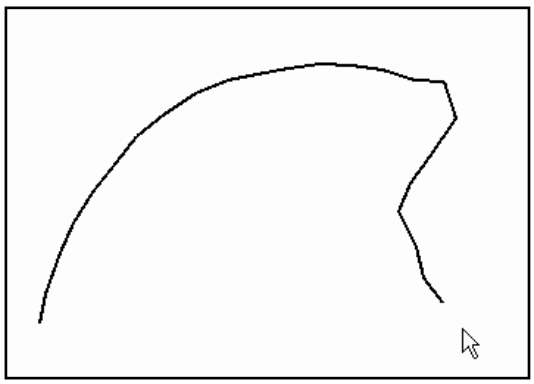
Schets algoritme

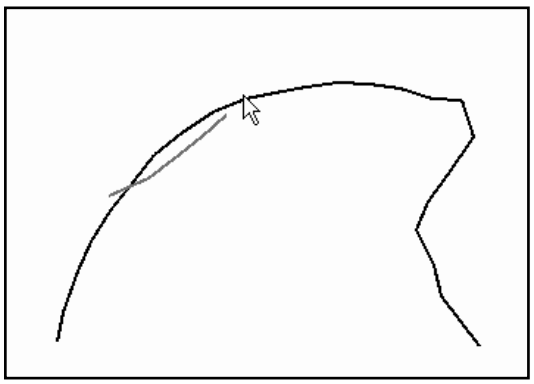
De ontwerpomgeving moet een schetsactige manier van werken ondersteunen. Schetsachtig betekend in dit geval dat vorm impliciet kan worden gedefineerd in plaats van expliciet. Een voorbeeld van een expliciet gedefineerde geometrieën zijn rechte lijnen of curves zoals splines. Dit zijn wiskundige functies: aaneenschakelingen van polynomen.

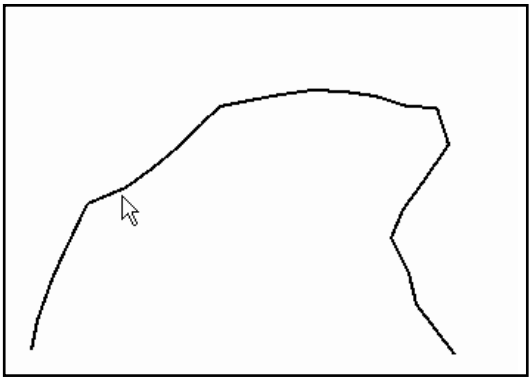
De vorm van deze curves kan aangepast worden door 'control' punten te verslepen. Dit is een harde mapping tussen implementatie en interface. Curves worden (wiskundig) gedefinieerd door deze punten, de interface is een directe manipulatie van deze punten. Hoewel de punten vrij direct te manipuleren zijn, is deze interface toch een laag tussen de intentie van de gebruiker en het resultaat. In dit experiment werd gekeken hoe een directere interface ontwikkeld kon worden die dichter bij het gevoel van schetsen ligt. De schetsbeweging sluit namenlijk intuitief goed met het eindresultaat(een lijn). Door de computer de vertaalslag van schets naar curve te laten maken verberg je een abstractielaag voor de gebruiker, en dit resulteert in een potentieel meer bruikbare interface.

Bij het schetsen defineert de tekenaar een lijn door vele lijnen te combineren. De uiteindelijke lijn is als het ware het 'gemiddelde' van alle lijnen tezamen. In dit experiment worden curves benaderd door ze op te delen in rechtlijnige segmenten. Door het gemiddelde te nemen van alle punten op de lijnen komt er een "gemiddelde" lijn uit. Zo kan de gebruiker fuzzy input leveren dat uiteindelijk toch een discreet resultaat oplevert.

Dit principe werd toegepast in een algoritme. De gebruiker tekent eerst een lijn, en vervolgens kan de gebruiker die lijn aanpassen door erover heen te schetsen. Voor ieder punt op de schets-lijn wordt getekend welk punt het dichtst in de buurt ligt en die twee punten worden gemiddeld. Zo kan de gebruiker de lijn aanpassen en verfijnen door er telkens overheen te schetsen.





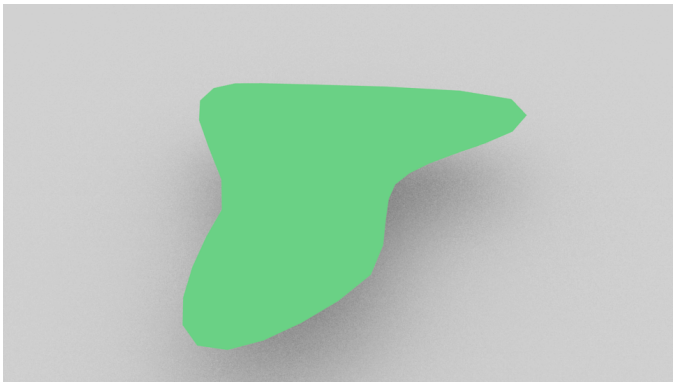


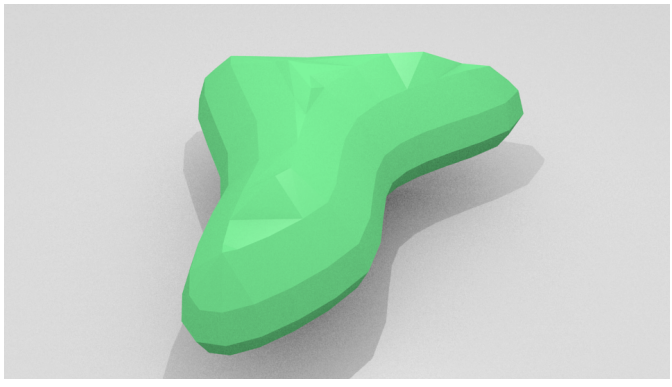
Inflatie algoritme
Er werd ook een nieuw algoritme bedacht om 2D polygonen om te zetten in een 3D vorm. De basis voor het algoritme is het idee van het opblazen van een ballon: door een lege 'platte' ballon op te blazen verkrijgt de balloon zijn 3D vorm. Het nieuwe algoritme is een oplossing voor het probleem dat een eenvoudige extrusie & schaal operatie goed werkt om een opgeblazen vorm te genereren voor convexe polygonen, maar bij concave polygonen levert dit een onverwachts resultaat op. De oplossing is om in plaats van het polygon te schalen, een equidistante polygon(offset) te genereren. Het algoritme werkt als volgt:

① Neem een lijst met punten die een polygon defineert.
② Bepaal of de punten met de klok mee, of tegen de klok in zijn gespecificeerd (dit noem je de polygon winding).
③ Neem het polygon, en genereer hiervan een equidistante polygon.
Dit doe je door voor ieder lijnsegment in de polygon een equidistante lijn te vinden. Dit doe je door voor ieder punt op de lijn de vector in de richting van het andere punt te nemen, en die 90 graden te draaien. (Hiervoor is het belangrijk de polygon winding te weten)
Combineer alle gegenereerde lijnen tot een nieuw polygon door de snijpunten te berekenen van iedere lijn.
Verwijder eventuele loops

④ Trianguleer deze twee polygonen, waarbij het equidistante polygon een gat vormt in het eerste polygon. Dit kan bijvoorbeeld met behulp van het 'ear clipping algoritme'.
⑤ Verplaats de punten van het nieuwe polygon verticaal tenopzichte van het origieneel.

⑥ Herhaal dit process recursievelijk voor het nieuwe polygon, totdat er geen punten meer over zijn.





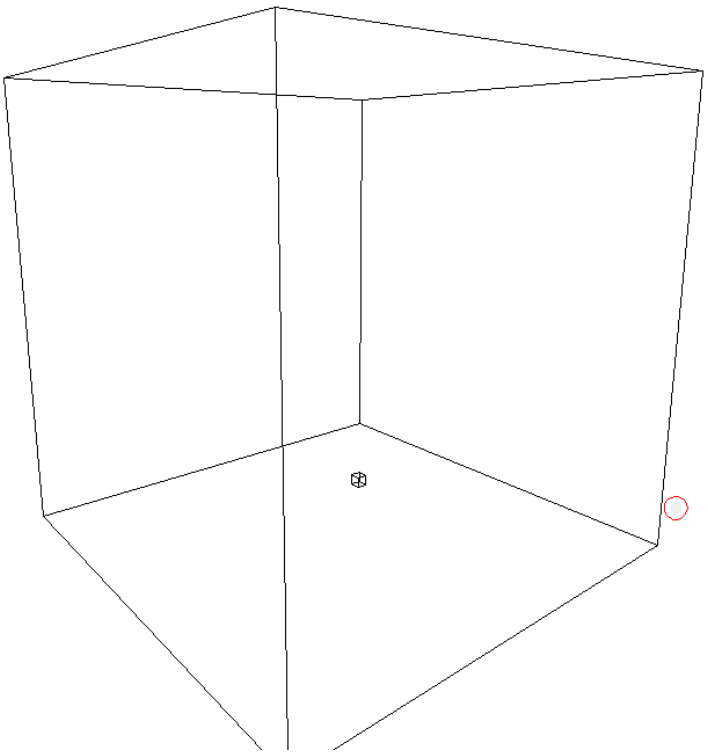
Interactie met leap motion

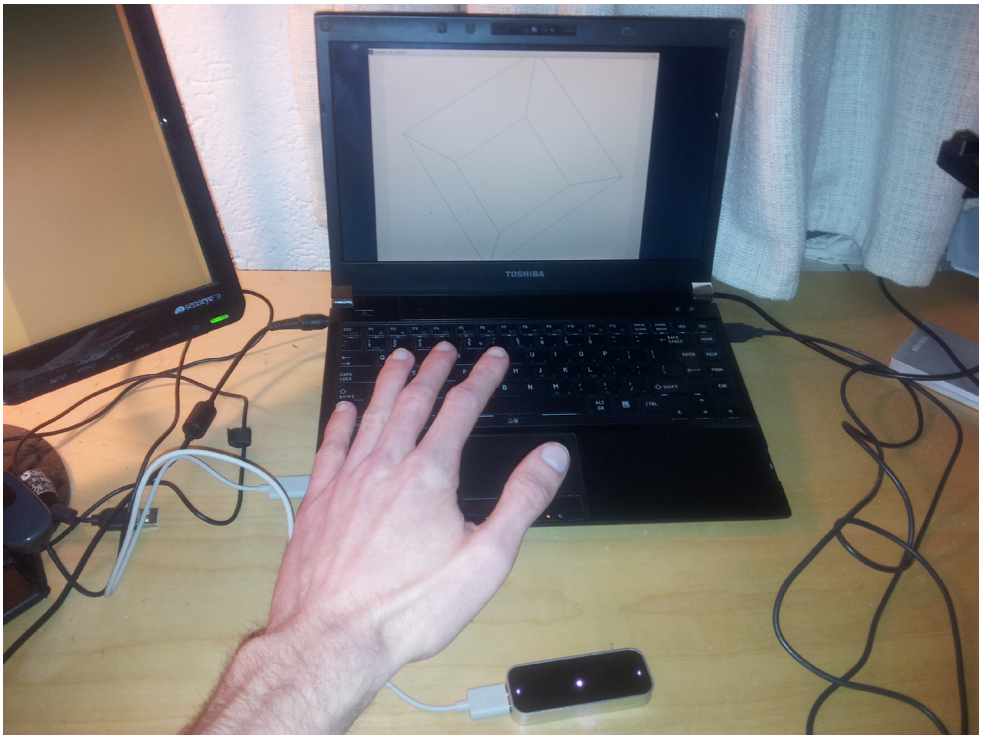
Er werd een tweetal experimenten gedaan met behulp van een Leap-motion controller. Een apparaat waarmee met behulp van gebaren met de computer kan worden geïnteracteerd. Er werd een experiment gedaan waarbij het draaien van de pols resulteerde in het draaien van een object.

In het tweede experiment werd onderzocht hoe een object in de 3d ruimte verplaatst kan worden met behulp van een grijp gebaar.

K

K





Design tools en rand apparatuur

L

①

muis

✔ vooraf bekend, precies, taktiel
✘ 2d, klikken non expressief

②

toetsenbord

✔ vooraf bekend, tekst invoer
✘ knoppen non expressief

③

leap motion

✔ 3d, twee handen, expressief, gebaren
✘ onnauwkeurig, niet tactiel

④

pen & tablet

✔ taktiel, expressief, intuitief
✘ 2d, learning curve

⑤

spacenavigator

✓ 3d, precies, taktiel, knoppen
✗ alleen navigatie, non expressief

⑥

MYO armband

✓ gebaren, intuitief
✗ geen beweging, non taktiel

L

⑦

smartphone app

✓ vooraf bekend, dynamische interfaces
✗ alleen 3d rotatie, non taktiel/swipe

⑧

game controller

✓ vooraf bekend, knoppen
✗ 2 handen bezet

⑨

midi controllers

✓ expressief, dynamische toewijzing
✗ gericht op muziek

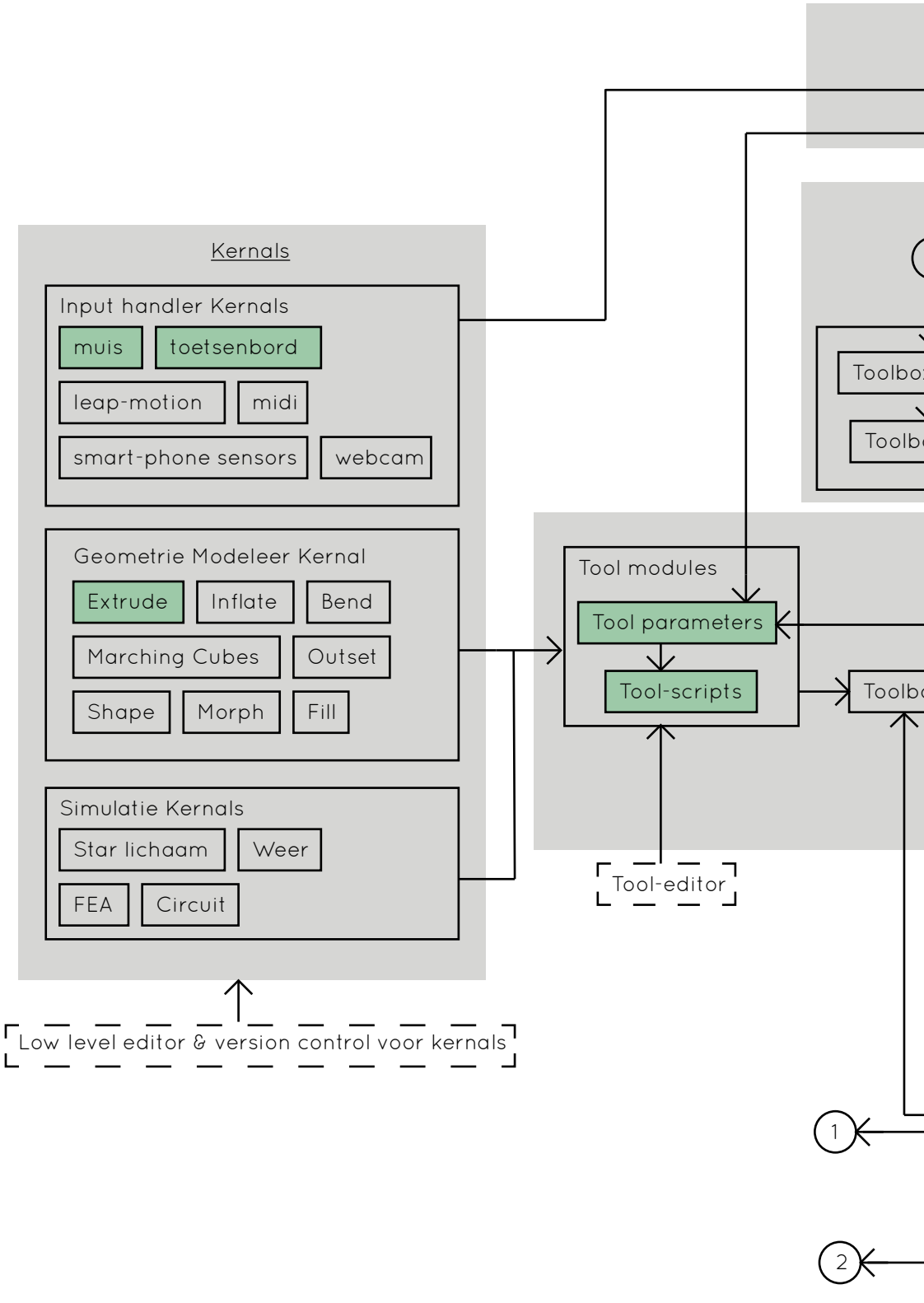
⑩

3d Trackers

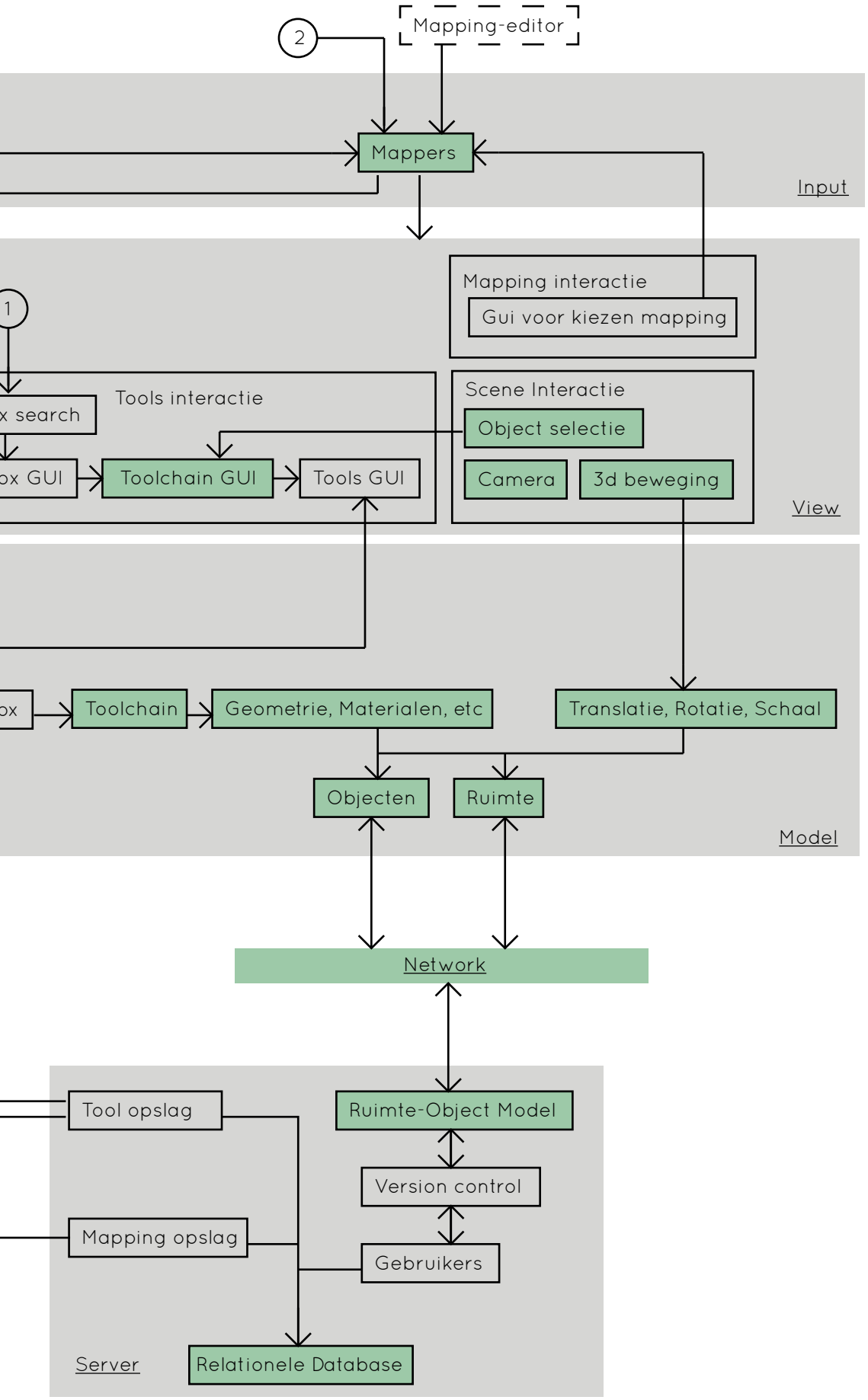
✓ expressief, zelf ontwikkeld
✗ niet makkelijk te verkrijgen

Geimplementeerde architectuur

M



= geimplementeerd


2
Mapping-editor
Mappers
Input
Mapping interactie
Gui voor kiezen mapping
1
Tools interactie
x search
ox GUI
Toolchain GUI
Tools GUI
Scene Interactie
Object selectie
Camera
3d beweging
View
ox
Toolchain
Geometrie, Materialen, etc
Translatie, Rotatie, Schaal
Objecten
Ruimte
Model
Network
Tool opslag
Ruimte-Object Model
Version control
Mapping opslag
Gebruikers
Server
Relationele Database

M